## The Viewing Pipeline

- **Window:** Area selected in world-coordinate for display is called window. It defines what is to be viewed.
- **Viewport:** Area on a display device in which window image is display (mapped) is called viewport. It defines where to display.
- In many case window and viewport are rectangle, also other shape may be used as window and viewport.
- In general finding device coordinates of viewport from word coordinates of window is called as **viewing transformation.**
- Sometimes we consider this viewing transformation as window-to-viewport transformation but in general it involves more steps.

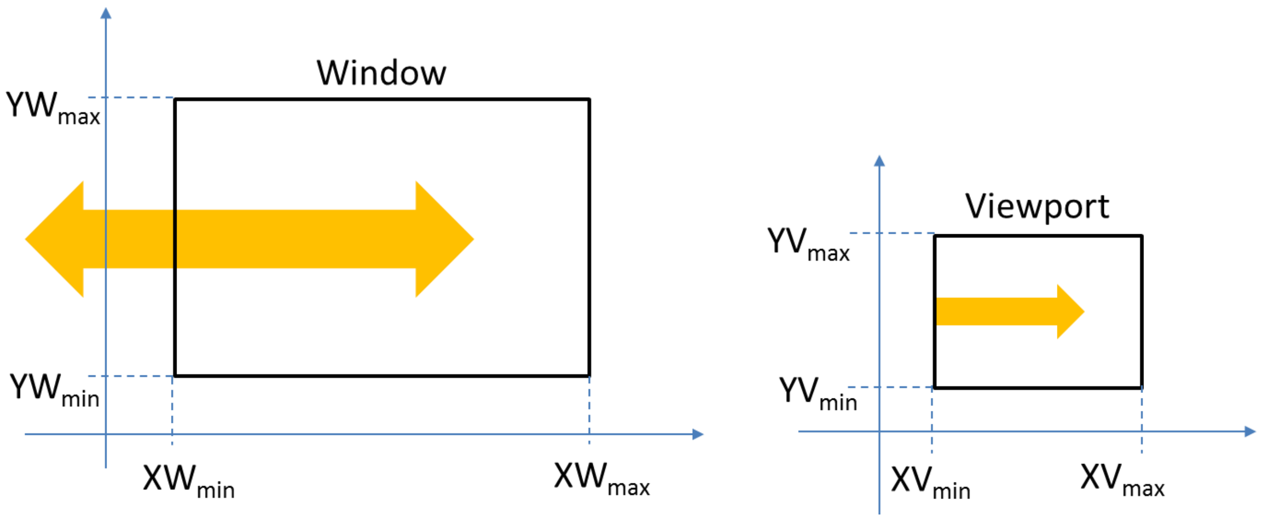
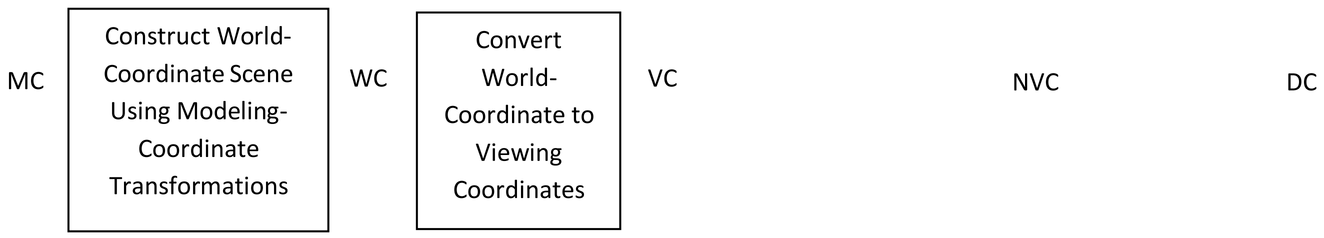


Fig. 3.1: - A viewing transformation using standard rectangles for the window and viewport.

- Now we see steps involved in viewing pipeline.


MC
Construct World-Coordinate Scene Using Modeling-Coordinate Transformations
WC
Convert World-Coordinate to Viewing Coordinates
VC
NVC
DC

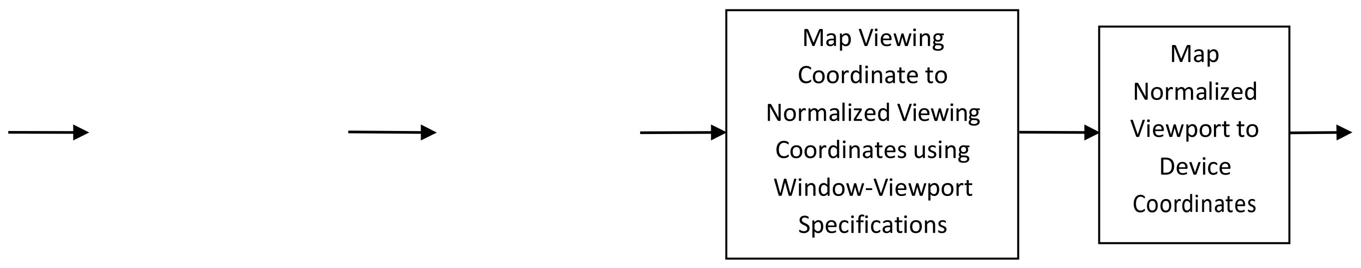

Map Viewing Coordinate to Normalized Viewing Coordinates using Window-Viewport Specifications
Map Normalized Viewport to Device Coordinates

**Fig. 3.2: - 2D viewing pipeline**.

- As shown in figure above first of all we construct world coordinate scene using modeling coordinate transformation.
- After this we convert viewing coordinates from world coordinates using window to viewport transformation.
- Then we map viewing coordinate to normalized viewing coordinate in which we obtain values in between 0 to 1.
- At last we convert normalized viewing coordinate to device coordinate using device driver software which provide device specification.
- Finally device coordinate is used to display image on display screen.
- By changing the viewport position on screen we can see image at different place on the screen.
- By changing the size of the window and viewport we can obtain zoom in and zoom out effect as per requirement.
- Fixed size viewport and small size window gives zoom in effect, and fixed size viewport and larger window gives zoom out effect.
- View ports are generally defines with the unit square so that graphics package are more device independent which we call as normalized viewing coordinate.

## Viewing Coordinate Reference Frame

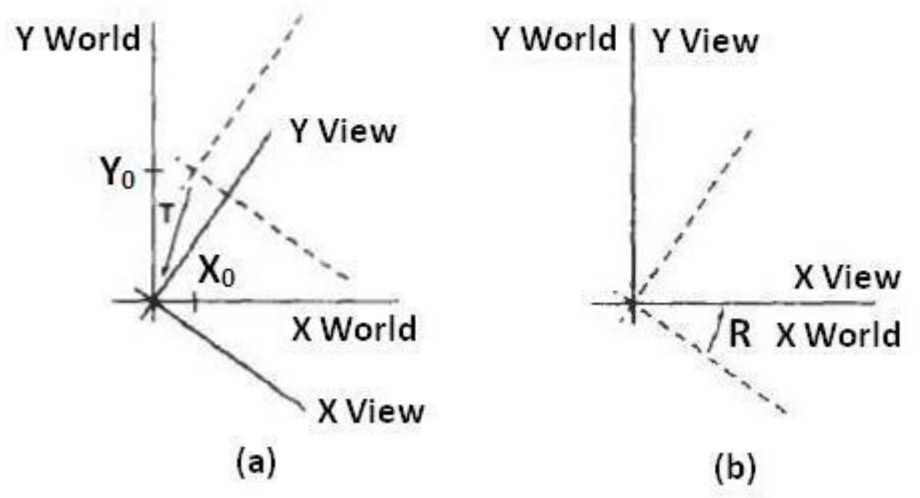


Fig. 3.3: - A viewing-coordinate frame is moved into coincidence with the world frame in two steps: (a) translate the viewing origin to the world origin, and then (b) rotate to align the axes of the two systems.

- We can obtain reference frame in any direction and at any position.
- For handling such condition first of all we translate reference frame origin to standard reference frame origin and then we rotate it to align it to standard axis.
- In this way we can adjust window in any reference frame.
- this is illustrate by following transformation matrix:

  $\mathbf{M}_{\mathbf{wc,vc}} = \mathbf{RT}$
- Where T is translation matrix and R is rotation matrix.

## Window-To-Viewport Coordinate Transformation

- Mapping of window coordinate to viewport is called window to viewport transformation.
- We do this using transformation that maintains relative position of window coordinate into viewport.
- That means center coordinates in window must be remains at center position in viewport.
- We find relative position by equation as follow:

$$\frac{x_v - x_{vmin}}{x_{vmax} - x_{vmin}} = \frac{x_w - x_{wmin}}{x_{wmax} - x_{wmin}}$$

$$\frac{y_v - y_{vmin}}{y_{vmax} - y_{vmin}} = \frac{y_w - y_{wmin}}{y_{wmax} - y_{wmin}}$$

- Solving by making viewport position as subject we obtain:

$$x_v = x_{vmin} + (x_w - x_{wmin})s_x$$
$$y_v = y_{vmin} + (y_w - y_{wmin})s_y$$

- Where scaling factor are :

$$s_x = \frac{x_{vmax} - x_{vmin}}{x_{wmax} - x_{wmin}}$$
$$s_y = \frac{y_{vmax} - y_{vmin}}{y_{wmax} - y_{wmin}}$$

- We can also map window to viewport with the set of transformation, which include following sequence of transformations:
  1. Perform a scaling transformation using a fixed-point position of ($x_{Wmin}$,$y_{wmin}$) that scales the window area to the size of the viewport.
  2. Translate the scaled window area to the position of the viewport.
- For maintaining relative proportions we take ($s_x = s_y$). in case if both are not equal then we get stretched or contracted in either the x or y direction when displayed on the output device.
- Characters are handle in two different way one way is simply maintain relative position like other primitive and other is to maintain standard character size even though viewport size is enlarged or reduce.
- Number of display device can be used in application and for each we can use different window-to-viewport transformation. This mapping is called the **workstation transformation.**

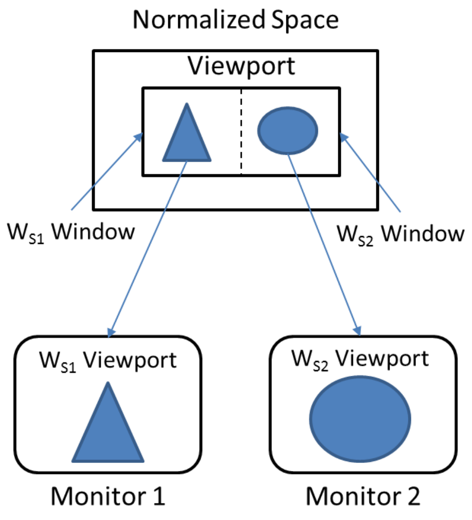


Fig. 3.4: - workstation transformation.

- As shown in figure two different displays devices are used and we map different window-to-viewport on each one.

## Clipping Operations

- Generally, any procedure that identifies those portions of a picture that are either inside or outside of a specified region of space is referred to as a **clipping algorithm**, or simply **clipping**. The region against which an object is to clip is called a **clip window.**
- Clip window can be general polygon or it can be curved boundary.

### Application of Clipping

- It can be used for displaying particular part of the picture on display screen.
- Identifying visible surface in 3D views.
- Antialiasing.
- Creating objects using solid-modeling procedures.
- Displaying multiple windows on same screen.
- Drawing and painting.

## Point Clipping

- In point clipping we eliminate those points which are outside the clipping window and draw points which are inside the clipping window.
- Here we consider clipping window is rectangular boundary with edge ($x_{wmin}$,$x_{wmax}$,$y_{wmin}$,$y_{wmax}$).
- So for finding wether given point is inside or outside the clipping window we use following inequality:
  $x_{wmin} \leq x \leq x_{wamx}$
  $y_{wmin} \leq y \leq y_{wamx}$
- If above both inequality is satisfied then the point is inside otherwise the point is outside the clipping window.

## Line Clipping

- Line clipping involves several possible cases.
  1. Completely inside the clipping window.
  2. Completely outside the clipping window.
  3. Partially inside and partially outside the clipping window.

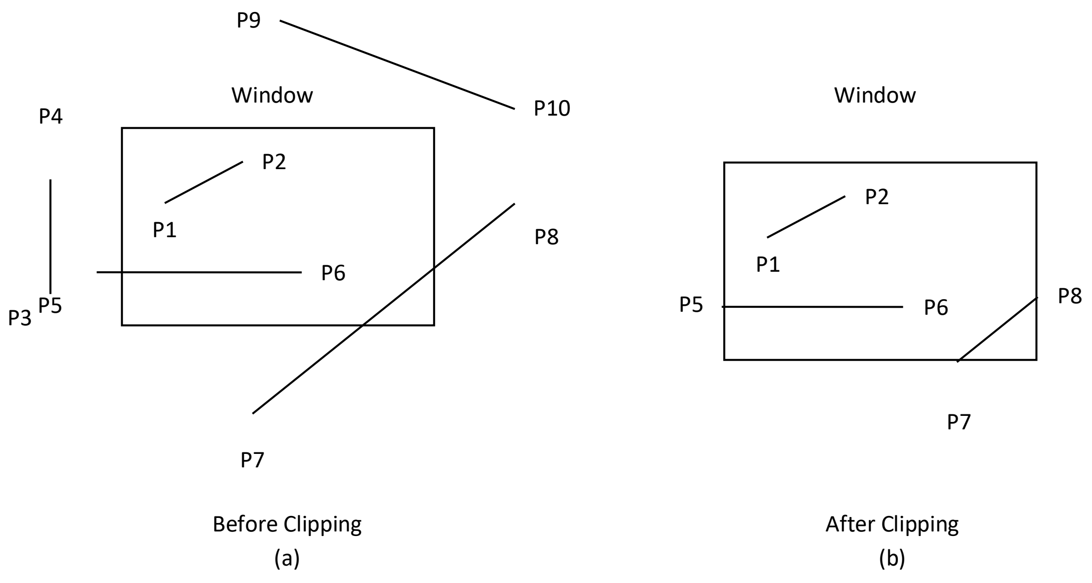


Fig. 3.5: - Line clipping against a rectangular window.

- Line which is completely inside is display completely. Line which is completely outside is eliminated from display. And for partially inside line we need to calculate intersection with window boundary and find which part is inside the clipping boundary and which part is eliminated.
- For line clipping several scientists tried different methods to solve this clipping procedure. Some of them are discuss below.

## Cohen-Sutherland Line Clipping

- This is one of the oldest and most popular line-clipping procedures.

### Region and Region Code

- In this we divide whole space into nine region and assign 4 bit code to each endpoint of line depending on the position where the line endpoint is located.

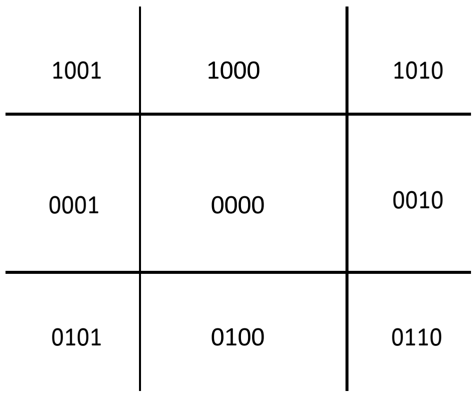


Fig. 3.6: - Workstation transformation.

- Figure 3.6 shows code for line end point which is fall within particular area.
- Code is deriving by setting particular bit according to position of area.
  Set bit 1: For left side of clipping window.
  Set bit 2: For right side of clipping window.
  Set bit 3: For below clipping window.
  Set bit 4: For above clipping window.
- All bits as mention above are set means 1 and other are 0.

### Algorithm

**Step-1:**

Assign region code to both endpoint of a line depending on the position where the line endpoint is located.

**Step-2:**

```
If both endpoint have code ‘0000’
        Then line is completely inside.
Otherwise
        Perform logical ending between this two codes.

        If result of logical ending is non-zero
                Line is completely outside the clipping window.
        Otherwise
                Calculate the intersection point with the boundary one by one.
                Divide the line into two parts from intersection point.
                Recursively call algorithm for both line segments.
```

**Step-3:**
Draw line segment which are completely inside and eliminate other line segment which found completely outside.

## Intersection points calculation with clipping window boundary

- For intersection calculation we use line equation "$y = mx + b$".
- '$x$' is constant for left and right boundary which is:
    - for left "$x = x_{wmin}$"
    - for right "$x = x_{wmax}$"
- So we calculate $y$ coordinate of intersection for this boundary by putting values of $x$ depending on boundary is left or right in below equation.
  $\boldsymbol{y = y_1 + m(x - x_1)}$
- '$y$' coordinate is constant for top and bottom boundary which is:
    - for top "$y = y_{wmax}$"
    - for bottom "$y = y_{wmin}$"
- So we calculate $x$ coordinate of intersection for this boundary by putting values of $y$ depending on boundary is top or bottom in below equation.
  $$\boldsymbol{x = x_1 + \frac{y - y_1}{m}}$$

## Liang-Barsky Line Clipping

- Line clipping approach is given by the Liang and Barsky is faster than cohen-sutherland line clipping. Which is based on analysis of the parametric equation of the line which are as below.
  $\boldsymbol{x = x_1 + u\Delta x}$
  $\boldsymbol{y = y_1 + u\Delta y}$
  Where $0 \leq u \leq 1$ ,$\Delta x = x_2 - x_1$ and $\Delta y = y_2 - y_1$.

### Algorithm

```
1. Read two end points of line $P_1(x_1, y_1)$ and $P_2(x_2,y_2)$
2. Read two corner vertices, left top and right bottom of window: $(x_{wmin}, y_{wmax})$ and $(x_{wmax}, y_{wmin})$
3. Calculate values of parameters $p_k$ and $q_k$ for $k = 1, 2, 3, 4$ such that,
   $p_1 = -\Delta x,$      $q_1 = x_1 - x_{wmin}$
   $p_2 = \Delta x,$       $q_2 = x_{wmax} - x_1$
   $p_3 = -\Delta y,$      $q_3 = y_1 - y_{wmin}$
   $p_4 = \Delta y,$       $q_4 = y_{wmax} - y_1$
4. If $p_k = 0$ for any value of $k = 1, 2, 3, 4$ then,
        Line is parallel to $k^{th}$ boundary.

        If corresponding $q_k < 0$ then,
            Line is completely outside the boundary. Therefore, discard line segment and Go to Step 10.
        Otherwise
            Check line is horizontal or vertical and accordingly check line end points with corresponding boundaries.

            If line endpoints lie within the bounded area
                Then use them to draw line.
            Otherwise

                Use boundary coordinates to draw line. And go to Step 8.
5. For $k = 1,2,3,4$ calculate $r_k$ for nonzero values of $p_k$ and $q_k$ as follows:
   $r_k = \frac{q_k}{p_k}, for\ k = 1,2,3,4$
6. Find $u_1\ and\ u_2$ as given below:
   $u_1 = \max\{0, r_k | where\ k\ takes\ all\ values\ for\ which\ p_k < 0\}$
   $u_2 = \min\{1, r_k | where\ k\ takes\ all\ values\ for\ which\ p_k > 0\}$
7. If $u_1 \leq u_2$ then
        Calculate endpoints of clipped line:
        $x_1' = x_1 + u_1\Delta x$
        $y_1' = y_1 + u_1\Delta y$
        $x_2' = x_1 + u_2\Delta x$
        $y_2' = y_1 + u_2\Delta y$
        Draw line $(x_1', y_1', x_2', y_2')$;
8. Stop.
```

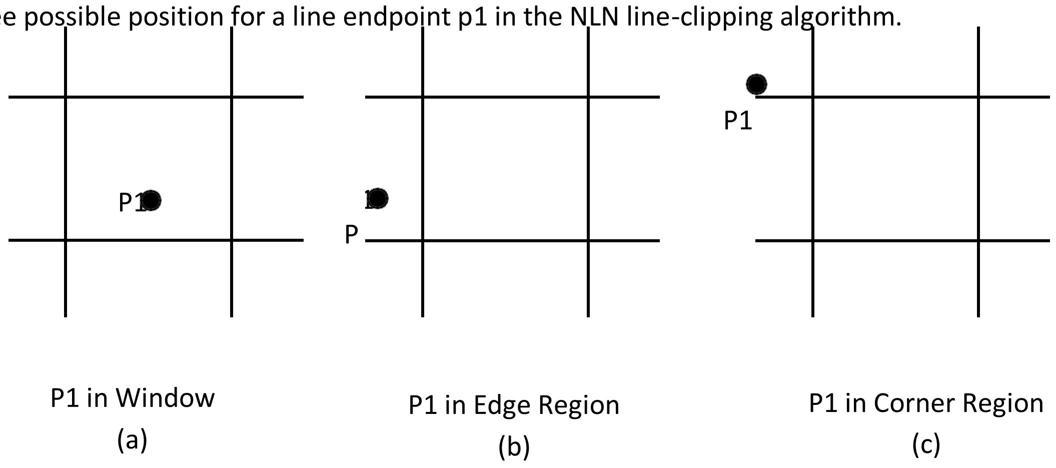
### Advantages

1. More efficient.
2. Only requires one division to update $u_1$ and $u_2$.
3. Window intersections of line are calculated just once.

## Nicholl-Lee-Nicholl Line Clipping

- By creating more regions around the clip window the NLN algorithm avoids multiple clipping of an individual line segment.
- In Cohen-Sutherlan line clipping sometimes multiple calculation of intersection point of a line is done before actual window boundary intersection or line is completely rejected.
- These multiple intersection calculation is avoided in NLN line clipping procedure.
- NLN line clipping perform the fewer comparisons and divisions so it is more efficient.
- But NLN line clipping cannot be extended for three dimensions while Cohen-Sutherland and Liang-Barsky algorithm can be easily extended for three dimensions.
- For given line we find first point falls in which region out of nine region shown in figure below but three region shown in figure by putting point are only considered and if point falls in other region than we transfer that point in one of the three region.

Fig. 3.7: - Three possible position for a line endpoint p1 in the NLN line-clipping algorithm.



- We can also extend this procedure for all nine regions.
- Now for p1 is inside the window we divide whole area in following region:

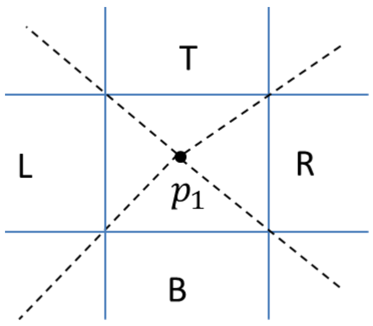


Fig. 3.8: - Clipping region when p1 is inside the window.

- Now for p1 is in edge region we divide whole area in following region:

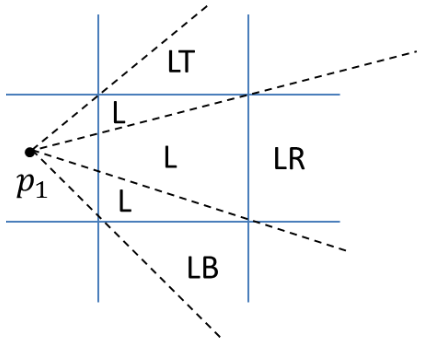


Fig. 3.9: - Clipping region when p1 is in edge region.

- Now for p1 is in corner region we divide whole area in following region:

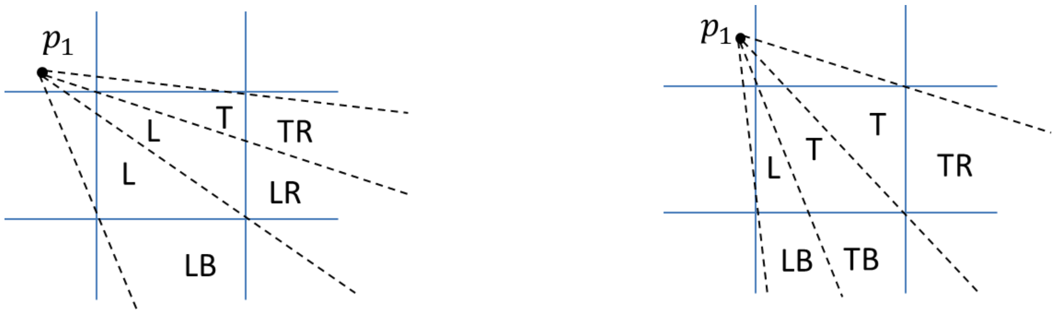


Fig. 3.10: - Two possible sets of clipping region when p1 is in corner region.

- Regions are name in such a way that name in which region p2 falls is gives the window edge which intersects the line.
- For example region LT says that line need to clip at left and top boundary.
- For finding that in which region line $p_1p_2$ falls we compare the slope of the line to the slope of the boundaries:

  $slope\ \overline{p_1p_{B1}} < slope\ \overline{p_1p_2} < slope\ \overline{p_1p_{B2}}$

  Where $\overline{p_1p_{B1}}$ and $\overline{p_1p_{B2}}$ are boundary lines.
- For example p1 is in edge region and for checking whether p2 is in region LT we use following equation.

  $slope\ \overline{p_1p_{TR}} < slope\ \overline{p_1p_2} < slope\ \overline{p_1p_{TL}}$

  $$\frac{y_T - y_1}{x_R - x_1} < \frac{y_2 - y_1}{x_2 - x_1} < \frac{y_T - y_1}{x_L - x_1}$$
- After checking slope condition we need to check weather it crossing zero, one or two edges.
- This can be done by comparing coordinates of $p_2$ with coordinates of window boundary.
- For left and right boundary we compare $x$ coordinates and for top and bottom boundary we compare $y$ coordinates.
- If line is not fall in any defined region than clip entire line.

- Otherwise calculate intersection.
- After finding region we calculate intersection point using parametric equation which are:
- $\boldsymbol{x = x_1 + (x_2 - x_1)u}$
- $\boldsymbol{y = y_1 + (y_2 - y_1)u}$
- For left or right boundary $x = x_l$ *or* $x_r$ respectively, with $u = (x_{l/r} - x1)/\ (x2 - x1)$, so that $y$ can be obtain from parametric equation as below:
- $\boldsymbol{y = y_1 + \frac{y2-y1}{x_2 - x_1}(x_L - x_1)}$
- Keep the portion which is inside and clip the rest.

## Polygon Clipping

- For polygon clipping we need to modify the line clipping procedure because in line clipping we need to consider about only line segment while in polygon clipping we need to consider the area and the new boundary of the polygon after clipping.

### Sutherland-Hodgeman Polygon Clipping

- For correctly clip a polygon we process the polygon boundary as a whole against each window edge.
- This is done by whole polygon vertices against each clip rectangle boundary one by one.
- Beginning with the initial set of polygon vertices we first clip against the left boundary and produce new sequence of vertices.
- Then that new set of vertices is clipped against the right boundary clipper, a bottom boundary clipper and a top boundary clipper, as shown in figure below.

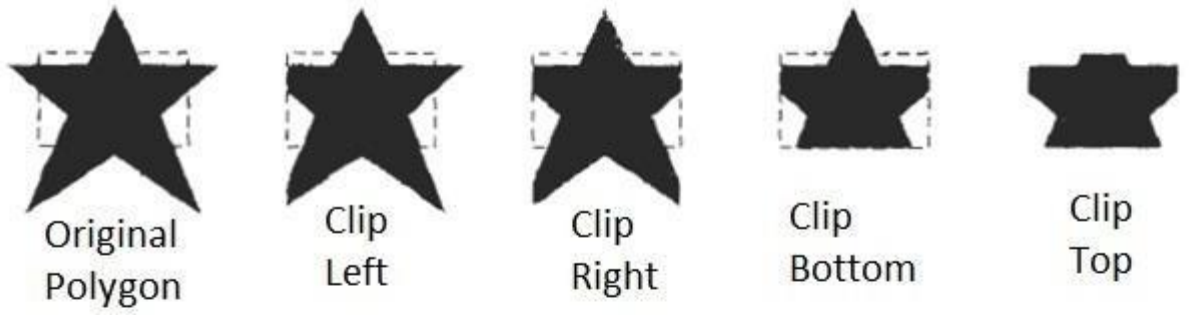
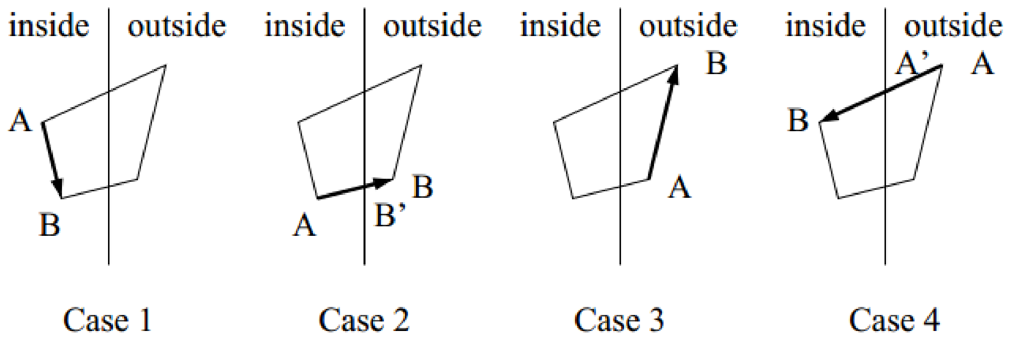


Fig. 3.11: - Clipping a polygon against successive window boundaries.

$$\begin{tikzcd}
\text{in} \arrow[r] & \boxed{\text{Left Clipper}} \arrow[r] & \boxed{\text{Right Clipper}} \arrow[r] & \boxed{\text{Bottom Clipper}} \arrow[r] & \boxed{\text{Top Clipper}} \arrow[r] & \text{out}
\end{tikzcd}$$

Fig. 3.12: - Processing the vertices of the polygon through boundary clipper.



- There are four possible cases when processing vertices in sequence around the perimeter of a polygon.

Fig. 3.13: - Clipping a polygon against successive window boundaries.

- As shown in case 1: if both vertices are inside the window we add only second vertices to output list.
- In case 2: if first vertices is inside the boundary and second vertices is outside the boundary only the edge intersection with the window boundary is added to the output vertex list.
- In case 3: if both vertices are outside the window boundary nothing is added to window boundary.
- In case 4: first vertex is outside and second vertex is inside the boundary, then adds both intersection point with window boundary, and second vertex to the output list.
- When polygon clipping is done against one boundary then we clip against next window boundary.
- We illustrate this method by simple example.

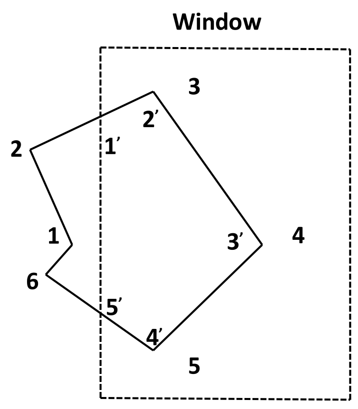


Fig. 3.14: - Clipping a polygon against left window boundaries.

- As shown in figure above we clip against left boundary vertices 1 and 2 are found to be on the outside of the boundary. Then we move to vertex 3, which is inside, we calculate the intersection and add both intersection point and vertex 3 to output list.
- Then we move to vertex 4 in which vertex 3 and 4 both are inside so we add vertex 4 to output list, similarly from 4 to 5 we add 5 to output list, then from 5 to 6 we move inside to outside so we add intersection pint to output list and finally 6 to 1 both vertex are outside the window so we does not add anything.
- Convex polygons are correctly clipped by the Sutherland-Hodgeman algorithm but concave polygons may be displayed with extraneous lines.
- For overcome this problem we have one possible solution is to divide polygon into numbers of small convex polygon and then process one by one.
- Another approach is to use Weiler-Atherton algorithm.

## Weiler-Atherton Polygon Clipping

- In this algorithm vertex processing procedure for window boundary is modified so that concave polygon also clip correctly.
- This can be applied for arbitrary polygon clipping regions as it is developed for visible surface identification.
- Main idea of this algorithm is instead of always proceeding around the polygon edges as vertices are processed we sometimes need to follow the window boundaries.
- Other procedure is similar to Sutherland-Hodgeman algorithm.
- For clockwise processing of polygon vertices we use the following rules:
    - For an outside to inside pair of vertices, follow the polygon boundary.
    - For an inside to outside pair of vertices, follow the window boundary in a clockwise direction.
- We illustrate it with example:

- As shown in figure we start from v1 and move clockwise towards v2 and add intersection point and next point to output list by following polygon boundary, then from v2 to v3 we add v3 to output list.
- From v3 to v4 we calculate intersection point and add to output list and follow window boundary.
- Similarly from v4 to v5 we add intersection point and next point and follow the polygon boundary, next we move v5 to v6 and add intersection point and follow the window boundary, and finally v6 to v1 is outside so no need to add anything.
- This way we get two separate polygon section after clipping.