Assignment No II

Polygon and Clipping Algorithm

Q1. Explain concave & convex polygons with example.

Q2. Explain Boundary_fill ad Seed fill algorithm for polygon.

Q3. what are the steps involved in filling polygon in scan_line method.?

Q 4. What is windowing & clipping? What is interior & exterior clipping?

Q5. Explain Cohen-Sutherland algo with example.

Q 6. Explain why Sutherland- hodgman polygon clipping algo works only for convex clipping regions.

Q7. Use cohen-Sutherland outcode algo to clop two lines p1(40,15)- p2(75,45) & p3(70,20)-p4(100,10) against a window A(50,10),B(80,10),C(80,40),D(50,40).

Q.8 Explain with filling with pattern

Unit - II

Prof. A. D. Sawant.

Polygons and Clipping Algorithms

Syllabus :- Introduction, types : convex, concave and complex. Representation of polygon, Inside test, polygon filling Algo. flood fill, seed fill and scan line fill and filling with pattern. Windows and Clipping :- Viewing transformations, 2D clipping cohen - Sutherland Algo, Polygon clipping - Sutherland - Hodgeman

Introduction :-

A polygon may be represented as a number of line segments connected, end to end to form a closed figure.

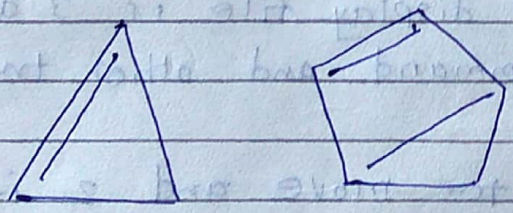
The end points of the sides called edges vertices and line segments called edges or sides. The simplest polygon is called the triangle having three sides and three vertex points.

Polygon is divided into three classes.

1) Convex 2) Concave 3) complex.

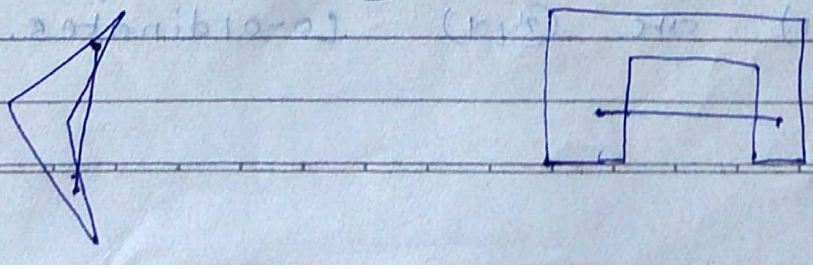
1) Convex Polygon :- When line joining the two any points inside the polygon is completely inside the polygon is called convex polygon

eg.



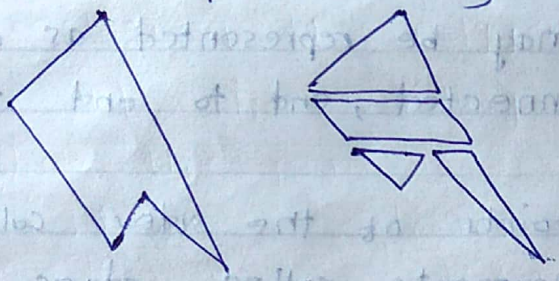
2) Concave polygon :- When line joining the any two points inside the polygon is not completely inside the polygon then it is called concave polygon.

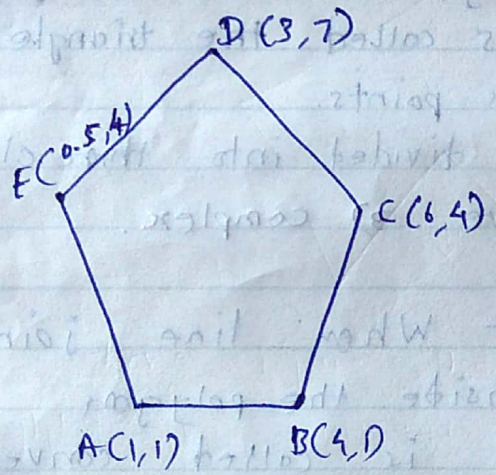
eg.



complex:

Polygon Representation :- Trapezoid primitive means polygons are imaged or drawn in the form of trapezoids. Trapezoids are formed by using two scan lines and two line segments. Every polygon can be broken up into trapezoids and any polygon can be represented by the series of trapezoids.





op	X	Y
5	1	1
2	4	1
2	6	4
2	3	7
2	0.5	4
2	1	1

The structure of display file is 3 arrays, one for opcode i.e. command and other two stores x and y values.

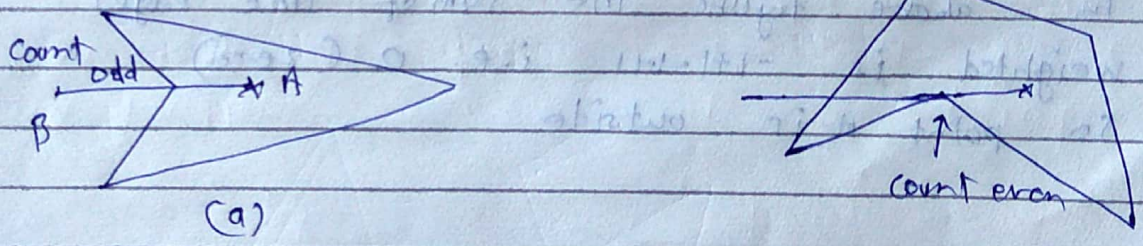
Command 1 is for move and 2 is for line The first command 5 is used to tell the how many edges are there for polygon and the starting point co-ordinates for the first line. 5 is no. of eqdge and (1, 1) are (x, y) co-ordinates.

Inside And Outside Test of Polygon :-

a) Even-Odd Method (odd-parity rule)
b) Winding number Method

a) Even-Odd Method

Construct a line segment bet^n the point in question and a point known to be outside the polygon If the no. of polygon edge crossed by this line is odd, then this point is an interior point otherwise this point is an exterior point But when line crosses at vertex point, then when the other two end point of polygon line segment crossing the line at one vertex are opposite side of line then count is considered as odd and when the other ~~poin~~ end points are at one side of line then count is considered as even,



(a) (b)

fig a: count is 1 i.e. odd so point A is inside.
fig b: count is 1+2 = 3 i.e. odd so point is inside

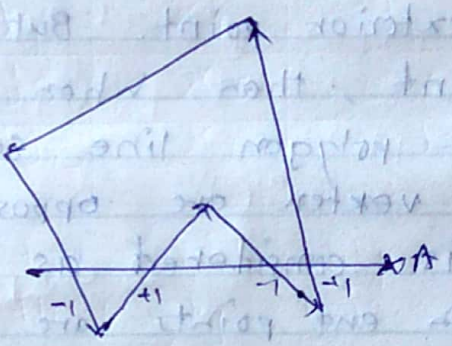
b) Winding Number Method :-

Stretch the a piece of elastic between the point in question and a point on the polygon boundary and slide the elastic string along the boundary of the polygon. Until it has made one complete circuit

If it wound at least once then the point is inside. If no net winding then point is outside.

In this method instead of just counting the intersections, we give each boundary line crossed a direction number. We sum these direction numbers. The direction number indicates direction of the polygon edge.

If the winding number is non-zero, then the point is in interior region, otherwise point is in exterior region. i.e outside the polygon.

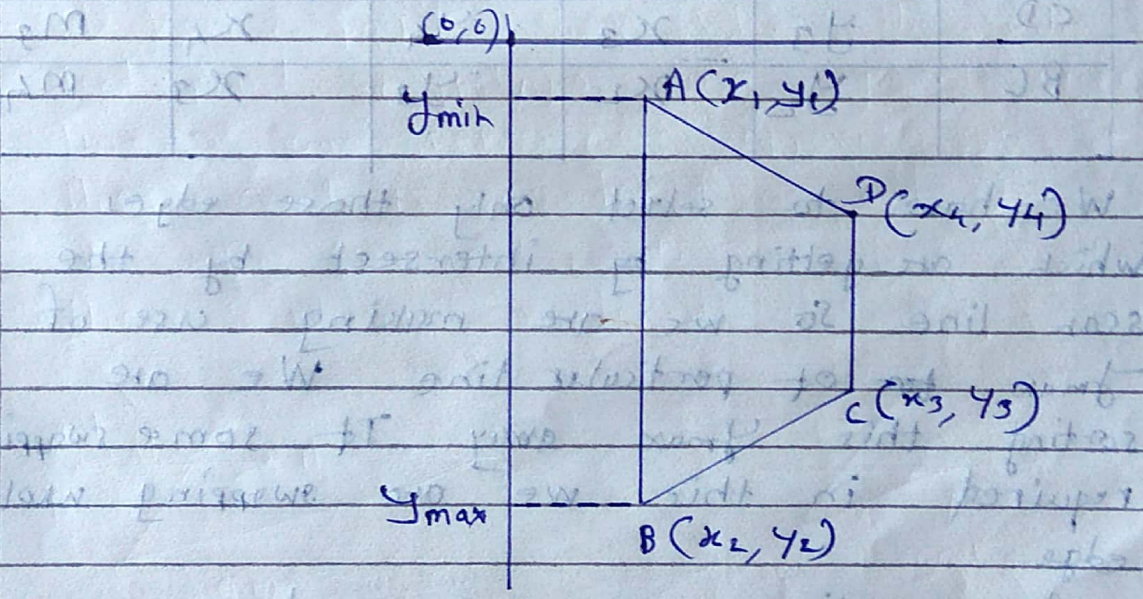


In above figure the sum of line edges weighted is $-1+1-1+1$ i.e. 0 (zero) so point A is outside.

Scan Line Algorithm :-

This algorithm is defined at geometric level i.e. co-ordinates vertices etc. This algorithm start at first scan line and ~~end~~ proceede line by line toward the last scan line. It checks whether every pixel ~~of~~ on that scan line satisfies our inside test or not. It checks for which points ~~of~~ on that scan line are inside or not. This method avoids need for seed point.

Let us take example of convex polygon



Here algorithm start with the first scan line y_{max} and end at y_{min}.

Here we are storing X_{max}, X_{min} y_{max} and y_{min} of the edges of the polygon along with their slopes.

X_{max} and X_{min} are corresponding x component of Y_{max} and Y_{min} along with this we are storing slope of the edges.

For line AB

$x_{max} = x_2$

$x_{min} = x_1$

$y_{max} = y_2$

$y_{min} = y_1$

Let us tabularize this.

Edge	y_{max}	x_{max}	y_{min}	x_{min}	slope
AB	y_2	x_2	y_1	x_1	m_1
AD	y_4	x_4	y_1	x_1	m_2
CD	y_3	x_3	y_4	x_4	m_3
BC	y_2	x_2	y_3	x_3	m_4

We have to select only those edges which are getting intersect by the scan line. So we are making use of y_{max} of perticular line. We are sorting this y_{max} array. If some swapping required in this we are swapping whole edge.

After sorting y_{max} we get,

edge	y_{max}	x_{max}	y_{min}	x_{min}	slope
AB	y_2	x_2	y_1	x_1	m_1
BC	y_2	x_2	y_3	x_3	m_4
CD	y_3	x_3	y_4	x_4	m_3
AD	y_4	x_4	y_1	x_1	m_2

Now y_{max} array is sorted. We can find out the intersection of scan line with first two edges from sorted attribute table. i.e AB and BC every time we are decreasing scan line by 1 from y_{max} to y_{min} of the polygon. This may happen that the edge we have selected to find intersection may get finished i.e. scan line goes bellow the y_{min} of selected edge. In that case we have to discard that edge and select next edge from the sorted table and continue till scan line becomes equal to y_{min} of polygon.

Here important is how to find out the intersection of point co-ordinates of scan line with particular edge. We can use slope of the edge.

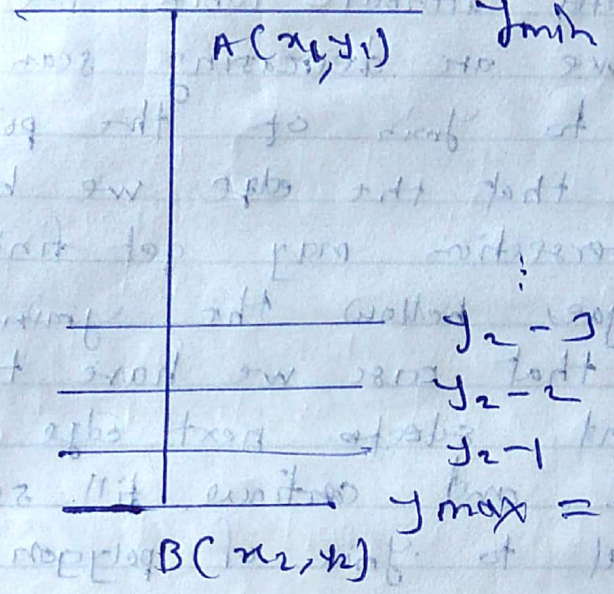
When we are moving from y_{max} to y_{min} everytime we are decreasing y by 1, the distance beth two scan line is 1 so we know new y value. Which will be $y_{max} - 1$ We will calculate corresponding x coordinate. We know that:

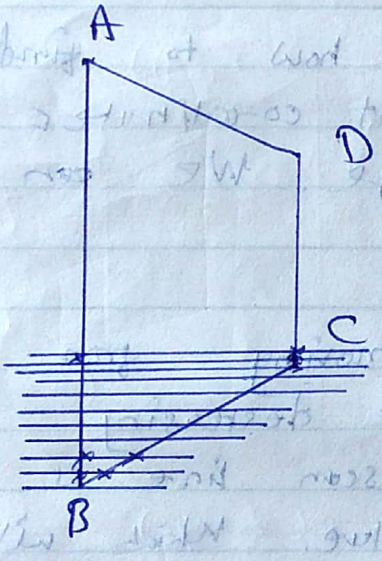
$$\text{Slope } (m) = \frac{\Delta y}{\Delta x} = \frac{\text{change in } y}{\text{change in } x}$$

$$= \frac{y_{new} - y_{old}}{x_{new} - x_{old}}$$

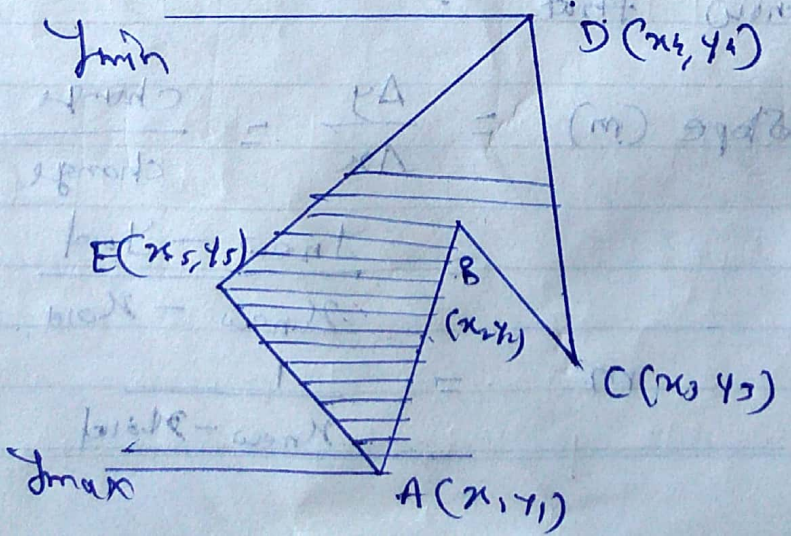
$$m = \frac{1}{x_{new} - x_{old}}$$

$$x_{new} = x_{old} + \frac{1}{m}$$



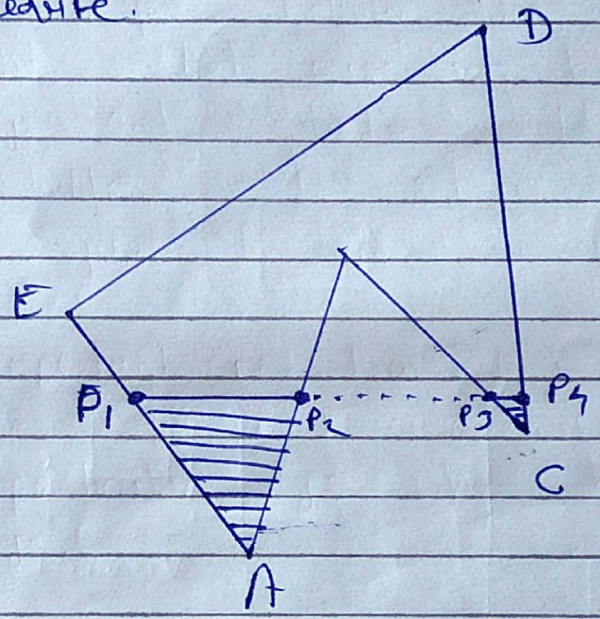


This will not work for concave polygon



To handle this we have to slightly modify this algorithm.

We should check for every scan line that the y coordinate of scan line is whether crossing the y_{max} of other edges of the polygon if yes then we have include the other two polygon edges in the procedure.

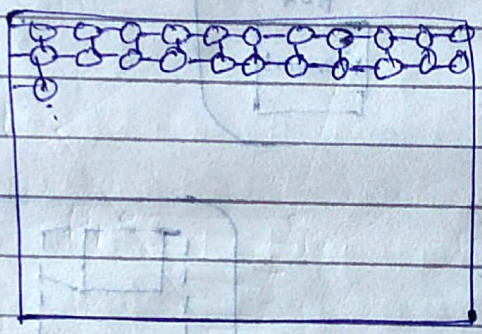


Steps for Scan Line Algorithm :-

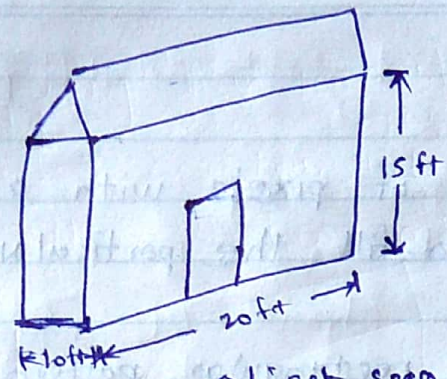
1. Read n the number of of vertices of polygon
2. Read x and y co-ordinates for all vertices in an array x[n] and y[n].

* Filling with Patterns :-

- Pattern is a small group of pixels with a fixed color combination used to fill the particular area in the picture.
- Filling an area with a rectangular pattern is called tiling and rectangular patterns are sometimes referred to as tiling patterns.
- Rather than filling a polygon with a solid color, we may want to fill it with a pattern or texture.



We store the pattern in an M x M array (eg. 8 x 8) then.
Pattern will be in the column 0, 8, 16 etc of the polygon.
So, we will have a row pointer and a column pointer :
For each scan line (Y) : rowptr ← y mod 8
For each colum (X) : colptr ← x mod 8.
Then to display the pixel, we use ~~set-pixel~~
set-pixel (x, y, pattern [rowptr, colptr])
Relating the area of the pattern with the area of the primitive of a polygon is known as anchoring.



object space

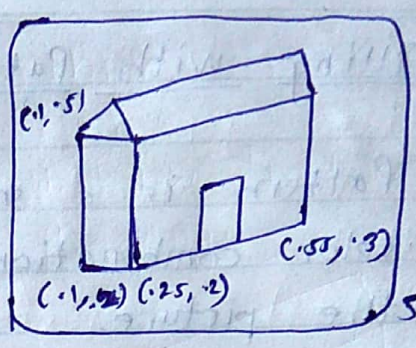
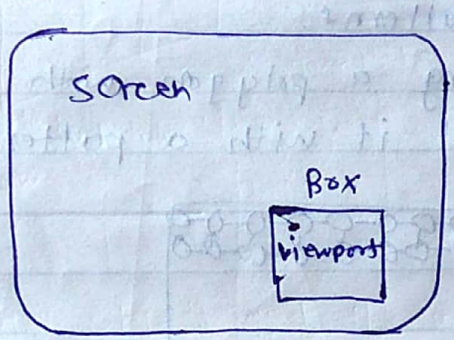
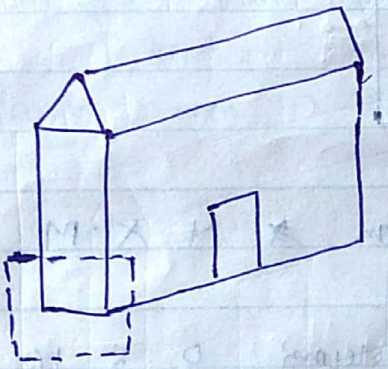
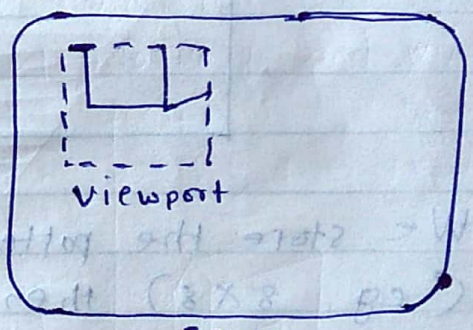
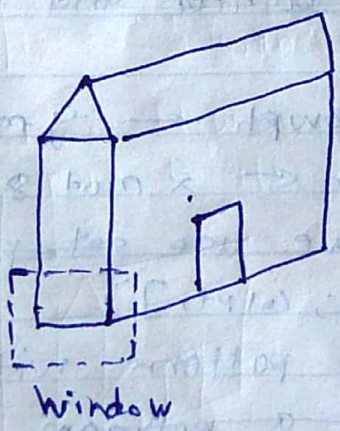


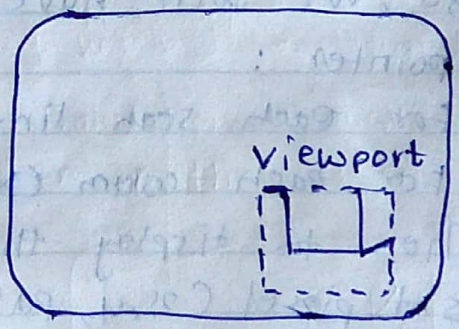
Image space











Windowing :- The Method of selecting and enlarging the portions of a drawing is called windowing.

Clipping :- Determining which portion to omit or suppress is called clipping.

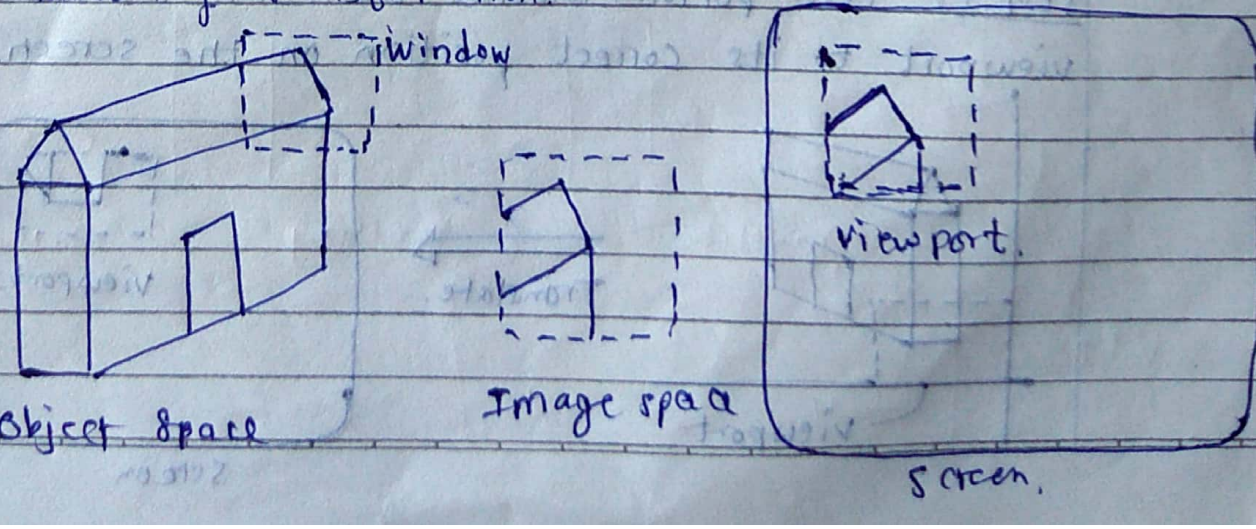
Viewing Transformation :-

World Co-ordinate System :- The object space contains the dimensions as actual which is called as world co-ordinate system.

Window :- A world co-ordinate area selected for display is called window.

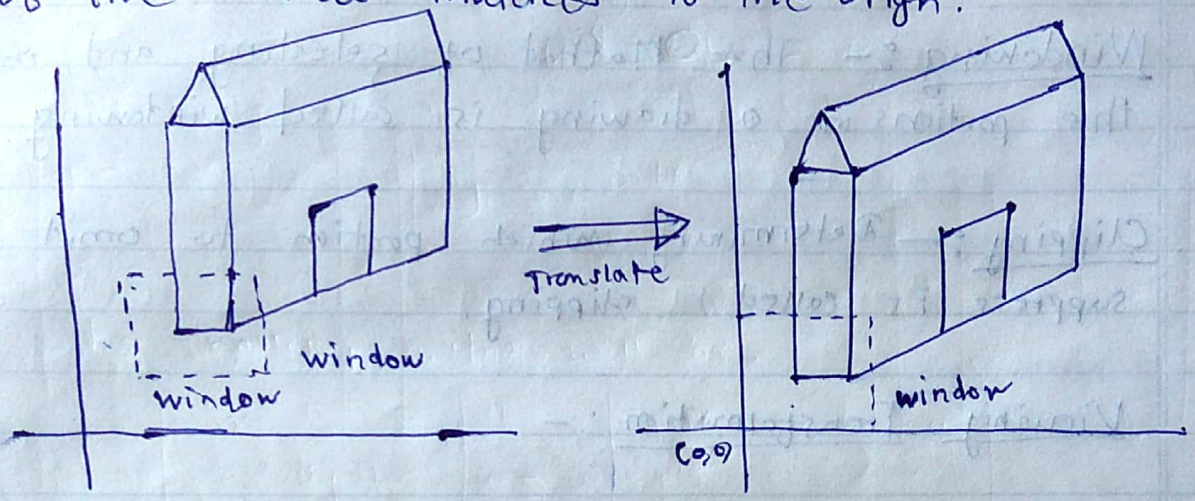
Viewport :- An area on a display device to which a window is mapped is called viewport.

The mapping of a part of world co-ordinate scene to ~~display~~ device co-ordinates is called as viewing transformation or window-viewport transformation or windowing transformation.

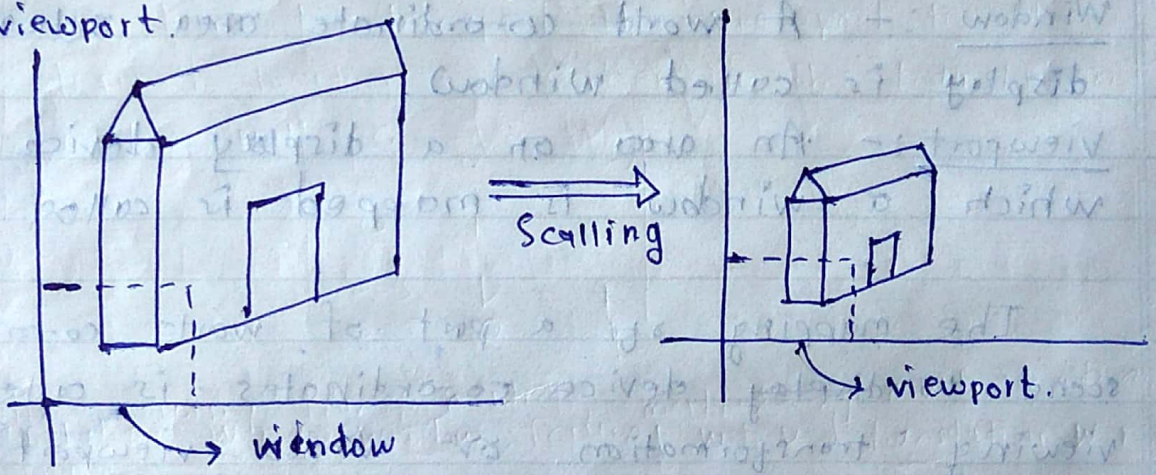


Viewing transformation is a three step process.

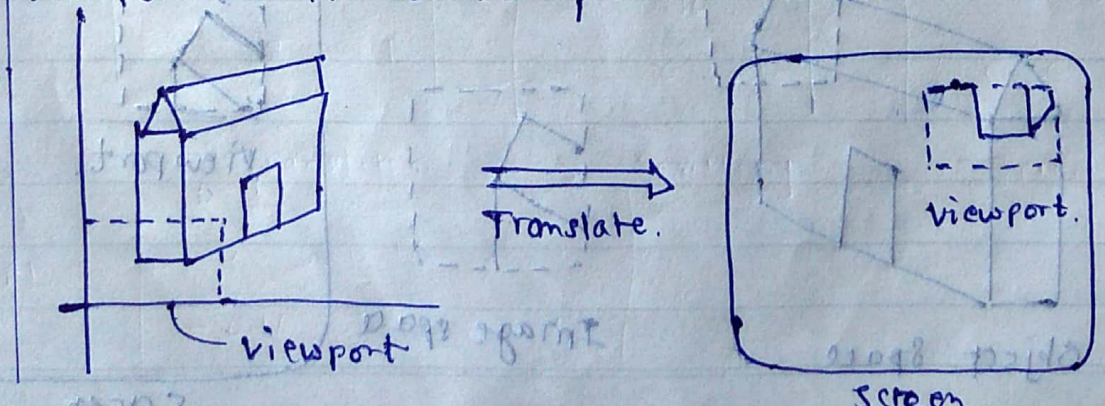
Step 1:- The object with its window is shifted or translated until the lower left corner of the window matches to the origin.



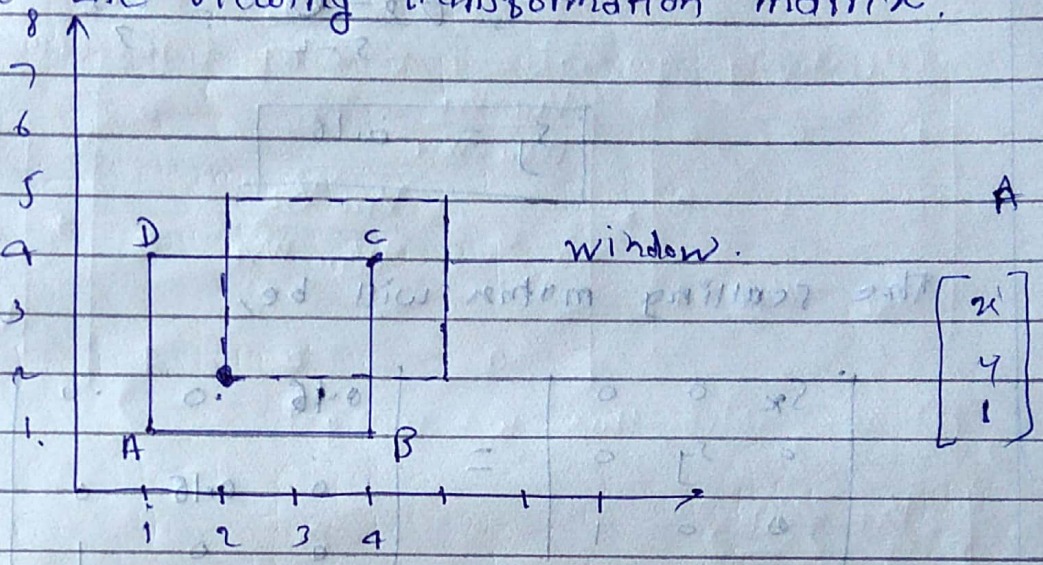
Step 2: The object and the window are now scalled until the window has the dimension same as viewport. In short we are converting object into image and window in viewport.



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



Q. Suppose there is a rectangle ABCD whose co-ordinates are A (1,1), B (4,1), C (4,4), D (1,4) And the window co-ordinates are (2,2), (5,2), (5,5), (2,5) And given viewport location is (0.5,0), (1,0), (1,0.5), (0.5,0.5), Derive the viewing transformation matrix.



For viewing transformation we have to perform three steps First is translation.

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x & -y & 1 \end{vmatrix}$$

Second step is scalling : Here we have to scale window to viewport i.e. old dimensions will be of windows dimensions and new will be of viewports.

For scalling in x direction S_x will be,

$$S_x = \frac{\text{new}}{\text{old}} = \frac{\text{width of viewport}}{\text{width of window}}$$

$$= \frac{1-0.5}{5-2} = \frac{0.5}{3}$$

$$\boxed{S_x = 0.16}$$

Similarly, $s_y = \frac{new}{old}$

$= \frac{\text{height of viewport}}{\text{height of window}}$

$= \frac{0.5 - 0}{5 - 2} = \frac{0.5}{3}$

$$s_y = 0.16$$

The scalling matrix will be,

$$\begin{vmatrix} S_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 0.16 & 0 & 0 \\ 0 & 0.16 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

We have to transport the viewport to the desired location on screen, so the translation matrix will be,

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.5 & 0 & 1 \end{vmatrix}$$

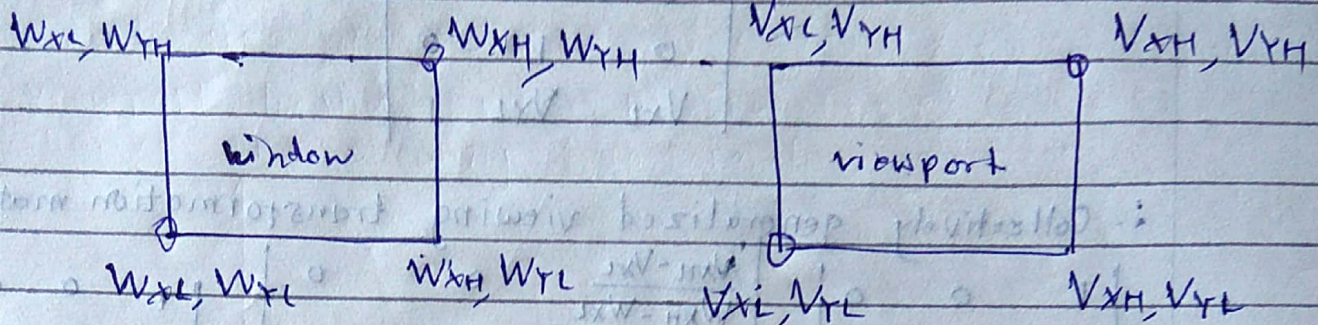
The viewing transformation matrix by multiplication of translation w.r.to. window, scalling w to v & translation w.r. to viewport.

$$= \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -2 & -2 & 1 \end{vmatrix} * \begin{vmatrix} 0.16 & 0 & 0 \\ 0 & 0.16 & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.5 & 0 & 1 \end{vmatrix}$$

Translation Scalling Translation

$$V.T.M. = \begin{vmatrix} 0.16 & 0 & 0 \\ 0 & 0.16 & 0 \\ -0.16 & -0.32 & 1 \end{vmatrix}$$

Here, consider W for window, V for viewport co-ordinate.



The matrix for translation with respect to window will be

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & \\ -W_{XL} & -W_{YL} & 1 \end{vmatrix}$$

The matrix for scalling will be

$$\begin{vmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Where, $S_x = \dfrac{\text{Width of viewport}}{\text{Width of Window}}$

$\therefore S_x = \dfrac{V_{XH} - V_{XL}}{W_{XH} - W_{XL}}$

Similarly, $S_y = \dfrac{\text{height of Viewport}}{\text{height of Window}}$

$\therefore S_y = \dfrac{V_{YH} - V_{YL}}{W_{YH} - W_{YL}}$

Scalling matrix will be,

$$\begin{vmatrix} \dfrac{V_{XH} - V_{XL}}{W_{XH} - W_{XL}} & 0 & 0 \\ 0 & \dfrac{V_{YH} - V_{YL}}{W_{YH} - W_{YL}} & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

like, that translation matrix with respect to viewport will be.

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ V_{xL} & V_{yL} & 1 \end{vmatrix}$$

∴ Collectively generalized viewing transformation matrix will be.

$$= \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -W_{xL} & -W_{yL} & 1 \end{vmatrix} * \begin{vmatrix} \frac{V_{xH}-V_{xL}}{W_{xH}-W_{xL}} & 0 & 0 \\ 0 & \frac{V_{yH}-V_{yL}}{W_{yH}-W_{yL}} & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ V_{xL} & V_{yL} & 1 \end{vmatrix}$$

If we solve this we will get.

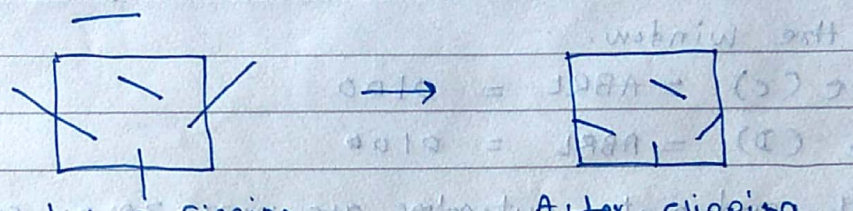
$$\begin{vmatrix} \frac{V_{xH}-V_{xL}}{W_{xH}-W_{yL}} & 0 & 0 \\ 0 & \frac{V_{yH}-V_{yL}}{W_{yH}-W_{yL}} & 0 \\ (V_{xL}-W_{xL})\cdot\frac{V_{xH}-V_{xL}}{W_{xH}-W_{xL}} & (V_{yL}-W_{yL})\cdot\frac{V_{yH}-V_{yL}}{W_{yH}-W_{yL}} & 1 \end{vmatrix}$$

If the height of and width of window is not in proportion with viewport then distortion occurs and the image may get elongated or minimized.

Clipping: Omitting the part of picture which lies outside the window and display which is inside the window.

2D-Clipping :-

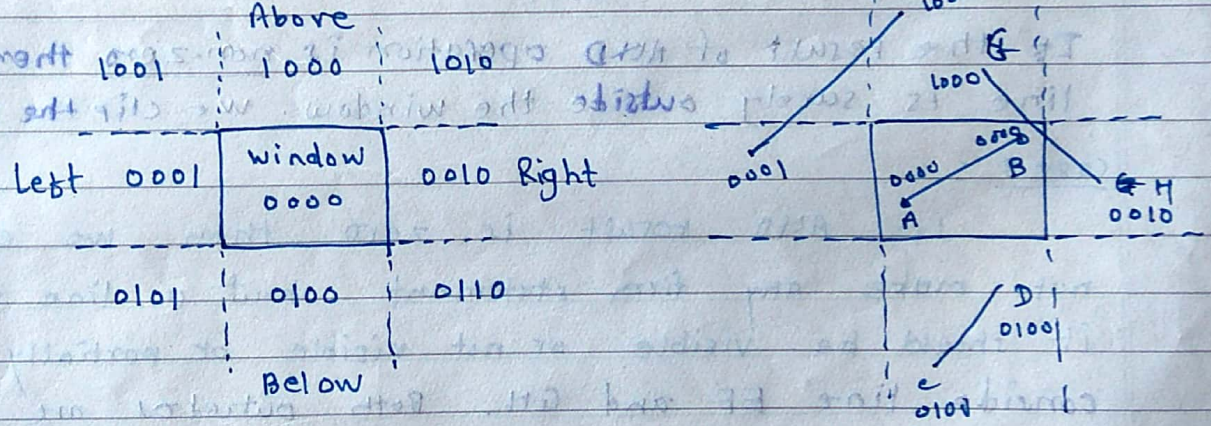
Line Clipping :-



before Clipping window

After clipping window.

* Cohen-Sutherland Algorithm :-



End point of each line is assigned a four bit binary code which is called as outcode.

"ABRL"

where A - Above, B - below, R - Right, L - Left.

Case 1 : Conside a line AB Here both end points of AB are surely inside the window. ∴ The outcode for end point A will be.

Outcode (A) = ABRL = 0000

outcode (B) = ABRL = 0000

Both outcodes are 0000. so line is fully visible.

Case 2 : Conside line CD Here end points of CD are surely outside the window.

outcode (C) = ABRL = 0100

outcode (D) = ABRL = 0100

Here both endpoints outcodes are non-zero, so we have to perform logical AND operation of both outcodes.

outcode (C) = 0100

outcode (D) = 0100 AND

0100

If the result of AND operation is non-zero then line is surely outside the window. we clip the complete line.

Case 3 :

If AND result is zero then we can not make any firm statement about a line whether it should be visible or not visible or partially visible. conside line EF and GH. Both outcodes are non-zero so we have to perform logical AND.

outcode (E) = ABRL = 0001

outcode (F) = ABRL = 1000 AND

= 0000

The result is zero (0). It means line EF is not lying completely on any one side of window.

Similar of line GH

outcode (G) = ABRL = 0010

outcode (H) = ABRL = 1000

Logical AND = 0000

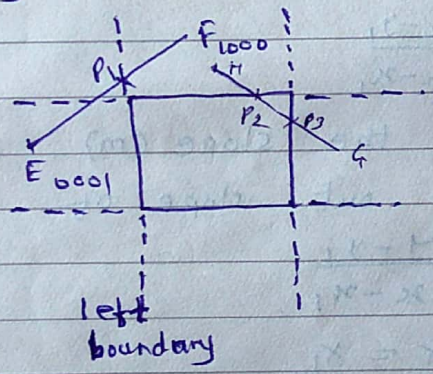
Both line GH & EF, AND result is zero. But line GH is partially visible and line EF is fully invisible, so for such line we have to find out which edge of the window line is intersecting and then find out the intersection point.

Here we have to compare first bit of both outcodes

E = 0001

F = 1000

The first bit of E is 1 & first bit of F is 0. means point E of the line EF is lying outside the left bound of window. so we findout the intersection of line EF with left boundary of window. call that intersection as P1



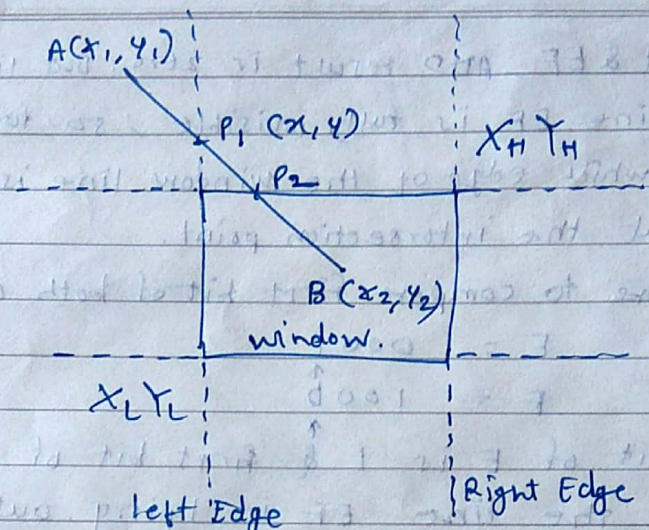
As point E is surely outside the left boundary of window. so we are clipping EP1. Now we will concentrate on P1F. Now outcode of P1 is 1000. Here both outcodes are nonzero so logical AND will be nonzero i.e. 1000. It means line P1F is lying completely on one side of window i.e. above the window. so discard the line

Consider line GH.

outcode G - 0010

H - 1000

Point H is Above the window and G is right side of window so we clip P2H and P3G. then outcodes of P2 & P3 are 0000 & 0000 so we display that line P2P3



Slope of line AB.

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

once we know the slope (m) then we find out slope of P_1A.

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

Here $x = x_L$

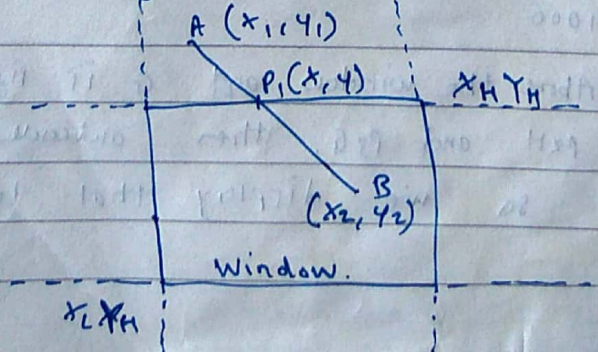
so $y = y_1 + m(x_L - x_1)$

so intersection point will be (x_L, y).

Similary intersection point ~~of~~ with right edge of window, Here x is x_R and

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

Similarly for top boundary the intersection point will lie on top edge of window y_H and x co-ordinate will be



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

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

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

Similarly for bottom edge y co-ordinate will be y_L and x co-ordinate will be.

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

Cohen - Sutherland line clipping Algorithm.

1. Accept the window co-ordinates from use and store in X_L, X_H, Y_L, Y_H
2. Accept the endpoints A and B of line with components A_x, A_y, B_x, B_y
3. Initialize A and B code as array of size four (4)
4. Calculate Acode and Bcode 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 Acode and Bcode are 0000. if yes display line AB. if not then proceed
6. Take logical AND of Acode and Bcode, check the result if it is nonzero, discard the line & go to step (8)
7. Subdivide the line by finding intersection of a line with window boundary (checking outcode from 1 to 4) and give new outcodes (recursively) to the intersection point and goto step (5) ~~recursively for array line segment.~~
8. else proceed.

if one end is boundary point and another end is outside that boundary

Page No.
Date

Seed Fill :-

The seed fill algorithm is further classified as i) boundary fill algorithm (edge-fill)
ii) ~~Seed~~ flood fill algorithm.

i) Boundary fill Algorithm

- In this method starting with some seed, any point inside the polygon we examine the neighbouring 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.
- Boundary defined regions may be either 4-connected or 8-connected as shown in figure.

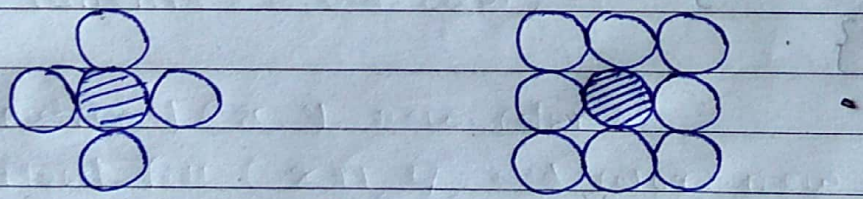


fig (a): Four connected region (b) Eight connected region.

In 4-connected method every pixel in the region may be reached by four moves in in directions : left, right, up and down.
In 8-connected method every pixel in the region may be reached by a combination of moves in two horizontal, two vertical, and four diagonal directions.

8-connected method is more accurate than 4-connected method. 4-connected method may produce the partial fill.

The following procedure illustrate the recursive method using 4-connected method.

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

Page No.
Date

Flood Fill Algorithm :

Sometimes if 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.

Here we start at some seed pixel and examine the neighbouring pixels. However, here pixels are checked for a specified interior color instead of boundary color and they are replaced by new color.

```
flood_fill (x, y, old_color, new_color)
{
    current = getpixel (x, y);
    if (current == old_color)
    {
        putpixel (x, y, new_color);
        flood_fill (x+1, y, old_color, new_color);
        flood_fill (x-1, y, old_color, new_color);
        flood_fill (x, y+1, old_color, new_color);
        flood_fill (x, y-1, old_color, new_color);
        flood_fill (x+1, y+1, old_color, new_color);
        flood_fill (x-1, y-1, old_color, new_color);
        flood_fill (x+1, y-1, old_color, new_color);
        flood_fill (x-1, y+1, old_color, new_color);
    }
}
```

Page No.
Date

Sutherland - Hodgeman polygon clipping.

A polygon can be clipped by processing its boundary as a whole against each window edge.

Here we first clip the polygon against the left rectangle boundary to produce new sequence of vertices.

The new set of vertices could then be successively passed to the right boundary clipper, a top boundary clipper, and a bottom boundary clipper.

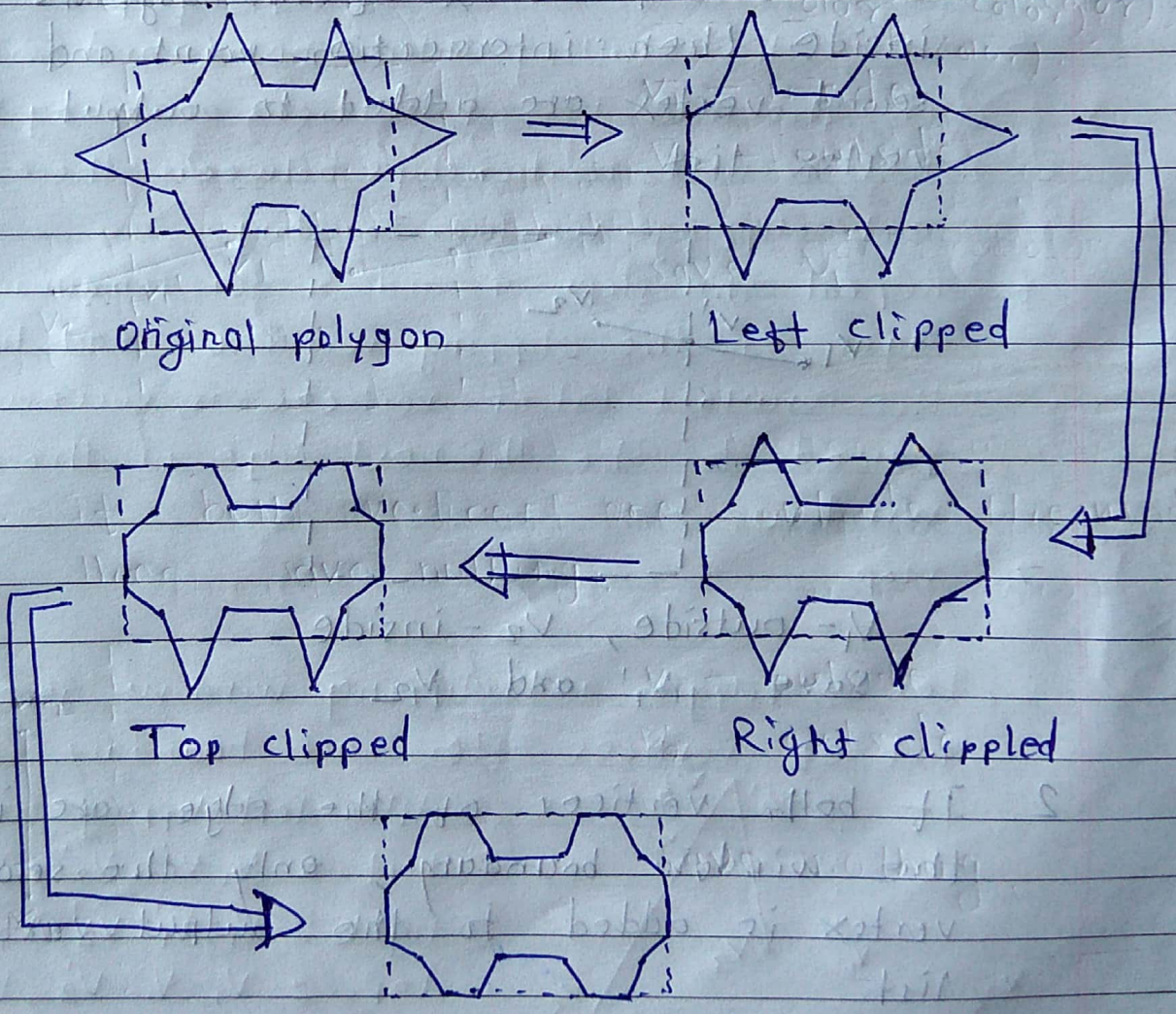
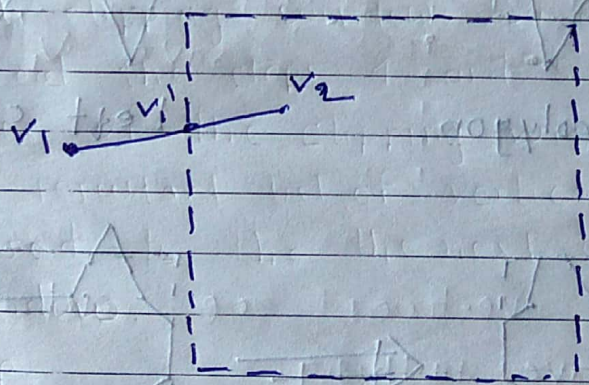


fig. clipping a polygon against successive window boundaries.

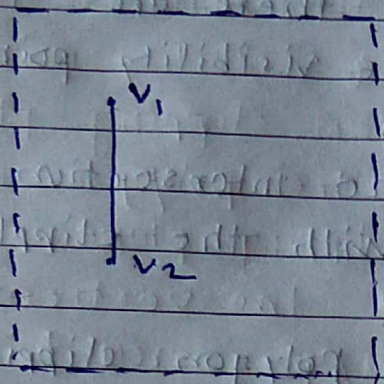
The output of the algorithm is a list of polygon vertices all of which are on the visible side of ~~polygon~~ clipping plane. Such each edge of the polygon is individually compared with the clipping plane. This is achieved by processing two vertices of each edge of the polygon around the the clipping boundary or plane. This results in four possible relationships.

1. If the first vertex of the edge is outside the clipping window boundary and second vertex of the edge is inside then intersection point and second vertex are added to output vertex list.



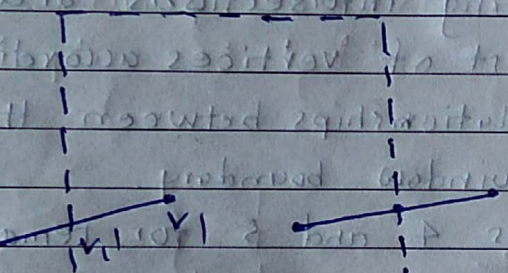
V_1 - outside, V_2 - inside
save V_1' and V_2

2. If both vertices of the edge are inside the window boundary, only the second vertex is added to the output vertex list.



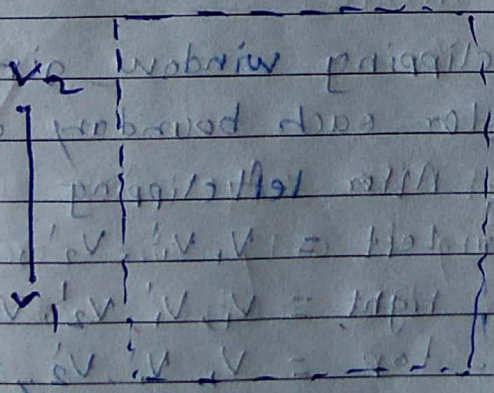
V_1 – inside
V_2 – inside
save V_2.

3. If first vertex is inside the window and second vertex is outside the window then only edge intersection with the window boundary point is added to the output vertex list.



V_1 – inside
V_2 – outside
save V_1'

4. If both vertices are outside the window then save nothing.



V_1 – outside
V_2 – outside
save nothing.