Computer Graphics

2D Viewing & Clipping

Md. Biplob Hosen
Assistant Professor, IIT-JU.
Email: biplob.hosen@juniv.edu

Lecture Outlines

- Coordinate Systems
- 2D Viewing Transformation
- Window to Viewport Mapping

Coordinate Systems

- **Cartesian** – along the x and y axis from (0,0).
- **Polar** – rotation around the angle θ e.g. (r, θ).
- Graphic libraries mostly uses Cartesian coordinates.
- Any polar coordinates must be converted to Cartesian coordinates.
- Four Cartesian coordinates systems in computer Graphics:
    1. Modeling coordinates;
    2. World coordinates;
    3. Normalized device coordinates;
    4. Device coordinates.

Modeling Coordinates

- A coordinate system used for three-dimensional modeling in which each object possesses its own set of coordinates which can then be converted to a set of world coordinates.
- Also known as local coordinate.
- Each object has an origin (0,0).
- So the part of the objects are placed with reference to the object’s origin.
- In terms of scale it is user defined; so coordinate values can be any size.

World Coordinates

- The world coordinate system describes the relative positions and orientations of every generated objects.
- The scene has an origin (0,0).
- The object in the scene are placed with reference to the scenes origin.
- World coordinate scale may be the same as the modeling coordinate scale or it may be different.
- However, the coordinates values can be any size (similar to MC).

Normalized Device Coordinates

- Output devices have their own coordinates.

Coordinate values:

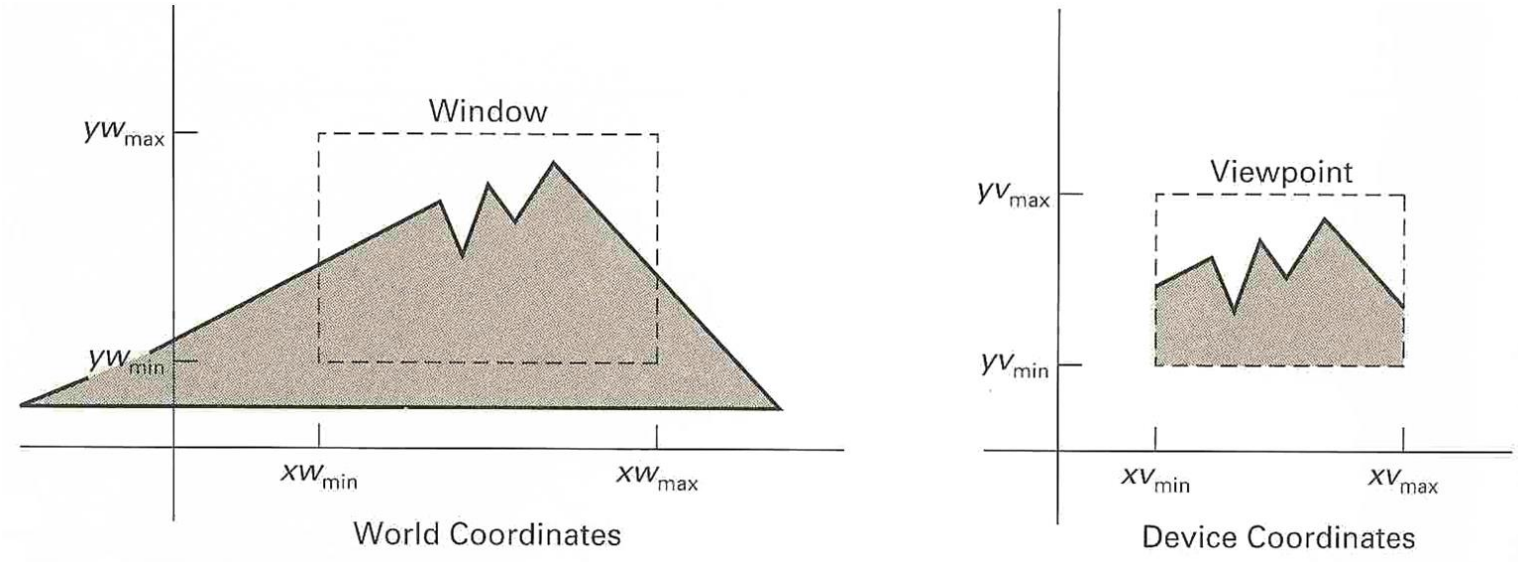
- The x and y axis range from 0 to 1.
- All the x and y coordinates are floating point numbers in the range of 0 to 1.
- This makes the system independent of the various devices coordinates.
- This is handled internally by graphic system without user awareness.

Device Coordinates

- Specific coordinates used by a device.
    - Pixels on a monitor
    - Points on a laser printer.
    - mm on a plotter.
- The transformation based on the individual device is handled by computer system without user concern.

2D Viewing Transformation

- The mapping of a part of a world-coordinate scene to device coordinates.
- 2D viewing transformation = window-to-viewport, windowing transformation.



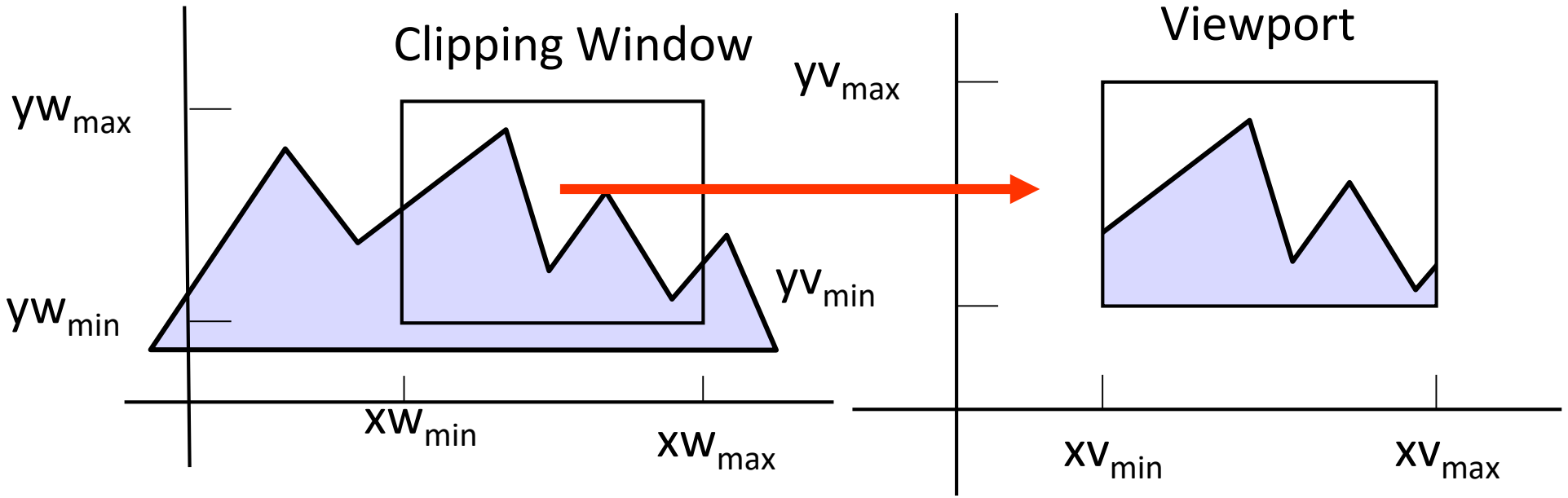
Continue…

- **window**
  - a world-coordinate area selected for display.
  - define what is to be viewed.
- **view port**
  - an area on a display device to which a window is mapped.
  - define where it is to be displayed.
  - define within the unit square.
  - the unit square is mapped to the display area for the particular output device in use at that time.
- **windows & viewport**
  - be rectangles in standard position, with the rectangle edges parallel to the coordinate axes.

2D Viewing Transformation Steps

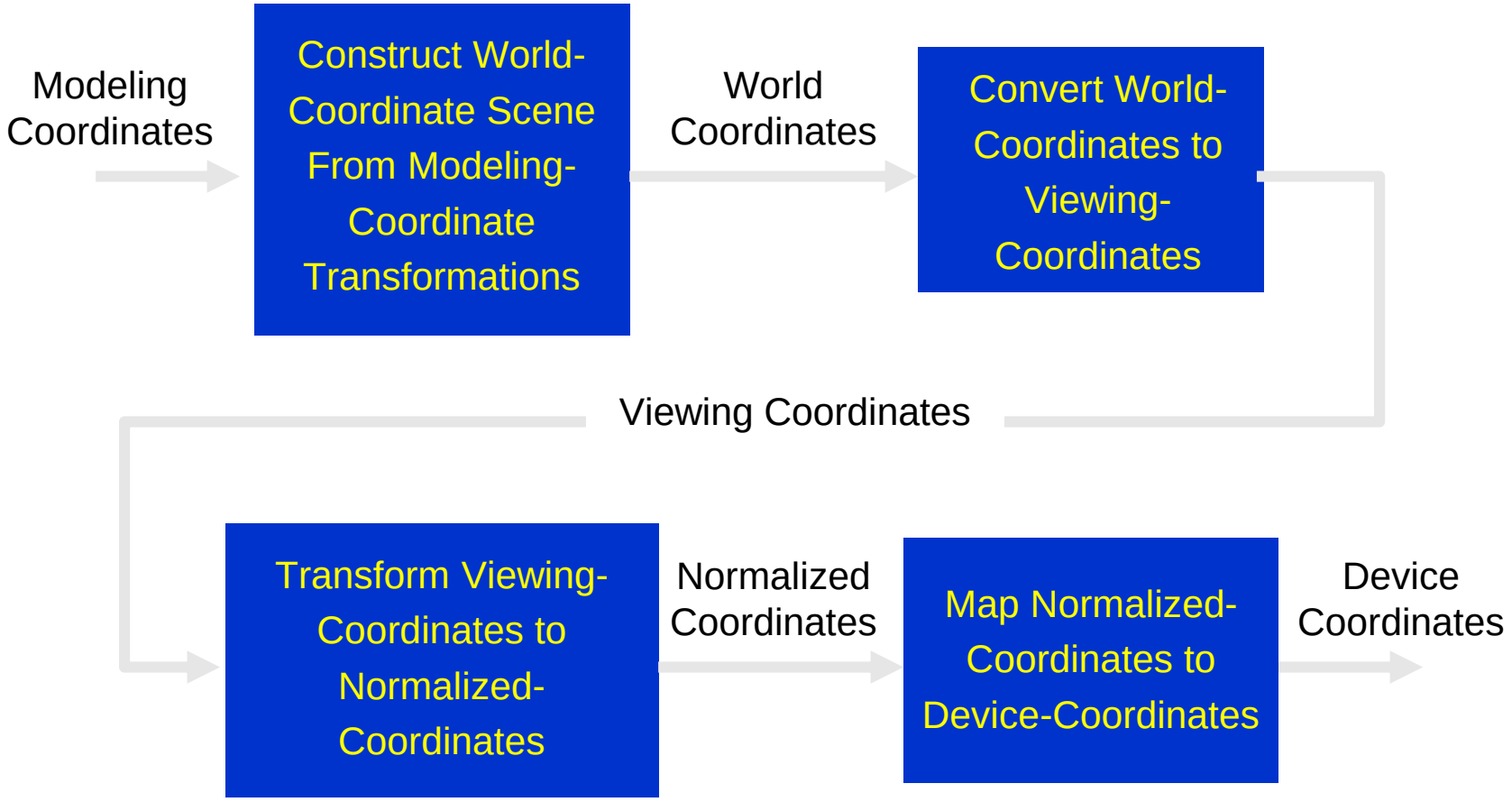
1. Construct the world-coordinate scene.
2. Transform descriptions in world coordinates to viewing coordinates.
3. Map the viewing-coordinate description of the scene to normalized coordinates.
4. Transfer to device coordinates.

Example


Clipping Window
Viewport
yw_{max}
yw_{min}
yv_{max}
yv_{min}
xw_{min}
xw_{max}
xv_{min}
xv_{max}
World Coordinate
Device Coordinate

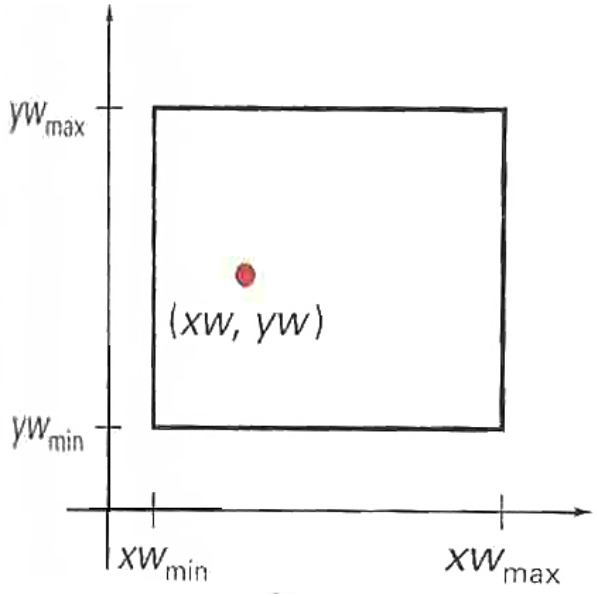
Viewing Transformation Pipeline

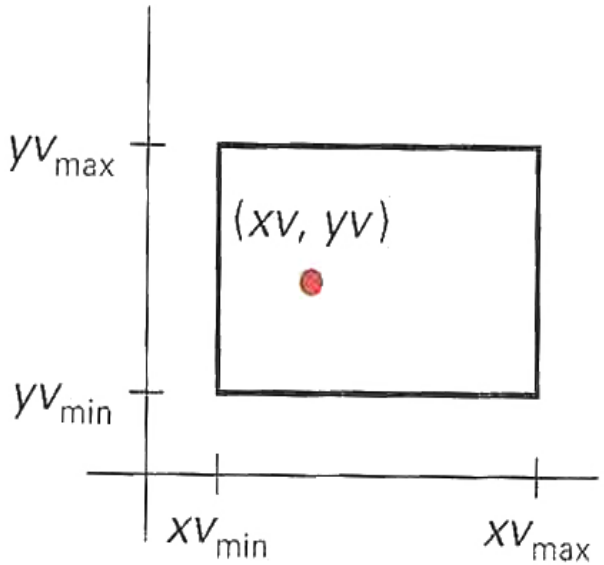
- The two-dimensional viewing-transformation pipeline



Window to Viewport Mapping

A point at position (xw, yw) in a designated window is mapped to viewport coordinates (xv, yv) so that relative positions in the two areas are the same.





Continue…

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

$$\frac{xv - xv_{\min}}{xv_{\max} - xv_{\min}} = \frac{xw - xw_{\min}}{xw_{\max} - xw_{\min}} \quad \text{and} \quad \frac{yv - yv_{\min}}{yv_{\max} - yv_{\min}} = \frac{yw - yw_{\min}}{yw_{\max} - yw_{\min}}$$

Thus-

$$xv = xv_{\min} + (xw - xw_{\min})sx$$
$$yv = yv_{\min} + (yw - yw_{\min})sy$$

where

$$sx = \frac{xv_{\max} - xv_{\min}}{xw_{\max} - xw_{\min}}$$
$$sy = \frac{yv_{\max} - yv_{\min}}{yw_{\max} - yw_{\min}}$$

Continue…

Now we can express those two formulas for computing (vx, vy) from (wx, wy) in terms of a **translate-scale-translate** transformation N.

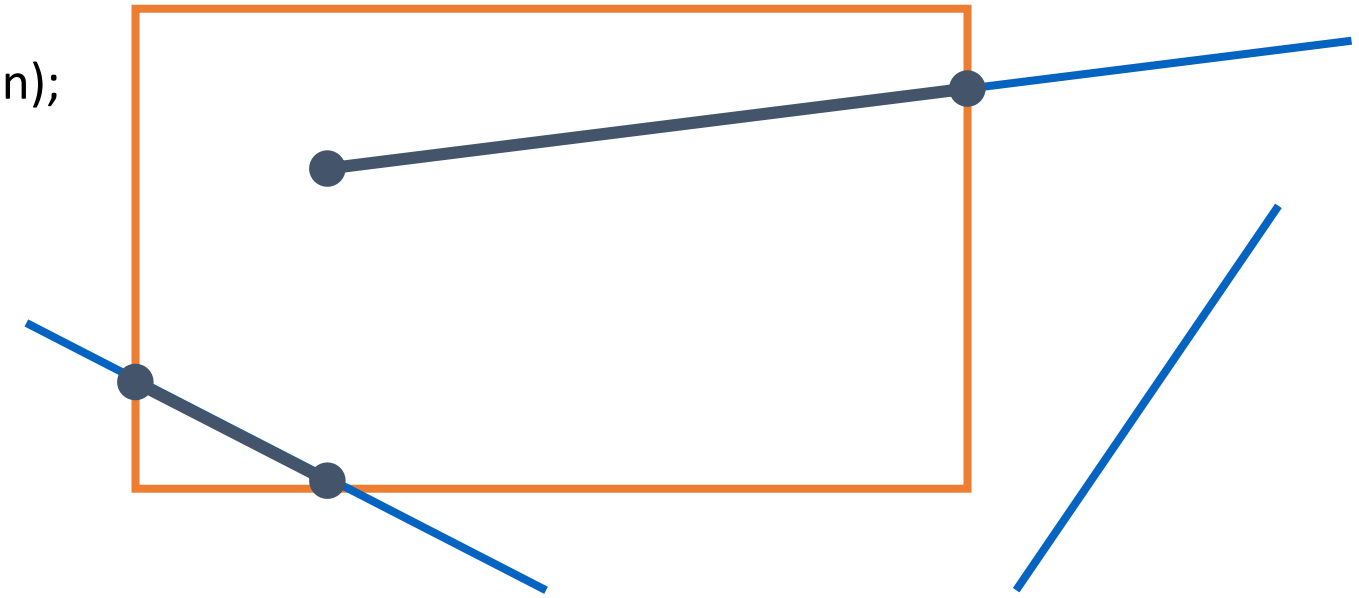
$$\begin{pmatrix} vx \\ vy \\ 1 \end{pmatrix} = N. \begin{pmatrix} wx \\ wy \\ 1 \end{pmatrix}$$

where

$$\mathrm{N} = \begin{bmatrix} 1 & 0 & xv_{\min} \\ 0 & 1 & yv_{\min} \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} \dfrac{xv_{\max} - xv_{\min}}{xw_{\max} - xw_{\min}} & 0 & 0 \\ 0 & \dfrac{yv_{\max} - yv_{\min}}{yw_{\max} - yw_{\min}} & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} 1 & 0 & -xw_{\min} \\ 0 & 1 & -yw_{\min} \\ 0 & 0 & 1 \end{bmatrix}$$

Clipping

- Analytically calculating the portions of primitives within the viewport.
- Adaptive primitive types:
  - Point;
  - Line;
  - Area (e.g. Polygon);



Point Clipping

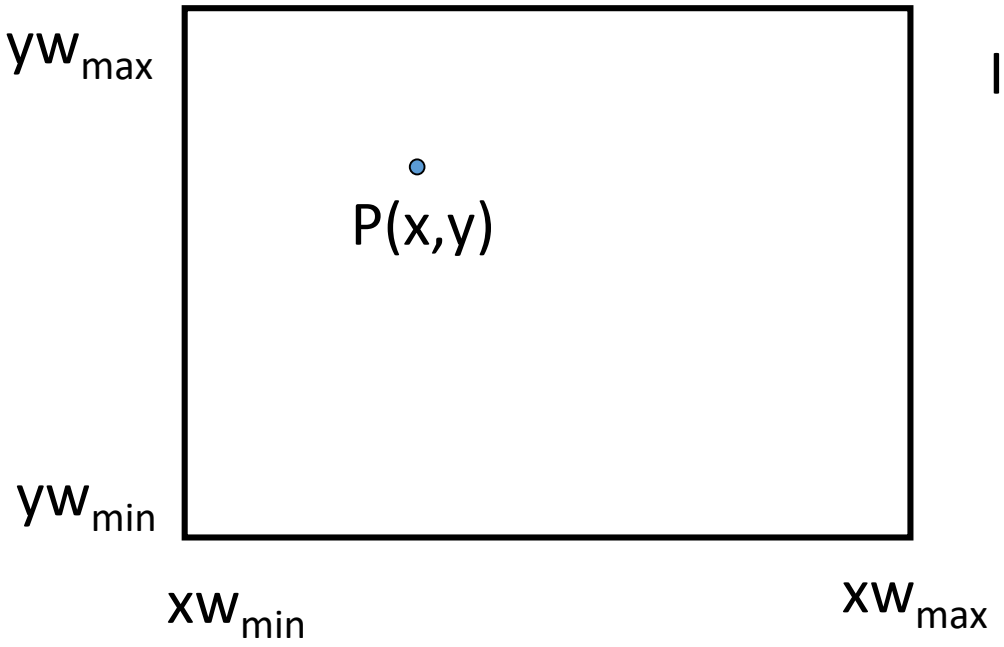
- Assuming that the clip window is a rectangle in standard position.
- For a clipping rectangle in standard position, we save a 2-D point P(x, y) for display if the following inequalities are satisfied:

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

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

- If any one of these four inequalities is not satisfied, the point is clipped (not saved for display).
- Where $x_{\min}, x_{\max}, y_{\min}, y_{\max}$ define the clipping window.

Point Clipping



If P(x,y) is inside the window?

$$xw_{\min} \leq x \leq xw_{\max}$$

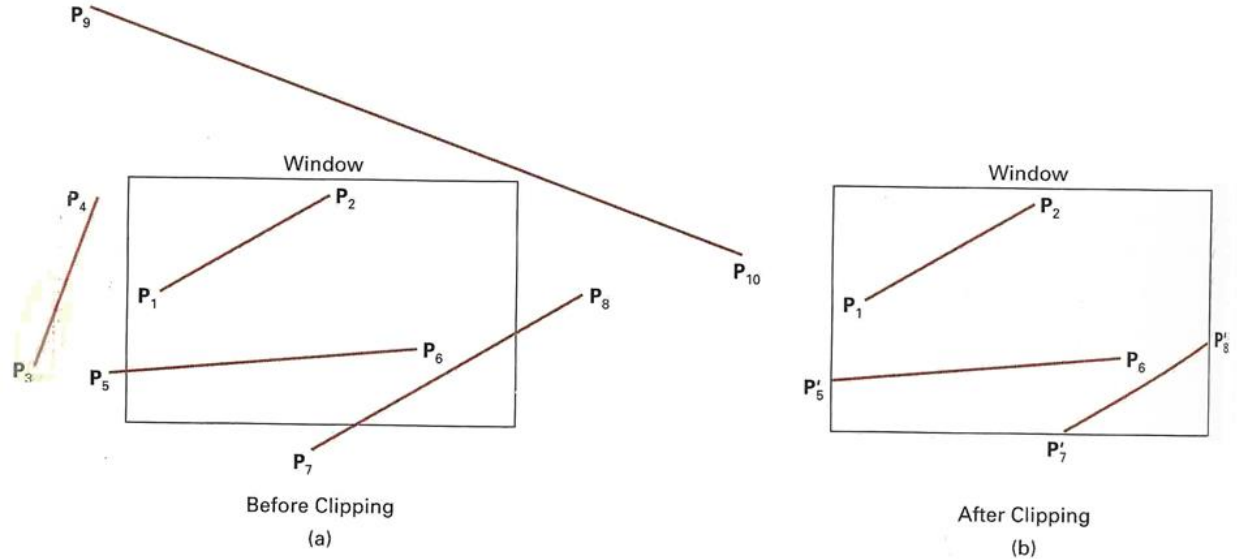
$$yw_{\min} \leq y \leq yw_{\max}$$

Line clipping

- **Line clipping procedure**
  - test a given line segment to determine whether it lies completely inside the clipping window.
  - if it doesn’t, we try to determine whether it lies completely outside the window.
  - if we can’t identify a line as completely inside or completely outside, we must perform intersection calculations with one or more clipping boundaries.

Line clipping

- Checking the line endpoints ⇒ inside-outside test.



- Line clipping Algorithm:
  - Cohen-Sutherland line clipping Algorithm;
  - Liang-Barsky line clipping Algorithm.

Thank you☺