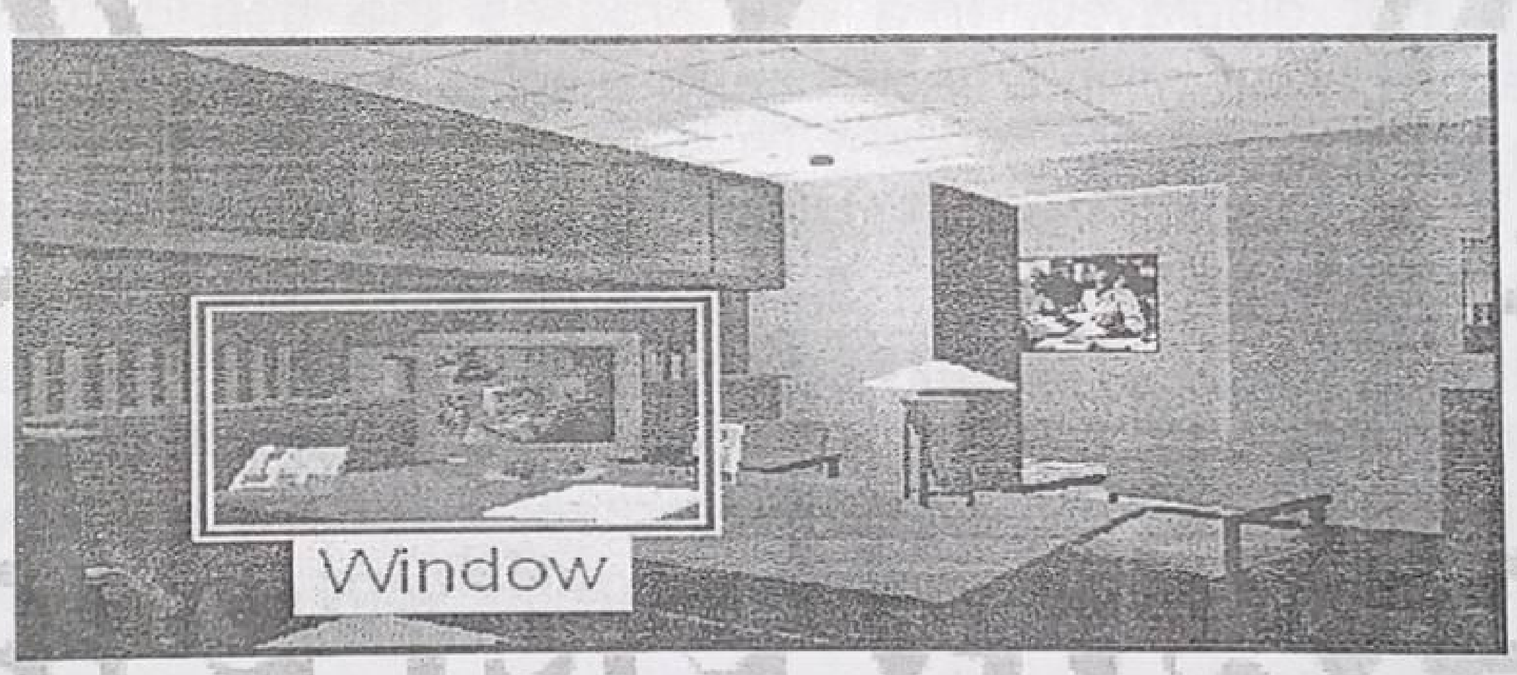
4. 2D VIEWING & CLIPPING

All objects in the real world have size. We use a unit of measure to describe both the size of an object as well as the location of the object in the real world. For example, meters can be used to specify both size and distance. When showing an image of an object on the screen, we use a screen coordinate system that defines the location of the object in the same relative position as in the real world. After we select the screen coordinate system, we change the picture to display interior screen that means change it into screen coordinate system.

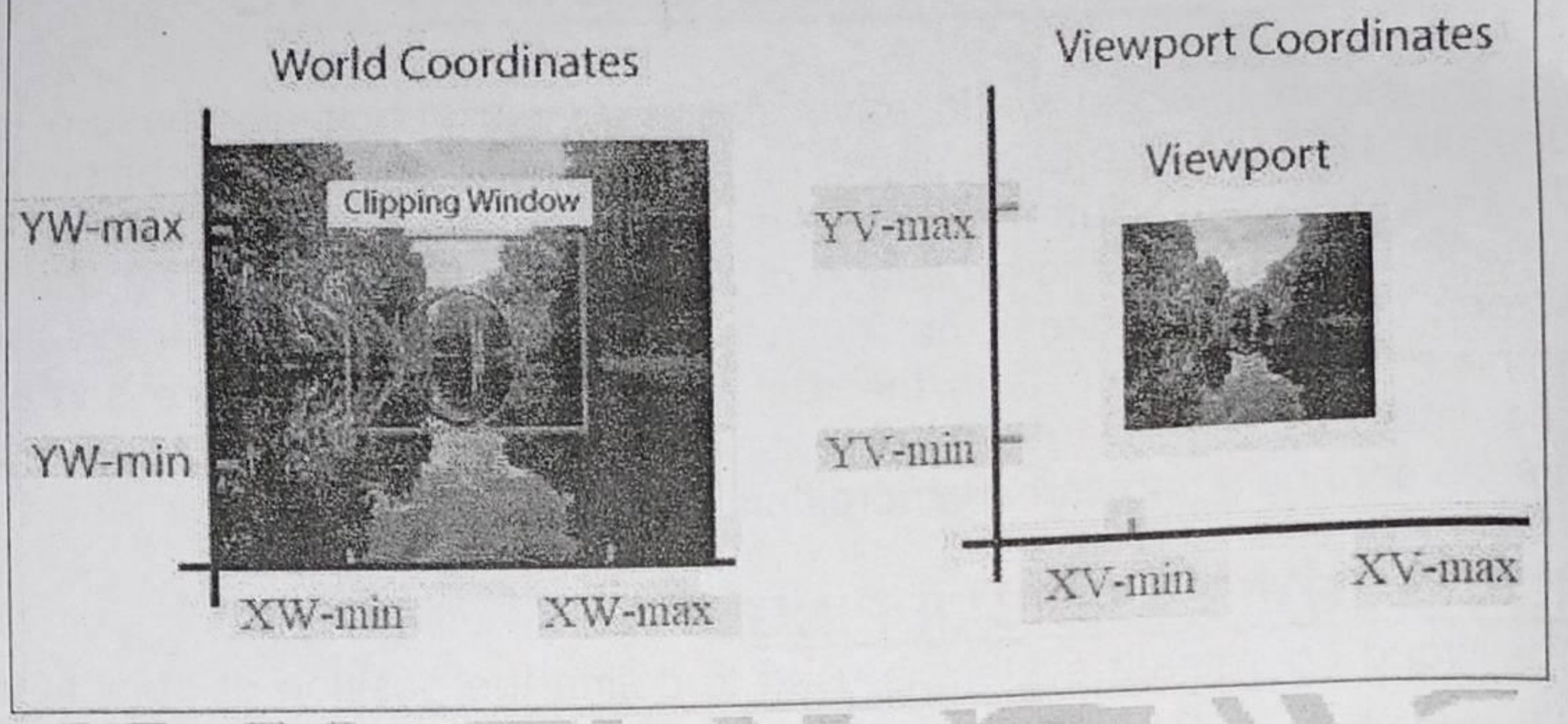
1. WINDOWS AND CLIPPING

The world coordinate system is used to define the position of objects in the real world. This system does not depend on the screen coordinate system , so the interval of number can be anything (positive, negative or decimal). Sometimes the complete picture of object in the world coordinate system is too large and complicate to clearly show on the screen, and we need to show only some part of the object.

A Window is a rectangular region in the world coordinate system. that defines **what is to be viewed**.



A Viewport is the section of the screen where the images encompassed by the window on the world coordinate system will be drawn or displayed. A coordinate transformation is required to display the image, from world coordinate to viewport coordinates (device coordinates). The viewport uses the screen coordinate system so effectively this transformation is from the world coordinate system to the screen coordinate system.

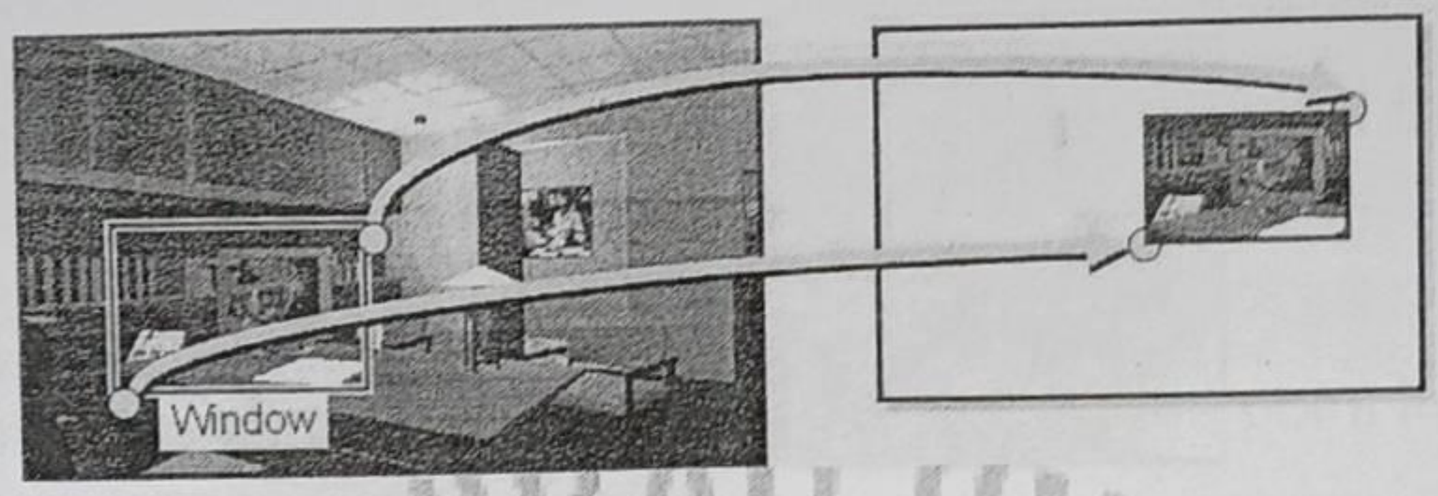


When a window is "placed" on the real world scenes, only certain objects and parts of objects can be seen. Points and lines which are outside the window are "cut off" from view. This process of "cutting off" parts of the image of the world is called Clipping. In clipping, we examine each line to determine whether or not it is completely inside the window, completely outside the window, or crosses a window boundary. If it is inside the window, the line is displayed. If it is outside the window, the lines and points are not displayed. If a line crosses the boundary, we must determine the point of intersection and display only the part which lies inside the window.

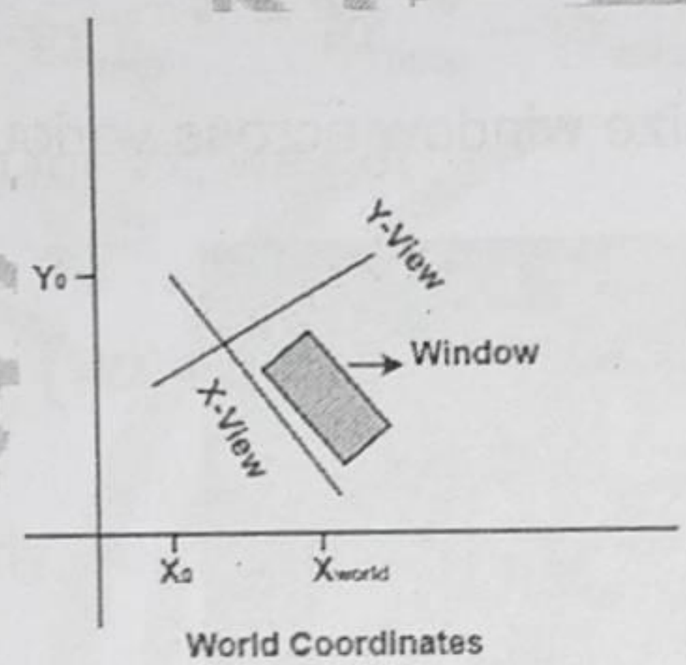
Window to Viewport Transformation is the transformation from world to device coordinates. It is actually a combination of many transformations. It basically involves **translation, scaling, rotation** transformations. It also requires methods to delete those portions of the scene which are outside the selected display area. This is called **clipping**.

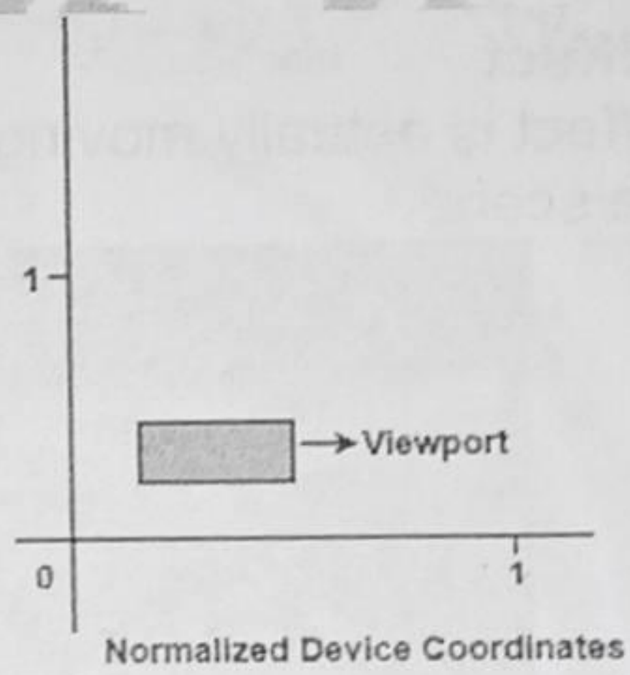
1.1 Window to Viewport Transformation

Now there are two coordinate systems. One is the World Coordinate system. While other is the Device Coordinate system. Therefore, The mapping of a world coordinate scene to device coordinate is **Window to Viewport Transformation or Windowing transformation.**



1.2 Normalized Coordinates





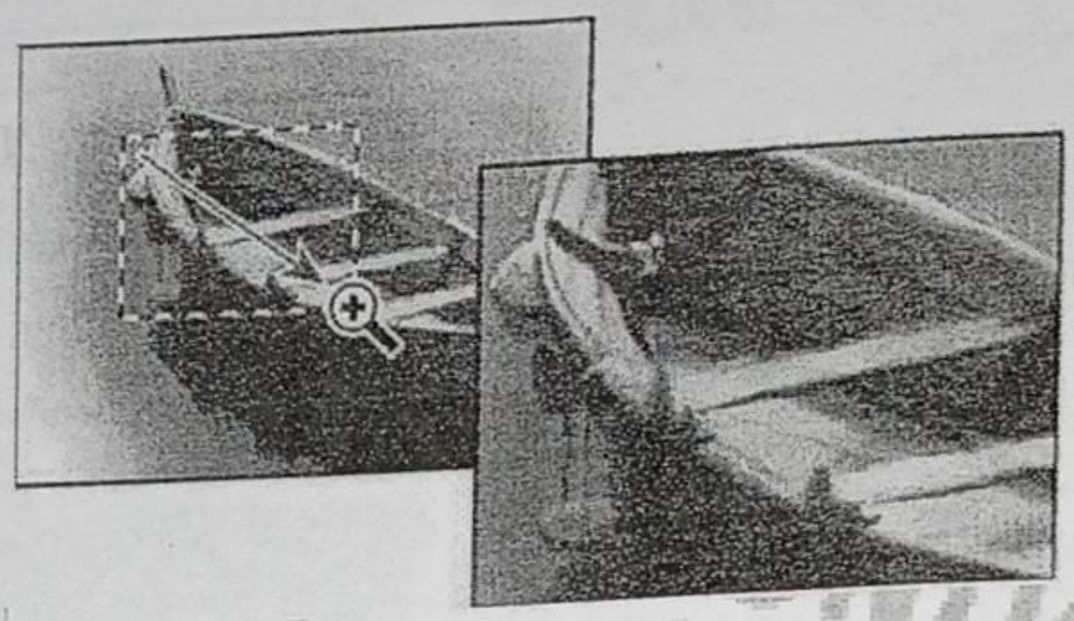
Viewports are defined within the unit square called normalized coordinates. Normalized coordinates are useful for separating viewing and other transformations from specific output device requirements so that the graphics package is largely device-independent.

Applications of Viewport

We can view objects at different positions on the display area of an output device by changing the positions of the viewport. Therefore, We can change the size and proportions of displayed objects by varying the size of the viewport. It is also possible to determine many viewports on different areas of display and view the same object at a different angle in each viewport.

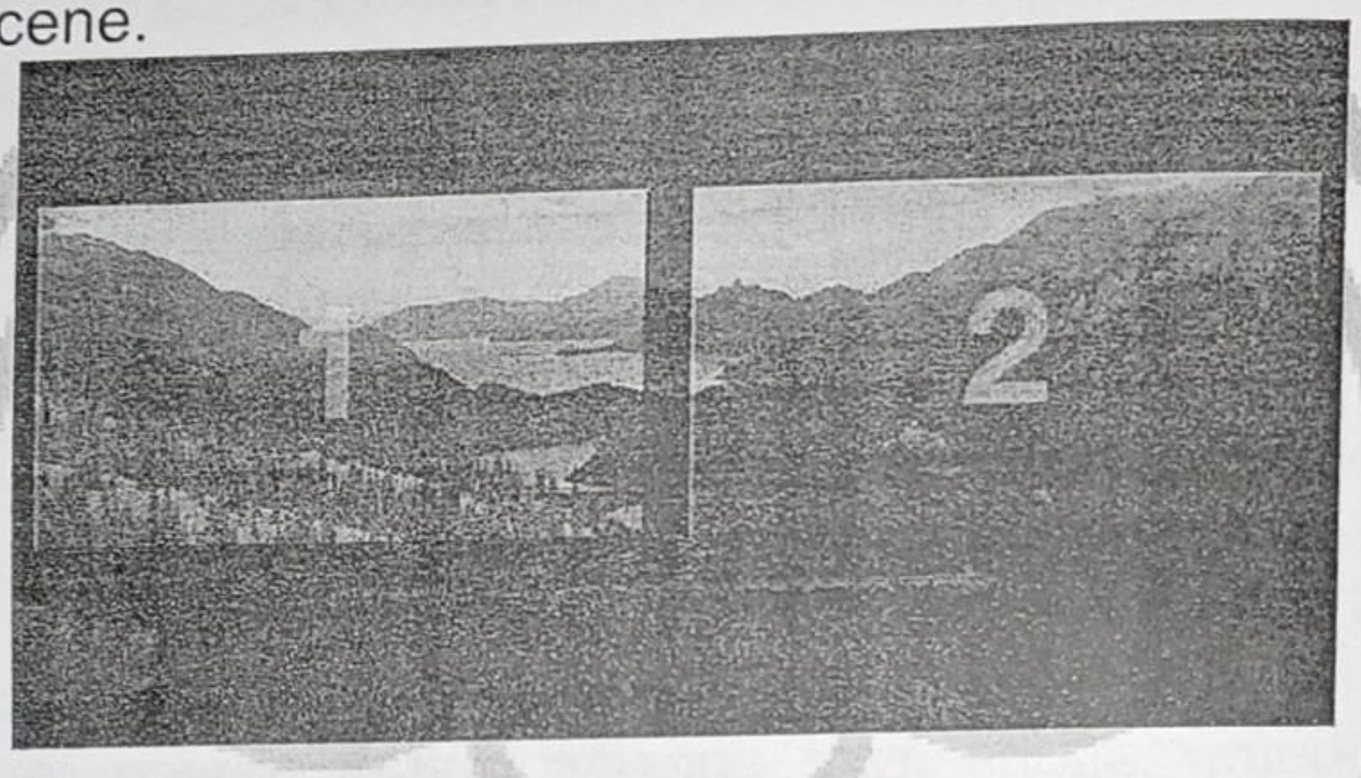
Zooming Effect

Zooming Effect is Successively mapping different sized windows on a fixed size viewport. As the windows are made smaller, so we zoom in on some part of a scene to view details that were not visible in large windows. Also, We can obtain more overview by zooming out from a section of a scene with successively larger windows.



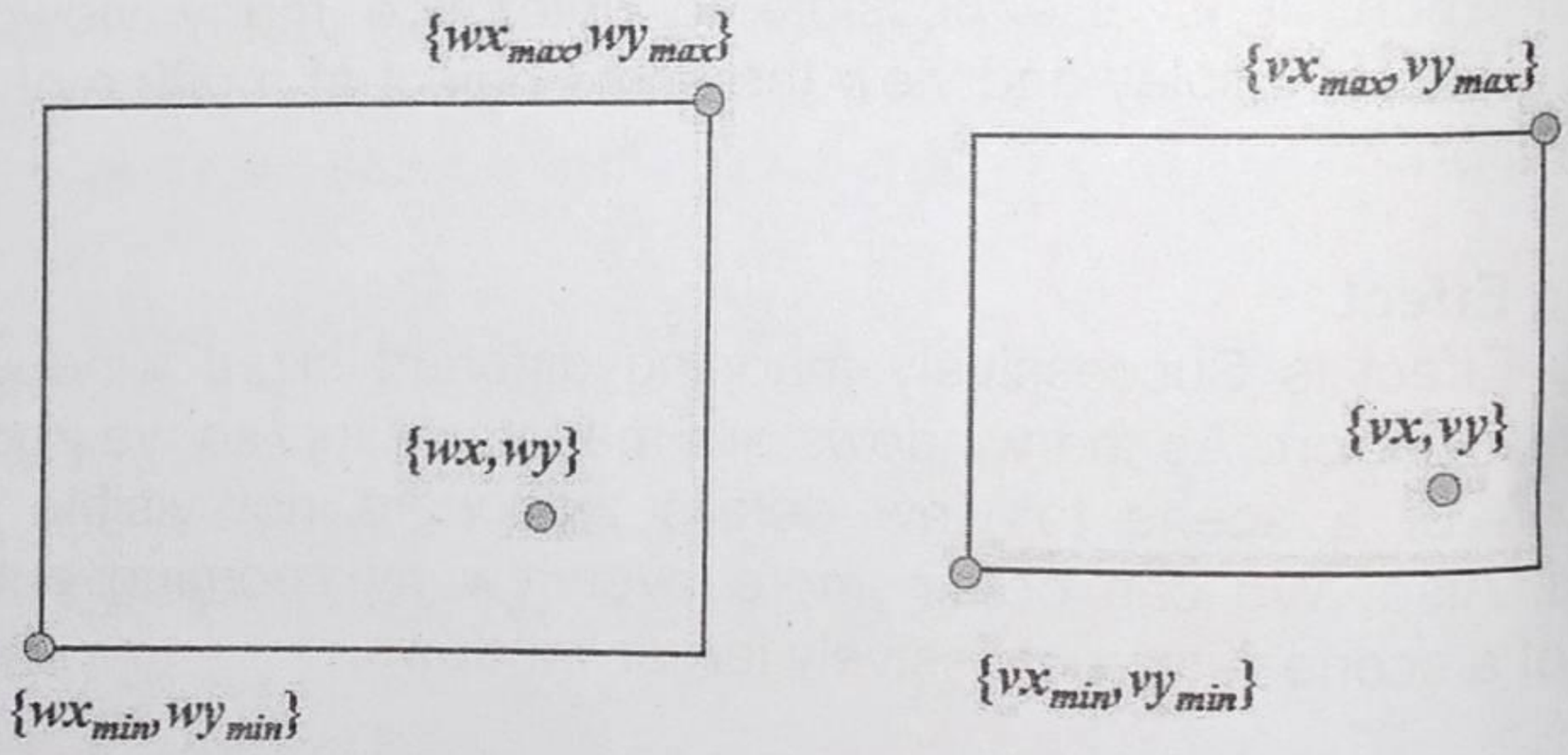
Panning Effect

Panning Effect is actually moving a fixed size window across various objects in a scene.



Window to viewport Transformation

A window is specified by 4 world coordinates, wx_{max}, wx_{min}, wy_{max}, wy_{min} . While, A viewport is specified by vx_{max}, vx_{min}, vy_{max}, vy_{min}. Moreover, The objective of window to viewport mapping is to convert the world coordinate (wx,wy) of an arbitrary point to its corresponding normalized device coordinate (vx,vy).



Viewport Transformation

You have to map a point at the position w_x, w_y in the window into the position v_x, v_y in the viewport.

Window to Viewport Mapping

In order to maintain the same relative placement of the point in the viewport as in window, we require

$$\frac{wx - wx_{min}}{wx_{max} - wx_{min}} = \frac{vx - vx_{min}}{vx_{max} - vx_{min}} \quad and \quad \frac{wy - wy_{min}}{wy_{max} - wy_{min}} = \frac{vy - vy_{min}}{vy_{max} - vy_{min}}$$

Solving for Vx, we get

$$vx = (wx - wx_{min}) * \frac{vx_{max} - vx_{min}}{wx_{max} - wx_{min}} + vx_{min}$$

Similarly, solving for Vy, we get

$$vy = (wy - wy_{min}) * \frac{vy_{max} - vy_{min}}{wy_{max} - wy_{min}} + vy_{min}$$

Algorithm

Step1: Translate window to origin $T_x = -wx_{min}$ $T_y = -wy_{min}$

Step2: Then, Scaling of the window to match its size to the viewport

$S_x = (vx_{max} - vx_{min}) / (wx_{max} - wx_{min})$

$S_y = (vy_{max} - vy_{min}) / (wy_{max} - wy_{min})$

Step3: Again translate viewport to its correct position on screen.

$T_x = vx_{min}$

$T_y = vy_{min}$

Moreover, We can represent above three steps in matrix form:

VT=T1 * S * T

T = Translate window to the origin

S=Scaling of the window to viewport size

T1=Translating viewport on screen.

Viewing Transformation=T1 * S * T

$$N=\begin{bmatrix}1 & 0 & vx_{min}\\ 0 & 1 & vy_{min}\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}sx & 0 & 0\\ 0 & sy & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & -wx_{min}\\ 0 & 1 & -wy_{min}\\ 0 & 0 & 1\end{bmatrix}$$

$$\begin{bmatrix}vx\\ vy\\ 1\end{bmatrix}=\begin{bmatrix}1 & 0 & vx_{min}\\ 0 & 1 & vy_{min}\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}sx & 0 & 0\\ 0 & sy & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & -wx_{min}\\ 0 & 1 & -wy_{min}\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}wx\\ wy\\ 1\end{bmatrix}$$

Example: Find normalization transformation that maps a window whose lower-left corner is at (1,1) and upper right corner is at (3,5) onto: a)Viewport with lower-left corner (0,0) and upper right corner (1,1) b) Viewport with lower left corner (0,0) and upper right corner (1/2,1/2)

wx_{min}=1, wy_{min}=1, wx_{max}=3, wy_{max}=5

a) vx_{min}=0, vy_{min}=0, vx_{max}=1, vy_{max}=1

b) vx_{min}=0, vy_{min}=0, vx_{max}=1/2 , vy_{max}=1/2

a) $S_x = (vx_{max} - vx_{min}) / (wx_{max} - wx_{min})$, $S_y = (vy_{max} - vy_{min}) / (wy_{max} - wy_{min})$

While, Putting the values:

Sx = 1-0/3-1 = 1/2 ,

Sy=1-0/5-1 = 1/4

$$N=\begin{bmatrix}1 & 0 & vx_{min}\\ 0 & 1 & vy_{min}\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}sx & 0 & 0\\ 0 & sy & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & -wx_{min}\\ 0 & 1 & -wy_{min}\\ 0 & 0 & 1\end{bmatrix}$$

$$N=\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}\begin{bmatrix}1/2&0&0\\0&1/4&0\\0&0&1\end{bmatrix}\begin{bmatrix}1&0&-1\\0&1&-1\\0&0&1\end{bmatrix}$$

$$N=\begin{bmatrix}1/2&0&-1/2\\0&1/4&-1/4\\0&0&1\end{bmatrix}$$

b) $S_x = (vx_{max} - vx_{min}) / (wx_{max} - wx_{min})$, $S_y = (vy_{max} - vy_{min}) / (wy_{max} - wy_{min})$

While, Putting the values:

Sx = (1/2-0)/3-1 = (½)/2 = 1/4

Sy=(1/2-0)/5-1 = (1/2)/4 = 1/8

$$N=\begin{bmatrix}1&0&vx_{min}\\0&1&vy_{min}\\0&0&1\end{bmatrix}\begin{bmatrix}sx&0&0\\0&sy&0\\0&0&1\end{bmatrix}\begin{bmatrix}1&0&-wx_{min}\\0&1&-wy_{min}\\0&0&1\end{bmatrix}$$

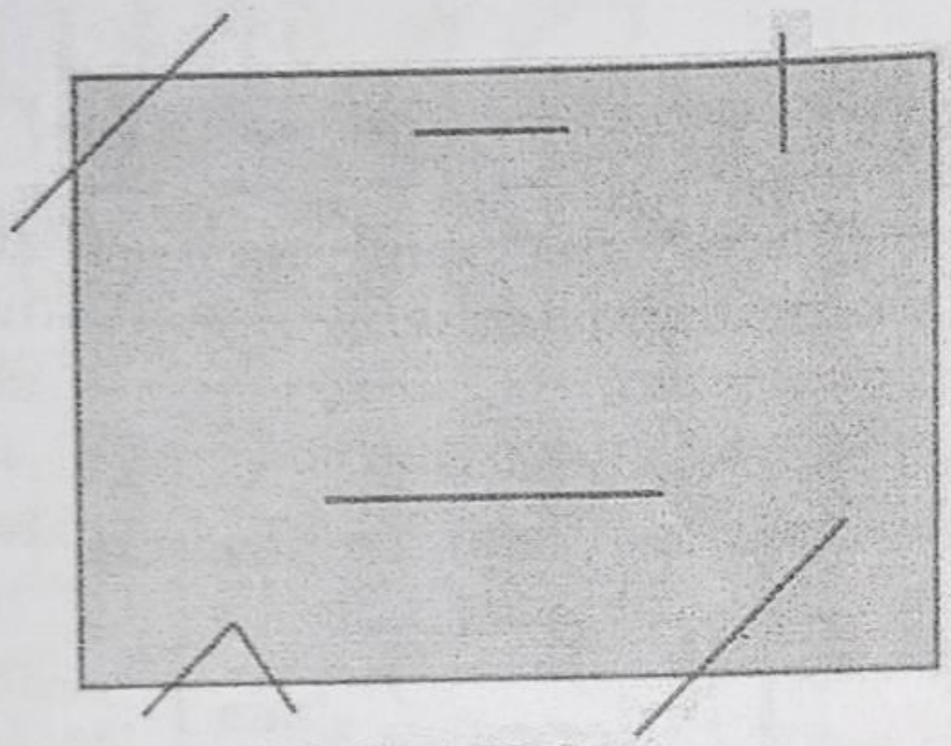
$$N=\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}\begin{bmatrix}1/4&0&0\\0&1/8&0\\0&0&1\end{bmatrix}\begin{bmatrix}1&0&-1\\0&1&-1\\0&0&1\end{bmatrix}$$

$$N=\begin{bmatrix}1/4&0&-1/4\\0&1/8&-1/8\\0&0&1\end{bmatrix}$$

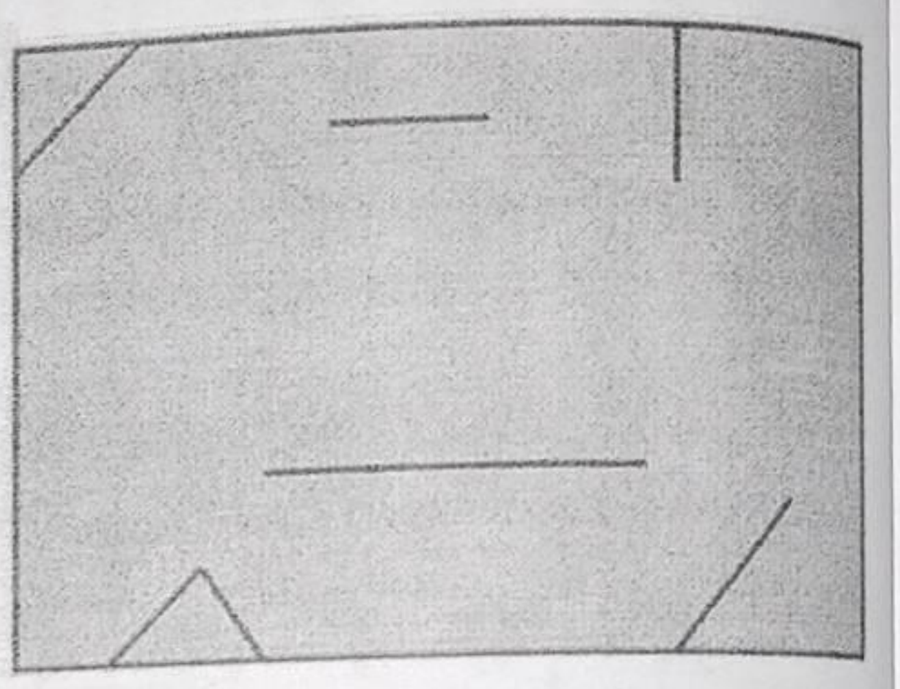
2. CLIPPING

2.1 Clipping and Clip Window

First of all we define a region. We can call this region Clip Window. Then, we will keep all the portions or objects which are inside this window. While all the things outside this window are discarded. So we select a particular portion of a scene and then we display only that part. Rest of the part is not displayed. So clipping involves, identifying which portion is outside clip window. Then discarding that portion. Clip window may be rectangular or any other polygon or it may be a curve also. The following picture shows the clipping effect.



Original Picture
or
Before Clipping



After Clipping

Applications of Clipping

- We can select a part from a scene for displaying. Just like we use camera to click a picture. We set our camera to focus at a particular portion and then discard the things which are out of frame.
- In three-dimensional views, only some surfaces are visible and others are hidden. We can use clipping to identify visible surfaces.
- During the process of Rasterization, We see aliasing effect (zig-zag) in line segments or object boundaries. We can use clipping for antialiasing.
- Solid-modeling procedures also use Clipping.
- We can use clipping for multiwindow environment
- With the help of Clipping Operation we can select a part of picture. Then we can copy, move, or delete that part. Various painting and Drawing Applications have this feature.

Approaches of Clipping

I hope you have already read my post about Window and Viewport. If not you can quickly read it to understand the concepts of viewport and window. Clipping can be performed in two ways:

We can perform Clipping in world coordinates before mapping it to device coordinates. In this case, we can save unnecessary operations. We need not map those objects that we will ultimately discard. While in Viewport clipping we perform clipping after mapping it to device coordinates.

Types of Clipping

There are various types of Clipping:

- Point Clipping
- Line Clipping
- Polygon Clipping
- Text Clipping
- Curve Clipping

Point Clipping

First of all consider a point P(x, y). We can specify Edges of Clip Window by (xw_{min}, xw_{max}, yw_{min}, yw_{max}).

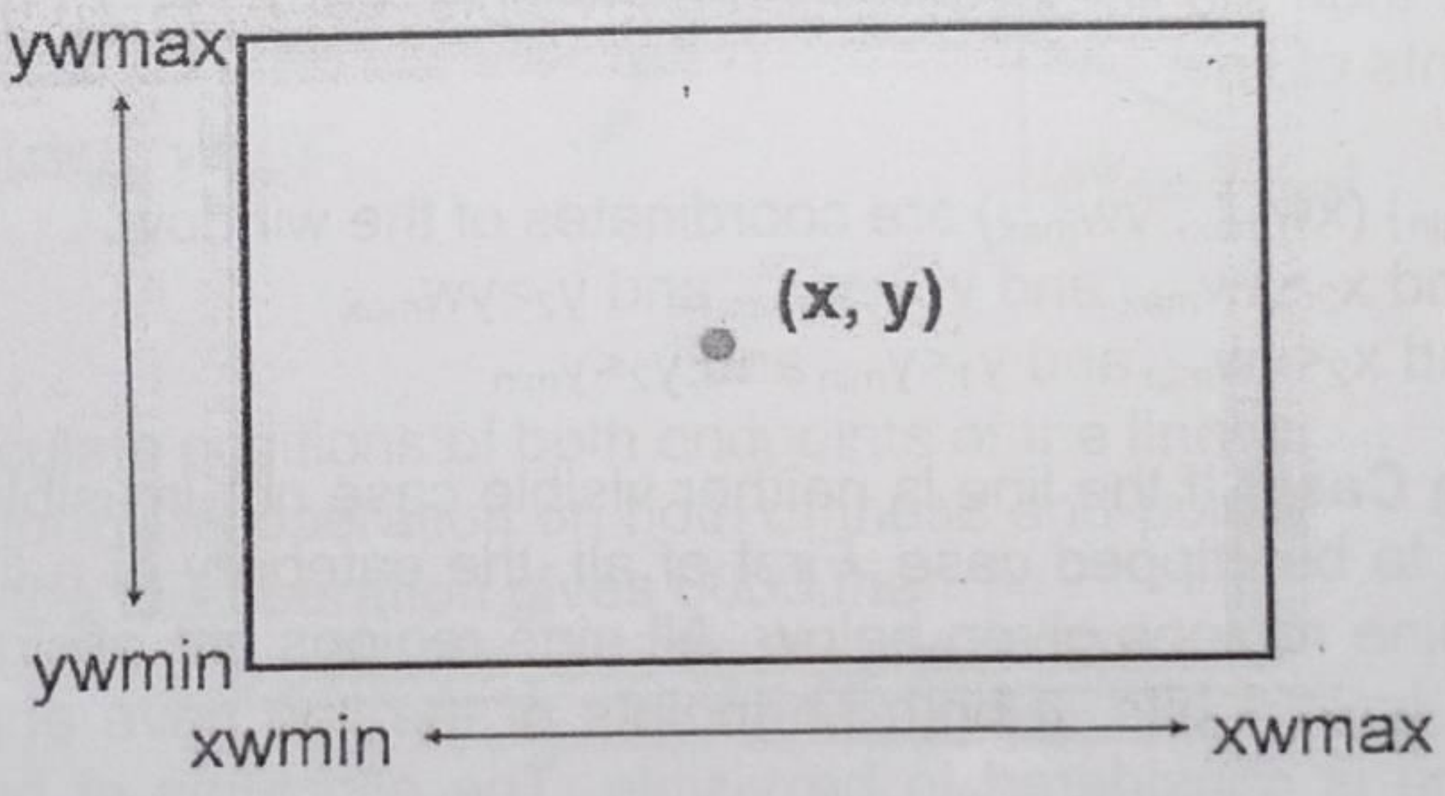
If following four inequalities are not satisfied,

$xw_{min} <= x <= xw_{max}$

$yw_{min} <= y <= yw_{max}$

Then point is clipped

Otherwise, displayed.



Line Clipping

We specify a line with its end-points. There are three possible cases for a line that we need to consider.

- First of all we will check if a line is completely inside the window. If it is then we will display it completely.
- Otherwise, we will check if a line is completely outside the window. If it is then we will clip that line.
- Thirdly, line is neither completely inside nor completely outside. This line is partially visible. This is a line intersecting clipping boundaries.

2.2 Cohen Sutherland Line Clipping Algorithm

Cohen Sutherland Algorithm is a line clipping algorithm that cuts lines to portions which are within a rectangular area. It eliminates the lines from a given set of lines and rectangle area of interest (view port) which belongs outside the area of interest and clip those lines which are partially inside the area of interest.

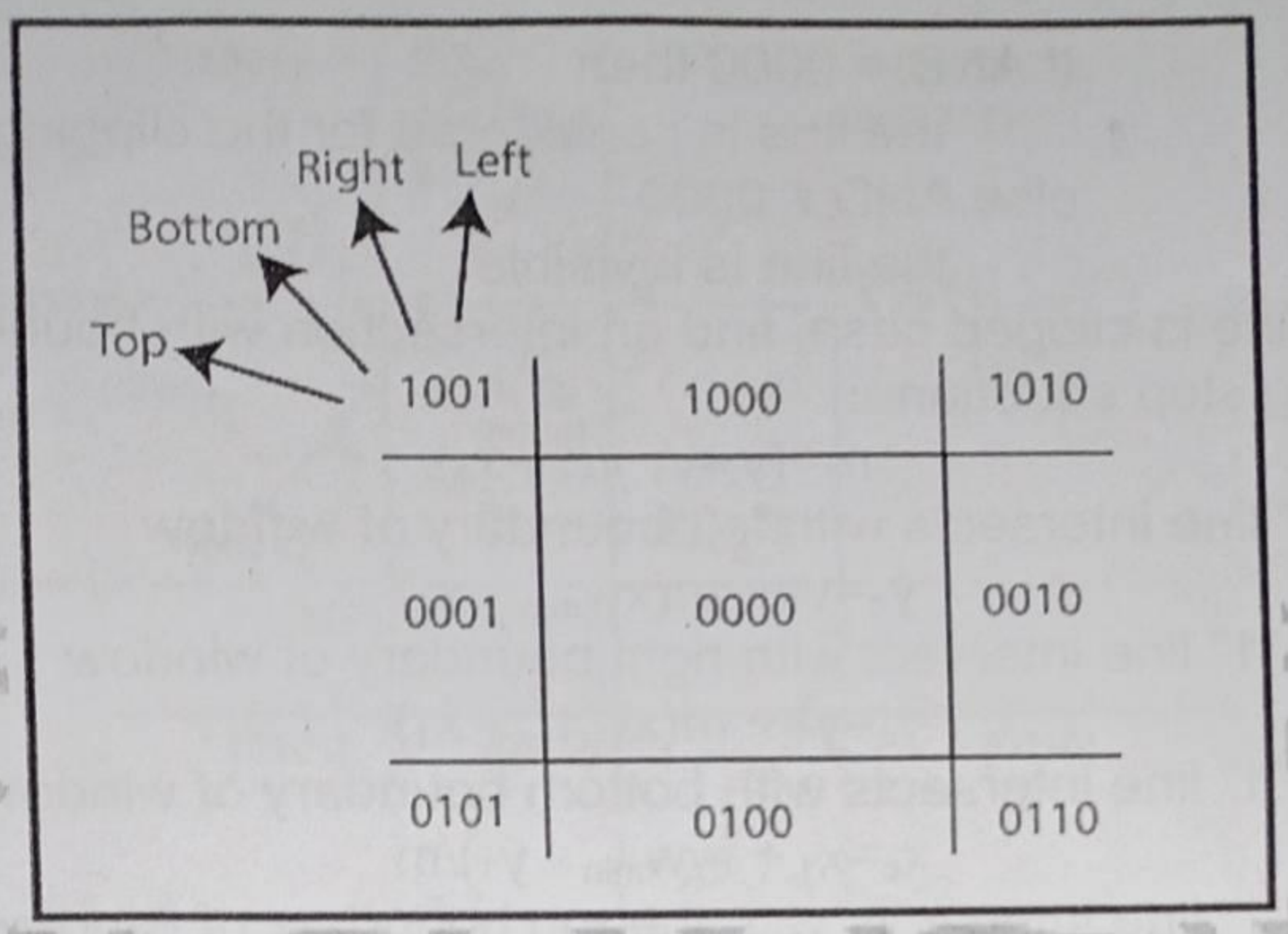
In the algorithm, first of all, it is detected whether line lies inside the screen or it is outside the screen. All lines come under any one of the following categories:

1. Visible: If a line lies within the window, i.e., both endpoints of the line lies within the window. A line is visible and will be displayed as it is.

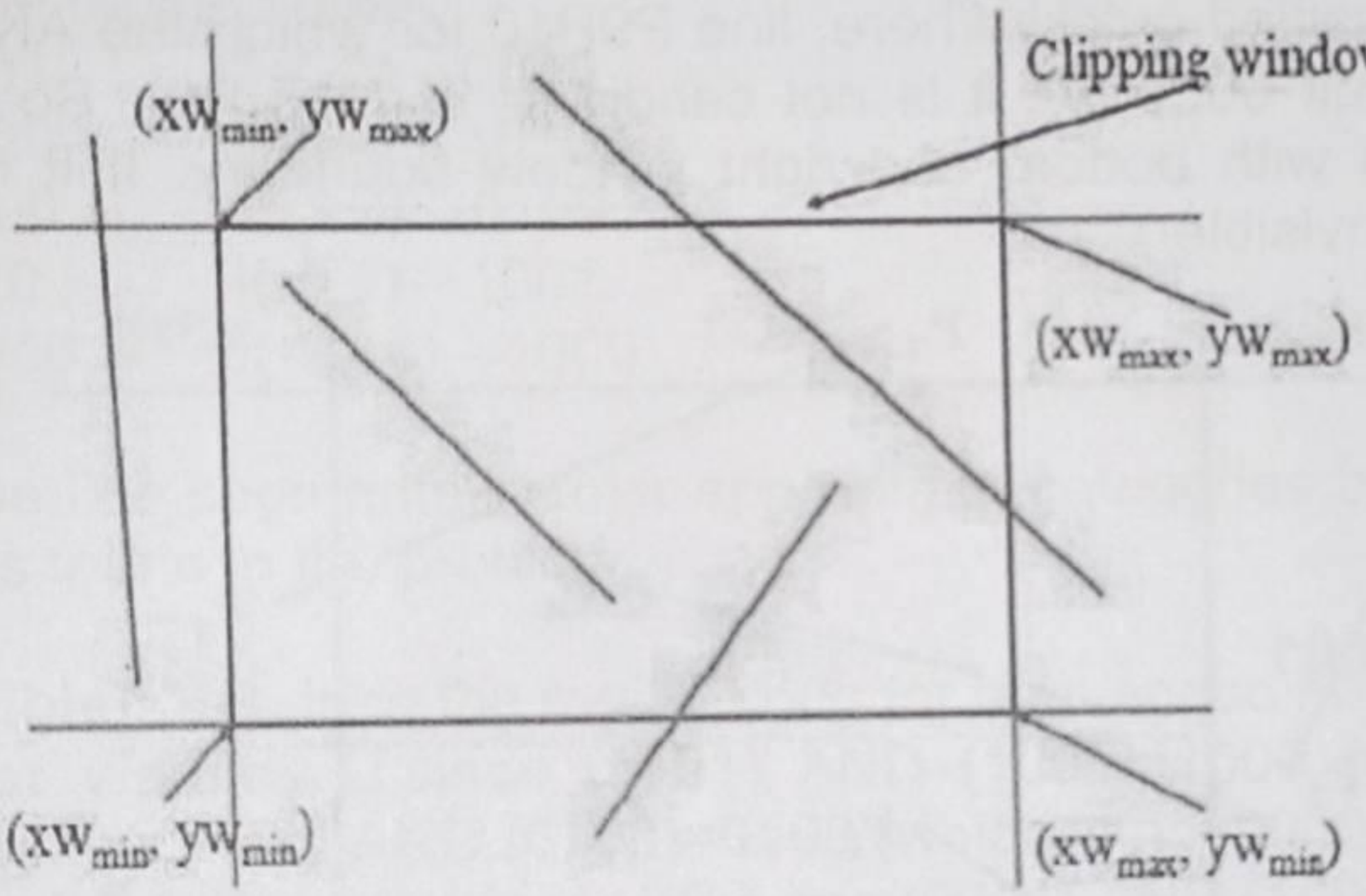
2. Not Visible: If a line lies outside the window it will be invisible and rejected. Such lines will not display. If any one of the following inequalities is satisfied, then the line is considered invisible. Let A (x_1,y_2) and B (x_2,y_2) are endpoints of line.

(xw_{min}, yw_{min}) (xw_{max}, yw_{max}) are coordinates of the window.
$x_1>xw_{max}$ and $x_2>xw_{max}$ and $y_1>yw_{max}$ and $y_2>yw_{max}$
$x_1<xw_{min}$ and $x_2<xw_{min}$ and $y_1<y_{min}$ and $y_2<y_{min}$

3. Clipping Case: If the line is neither visible case nor invisible case. It is considered to be clipped case. First of all, the category of a line is found based on nine regions given below. All nine regions are assigned codes. Each code is of 4 bits. If both endpoints of the line have end bits zero, then the line is considered to be visible. The allocation of bits depends on "TBRL" (Top, Bottom, Right, Left) rule.



The center area is having the code, 0000, i.e., region 5 is considered a rectangle window. **Following figure show lines of various types**



Algorithm

Step 1: Calculate positions of both endpoints of the line
Step 2: Perform OR operation on both of these end-points
Step 3: If the OR operation gives 0000 then
line is considered to be visible
else
Perform AND operation on both endpoints

If AND = 0000 then
the line is considered for the clipping.
else AND ≠ 0000
the line is invisible

Step 4: If a line is clipped case, find an intersection with boundaries of the window using slop equation

$$m=(y_2-y_1)/(x_2-x_1)$$

(a) If L bit "1" line intersects with left boundary of window

$$y_c=y_1 + m(xw_{min} - x_1)$$

(b) If R bit is "1" line intersect with right boundary of window

$$y_c=y_1 + m(xw_{max} - x_1)$$

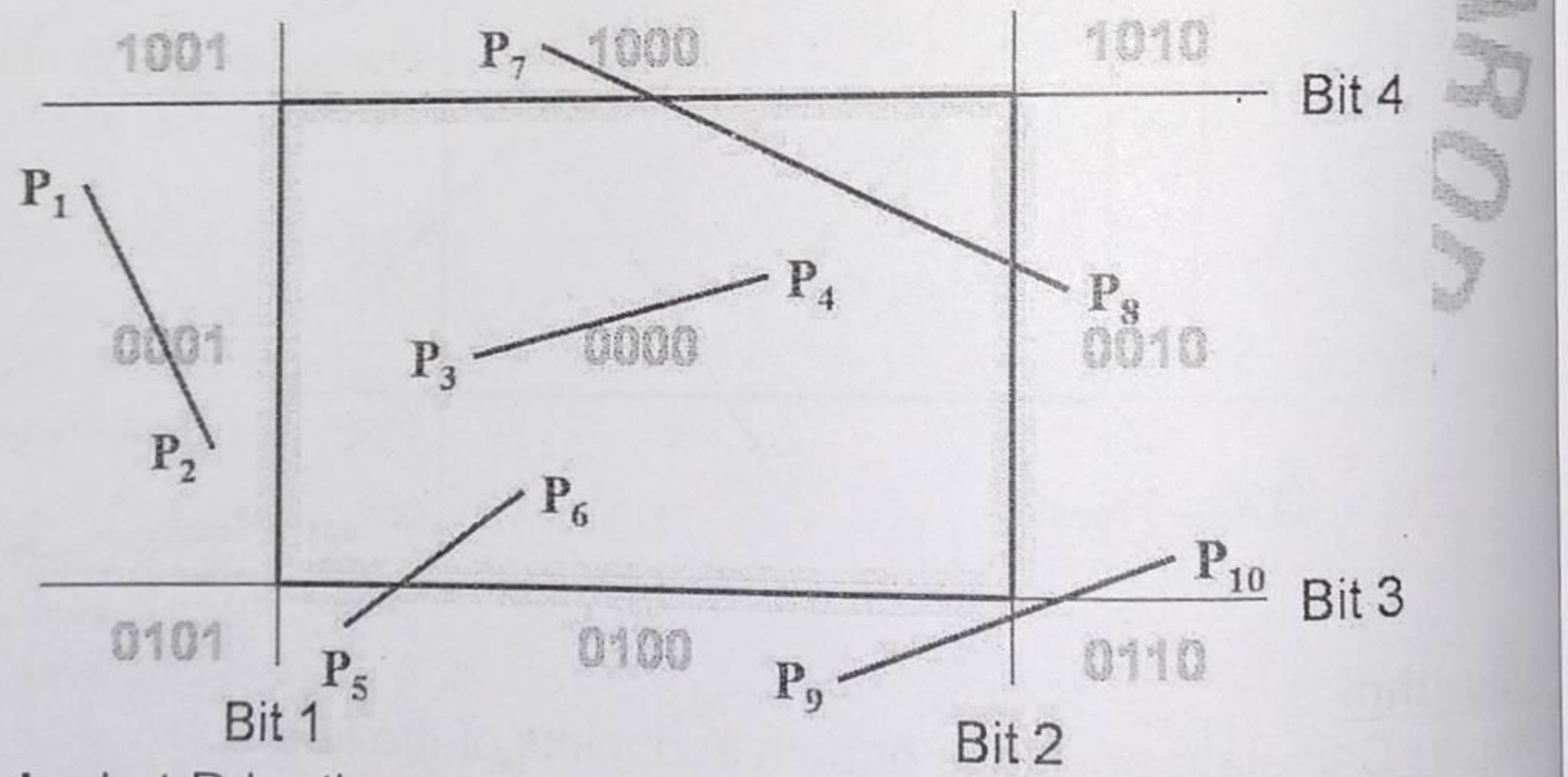
(c) If B bit is "1" line intersects with bottom boundary of window

$$x_c=x_1 + (yw_{min} - y_1)/m$$

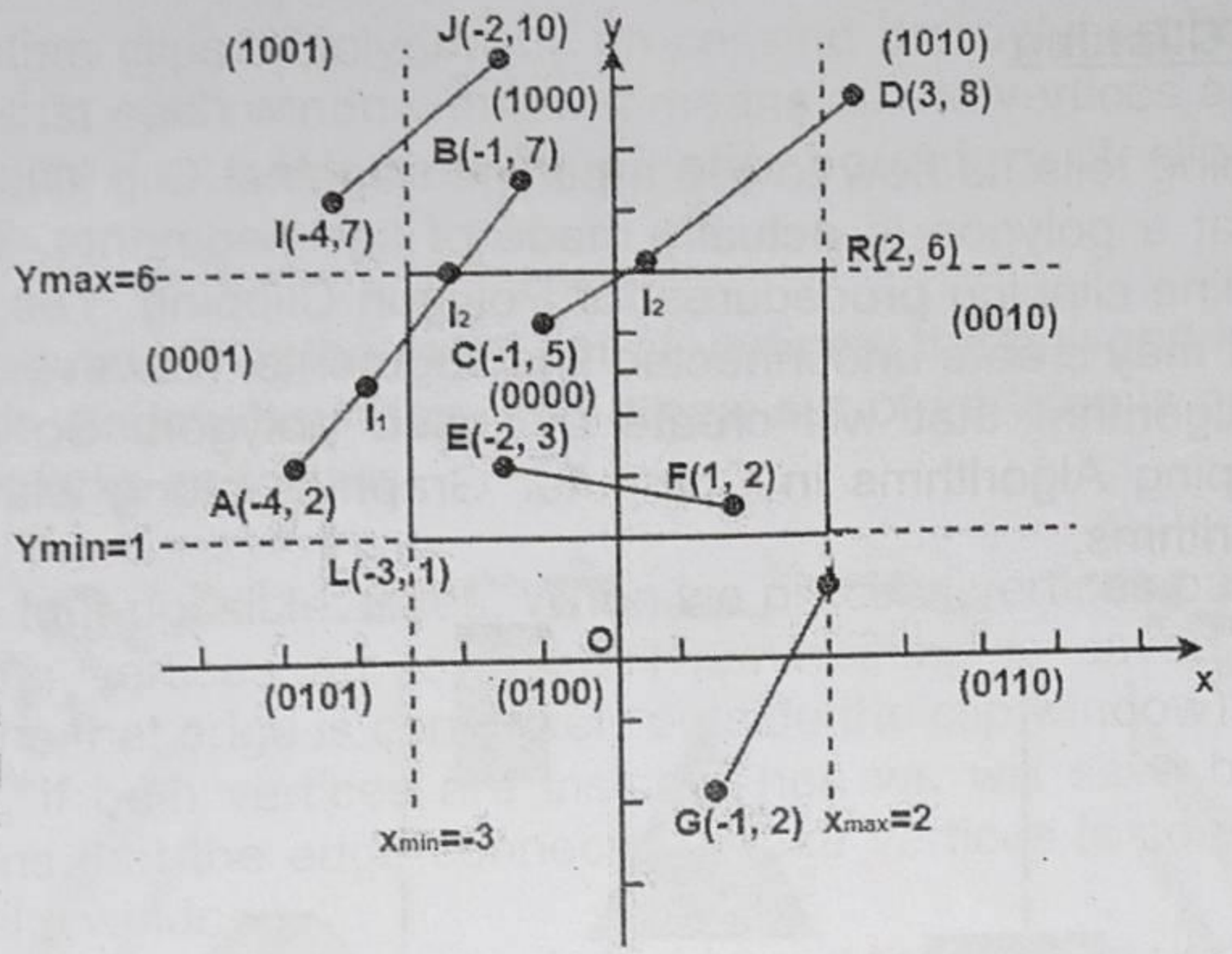
(d) If T bit is "1" line intersects with the top boundary of window

$$x_c=x_1 + (yw_{max} - y_1)/m$$

Special Case: Compute intersections with window boundary for lines that can't be classified quickly (here, line P9P10 for which the AND operation will give result 0000 yet it is not candidate for clipping. So find out the intersections with bottom and right window boundary. If it is not found then line is invisible.)



Example: Let R be the rectangular window whose lower left-hand corner is at L (-3, 1) and upper right-hand corner is at R (2, 6). Find the region codes for the endpoints in fig:



Solution:

So, as per the region codes,

A(-4,2)→0001	F(1,2)→0000
B(-1,7)→1000	G(1,-2)→0100
C(-1,5)→0000	H(3,3)→0100
D(3,8)→1010	I(-4,7)→1001
E (-2,3)→0000	J(-2,10)→1000

We place the line segments in their appropriate categories by testing the region codes found in the problem.

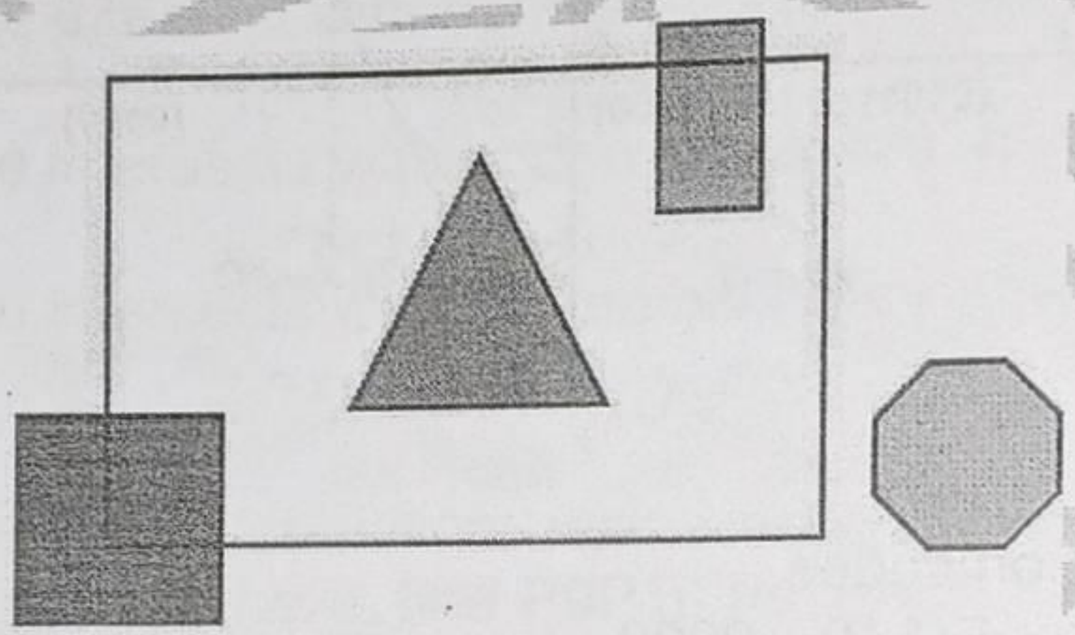
Case 1 (visible): EF since the region code for both endpoints is 0000.
Case 2 (not visible): IJ since (1001) AND (1000)=1000 (which is not 0000). GH since (0100) AND (0100)=0100 (which is not 0000).
Case 3 (candidate for clipping): AB since (0001) AND (1000)=0000, CD since (0000) AND (1010)=0000.

Finding intersection points for y for left and x for top boundary of AB, we get, $(-3, {}^{11}/_{3})$ and $(-{}^{8}/_{5}, 6)$ as end points.

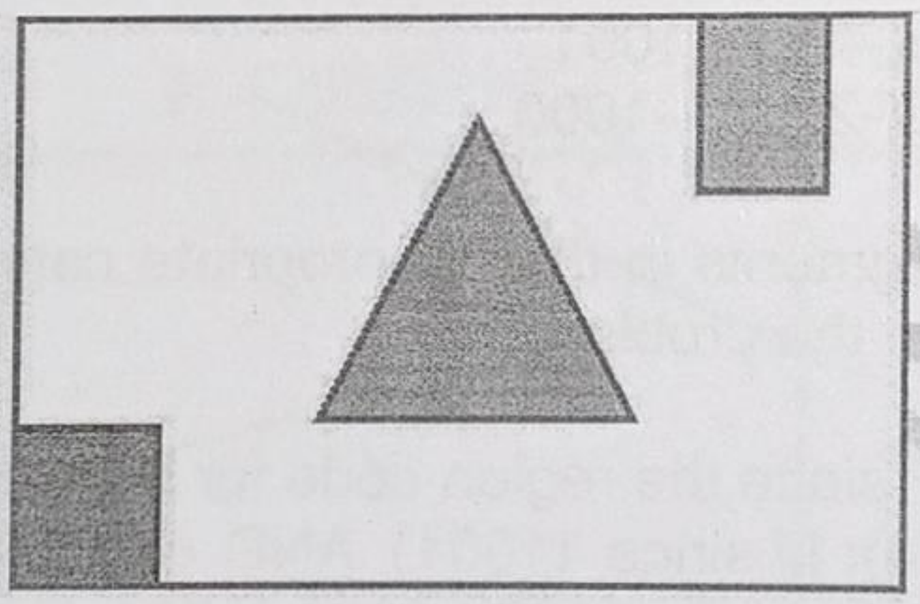
For CD, C is inside we have to just calculate intersection points with top boundary. By using the formula, we get $({}^{3}/_{5}, 6)$

2.3 Polygon Clipping

Polygon clipping tells us how to clip a polygon against Clip Window. You may think that a polygon is actually made of line segments. Therefore, We can use line clipping procedures for Polygon Clipping. Yes, you may do that. But it may create unconnected line segments. However, we need a Clipping Algorithm that will create a closed polygon. So there are Polygon Clipping Algorithms in Computer Graphics along with the Line Clipping Algorithms.



Before Clipping



After Clipping

Sutherland-Hodgeman Polygon Clipping Algorithm:

This algorithm was purposed by Sutherland and Hodgeman in 1974 to determine which portions of the polygon were visible within a window and clipping out the portions that are outside the window.

This algorithm accepts as input a series of polygon vertices, V_1, V_2, ... V_n. These vertices define polygon edges from V_i, to V_{i+1} and from V_n to V_1.

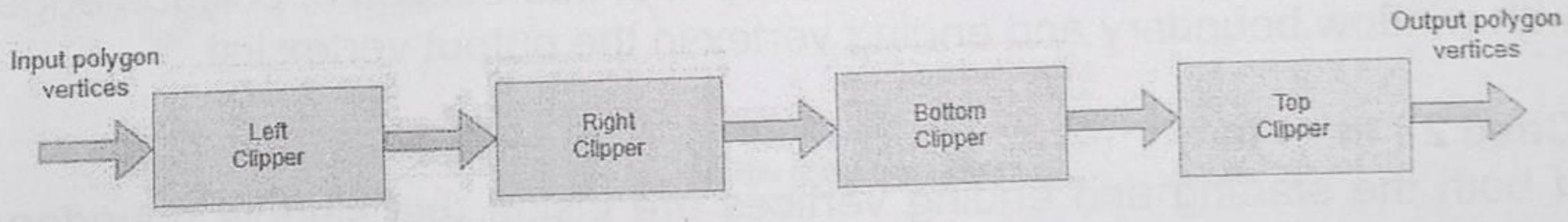
The algorithm clips a polygon by processing the polygon boundary as a whole against each window edge. It means all the vertices are processed for clipping against **each individual clip boundary** of clipping window one by one.

It means starting with the initial set of vertices, the polygon is first clipped against left window boundary and a new set of vertices is produced. The algorithm works as follows:

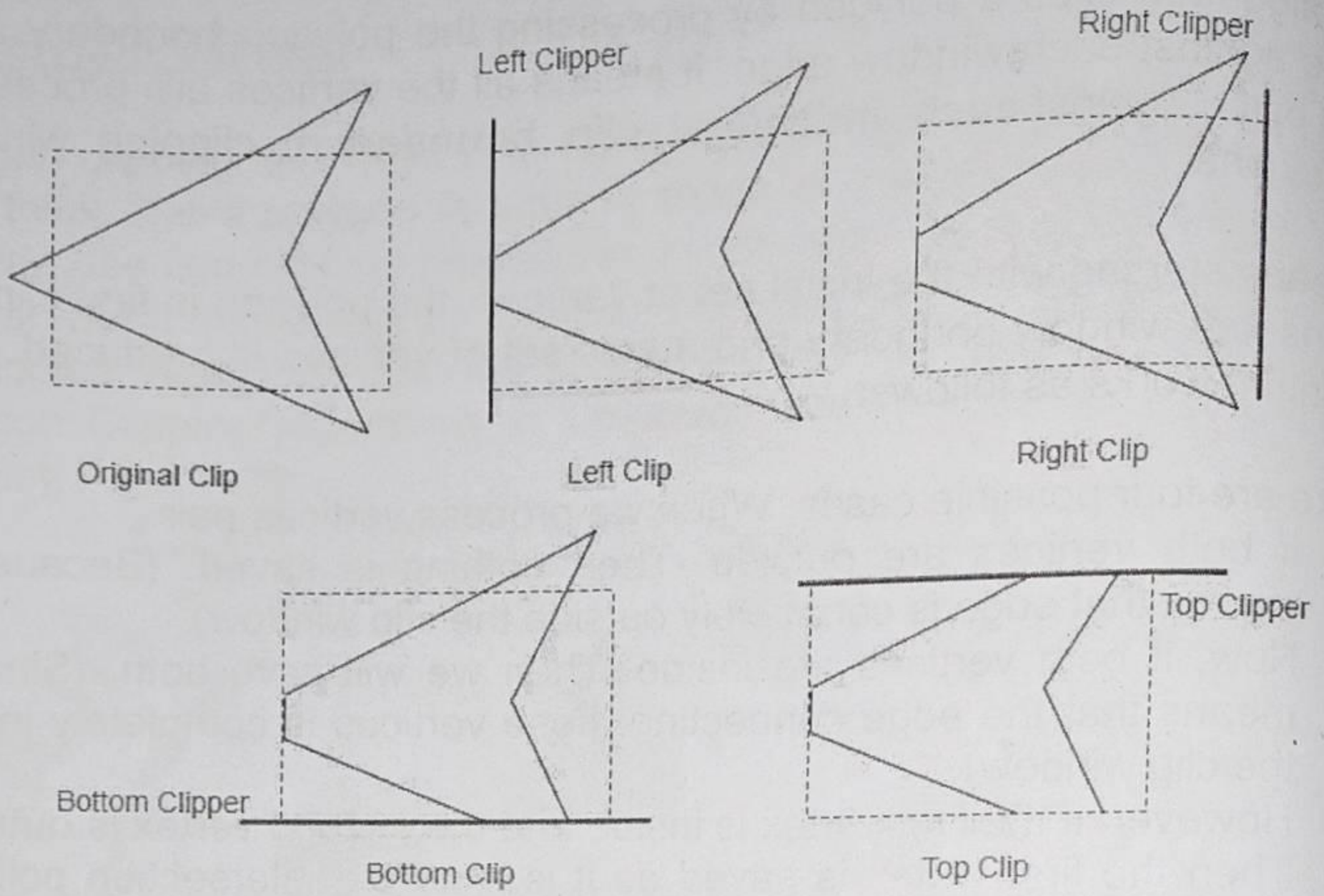
There are four possible cases. When we process vertices pair.

1. If both vertices are outside. Then nothing is saved. (Because It means that edge is completely outside the clip window).
2. Now, if both vertices are inside. Then we will save both. (Since it means that the edge connecting those vertices is completely inside the clip window).
3. However, if the first vertex is inside and the second vertex is outside. Then the first vertex is saved as it is. And the intersection point of that polygon edge with clip window is saved.
4. But if the second vertex is inside and the first vertex is outside. Then we will save the second vertex. We will also save the intersection point of the polygon edge with a clip window.

This set is given as input to next clipping boundary and the polygon is clipped against right clipping boundary to produce a new set of vertices and is then clipped against bottom and top clipping boundary to produce final set of vertices . ***[Left-Right-Bottom-Top]***



Thus, at each step a new set of output vertices is generated and is passed to the next window boundary clipper.

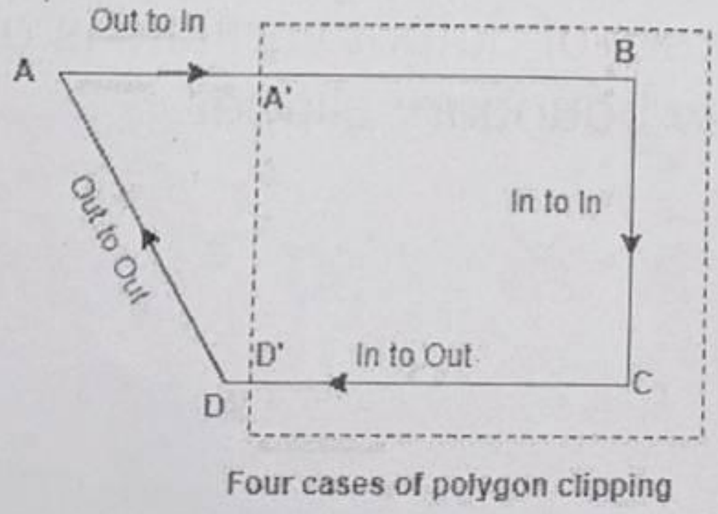


Clipping Polygon against all four clip boundaries

When all the vertices of a given polygon are processed against any clipping side, there could be any one of the following four possibilities for a successive pair of two vertices of polygon:

Case 1: out → in :
If the starting vertex is outside the window boundary and ending vertex is inside then we preserve or save both point of intersection of polygon edge with window boundary and ending vertex in the output vertex list.

Case 2 : in → in :
If both the starting and ending vertices are inside the window boundary then only the ending vertex is preserved or saved in output vertex list.



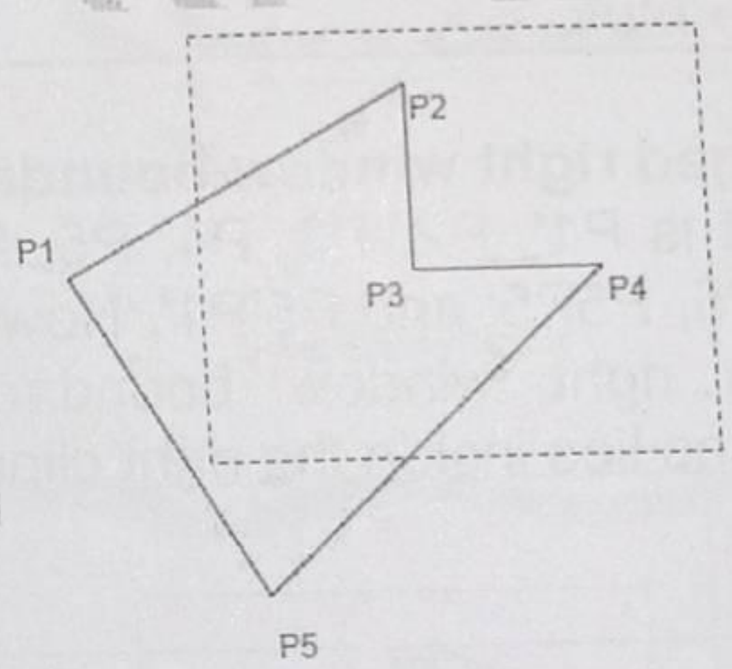
Four cases of polygon clipping

Case 3: in → out :
If the starting vertex is inside the window and ending vertex is outside then only the point of intersection of polygon edge with window boundary is saved in output list.

Case 4: out → out :
If both the starting and ending vertices are outside the window boundary then nothing is added or saved in the output vertex list.

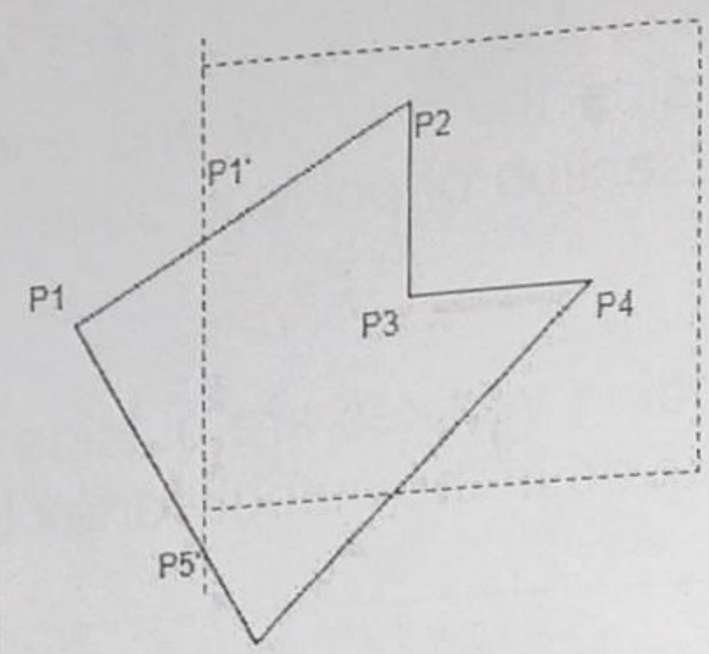
Case	Output
Out→In	**A'B** *[2 Vertices, Point of intersect with window and ending edge]*
In→In	**C** *[1 ending vertex]*
In→Out	**D'** *[Point of intersect with window]*
Out→Out	**N/A** *[Nothing is included]*

Example: Let us consider a polygon with vertices P1, P2, P3, P4, P5. Inside in a viewing window. Applying Sutherland-Hodgeman Polygon Clipping Algorithm on this polygon, we get:



Example of Polygon Clipping

Clipping polygon engaged left window boundary:
Here the input vertex list is P1, P2, P3, P4 P5 and the edges are P1P2 P2P3, P3P4, P4P5 and P5P1. Now, consider all the edge with respect to (w.r.t.) left window boundary.

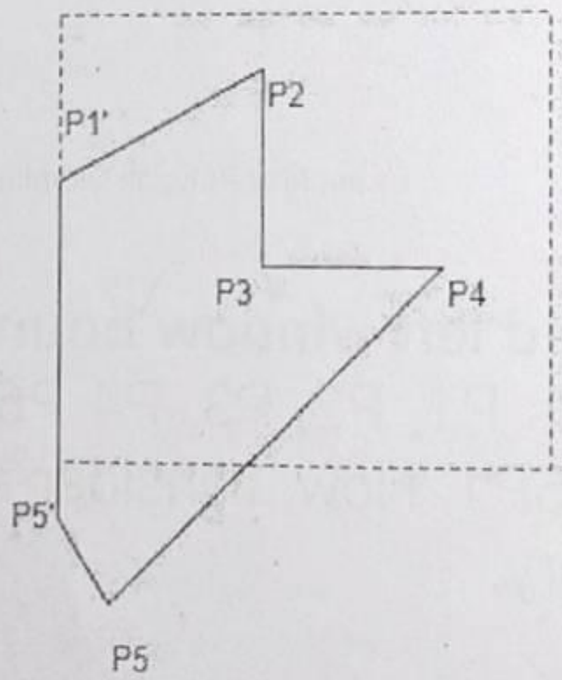


Using Left Clipper

Edge	Type	Output
P1P2	Out → In	P1'P2
P2P3	In → In	P3
P3P4	In → In	P4
P4P5	In → In	P5
P5P1	In → Out	P5′

Clipping polygon engaged right window boundary:

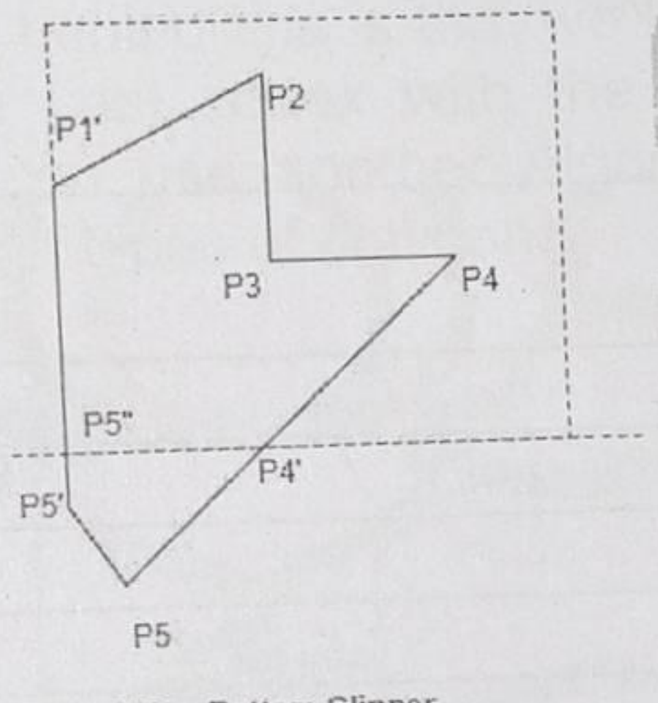
Here the input vertex list is P1′, P2, P3, P4, P5, P5′ and the edges a P1'P2, P2P3, P3P4, P4P5, P5P5′ and P5'P1′. Now, consider all the ed with respect to (w.r.t.) right window boundary. There will be **changes** because all edges lies inside the right clipping window.



Using Right Clipper

Edge	Type	Output
P1'P2	In → In	'P2
P2P3	In → In	P3
P3P4	In → In	P4
P4P5	In → In	P5
P5P5′	In → In	P5′
P5'P1′	In → In	P1′

Clipping polygon engaged bottom window boundary:
Here the input vertex list is P1′, P2, P3, P4, P5, P5′ and the edges are P1'P2, P2P3, P3P4, P4P5 and P5P5′. Now, consider all the edge with respect to (w.r.t.) bottom window boundary.

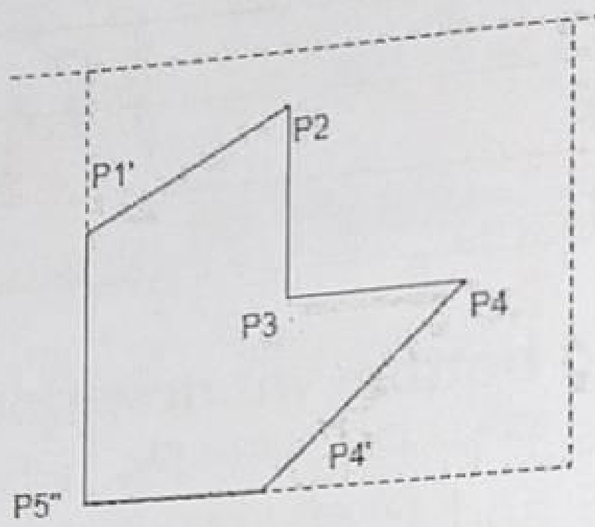


Using Bottom Clipper

Edge	Type	Output
P1'P2	In → In	P2
P2P3	In → In	P3
P3P4	In → In	P4
P4P5	In → Out	P4′
P5P5′	Out → Out	N/A
P5'P1′	Out → In	P5"

Clipping polygon engaged top window boundary:
Here the input vertex list is P1′, P2, P3, P4, P4′, P5" and the edges are P1'P2, P2P3, P3P4, P4P4′, P4'P5" and P5"P1. Now, consider all the

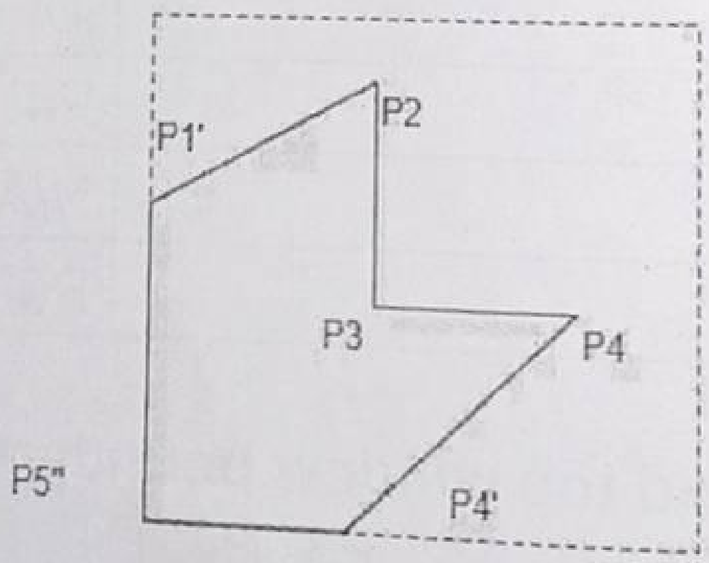
edge with respect to (w.r.t.) top window boundary. Here the input vertex list is P1′, P2, P3, P4, P5, P5′ and the edges are P1'P2, P2P3, P3P4, P4P5, P5P5′ and P5'P1′. Now, consider all the edge with respect to (w.r.t.) right window boundary. There will be **no changes** because all edges lies inside the top clipping window.



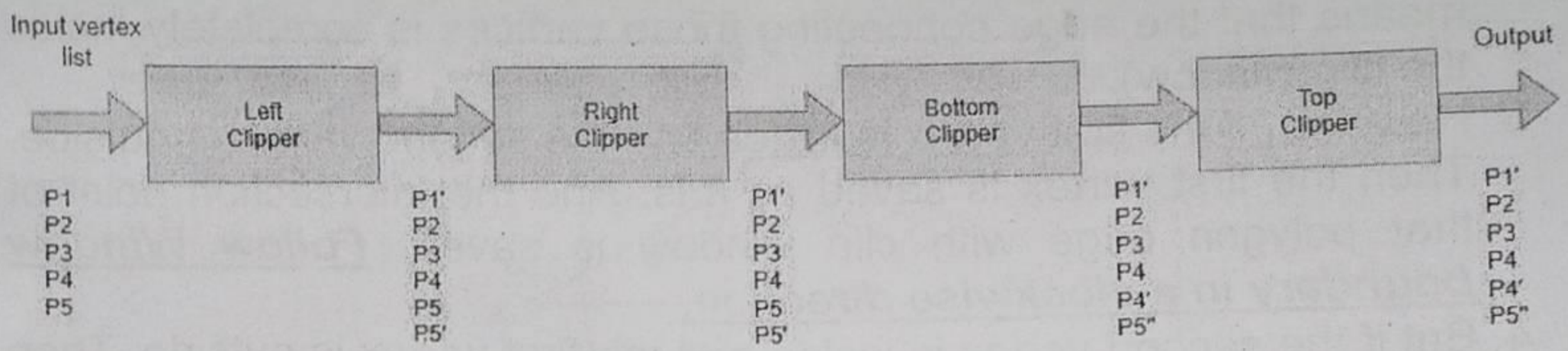
Using Top Clipper

Edge	Type	Output
P1'P2	In → In	P2
P2P3	In → In	P3
P3P4	In → In	P4
P4P4′	In → In	P4′
P4'P5"	In → In	P5"
P5"P1′	In → In	P1′

Final output using Sutherland-Hodgeman Polygon Clipping Algorithm will be:



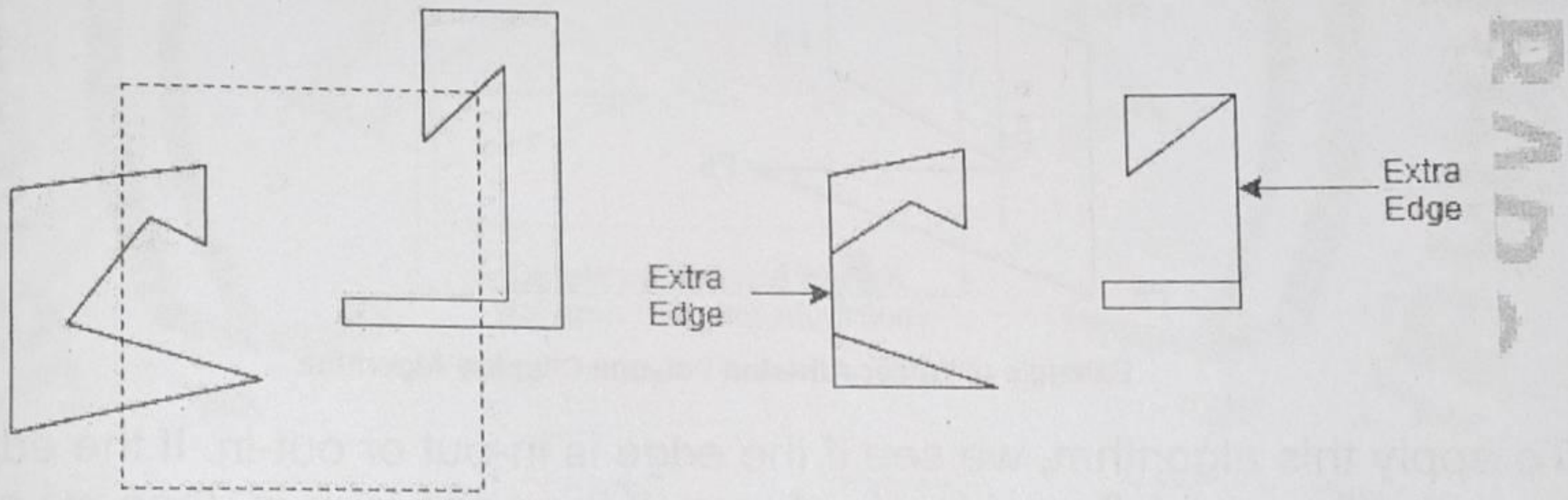
Final Output



Vertex list produced by various clippers

Limitations of Sutherland-Hodgeman Polygon Clipping Algorithm

We can clip Convex Polygons correctly with Sutherland-Hodgeman Polygon Clipping. But we may get wrong results if we process concave polygons with it. The reason behind this is that we have a set of vertices as output. And we join the Last vertex with the first one to make a bounded polygon. So, we can use another Algorithm Weiler-Atherton Algorithm which works for both types of polygons.



Concave polygon not clipped corrects

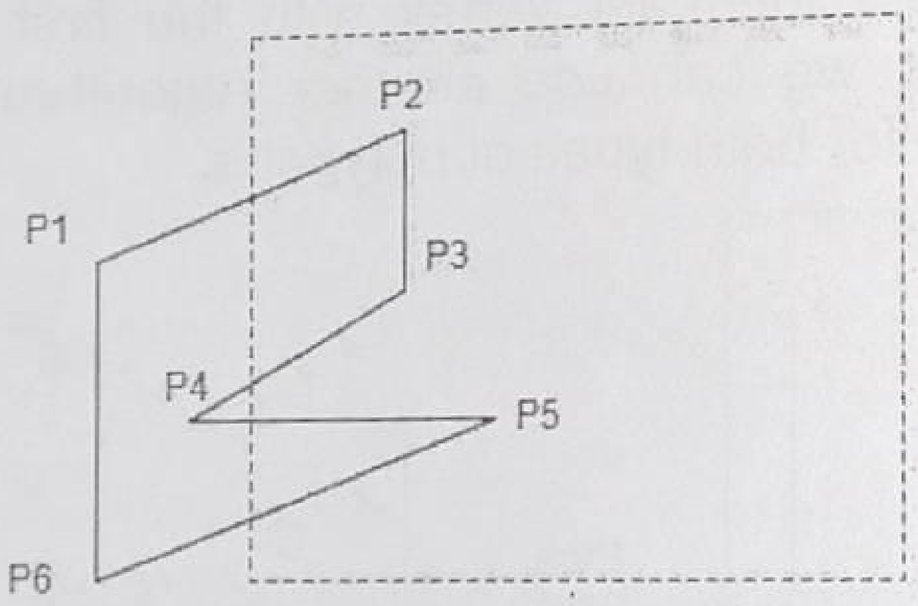
Weiler-Atherton Polygon Clipping

This Clipping procedure is suitable for Concave Polygons. We process pair of vertices of polygon either clockwise or anticlockwise and test them against the Clip window boundary one by one. Suppose we are processing vertices in a clockwise direction.

1. If both vertices are outside. Then nothing is saved. (Because It means that edge is completely outside the clip window).

2. Now, if both vertices are inside. Then we will save both. (Since it means that the edge connecting those vertices is completely inside the clip window).
3. However, if the first vertex is inside and the second vertex is outside. Then the first vertex is saved as it is. And the intersection point of that polygon edge with clip window is saved. ***Follow Window boundary in a clockwise direction.***
4. But if the second vertex is inside and the first vertex is outside. Then we will save the second vertex. We will also save the intersection point of the polygon edge with a clip window. ***Follow Polygon boundary***.

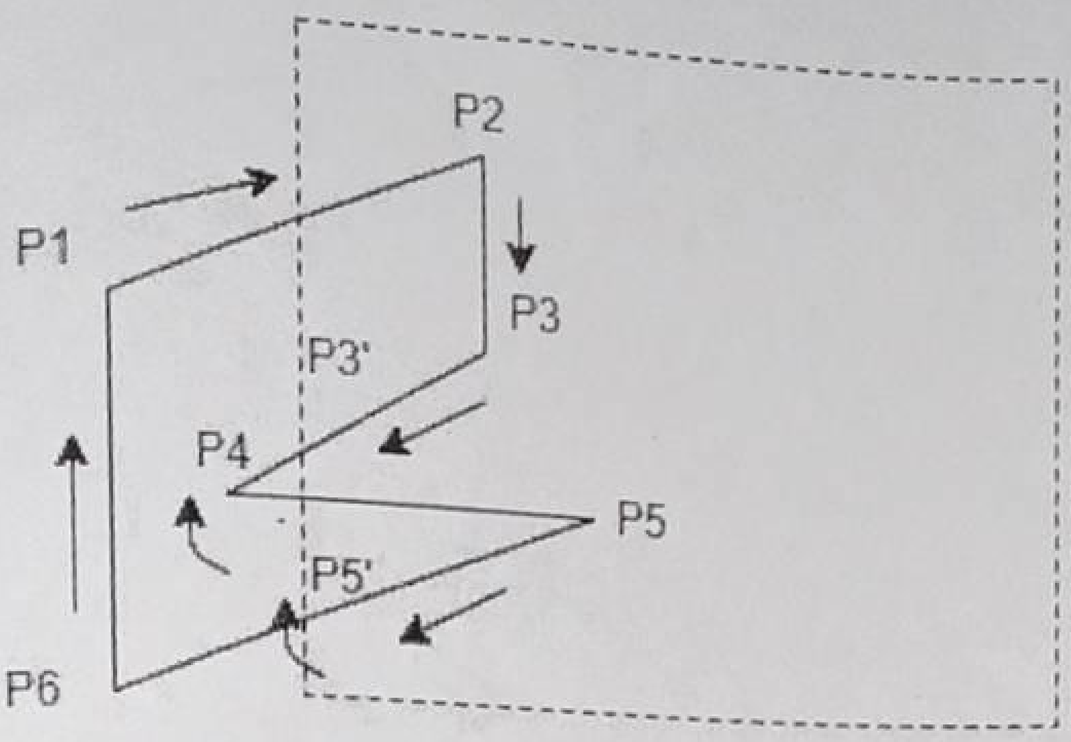
Example: Let us consider a polygon P1, P2, P3, P4, P5, P6 with edges P1P2, P2P3, P3P4, P4P5, P5P6 and P6P1.



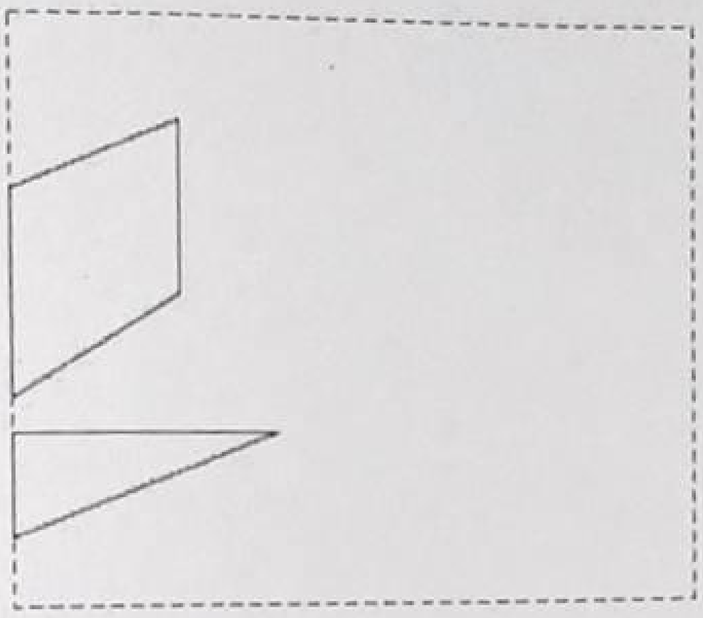
Example of Weiler-Atherton Polygon Clipping Algorithm

To apply this algorithm, we see if the edge is in-out or out-in. If the edge is out-in, then we follow side of polygon. If the edge is in-in, then we save intersecting point on window boundary.

1. P1P2 is out-in edge, following side of polygon.
2. P2P3 is in-in edge, we skip this and include this edge.
3. P3P4 is in-out edge, saving intersecting point on window boundary i.e. P3′.
4. P4P5 is out-in edge, following side of polygon.
5. P5P6 is in-out edge, saving intersecting point on window boundary i.e. P5′.
6. P6P1 is out-out edge, we skip this and exclude this edge.



ere two separate polygon areas generated without any extra line onnecting two polygons.



Output of Weiler-Atherton
Polygon Clipping Algorithm