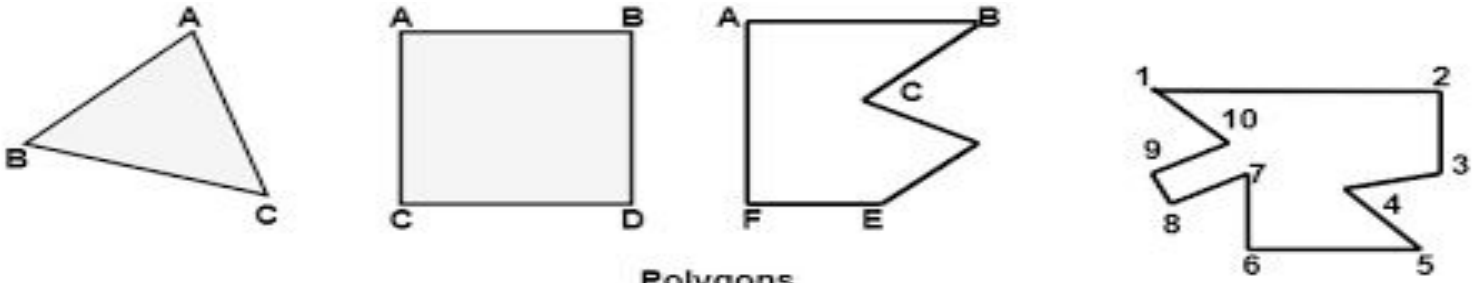
Computer Graphics
Unit 2 Polygon, Windowing & Clipping

Dr. D.Y.Patil, Pimpri
Ms. Chetana Shravage

Polygon

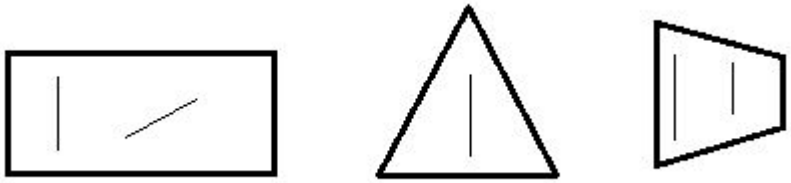
- Polygon having many sides.It may be represented as a number of line segments connected end to end to form a closed figure.



Polygons

- **Types of Polygon-**

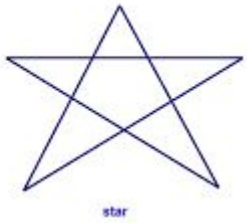
1) **Convex-**

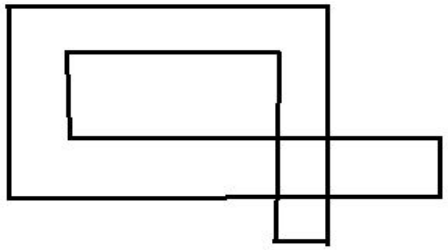


It is the polygon in which if we take any 2 points which are inside polygon & if we draw a line joining these 2 points, all the points on that line lies in the polygon called convex polygon.

2) Concave-It is the polygon in which if we take any 2 points which are inside polygon & if we draw a line joining these 2 points, some points on that line lies in the polygon and some are outside called convex polygon.

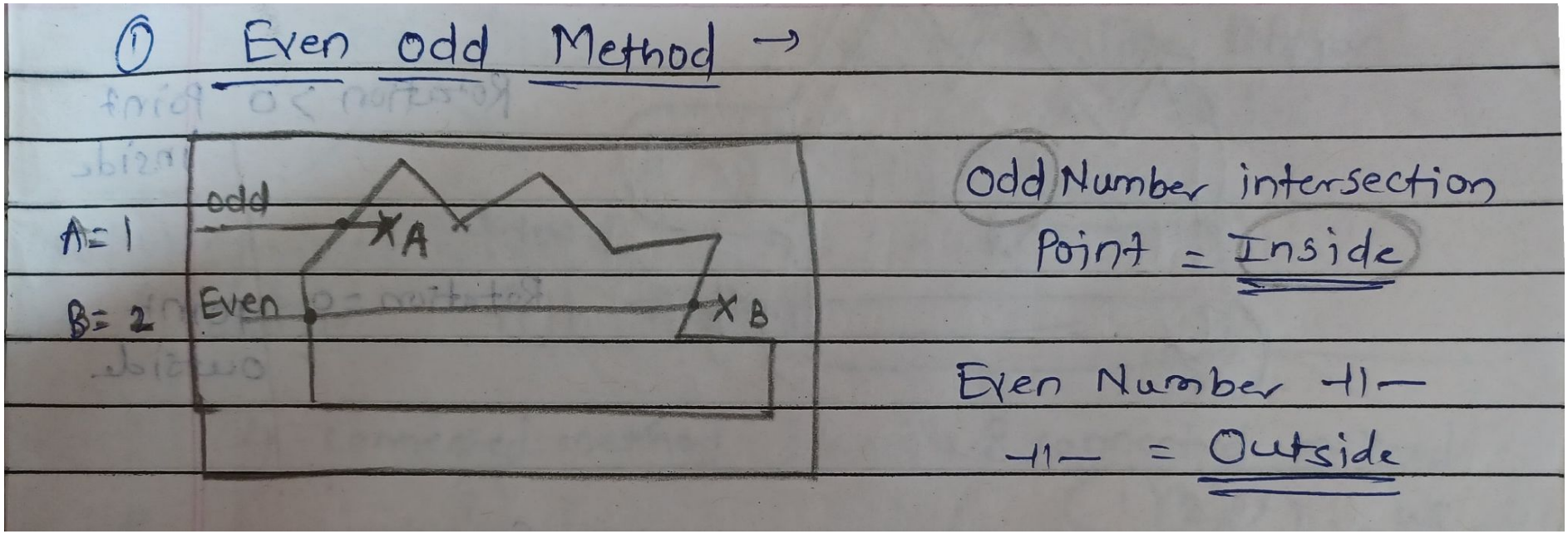
3) Complex-A polygon neither Convex nor concave.A polygon which intersect itself or overlap itself.





Inside Test of Polygon

1) **Even Odd Method**- If odd no. of intersection then point inside , if even then outside.



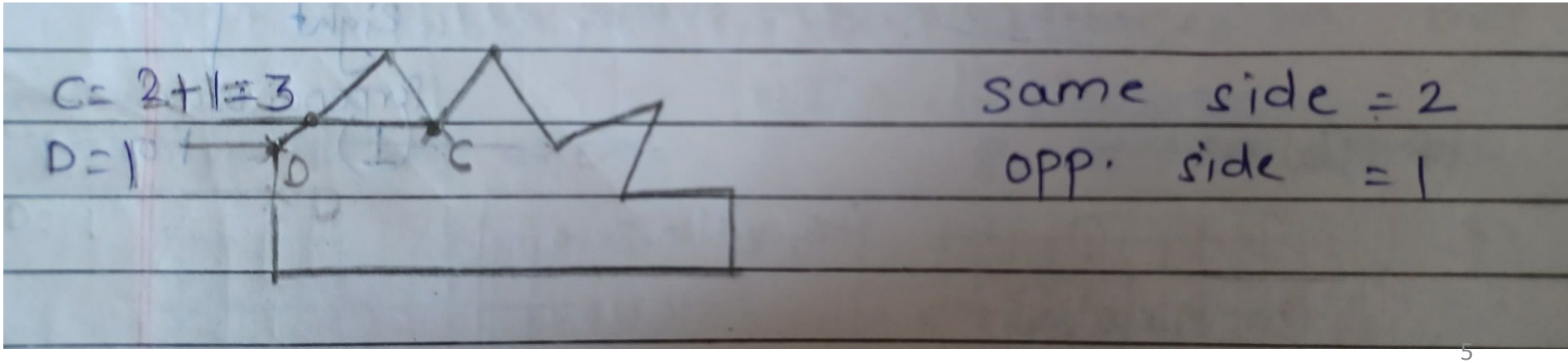
Inside Test of Polygon..

1) **Even Odd Method**…

If odd no. of intersection = inside

even = outside

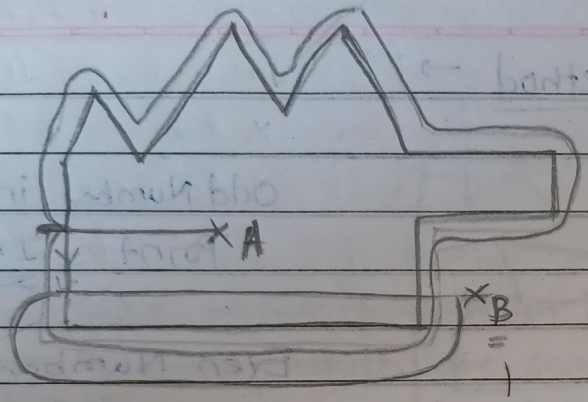
but when intersection point is a vertex then look at other endpoints of 2 segments of a polygon if lie on same side count as a 2 otherwise 1.



Inside Test of Polygon..

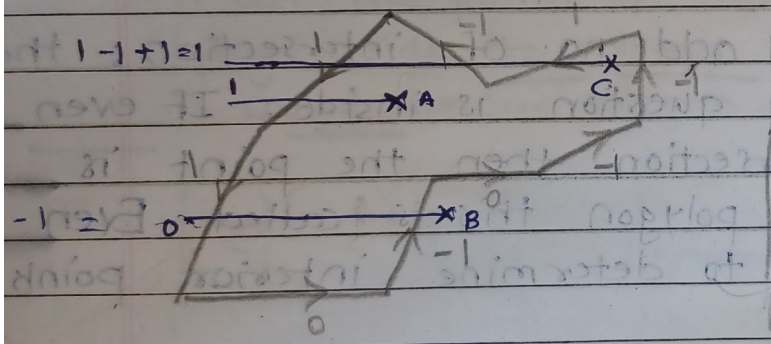
2) **Winding Number Method-**

Second method of inside test to check each pixel is inside or outside.



Rotation > 0 Point inside

Rotation = 0 point outside



Give Anticlockwise direction to line of polygon

if line downside = 1
upside = -1
Left } 0
Right }
Perpendicular (I) = 1 or -1

if Value:

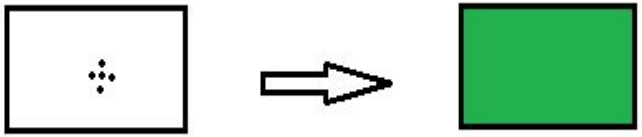
0	Non-Zero
Outside or Boundry	Inside

Boundry / outside Pixel → No colour.

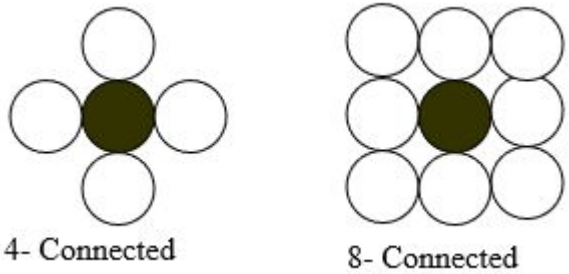
Polygon Filling

Polygon filling types are-

1) **Seed Fill Polygon fill Algorithm**
   a) Boundary fill Algorithm
   b) Flood fill Algorithm
2) **Scan line Polygon fill Algorithm**

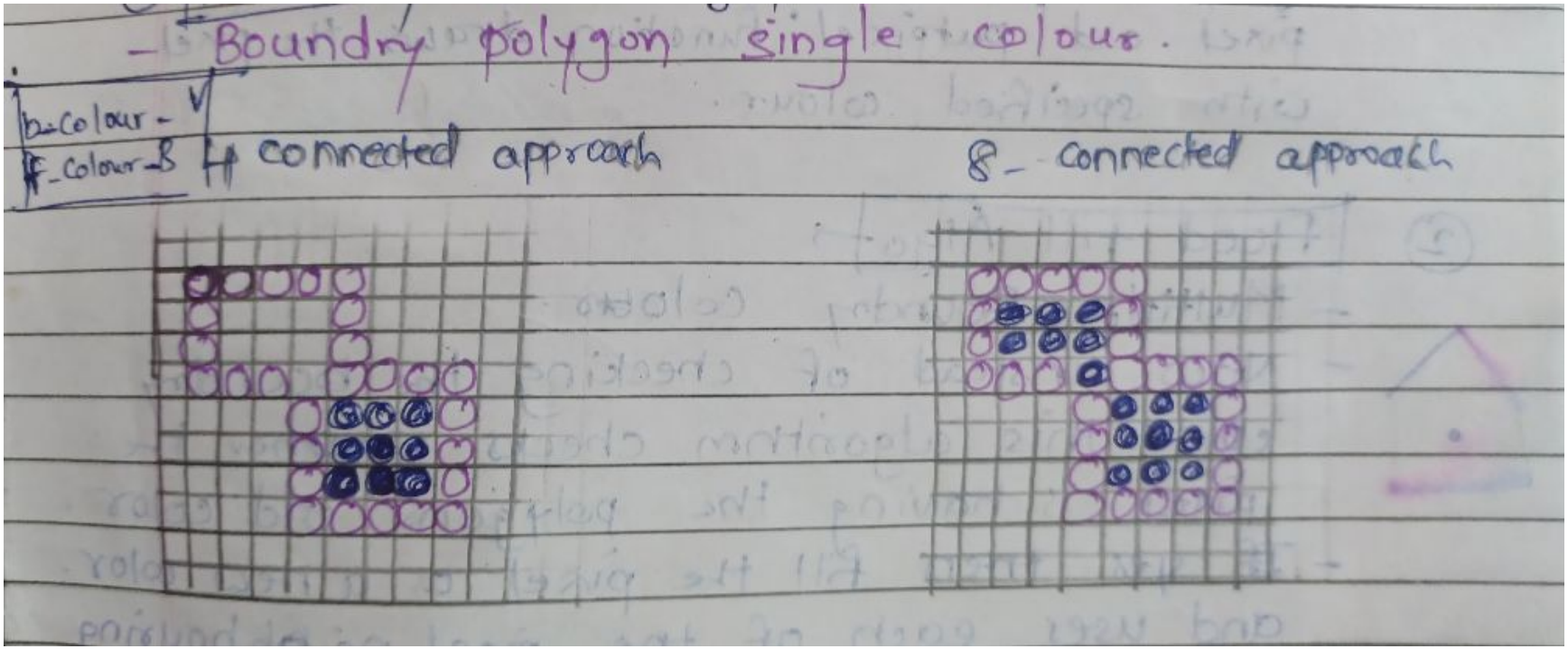


Before algorithm see 2 methods



a) Boundary fill Algorithm

- Single boundary color

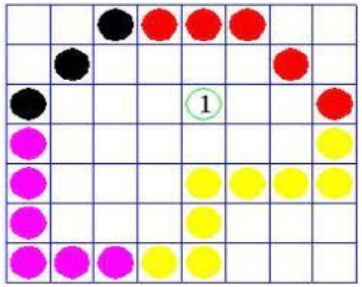


a) Boundary fill Algorithm…

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

Note-putpixel()- Draw the pixel with specified color
getpixel()- Return the color of specified pixel.

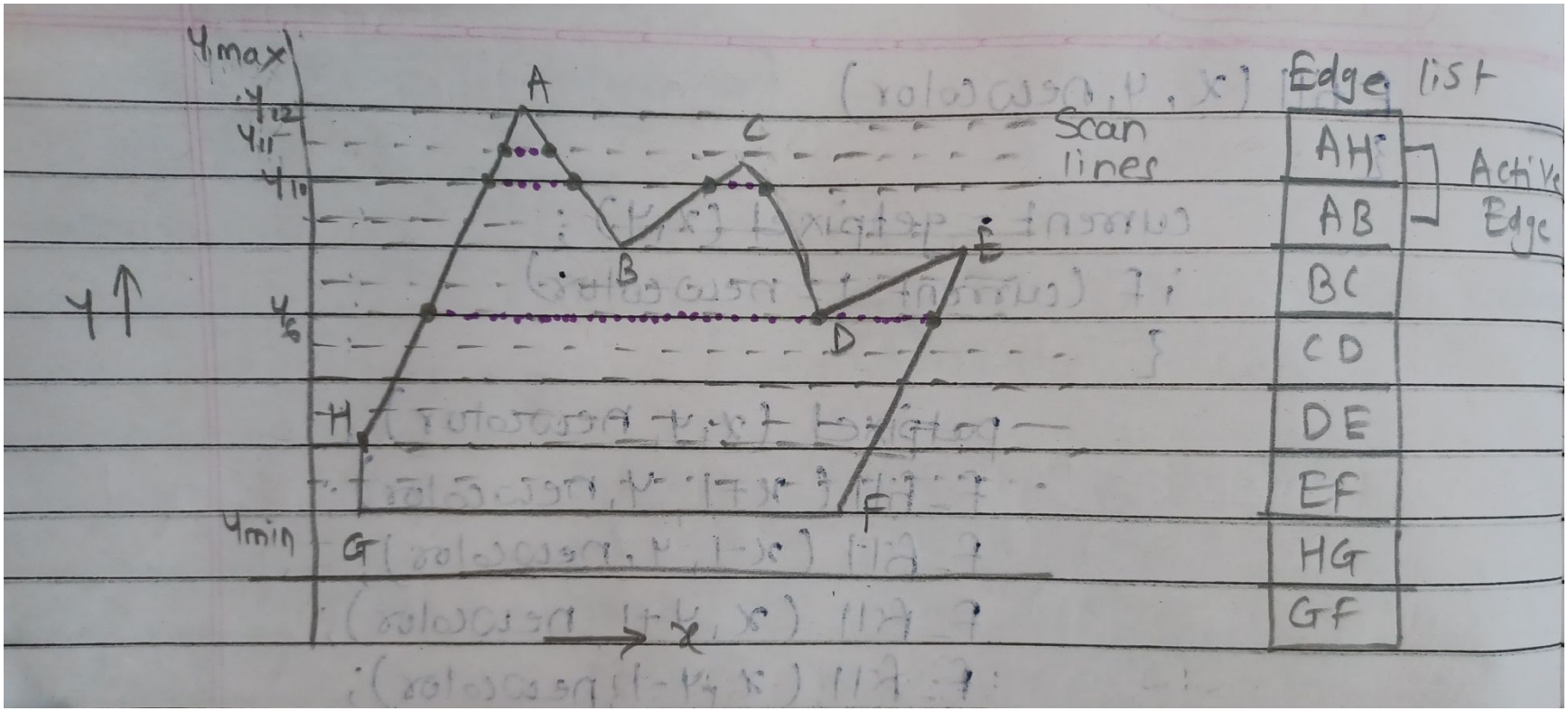
b)Flood Fill Algorithm



-Multiple Boundary color
–The efficiency of flood fill more than boundary fill algorithm.

```
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);
    }
}
```

2) Scan Line Algorithm



2) Scan Line Algorithm…

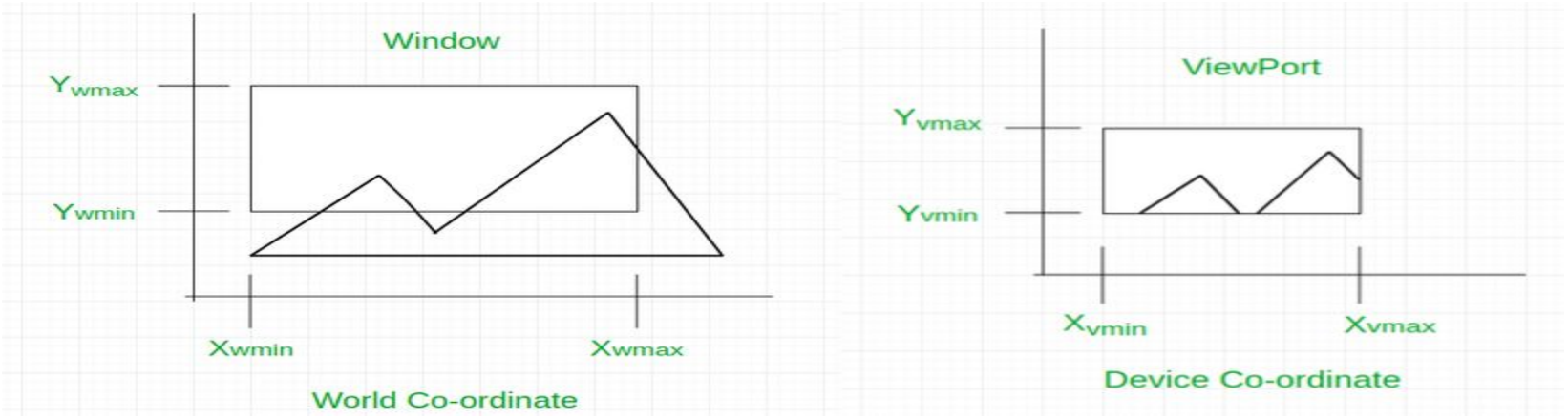
1) Locate the intersection points of the scan line with the polygon edges.
2) Pairing intersection points.{(7,11),(8,11)}
3) Move down side as per scan line & sort all pairs.{(7,11),(8,11)}, {(6,10),(9,10)}, {(15,10),(17,10)},.....{(4,6),(23,6)}, {(23,6),(26,6)},.....
4) All pairs are sorted ymax to ymin
5) Sides get sorted on intersection point bases.
6) Area fills starts now.

Windowing & Clipping

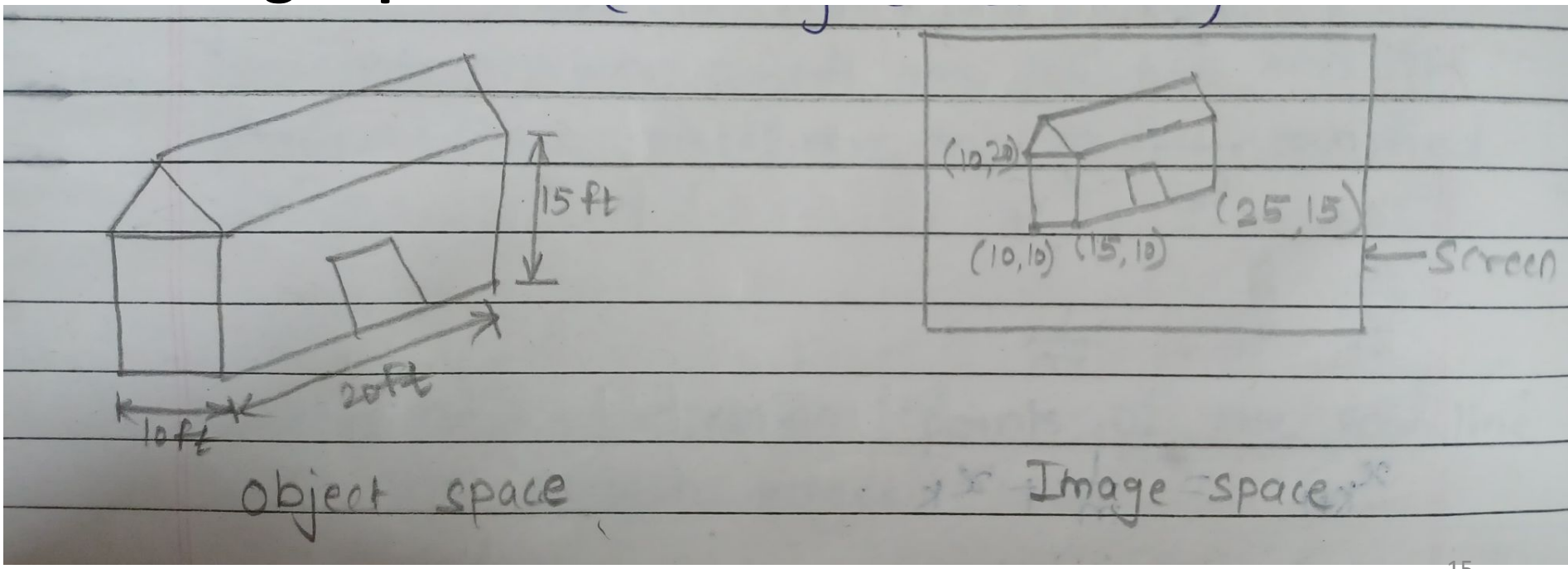
Viewing Transformation-

Window-Part of the picture to be displayed.
(world coordinate)

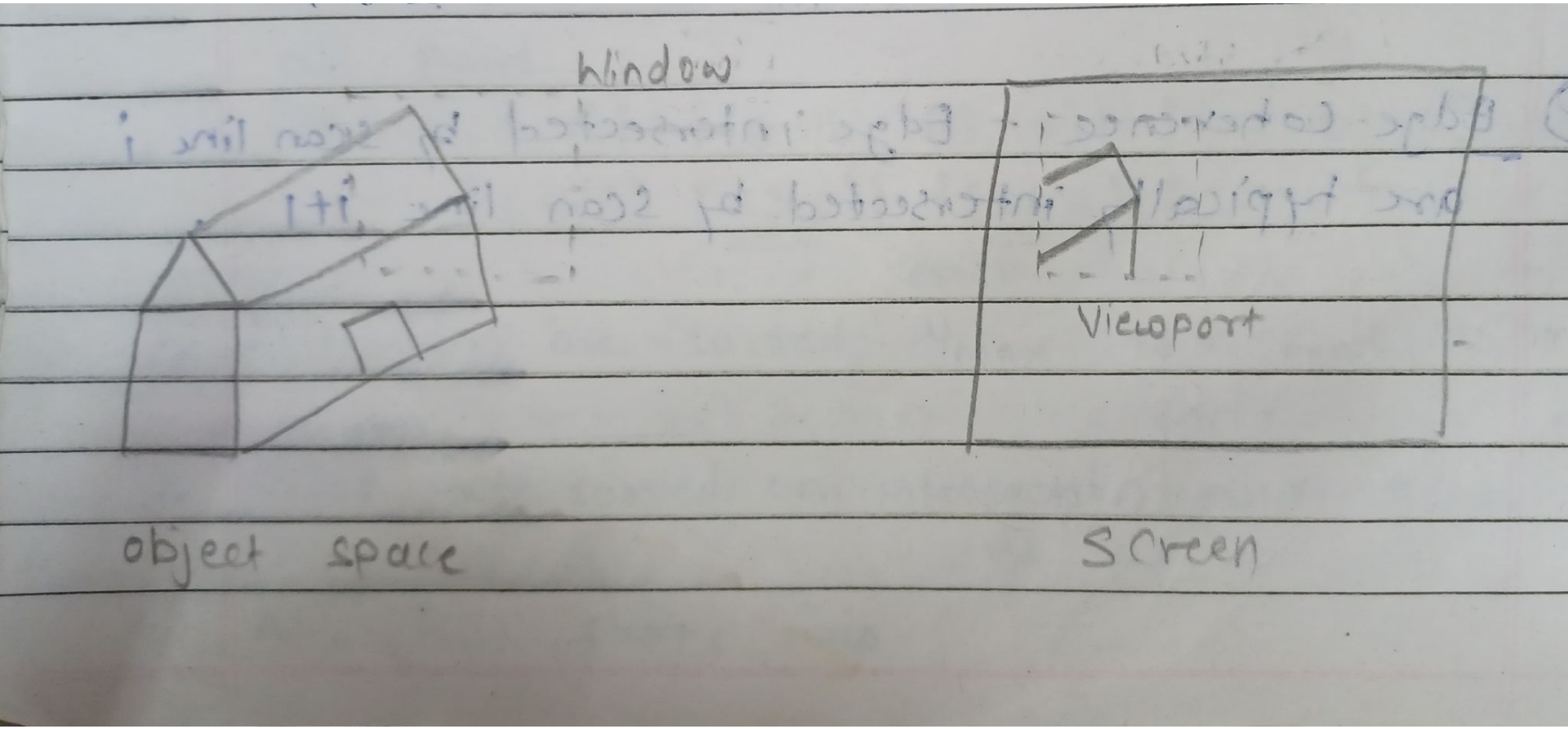
Viewport-Where the area picture to be displayed.
(viewing /device coordinate)



- The space where object model reside is called **object space**.
- Image model is one which appears on display device.
- The space where image model reside is called **Image space**.

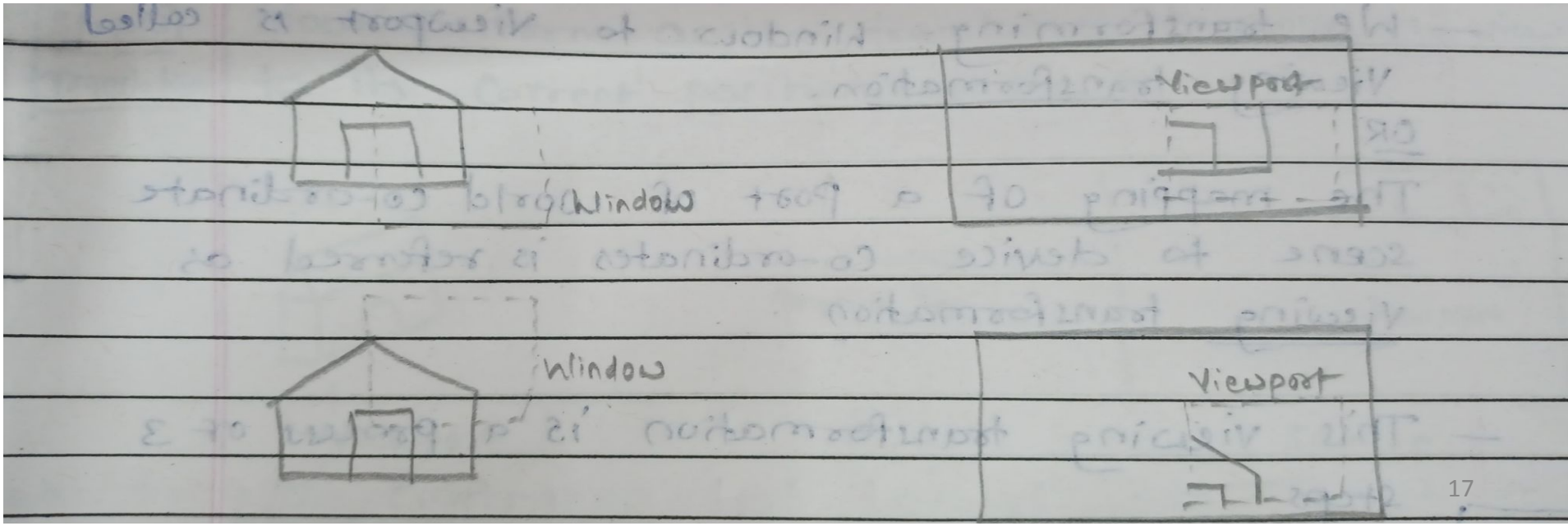


Exa. Window & Viewport



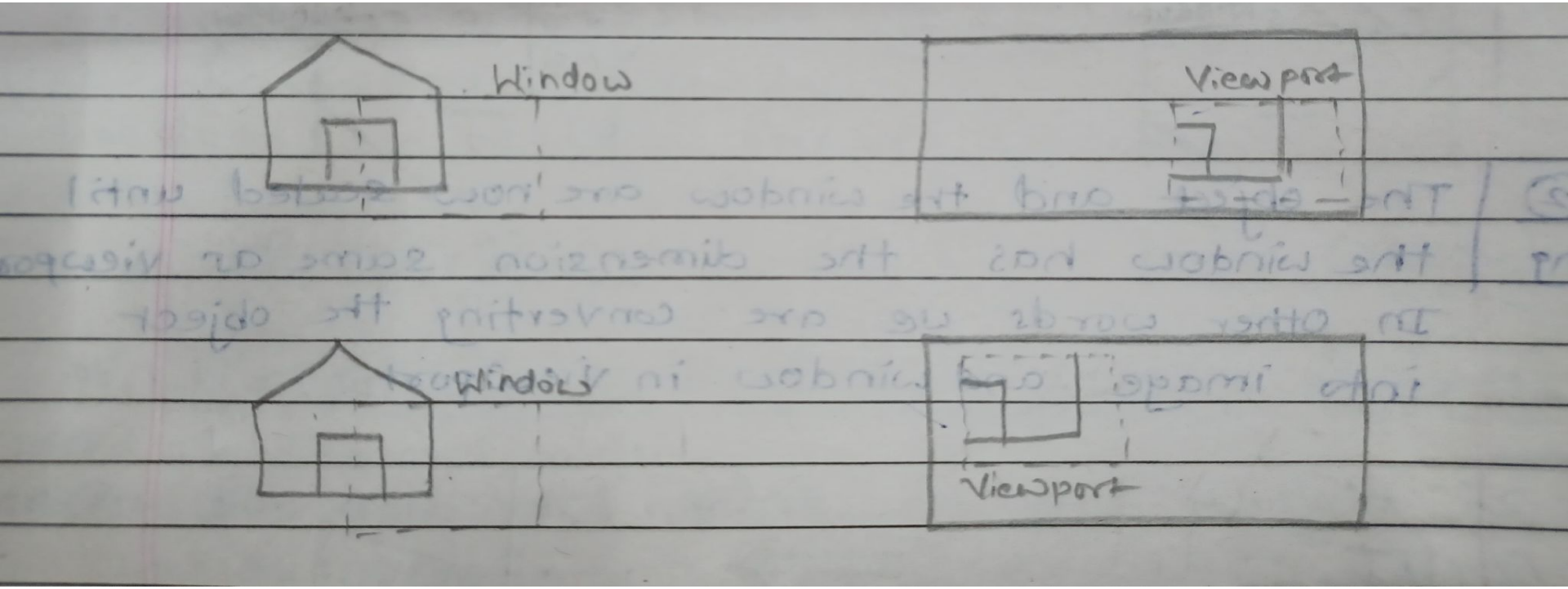
Different window same viewport-

- If we are changing the position of window by keeping the viewport location constant then the different part of the object is displayed at same position on the display device.



Different viewport same window-

- Similarly if we are change the location of viewport then we will see the same part of the object drawn at different places of screen.



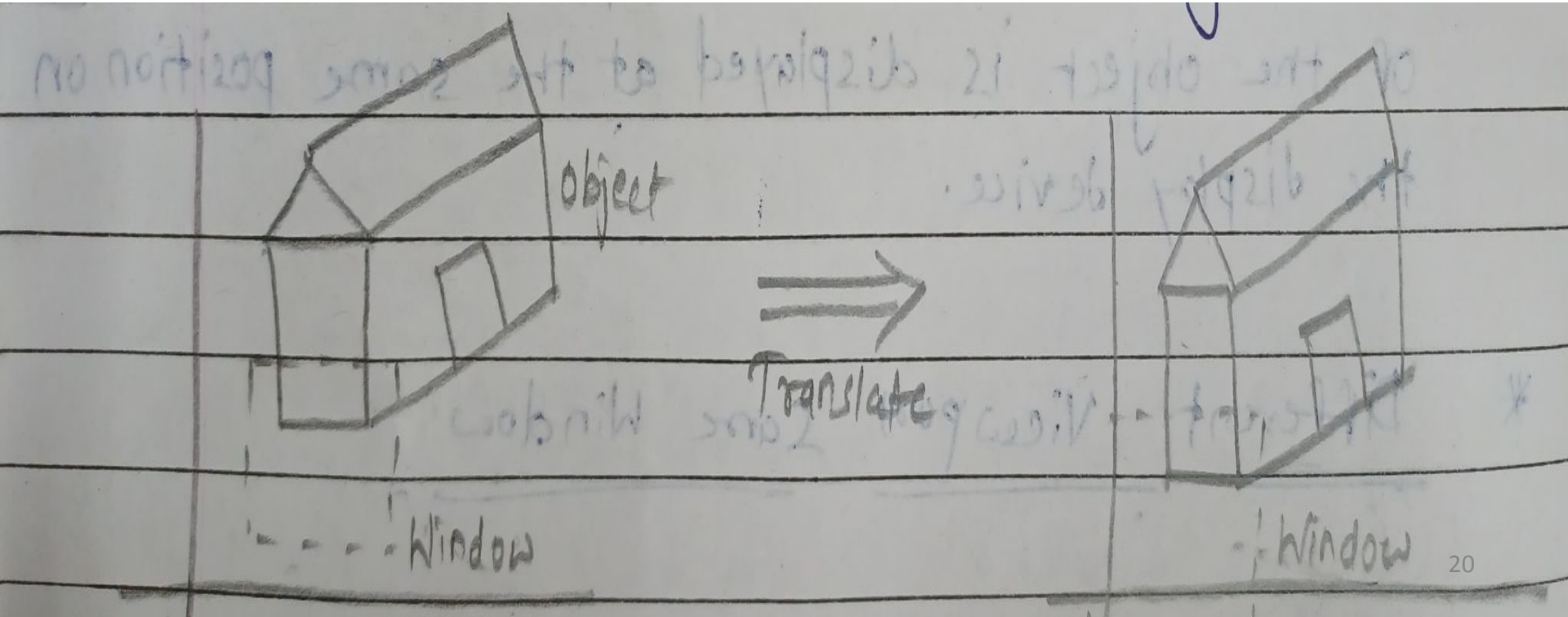
Viewing Transformation

- Transforming window to viewport is called viewing transformation.
- The mapping of a port of world coordinate scene to device coordinates is referred as viewing transformation.
- This viewing transformation is a process of 3 steps.

  1) Translation
  2) Scaling
  3) Translation

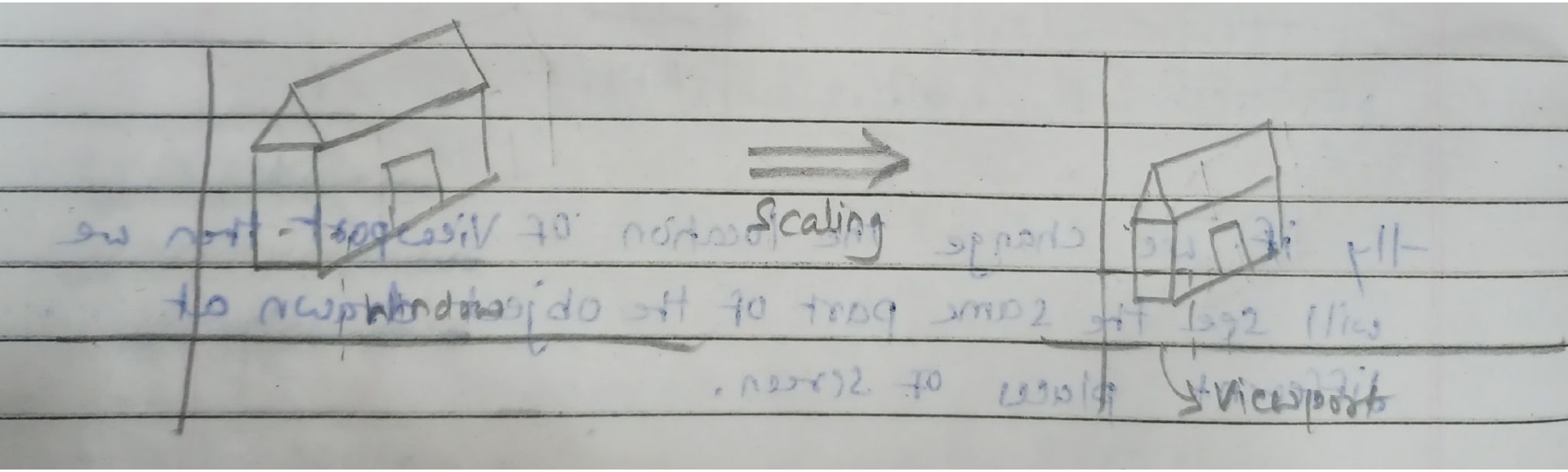
Step 1) Translation-

- The object with its window is shifted / translated until the lower left corner of the window matches to the origin.



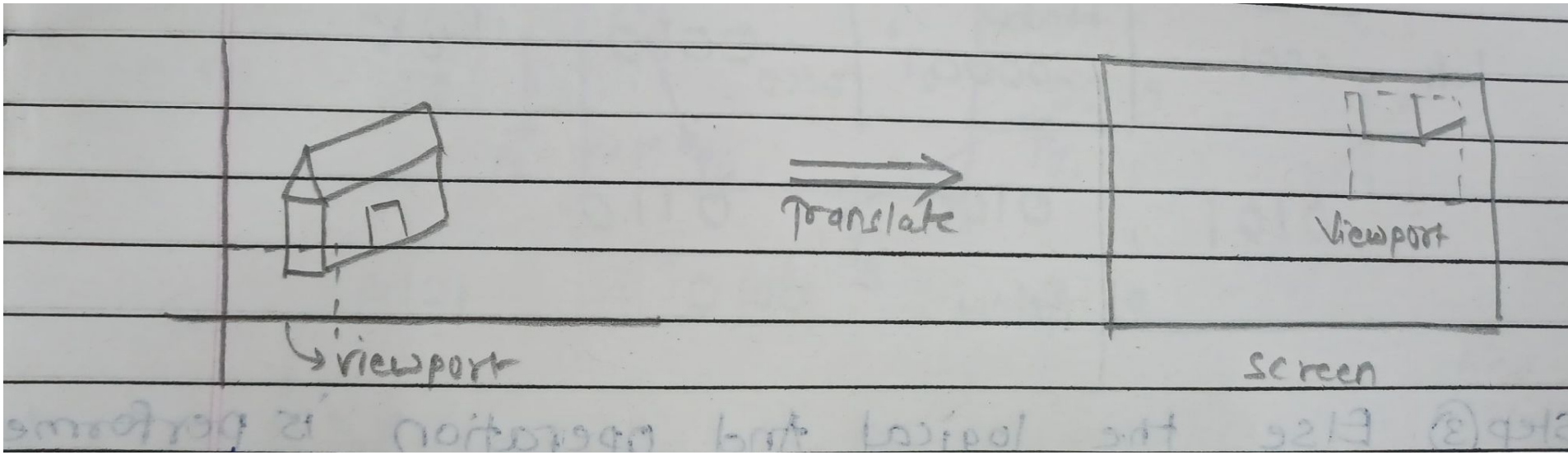
Step 2) Scaling-

- The object and window are now scaled until the window has the dimension same as viewport. In other words we are converting the object into image & window in viewport.



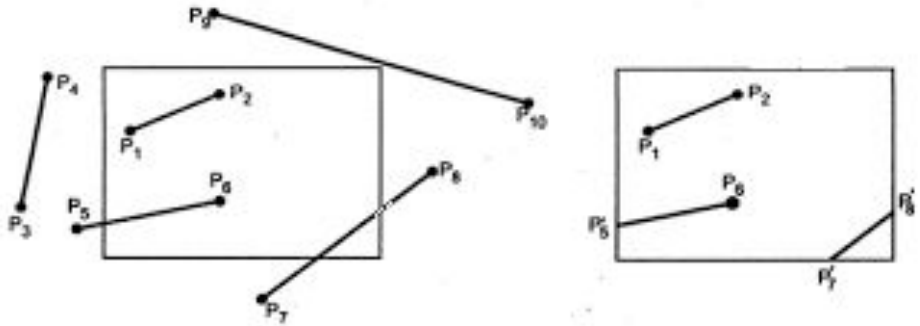
Step 3) Translation-

- Now perform another translation to move the viewpoint to its correct position on the screen.



2D Clipping

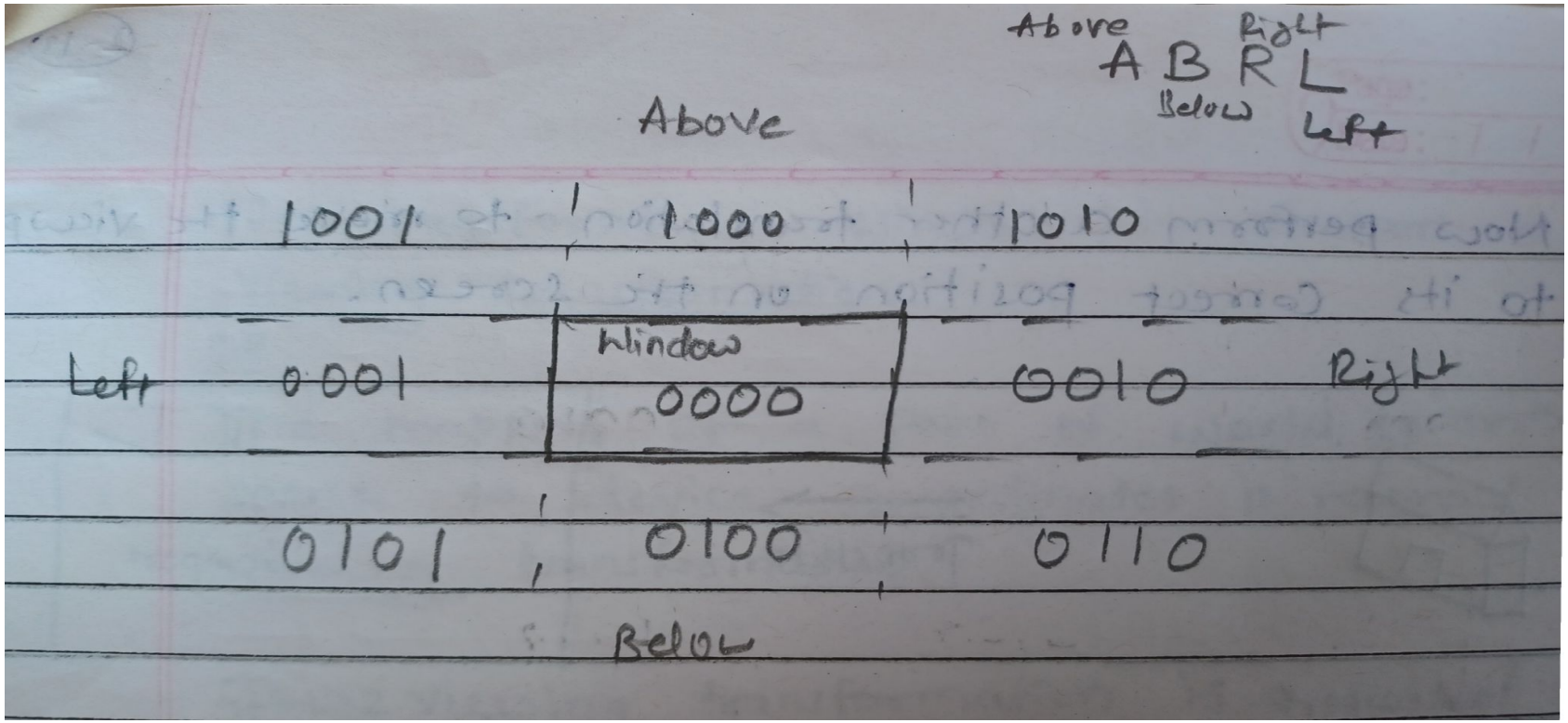
- Clipping means deleting the part of picture which lies outside the window and displaying the portion of picture which is inside the window.
- **Line Clipping-**



(a) Before clipping (b) After clipping

Cohen Sutherland Line Clipping Algorithm-

Step 1-For each points a region code is assigned.



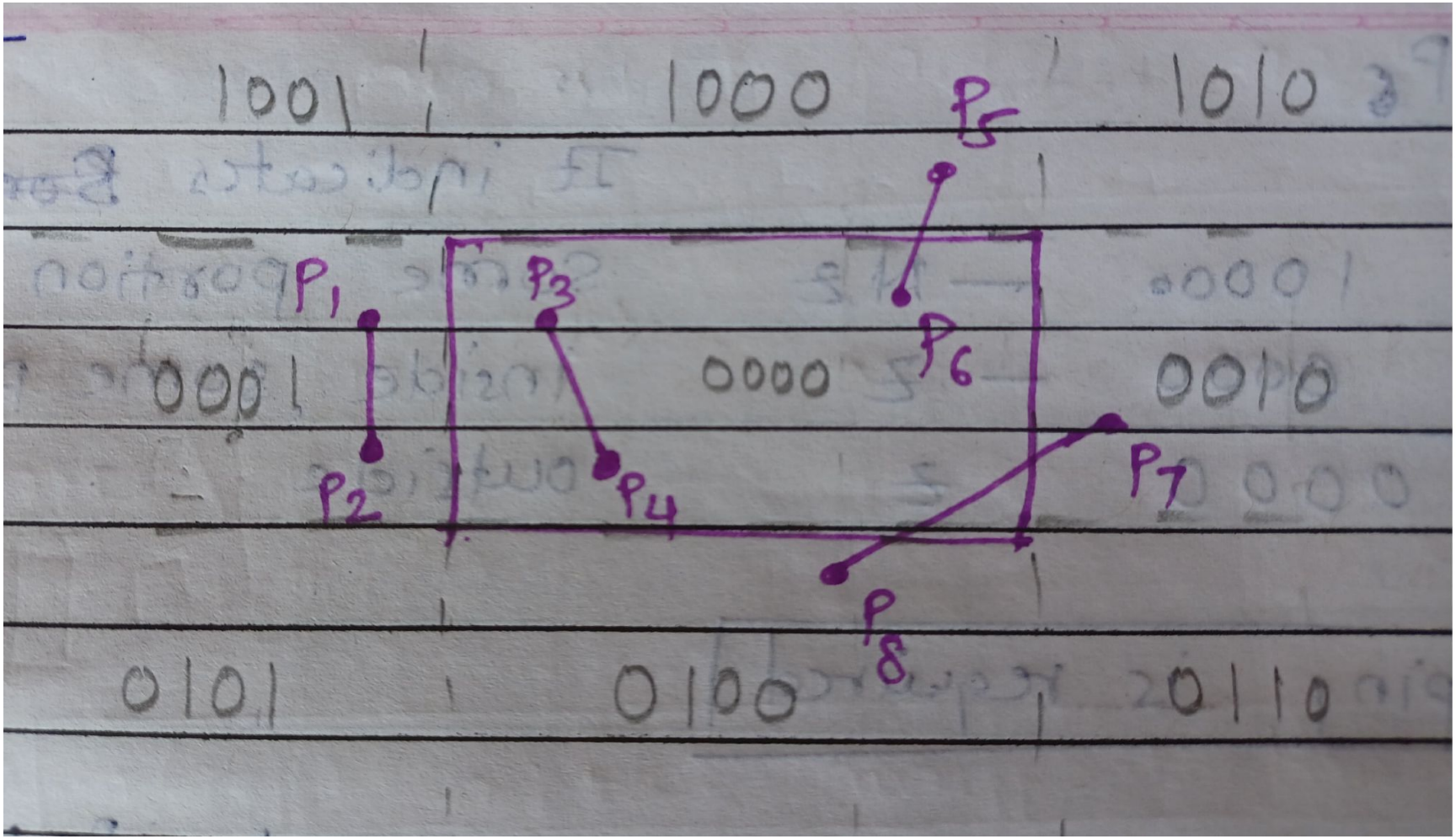
Step 2- The line is accepted if both end points have a region code 0000.

Step 3- Else the logical AND operation is performed for both region code.

- Step 3.1- If the result is not 0000 then Reject the line.
- Step 3.2- Else Clipping is required.
  - Step 3.2.1- An end point of line is selected that is outside the window.
  - Step 3.2.2- Find the intersection point at the window boundary.
  - Step 3.2.3- The end point is replaced with the intersection point & the region code is updated.
  - Step 3.2.4- Repeat Step 2 until we find a clipped line either trivially accepted or trivially rejected.

Step 4- Repeat from Step 1 for other lines.

Example


1001
1000
1010
P5
P1
P3
P6
0001
0000
0010
P7
P2
P4
P8
0101
0100
0110

Case 1- P1 & P2

P1 - 0001 - Nonzero

AND P2 - 0001 - Nonzero

—----------

0001 - Nonzero

AND Truth Table

Inputs		Output
A	B	Y = A.B
0	0	0
0	1	0
1	0	0
1	1	1

If all Nonzero are there it indicated P1 & P2 are completely outside the window.

Reject Line

Case 2- P3 & P4

P3 - 0000 - zero

AND P4 - 0000 - zero

—----------

0000 - zero

No Clipping is Required.

<u>Accept Line</u>

Case 3- P5 & P6

P5 - 1000 - Nonzero

AND P6 - 0000 - zero

—----------

0000 - zero

It Indicate some portion lies inside, some portion outside.

Clipping is required.

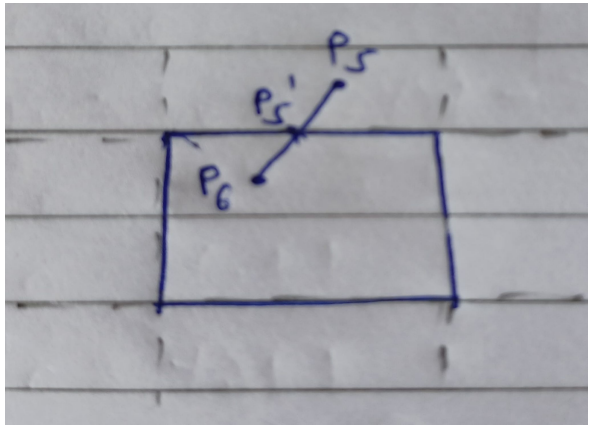
Apply P5’ & P6

P5’ - 0000 - zero

AND P6 - 0000 - zero

—----------

0000 - zero



Accept P5’ & P6 and Clip P5 & P5’

Case 4- P7 & P8

P7 - 0010 - Nonzero

AND P8 - 0100 - Nonzero

—----------

0000 - zero

Clipping is required.P7 & P7’ Clipped.

Now Calculate P7’ & P8

P7’ - 0000 - zero

AND P8 - 0100 - Nonzero

—----------

0000 - zero

Again Clipping required. P8 & P8’ Clipped.

Now Calculate P7’ & P8’

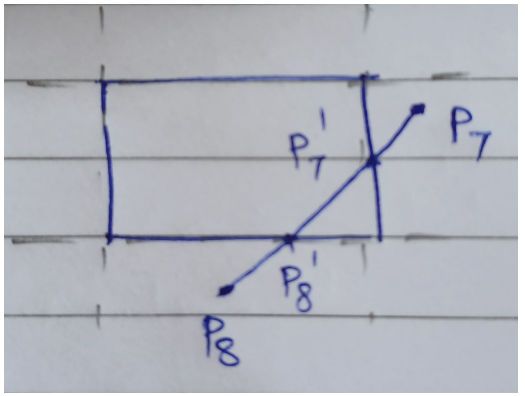
P7’ - 0000 - zero

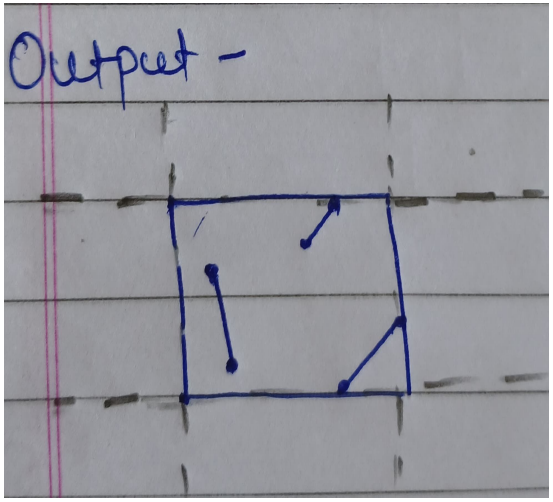
AND P8’ - 0000 - zero

—----------

0000 - zero

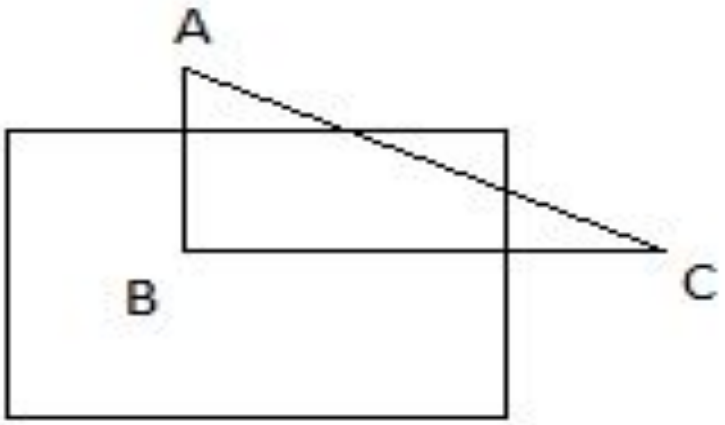
Accept the line P7’ P8’



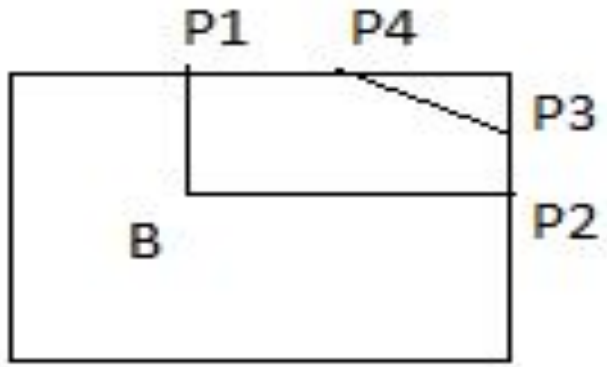


Polygon Clipping-

Clipping polygon may produce a polygon with fewer vertices/ one with more vertices than the origin one.



Before Clipping



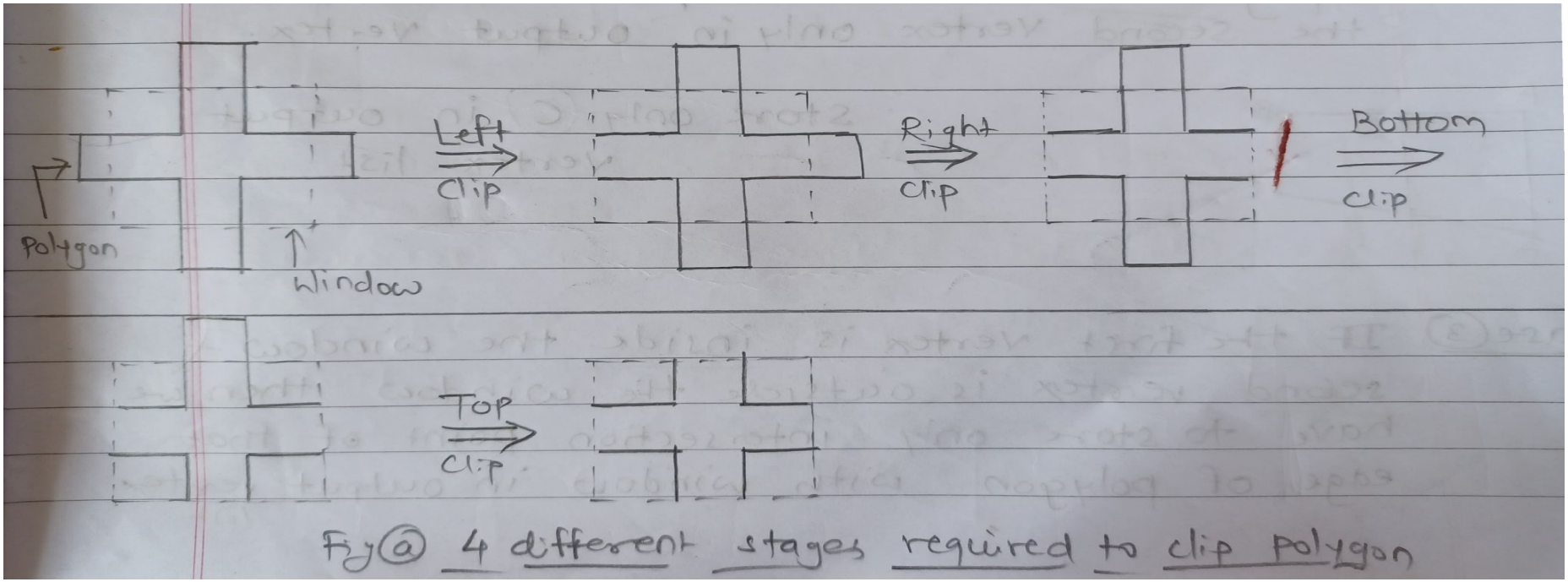
After Clipping

Polygon Clipping Algorithm-

1) Sutherland Hodgeman Polygon Clipping Algorithm
2) Weiler Atherton Polygon Clipping / Generalized Clipping Algorithm

Sutherland Hodgeman Polygon Clipping Algorithm

We can clip a polygon by considering whole polygon against each boundary of the window.



Sutherland Hodgeman Polygon Clipping Algorithm…

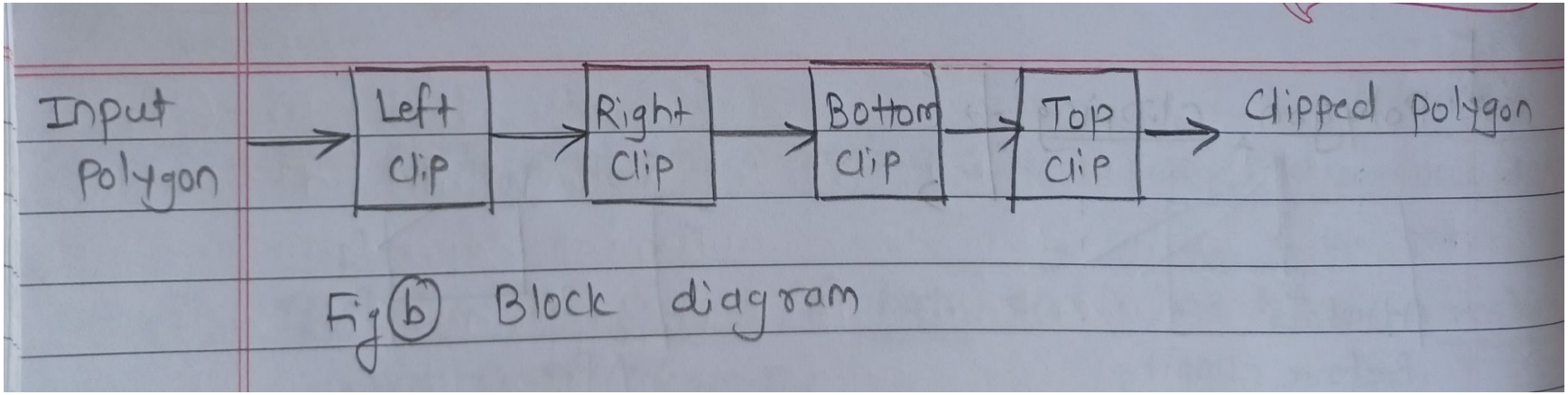
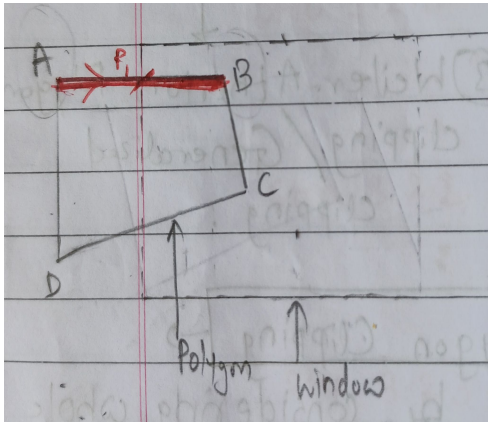


Fig (b) Block diagram

For Clipping we have 4 cases.

Case 1- If first vertex is outside the window boundary and second vertex is inside the window then the intersection point of polygon with boundary edge of window is stored in a output vertex list.



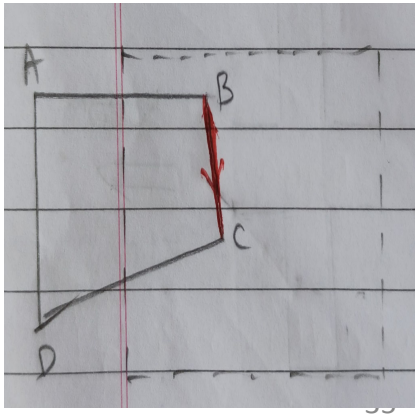
store P1 & B in outside vertex list {P1,B}

(Out → in)

Case 2-If both first & second vertices of a polygon are lying inside the window,then we have to store the second vertex only in output vertex.

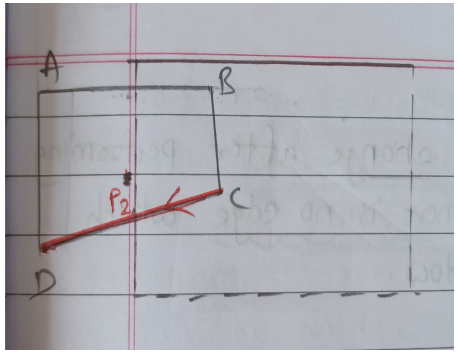
store only C in output vertex list {C}

(in → in)



Case 3- If first vertex is inside the window and second vertex is outside the window then

we have to store only intersection point of that edge of polygon with window in a output vertex list.



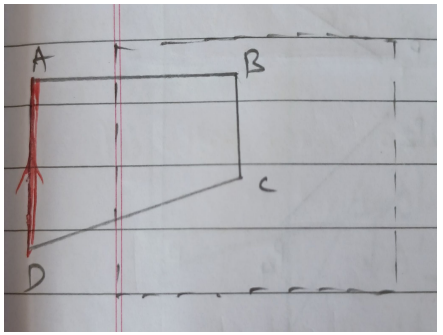
store P2 only in outside vertex list {P2}

(in → out)

Case 4-If both first & second vertices of a polygon are lying outside the window,then no vertex is stored in output vertex list.

Nothing is store in output vertex list { }

(out→out)

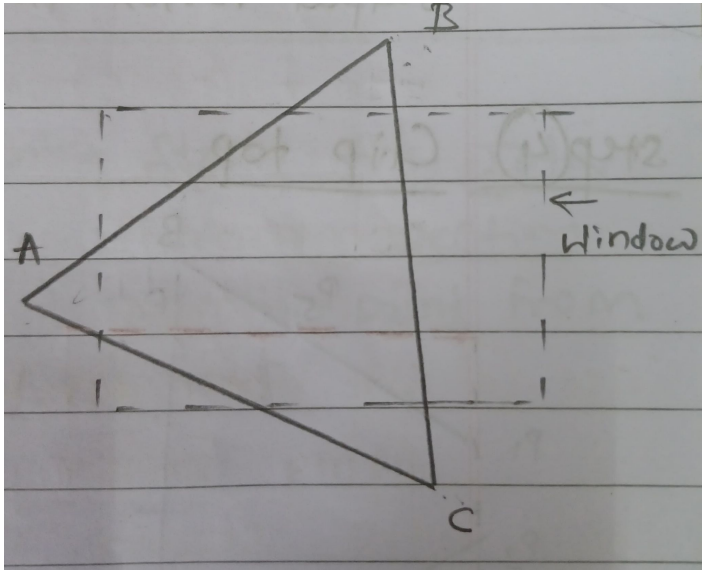


Example.

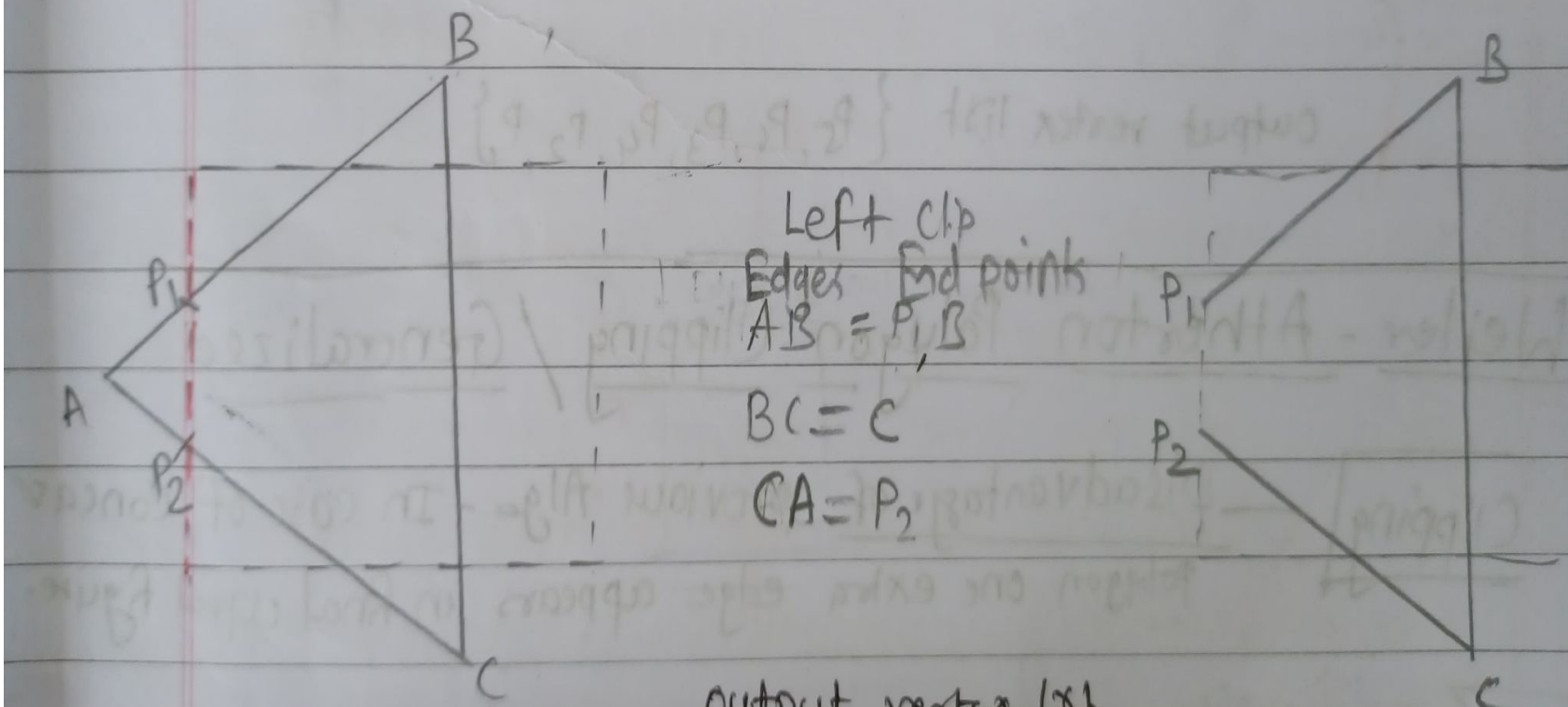
Suppose we have polygon ABC which we want to clip against a rectangular window.

Vertex list ={A,B,C}

Edges will be AB,BC,CA



Step ① : Clip left



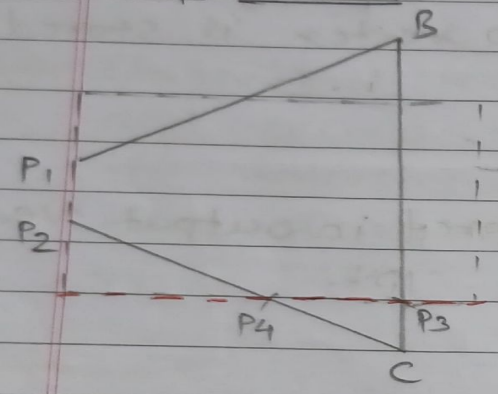
output vertex list

{P1, B, C, P2}

Step (2) Clip Right

The o/p vertex list will not change after performing clip right procedure because there is no edge which lies on right side of window.

Step (3) Clip bottom



Bottom clip

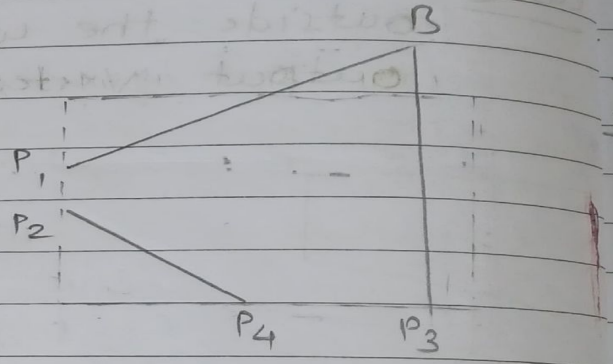
End point

$P_1 B = B$

$BC = P_3$

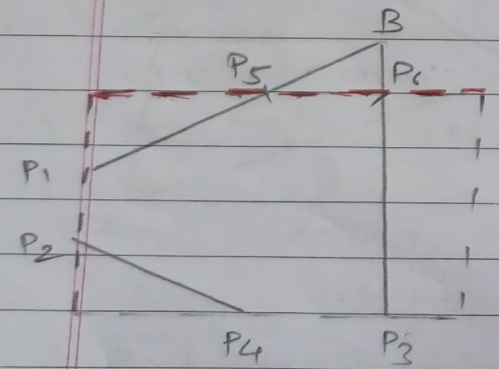
$CP_2 = P_4, P_2$

$P_2 P_1 = P_1$



output vertex list = $\{B, P_3, P_4, P_2, P_1\}$

Step (4) Clip top



Top clipping

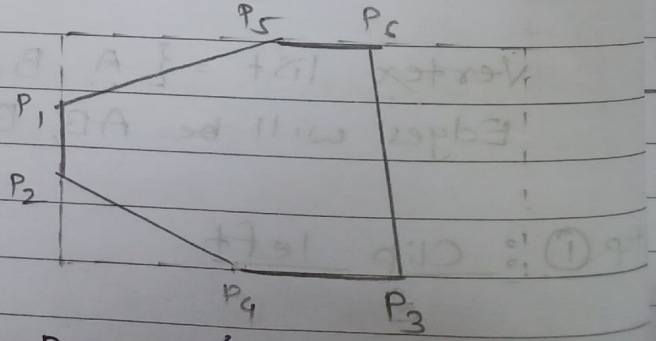
$P_1 B = P_5$

$B P_3 = P_6, P_3$

$P_3 P_4 = P_4$

$P_4 P_2 = P_2$

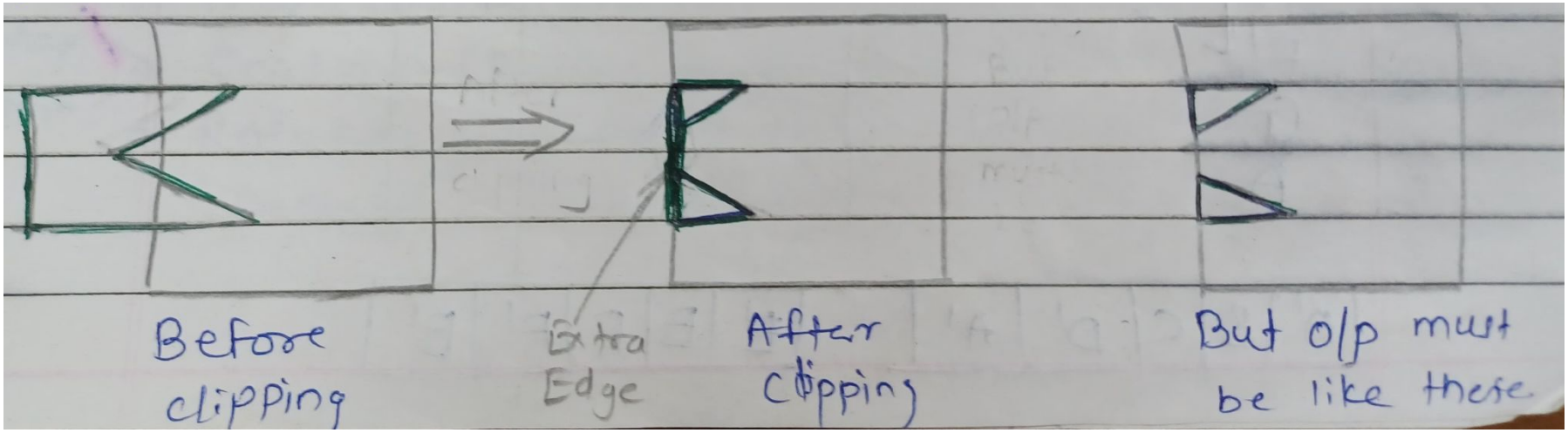
$P_2 P_1 = P_1$



output vertex list $\{P_5, P_6, P_3, P_4, P_2, P_1\}$

2) Weiler Atherton Polygon Clipping / Generalized Clipping-

All convex Polygon get correctly clipped by the sutherland Hodgeman algorithm but some concave polygon may be displayed with extra edges.



- Give clockwise direction to polygon.
- Make 2 list

  1) **Subject polygon list**-All vertices including intersection points.

  2) **Clip polygon list**- Clipping window with intersection point.

- **Algorithm**-

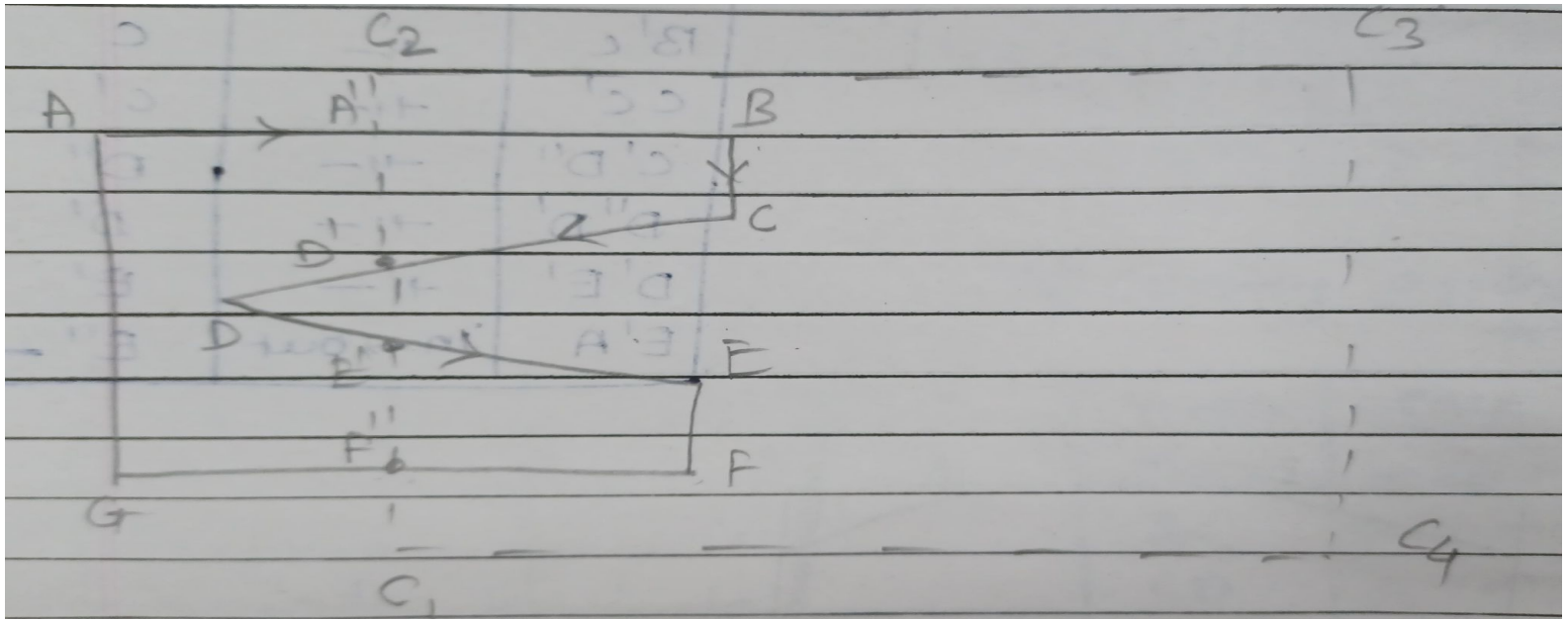
  1) If the edge enter the clipping window, record the intersection points & continue to trace the subject polygon list.

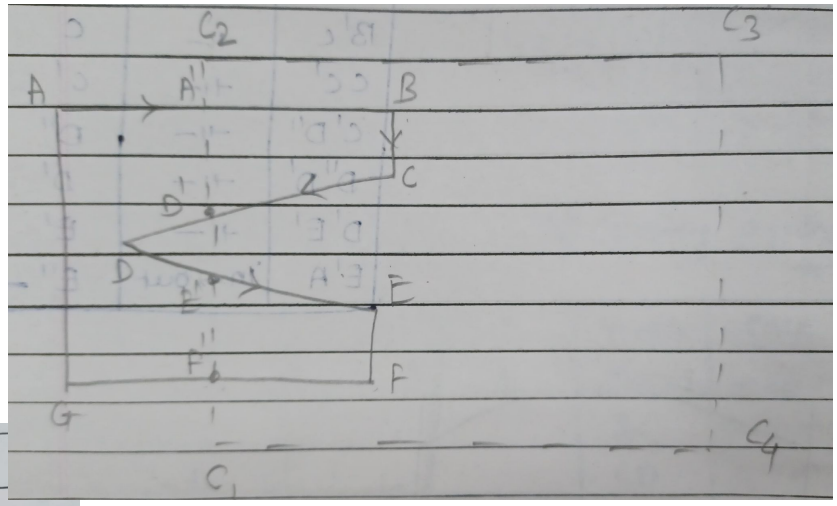
  2) If the edge leaves the clipping window,record the intersection point and make a right follow the clip polygon list in same manner. i.e.treat clip polygon as a subject polygon.

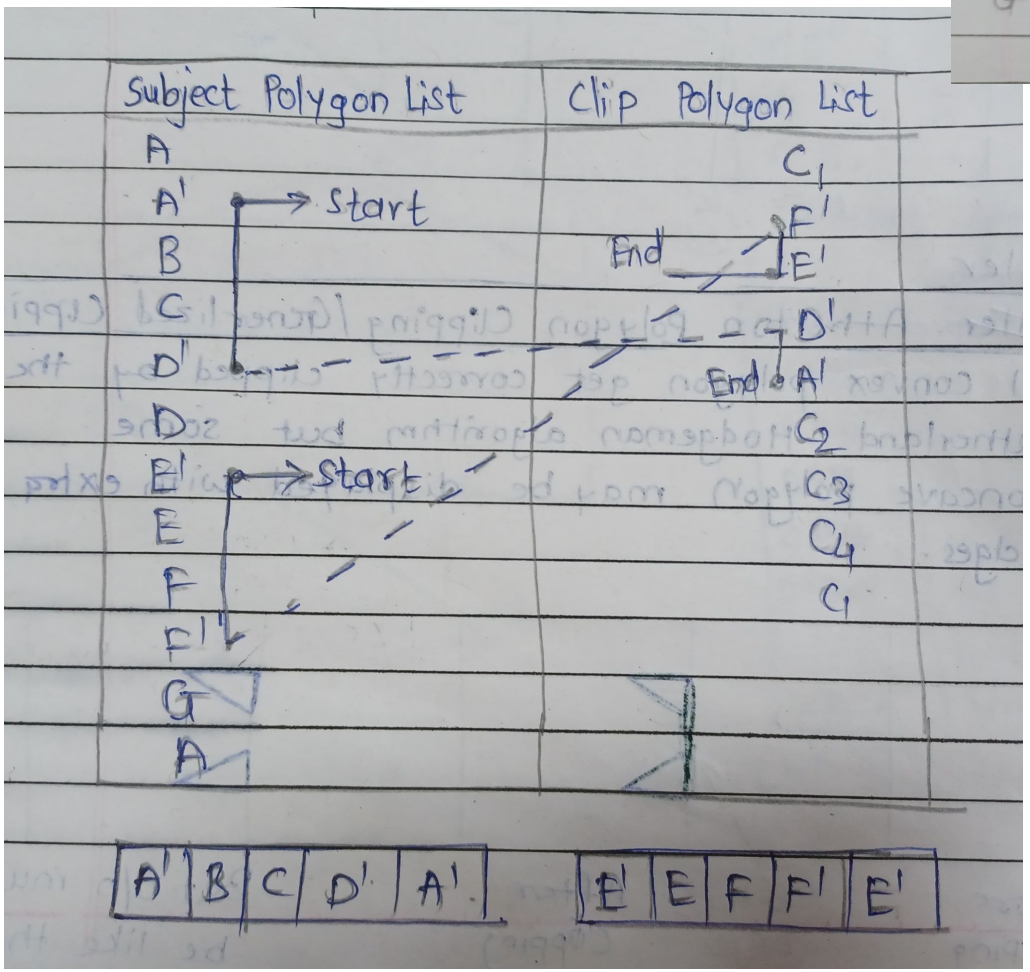
  3) Continue until vertex reach visited vertex.

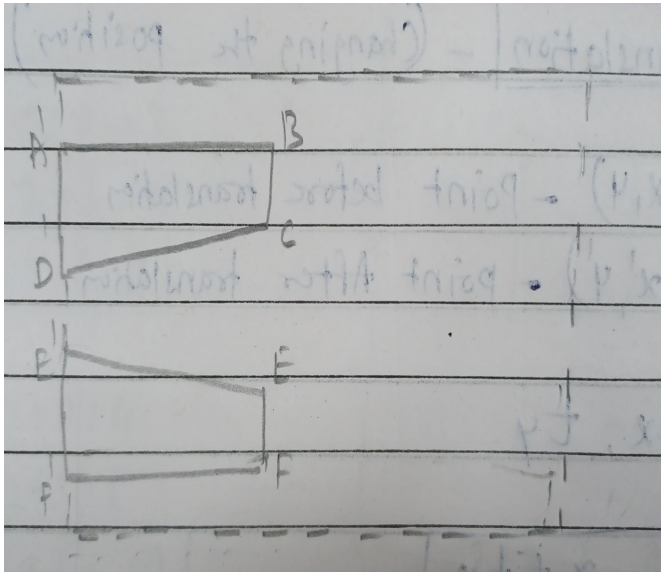
Example

Solve given polygon by using Weiler Atherton Polygon clipping algorithm




C2
C3
A
A'
B
C
D'
D
E'
E
F'
F
G
C4
C1


Subject Polygon List
Clip Polygon List
A
A'
Start
B
C
D'
D
E'
Start
E
F
F'
G
A
C1
F'
End
E'
D'
End
A'
C2
C3
C4
C1
A' B C D' A'
E' E F F' E'


A'
B
C
D
E'
E
F
F'