View Transformation

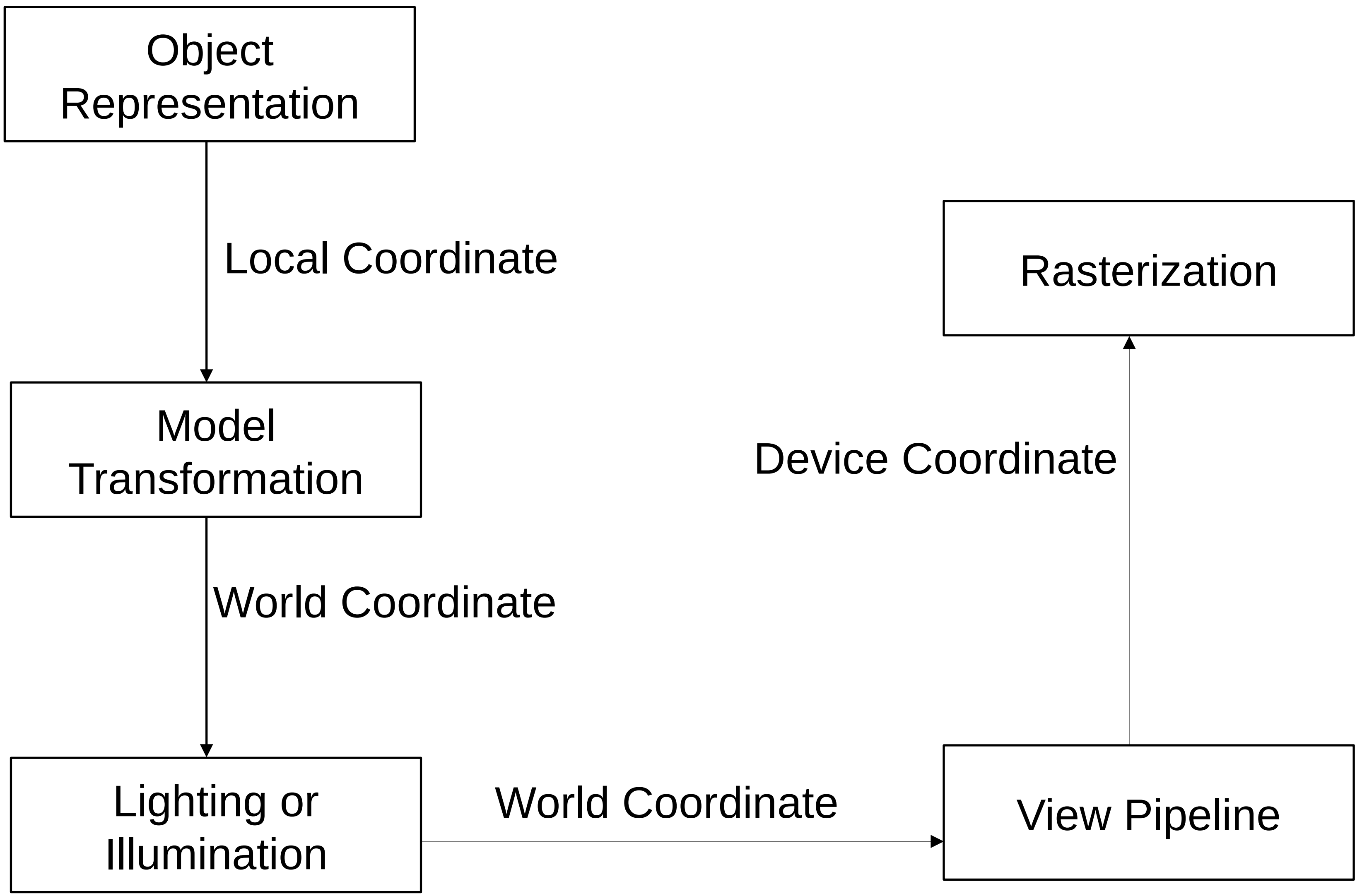
Instructor: Dinesh Maharjan

2022

Graphics Pipeline

- Computer graphics is the process of rendering scene on display screen efficiently.
- This takes several processing and computations.
- These stages are called graphics pipeline.
- Graphics pipeline goes mainly through 5 stages

Graphics Pipeline



Object Representation

- Objects are represented in their local coordinates.
- Their position, size is not important in local coordinate.

Modeling Transformation

- Objects are combined to create scenes.
- This will transform objects from local coordinate system to world coordinate system.
- Their position, size and location is important in world coordinate.
- This stage is also known as geometric transformation.

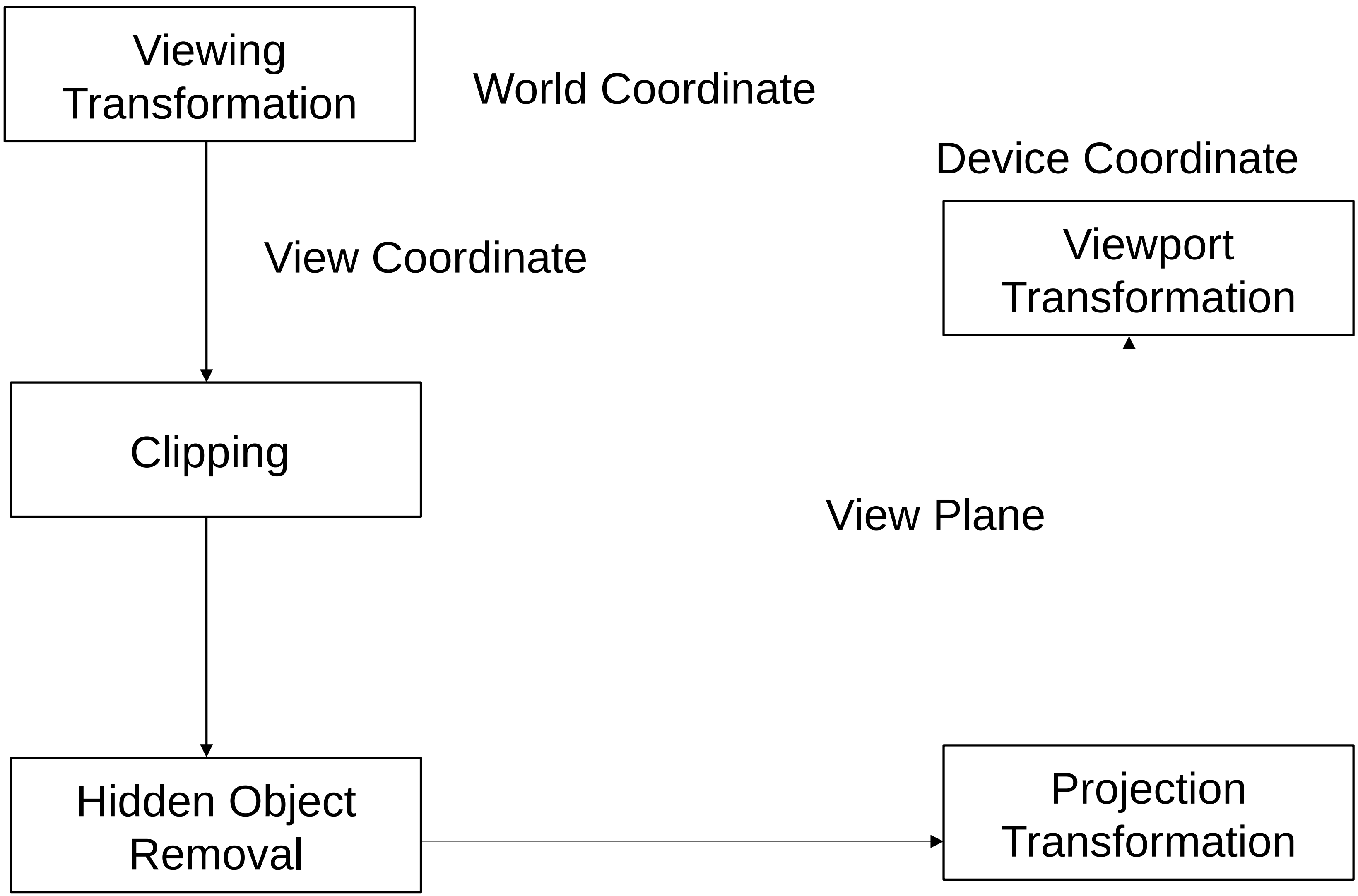
Lighting or Illumination

- Applies color to surface of objects.
- Colors gives sensation of 3D.
- It makes clear which surface is closer and which is further.

Viewing Pipeline

- 3D world coordinate scene is mapped to 2D view plane scene.
- This is done in several sub stages.
- That’s why it is called pipeline.
- Mainly 5 sub stages.
- Viewing Transformation
- Clipping
- Hidden Objects Removal
- Projection Transformation
- Viewport Transformation

View Pipeline



Viewing Transformation

- The world coordinate scene is transferred to view coordinate system.
- View coordinate system is similar to camera coordinate system.
- 3D coordinate system is transformed to 3D view coordinate system.

Clipping

- For this, we need to define a region in view coordinate space.
- The region in called view volume.
- We need to capture the objects inside the view volume.
- Typically same to clicking photographs with camera.
- It means we select region and capture it.
- The process of removing objects partially or fully outside of view volume is called clipping.

Hidden Objects Removal

- When we project, we consider the viewer position.
- Some objects will be fully visible and some partially with in view volume.
- Objects behind other objects are not visible.
- Such hidden objects are removed in this stage.
- It is also known as visible surface detection.

Projection Transformation

- The scene in view coordinate system is projection on view plane.
- Objects outside view volume will not be projected.

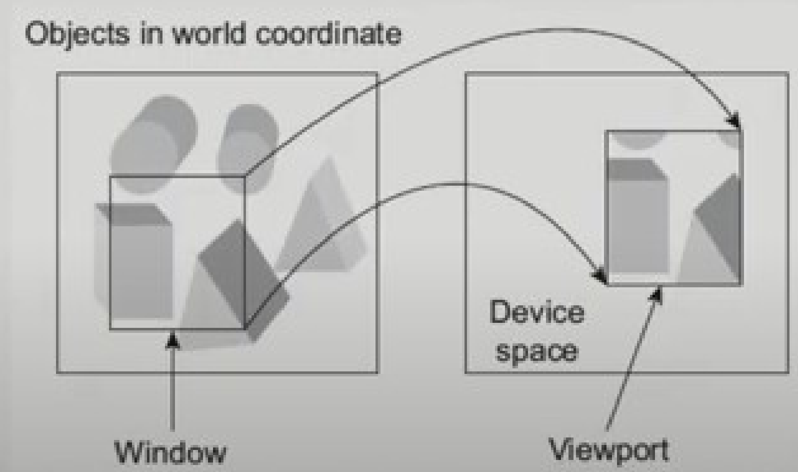
Viewport Transformation

- 2D projected scene is also called window.
- Window can be displayed at any portion of the computer screen.
- The portion of the screen where window is displayed is called viewport.
- Transferring the content from window to viewport is called window to viewport transformation.
- This transforms view coordinate to device coordinate.

Viewport Transformation

Window & Viewport

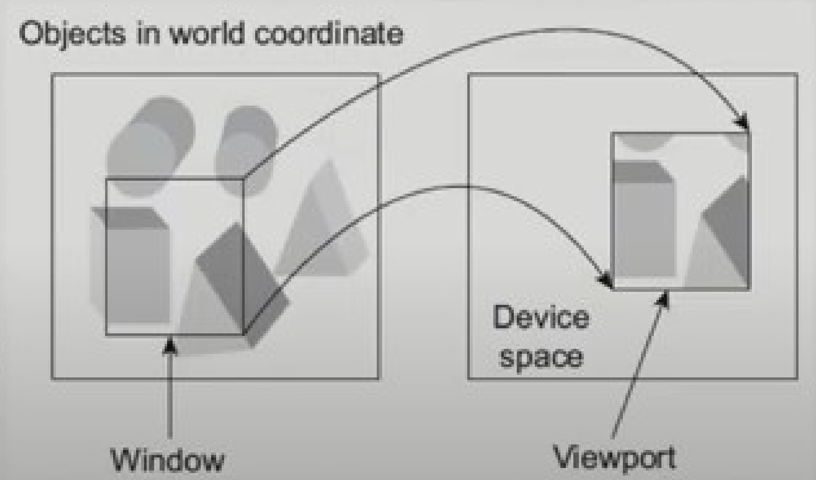
- Clipping window abstract and intermediate concept in image synthesis process
  - Points on clipping window constitute objects to show on screen
  - However, scene may or may not occupy whole screen



Viewport Transformation

Window & Viewport

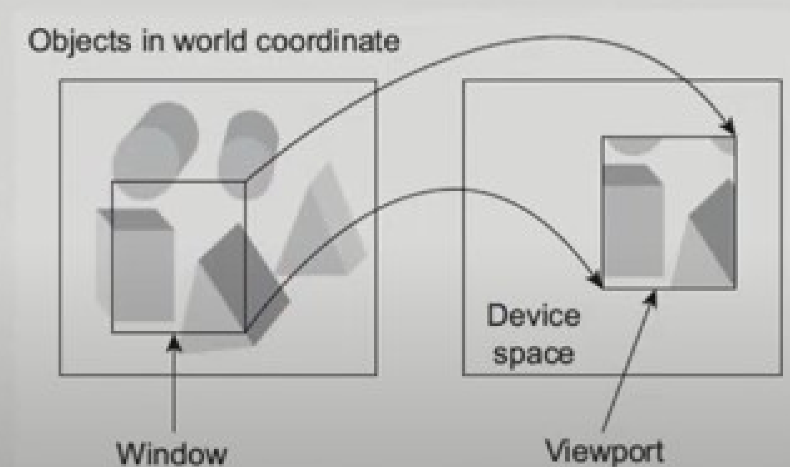
- **Window -** same as (normalized) clipping window
  - WC objects projected on this window



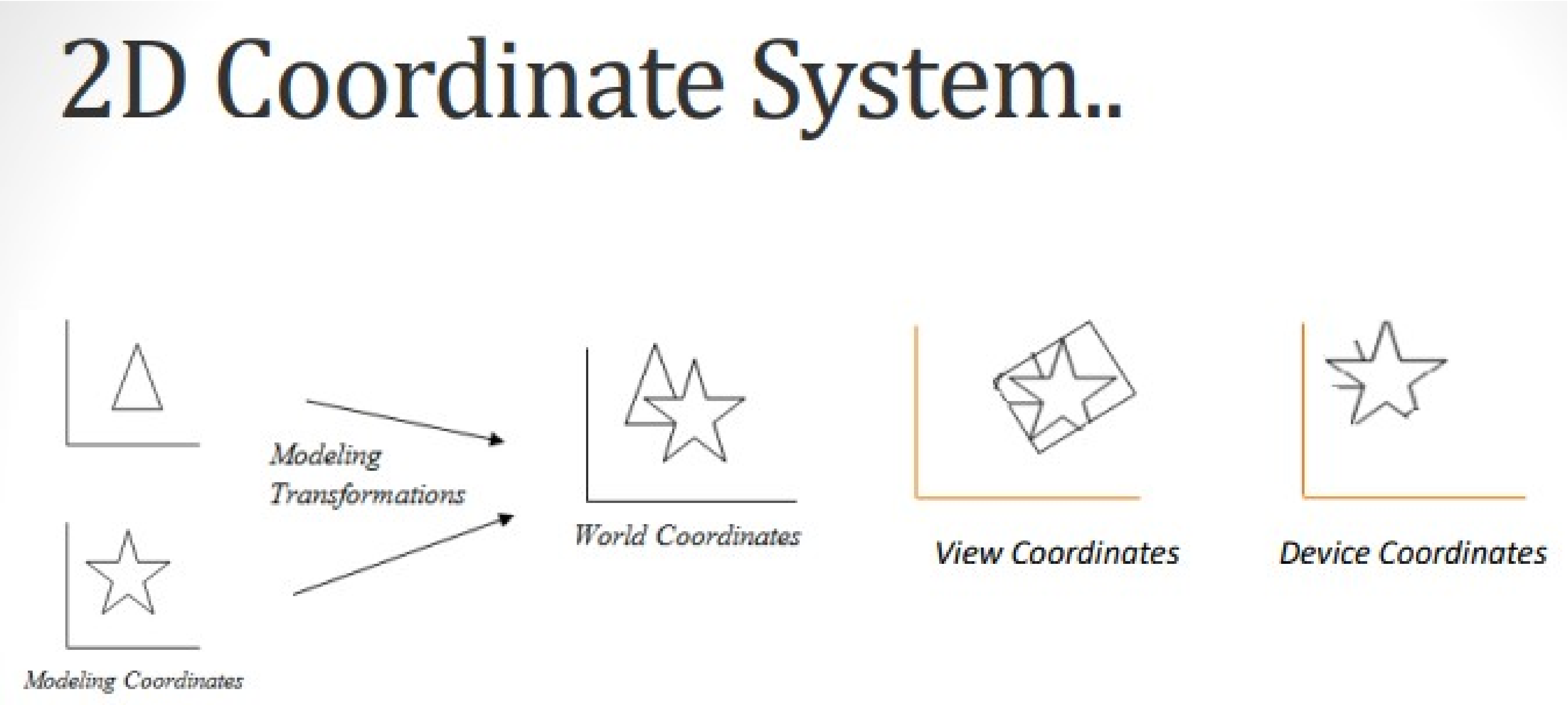
Viewport Transformation

Window & Viewport

- Viewport defined in **device space –** w.r.t screen origin and dimensions
  - One more transformation required to transfer points from window (VC) to viewport (DC)



Viewport Transformation



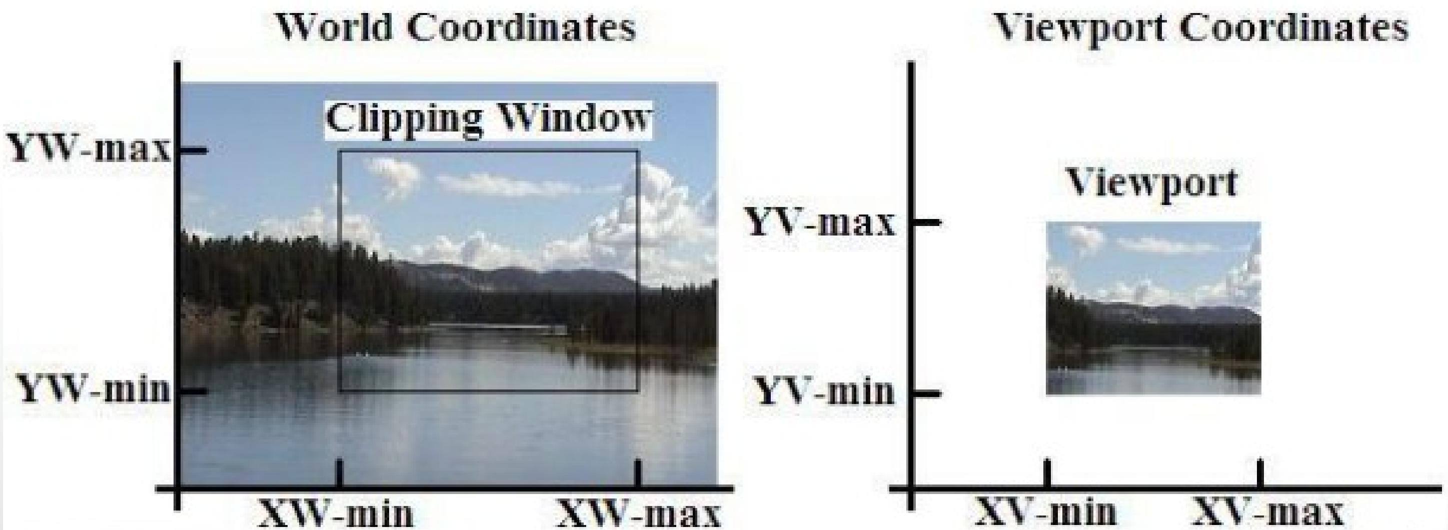
Window and Viewport

Window:

- A world-coordinate area selected for display is called a *window or clipping window*. That is, window is the section of the 2D scene that is selected for viewing.
- The window defines what is to be viewed.

Viewport

- An area on a display device to which a window is mapped is called a viewport.
- The viewport indicates where on an output device selected part will be displayed.



Rasterization

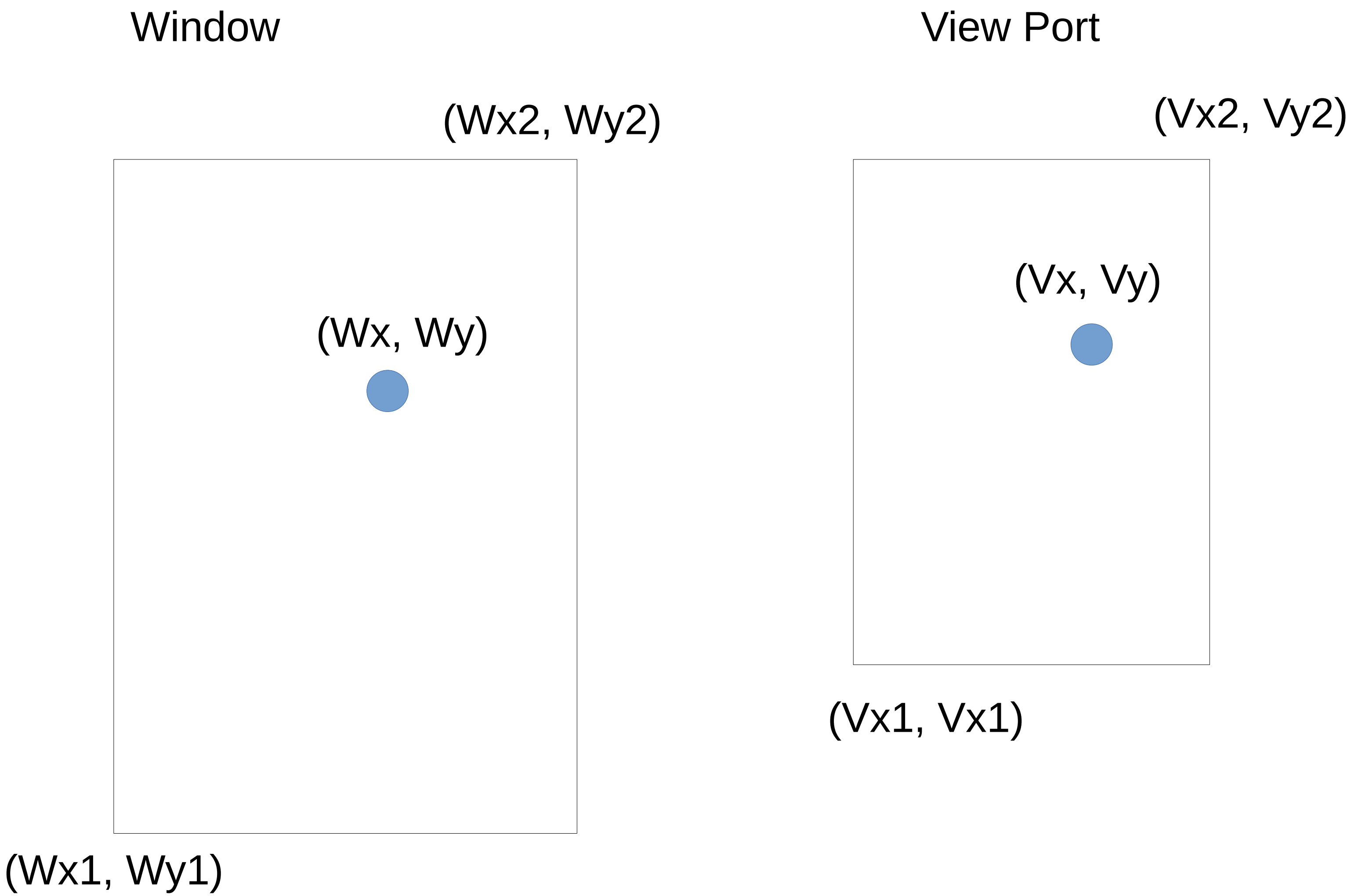
- Viewport is abstract representation of the actual display.
- Actual display is a pixel grid.
- Display locations are discrete.
- Point (1.5,2.4) cannot be excited in display screen.
- These points must be mapped to integer coordinates by rounding them.
- This mapping is called scan line conversion.

Window to Viewport

- The projected scene can be displayed at any location and with any size on display screen.
- However, we want to maintain relative positive and size of this point with respect to size of window or its boundaries.
- Same this should be maintained while we transform it from window to viewport.
-

Window to Viewport

- Let (W_x,W_y) be any point on window or view plane.
- We want to map this point to (V_x,V_y) on view port.
- Let (W_{x1},W_{y1}) and (W_{x2},W_{y2}) be two opposite points of window.
- And (V_{x1},V_{y1}) and (V_{x2},V_{y2}) be two opposite points of View port.
- Provides flexibility of display the projected image anywhere on the screen with any size, irrespective of the size of the clipping window.


Window
(Wx2, Wy2)
(Wx, Wy)
(Wx1, Wy1)
View Port
(Vx2, Vy2)
(Vx, Vy)
(Vx1, Vx1)

Window to Viewport

- To maintain the relative position with respect to size, we should have
-

$$\frac{Wx-Wx1}{Wx2-Wx1} = \frac{Vx-Vx1}{Vx2-Vx1}$$

$$\text{Or, } Vx = \frac{Vx2-Vx1}{Wx2-Wx1}(Wx - Wx1) + Vx1$$

$$\text{Or, } Vx = sx.Wx - sx.Wx1 + Vx1 \text{ Where, } sx = \frac{Vx2-Vx1}{Wx2-Wx1}$$

$$\text{Or, } Vx = sx.Wx + tx \text{ Where, } tx = sx.(-Wx1) + Vx1$$

Window to Viewport

Similary, we must have

$$\frac{Wy - Wy1}{Wy2 - Wy1} = \frac{Vy - Vy1}{Vy2 - Vy1}$$

Or, $Vy = sy.Wy + ty$
Where, $sy = \frac{Vy2 - Vy1}{Wy2 - Wy1}$

$$ty = sy.(-Wy1) + Vy1$$

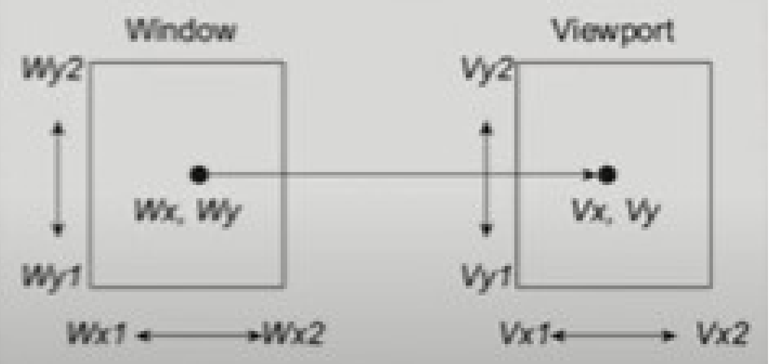
Window to Viewport

- Firstly we need to translate the window to origin i.e. T(-Wx1,-Wy1)
- Scale to the size of viewport i.e. S(sx,sy)
- Translate scaled window to the position of viewport.

Transformation Matrix

- Using the expressions, we can get transformation matrix

$$T_{vp} = \begin{bmatrix} sx & 0 & tx \\ 0 & sy & ty \\ 0 & 0 & 1 \end{bmatrix}$$

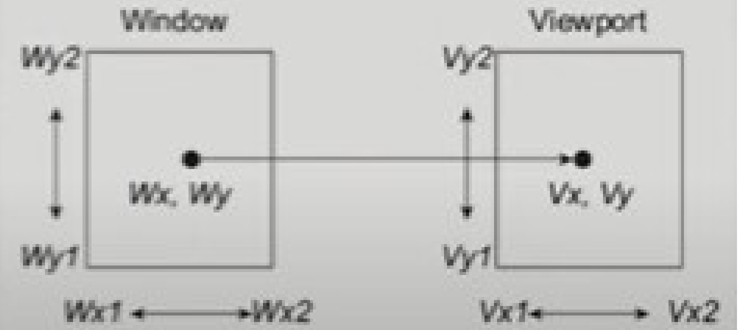


Transformation Matrix

- to get transformed point

$$\begin{bmatrix} x'' \\ y'' \\ w \end{bmatrix} = T_{vp}P_w = \begin{bmatrix} sx & 0 & tx \\ 0 & sy & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

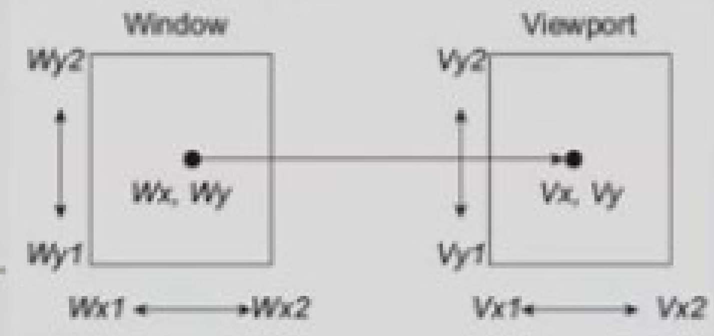
$$x' = \frac{x''}{w}, y' = \frac{y''}{w}$$



Example 3

- Let us assume point is projected on a normalized clipping window (as we saw, projected point in either parallel or perspective projection is $(0,0,-0.5)$, which lies at the center of the normalized window). We want to show the scene on a viewport having lower left and top right corners at $(4,4)$ and $(6,8)$ respectively. What would the position of the point be in the viewport?

Example 3



- Since clipping window is normalized, we have $Wx1 = -1, Wx2 = 1, Wy1 = -1$ and $Wy2 = 1$
- Also, from viewport specification, we have $Vx1 = 4$, $Vx2 = 6$, $Vy1 = 4$ and $Vy2 = 8$

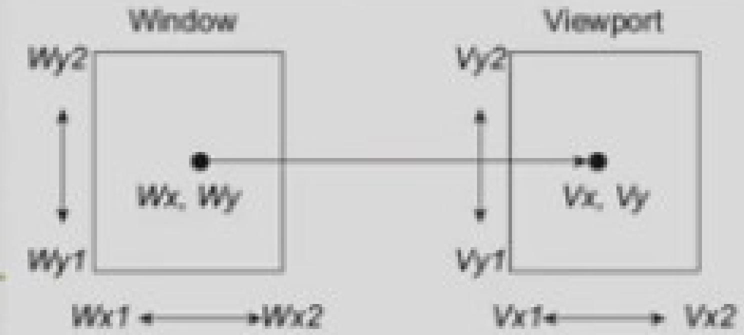
$$sx = \frac{6-4}{1-(-1)} = 1$$

$$sy = \frac{8-4}{1-(-1)} = 2$$

$$tx = 1 - (-1) + 4 = 5$$

$$ty = 2 - (-1) + 4 = 6$$

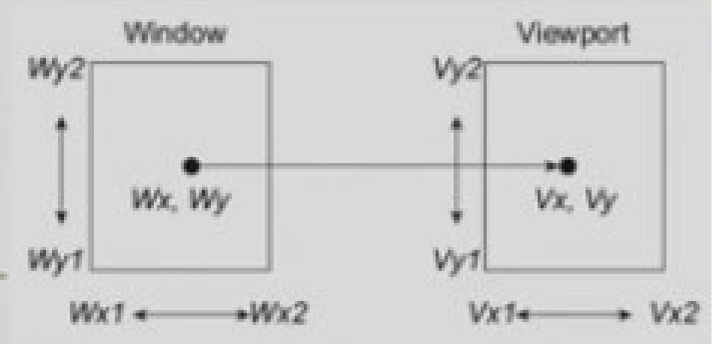
Example 3



- So, transformation matrix (W-V)

$$T_{vp} = \begin{bmatrix} sx & 0 & tx \\ 0 & sy & ty \\ 0 & 0 & 1 \end{bmatrix} \qquad T_{vp} = \begin{bmatrix} 1 & 0 & 5 \\ 0 & 2 & 6 \\ 0 & 0 & 1 \end{bmatrix}$$

Example 3



- Transformed point (homogeneous coordinate)

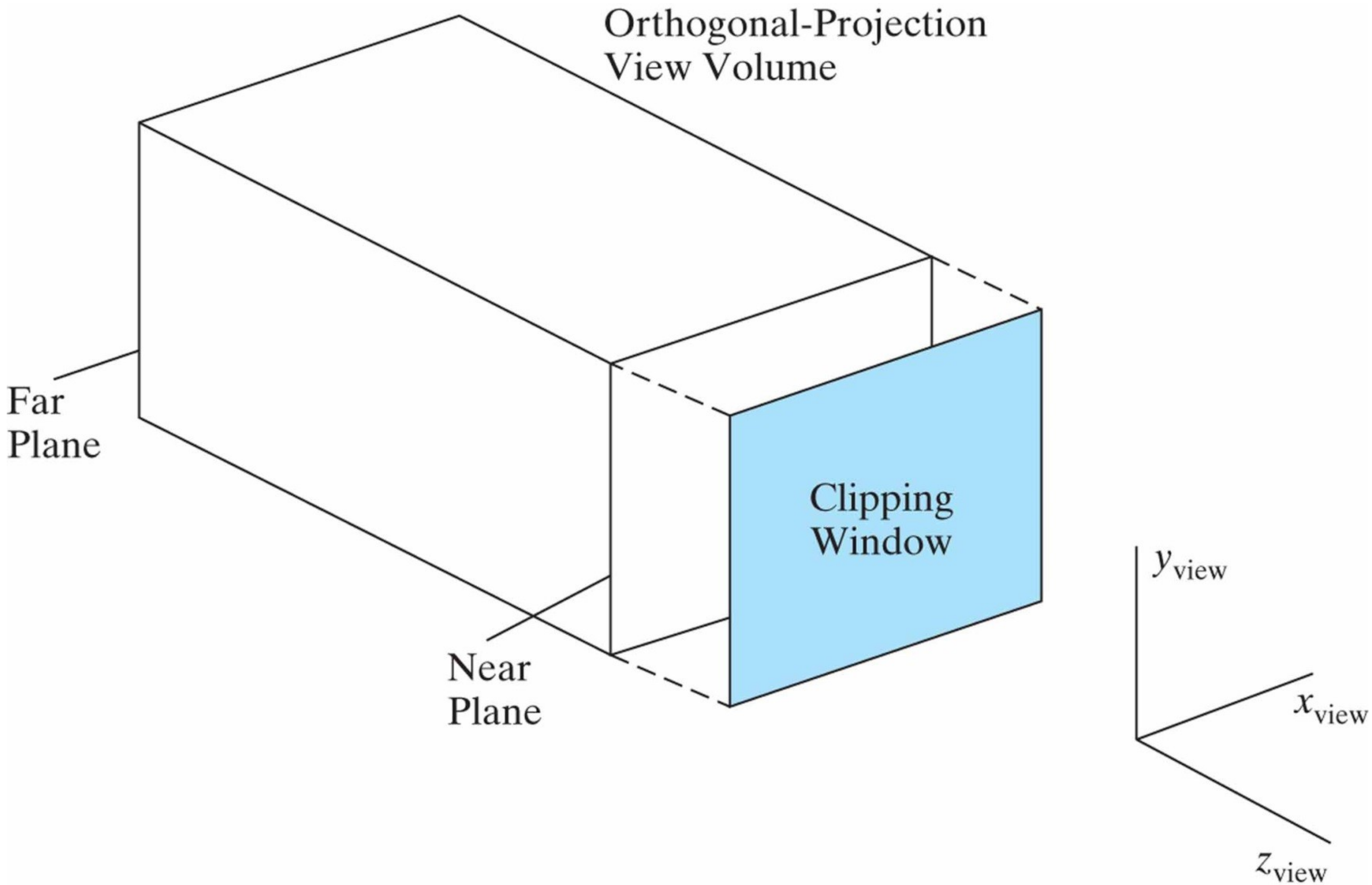
$$\begin{bmatrix} 1 & 0 & 5 \\ 0 & 2 & 6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -2.5 \\ -3 \\ -0.5 \end{bmatrix}$$

- The point is $\left(\frac{-2.5}{-0.5}, \frac{-3}{-0.5}\right)$ or (5,6)

Clipping

- We have discussed that we define a view volume to display portion of the whole scene on the screen.
- Everything outside of this view volume is clipped and the projected.
- It will be difficult to check if objects lie inside the view volume.
- So, we define view volume and project it on the view plane.

Clipping



Orthogonal-Projection
View Volume
Far
Plane
Clipping
Window
Near
Plane
y_{view}
x_{view}
z_{view}

Clipping

- We have discussed that we define a view volume to display portion of the whole scene on the screen.
- Everything outside of this view volume is clipped and the projected.
- So, we define view volume and project it on the view plane.

Clipping

- The projected view volume is called view window, or clipping window or simply window.
- Note that we do not project view volume only.
- We project all the world coordinate objects to the view plane.
- The objects outside of window is clipped.

Clipping

- Point Clipping
- Line Clipping
- Polygon Clipping

Point Clipping

- In a rectangular clip window save a point P(x,y) for display if the following is true.
- $Wx1 <= x <= Wx2$ and
- $Wy1 <= y <= Wy2$
- Otherwise is clipped.

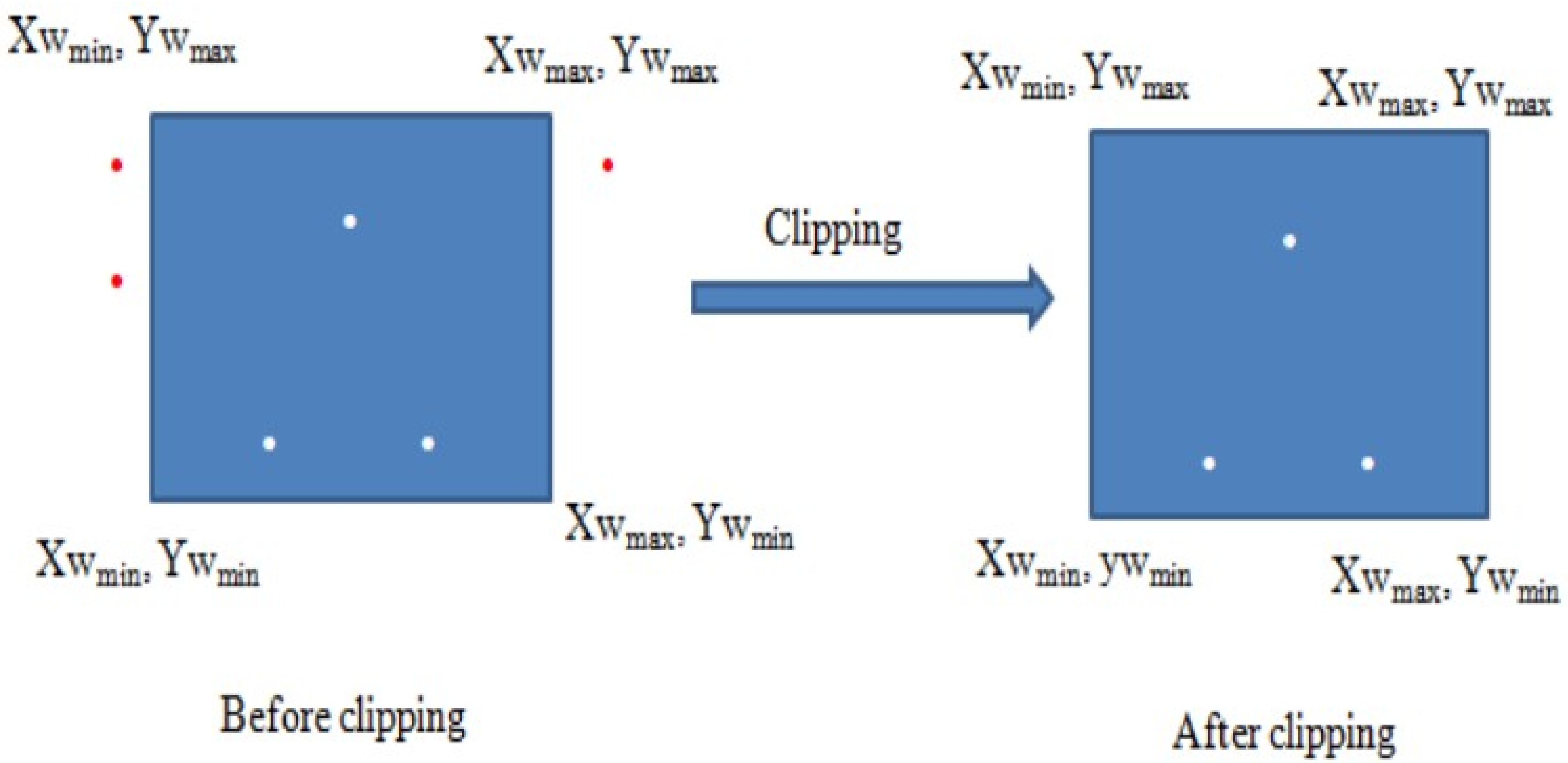
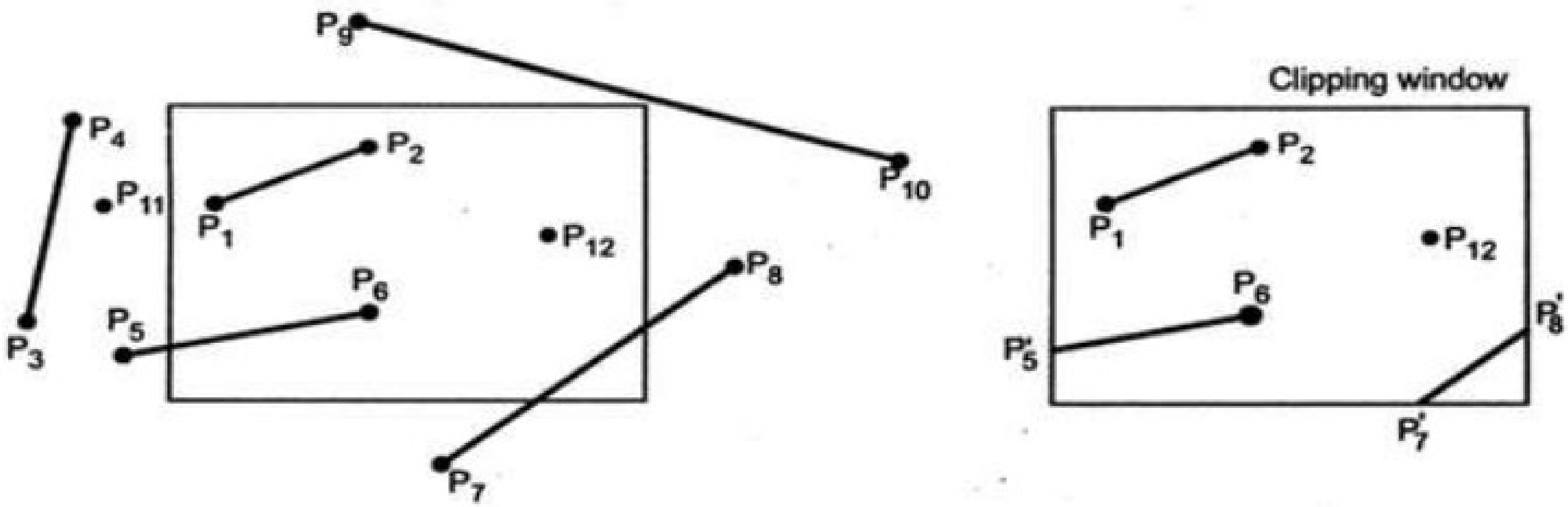


Fig: Point Clipping process

Line clipping

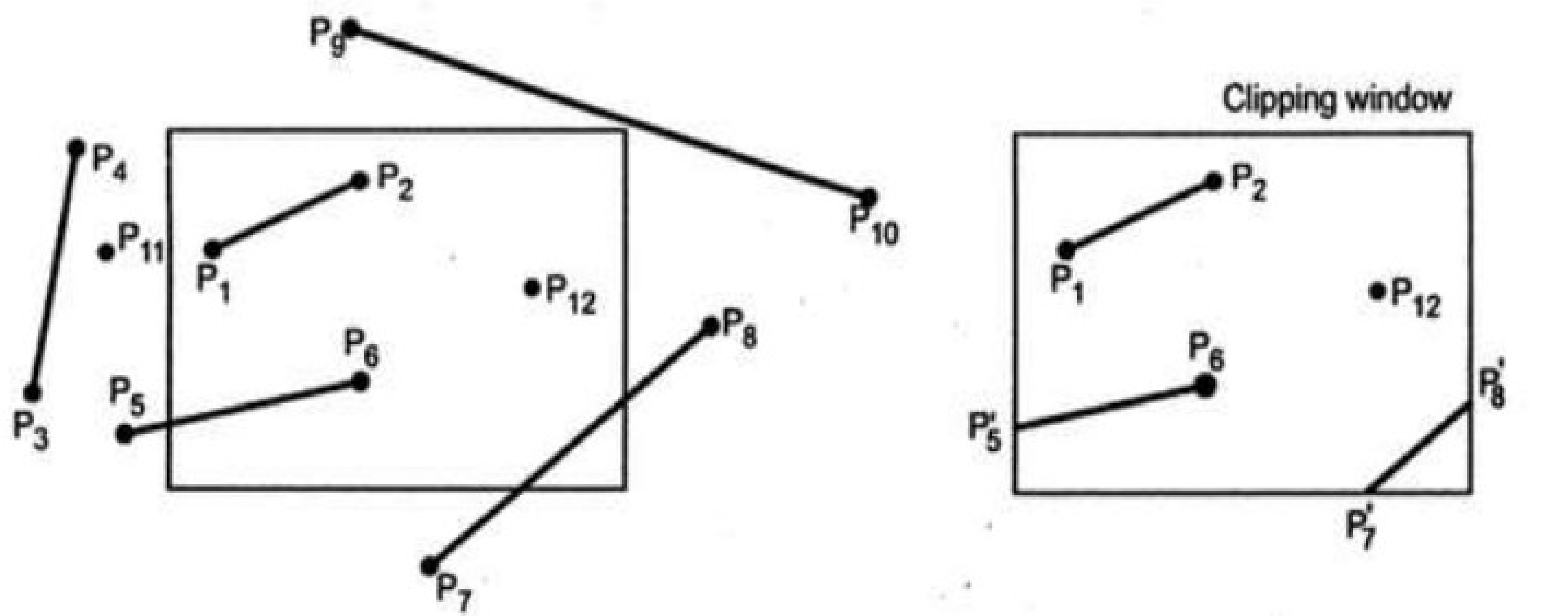
- The visible segment of a straight line can be determined by **inside – outside test:**
  - A line with both endpoints **inside** clipping boundary, such as the line from $\mathbf{p_1}$ to $\mathbf{p_2}$, is saved.



(a) Before clipping

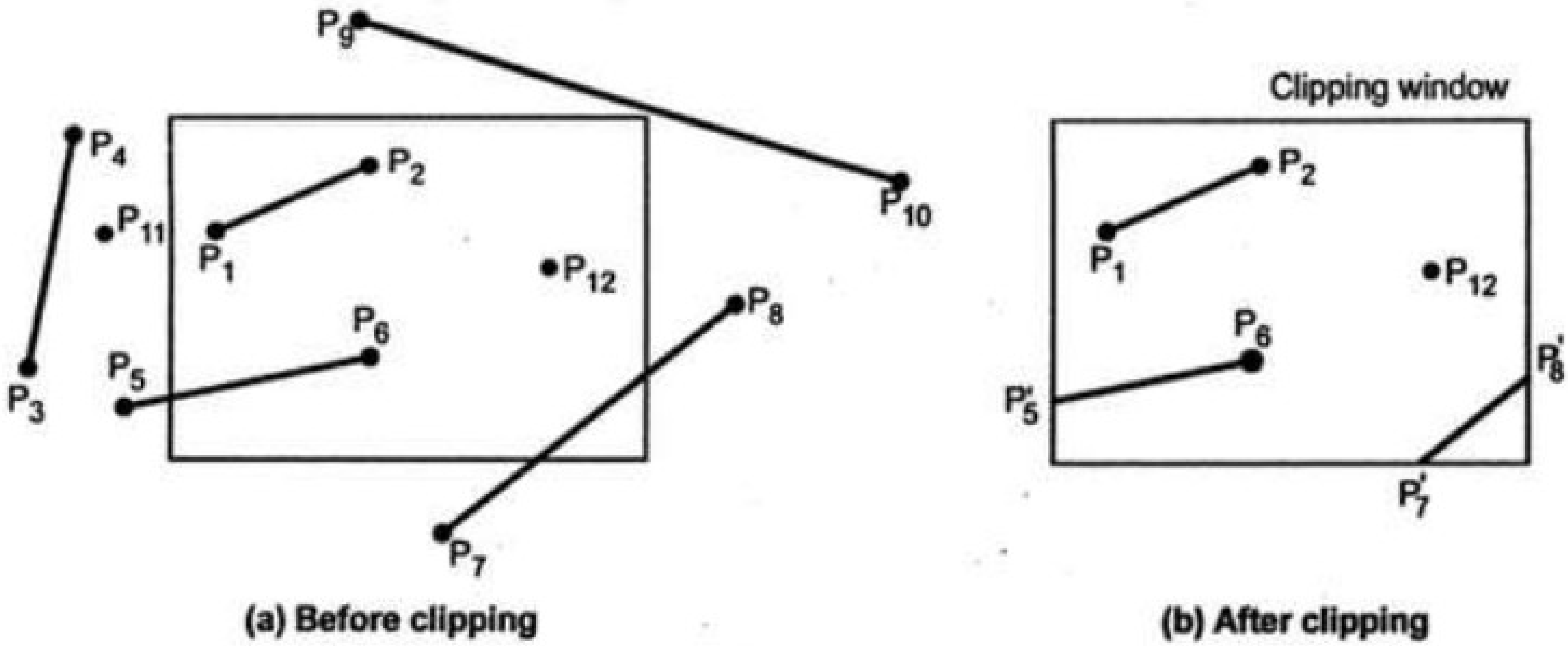
(b) After clipping

- A line with both endpoints **outside** the clip boundary, such as the line from $\mathbf{p_3}$ to $\mathbf{p_4}$, is not saved.



(a) Before clipping (b) After clipping

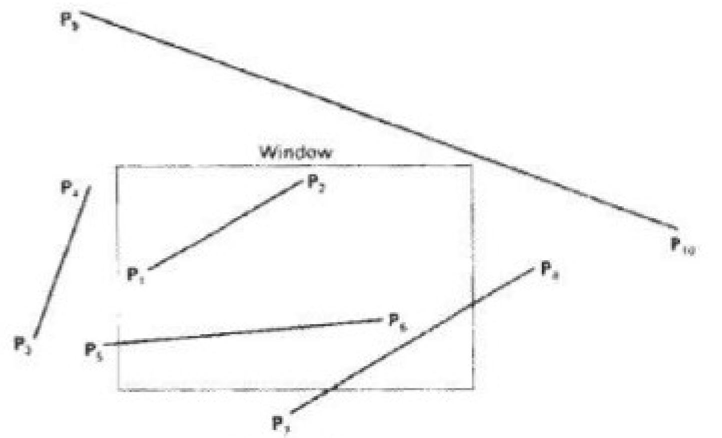
- If the line is not completely inside or completely outside (e.g., p7 to p8), then perform intersection calculations with one or more clipping boundaries.



Line Point Clipping

- *A* line clipping procedure involves several parts.

1. First, we can test a given line segment to determine **whether it lies completely inside** the clipping window.
2. If it does not, we try to determine whether it **lies completely outside** the window.
3. Finally, we must perform **intersection calculations with one or more clipping boundaries.**



Before Clipping
(a)

Cohen-Sutherland Algorithm

- Window and its surrounding is assumed to be divided into 9 regions.
- It means view plane is dividend into 9 planes.

Above Left	Above	Above Right
Left	Window	Right
Below Left	Below	Below Right

Cohen-Sutherland Algorithm

- Assign 4bit code to each region.
- Each bit represents it position.
- Position order can be above, below, right, left.
- 1010 is code for above right region.

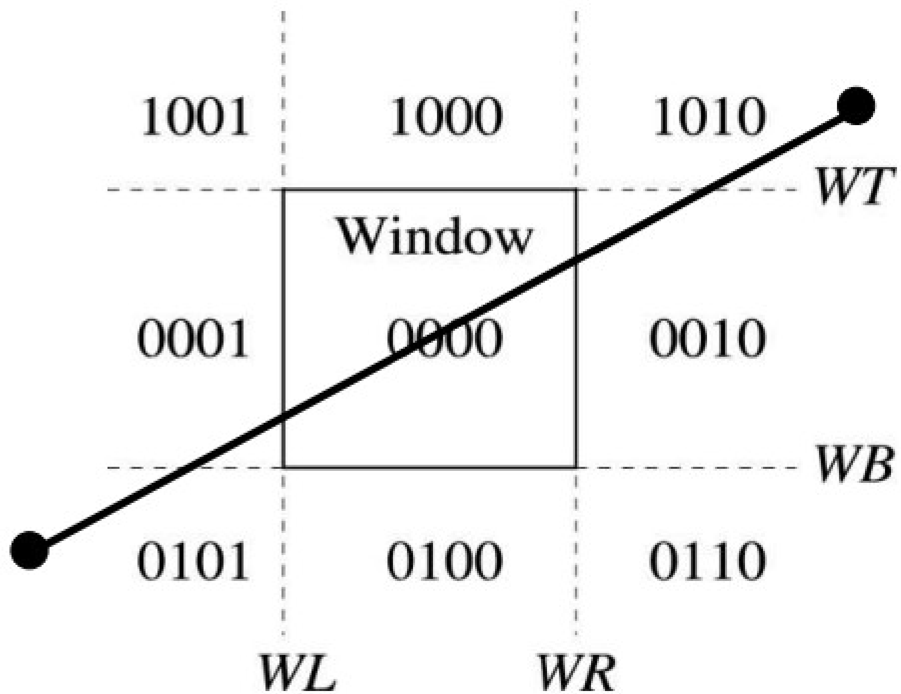
Above Left	Above	Above Right
Left	Window	Right
Below Left	Below	Below Right

1001	1000	1010
0001	0000	0010
0101	0100	0110

Above	Below	Right	Left

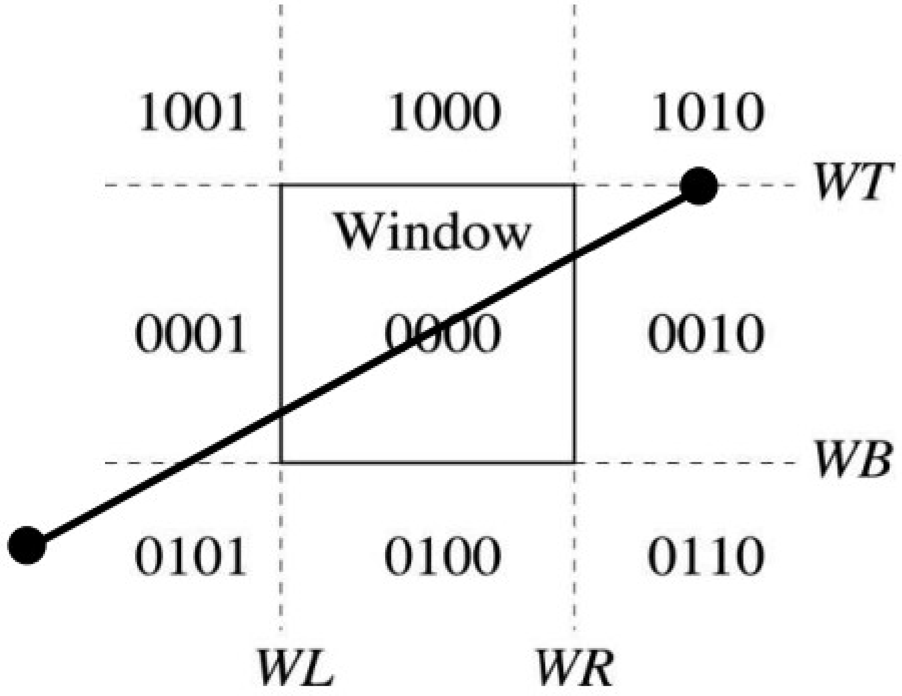
Cohen Sutherland

0101
1010



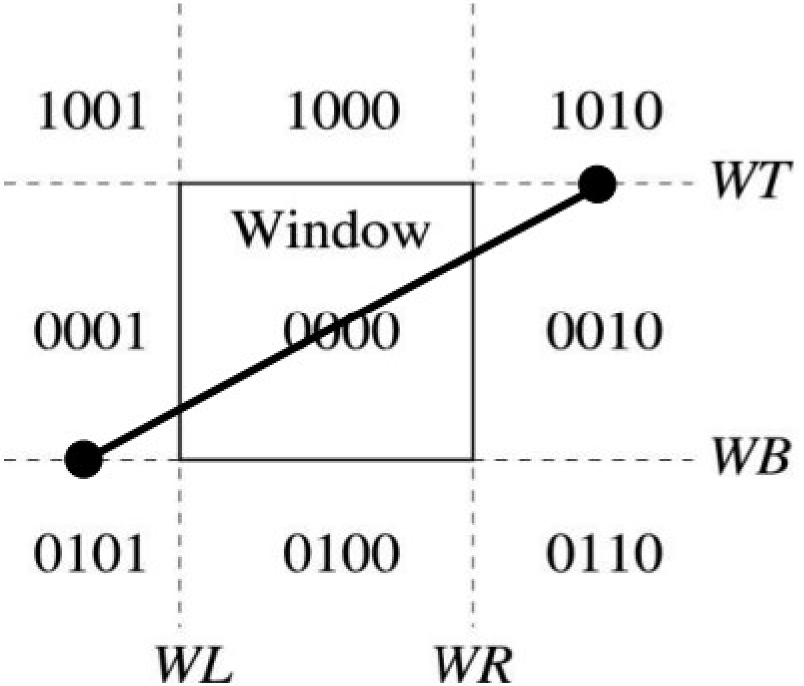
Cohen Sutherland

0101
0010



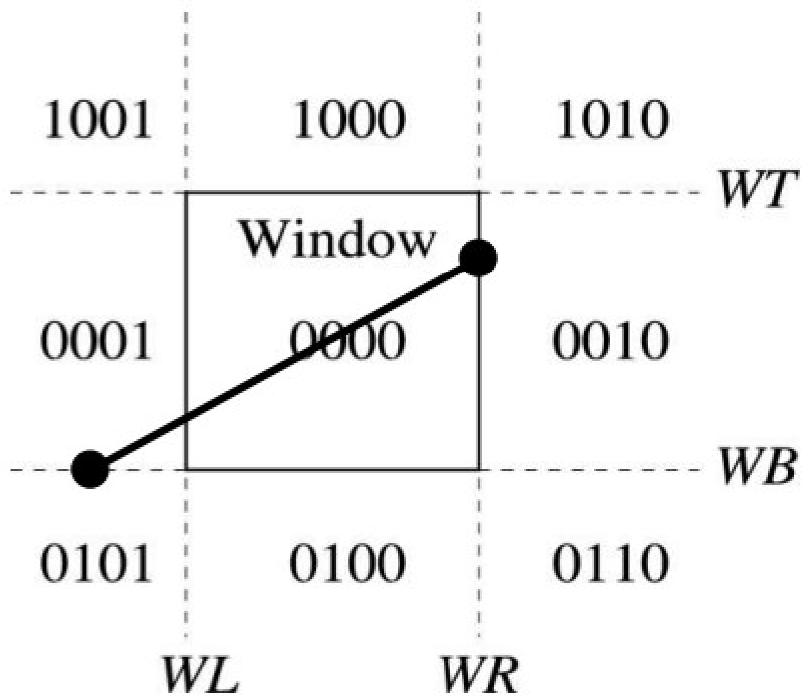
Cohen Sutherland

0001
0010



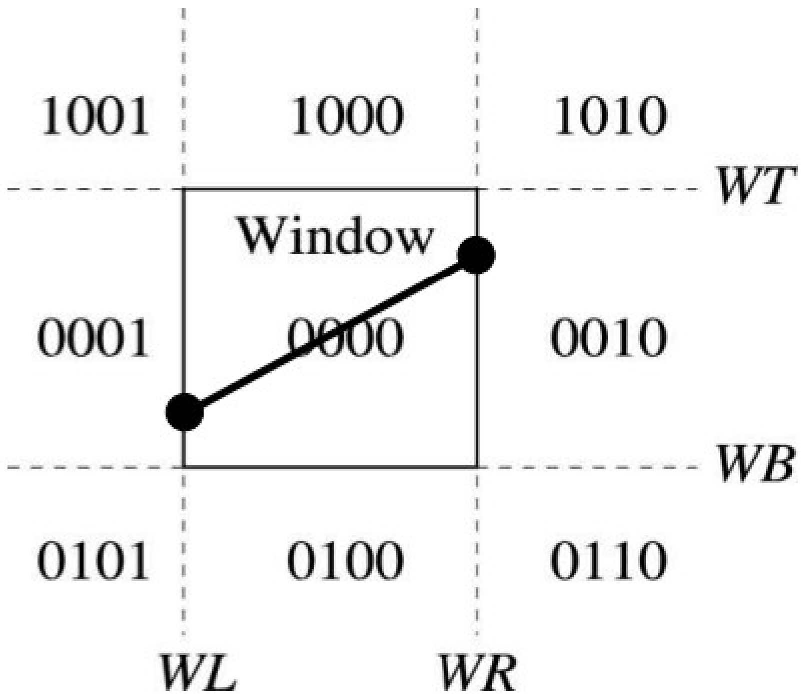
Cohen Sutherland

0001
0000



Cohen Sutherland

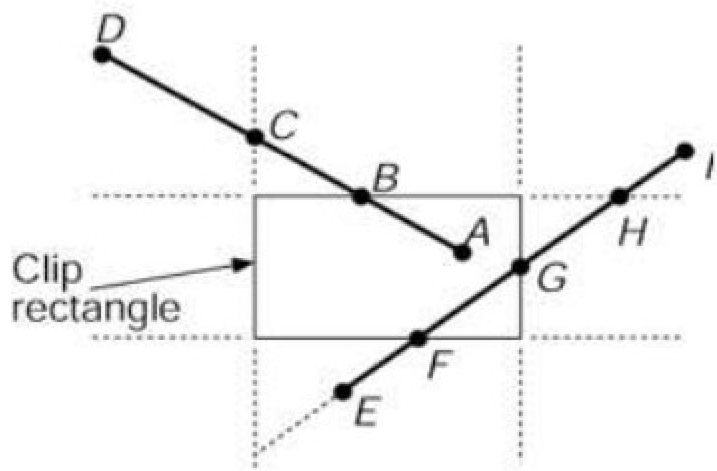
0000
0000



Cohen Sutherland Line Clipping Algorithm

Basic Idea:

- **First**, do easy test
  - *completely* inside or outside the box?
- **If no**, we will need to figure out how line intersects the box



Cohen-Sutherland Algorithm

- Step 1: Assign region codes for end points.
- Let P(x,y) be one of the end point and window is specified with (xmin,ymin,xmax,ymax)

1001	1000	1010
0001	0000	0010
0101	0100	0110

Above	Below	Right	Left

Cohen-Sutherland Algorithm

- Bit3 =sign(y-ymax)
- Bit2=sign(ymin-y)
- Bit1=sign(x-xmax)
- Bit0=sign(xmin-x)
-
- sign(x)=1, if x>0
- Otherwise 0.

1001	1000	1010
0001	0000	0010
0101	0100	0110

Above	Below	Right	Left

Cohen-Sutherland Algorithm

- Step 2: if both ends are 0, then line lies completely inside the window.
- Accept the line without clipping.
- Otherwise, if bitwiseAnd(code1,code2)!=0, then line completely lies outside the window.
- Discard it.

1001	1000	1010
0001	0000	0010
0101	0100	0110

Above	Below	Right	Left

Cohen-Sutherland Algorithm

1001	1000	1010
0001	0000	0010
0101	0100	0110

- Step 3: If none of these cases are true, then line is partially inside the window and we need to clip it.
- For clipping we need to calculate the intersection point for each boundary of window with the line.
- We can start with any boundary or any order.
- Let order be above, below, right and left.

Cohen-Sutherland Algorithm

1001	1000	1010
0001	0000	0010
0101	0100	0110

- If corresponding bits of two ends of line is not same then line intersects the boundary.
- For eg. 1001 and 0101, the bit3 is not same, so it intersects with top boundary of window.
- In the same way bit2 is also not same, so it intersects below boundary.
- Set the intersection point as the new end of the line and get is code.
- Note that we discard the line segment outside of window.

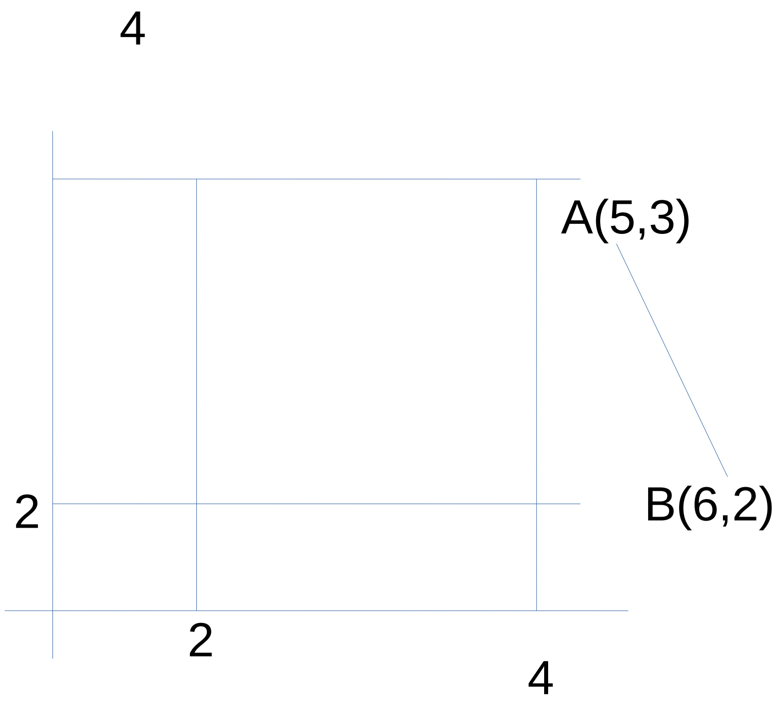
Cohen-Sutherland Algorithm

1001	1000	1010
0001	0000	0010
0101	0100	0110

- Now, compare new end points to see if they lie completely inside.
- If not, use other end point to check if it again intersects the boundaries of window.
-

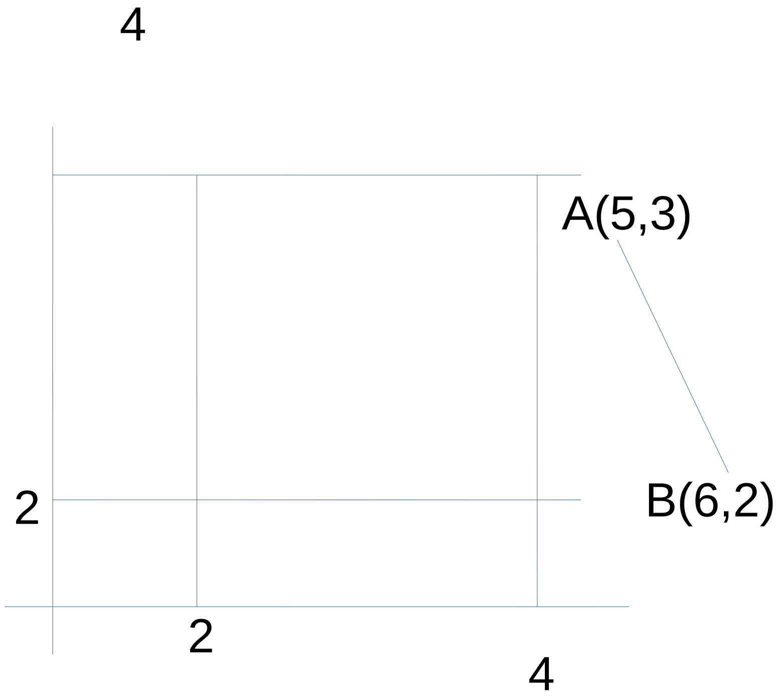
Cohen-Sutherland Algorithm

- Code For A
- Bit3=sign(3-4)=0
- Bit2=sign(2-3)=0
- Bit1=sign(5-4)=1
- Bit0=sign(2-3)=0
-
- Similarly code for B
- 0010



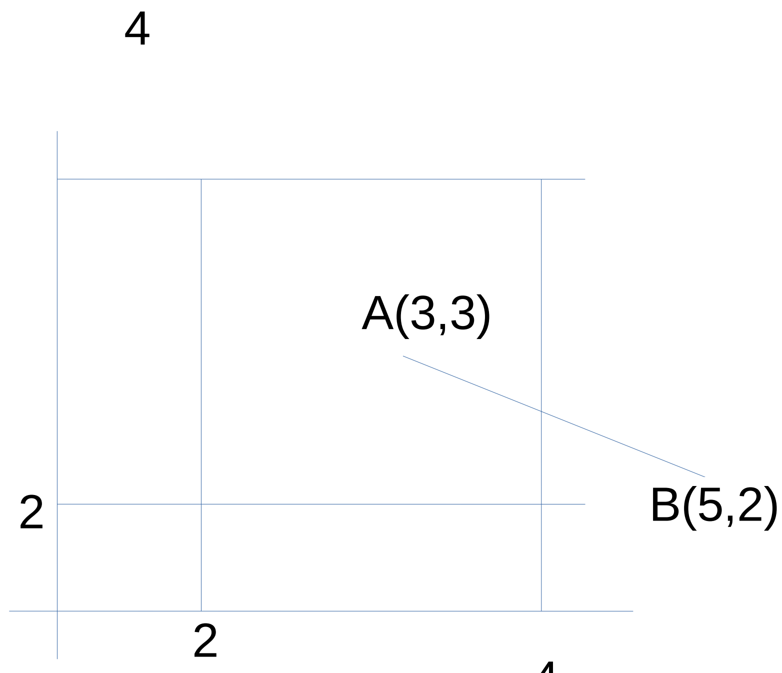
Cohen-Sutherland Algorithm

- Both A and B are not 0.
- So, we need to check their bitwise operation result.
- Its 0010 Which is not 0.
- So line completely lies outside and discard it.



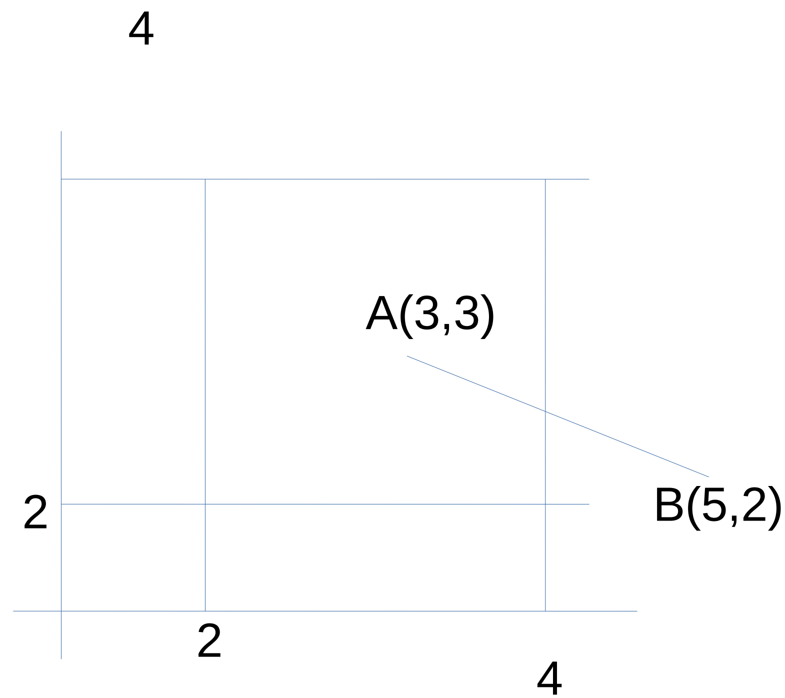
Cohen-Sutherland Algorithm

- Code for A
- bit3=sign(3-4)=0
- bit2=sign(2-3)=0
- bit1=sign(3-4)=0
- bit0=sign(2-3)=0
-
- Similarly code for B is
- 0010



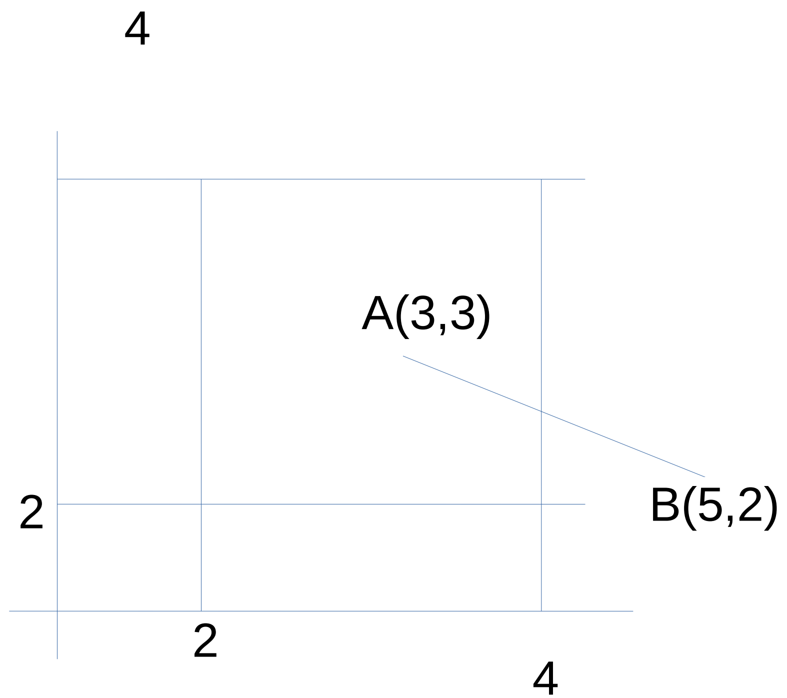
Cohen-Sutherland Algorithm

- Even A is 0 but B is not
- So, we need to check their bitwise AND.
- Its 0.
- It means line intersects the boundaries of window.



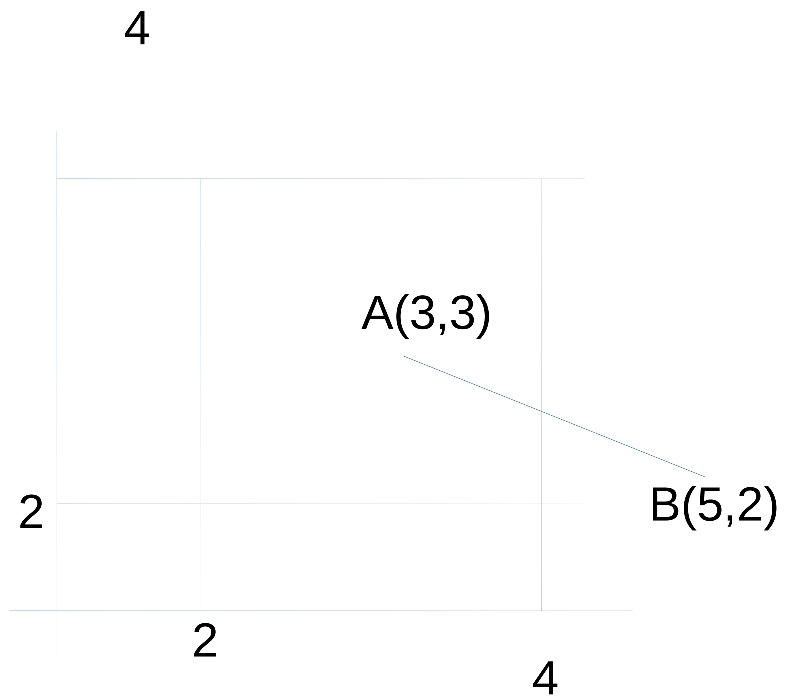
Cohen-Sutherland Algorithm

- Equation of AB is
- Y= - 1/2x + 9/2
- Bit3 of A and B is same.
- So AB does not intersect above boundary.
-



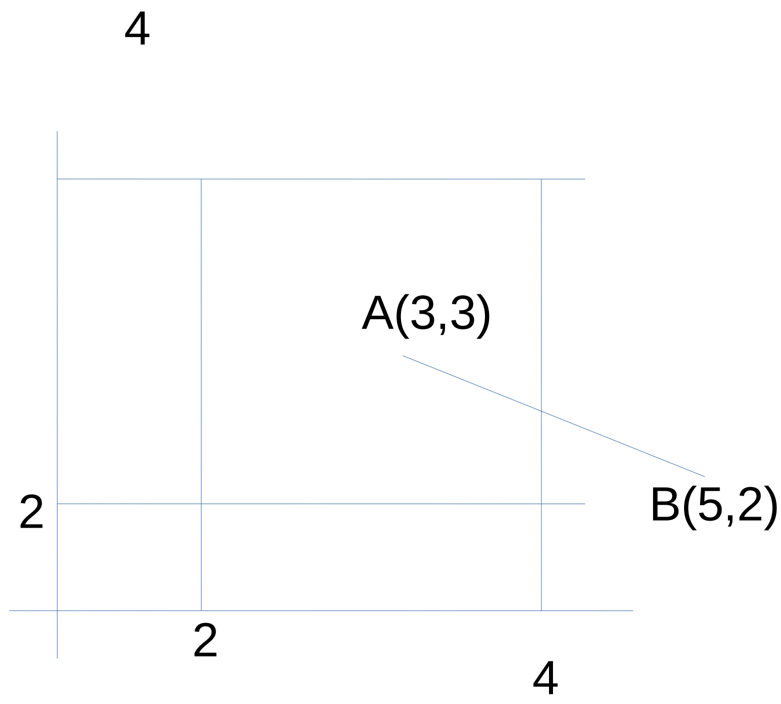
Cohen-Sutherland Algorithm

- Bit2 of A and B is same.
- So AB does not intersect below boundary.
-



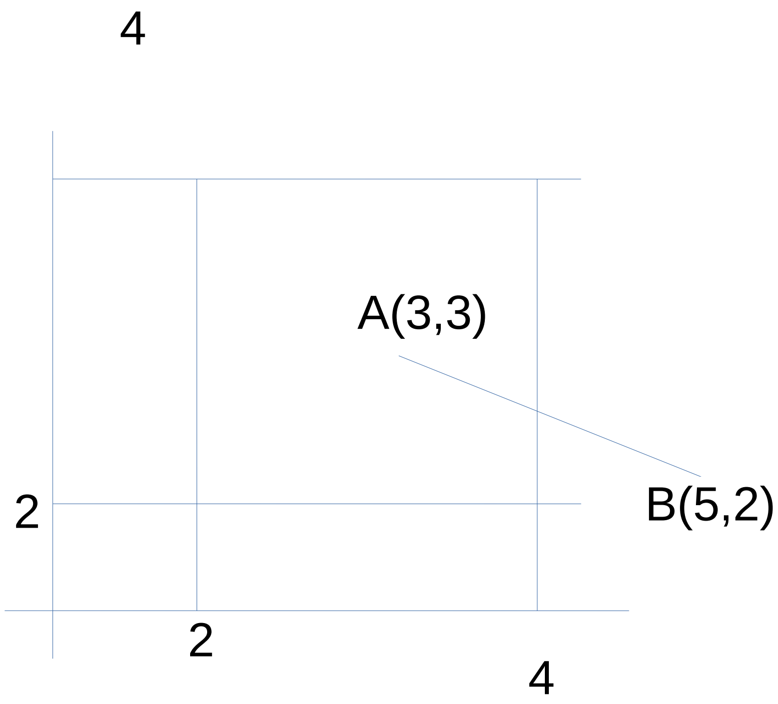
Cohen-Sutherland Algorithm

- Bit1 of A and B is different.
- So AB intersects right boundary.
- Equation of right boundary is x=4
- Point of intersection is (4,5/2)
- Outside line segment is clipped.



Cohen-Sutherland Algorithm

- Bit1 of A and B is different.
- So AB intersects right boundary.
- Equation of right boundary is x=4
- Point of intersection is (4,5/2)
- Outside line segment is clipped.



Liang-Barsky Line Clipping

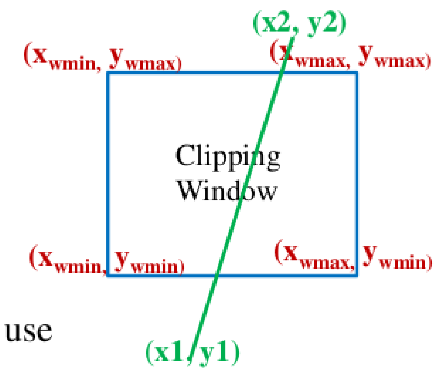
- Line clipping approach is given by the Liang and Barsky is faster then cohen-sutherland line clipping.
- Which is based on analysis of the **parametric equation of the line** which are,

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

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

if t=0 then x=x1,y=y1(starting point)

If t=1 then x=x2,y=y2(ending point)



- From point clipping we already know;
- To find weather given point is inside or outside the clipping window we use following inequality:

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

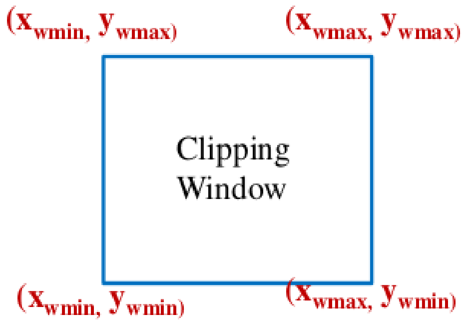
- $x_{wmin} \le x_1 + t\Delta x \le x_{wmax}, y_{wmin} \le y_1 + t\Delta y \le y_{wmax}$

Liang-Barsky Line Clipping

- $x_{wmin} \leq x_1 + t\Delta x \leq x_{wmax}, y_{wmin} \leq y_1 + t\Delta y \leq y_{wmax}$

Can be written as:

- $t\Delta x \geq x_{wmin} - x_1,$ *left*
- $t\Delta x \leq x_{wmax} - x_1,$ *right*
- $t\Delta y \geq y_{wmin} - y_1,$ *bottom*
- $t\Delta y \leq y_{wmax} - y_1,$ *top*



Represent all equations in ≤ form

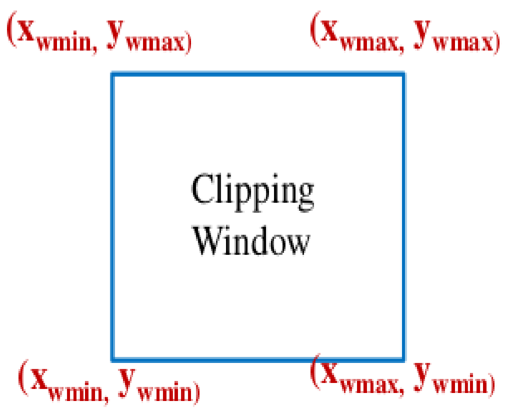
- $-t\Delta x \leq x_1 - x_{wmin}$
- $t\Delta x \leq x_{wmax} - x_1$
- $-t\Delta y \leq y_1 - y_{wmin}$
- $t\Delta y \leq y_{wmax} - y_1$

Liang-Barsky Line Clipping

- $-t\Delta x \le x_1 - x_{wmin}$
- $t\Delta x \le x_{wmax} - x_1$
- $-t\Delta y \le y_1 - y_{wmin}$
- $t\Delta y \le y_{wmax} - y_1$

General form for equations is: $t * p_k \le q_k$, where k=1,2,3,4(left, right, bottom & top edge)

- $p_1 = -\Delta x$, $q_1 = x_1 - x_{wmin}$, $t = q_1/p_1$
- $p_2 = \Delta x$, $q_2 = x_{wmax} - x_1$, $t = q_2/p_2$
- $p_3 = -\Delta y$, $q_3 = y_1 - y_{wmin}$, $t = q_3/p_3$
- $p_4 = \Delta y$, $q_4 = y_{wmax} - y_1$, $t = q_4/p_4$



Liang-Barsky Line Clipping Algorithm

Parametric definition of line

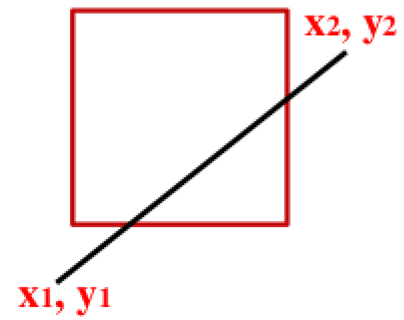
$x_{min} \le x_1 + t\,\Delta x \le x_{max}$

$y_{min} \le y_1 + t\,\Delta y \le y_{max}$

$\Delta x = x_2 - x_1$

$\Delta y = y_2 - y_1$

$0 \le t \le 1$



Initializations

- Set viewport points as x_{min}, x_{max}, y_{min} and y_{max}
- Set the line with 2 points $A(x_1,y_1)$ and $B(x_2,y_2)$
- Set line intersection parameters t_{entry} $(t_1) = 0.0$ and $t_{leaving}$ $(t_2) = 1.0$

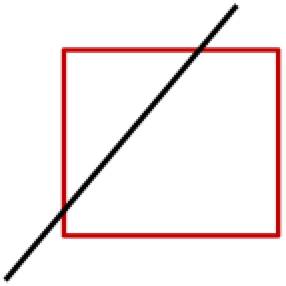
Calculations

- Find $dx = x_2 - x_1$ and $dy = y_2 - y_1$
- Update t_1 or t_2 depending upon dx or dy

Liang-Barsky Line Clipping Algorithm

Step 1 :

Set line intersection parameters t_{entry} $(t_1) = 0.0$ and $t_{leaving}$ $(t_2) = 1.0$



Step 2 :

Obtain p_k and q_k for $k = 1: 4$

Edge & K value	p	q	t
Left **k=1**	p1 = -dx	q1 = x0 - xwmin	t = q1/p1
Right **k=2**	p2 = dx	q2 = xwmax - x0	t = q2/p2
Bottom **k=3**	p3 = -dy	q3 = y0 - ywmin	t = q3/p3
Top **k=4**	p4 = dy	q4= ywmax - y0	t = q4/p4

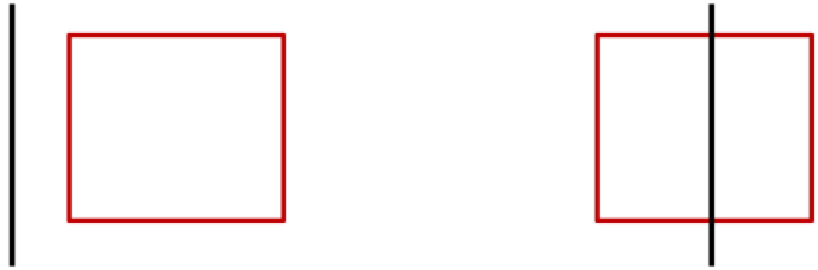
Liang-Barsky Line Clipping Algorithm

Step 3 :

If pk = 0, the line is parallel to the corresponding clipping boundary.

a) If pk = 0 and qk<0, the line is completely outside the boundary.

b) If pk = 0 and qk>=0, the line is inside the parallel clipping boundary.



(a) **(b)**

Liang-Barsky Line Clipping Algorithm

Step 4 :

Using p_k and q_k ($k = 1:4$), find if the line can be rejected or intersection parameters (t_1 and t_2) must be adjusted.

- if $p_k < 0$, update t_1 as ***max***$[0, q_k/p_k]$ where $k = 1:4$
- if $p_k > 0$, update t_2 as ***min***$[1, q_k/p_k]$ where $k = 1:4$

After the update,

- If $t_1 > t_2$, reject the line
- If $t_1 > 0$, calculate new values of x1, y1
- If $t_2 < 1$, calculate new values of x2, y2

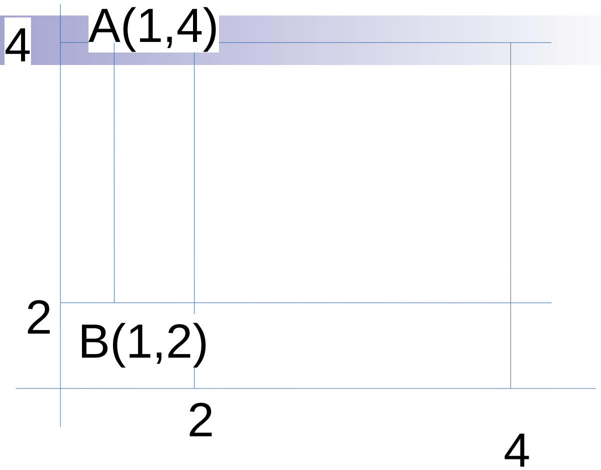
Advantages of Liang-Barsky Line Clipping

1. More efficient.
2. Only requires one division to update t_1 and t_2.
3. Window intersections of line are calculated just once.

Liang-Barsky

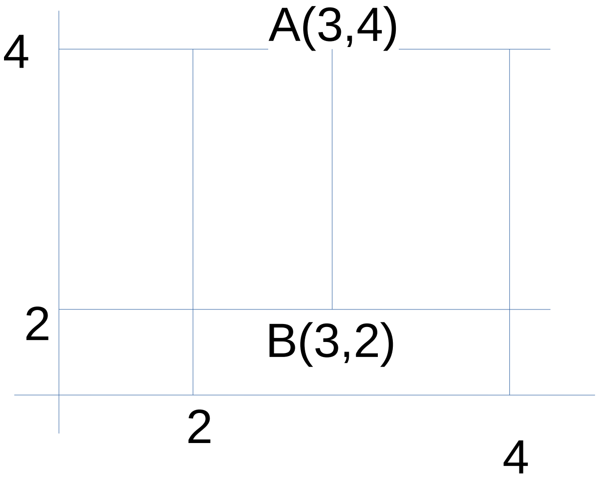
- Input: set of edges, wxmin, wymin, wxmax, wymax
- Output: Clipped edges
- For each edges repeat
- t1=0, t2=1
- dx=x2-x1, dy=y2-y1
- p1=-dx, p2=dx, p3=-dy, p4=dy
- If any of p_i=0 and any of q_i<0 then
- Line is completely outside, discard it.
- continue
- for each pi<0, t1=max(0,qi/pi)
- For each p_i>0, t2=min(1,q_i/p_i)
- If t1>t2, discard the line.
- If t1>0, x1=x1+t1dx, y1=y1+t1dy
- If t2<1, x2=x2+t2dx, y2=y2+t2dy

Liang-Barsky



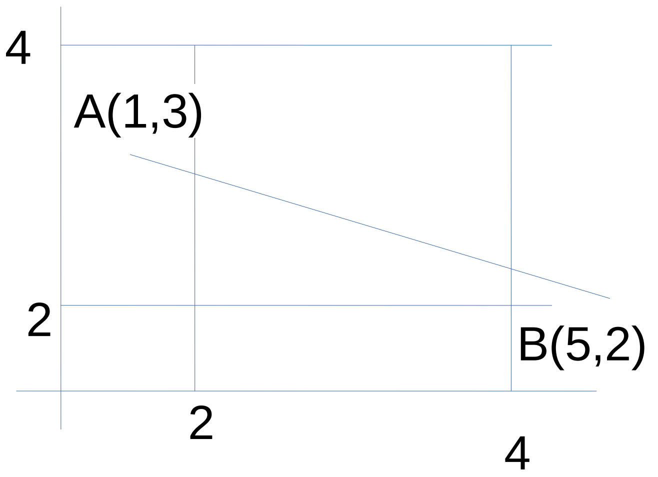
- t1=0, t2=1
- x1=1, y1=2, x2=1, y2=4
- Wxmin=2, wymin=2,wxmax=4, wymax=4
- dx=0, dy=2
- p1=-dx=0, p2=0, p3=-2, p4=2
- p1 and p2 = 0 and q3=0 so line is parallel to vertical boundaries.
- Q1=-1, q2=3, q3=0, q4=2
- Here q1<0, so line is complete outside.
- Discard the line without any further steps

Liang-Barsky



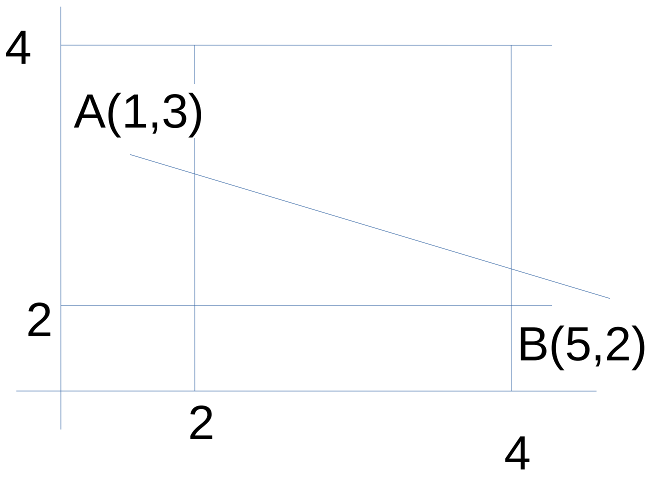
- t1=0, t2=1
- X1=3, y1=2, x2=3, y2=4
- Wxmin=2, wymin=2,wxmax=4, wymax=4
- dx=0, dy=2
- p1=0, p2=0, p3=-2, p4=2
- p1 and p2 is 0, so line is parallel to vertical boundaries
- Q1=1, q2=1, q3=0, q4=2
- Here q1, q2, q3, and q4>=0, so line is complete is inside.
- t1=max(0,0)=0
- t2=min(1,1)=1

Liang-Barsky



- t1=0, t2=1
- X1=1, y1=3, x2=5, y2=2
- Wxmin=2, wymin=2,wxmax=4, wymax=4
- dx=4, dy=-1
- p1=-4, p2=4, p3=1, p4=-1
- P1, p2, p2, p3, p4 are not 0
- so line is not parallel any boundaries
- Q1=-1, q2=3, q3=1, q4=1
- Here p1 and p4<0,
- t1=max(0,1/4,-1)=1/4
-

Liang-Barsky



- And p2, p3>0, so
- t2=min(1,3/4,1)=3/4
- Here, t1<t2, we do not discard the line.
- T1>0, so update x1 and y1
- x1=x1+t1dx=1+1/4*4=2
- y1=y1+t1dy=3+1/4* (-1)=11/4
- T2<1, so update x2, y2
- x2=x1+t2dx=1+3/4*4=4
- y2=y1+t2dy=3+3/4*(-1)=9/4

Polygon Clipping

- Clipping window as four boundaries.
- Each boundary is used as clipper.
- Each clipper takes set of vertices as input
- Generate output another set of vertices which is successively fed to other clippers.
- Order of clipper to test can be arbitrary.
- Let it be Left, Right, Bottom, and Top clipper
- It means firstly all vertices are fed to left clippers

Polygon Clipping

- For each pair of vertices (vi,vj), it checks if it is inside the clipper or outside.
- If vi is inside and vj is outside, then it keeps vi and intersection of clipper and edge joining vi and vj.
- If both indices are inside, keeps vj
- If vi is outside and vj is inside, the keeps intersection and vj
- If both vertices are outside, discard them.

Polygon Clipping

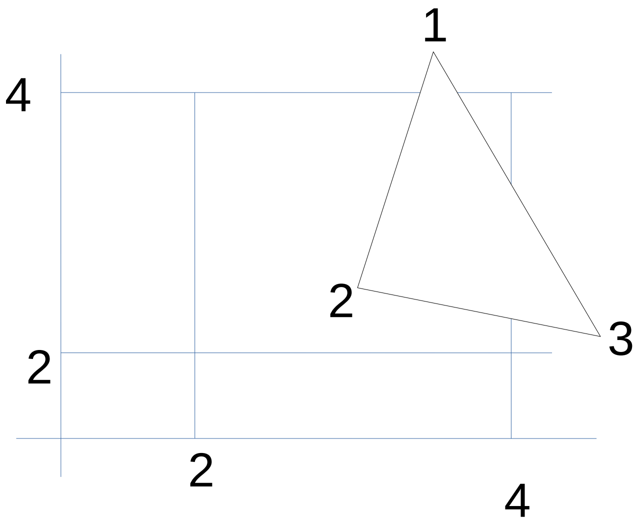
- Terms inside and outside are interpreted differently for each clipper.
- Left Clipper: vertex on its right side is inside otherwise outside
- Right Clipper: vertex on its left is inside otherwise outside.
- Top Clipper: vertex below means inside, else outside.
- Below Clipper: vertex above is inside, else outside.

Polygon Clipping

- For all clippers, vertices on the boundary are inside.
-

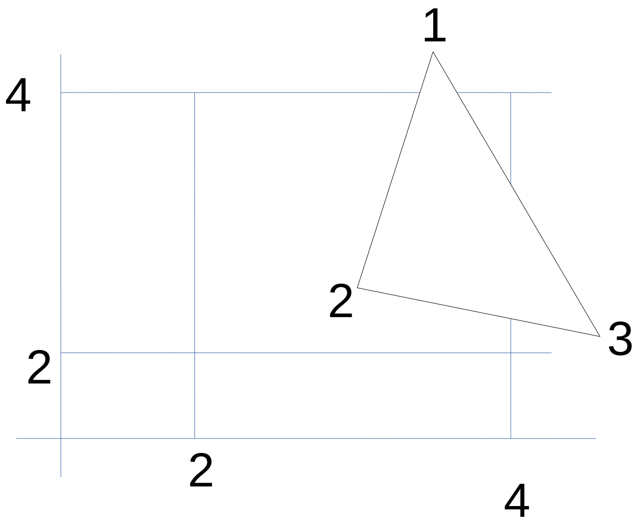
Polygon Clipping

- For all clippers, vertices on the boundary are inside.
- Set of vertices {1,2,3}
- First, these vertices are passed to left clipper.
- Left clipper checks edges such as {1,2}, {2,3} and {3,1}
- For edge {1,2}, both vertices are inside
- So, result= {2}



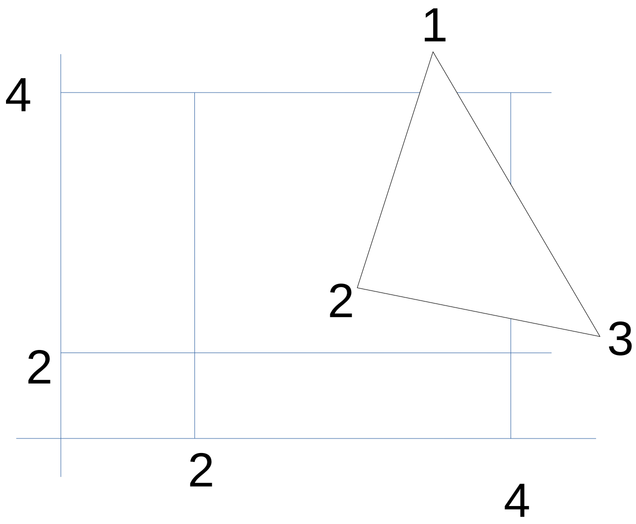
Polygon Clipping

- For edge {2,3}, both vertices are inside.
- Result={2,3}
- For edge {3,1}, both vertices are inside,
- Result={2,3,1}



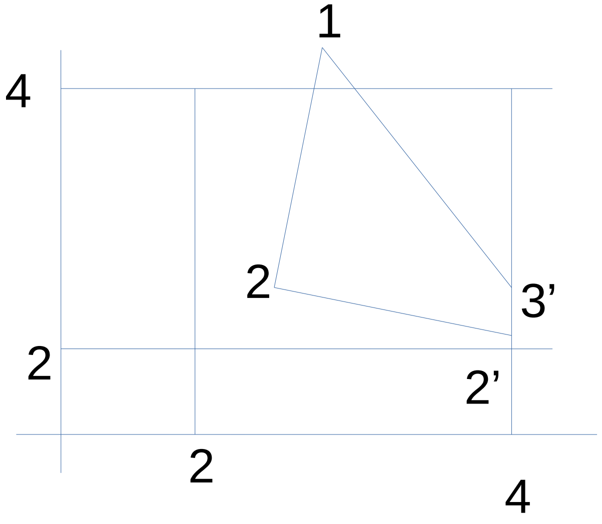
Right Clipper

- Input vertices are {2,3,1}
- Edges are {2,3},{3,1},{1,2}
- Result={}
- For edge {2,3}, 2 is inside but 3.
- Let intersection of clipper and edge be 2’
- Result={2,2’}



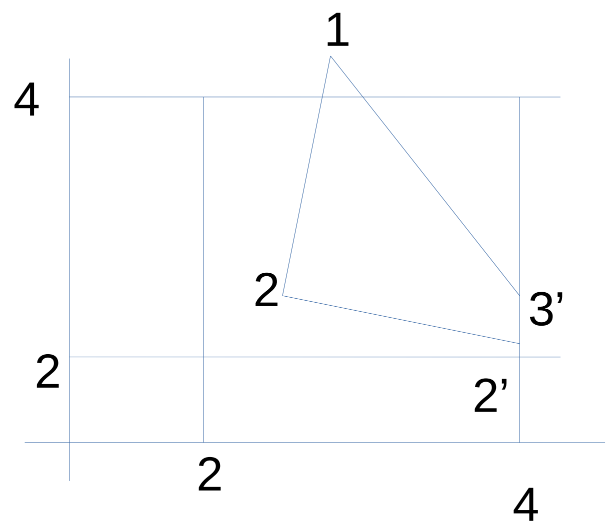
Right Clipper

- For edge {3,1}, 3 is outside and but 1
- Let 3’ be intersection of the edge and clipper.
- Result={2,2’,3’,1}
- For edge {1,2},
- Both vertices are inside.
- Result={2,2’,3’,1}



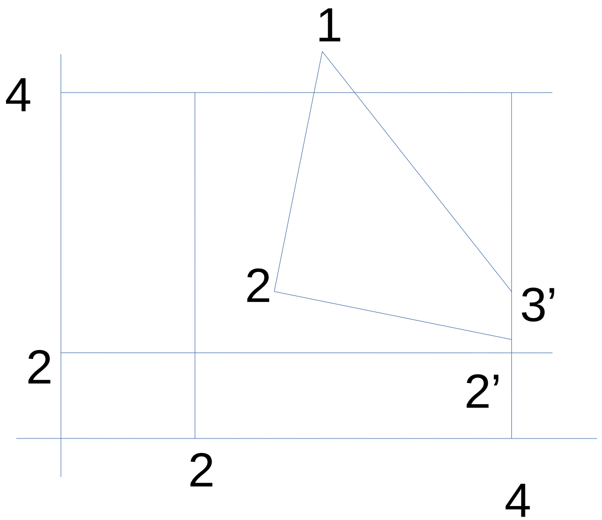
Bottom Clipper

- Input vertices {2,2’,3’,1}
- Edges {2,2’},{2’,3’},{3’,1},{1,2}
- Result={}
- For edge {2,2’},
- Both vertices are inside.
- Result={2’}



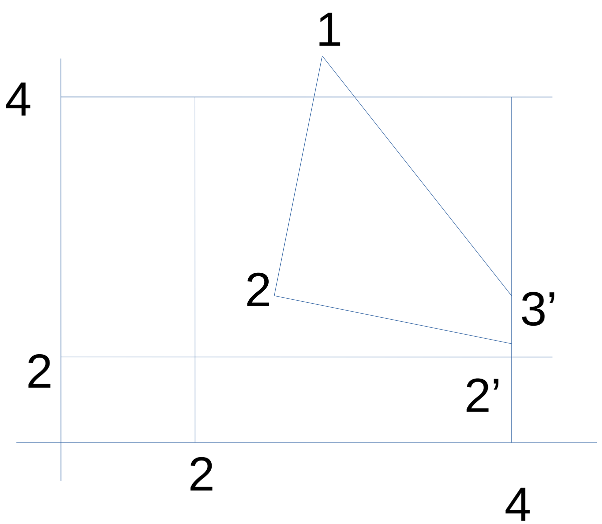
Bottom Clipper

- For edge {2’,3’},
- Both vertices are inside.
- Result={2’,3’}
- For edge {3’,1}
- Both vertices are inside
- Result={2’,3’,1}
- For edge {1,2}
- Result={2’,3’,1,2}



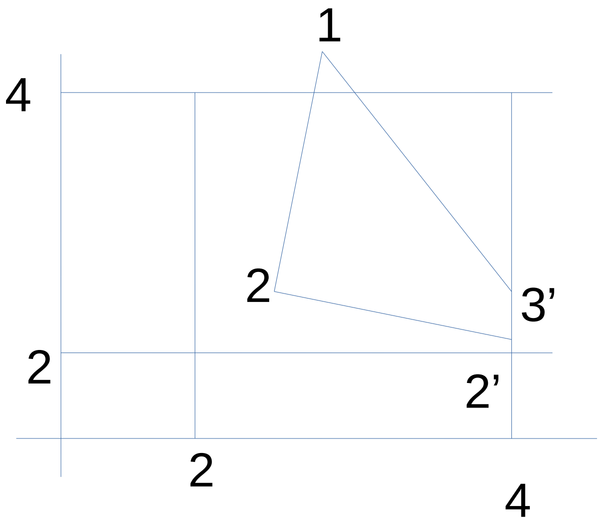
Top Clipper

- vertices {2’,3’,1,2},
- Edges {2’,3’}, {3’,1}, {1,2}, {2,2’}
- Result={}
- For edge {2’,3’}, both vertices are inside.
- Result={3’}
- For edge {3’,1}, 3’ is inside but 1
- Let intersection be 3’’
- Result={3’,3’’}



Top Clipper

- For {1,2}, 1 is outside but 2
- Let 1’ be intersection
- Result={3’,3’’,1’,2}
- For edge {2,2’}, both vertices are inside.
- Result={3’,3’’,1’,2,2’}
-



Clipped Polygon is