Viewing Pipeline & Clipping

Khushbu Maurya

The Viewing Pipeline

- 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.
- Let’s see steps involved in viewing pipeline.

Clipping

- 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

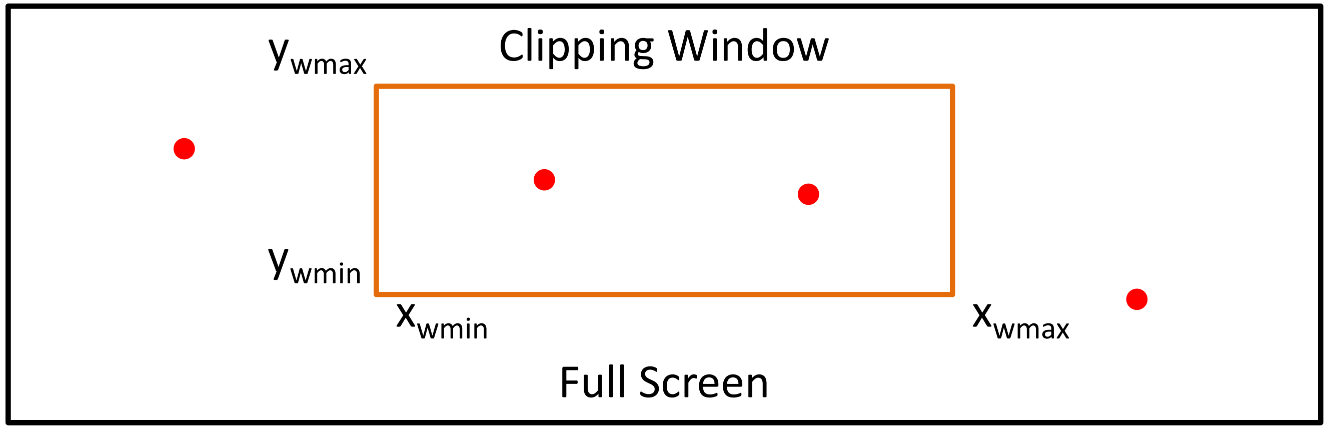
- It can be used for displaying particular part of the picture on display screen.
- Identifying visible surface in 3D views.
- Antialiasing.
- Creating objects using solid-modeling procedures.
- Displaying multiple windows on same screen.
- Drawing and painting.

Point Clipping

- In point clipping we eliminate outside points and draw points which are inside the clipping window.
- Here we consider clipping window is rectangular boundary with edge $(x_{wmin}, xw_{max}, y_{wmin}, yw_{max})$.
- So for finding weather given point is inside or outside the clipping window we use following inequality:

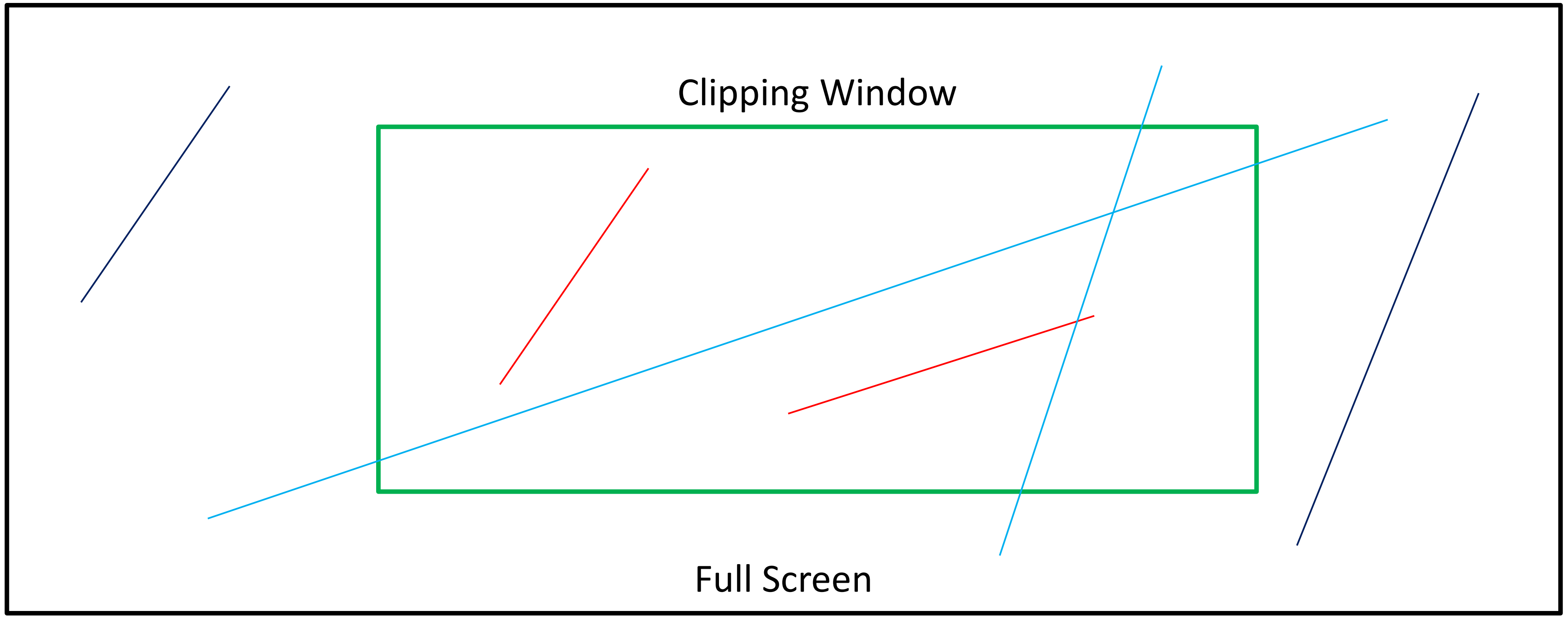
$$\boldsymbol{x_{wmin} \leq x \leq x_{wmax}}, \qquad \boldsymbol{y_{wmin} \leq y \leq y_{wmax}}$$

- 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.

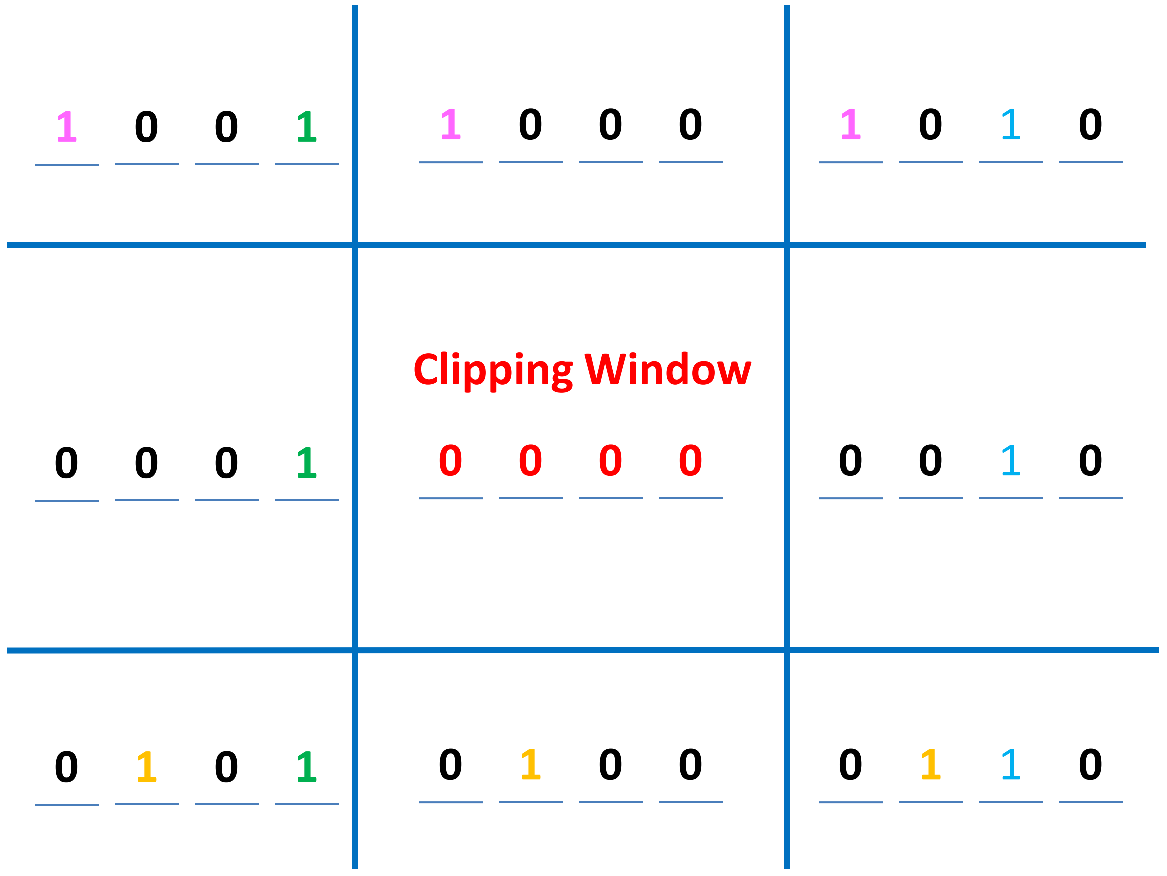


Line Clipping

- For line clipping several scientists tried different methods to solve this clipping procedure.
- Some of them we will discuss. Which are:
  1. Cohen-Sutherland Line Clipping
  2. Liang-Barsky Line Clipping
  3. Nicholl-Lee-Nicholl Line Clipping

Region Code in Cohen-Sutherland Line Clipping

- This is one of the oldest and most popular line-clipping procedures.
- In this we divide whole space into nine region and give 4 bit region code.(A B R L)
- Code is deriving by:
  - 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.



Put all other bits zero (Reset remaining bit)

Steps Cohen-Sutherland Line Clipping

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.(line is not visible)

Otherwise

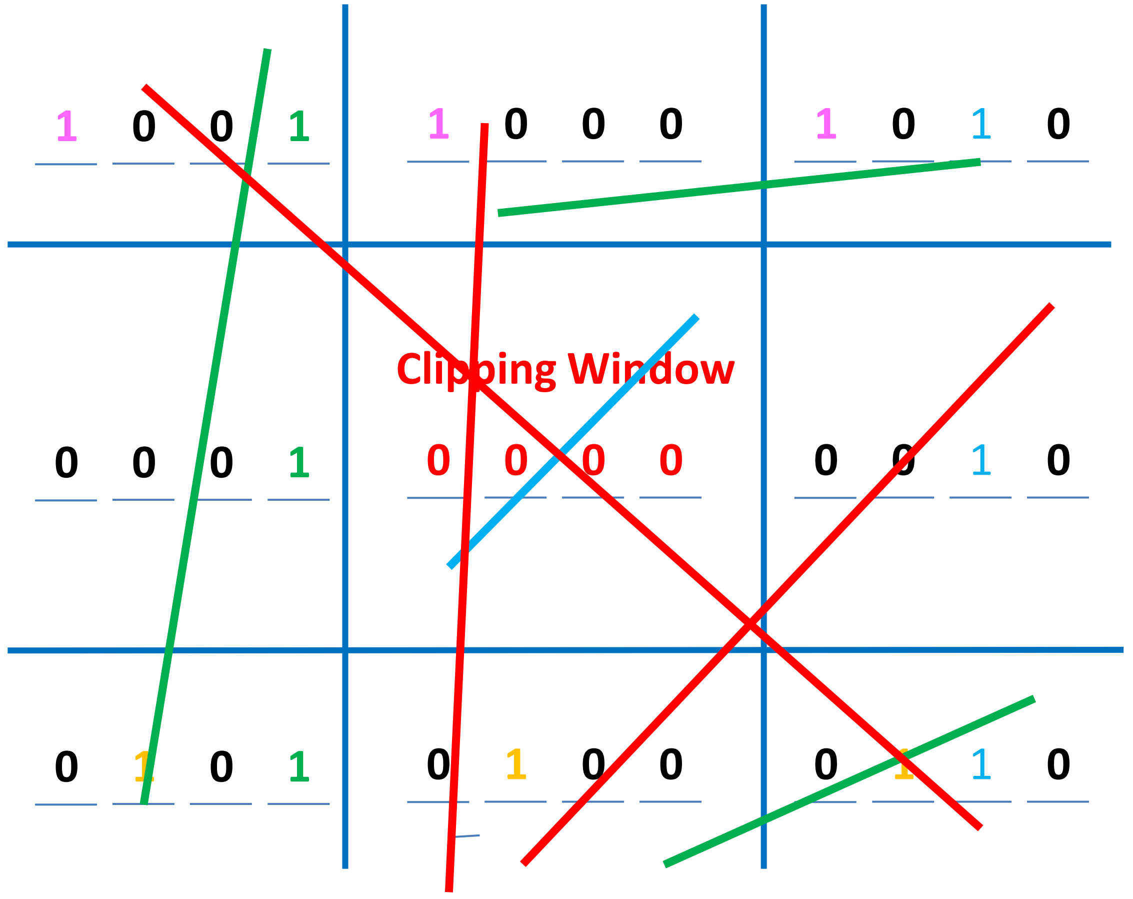
(line is partially visible or not visible)

Calculate the intersection point with the boundary one by one.

Divide the line into two parts from intersection point.

Repeat from Step-1 for both line segments.

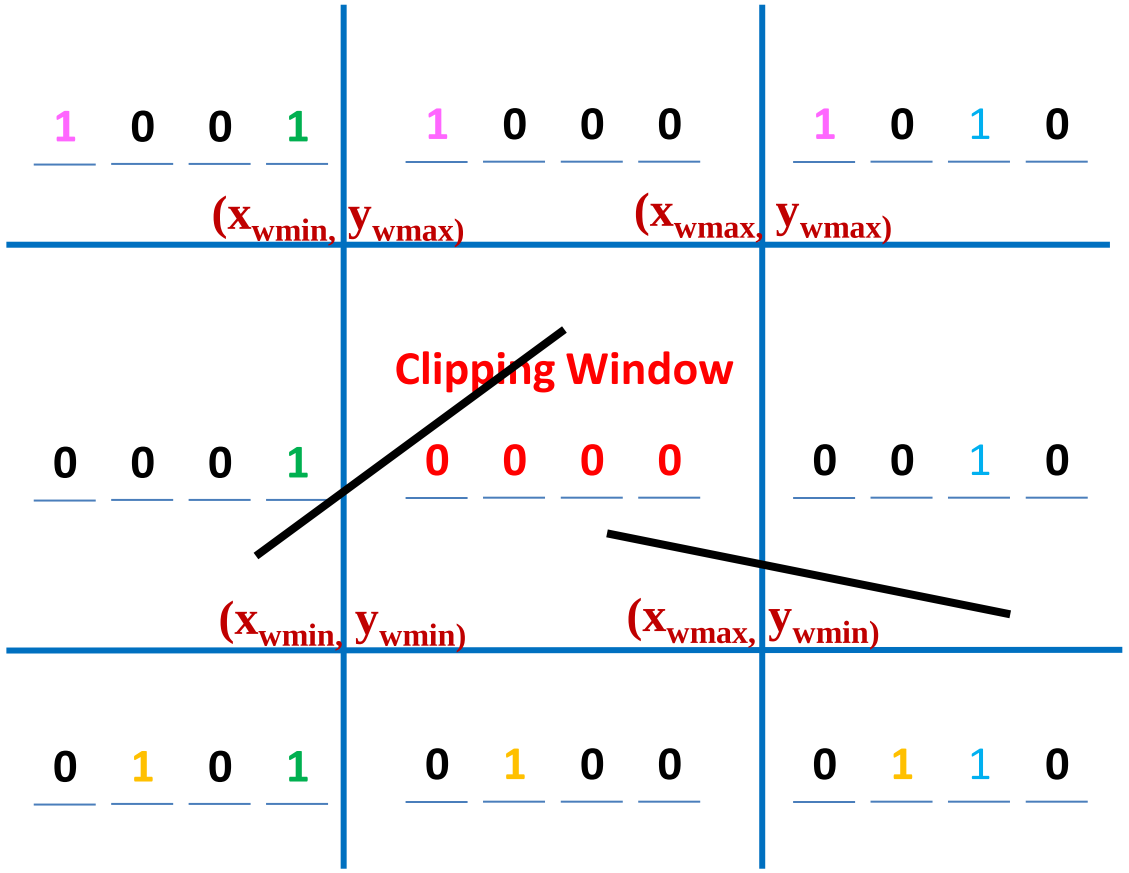
Step-3: Draw line segment which are completely inside and eliminate other line segment which found completely outside.



Intersection points- Cohen-Sutherland Algorithm

- 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.

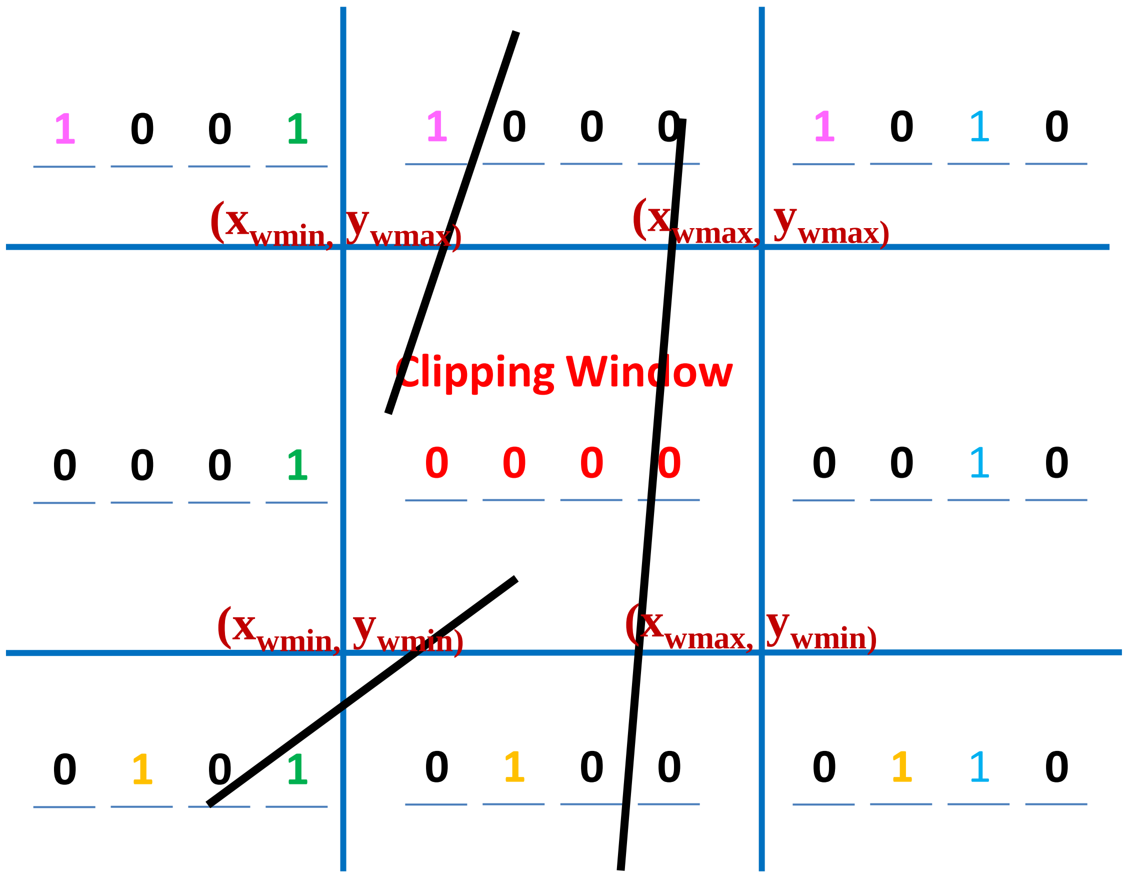
$$\boldsymbol{y = y_1 + m(x - x_1)}$$



Contd.

- "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.

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



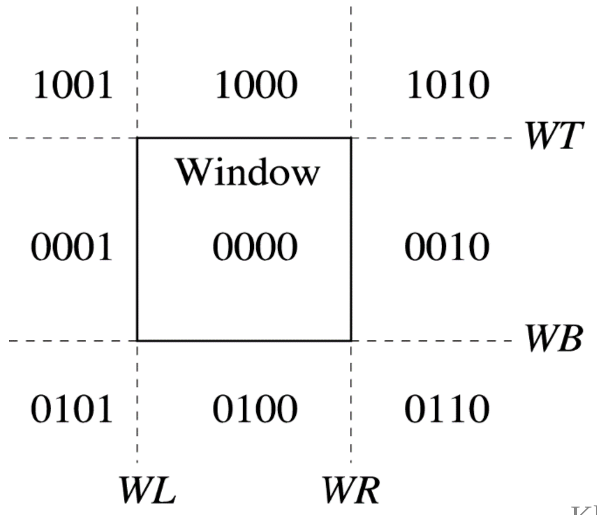
Cohen-Sutherland

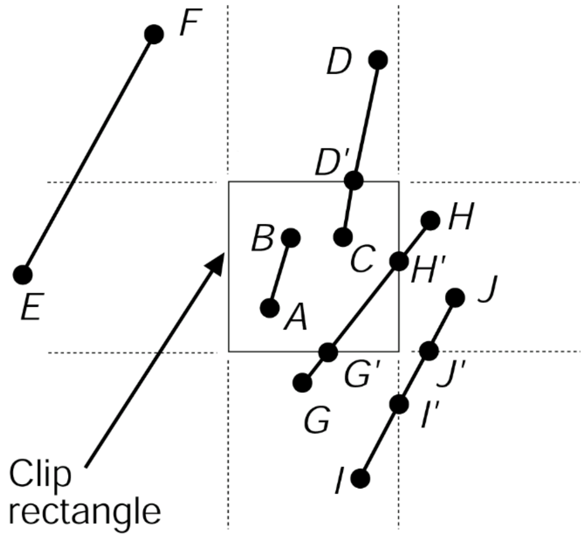
- Line is completely visible iff both code values of endpoints are *0*, i.e.

$$C_0 \vee C_1 = 0$$

- If line segments are completely outside the window, then

$$C_0 \wedge C_1 \neq 0$$





Pics/Math courtesy of Dave Mount @ UMD-CP

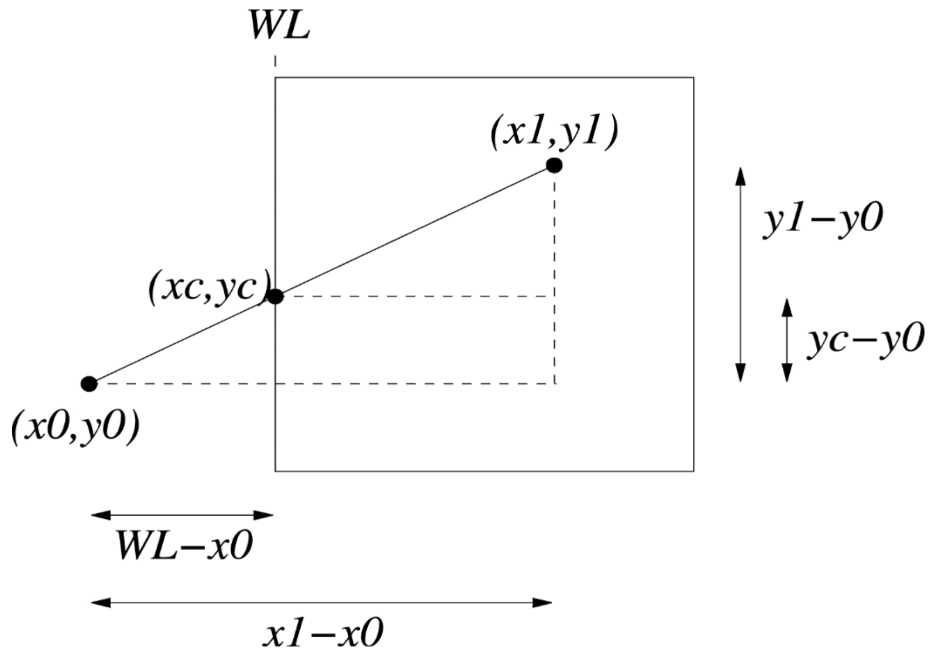
Cohen-Sutherland

- Clearly: $x_c = WL$
- Using *similar* triangles

$$\frac{y_c - y_0}{y_1 - y_0} = \frac{WL - x_0}{x_1 - x_0}$$

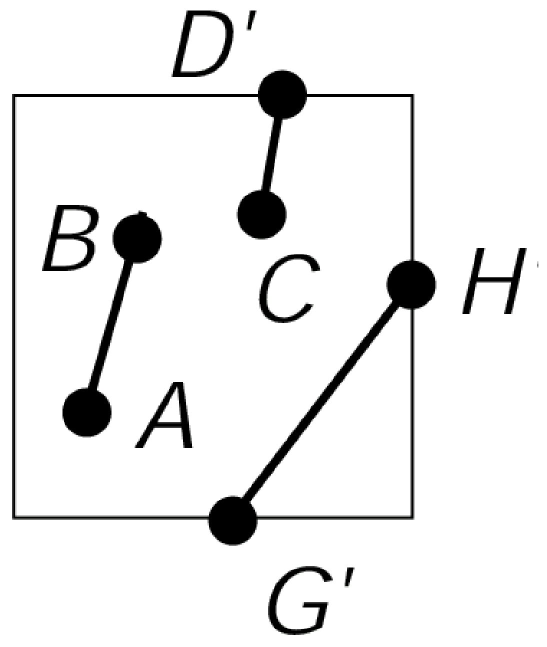
- Solving for y_c gives

$$y_c = \frac{WL - x_0}{x_1 - x_0}(y_1 - y_0) + y_0$$



Pics/Math courtesy of Dave Mount @ UMD-CP

Cohen-Sutherland

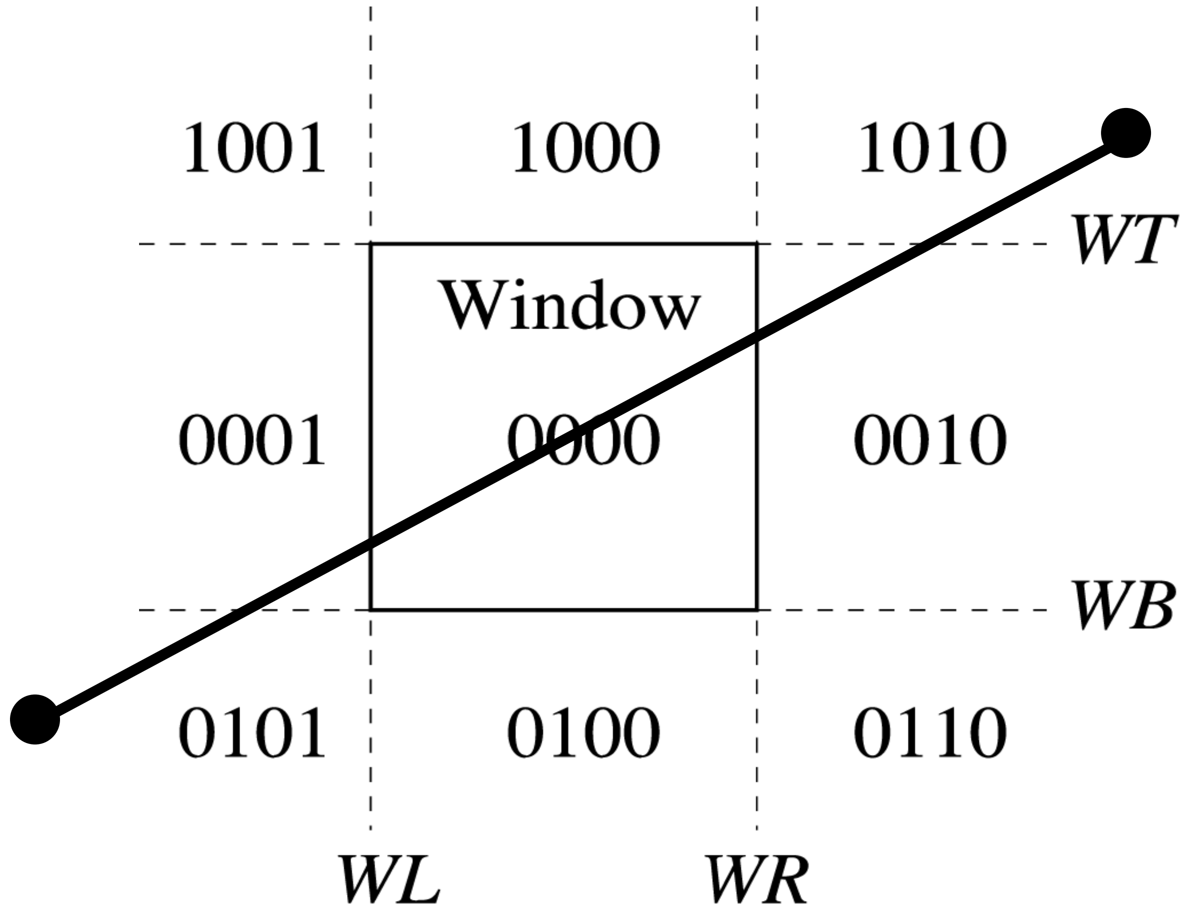


- Replace (x_0, y_0) with (x_c, y_c)
- Re-compute codes
- Continue until all bit flips (clip lines) are processed, i.e. all points are inside the clip window

Pics/Math courtesy of Dave Mount @ UMD-CP

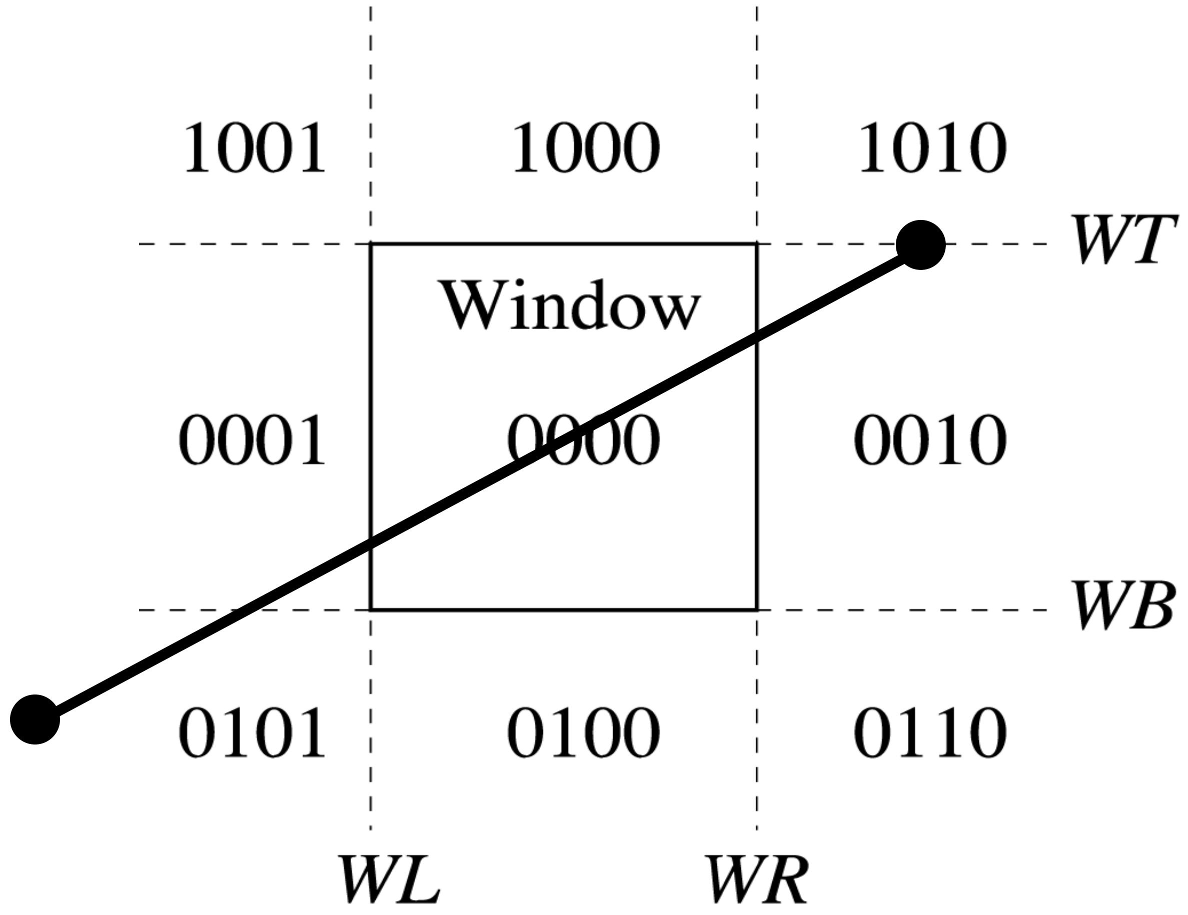
Cohen Sutherland

0101
1010



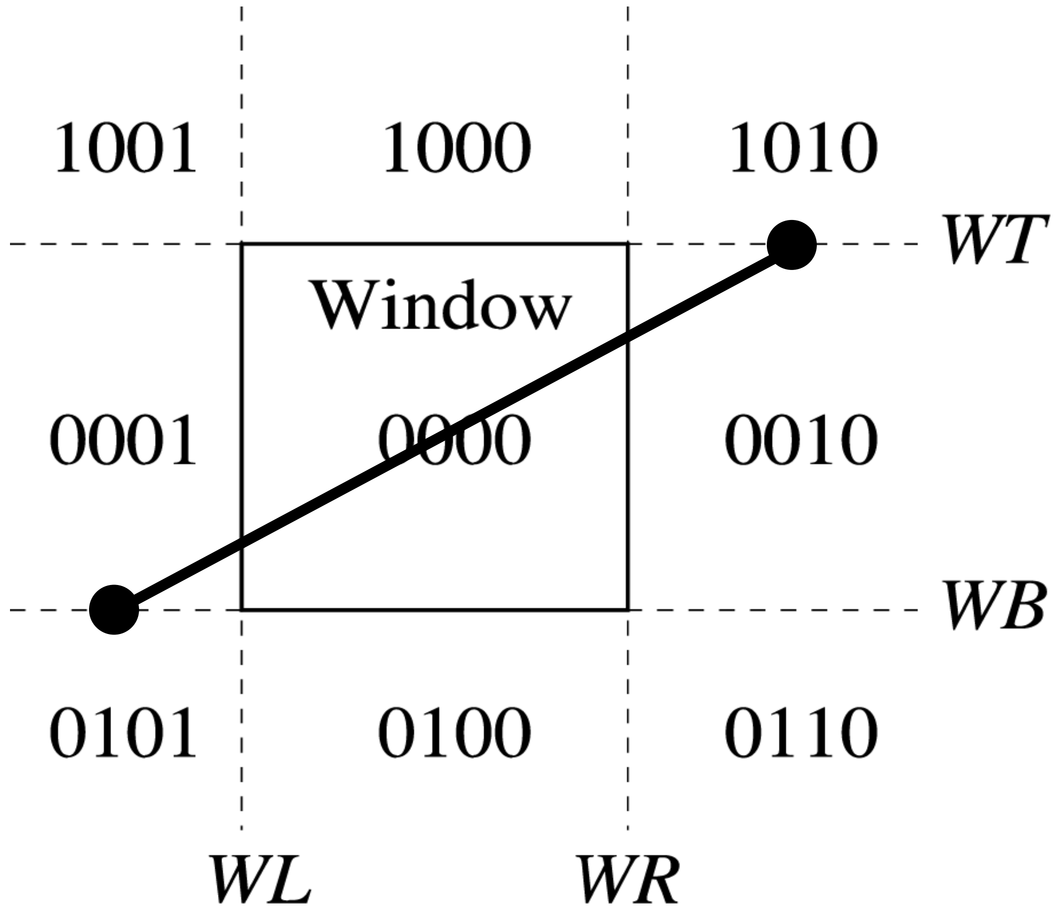
Cohen Sutherland

0101
0010



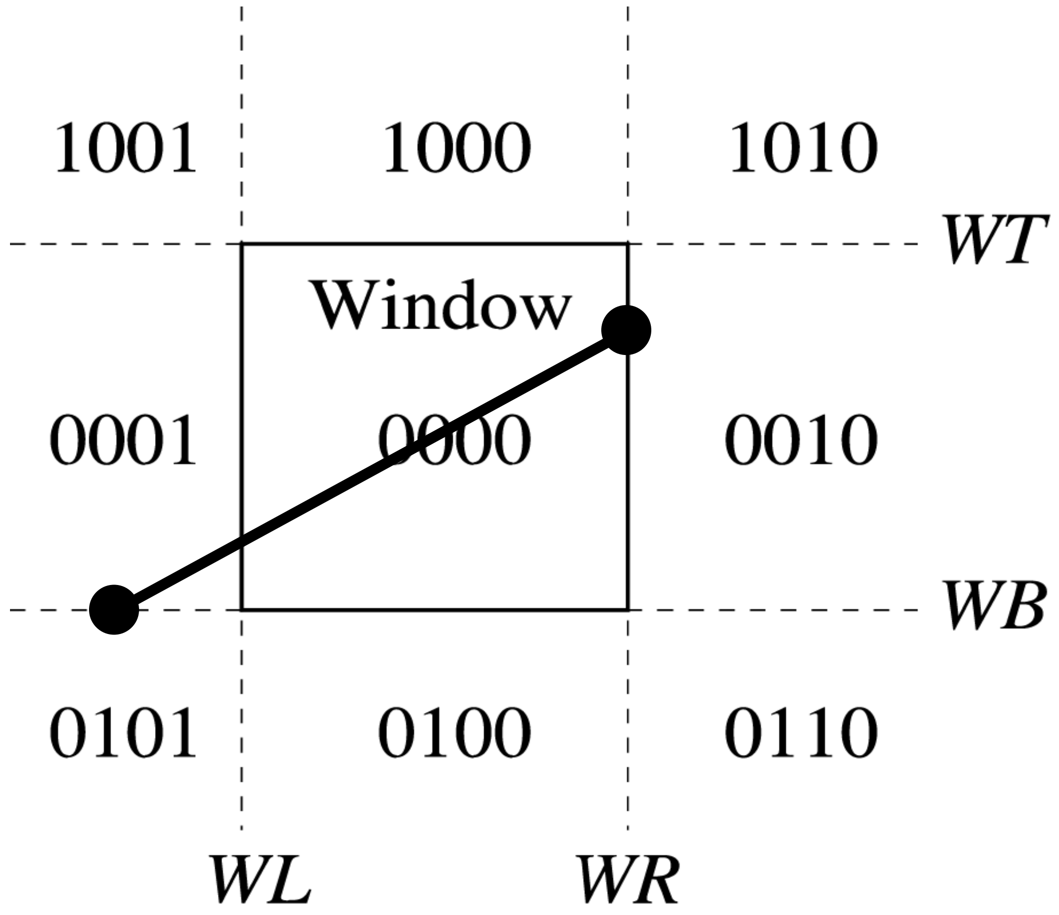
Cohen Sutherland

0001
0010



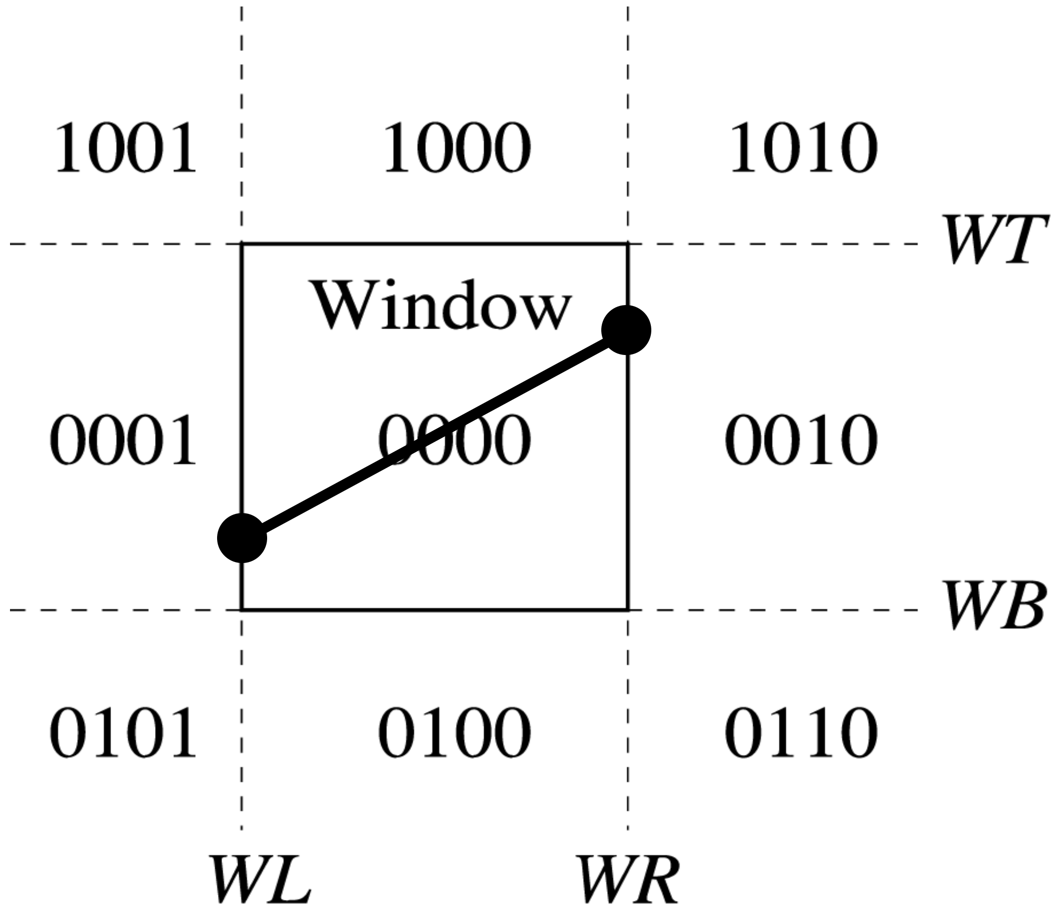
Cohen Sutherland

0001
0000



Cohen Sutherland

0000
0000



Example

P1= (0, 50)

P2= (500, 450)

Xmin = 200, Xmax= 600, Ymin= 160, Ymax= 400

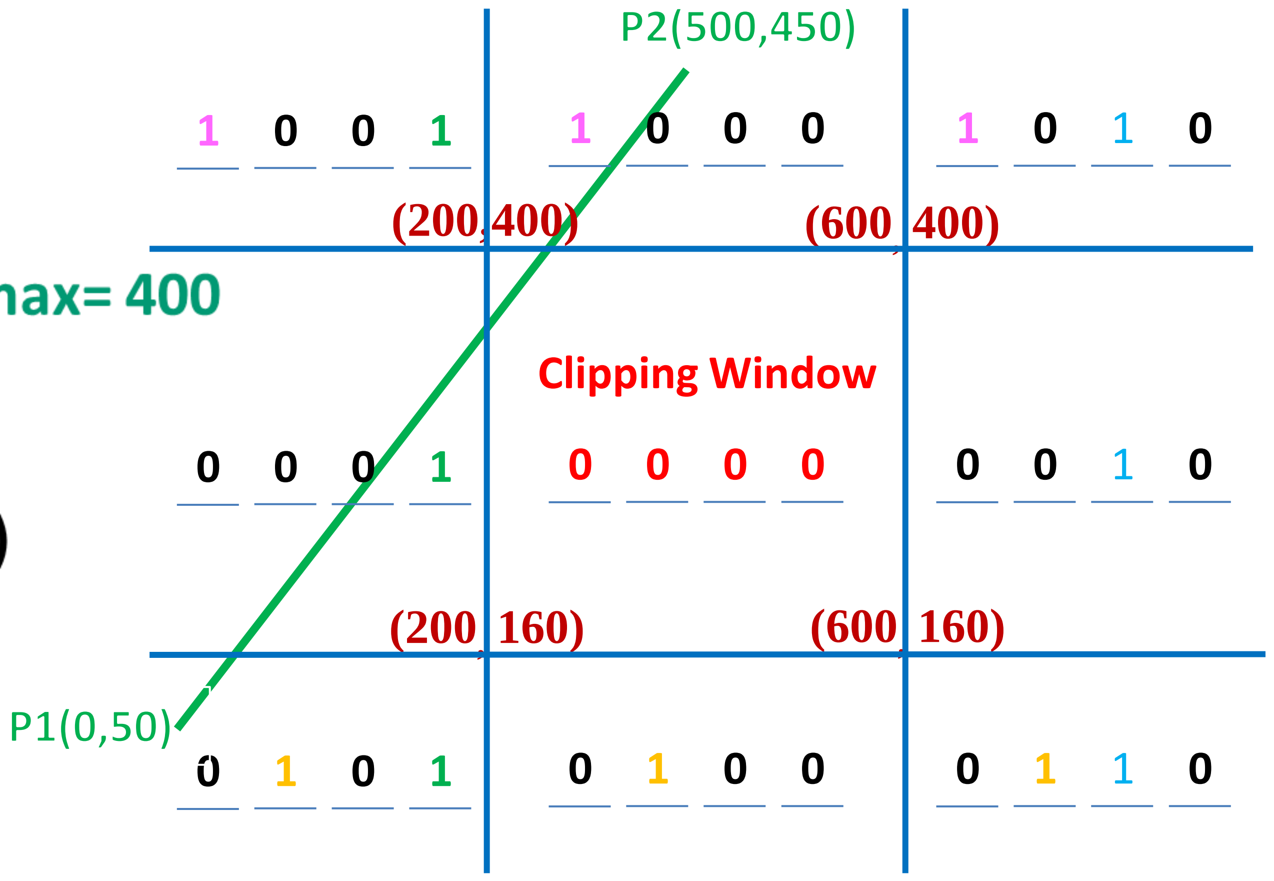
M = dy/dx = 400/500=$\frac{4}{5}$

P1 code → top bottom right left (0101)

P2 code → (1000)

P1 ^ P2==0000

No trivial accept, No trivial reject

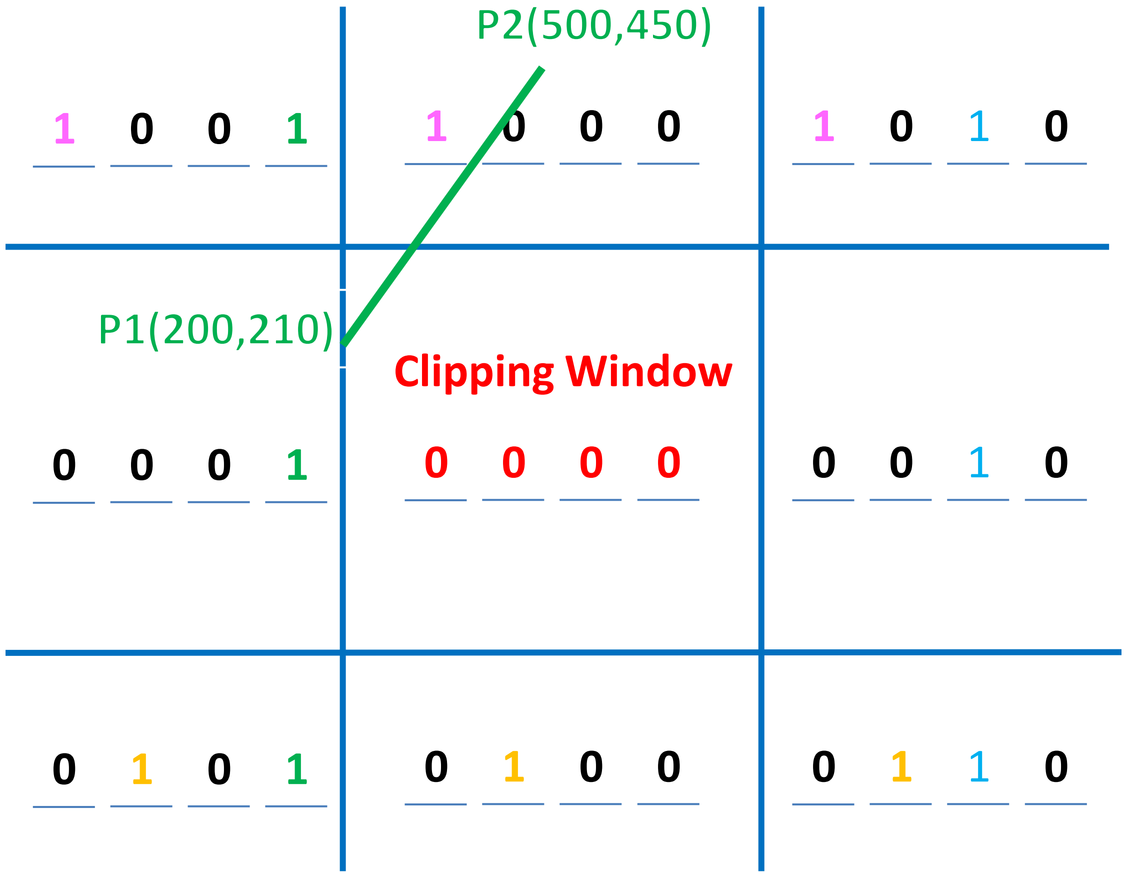


Example (cont.)

- **P1**

$$\boxed{\begin{gathered}X=xmin=200\\ Y= y1+ m\,(x-x1)\\ Y=50+\frac{4}{5}(200-0)=210\end{gathered}}$$

- **P1= (200,210) P2= (500, 450)**
- **P1 code → 0000**
- **P2 code → (1000)**
- **No trivial accept, No trivial reject**

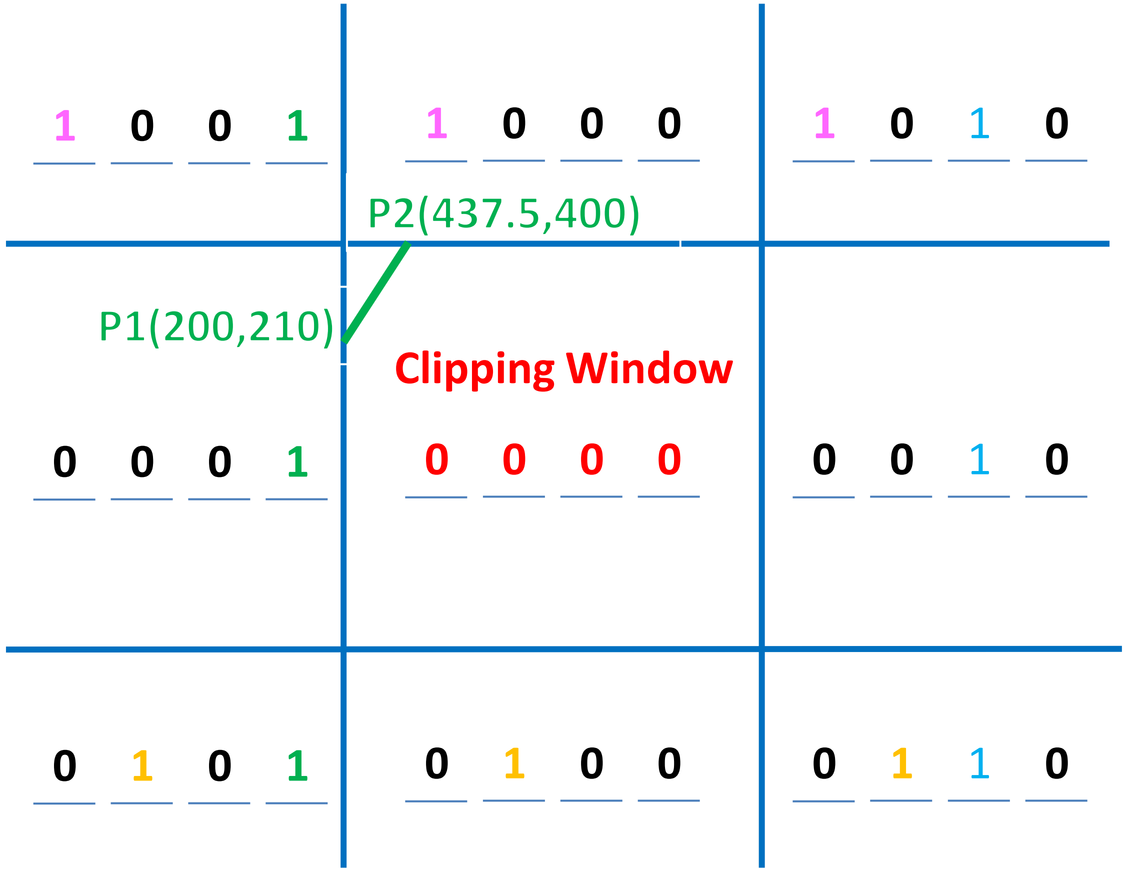


Example (Cont.)

- **P2**

$$Y = ymax = 400$$
$$x = x2 + (y - y2)/m$$
$$x = 500 + (400 - 450)/\frac{4}{5} = 437,5$$

- **P1= (200,210) P2= (437.5, 400)**
- **P1 code → 0000**
- **P2 code → 0000**
- **Trivial accept**



Liang-Barsky Line Clipping

- Line clipping approach is given by the Liang and Barsky is faster then cohen-sutherland line clipping.
- Which is based on analysis of the parametric equation of the line which are,

$$x = x_1 + t\Delta x$$
$$y = y_1 + t\Delta y$$

- Where $0 \leq t \leq 1$,$\Delta x = x_2 - x_1$ and $\Delta y = y_2 - y_1$.

Liang-Barsky Line Clipping 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,

$p_1 = -\Delta x,\qquad q_1 = x_1 - x_{wmin}$

$p_2 = \Delta x,\qquad q_2 = x_{wmax} - x_1$

$p_3 = -\Delta y,\qquad q_3 = y_1 - y_{wmin}$

$p_4 = \Delta y,\qquad q_4 = y_{wmax} - y_1$

Contd.

```
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 8.
        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.
```

Contd.

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 $t_1\ \ and\ \ t_2$ as given below:

$$t_1 = \max\{0, r_k | where\ k\ takes\ all\ values\ for\ which\ p_k < 0\}$$

$$t_2 = \min\{1, r_k | where\ k\ takes\ all\ values\ for\ which\ p_k > 0\}$$

7. If $u_1 \leq u_2$ then

    Calculate endpoints of clipped line:

    $x_1' = x_1 + t_1 \Delta x$

    $y_1' = y_1 + t_1 \Delta y$

    $x_2' = x_1 + t_2 \Delta x$

    $y_2' = y_1 + t_2 \Delta y$

    Draw line (x_1', y_1', x_2', y_2');

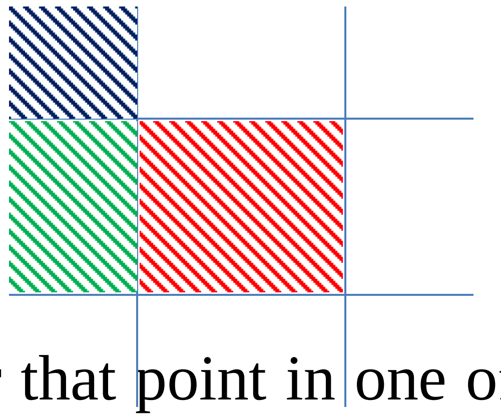
8. Stop.

Nicholl-Lee-Nicholl Line (NLN) 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.
- 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.

Contd.

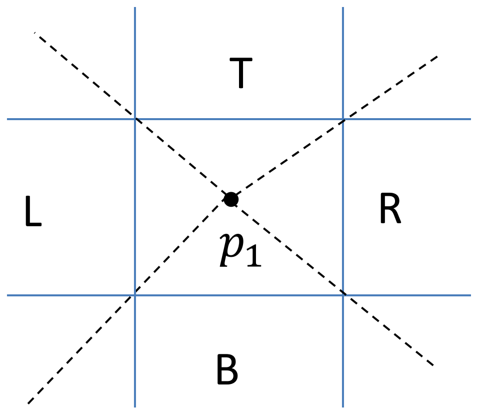
- For given line we find first point falls in which region out of nine region shown in figure.
- Only three region are considered which are.
  - Window region
  - Edge region
  - Corner region



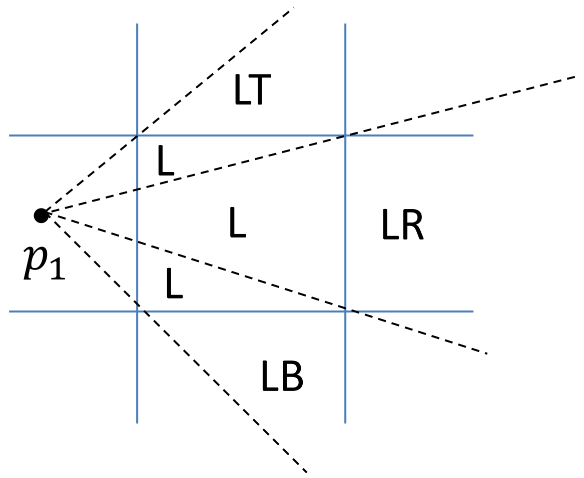
- If point falls in other region than we transfer that point in one of the three region by using transformations.
- We can also extend this procedure for all nine regions.

Dividing Region in NLN

- Based on position of first point out of three region highlighted we divide whole space in new regions.
- Regions are name in such a way that name in which region p2 falls is gives the window edge which intersects the line.
- p_1 is in window region

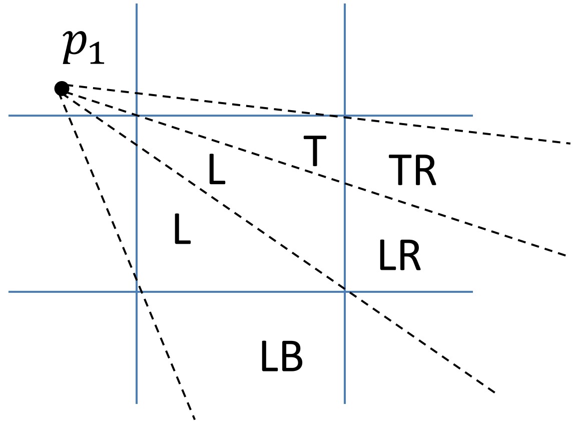


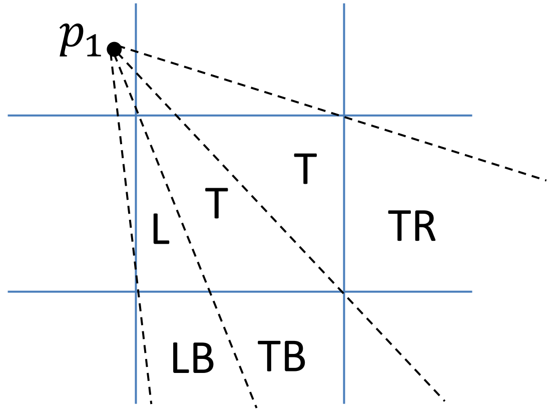
- p_1 is in edge region



Contd.

- p_1 is in Corner region (one of the two possible sets of region can be generated)





Finding Region of Given Line in NLN

- For finding that in which region line $\boldsymbol{p_1 p_2}$ falls we compare the slope of the line to the slope of the boundaries:

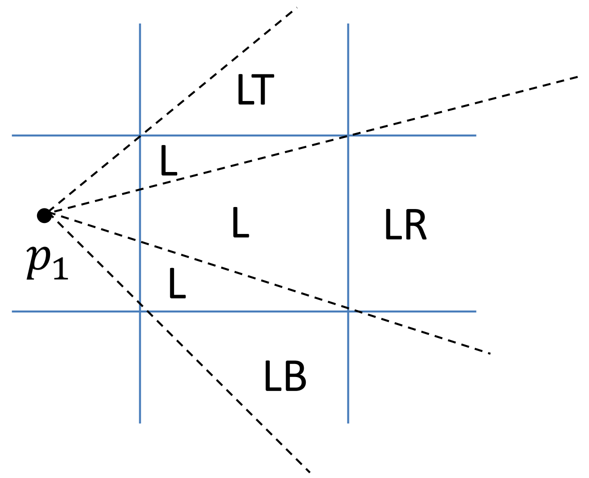
  $$\boldsymbol{slope\,\overline{p_1 p_{B1}}} < \boldsymbol{slope\,\overline{p_1 p_2}} < slope\,\boldsymbol{\overline{p_1 p_{B2}}}$$

  Where $\boldsymbol{\overline{p_1 p_{B1}}}$ and $\boldsymbol{\overline{p_1 p_{B2}}}$ are boundary lines.

- For example p1 is in edge region and for checking whether p2 is in region LT we use following equation.

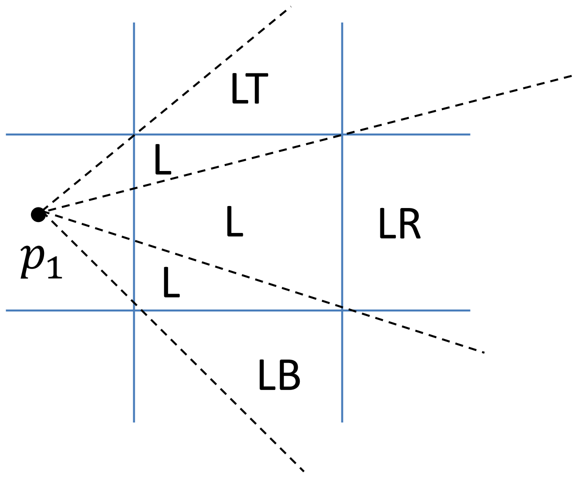
  $$\boldsymbol{slope\,\overline{p_1 p_{TR}}} < \boldsymbol{slope\,\overline{p_1 p_2}} < slope\,\boldsymbol{\overline{p_1 p_{TL}}}$$

  $$\boldsymbol{\frac{y_T - y_1}{x_R - x_1} < \frac{y_2 - y_1}{x_2 - x_1} < \frac{y_T - y_1}{x_L - x_1}}$$



Contd.

- 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.



Intersection Calculation in NLN

- 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 $u = (x_{l/r} - x_1)/(x_2 - x_1)$, so that y can be obtain from parametric equation as below:

  $$\boldsymbol{y = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x_L - x_1)}$$

- Keep the portion which is inside and clip the rest.

Contd.

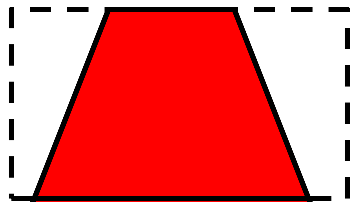
- Similarly for top or bottom boundary $y = y_t\ or\ y_b$ respectively, and $u = (y_{t/b} - y_1)/(y_2 - y_1)$, so that we can calculate x intercept as follow:

$$\boldsymbol{x = x_1 + \frac{x_2 - x_1}{y_2 - y_1}(y_T - y_1)}$$

Polygon Clipping

- For polygon clipping we need to modify the line clipping procedure.
- In line clipping we need to consider about only line segment.
- In polygon clipping we need to consider the area and the new boundary of the polygon after clipping.
- Various algorithm available for polygon clipping are:

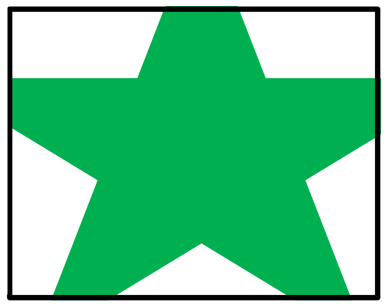
1. Sutherland-Hodgeman Polygon Clipping
2. Weiler-Atherton Polygon Clipping etc.



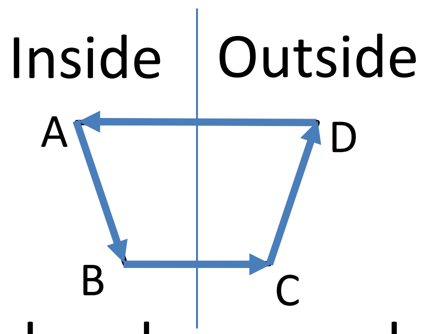
Sutherland-Hodgeman Polygon Clipping

- 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.

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



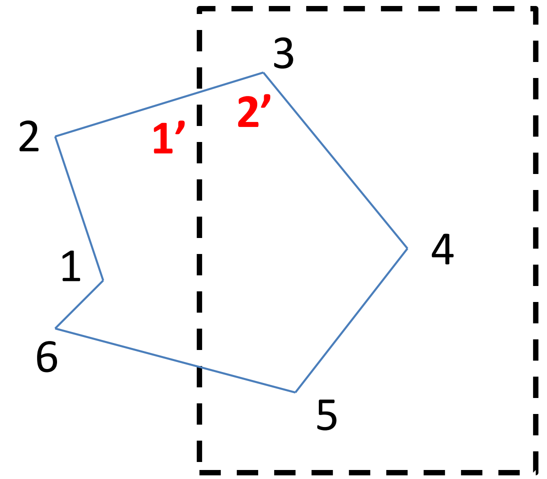
Processing Steps



- We process vertices in sequence as a closed polygon.
- Four possible cases are there.

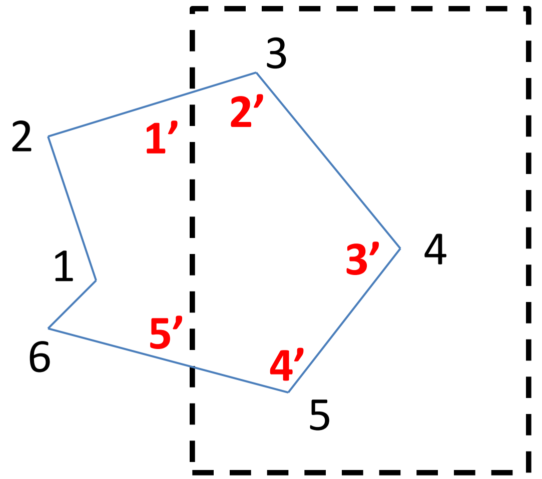
1. If both vertices are inside the window we add only second vertices to output list.
2. If first vertices is inside the boundary and second vertices is outside the boundary only the edge intersection with the window boundary is added to the output vertex list.
3. If both vertices are outside the window boundary nothing is added to window boundary.
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.

Example



- As shown in figure 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.

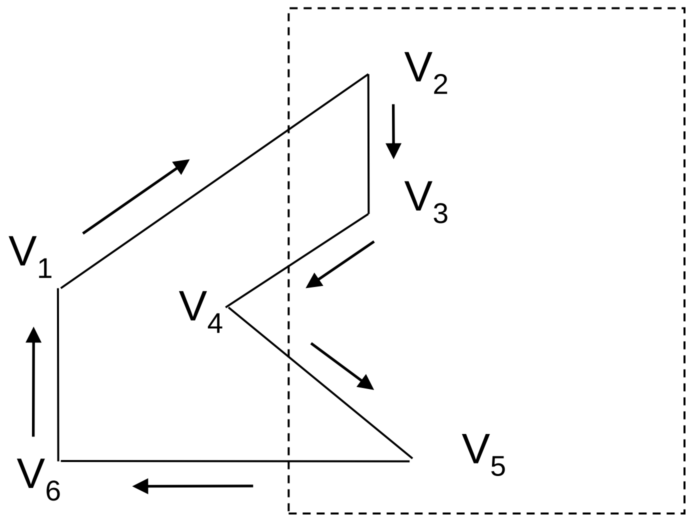
Contd.



- 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.
- From 5 to 6 we move inside to outside so we add intersection pint to output list.
- Finally 6 to 1 both vertex are outside the window so we does not add anything.

Limitatin of Sutherlan-Hodgeman Algorithm

- It may not clip concave polygon properly.

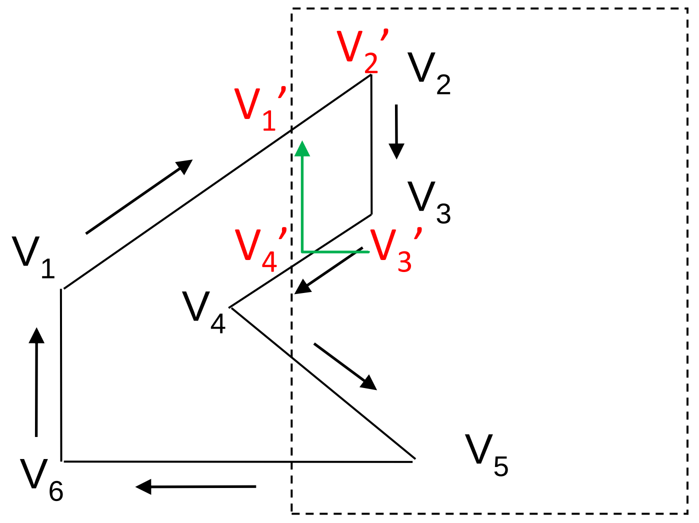


- One possible solution is to divide polygon into numbers of small convex polygon and then process one by one.
- Another approach is to use Weiler-Atherton algorithm.

Weiler-Atherton Polygon Clipping

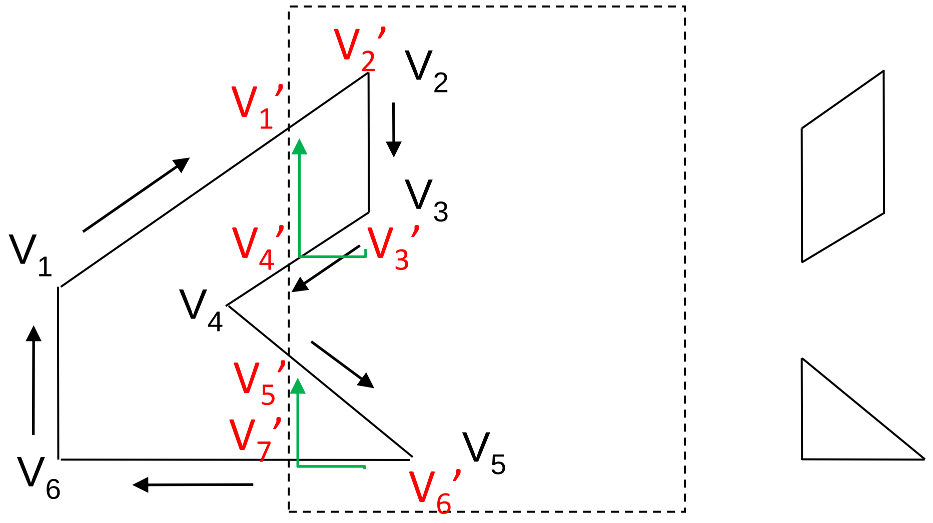
- It modifies Sutherland-Hodgeman vertex processing procedure for window boundary so that concave polygon also clip correctly.
- This can be applied for arbitrary polygon clipping regions as it is developed for visible surface identification.
- Procedure is similar to Sutherland-Hodgeman algorithm.
- Only change is sometimes need to follow the window boundaries Instead of always follow polygon boundaries.
- For clockwise processing of polygon vertices we use the following rules:
  1. For an outside to inside pair of vertices, follow the polygon boundary.
  2. For an inside to outside pair of vertices, follow the window boundary in a clockwise direction.

Example



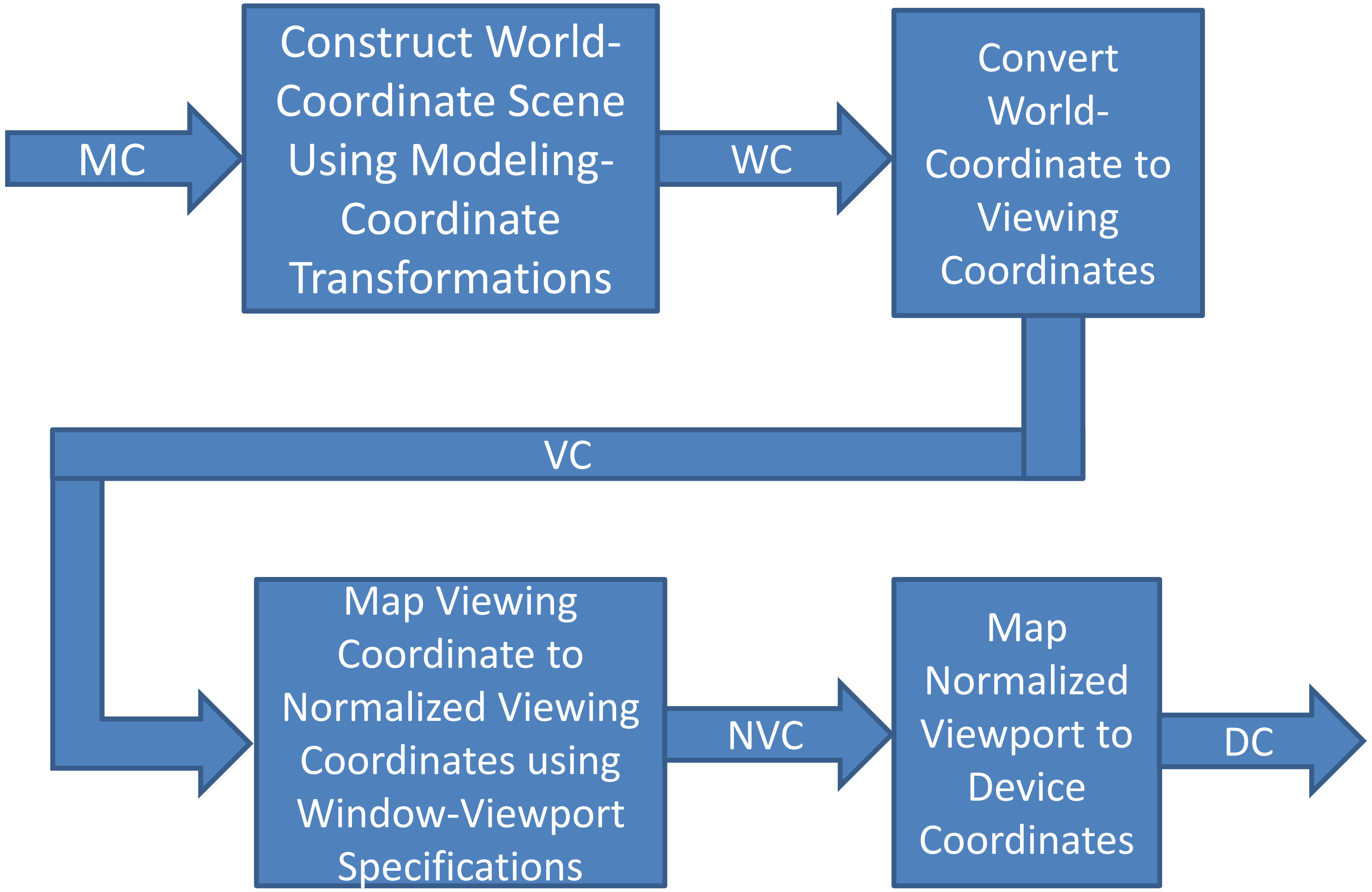
- Start from v1 and move clockwise towards v2 and add intersection point and next point to output list by following polygon boundary,
- then from v2 to v3 we add v3 to output list.
- From v3 to v4 we calculate intersection point and add to output list and follow window boundary.

Contd.



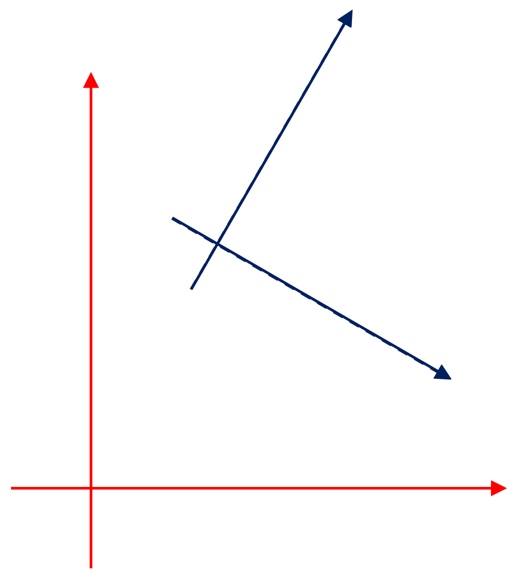
- Similarly from v4 to v5 we add intersection point and next point and follow the polygon boundary,
- next we move v5 to v6 and add intersection point and follow the window boundary, and
- finally v6 to v1 is outside so no need to add anything.
- This way we get two separate polygon section after clipping.

Contd.


MC
Construct World-Coordinate Scene Using Modeling-Coordinate Transformations
WC
Convert World-Coordinate to Viewing Coordinates
VC
Map Viewing Coordinate to Normalized Viewing Coordinates using Window-Viewport Specifications
NVC
Map Normalized Viewport to Device Coordinates
DC

Viewing Coordinate Reference Frame

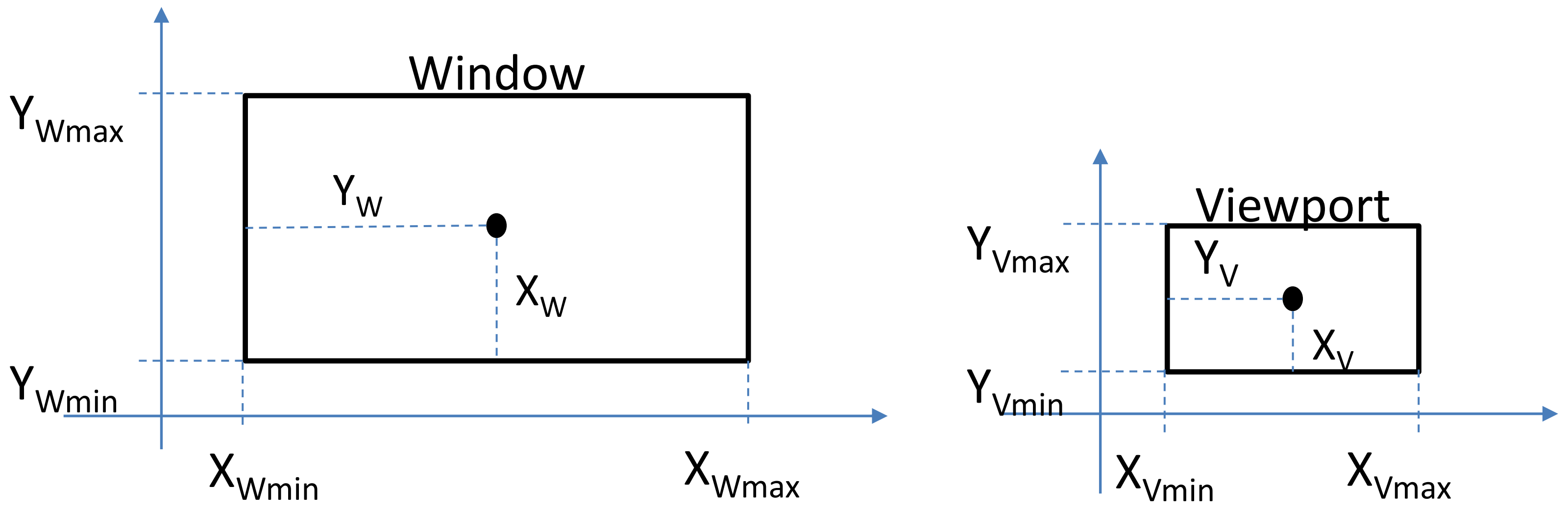
- We can obtain reference frame in any direction and at any position.
- For handling such condition
  - first of all we translate reference frame origin to standard reference frame origin.
  - Then we rotate it to align it to standard axis.



- In this way we can adjust window in any reference frame.
- It is illustrate by following transformation matrix: $M_{wc,vc} = RT$
- 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.



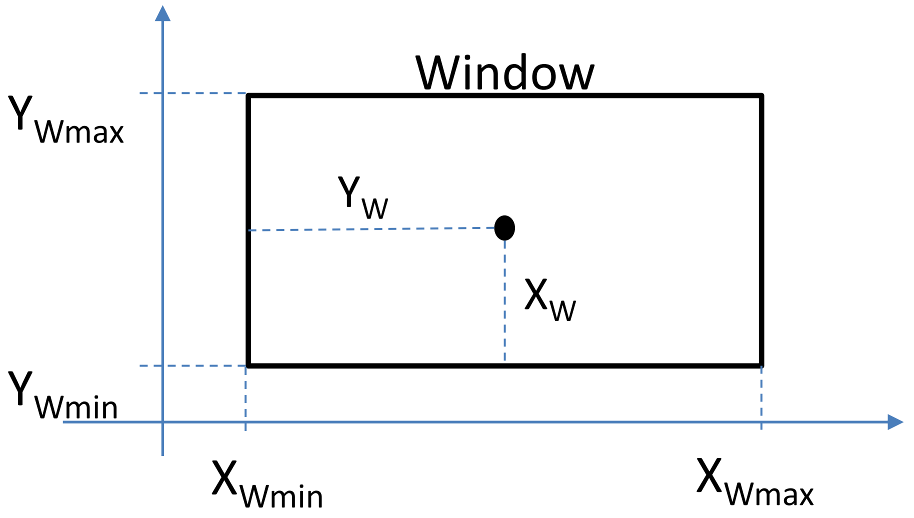
Contd.

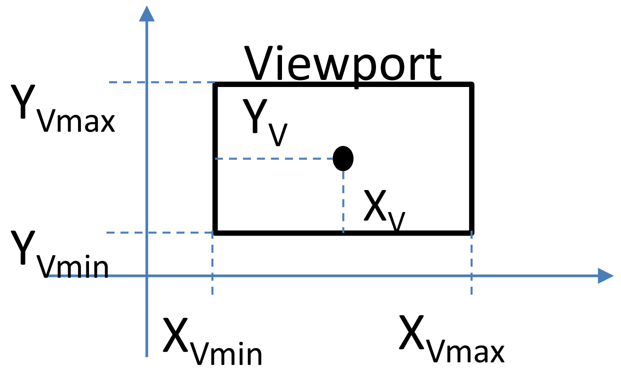
- We find relative position by equation as follow:

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

- *Similarly*

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





Contd.

- Solving for x direction by making viewport position as subject we obtain:

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

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

$$x_v = x_{vmin} + (x_w - x_{wmin}) s_x$$

- *Where*

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

Contd.

- Similarly Solving for y direction by making viewport position as subject we obtain:

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

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

$$y_v = y_{vmin} + (y_w - y_{wmin}) s_y$$

- $Where$

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

Contd.

- We can also map window to viewport with the set of transformation:
  - Perform a scaling transformation using a fixed-point position of $(x_{wmin}, ywmin)$ that scales the window area to the size of the viewport.
  - Translate the scaled window area to the position of the viewport.
- For maintaining relative proportions we take $(s_x = s_y)$.
- 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.
  - Other is to maintain standard character size even though viewport size is enlarged or reduce.