Unit II

Polygon, Windowing and Clipping

Polygons: Introduction to polygon, types: convex, concave and complex. Inside test.

Polygon Filling: flood fill, seed fill, scan line fill.

Windowing and clipping: viewing transformations, 2-D clipping: Cohen – Sutherland algorithm line Clipping algorithm, Sutherland Hodgeman Polygon clipping algorithm, Weiler Atherton Polygon Clipping algorithm.

Text Books:

1. S. Harrington, "Computer Graphics"‖, 2nd Edition, McGraw-Hill Publications, 1987, ISBN 0 – 07 – 100472 – 6.
2. Donald D. Hearn and Baker, "Computer Graphics with OpenGL", 4th Edition, ISBN-13: 9780136053583.
3. D. Rogers, "Procedural Elements for Computer Graphics", 2nd Edition, Tata McGraw-Hill Publication, 2001, ISBN 0 – 07 – 047371 – 4.

Reference Books:

1. J. Foley, V. Dam, S. Feiner, J. Hughes, "Computer Graphics Principles and Practice"‖, 2nd Edition, Pearson Education, 2003, ISBN 81 – 7808 – 038 – 9.
2. D. Rogers, J. Adams, "Mathematical Elements for Computer Graphics"‖, 2nd Edition, Tata McGraw Hill Publication, 2002, ISBN 0 – 07 – 048677 – 8.

Unit-II

Page No. 1
Date

polygons & clipping Algorithms

Course outcomes - To understand and implement the concept of polygon filling, windowing and clipping.

[Reference : TB1]

Introduction to polygons-

→ polygon is figure having many sides. it may be represented as a no. of lines segments connected end to end to form a closed figure.

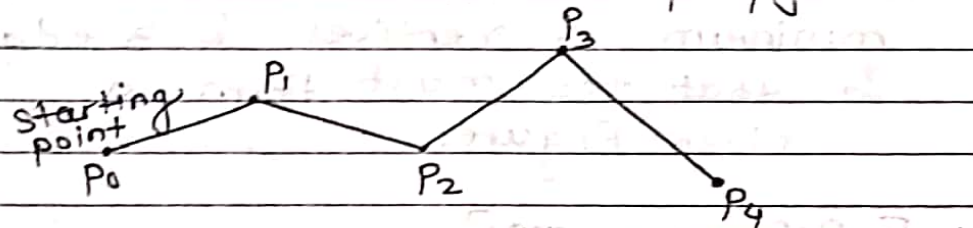
→ line segments which form the boundary of polygon are called as edges or sides of polygon.

→ The end points of the sides are called the polygon vertices.

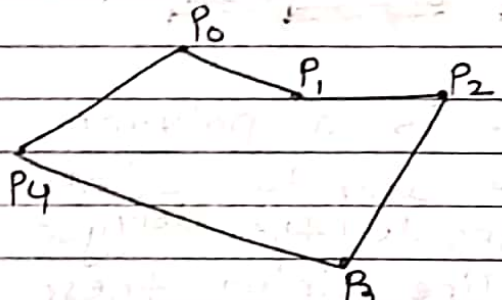
→ it is specified by giving the vertices (nodes) P_0, P_1, P_2 --- and so on.

→ The 1st vertex is called the initial or starting point & the last vertex is called the final or terminal point.

→ when starting point & terminal point of any polyline is same. i.e when polyline is closed then it is called polygon.



polyline

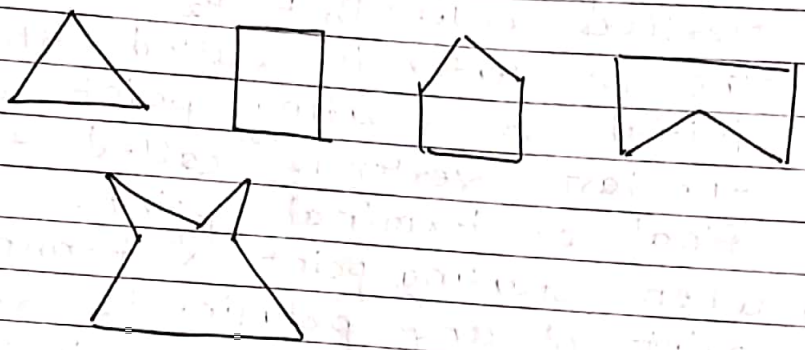


polygon.

Polygon -

- polygon is a figure having many sides. it may be represented as a number of line segments connected end to end to form a closed figure.
- The line segments which forms the boundary of polygon are called as edges or sides of polygon.
- The end points of the sides are called the polygon vertices.

e.g Triangle
3 sides & 3 vertices.



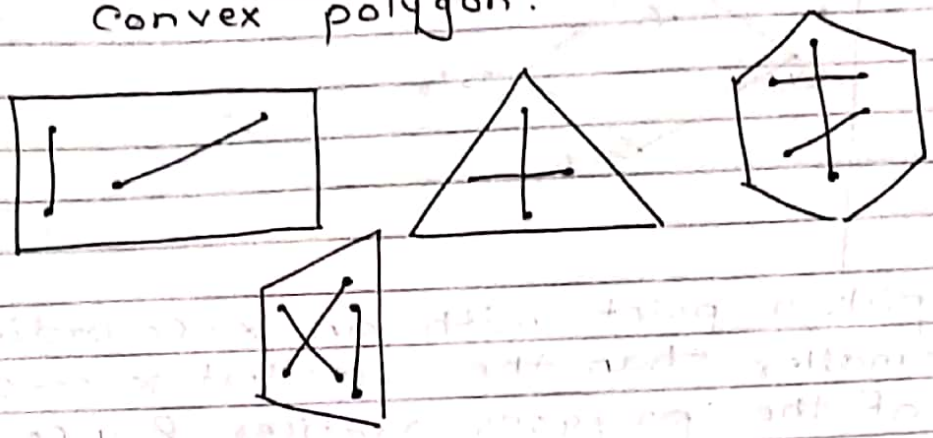
- To form a polygon we need minimum 3 vertices & 3 edges & that too must form a close figure..

[Reference : TBI]

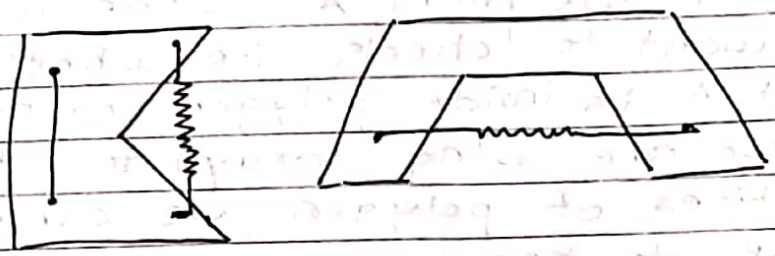
Types of polygons -

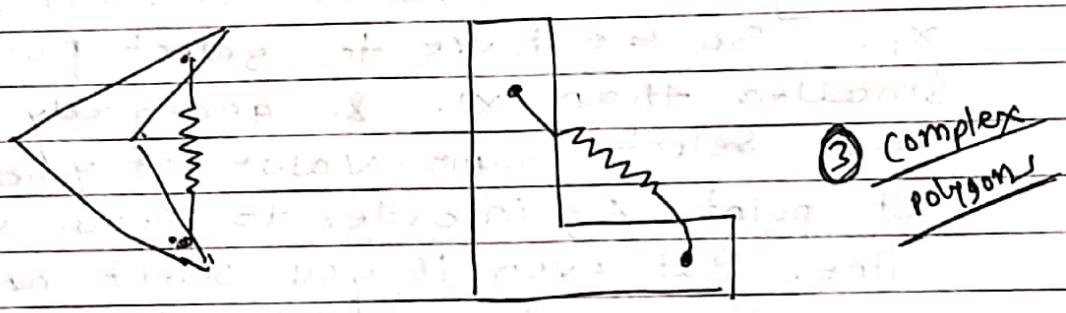
1] **Convex** :- is a polygon in which if you take any two points which are surely inside the polygon & if you draw a line joining these two points

and if all the points on that line lies inside the polygon. then such a polygon is called as convex polygon.



2] **Concave**:- The polygons which are not convex are called as concave polygons.





[Reference : TB1]

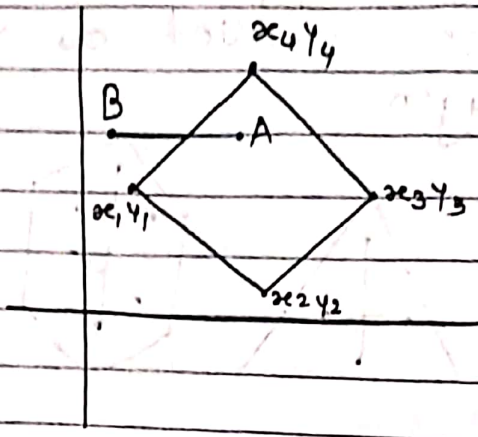
Inside Test of polygon :-

when we want to determine whether a point is inside the polygon or outside ? it generally determine by using two methods

1) even odd method
2) winding no. method

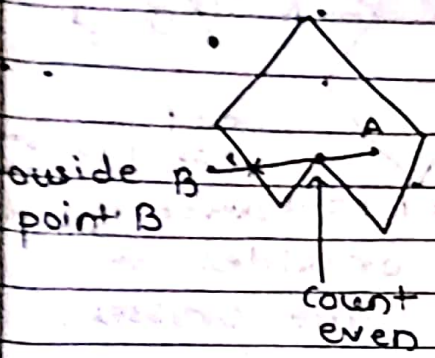
Page No.
Date

1) Even odd Method :-



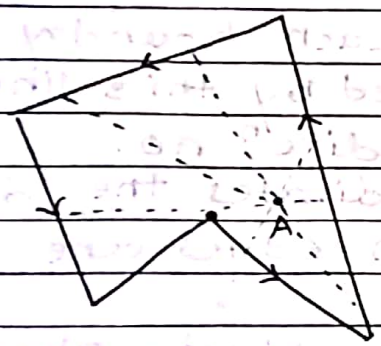
- pick a point with an x-co-ordinate smaller than the smallest x-cordinate of the polygons vertices & y co-ordinate will be any y value, or for simplicity we will take y value same as the y value of point in question.
- In this case point A is one, which we want to check i.e whether point A is inside polygon or not.
- As we are using arrays to store vertices of polygon we can easily come to know the vertex which is having lowest value of x & that is x_1. So we have to select point smaller than x1. & generally we select same value of y as that of point A, in order to draw straight line. But even if you select any y value for outside point, it will not make any difference.
- Then count how many intersections are occuring with polygon boundary by this line till the point in question. i.e 'A' is reached. if there are an odd no of intersections then the point in question is inside.

If even no of intersections then the point is outside the polygon.

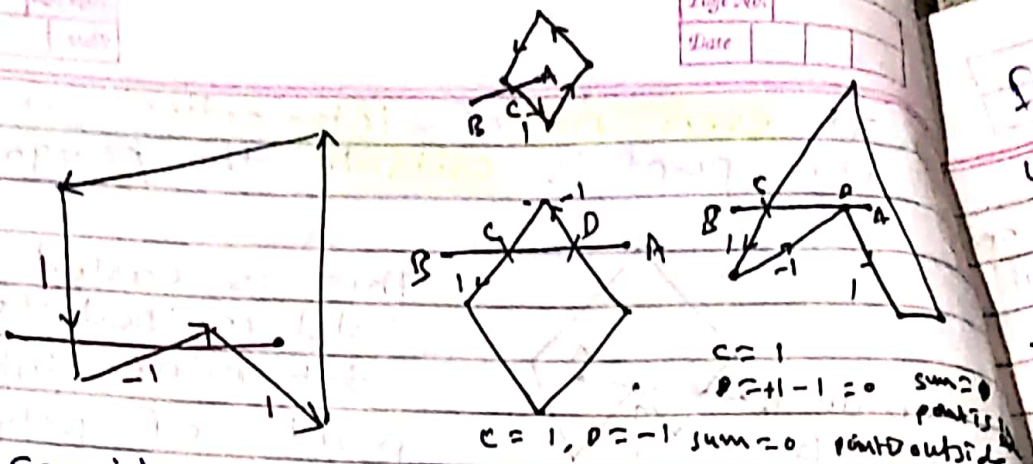


This is called even odd method to determine interior points of polygon.

2] Winding Number Method :-



- Consider a piece of elastic between point of question (A) & a point on polygon boundary.
- elastic is tied to point of question firmly & the other end of elastic is sliding along the boundary of polygon until it has made one complete circuit.
- Then we check how many times the elastic has been wound around the point of question (A). if it is wound at least once then the point is inside. if no net winding then point is outside.
- To explain in more simple words, we begin with even odd method. we draw a line between point of question & outside point.



- Consider the edges or sides of polygon where this line crosses.
- In even-odd method we just count the no. of intersections. But in winding no. method we give direction no. of each boundry line which is crossed by this line & we sum these dire^n no.
- Direction no. indicates the dire^n in which polygon edges are drawn.
- The side which starts above the drawn line & crosses line & then ends below the line, we give -1 to direction no. then find the sum of these no. if it is non-zero, then the point is inside.
- if sum is zero, the point is outside.

[Reference : TB1]

Polygon Filling :-

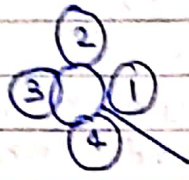
- filling is the process of "coloring in" a fixed area or region.
- region may be defined at pixel level or geometric levels.
- When the regins are defined at pixel level, we are having diff algorithms.

① Boundry fill ③ Edge fill
② Flood Fill ④ scanline

In case of geometric level we are having similar algorithm

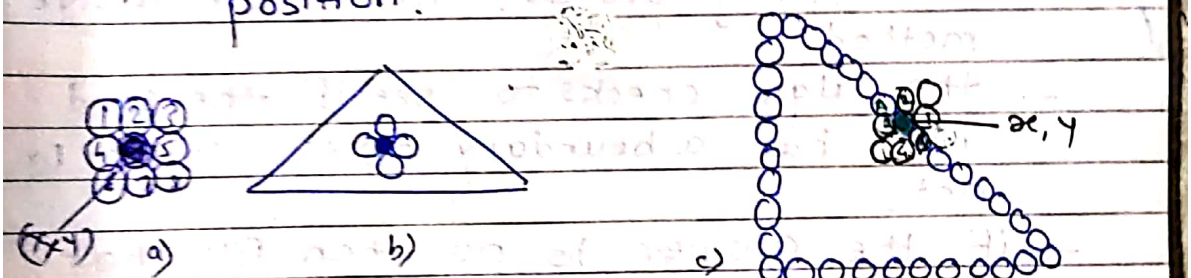
4 connected & 8-connected pixel concept:-

- These two techniques are the ways in which pixels are considered as connected to each other
- In 4- Connected method the pixel may have upto 4 neighbouring pixel.



let's assume that the current pixel is (x, y). from this pixel we can find four neighboring pixels as right, above, left & below of the current pixel.

8 connected method :- pixels may have upto 8 neighbouring pixels. Here from one pixel location we are finding the neighbouring 8 pixels position.



- By using any one of these technique we can fill the interior of the polygon.
- But there are some cases where the use of 4- connected method is not efficient.

Page No.
Date

- Suppose we want to fill a triangle, so we are using 4 conn... method is not efficient. it will work fine but at boundary this method is not efficient.

Fig. b

- Suppose we have filled point (x, y) now from that point we have to select either point A or B which is not possible by using 4-connected method because in 4-connected method we can go from (x, y) to either 1, 2, 3, or 4 and not A & B.

Fig. c

- So in this case we have to use 8-connected method where we can choose any one of the neighbouring pixel.

Boundary Fill Algorithm -

- This algo is very simple. it needs one point which is surely inside the polygon. This point is called as seed point. which is nothing but a point from which we are starting the filling process. This is a recursive method.
- the algo checks to see if the seed pixel has a boundary pixel color or not.
- if the answer is no then fill that pixel with color of boundary & make recursive call to itself using each of its neighbouring pixels as new seed.
- if the pixel color is same as boundary color then return to its caller.

- This algo works for any shaped polygon & fills that polygon with boundary color.
- But as it uses high number of recursive calls it takes more time & memory.

The following procedure illustrates a recursive method for boundary fill by using 4-connected method.

```
bfill (x, y, newcolor)
{
    current = getpixel (x, y);
if (current != newcolor) && (current != boundary
                                color)
{
    putpixel (x, y, newcolor);
    bfill (x+1, y, newcolor);
    bfill (x-1, y, newcolor);
    bfill (x, y+1, newcolor);
    bfill (x, y-1, newcolor);
}
}
```

- As we are using recursive function there is a chance that system stack may become full.
- So we have to develop our own logic to solve the stack problem.
- one solution is instead of using system stack we can use our own stacks & write our own PUSH & pop functions for that stack.

- second solution could be we of link list i.e dynamic memory allocation. for implementation of stack.
- Another solution could be develop some logic while pushing the pixels on stack. i.e. before pushing the pixel on stack check whether that pixel is already visited or not. if it is already visited there is no point to store that pixel again.
- Another solution could be instead of using stack we can use of queue also.

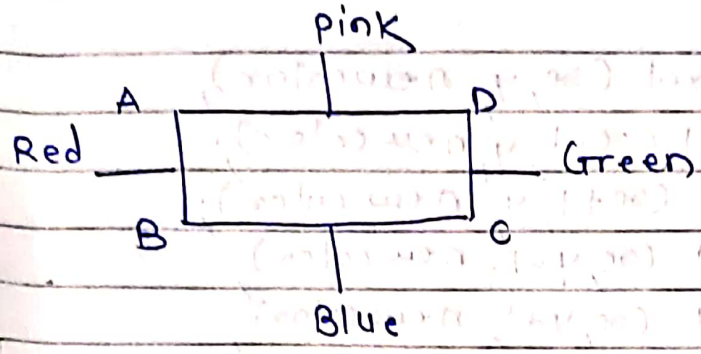
limitations of Boundary fill algo-

1] it fills the polygon having unique boundary color.
2] if the polygon is having boundaries with different colors then this algo fails.

Flood fill Algorithm :- Seed fill

1] The limitations of boundary fill algo are overcame in flood fill algo. like boundary fill algo this algo also begins with seed point which must be surely inside the polygon.
2] instead of checking the boundary color this algo checks whether the pixel is having the polygon's original color. i.e previous or old color.
3] if yes, then fill that pixel with

new color & uses each of the pixels neighbouring pixel as a new seed in a recursive call. if the answer is no. i.e the color of pixel is already changed then return to its caller.



- sometimes we want to fill an area that is not defined within a single color boundary.
- edge AB, BC, CD & DA are having red, blue, green & pink color respectively.
- This figure shows an area bordered by several color regions. we can print such areas by replacing a specified interior color instead of searching for a boundary color value. we are setting empty pixel with new color till we get any colored pixel.
- flood fill algorithm is particularly useful when the region or polygon has no uniformed colored boundaries.
- flood fill algorithm is sometimes also called as seed fill algo or forest fire fill algo. Because it spreads from a single point i.e seed point in all the direction.

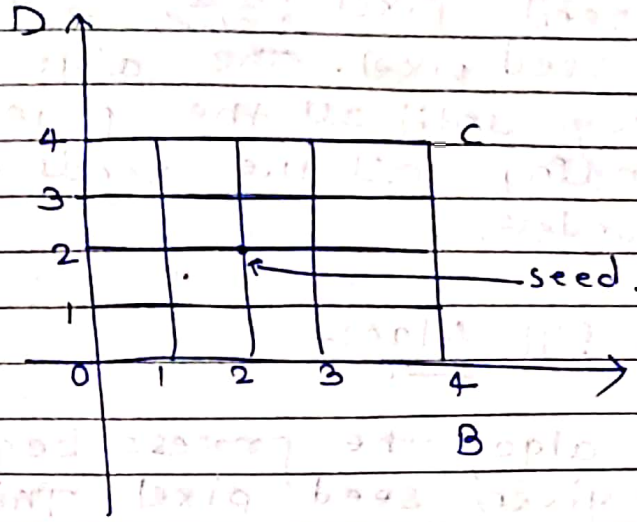
- flood fill by using 4-connected method

```
f-fill (x, y, newcolor)
{
  current = getpixel (x, y);
  if (current != newcolor)
  {
    putpixel (x, y, newcolor);
    f-fill (x-1, y, newcolor);
    f-fill (x+1, y, newcolor);
    f-fill (x, y-1, newcolor);
    f-fill (x, y+1, newcolor);
  }
}
```

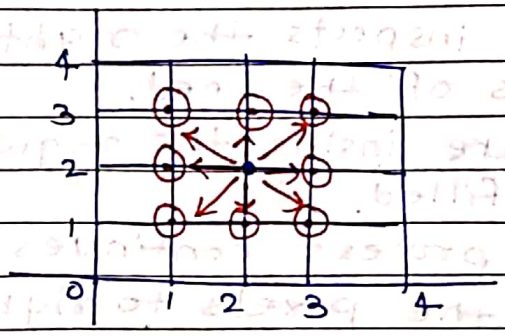
flood fill Algorithm :-

1] In flood fill algorithm the given initial interior pixel is considered as seed pixel.
2] Starting from this seed, the algo inspects all the surrounding eight pixels to check whether these pixels are boundary pixels or they are inside the region.
3] Those pixels which are inside the region are set to polygon value, i.e they are coloured & the boundary pixels are left as it is.
4] This process is repeated till all the interior pixels are inspected.

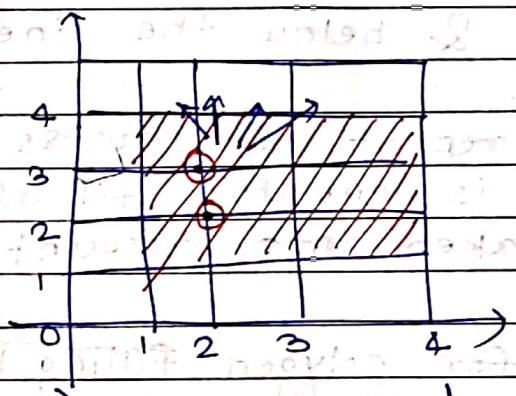
example :- Consider a polygon ABCD having vertices A(0,0), B(4,0), C(0,4) & D(4,4). seed pixel is (2,2). Now we have to fill this polygon using food fill algo.



i)



ii) Adjacent eight pixels to be inspected.



iii) (2,3) as new seed pixel

Initially (2,2) is seed pixel, in next fig. the algo inspects all the eight points surrounding the seed (1,1) (1,2) (1,3) (2,3) (3,3) (3,2) (3,1) & (2,1) since none of the pixels are boundary pixels hence each one will be filled. Consider any pixels adjacent to seed (2,2)

as new seed. pixel (3,4) is consid as new seed pixel. The algo contin in this way until all the points surrounding all the seeds enter the border.

Boundary fill Algo:-

1] In this algo, the process begins with a given seed pixel. Then the algo inspects the right & left pixels of the seed.
2] If they are inside the region then they are filled.
3] Then the process continues to identify the pixels to right & left to it, till the leftmost & rightmost boundary pixels have been reached.
4] The algo then finds the pixels above & below the line just drawn.
5] And repeat the process. This process is repeated till all pixels are taken into account.

Algorithm for polygon filling by seed fill algo :-

stack is used as main data stru.

1. push the seed pixel onto the stack.
2. while (stack not empty) do.
3. pop the pixel from the stack.
4. Set the pixel to polygon value.
5. for each of the four connected

pixels adjacent to current pixel, check if it is a boundary pixel or it is already set to the polygon value.

6. In either cases ignore it.
7. otherwise push pixel into the stack.
8. end while.

procedure is completed when stack is empty.

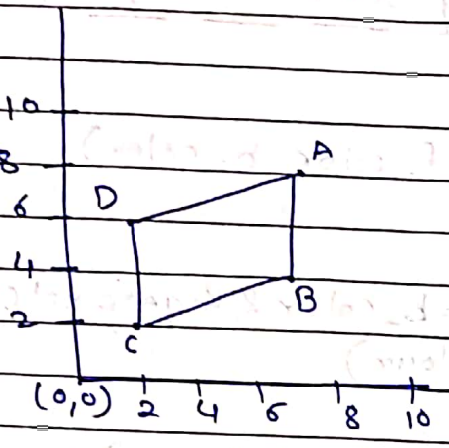
polygon filling by seed fill Algo:-

```
boundary_fill (x,y, f_color, b-color)
{
if (getpixel (x,y) != b_color && getpixel (x,y)
        != f_colour)
    {
      putpixel (x,y, f_colour)
    boundary_fill (x+1, y, f_colour, b_colour);
    boundary_fill (x, y+1, f_colour, b-colour);
    boundary_fill (x-1, y, f-colour, b-colour);
    boundary_fill (x, y-1, f_colour, b_colour);
    }
}
```

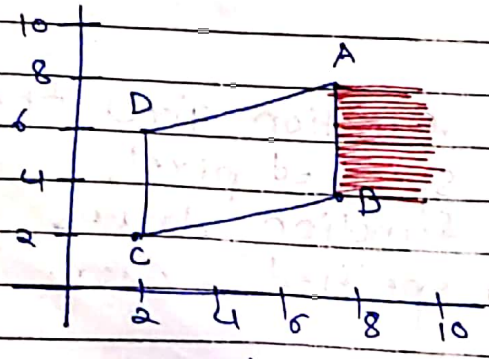
* getpixel function gives the colour of specified pixel.
* putpixel function draws the pixel with specified color.

Edge fill Algorithm :-

- In edge fill algorithm the polygon is filled by selecting each edge of the polygon.
- algo is very simple.
- statement :- for each scan line intersecting a polygon edge at (x_1, y_1) complement all pixels whose mid point lie to the right of (x_1, y_1) Here the order in which the polygon edges are considered is unimport.

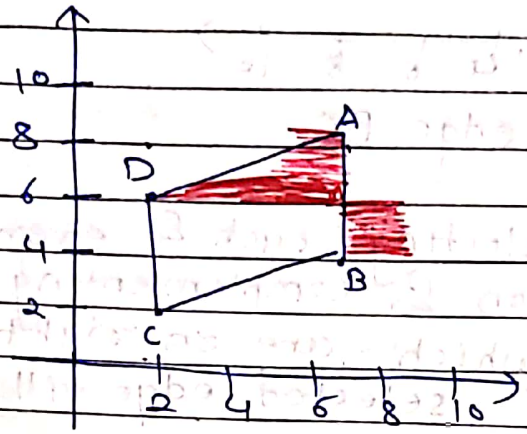


- we want to fill a polygon ABCD.
- statement of algo says we have to consider one edge at a time.
- Select edge AB.
- we have to complement all the pixels whose midpoints are lieing on right hand side of edge AB.



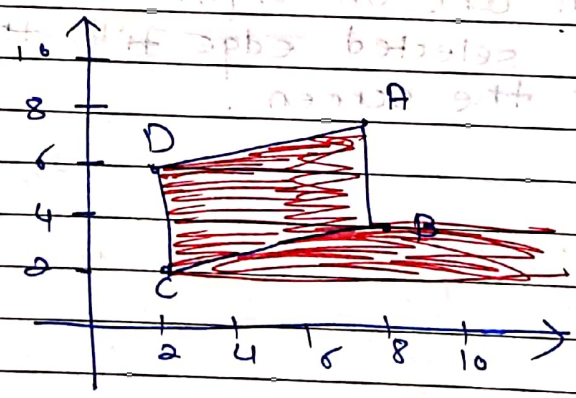
edge AB

- Now we have to select another edge say AD. order of selecting of edges is not important.
- Again complement all the pixels whose midpoints are lieing on right hand side of selected edge.

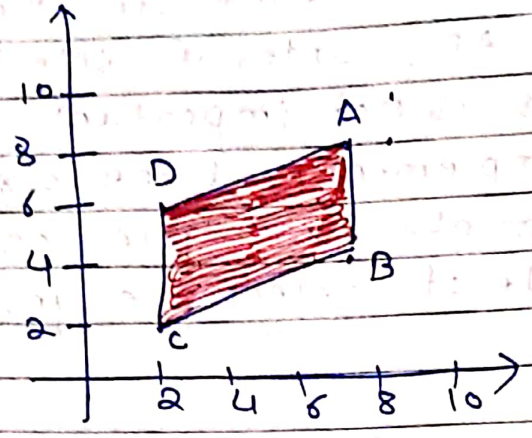


edge AD

- complement means if earlier the pixel is having background color, now we have to make color of that pixel as fill color & if earlier the pixel is having background color. now we have to make color of that pixel as fill color & if earlier it is ~~not~~ fill color then now we have to make it as background color.



edge DC



edge BC

- we are selecting each & every edge of polygon & complementing all the pixels which are on right hand side of selected edge till end of screen.

Drawback :-

- for complex picturers each individual pixel is addressed many times. So the algo requires more times.
- is even through the polygon is placed in upper left hand corner of the screen, every time we are complementing all the pixels which are on right hand side of the selected edge till the end of the screen.

[Reference - TBI]

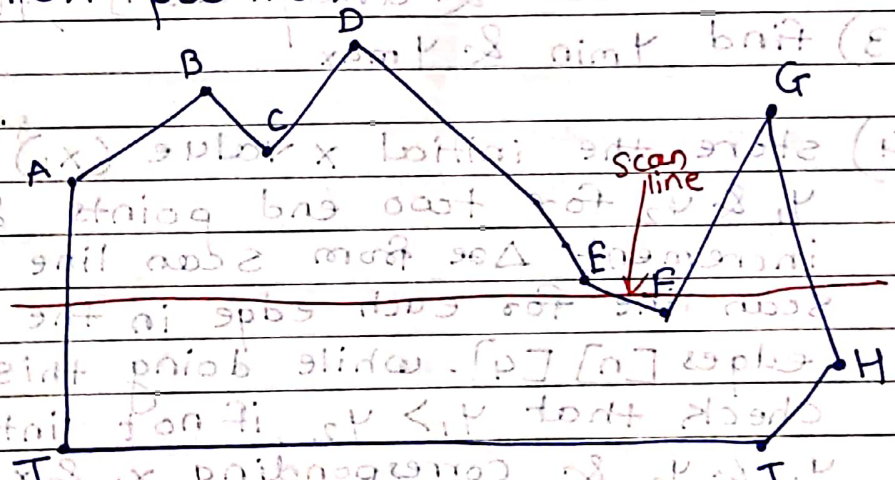
Scan line fill :-

Features :-

1] Recursive seed fill procedures require stacking of neighbouring points so to avoid this,

2] In this method fills horizontal pixel spans across scan lines, instead of preceeding to 4-connected or 8-connected neighbouring points. This is achieved by identifying the right most & leftmost pixels of the seed pixel & then drawing the horizontal line between these two boundary pixels.

3] With this efficient method, we have to stack only a beginning position for each horizontal pixel span instead of stacking all unprocessed neighbouring positions around the current position.



4] A scan line algo for filling polygons begins by ordering the polygon sides on the largest y values. It begins with the largest y value & scans down the polygon.

5] For each y, it determines which sides can be intersected & finds the

x values of these intersection points. it then sorts, pairs & passes these x values to a line drawing routine.

Characteristics of scan line polygon fill algo:-

- it fills horizontal pixel spans across the line.
- it avoids the stacking of neighbouring points.
- it is used in orthogonal projection
- it is non-recursive algo.
- used for filling of polygon.

Steps -

1) Read n, the no. of vertices of polygon.
2) Read x & y co-ordinates of all vertices & array x[n], y[n]
3) find y_{min} & y_{max}
4) store the initial x value (x_1), y values y_1 & y_2 for two end points & x increment Δx from scan line to scan line for each edge in the array edges[n][4]. while doing this, check that $y_1 > y_2$, if not interchange y_1 & y_2 & corresponding x_1 & x_2 so that for each edge y_1 represent its maximum y co-ordinate & y_2 represents its minimum y co-ordinate.
5) sort the rows of array edges[n][4] on desending order of y_1 descending order of y_2 & ascending order of x_2.

6] Set $y = y_{max}$

7] find the active edges & update active edge list.

if $(y > y_2 \ \& \ y \leq y_1)$
[edge is active]
else
[edge is not active]

8] Compute the x intersects for all active edges for current y value
[initially x-intersect is x_1 & x intersects for successive y values can be given as

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

where $\Delta x = -\frac{1}{m}$ &

$m = \frac{y_2 - y_1}{x_2 - x_1}$ i.e slope of a line segment.

9] if x intersect is vertex i.e x intersect $= x_1$ & $y = y_1$ then apply vertex test to check wheather to consider one intersect or two intersects.
Store all x intersects in the x-intersect [] array.

10] sort x-intersect [] array in the ascending order.

11] extract pairs of intersects from the ~~the~~ sorted x-intersect [] array.

12] pass pairs of x values to line drawing routine to draw corresponding line segments.

13] set $y = y - 1$,

14] Repeat steps 7 through 13 until $y \geq y_{min}$

15] stop.

[Reference : TB1]

Windowing and clipping -

- window is a rectangular portion through which we can see scene.
- it is opening through which we can see a portion of the world that exist outside.
- when we are looking from window some portion of the scene is visible to us.
- e.g some complex drawing or maps are drawn on comp. like Road map of state, if want to read the road map for pune city, Then it will be very difficult to read the map so that it will be useful to display only that part of the drawing in which we are intrested. if we want to see the road map of pune city then we are limiting our scope. & concentrate on that part of map.
- The method of selecting & enlarging the portions of a drawing is called windowing.

Clipping :-

- when we look through the camera towards some scene, we are able to see the scene but in addition to that scene

we see a rectangular area ( ) also
- The final photograph gets developed with scene which is inside the rectangular area.
- It does not mean that we should not see the image which is outside the rectangular area.
- Determining which portion to omit or suppress is called clipping.
- clipping eliminates the objects or portions of objects which are not visible through window.

Viewing Transformation -

- for displaying any scene, we have two models
  1) object model
  2) Image model
- Object Model - model of object which is stored in computer with the ~~Scene~~ actual scale.
- it may be millimeter or in kilometers.
- space where object model reside is called object space.
- Image Model :- is one which appears on display device. the image which we want to draw on display device must be measured in screen co-ordinates.
- There are different types of display devices with variations of screen co-ordinates. So to make it device independent we have to use normalized co-ordinates.
- we need to convert the object space units to image space co-ordinates.

Object space

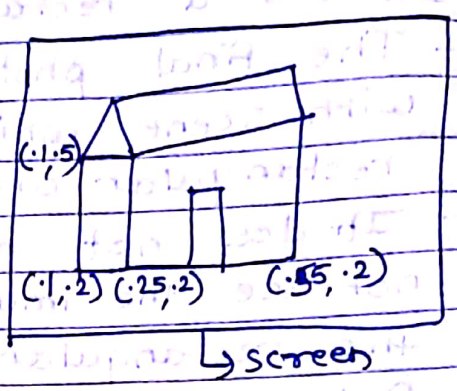
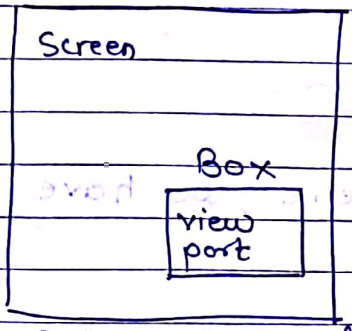


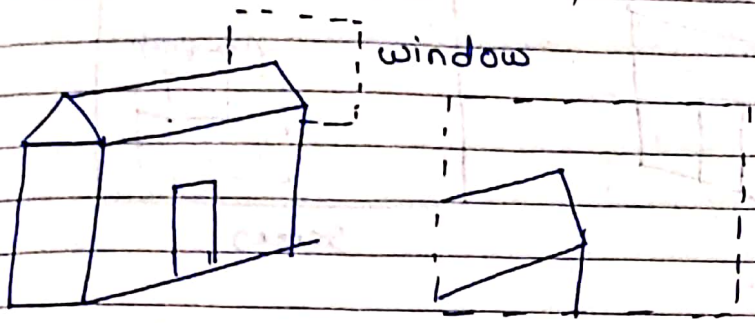
Image space

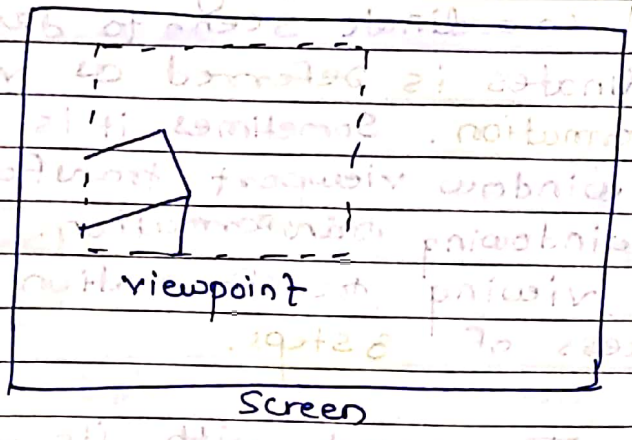
for this we have to use scaling transformation



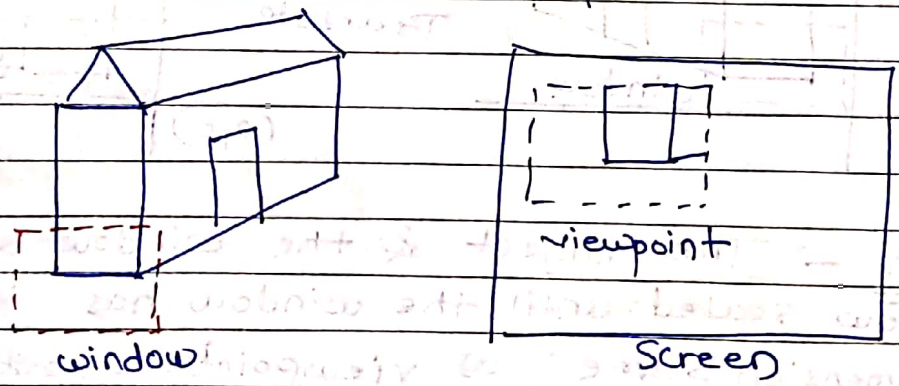
- Before proceeding with transformation, we will first see another term which is called as viewport.
- it may happen that we may not want to use entire screen for display.
- we just want some part of screen to display the image. So we will form a rectangular box on screen & in that box only we will display the image This box is called as viewpoint.
- The object space contains the dimensions as actual which is called as world co-ordinate sys.
- a world co-ordinate area selected for display is called window.
- An area on a display device to which a window is mapped is called viewpoint.
- So the window defines what is to be viewed, clipping means what to omit

viewport defines where it is to be displayed on display device



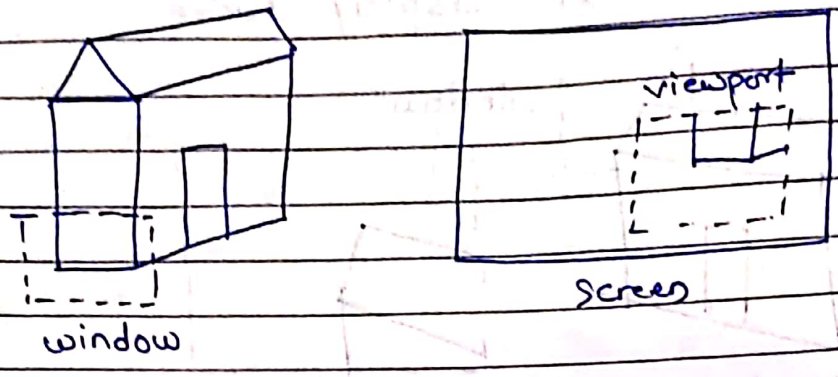


- If we are changing the position of window by keeping the viewpoint location constant, then the different part of the object is displayed at the same position on the display device. As the position of viewpoint is same on screen only the contents will get changed.



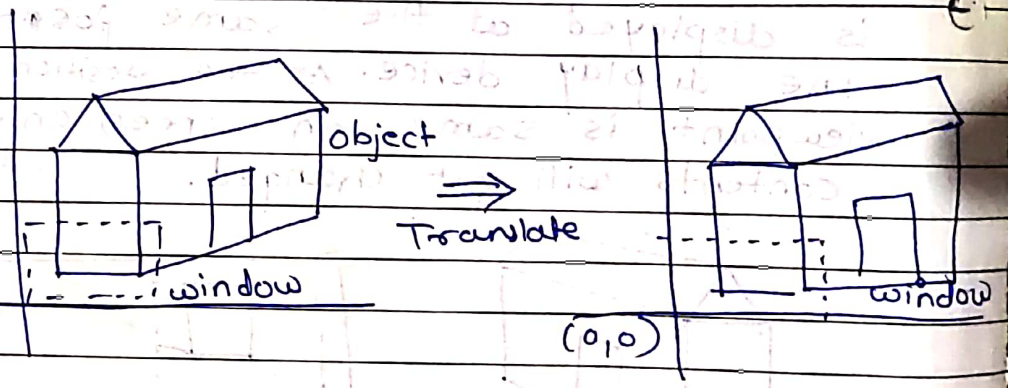
Similarly if we change the location of viewport then we will see the same part of the object drawn at different

places on screen as

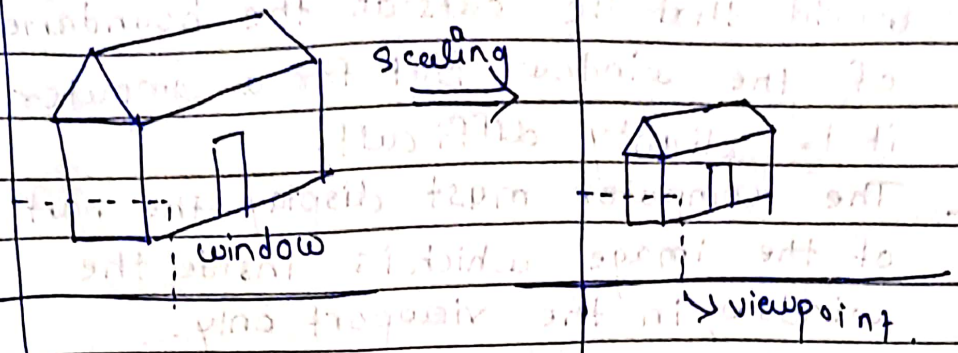


- In general the mapping of a part of world co-ordinate scene to device co-ordinates is referred as viewing transformation. Sometimes it is also called as window viewport transformation or windowing transformation.
- This viewing transformation is a process of 3 steps.

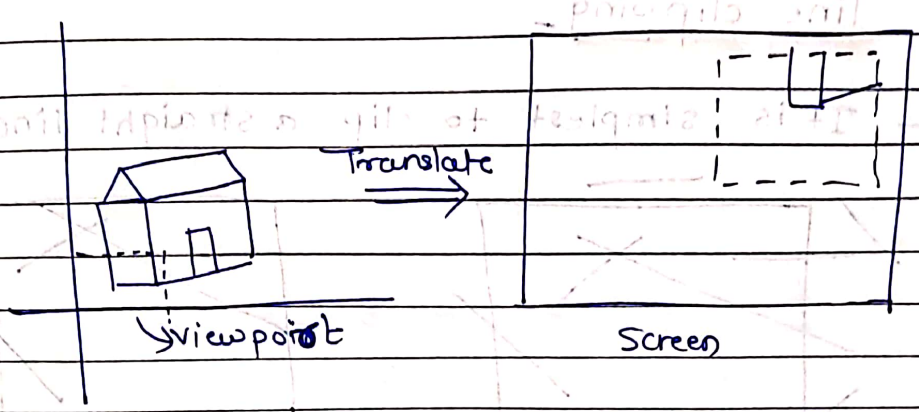
Step1 – The object with its window is shifted or translated until the lower left corner of the window matches to the origin.



Step2 – The object & the window are now scaled until the window has the dimension same as viewpoint. In other words we are converting the object into image & window in viewpoint. for this we have to use scaling.



Step 3:- perform another translation to move the viewport to its correct position on the screen.



So the viewing transformation performs 3 steps:

- Translation
- Scaling
- Translation

In reality we are using scaling to map window to viewpoint & translation to place viewpoint properly on screen.

[Reference: TB]

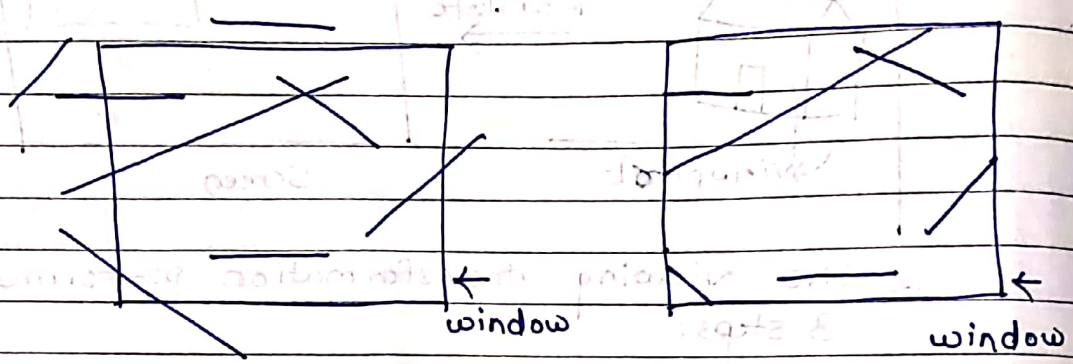
2D Clipping :-

- means omitting the part of picture which lies outside the window and displaying the portion of picture which is inside the window.
- clipping is easy for the human visual system, even with an imaginary window

- we just do not see those parts of the world that lie outside the boundaries of the window. But for a computer it is slighty difficult.
- The computer must display the part of the image which is inside the window, in the viewport only.
- The images may be lines, polygons or text. there are different algorithms to clip different types of images.

line clipping -

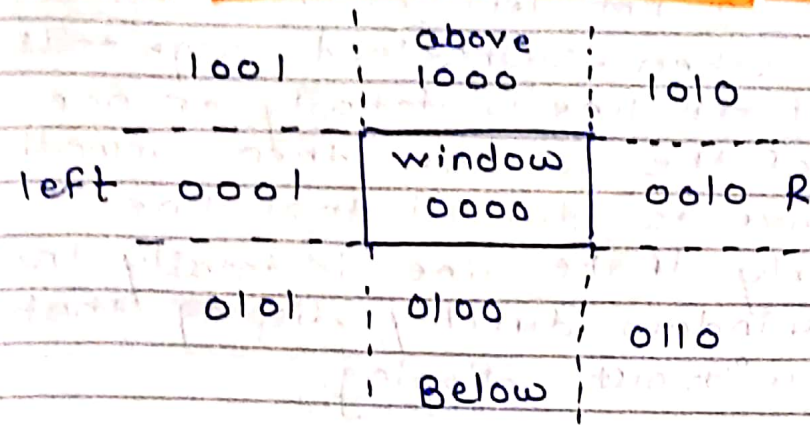
- It is simplest to clip a straight line.



- fig. shows lines of variow types before clipping & after clipping.
- we have to examine each & every line which we are going to display. we have to determine whether or not aline is completely inside the window. or lies completely outside the window, or crosses the boundary of window.
- If the line is completely inside then display it fully. If the line is totally outside then discard that line.
- But. if the line is crossing to the window boundary, then we have to find the intersection point of the

line with the edge of window & draw the portion which lies inside.

Cohen - Sutherland outcode Algo:-



- This is one of the popular line clipping Algo.
- This algo immediately removes the lines which are lieing totally outside the window.
- This algo divides the plane in nine parts & assigns the outcode or binary no. to each part.
- each point of each line is assigned a four bit binary code which is called as outcode.
  four bit code abbreviated as "ABRL"
  A = Above
  B = Below
  R = Right
  L = Left
- The highest bit among four digits i.e A will be set to 1 if the end point of a line is above window. If the end point is not above the window then it is set to 0.
- If B bit is set, means the end point is below the window. if the

endpoint is inside the window then it is assigned a code as 0000. Zero to all four bits indicates that the end points is not above, not below, not right & not left of the window.

— The Cohen-Sutherland algo tells that if the line is totally on one side of the window then immediately we have to discard that line.

— Similarly if the line is totally inside the window directly display that line without clipping.

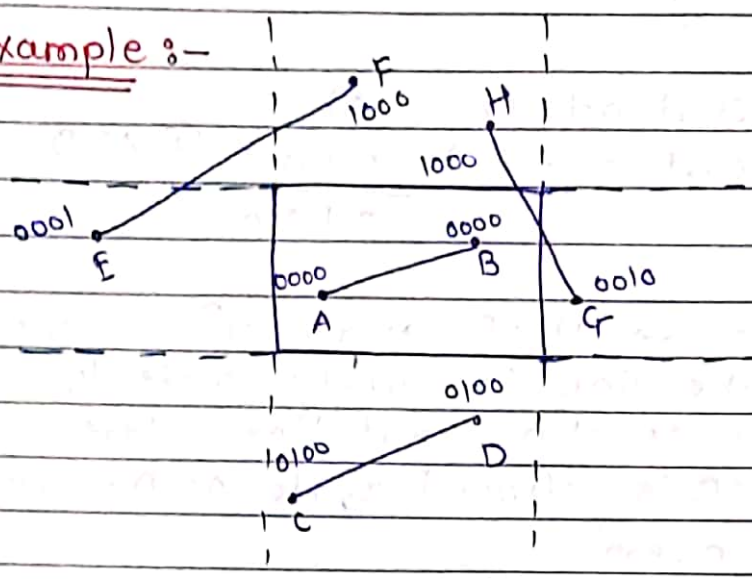
Cohen-Sutherland line clipping Algo:-

1. Accept the window co-ordinates from user & store them in x_L, x_H, y_L, y_H.
2. Accept the endpoints A & B of line with components A_x, A_y, B_x, B_y.
3. Initialize A & B code as array of size four.
4. Calculate A code & B code by comparing
   if $A_x < x_L$ then Acode(4) = 1 else 0
   if $A_x > x_H$ then Acode(3) = 1 else 0
   if $A_y < y_L$ then Acode(2) = 1 else 0
   if $A_y > y_H$ then Acode(1) = 1 else 0
5. check if A code & Bcode are 0000. If yes, display line AB. if not then proceed.
6. Take logical AND of Acode & Bcode, check the result, if it is non zero, discard the line & go to (8) else proceed.
7. Subdivide the line by finding intersection of a line with window boundary

and give new outcodes to the intersection point & go to (5)

8. ~~else proceed~~ stop.

Example :-



Case 1:-

Consider d line AB Here both end points of AB are surely inside the window.

∴ outcode for endpoint A will be

outcode (A) = ABRL
= 0000

outcode (B) = ABRL
= 0000

∴ algo says that if the outcodes of both endpoints of line are 0000, then the line is fully visible & we should not clip this time.

Case :-2

Consider line CD.

outcode (C) = ABRL
0100

outcode (D) = ABRL
= 0100

Now here for line CD outcodes of both endpoints of aline are not zero, so we have to perform logical AND operation of both outcodes.

0 0 0
0 1 0
1 0 0
1 1 1

outcode (C) = 0100
outcode (D) = 0100 ⊕ AND
0100

As the result of AND oper^n is nonzero then the line is surely outside window, so clip that line. Here line CD is clipped as its AND result is nonzero.

Case 3:- Consider line EF & GH. for line EF the outcode of E will be 0001 & out code of F 1000

⊕ AND E = 0001
F = 1000
0000

AND result is zero. it means the line EF is not lieing completely on any one side of window.

line GH

G = 0010
H = 1000
⊕ AND = 0000

— if we observe both lines GH & EF, the AND result of both lines is zero. But line GH is partially visible & line EF is fully visible..
So for such lines we have to go one step ahead.

— we have to find out, to which edge of the window the line is intersecting & then we have to find the intersection point.

— Consider line EF

E = 0001

F = 1000

we have to compare 1st bit of both outcodes.

E = 0001 ABRL

↑

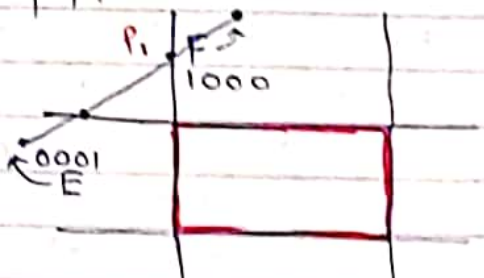
F = 1000

↑

1st bit of E is 1

& 1st bit of F is 0.

that means point E is line EF is lieing outside the left boundary of window so we have to find out intersection point of line EF with left boundary of window. call that intersection as P_1.



we are having two lines EP_1 & P_1F

— As point E is surely outside the left boundary of window so are clipping line EP_1.

— let consider P_1F Now we have to find outcode of P_1 which will be 1000.

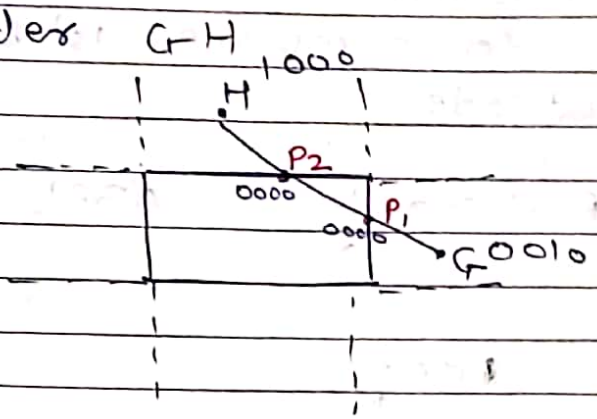
$P_1 = 1000$

$F = 1000$

So logical AND is 1000

— it means line P_1F is lieing completely on one side of window i.e above the window. so discard the line.

— consider GH



— here outcode are nonzero so take logical AND. & AND result will be zero. so we have to check individual bits of both outcodes.

outcode (G) = 0010

outcode (H) = 1000

∴ Both outcodes first bit is zero. it means both points are not lieing out side the left boundary of window. it means no interaction.

— so we have to check next bits.

outcode (G) = 0010 AND

outcode (H) = 1000

Second bit of G is 1. it means point G is outsides the right edge of window.

- So find out intersection point with right edge of window say point P_1. As point G is outside we have to discard line P_1G.
- Our line will be P_1H. The outcode of P_1 will be 0000

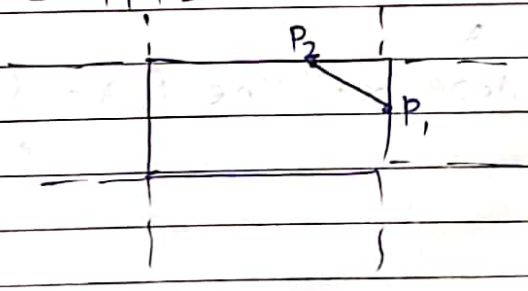
outcode (P_1) = 0000
outcode (H) = 1000

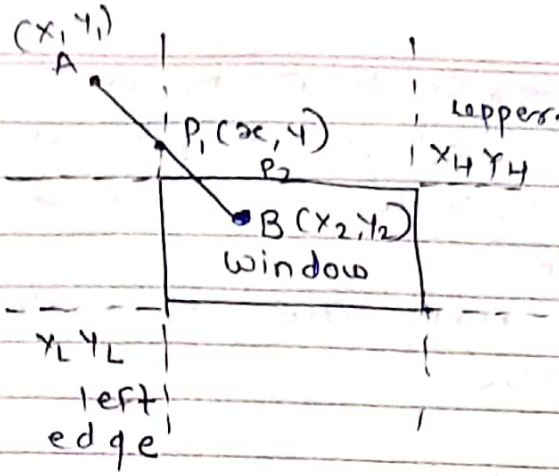
compare 3 bits of H & P_1. our line becomes P_1H we have to replace G by P_1.

- both bits are zero so compare last bits of P_1 & H.

(P_1) = 0000
(H) = 1000

- Here point H is outside, so find interaction with top boundary. Call that point as P_2 as point H is surely outside., we have to discard line HP_2.
- And now our line becomes P_1P_2. we have to find the outcode of P_2: which will be 0000. Again compare outcode of P_1 & P_2. both outcodes are zero so line P_1P_2 is visible





- The important factor in this algo is to find out the intersection point with window edge.
- if we want to find intersection with left edge of the window
- lower left corner of window will be (x_L, y_L), upper right corner will be (x_H, y_H).
- As we are intrested in finding intersection with left edge only. x-cordinate of intersection point must be x_L. i.e point must lie on left edge of window.
- now we have to find y-co-ordinate of intersection point.

for line AB

$$\text{slope}(m) = \frac{y_2 - y_1}{x_2 - x_1}$$

once the value of slope (m) is known. then we will find the slope bet^n intersection point P_1 & any one end point of line AB, say A.

$\therefore$ slope bet^n line $P_1A = \frac{(y - y_1)}{(x - x_1)}$

But here value of y is unknown & value of slope (m) is known

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

$$\therefore y - y_1 = m(x - x_1) \quad \text{here } x = x_L$$

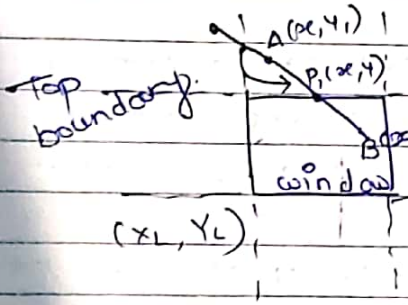
$$\therefore y = y_1 + m(x_L - x_1)$$

by using above equation, we can find y-coordinate for intersection point, so intersection points co-ordinates will be (x_L, y)

— similarly we can find the intersection point with right edge of window. Here x co-ordinate will be x_H of window & y-coordinate will be

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

— for top boundary the intersection point will lie on top edge of window y_H. & its x-cordinate will be calculated as



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

$$x - x_1 = \frac{y - y_1}{m} \quad \text{here } y = y_H$$

$$\therefore x = x_1 + \frac{y_H - y_1}{m}$$

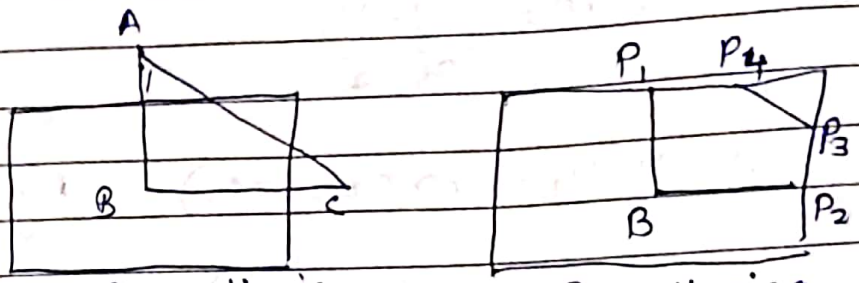
for bottom edge, y-co-ordinate will be y_L & x co-ordinate will be

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

[Reference : TB1]

Polygon Clipping -

- polygon - is a set of lines only. But it is closed figure.

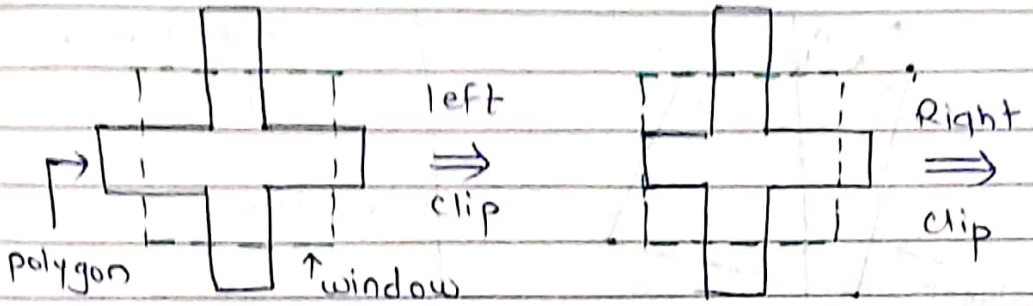


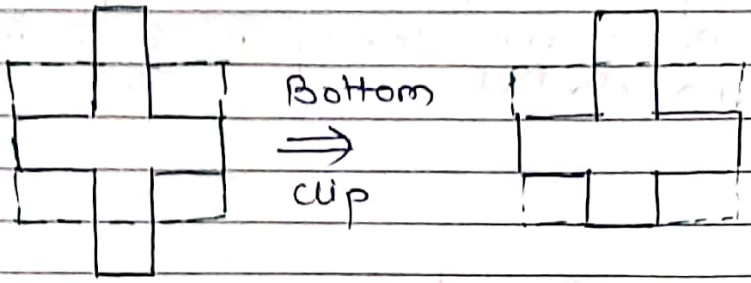
Before clipping after clipping.

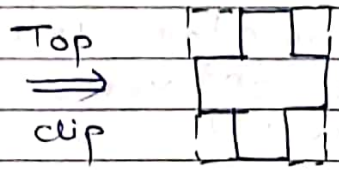
- clipping polygon may produce a polygon with fewer vertices or one with more vertices than the original one.
- it may even produce several unconnected polygons.

Sutherland-Hodgeman polygon clipping -

- we can clip a polygon by considering whole polygon against each boundary edge of a window.
- To represent a polygon we need a set of vertices.
- we will pass this set of vertices or a polygon to procedure which will clip the polygon against left edge of window.
- This left clip procedure generates new set of vertices which indicates left clipped polygon. Again this new set of vertices is passed to the right boundary clipper procedure.
- Again we will get new set of vertices. then we will pass this new set of vertices to bottom clipper & lastly to top boundary clipper procedure



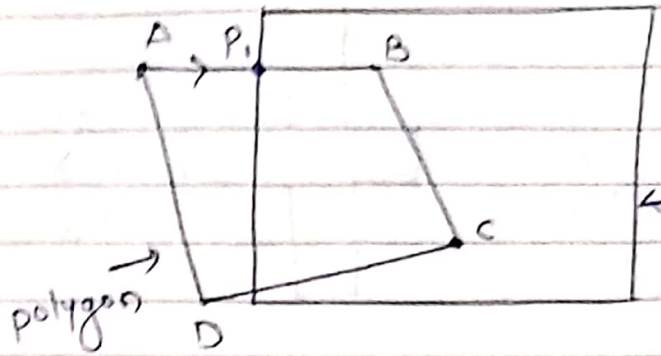




- At the end of every clipping stage a new set of vertices is generated & this new set or modified polygon is passed to next clipping stage.
- after clipping a polygon with respect to all the 4 boundaries we will get final clipped polygon.

Case I :-

- If the first vertex is outside the window boundary & second vertext is inside the window, then the intersection point of polygon with boundary edge of window & the vertex which is inside the window is stored in a output vertex list. i.e it will act as new vertex points.

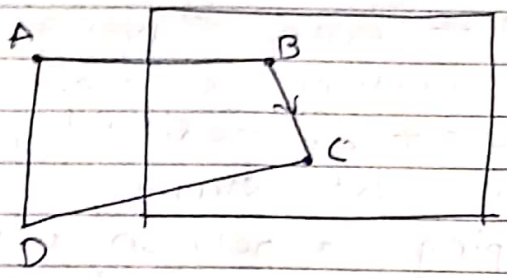


store P_1 & B in output vertex list.

for edge AB instead of storing vertex A & B we are storing P_1 & B in output vertex list.

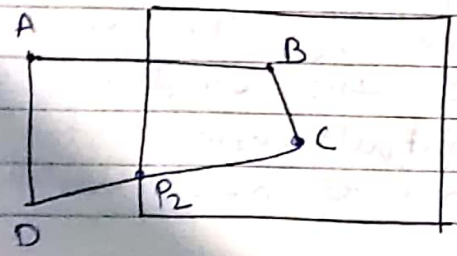
Case : 2

If both, 1st & 2nd vertex of a polygon are lieing inside the window, then we have to store the 2nd vertex only in output vertex list.



store only C in output vertex list.

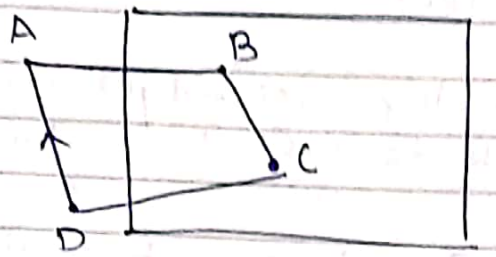
Case:3 if the 1st vertex is inside the window & 2nd vertex is outside the window i.e opposite to Case 1. then we have to store only intersection point of that edge of polygon with window in output vertex list.



store P_2 only in output vertex list.

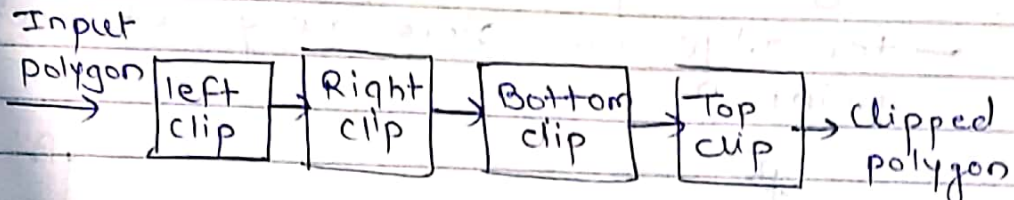
Case 3 4.

If both first & 2nd vertex of a polygon are lying outside the window then no vertex is stored in output vertex list.

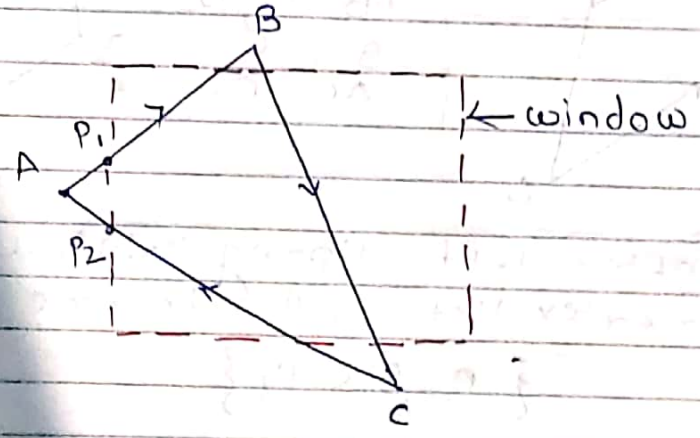


nothing is stored in output vertex list.

- once all vertices have been considered for one clip window boundary, the output list of vertices is clipped against the next window boundary.



Example :-



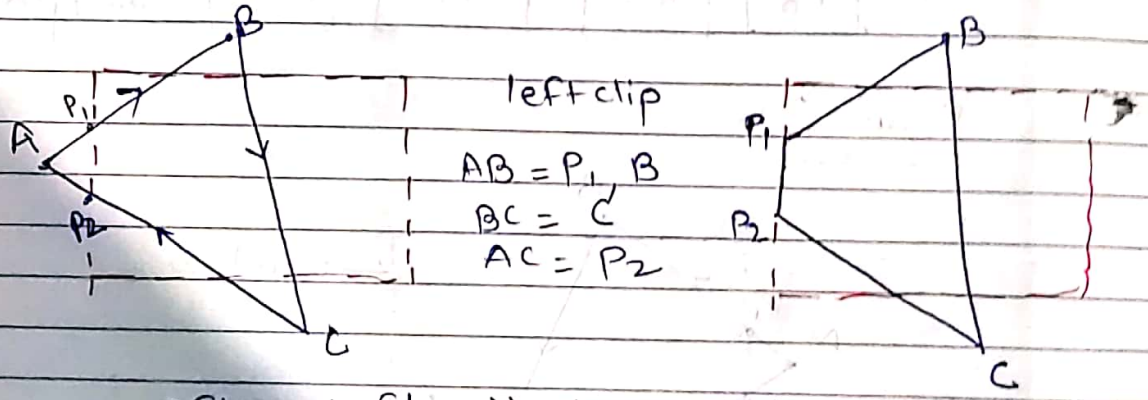
- To store a polygon ABC we will we an array to hold the vertices.
- So array will contain 3 vertices A, B & C.

- At the time of drawing a polygon, edge is drawn between vertex A & B then edge is drawn between B & C & finally to form a close figure. & edges will be AB, BC & CA.

- step 1 :- clip left

we will consider all edges of polygon with respect to left boundary of window.

- for edge AB, store intersection point of AB edge with left boundary i.e P_1 & inside point B.
- for edge BC, store only point C, as both B & C are inside the window. (consider only left boundary of window)
- for edge CA, store only intersection point P_2.



after left clipping output vertex list will become

$\{P_1, B, C, P_2\}$

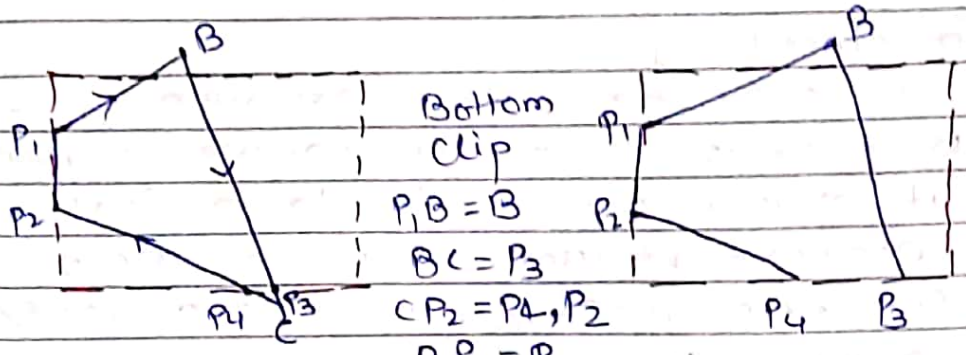
step : 2 clip Right

The output vertex list will not change after performing clip right

Procedure, because there is no edge which lie on right side of window.

step : 3 clip bottom

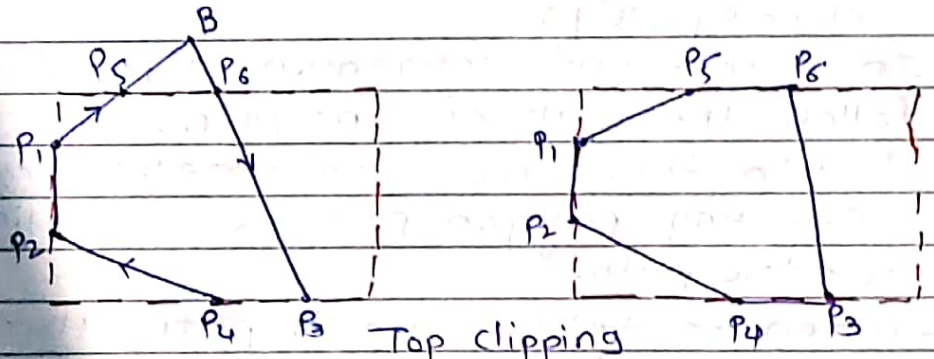
modified list of vertices is passed to this procedure.



after bottom clipping set of vertices be $\{B, P_3, P_4, P_2, P_1\}$

step : 4 clip top

The modified list of vertices is passed to this procedure.



so final output vertex list will become

$\{P_5, P_6, P_3, P_4, P_2, P_1\}$

- All convex polygons get correctly clipped by sutherland Hodgeman algo.
- Some concave polygons may be displayed with extra edges.
- problem of extra edge occurs since we are storing all the vertices in a single vertex list & to form a closed fig.
- To overcome this problem, one way is to split the concave polygon. it means we will divide a single concave polygon in two or more convex polygons & process each convex polygon separately.
- But dividing a concave polygon is not a simple method.

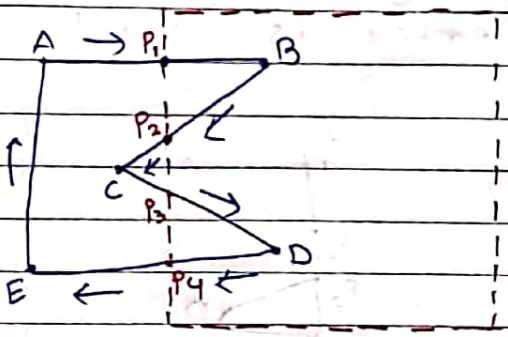
[Reference : TB1]

General Polygon clipping Algo:-

- weiler-Atherton polygon algo
- is one of the generalized polygon clipping algo.
- In sutherland Hodgeman algo, always follow the path of polygon. But in this case sometimewe are selecting polygon path & sometimes window path.
- when to follow which path, that will depend on polygon processing direction & whether a pair of polygon vertices currently being processed represents an outside to inside pair or an inside to outside pair.

- can follow polygon processing path either clockwise or anticlockwise. when we are following a path of polygon in clockwise direction at that time we have to we following rules -

- have to follow polygon boundary, same as that of sutherland - Hodgeman algo if the vertex pair is outside to inside.
- have to follow the window boundary in clockwise dire^n, if the vertex pair is inside to outside.

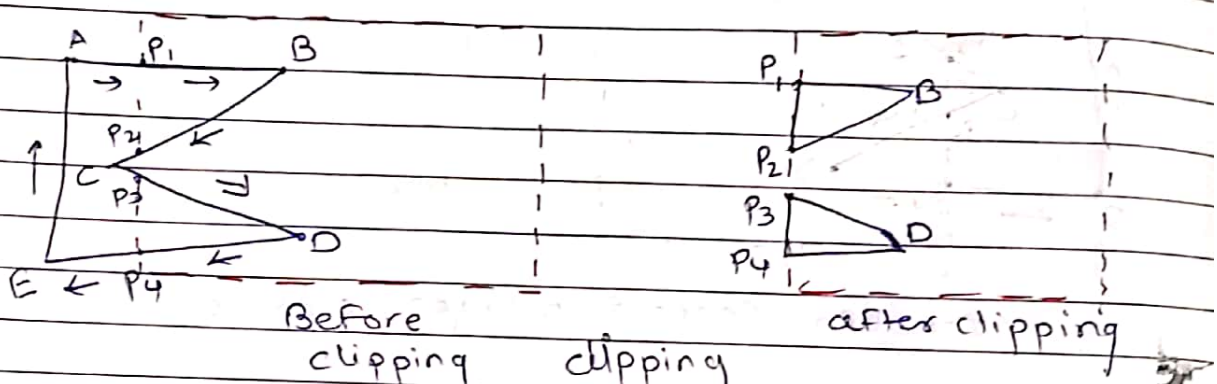


Consider Concave polygon ABCDE.

- let us process the polygon in clock wise path so the vertex list will be the set of all vertices i.e {A, B, C, D}

- for edge AB we are moving from outside to inside so store intersection point P_1 & second point i.e B, As we are following polygon in clockwise dire^n, so we have to consider next edge as BC, Here we are moving from inside to outside, so we are storing only intersection point P_2.

- upto this point it is similar to normal polygon clipping algo. as we are moving from inside to outside so we have to follow window boundary in clockwise diren & the next point on this path will be P_1. So we have to store that.
- it means we are storing P_2 & P_1. for next edge i.e CD, again we are storing intersection point window Boundary in clockwise diren. i.e storing P_3.
- for edge EA, as both points are outside, no vertex is stored



Before clipping

after clipping

AB = P_1, B

BC = P_2, P_1

CD = P_3, D

DE = P_4, P_3

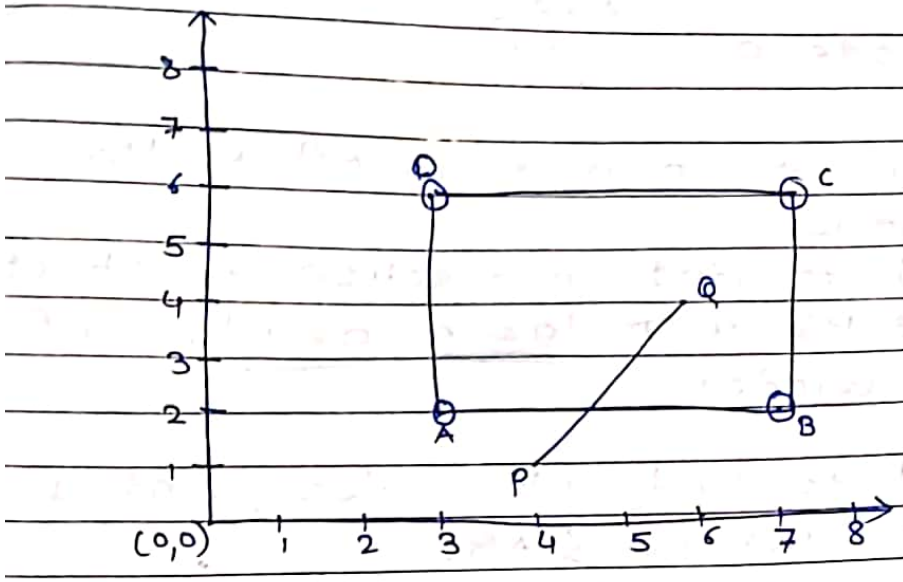
EA = —

- In this algo weiler Atherton algo, we are maintaing as such two different vertex list in single list.
- after clipping, vertex list will be {P_1, B, P_2, P_1, P_3, D, P_4, P_3}
- we have to draw edges by joining vertices as P_1 to B, B to P_2, P_2 to P_1,

- there is no need to form an edge between P_1 to P_3. We have to continue with edge P_3 to D, D to P_4 and P_4 to P_3.
- Again here there is no need to close the figure by drawing edge between P_3 to P_1 because already the figure is closed.
- Same thing for edge P_1 to P_3.
- Here how we will come to know when to not form edge, we may use some logic here such as when any vertex is stored twice in vertex list then don't form edge from that vertex to next vertex.
- we will not have edge P_1P_3 & P_3P_1. That's why we are saying. we are maintaining two sub array in single array of vertex.

clip the line PQ having coordinates A(4,1) & B(6,4) against the clip window having vertices A(3,2), B(7,2) C(7,5) & D(3,5) using Cohen Sutherland line clipping algo. mention

→ we draw line & window ABCD



⇒ from fig. it is clear that point P is lying outside the window & point Q is lying inside the window.

⇒ According to Cohen - Sutherland algo.

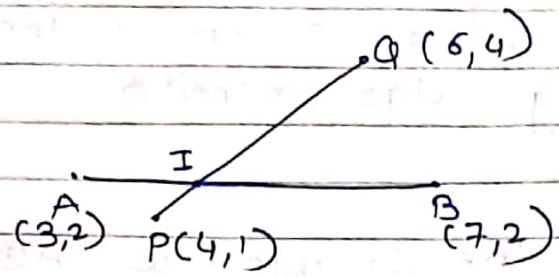
we need to find outcodes of endpoint of line PQ

∴ P = 0100
Q = 0000

outcodes are nonzero & need to logically AND them.

∴ 0100
⊕ 0000
0000 AND result is zero.

⇒ we have to decide PQ may be visible or may be partially invisible.



outcode P = 0100 (A B R L)

→ means the point P is below the window.

→ need to find intersection point of line PQ with lower boundary of window.

→ need to find intersection point I of lines PQ & AB.

Y value of intersection point I is known

$I = 2$

need to find X value.

slope of line PQ =

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

$$= \frac{4-1}{6-4}$$

$$= \frac{3}{2}$$

$$= 1.5$$

$\therefore \boxed{m = 1.5}$

$$X = X_1 + \frac{y_{window} - y_1}{m}$$

$$= 4 + \frac{2-1}{1.5}$$

$$= 4 + \frac{1}{1.5}$$

$$\therefore \boxed{X = 4.66}$$

. let ABCD be rectangular window with A(20,20), B(90,20), C(90,70) & D(20,70). find region codes for endpoint & use cohen sutherland algo. to clip the lines.

$P_1P_2 \rightarrow P_1(10,30)$ & $P_2(80,90)$

$q_1q_2 \rightarrow q_1(10,10)$ & $q_2(70,60)$

[Reference : TB1]

Filling with Pattern :-

- filling with pattern, which we do by adding extra control to the part of the scan conversion algo that actually writes each pixel.
- main issue for filling with pattern is the relation of area of the pattern to that of the primitive. i.e we need to decide where the pattern is anchored so that we know which pixel in the pattern corresponds to the current pixel of primitive.

Technique I :-
is to anchor the pattern at a vertex of polygon by placing the leftmost pixel in pattern's 1st row.
- This choice allows the pattern to move when the primitive is moved, visual effect that would be expected for patterns with a strong geometric organization.

Technique II :- Consider entire screen as being tiled with the pattern & to think of the primitive as consisting of an outline or filled area of transparent bits that let the pattern show through.
To apply the pattern to the primitive, we index it with the current pixels (x, y) coordinates.
- patterns are defined as small p by 0 bitmaps, we use modular

arithmetic to make the pattern repeat. The pattern [0,0] pixel is considered coincident with the screen origin & we can write, a bitmap pattern in transparent mode with the statement.

```
If (pattern [x % P] [y % Q])
    write pixel (x, y, value);
```

if we are filling an entire span in replace write mode, we can copy a whole of the pattern at once. assuming low level version of a copy pixel facility is available to write multiple pixels.