COMPUTER GRAPHICS

CH3 – Basic Transformation and Clipping

Outline

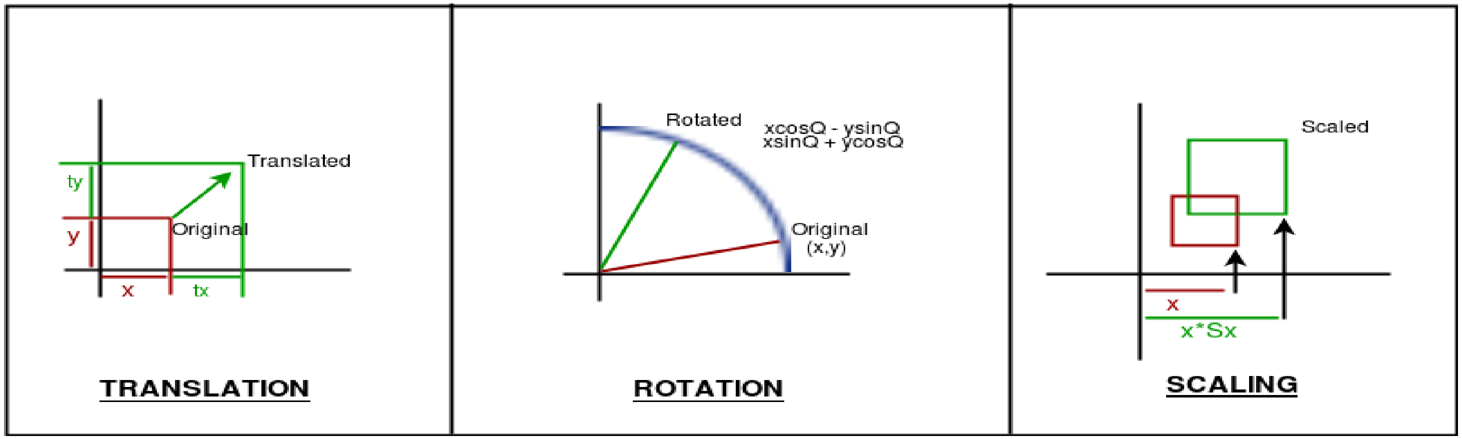
- Transformations and Clipping
  - Basic Transformations
    - Translation, Scaling, Rotation
  - Composite Transformation
  - Clipping Algorithms

Introduction

- Changing of an object after creation in terms of position or size is called **Transformation.**
- Using output primitives and their **attributes**, we can create variety of pictures and graphs - also a need for **altering or manipulating** displays.
- Animations are produced by moving the "camera" or the objects in a scene along **animation** paths.
- Changes in ***orientation***, ***size***, and ***shape*** are accomplished with ***geometric transformations*** that alter the coordinate descriptions of objects.
- The **basic geometric transformations** are ***translation***, ***rotation***, and ***scaling***.
- Other **transformations** that are often applied to **objects** include ***reflection*** and ***shear***.

Basic Transformations

- Here, we first discuss general procedures for applying ***Translation***, ***Rotation***, and ***Scaling*** parameters to ***reposition*** and ***resize*** two-dimensional objects.
- Then, we consider how **transformation equations** can be expressed in a more convenient matrix formulation that **allows efficient combination** of object transformations.
- There are three basic kinds of Transformations in Computer Graphics: 1. Translation 2. Rotation 3. Scaling



1. Translation

- A **translation** is applied to an object by **repositioning** it along a straight-line path from one coordinate location to another.
- It will **shift** the object from one position to an other position.
  - The point (x, y) is translated to the point (x', y') by adding the translation distances (t_x, t_y):

$$x' = x + t_x$$
$$y' = y + t_y$$

  - The above equations can be expressed in matrix form by:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

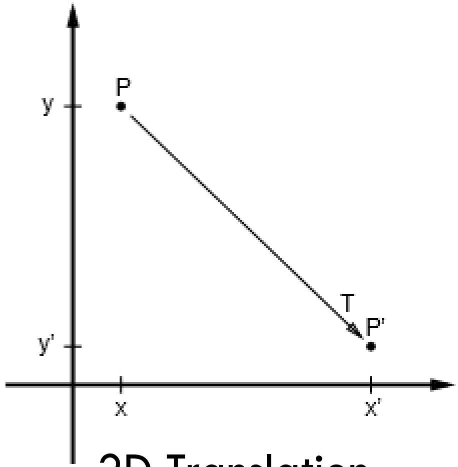
  - So, the translation of the **two dimensional vector**

P by T into P' is given by:

$$P' = P + T$$

where

$$P' = \begin{bmatrix} x' \\ y' \end{bmatrix}, \quad P = \begin{bmatrix} x \\ y \end{bmatrix}, \quad \text{and} \quad T = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$



2D Translation

Examples

- Suppose we want to shift a point with coordinates at A(30,100) and distance along x-axis is 10 units and 20 units along y-axis Using translation:

Give

- $t_x = 10$ and $t_y = 20$
- Original Coordinate A(30,100)

Solution

- New coordinates $A'(x_2, y_2)$:
- $x_2 = x + t_x = 30 + 10 = 130$
- $y_2 = y + t_y = 100 + 20 = 120$
- The point will be shifted to **A′ (130, 120)**

Examples

- Square (0,0) (2,0)(0,2)(2,2) translate with tx=2,ty=3

- Give triangle with vertex A(2,2),B(10,2) and C(5,5) Translate with t_x =5, t_y=6

2. Scaling

- A **scaling** transformation alters the size of an object by scaling factors $\boldsymbol{s_x}$ and $\boldsymbol{s_y}$.
  - The vertex (x, y) is scaled into the vertex (x', y') by **multiplying** it with the scaling factors $\boldsymbol{s_x}$ and $\boldsymbol{s_y}$:

$$x' = s_x x$$
$$y' = s_y y$$

  - This can be expressed in matrix form by:

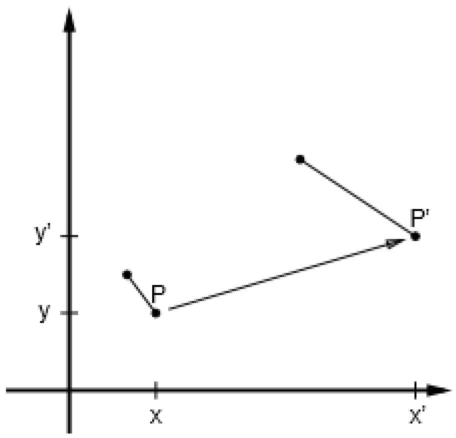
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

or

$$P' = S \cdot P$$

where

$$S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

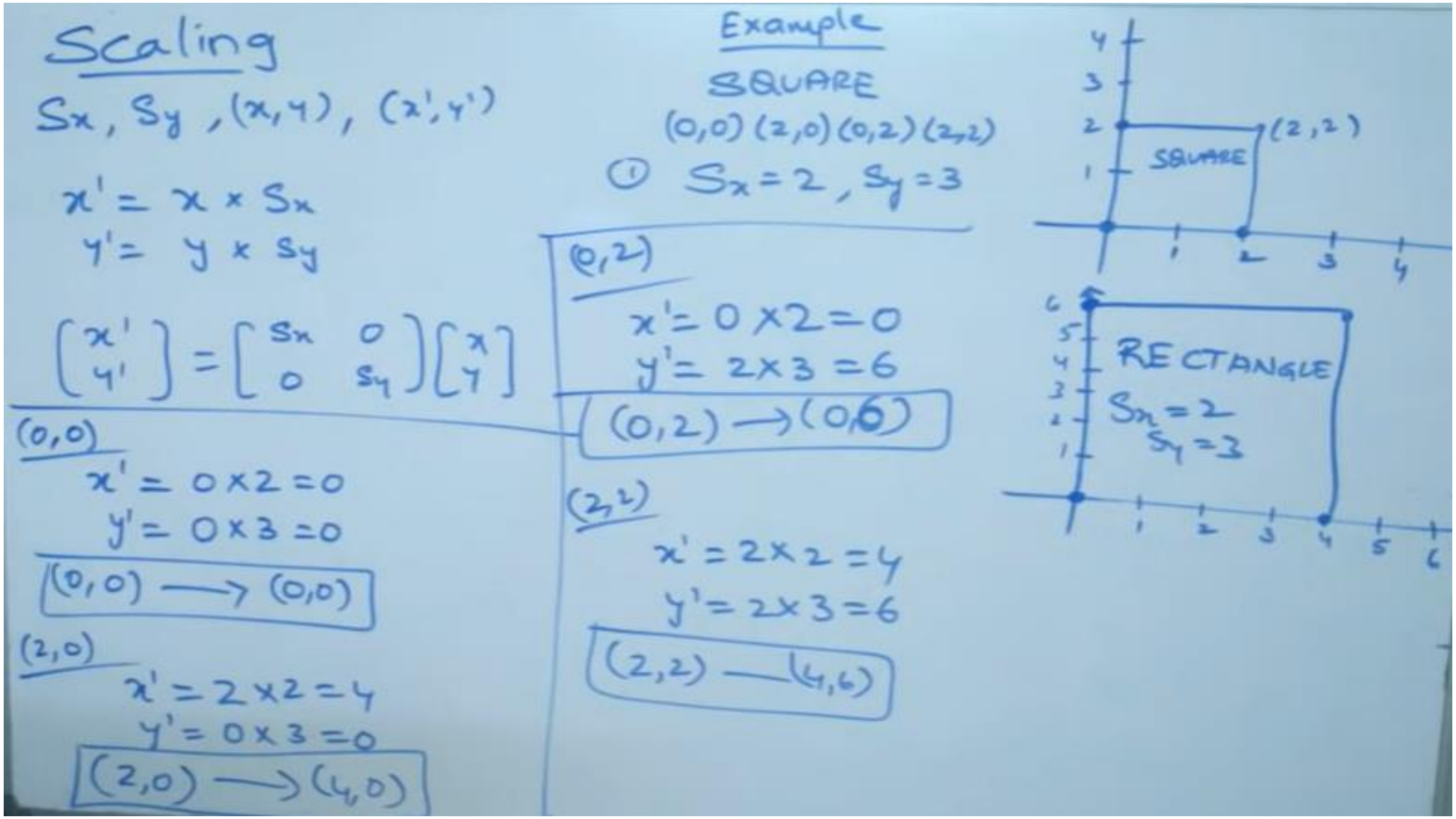


2D Scaling

- If $s_x = s_y$ the transformation is called a ***uniform scaling.***
- If $s_x \neq s_y$ it is called a ***differential scaling.***

Example

- Square (0,0)(2,0)(0,2)(2,2) scale with sx=2 and sy=3



3. Rotation

- Rotation is a process of changing the **angle** of the object.
- Rotation can be **clockwise** or **anticlockwise.**
- The point (x, y), or (r, φ) in polar **coordinates**, is rotated anti-clockwise about the origin by θ into the point (x', y'), or $(\boldsymbol{r}, \boldsymbol{\varphi} + \boldsymbol{\theta})$.
  - So

$$x' = rcos(\varphi + \theta) = r(\cos\varphi\cos\theta - \sin\varphi\sin\theta)$$
$$y' = rsin(\varphi + \theta) = r(\cos\varphi\sin\theta + \sin\varphi\cos\theta)$$

  - Now,

$$x = rcos\varphi \quad \text{and} \quad y = rsin\varphi$$

  - Therefore

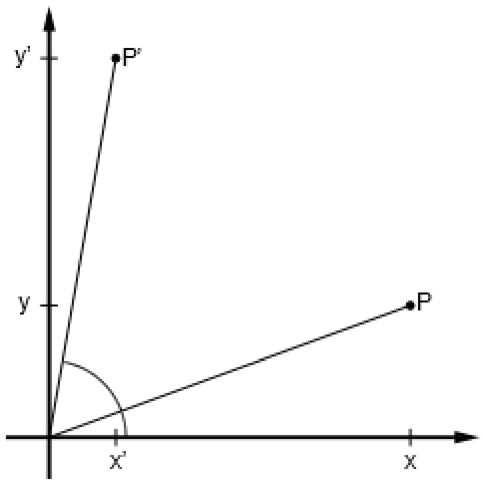
$$x' = xcos\theta - y\sin\theta$$
$$y' = xsin\theta + y\cos\theta$$

  - This can be expressed by:

$$\boldsymbol{P}' = \boldsymbol{R} \cdot \boldsymbol{P}$$

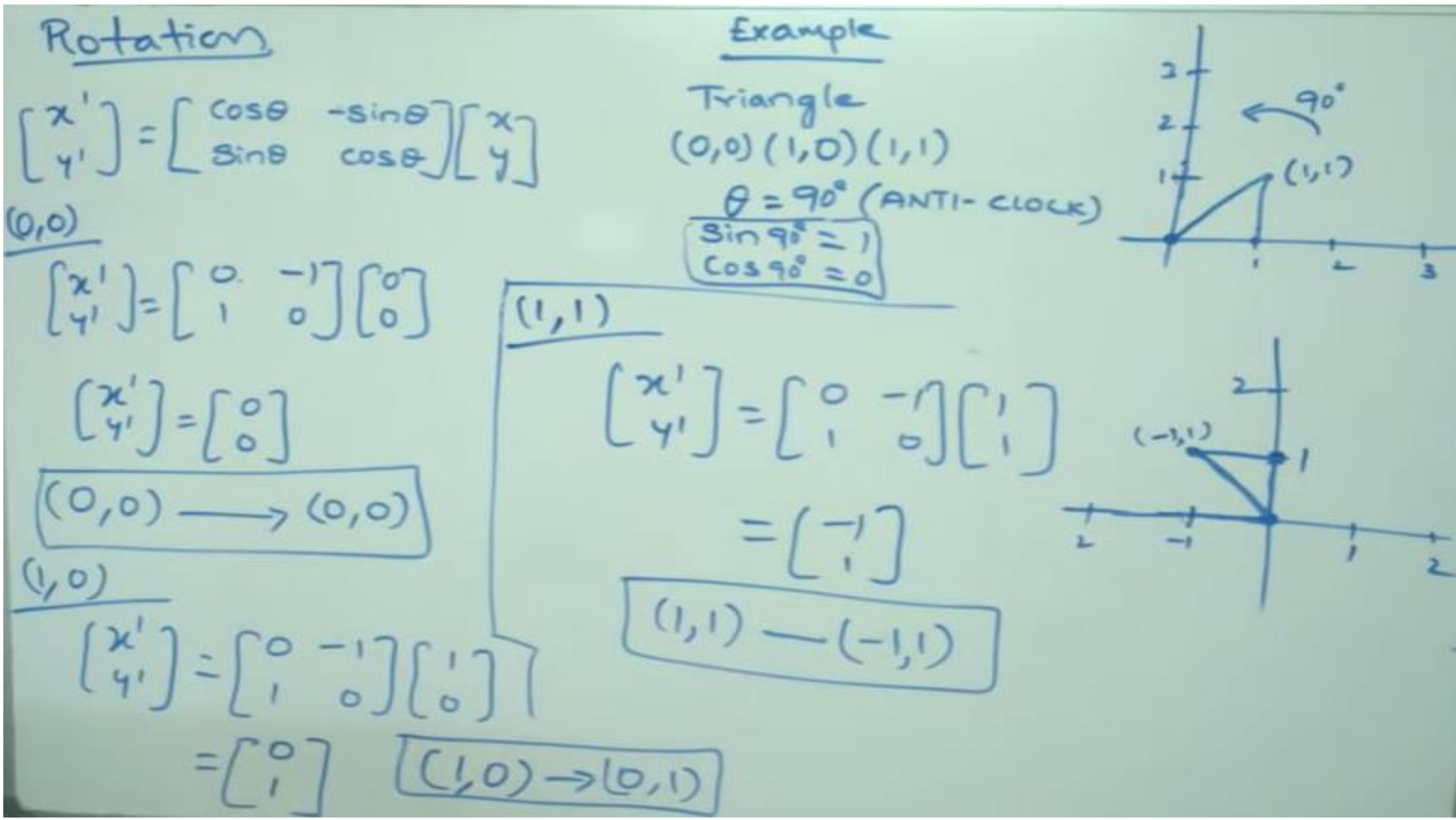
where

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

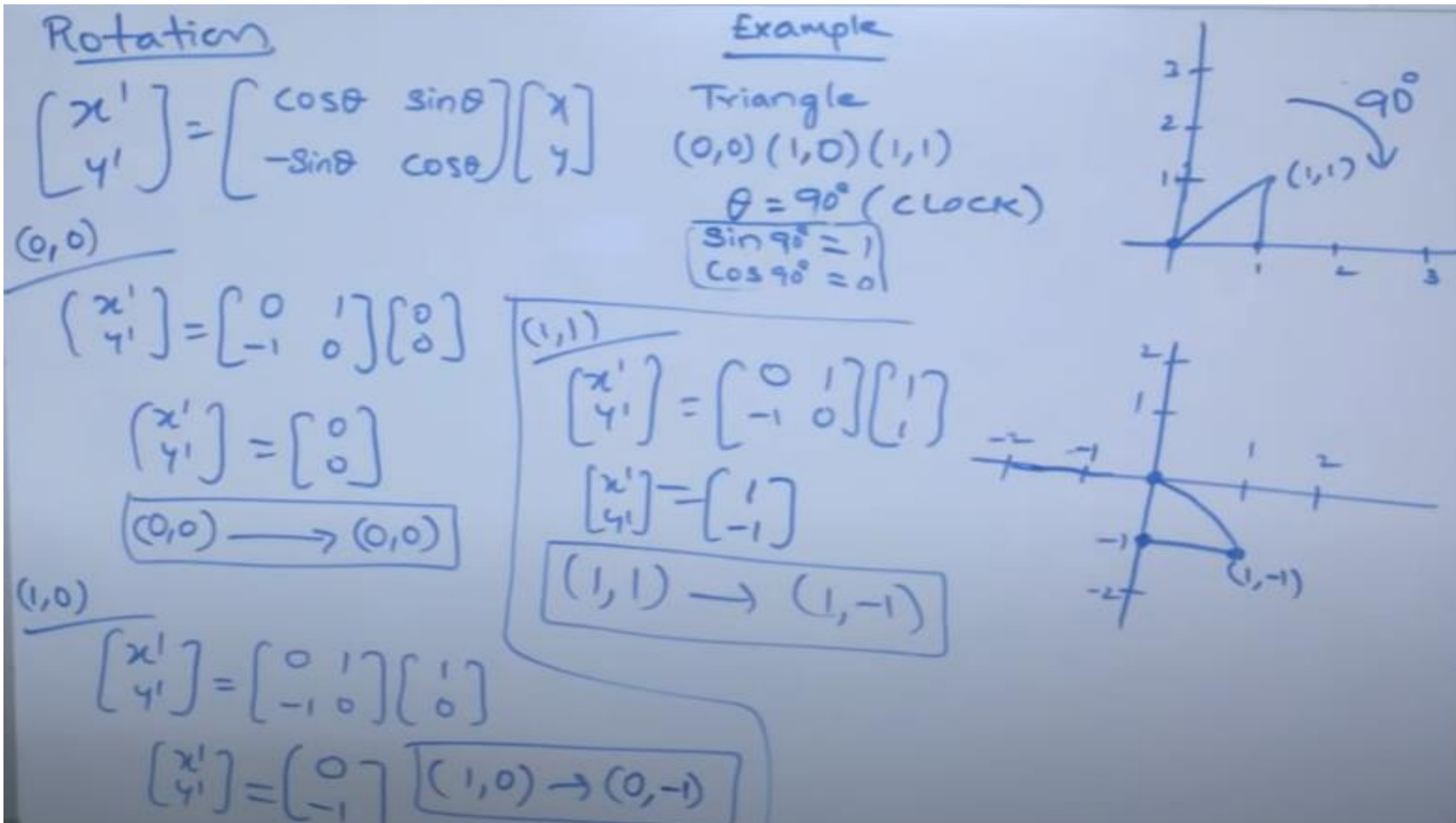


2D Rotation

❖ **Triangle (0,0) (1,0) and (1,1) rotate 90 degree anticlockwise**



❖ **Triangle (0,0) (1,0) and (1,1) rotate 90 degree clockwise**



Example

- One triangle is given (2,2) (8,2) (5,5) Rotate the triangle 90 degree.

Example 1. One triangle is given (2,2) (8,2) (5,5).

Rotate the triangle 90°.

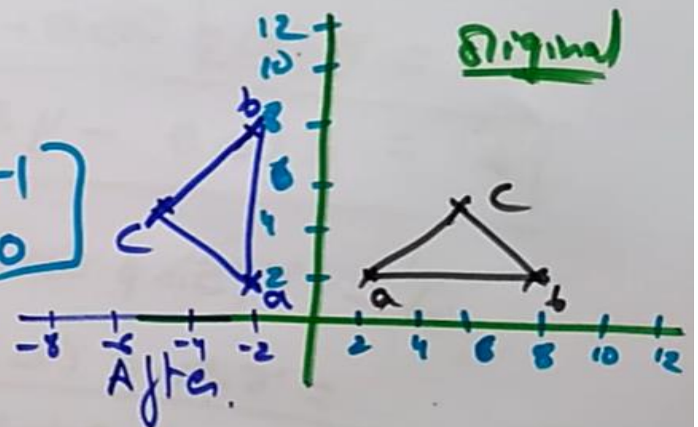
here $\theta = 90^\circ$

$$R = \begin{bmatrix} \cos 90^\circ & -\sin 90^\circ \\ \sin 90^\circ & \cos 90^\circ \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

$$A' = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} -2 \\ 2 \end{bmatrix}$$

$$B' = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 8 \\ 2 \end{bmatrix} = \begin{bmatrix} -2 \\ 8 \end{bmatrix}$$

$$C' = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 5 \\ 5 \end{bmatrix} = \begin{bmatrix} -5 \\ 5 \end{bmatrix}$$



Example

- Rotate line AB whose endpoints are A (2, 5) and B (6, 12) about origin through a 30° clockwise direction.

Solution: For rotation in the clockwise direction. The matrix is

$$R = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

Step1: Rotation of point A (2, 5). Take angle 30°

$$R = \begin{bmatrix} \cos 30^\circ & -\sin 30^\circ \\ \sin 30^\circ & \cos 30^\circ \end{bmatrix}$$

$$= [2, 5] \begin{bmatrix} \cos 30^\circ & -\sin 30^\circ \\ -\sin 30^\circ & \cos 30^\circ \end{bmatrix}$$

$$= [2, 5] \begin{bmatrix} .866 & -0.5 \\ .5 & .866 \end{bmatrix}$$

$$= [2 * .866 + 5 * .5 \qquad 2 * (-.5) + 5 (.866)]$$

$$= [(1.732 + 2.5 \qquad -1 + 4.33]$$

$$= [4.232 \qquad 3.33]$$

A point (2, 5) become (4.232, 3.33)

Step2: Rotation of point B (6, 12)

$$= [6 \ \ 12] \begin{bmatrix} \cos 30^\circ & -\sin 30^\circ \\ \sin 30^\circ & \cos 30^\circ \end{bmatrix}$$

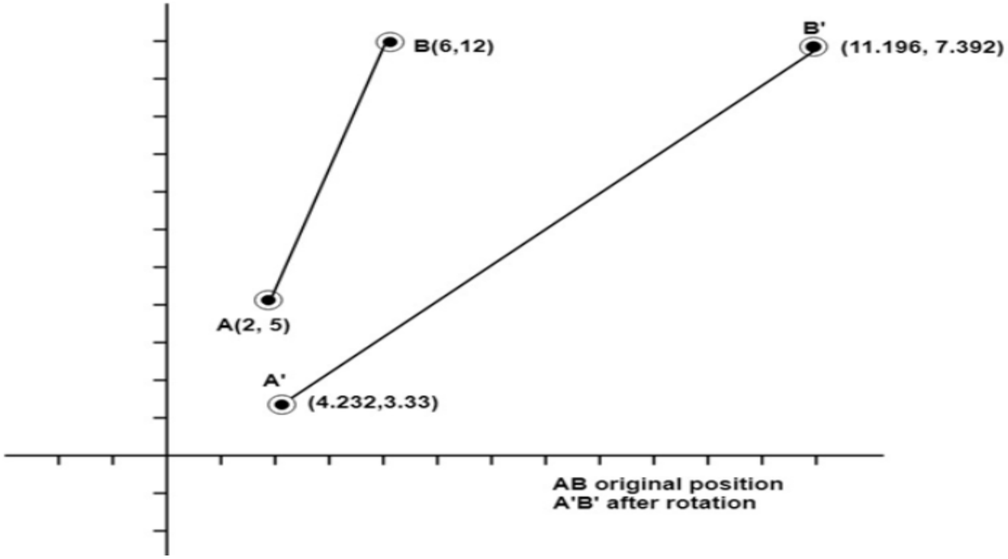
$$= [6 \ 12] \begin{bmatrix} .866 & -0.5 \\ .5 & .866 \end{bmatrix}$$

$$= [6 * .866 + 12 * .5 \qquad 6 * (-.5) + 12 * (.866)]$$

$$= [5.196 + 6 \qquad -3 + 10.392]$$

$$= [11.196 \qquad 7.392]$$

B point (6, 12) becomes (11.46, 7.312) after rotation 30° in clockwise direction.



Basic Transformations

- All of the three basic transformations are ***rigid body transformations*** – that moves objects without deformation.
- That is, every point on the object is *translated, scaled or rotated* by the ***same amount***.
  - A ***straight Line segment*** is *translated, scaled or rotated* by applying the transformation to each of the line endpoints and redrawing the line between the **new endpoint positions.**
  - ***Polygons*** are ***translated, scaled or rotated*** by adding the transformation vectors to the **coordinate position** of each vertex and regenerating the polygon using the new set of vertex coordinates and the current attribute settings.

Homogeneous Coordinates

- Many graphics applications involve sequences of geometric **transformations.**
  - E.g., An animation might require an object to be translated and rotated at each *increment* of the motion.
- In **design** and picture construction applications, we perform translations, rotations, and scaling to **fit the picture components** into their **proper positions.**
- We consider how the ***matrix representations*** can be reformulated so that such **transformation sequences** can be ***efficiently*** processed.

Homogeneous Coordinates

- A combination of *translations*, *rotations* and *scaling* can be expressed as:

$$
\begin{aligned}
P' &= S \cdot R \cdot (P + T) \\
&= (S \cdot R) \cdot P + (S \cdot R \cdot T) \\
&= M \cdot P + A
\end{aligned}
$$

- Because **scaling and rotation** are expressed as **matrix multiplication** but translation is expressed as **matrix addition**,
  - it is not, in general, possible to combine a set of operations into a single matrix operation.
- ***Composition of transformations*** is often desirable if the same set of operations have to be applied to a ***list of position vectors.***

Homogeneous Coordinates

- We can solve this problem by representing points in ***homogenous coordinates***.
- In homogenous coordinates we **add a third coordinate** to a point, i.e., instead of representing a point by a pair (x, y), we represent it as (x, y, W).
- Two sets of homogenous coordinates represent the **same point** if one is a **multiple** of the other.
  - E.g. $(2, 3, 6)$ and $(4, 6, 12)$ represent the **same point.**
- Given homogenous coordinates (x, y, W), the Cartesian coordinates (x', y') correspond to $(\frac{x}{W}, \frac{y}{W})$, i.e. the homogenous coordinates at which W = 1.
  - Points with $W = 0$ are called points at **infinity.**

Homogeneous Coordinates

- Representing points as 3-dimensional vectors, we can re-write our transformation matrices as follows:
- **Translation**
  - A translation by (t_x, t_y) is represented by:

$$\boldsymbol{P}' = \boldsymbol{T}(\boldsymbol{t_x}, \boldsymbol{t_y}) \cdot \boldsymbol{P}$$

where

$$\boldsymbol{T}(\boldsymbol{t_x}, \boldsymbol{t_y}) = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

- **Scaling**
  - Scaling by the factors $\boldsymbol{s_x}$ and $\boldsymbol{s_y}$ relative to the origin is given by:

$$\boldsymbol{P}' = \boldsymbol{S}(\boldsymbol{s_x}, \boldsymbol{s_y}) \cdot \boldsymbol{P}$$

where

$$\boldsymbol{S}(\boldsymbol{sx}, \boldsymbol{s_y}) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Homogeneous Coordinates

- Representing points as 3-dimensional vectors, we can re-write our transformation matrices as follows:
- **Rotation**
  - Rotation about the origin by θ is given by:

$$\boldsymbol{P'} = \boldsymbol{R(\theta)} \cdot \boldsymbol{P}$$

where

$$\boldsymbol{R(\theta)} = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Example

$P = (1,1)$ $\quad S_x = 2$, $S_y = 2$ $\quad$ Rotate by 90° $\quad T_x = 1$, $T_y = 0$ $\quad P' = ?$

$P_{sx} = S_x x = 2$ $\quad (2,2)$

$P_{sy} = S_y y = 2$

$P_{Rx} = x\cos\theta - y\sin\theta = -2$

$P_{Ry} = x\sin\theta + y\cos\theta = 2$

$(-2, 2)$

$P_{Tx} = x + T_x = -2 + 1 = -1$

$P_{Ty} = y + T_y = 2 + \emptyset = 2$ $\quad (-1, 2)$

Composition of 2D Transformations

- Using the associative property of matrix multiplication, we can combine several transformations into a single matrix.
- It can easily be shown that:

$$T\left(t_{x_2}, t_{y_2}\right) \cdot T\left(t_{x_1}, t_{y_1}\right) = T\left(t_{x_1} + t_{x_2}, t_{y_1} + t_{y_2}\right)$$

$$S(s_{x_2}, s_{y_2}) \cdot S(s_{x_1}, s_{y_1}) = S(s_{x_1} s_{x_2}, s_{y_1} s_{y_2})$$

$$R(\theta_2) \cdot R(\theta_1) = R(\theta_1 + \theta_2)$$

General fixed-point Scaling

- Scaling with factors s_x and s_y with respect to the fixed point (x, y) can be achieved by the following list of operations:
  1. **Translate by $(-\boldsymbol{x}, -\boldsymbol{y})$** so that the fixed point is moved to the origin;
  2. Scale by s_x and s_y with respect to the origin;
  3. Translate by (x, y) to return the fixed point back to its original position.
- So, the general **fixed-point scaling** transformation matrix is:

$$S_{x,y}\,(s_x, s_y) = T(x, y) \cdot S(s_x, s_y) \cdot T(-x, -y) = \begin{bmatrix} s_x & 0 & x(1 - s_x) \\ 0 & s_y & y(1 - s_y) \\ 0 & 0 & 1 \end{bmatrix}$$

Composition of 2D Transformations

General pivot-point rotation

- A rotation about the rotation-point or pivot-point (x, y) by θ is achieved as follows:
  1. Translate by $(-x, -y)$;
  2. Rotate by θ;
  3. Translate by (x, y).
- The general pivot-point rotation matrix is therefore given by:

$$R_{x,y}(\theta) = T(x, y) \cdot R(\theta) \cdot T(-x, -y)$$

$$= \begin{bmatrix} cos\theta & -sin\theta & x(1 - cos\theta) + ysin\theta \\ sin\theta & cos\theta & y(1 - cos\theta) - xsin\theta \\ 0 & 0 & 1 \end{bmatrix}$$

Example

Example 1

- Consider a square located at (1,1) (1,2) (2,1)(2,2)

i) Perform translations with (1,1) and (4,4)

ii) Perform Rotations with the angle 90^0 *and* 90^0

iii) Perform the Scaling with (4,4) and $({}^1/_2 , {}^1/_2)$

P1(1,1) P2(1,2) P3(2,1) P4(2,2)

$t_{x1} = 1, t_{y1} = 1\ t_{x2} = 4, t_{y2} = 4$

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} x + t_{x1} + t_{x2} \\ y + t_{y1} + t_{y2} \\ 1 \end{pmatrix}$$

$$P1' = \begin{pmatrix} x1' \\ y1' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 + 1 + 4 \\ 1 + 1 + 4 \\ 1 \end{pmatrix} = \begin{pmatrix} 6 \\ 6 \\ 1 \end{pmatrix}$$

$$P2' = \begin{pmatrix} 6 \\ 7 \\ 1 \end{pmatrix}\ P3' = \begin{pmatrix} 7 \\ 6 \\ 1 \end{pmatrix}\ P4' = \begin{pmatrix} 7 \\ 7 \\ 1 \end{pmatrix}$$

- $\theta_1 = 90^0 \quad \theta_2 = 90^0$

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} x\cos(\theta_1 + \theta_2) - y\sin(\theta_1 + \theta_2) \\ x\sin(\theta_1 + \theta_2) + y\cos(\theta_1 + \theta_2) \\ 1 \end{pmatrix}$$

$$P1' = \begin{pmatrix} x1' \\ y1' \\ 1 \end{pmatrix} = \begin{pmatrix} 1.\cos(90 + 90) - 1.\sin(90 + 90) \\ 1.\sin(90 + 90) + 1.\cos(90 + 90) \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ -1 \\ 1 \end{pmatrix}$$

$$s_{x1} = 4, s_{y1} = 4\ s_{x2} = {}^{1}/_{2}, s_{y2} = {}^{1}/_{2}$$

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} x.\,s_{x1}.\,s_{x2} \\ y.\,s_{y1}.\,s_{y2} \\ 1 \end{pmatrix}$$

$$P1' = \begin{pmatrix} x1' \\ y1' \\ 1 \end{pmatrix} = \begin{pmatrix} 1.4.\,{}^{1}/_{2} \\ 1.4.\,{}^{1}/_{2} \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix}$$

Composition of 2D Transformations

Inverse transformations

- The matrix of an inverse transformation is the **inverse matrix** of the transformation:
  - If $P' = M \cdot P$ then $P = M^{-1} \cdot P'$
  - It can easily be shown that:

$$T(t_x, t_y)^{-1} = T(-t_x, -t_y)$$

$$S(s_x, s_y)^{-1} = S\left(\frac{1}{s_x}, \frac{1}{s_y}\right)$$

$$R(\theta)^{-1} = R(-\theta)$$

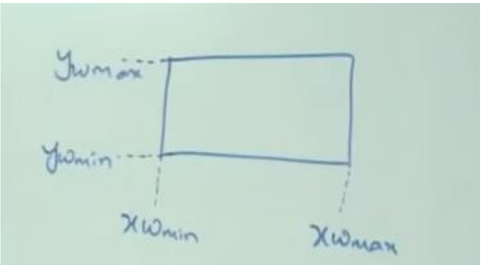
Clipping Algorithms

- Usually only a specified region of a picture needs to be displayed.
  - This region is called the **clip window.**
- Window –what to be displayed
- Clipping –Discarding the portion outside the window.
- They are 3types of clipping
  - Point clipping
  - Line clipping
  - Polygon clipping
- An algorithm which displays only those primitives (or part of the primitives) which lie inside a clip window is referred to as a ***clipping algorithm.***

Clipping Points

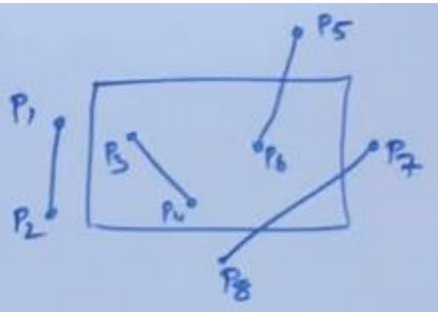
- A point (x, y) lies inside the rectangular clip window $(x_{min}, y_{min}, x_{max}, y_{max})$ if and only if the following inequalities hold:

$$x_{min} \leq x \leq x_{max}$$
$$y_{min} \leq y \leq y_{max}$$



Line Clipping

Line Clipping



$P_1P_2 \rightarrow$ Reject

$P_3P_4 \rightarrow$ consider (Accept)

$P_5P_6 \rightarrow$ Clipping Required

$P_7P_8 \rightarrow$ clipping Required.

The Cohen-Sutherland algorithm for line clipping

- Clipping a straight line against a rectangular clip window results in either:
    1. A line segment whose endpoints may be different from the original ones, or
    2. Not displaying any part of the line. This occurs if the line lies completely outside the clip window.
- The Cohen-Sutherland line clipping algorithm considers each endpoint at a time and truncates the line with the clip window's edges until the line can be trivially accepted or trivially rejected.
- A line is trivially rejected if both endpoints lie on the same outside half-plane of some clipping edge

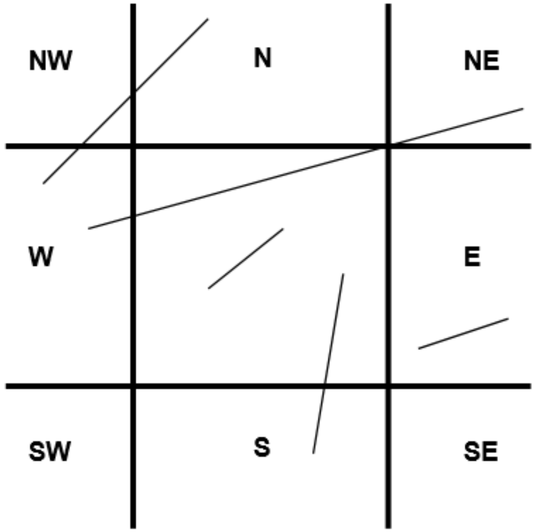
Clipping Algorithms

The Cohen-Sutherland algorithm for line clipping

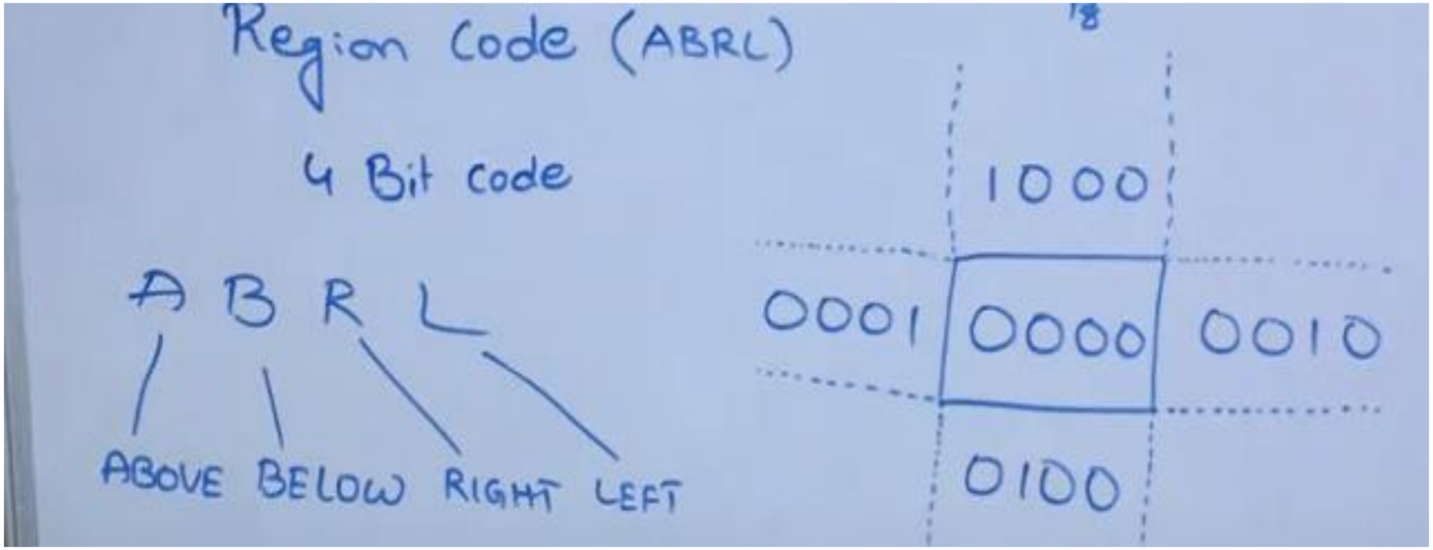
- The xy plane is partitioned into nine segments by extending the clip window's edges.
- Each segment is assigned a **4-bit code** according to where it lies with respect to the clip window:

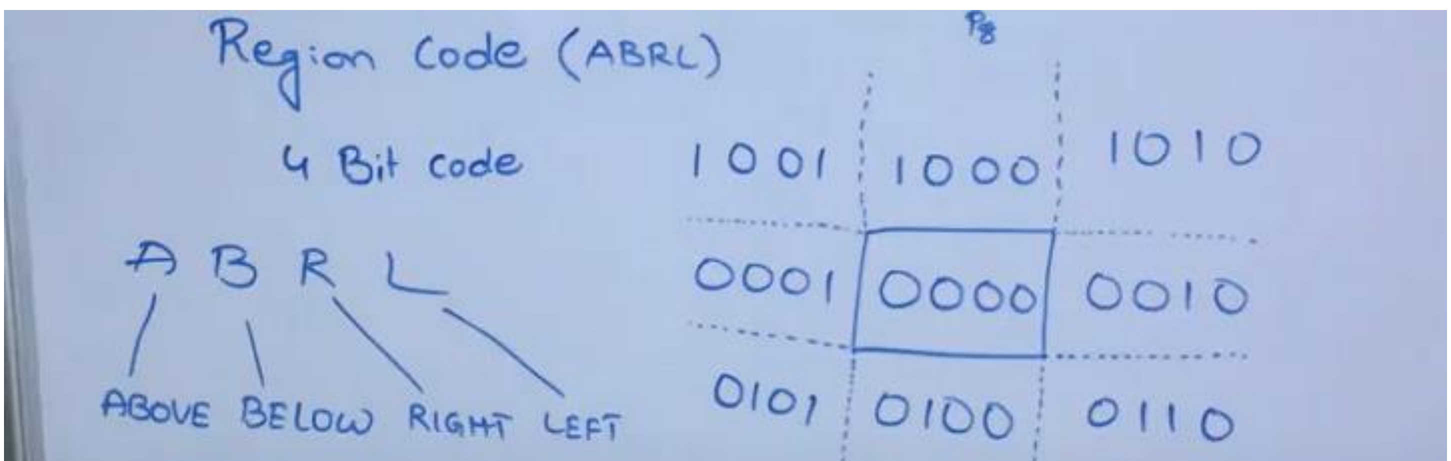
Bit	*Side*	*Inequality*
1	N	$y > y_{max}$
2	S	$y < y_{min}$
3	E	$x > x_{max}$
4	W	$x < x_{min}$

Partition of plane into 9 segments.

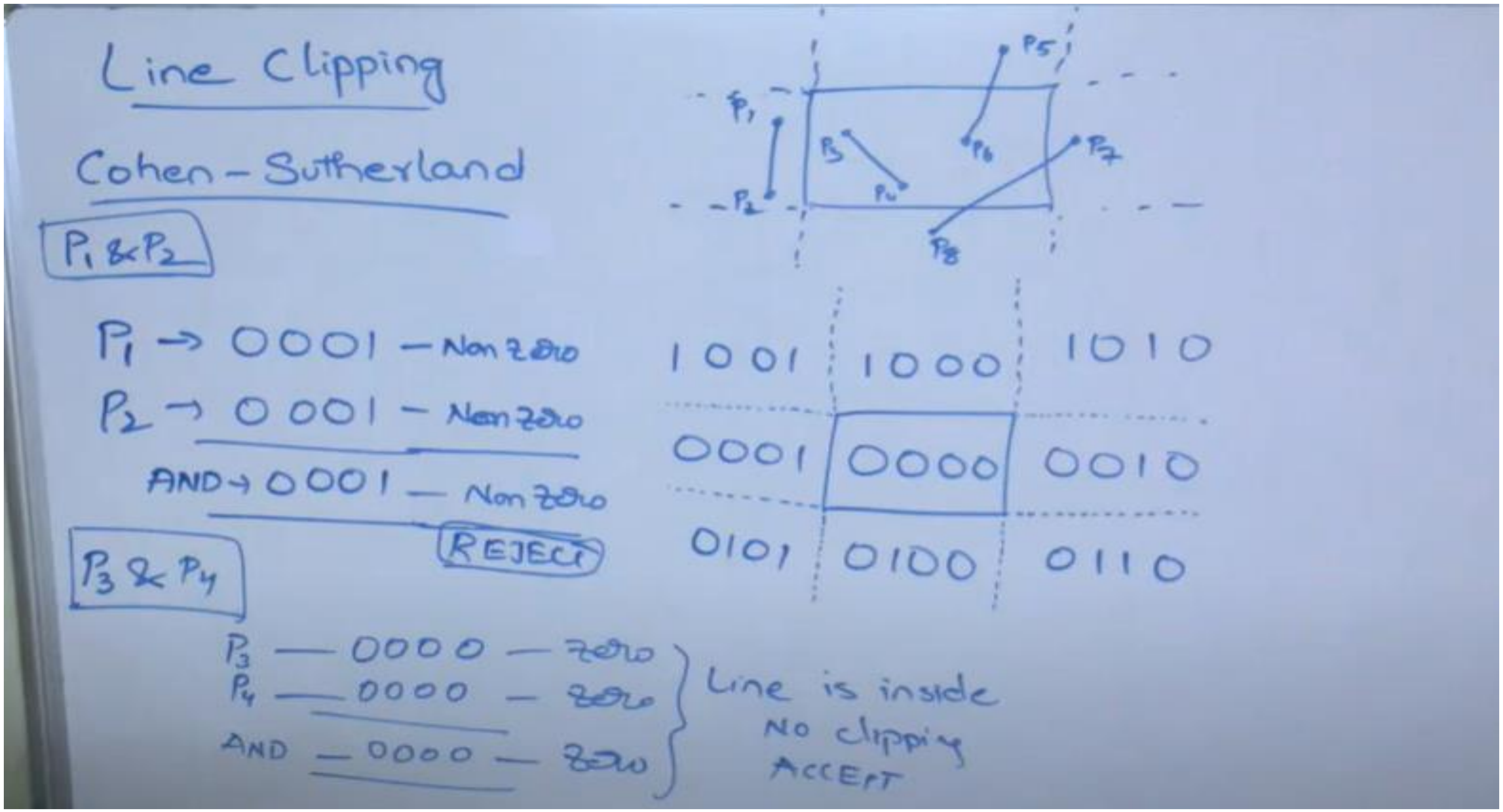


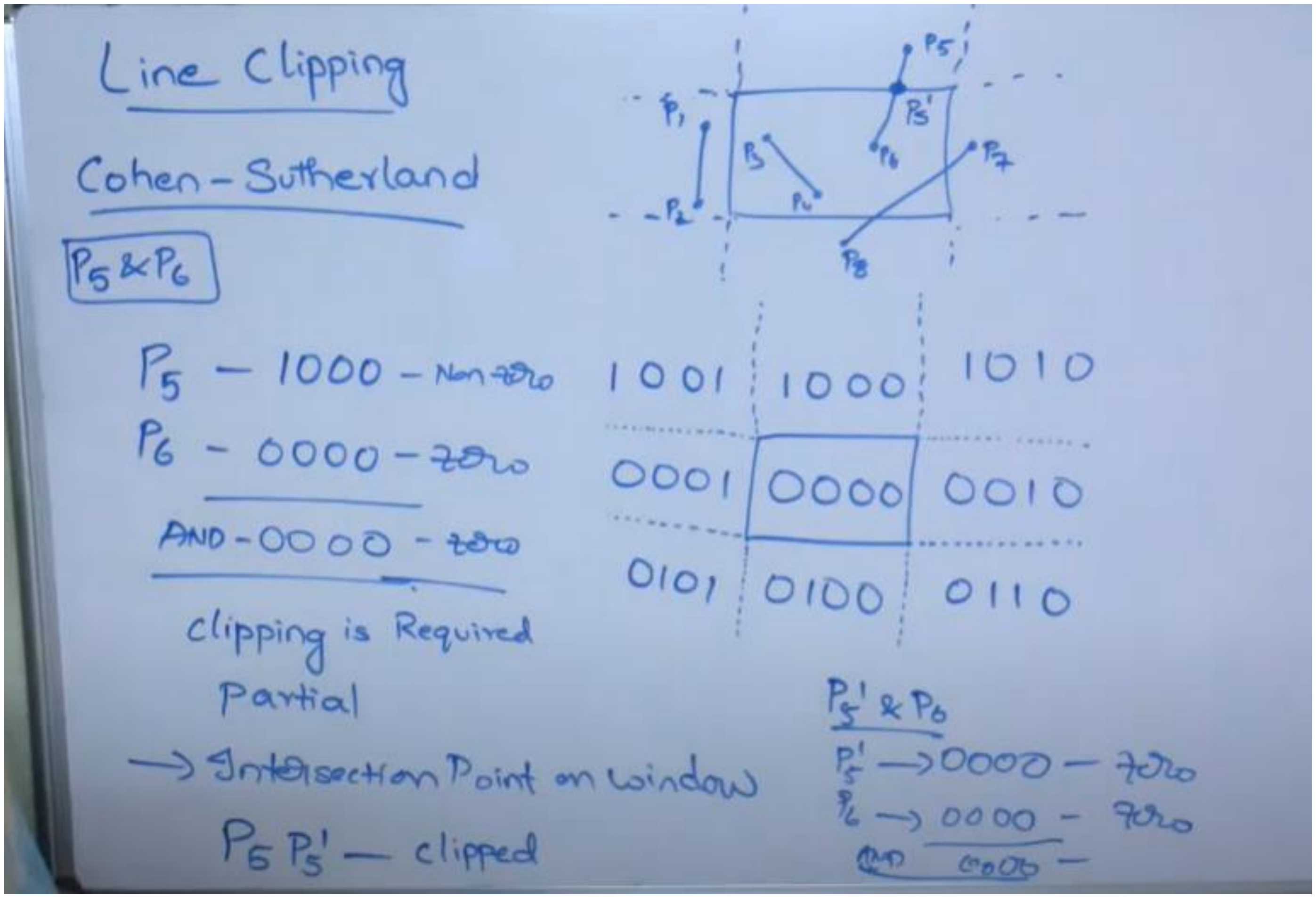
- For example bit 2 is set if and only if the region lies to the south of (below) the clip window.

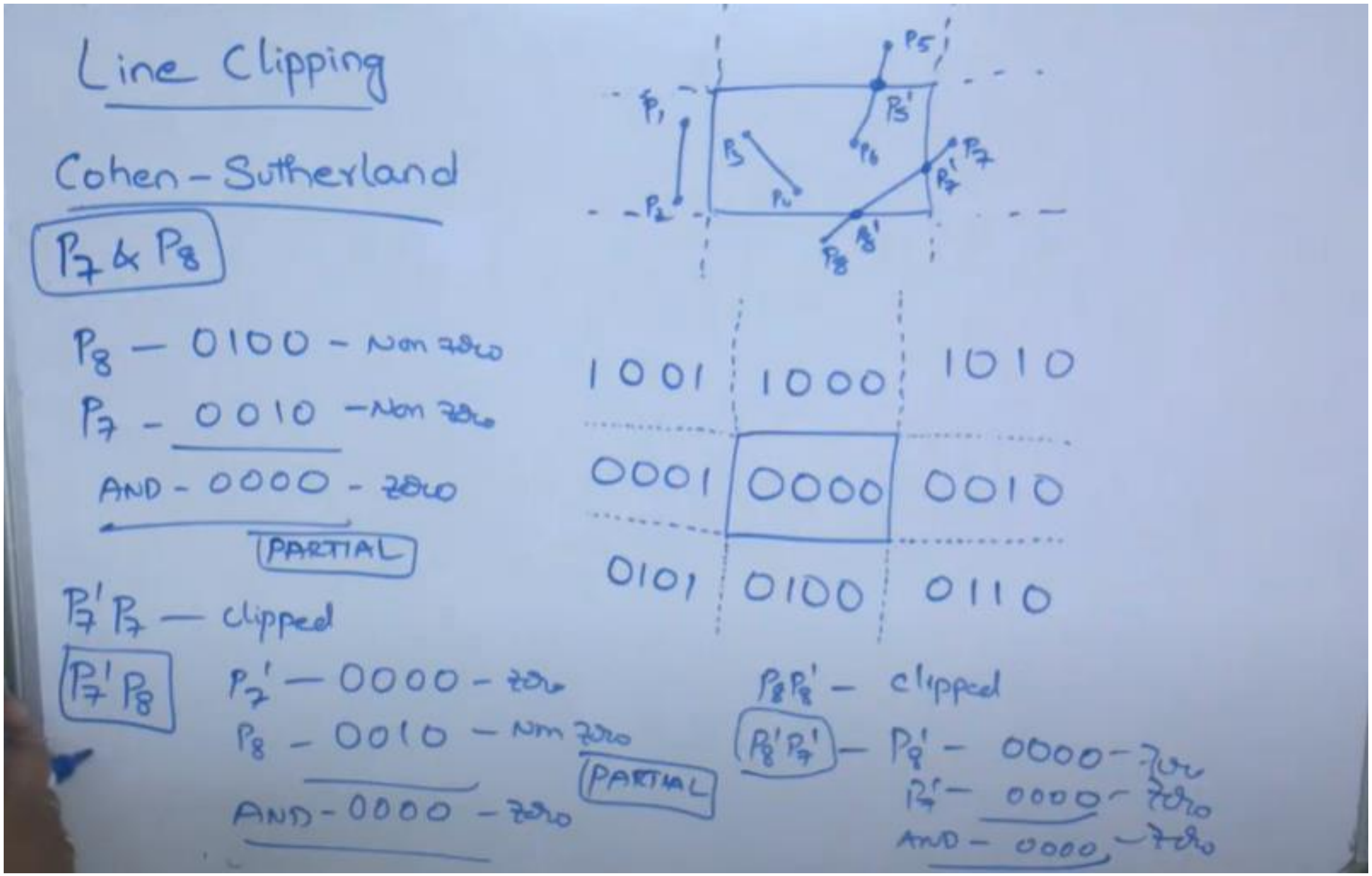

Region Code (ABRL)
4 Bit code
A B R L
ABOVE BELOW RIGHT LEFT
1000
0001
0000
0010
0100


Region Code (ABRL)
4 Bit code
A B R L
ABOVE BELOW RIGHT LEFT
1001
1000
1010
0001
0000
0010
0101
0100
0110

Example


Line Clipping
Cohen-Sutherland
P1 & P2
P1 → 0001 – Non zero
P2 → 0001 – Non zero
AND → 0001 – Non zero
REJECT
P3 & P4
P3 — 0000 — zero
P4 — 0000 — zero
AND — 0000 — zero
Line is inside
No clipping
ACCEPT
P1
P2
P3
P4
P5
P6
P7
P8
1001
1000
1010
0001
0000
0010
0101
0100
0110


Line Clipping
Cohen-Sutherland
P5 & P6
P5 — 1000 — Non zero
P6 — 0000 — zero
AND — 0000 — zero
clipping is Required
Partial
→ Intersection Point on window
P5 P5' — clipped
P1
P2
P3
P4
P5
P5'
P6
P7
P8
1001 1000 1010
0001 0000 0010
0101 0100 0110
P5' & P6
P5' → 0000 — zero
P6 → 0000 — zero
AND 0000 —


Line Clipping
Cohen-Sutherland
P7 & P8
P8 — 0100 - Non zero
P7 - 0010 - Non zero
AND - 0000 - zero
PARTIAL
P7' P7 — clipped
P7' P8
P7' — 0000 - zero
P8 - 0010 - Non zero
AND - 0000 - zero
PARTIAL
P5
P5'
P1
P2
P3
P4
P6
P7
P7'
P8'
P8
1001
1000
1010
0001
0000
0010
0101
0100
0110
P8 P8' — clipped
P8' P7'
P8' - 0000 - zero
P7' - 0000 - zero
AND - 0000 - zero

The Cohen-Sutherland algorithm for line clipping

- The **Cohen-Sutherland algorithm** starts by assigning an ***outcode*** to the two line endpoints (say c_1 and c_2).
  - If both *outcodes* are 0 ($\boldsymbol{c_1\ OR\ c_2 = 0}$) the line lies entirely in the **clip window** and is thus **trivially** accepted.
  - If the two *outcodes* have at least one bit in common ($\boldsymbol{c_1\ AND\ c_2 \neq 0}$), then they lie on the same side of the window and the line is **trivially rejected.**
  - If a line cannot be **accepted** or **rejected**, it is then truncated by an **intersecting clip edge** and the **previous steps** are repeated.

Clipping Algorithms

The Cohen-Sutherland algorithm for line clipping

Algorithm:

Step 1: Assign a **region code** for each end points.

Step2: If both endpoints have a region code 0000 then **accept** the line. Else apply Step 3.

Step3: Perform the **logical AND operation** for both region codes.

- If the result is not 0000, then reject the line.
- Else clip the line by:
  - Choose an endpoint of the line that is outside the window.
  - Find the intersection point at the window boundary (based on region code).
  - Replace endpoint with the intersection point and update the region code.

Step 4: Repeat Step 2 until we find a clipped line either trivially accepted or rejected.

Step 5: Repeat Step 1 for other lines.

Clipping Algorithms

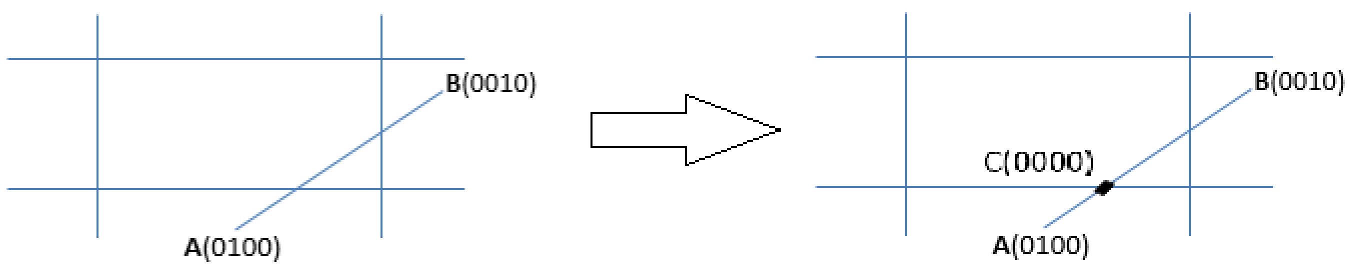
The Cohen-Sutherland algorithm for line clipping

<u>Example</u>:

A(0100) B(0010)

AND Operation

Then get the new point C(0000)



Clipping Algorithms

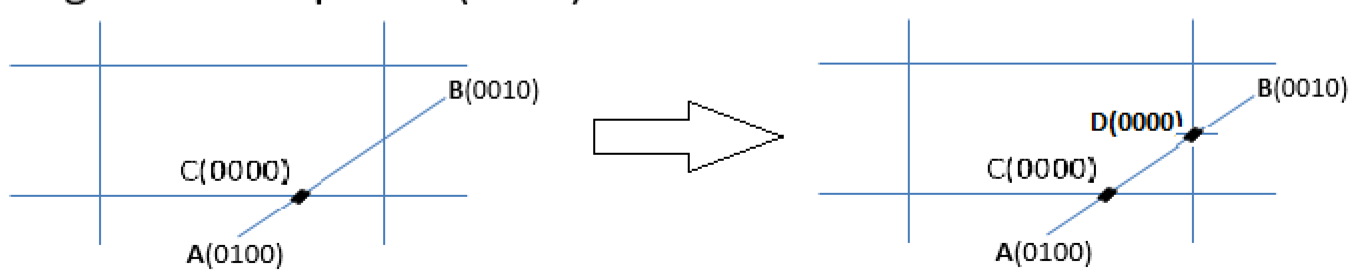
The Cohen-Sutherland algorithm for line clipping

Example:

C(0000) B(0010)

AND Operation

Then get the new point D(0000)



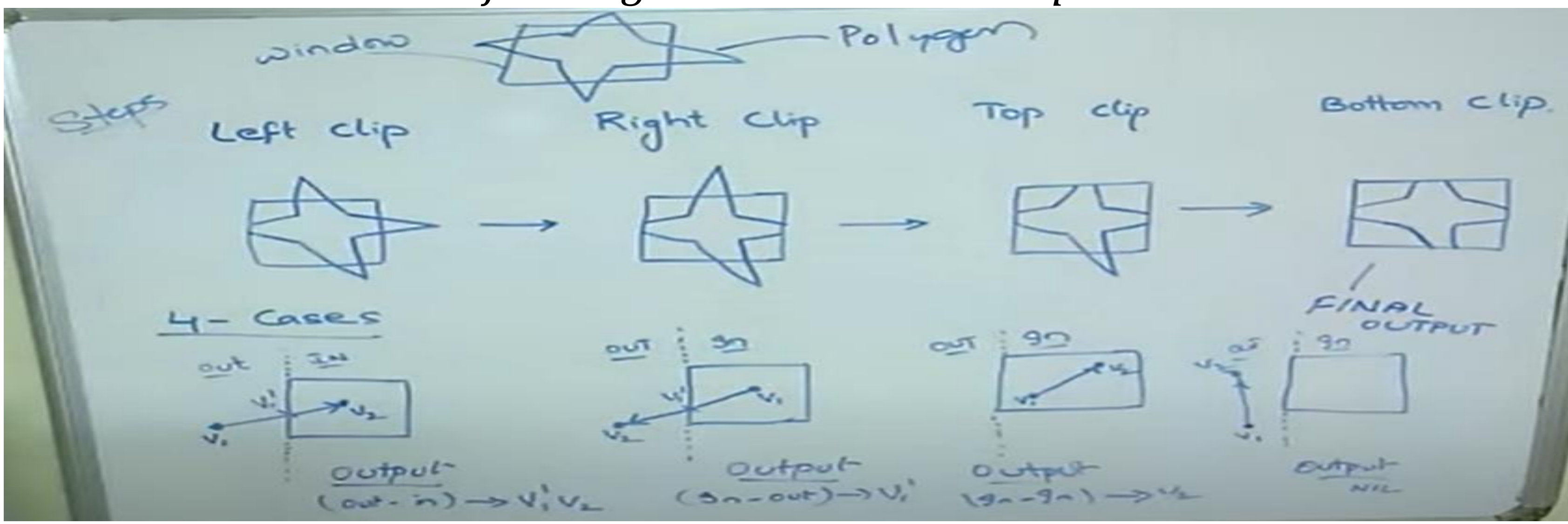
Then we have the final line after clipping CD.

Clipping Algorithms Polygon Clipping

Sutherland-Hodgman polygon clipping algorithm

- A polygon can be clipped against a rectangular clip window by considering each clip edge at a time.

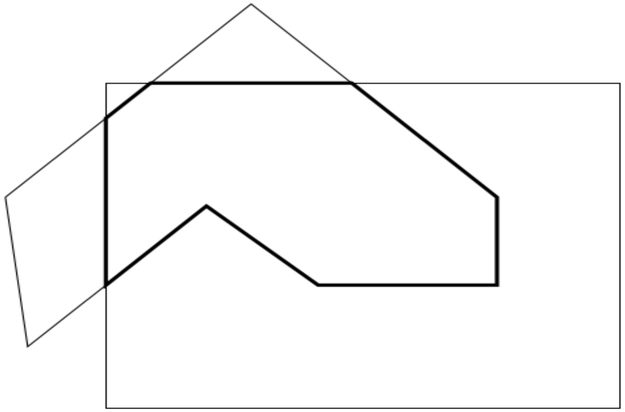
$$Left \rightarrow Right \rightarrow Bottom \rightarrow Top$$



Clipping Algorithms Polygon Clipping

Sutherland-Hodgman polygon clipping algorithm

- For filled polygons, a new polygon boundary must be computed.



- At each clipping stage, a new polygon is created by removing the outside vertices and inserting the intersections with the clip boundary.

Sutherland-Hodgman polygon clipping algorithm

Clipping against an edge

- Given the polygon $[v_1; v_2; \ldots; v_n]$, we can create a new polygon $[w_1; w_2; \ldots; w_m]$ by
  - considering the vertices in sequence and
  - deciding which ones (and any intersections with the edge) have to be inserted into the clipped polygon vertex list.
- Suppose that v_i has been processed in the previous step.
- There are four possible cases which need to be considered while processing the next vertex v_{i+1}.

Clipping Algorithms Polygon Clipping

Sutherland-Hodgman polygon clipping algorithm

Clipping against an edge

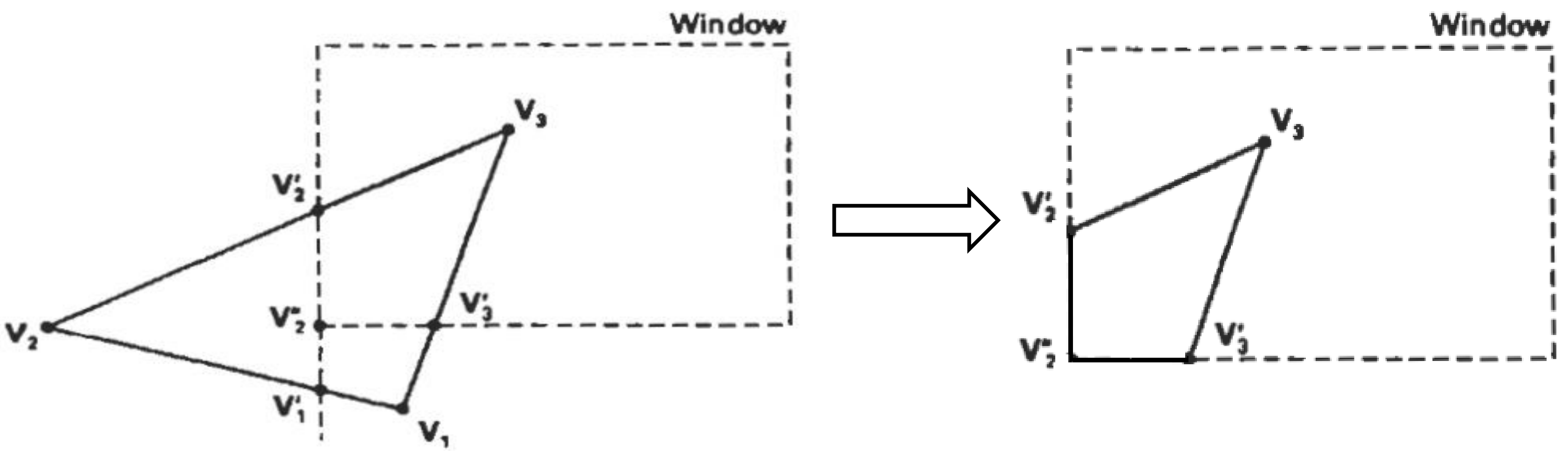
1. If v_i is outside the window and v_{i+1} is inside, then the intersection of the line $v_i \rightarrow v_{i+1}$ with the clipping edge, and the vertex v_{i+1} have to be inserted into the **new polygon list**.
2. If v_i and v_{i+1} are both outside the clip window then no point need be inserted.
3. If both v_i and v_{i+1} are inside the window, v_{i+1} is inserted into the list, and finally.
4. If v_i is inside the window and v_{i+1} is outside, only the intersection of the line $v_i \rightarrow v_{i+1}$ with the clip boundary is inserted.

Clipping Algorithms Polygon Clipping

Sutherland-Hodgman polygon clipping algorithm

Clipping against an edge

- **Example**: Processing the vertices of the polygon below through a boundary-clipping pipeline.



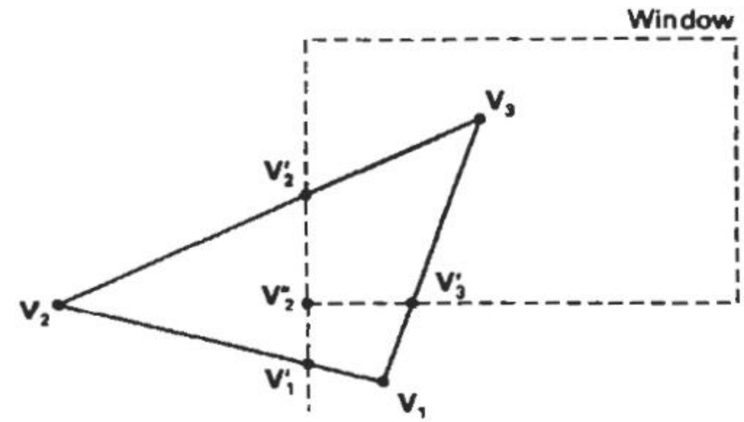
- After all vertices are processed through the pipeline, the vertex list for the clipped polygon is $[v_2'', v_2', v_3, v_3']$.

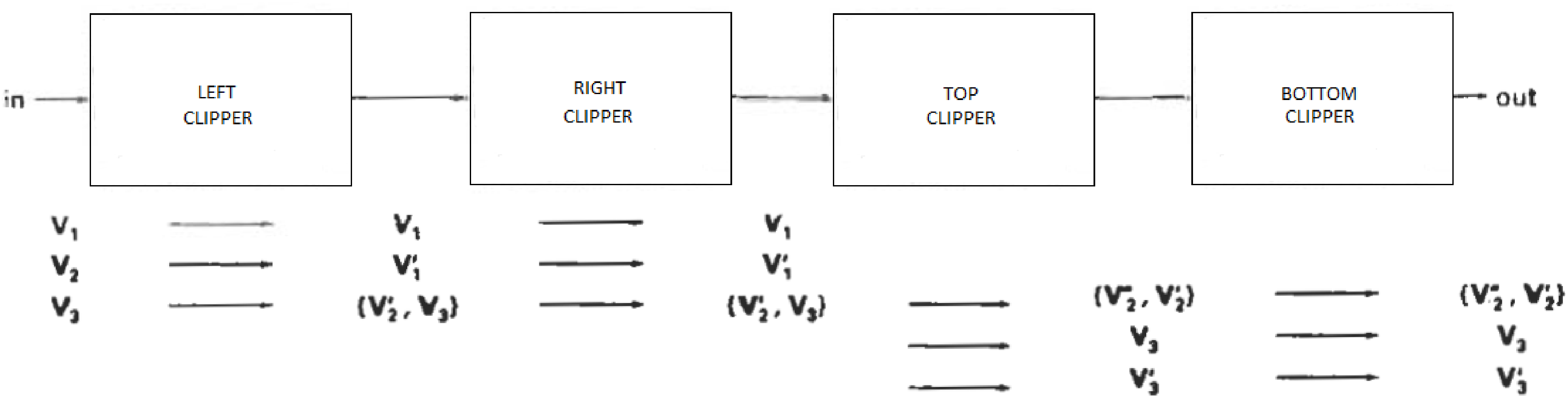
Clipping Algorithms Polygon Clipping

Sutherland-Hodgman polygon clipping algorithm

Clipping against an edge

- **Example:** cont….





Thank You!!!

End of CH3!

COMPUTER GRAPHICS

CH4 – Windows and Viewports

Outline

- Windows and Viewports
  - Co-ordinates Representation
  - Viewing Pipeline
  - Window to Viewport Transformation
  - Zooming and Panning effects with windows and viewports

Window port

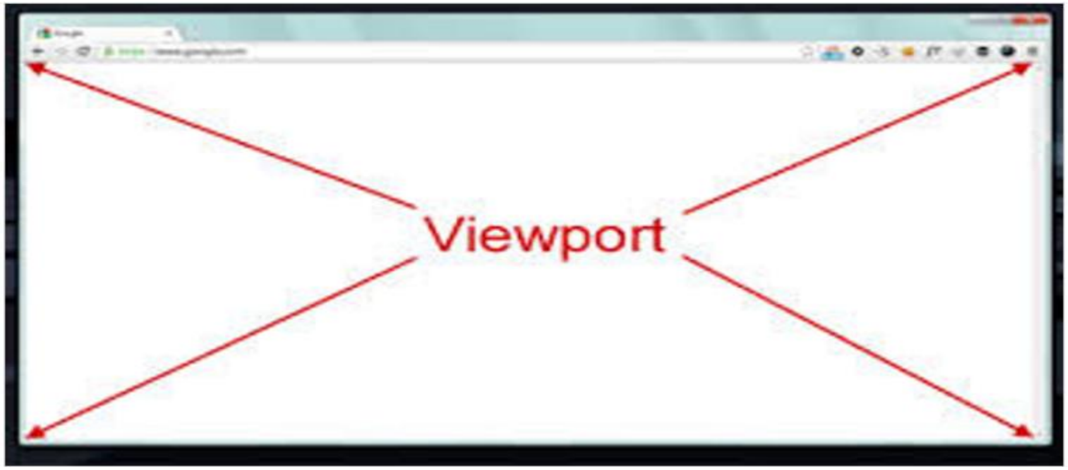
- The **window port** can be confused with the **computer window** but it isn't the **same**.
- The **window port** is the **area chosen** from the **real world for display**.
- This **window port** decides **what portion** of the real world should be captured and be displayed on the screen.
- The widow port can thus be defined as, "A ***world-coordinate area selected for display is called a window. A window defines a rectangular area in the world coordinates.***"

Co-ordinates Representation

- ***Viewing*** involves ***transforming*** an object specified in a **world coordinates** frame of reference into a **device coordinates frame** of reference.
  - ***A region*** in the world coordinate system which is **selected for display** is called a **window**.
  - The region of a **display device** into which a window is mapped is called a ***viewport***.
- The steps required for transforming world coordinates (or modelling coordinates) into device coordinates are referred to as the ***viewing pipeline***.

Viewport

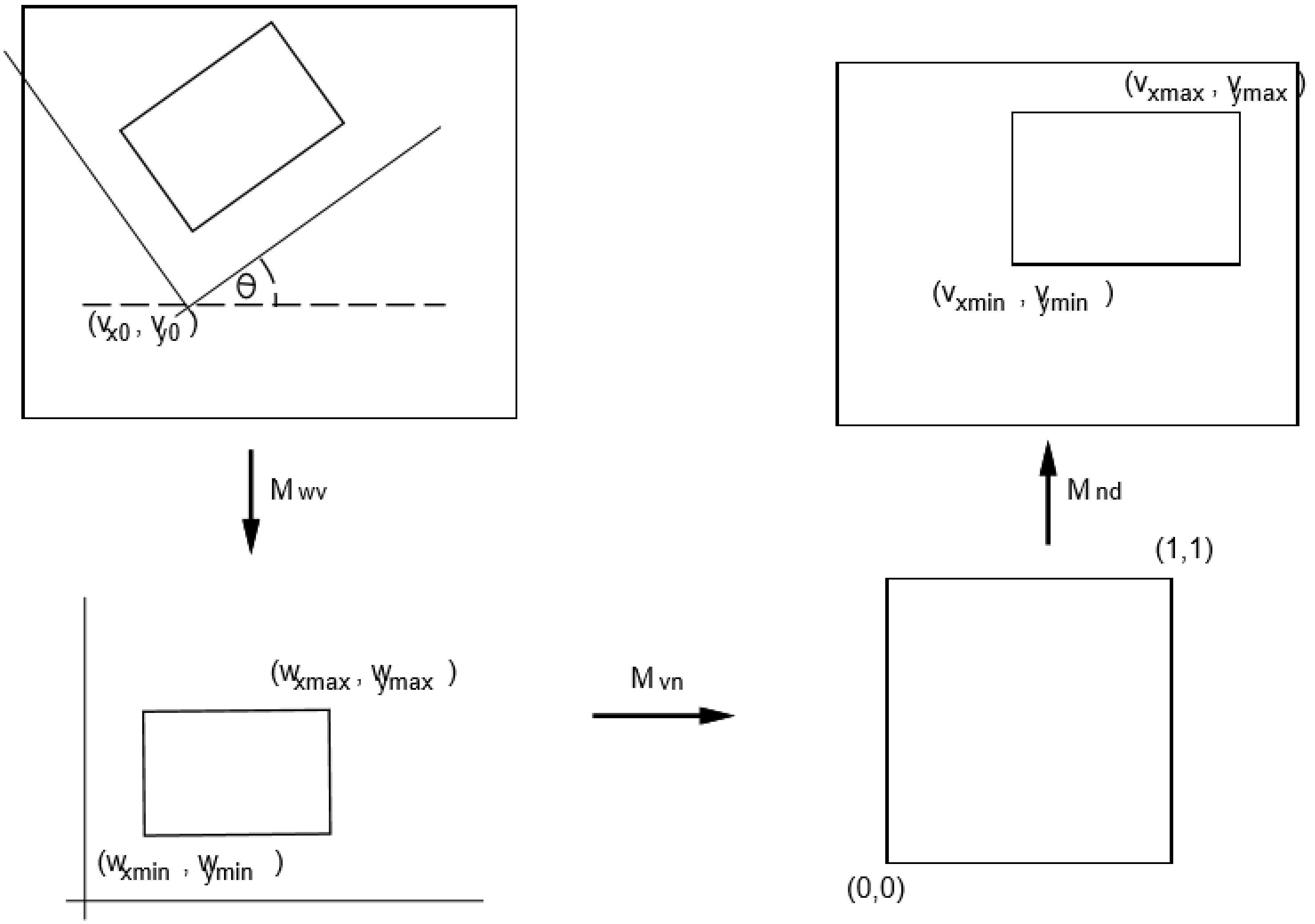
- **Viewport** is the **area on a display device** to which a window is mapped.
- viewport is our device's screen.
- ***A viewport is a polygon viewing region in computer graphics. The viewport is an area expressed in rendering-device-specific coordinates, e.g. pixels for screen coordinates, in which the objects of interest are going to be rendered."***



Viewing Pipeline

- Steps involved in the 2D viewing pipeline:
  1. Transforming ***world coordinates*** into ***viewing coordinates***, usually called the ***viewing transformation***.
     - Given 2D objects represented in world coordinates, a window is specified in terms of a viewing coordinate system defined relative to the world coordinate system.
  2. ***Normalizing*** the viewing coordinates.
     - The object's world coordinates are transformed into viewing coordinates and usually normalized so that the extents of the window fit in the rectangle with **lower left corner at (0,0) and the upper right corner at (1,1).**
  3. Transforming the normalized viewing coordinates into **device coordinates.**
     - The normalized viewing coordinates are then **transformed** into device coordinates to **fit into** the viewport.

Viewing Pipeline



Window to Viewport Transformation

Window to Viewport Transformation is the process of transforming 2D world-coordinate objects to device coordinates. Objects inside the world or clipping window are mapped to the viewport which is the area on the screen where world coordinates are mapped to be displayed.

A world-coordinate area selected for display is called a window. An area on a display device to which a window is mapped is called a viewport.

The window defines what is to be viewed;

The viewport defines where it is to be displayed.

[redacted]

Window to Viewport Transformation

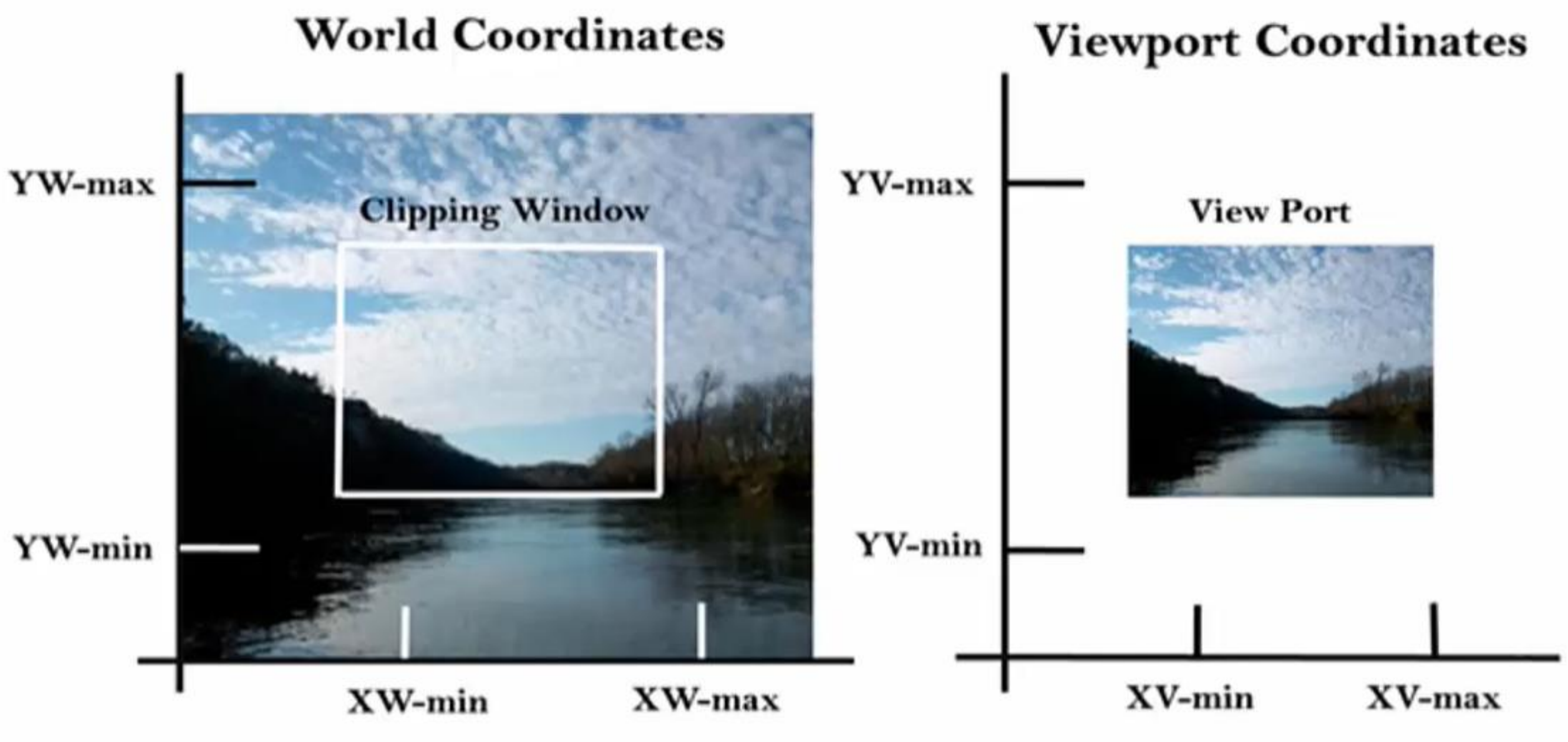


Fig: Window to viewport mapping

World coordinate – It is the Cartesian coordinate w.r.t which we define the diagram, like X_{wmin}, X_{wmax}, Y_{wmin}, Y_{wmax}

Device Coordinate –It is the screen coordinate where the objects are to be displayed, like X_{vmin}, X_{vmax}, Y_{vmin}, Y_{vmax}

Window –It is the area on world coordinate selected for display.

ViewPort –It is the area on the device coordinate where graphics is to be displayed.

It may be possible that the size of the Viewport is much smaller or greater than the Window. In these cases, we have to increase or decrease the size of the Window according to the Viewport and for this, we need some mathematical calculations.

(x_w, y_w): A point on Window
(x_v, y_v): Corresponding point on Viewport

we have to calculate the point (x_v, y_v)

WINDOW COORDINATE →NORMALISED COORDINATE→DEVICE COORDINATES
(within (0,1))

Window coordinate to normalized coordinate translation .
Scaling in the normalised coordinate system in order to make it match with viewport(device coordinate system) .
Translate to Device coordinate (view port) system.

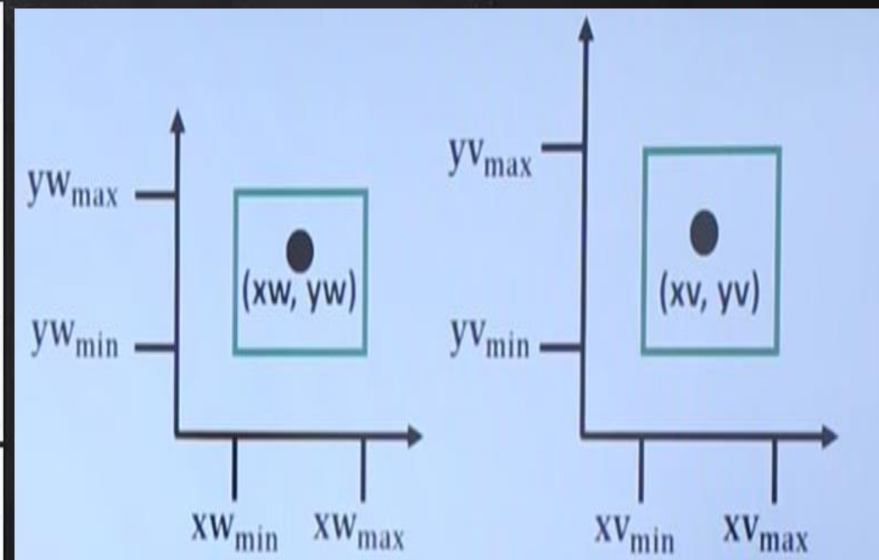
In order to maintain the same relative placement of the point in the viewport as in the window, we require

Normalized Point on Window

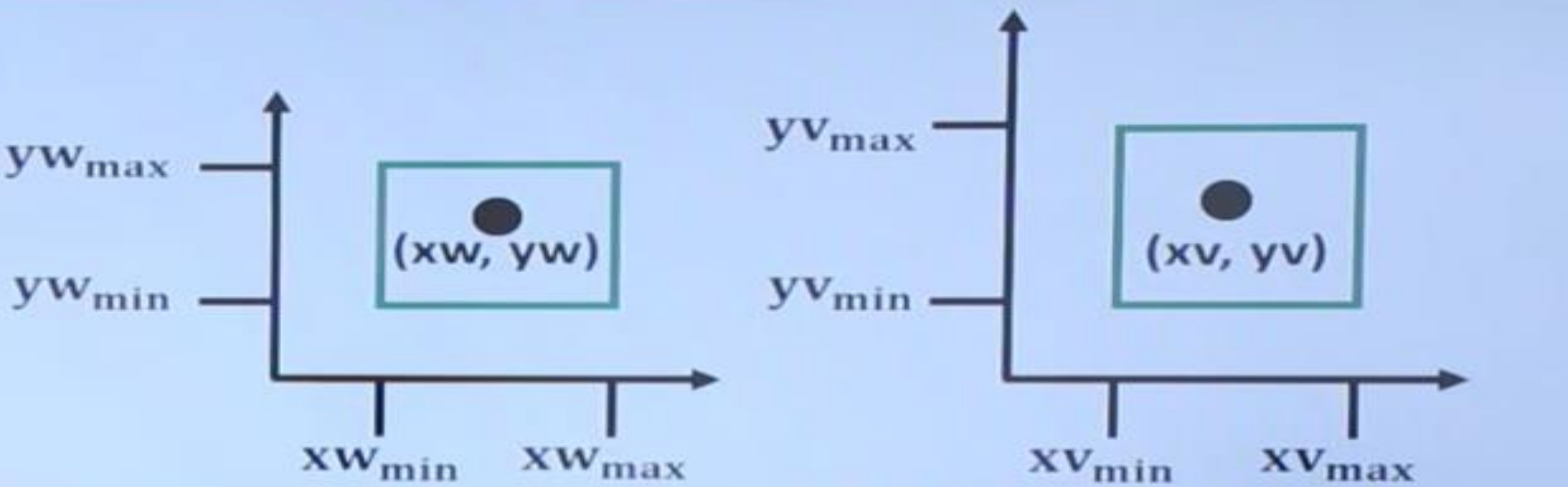
$$\left(\frac{X_W - X_{wmin}}{X_{wmax} - X_{wmin}}, \frac{Y_W - Y_{wmin}}{Y_{wmax} - Y_{wmin}} \right)$$

Normalized Point on Viewport

$$\left(\frac{X_V - X_{vmin}}{X_{vmax} - X_{vmin}}, \frac{Y_V - Y_{vmin}}{Y_{vmax} - Y_{vmin}} \right)$$



Now the relative position of the object in Window and [redacted] Viewport are same



(a) Window in object space (b) Viewport in device space

To maintain the same relative placement of the point in the viewport as in the window,

$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}}$$

And also,

$$\frac{yv - yv_{min}}{yv_{max} - yv_{min}} = \frac{yw - yw_{min}}{yw_{max} - yw_{min}}$$

$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}}$$

And also, $$\frac{yv - yv_{min}}{yv_{max} - yv_{min}} = \frac{yw - yw_{min}}{yw_{max} - yw_{min}}$$

$$xv = xv_{min} + \frac{xw - xw_{min}}{xw_{max} - xw_{min}} (xv_{max} - xv_{min})$$

$$yv = yv_{min} + \frac{yw - yw_{min}}{yw_{max} - yw_{min}} (yv_{max} - yv_{min})$$

If we consider $s_x = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}$ and $s_y = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$

Thus we have $xv = xv_{min} + (xw - xw_{min})\, s_x$

And $yv = yv_{min} + (yw - yw_{min})\, sy$

$$X_v = X_{vmin} + (X_w - X_{wmin})\, S_x$$

$$Y_v = Y_{vmin} + (Y_w - Y_{wmin})\, S_y$$

where s_x is the scaling factor of x coordinate and s_y is the scaling factor of y coordinate

$$S_x = \frac{X_{vmax} - X_{vmin}}{X_{wmax} - X_{wmin}}$$

$$S_y = \frac{Y_{vmax} - Y_{vmin}}{Y_{wmax} - Y_{wmin}}$$

Example

for window, $X_{wmin} = 20$, $X_{wmax} = 80$, $Y_{wmin} = 40$, $Y_{wmax} = 80$.

for viewport, $X_{vmin} = 30$, $X_{vmax} = 60$, $Y_{vmin} = 40$, $Y_{vmax} = 60$.

Now a point (X_w, Y_w) be (30, 80) on the window. We have to calculate that point on the viewport i.e (X_v, Y_v).

$$S_x = \frac{X_{vmax} - X_{vmin}}{X_{wmax} - X_{wmin}}$$

S_x = (60 - 30) / (80 - 20) = 30 / 60

S_y = (60 - 40) / (80 - 40) = 20 / 40

$$S_y = \frac{Y_{vmax} - Y_{vmin}}{Y_{wmax} - Y_{wmin}}$$

VINDOW TO VIEWPORT TRANSFORMATION

- So, now calculate the point on the viewport (X_v, Y_v).

$X_v = 30 + (30 - 20) * (30 / 60) = 35$

$Y_v = 40 + (80 - 40) * (20 / 40) = 60$

So, the point on window (X_w, Y_w) = (30, 80) will be

(X_v, Y_v) = **(35, 60)** on viewport.

Example

- **Window**(100,100,300,300)
- **viewport**(50, 50,150,150)
- Convert window port coordinate(200,200) to viewport

Window to Viewport Transformation

- Consider a viewing coordinate system defined as a Cartesian coordinate system with the origin having world coordinates (u_0, v_0).
- The axis of the viewing coordinate system makes an angle of θ with the world coordinate system $x - axis$.
- The transformation M_{wv} transforms a point with world coordinates P_w into viewing coordinates P_v.
- This transformation can be achieved by first translating by $(-u_0, -v_0)$ and then rotating by $-\theta$.
- So

$$P_v = M_{wv} \cdot P_w$$

where

$$M_{wv} = R(-\theta) \cdot T(-u_0, -v_0)$$

Window to Viewport Transformation

- The window is defined as the rectangle having bottom-left corner at (x_l, y_b), the opposite corner at (x_r, y_t), and edges parallel to the viewing coordinate axes.
- The ***normalizing transformation*** M_{vn} is obtained by first translating by $(-x_l, -y_b)$ and then scaling with factors $\frac{1}{x_r - x_l}$ and $\frac{1}{y_t - y_b}$, so that the lower left and upper right vertices of the window have normalized viewing coordinate $(0{,}0)$ and $(1{,}1)$ respectively.
- So a point with viewing coordinates P_v has normalized viewing coordinates:

$$P_n = M_{vn} \cdot P_v$$

where

$$M_{vn} = S\left(\frac{1}{x_r - x_l}, \frac{1}{y_t - y_b}\right) \cdot T(-x_l, -y_b)$$

Window to Viewport Transformation

- Transforming into the device coordinates is accomplished by the matrix M_{nd}.
- If the viewport is a rectangle with the lower left and upper right corners having device coordinates (u_l, v_b) and (u_r, v_t) respectively, then M_{nd} is achieved by first scaling with factors $(u_r - u_l)$ and $(v_t - v_b)$ and then translating by (u_l, v_b):

$$P_d = M_{nd} \cdot P_n$$

where

$$M_{nd} = T(u_l, v_b) \cdot S(u_r - u_l, v_t - v_b)$$

Window to Viewport Transformation

- Thus, the whole viewing pipeline can be achieved by concatenating the three matrices:

$$M_{wd} = M_{nd} \cdot M_{vn} \cdot M_{wv}$$

$$= T(u_l, v_t) \cdot S(u_r - u_l, v_t - v_b) \cdot S\left(\frac{1}{x_r - x_l}, \frac{1}{y_t - y_b}\right) \cdot R(-\theta) \cdot T(-u_0, -v_0)$$

$$= T(u_l, v_t) \cdot S\left(\frac{u_r - u_l}{x_r - x_l}, \frac{v_t - v_b}{y_t - y_b}\right) \cdot R(-\theta) \cdot T(-u_0, -v_0)$$

- Defining

$$S_x = \frac{u_r - u_l}{x_r - x_l}$$

$$S_y = \frac{v_t - v_b}{y_t - y_b}$$

Window to Viewport Transformation

- If $s_x \neq s_y$, M_{wd} scales objects differentially;
- However if we want objects to be scaled uniformly during the viewing process and that all objects in the window are mapped into the viewport, then we define

$$s' = \min(s_x, s_y)$$

and scale using $S(s', s')$ instead of $S(s_x, s_y)$.

Thank You!

- End of Ch4.