Salahaddin University-Erbil/College of Science
Department of Computer Science & IT

Computer Graphics

Lecture 13

Tarza Hasan

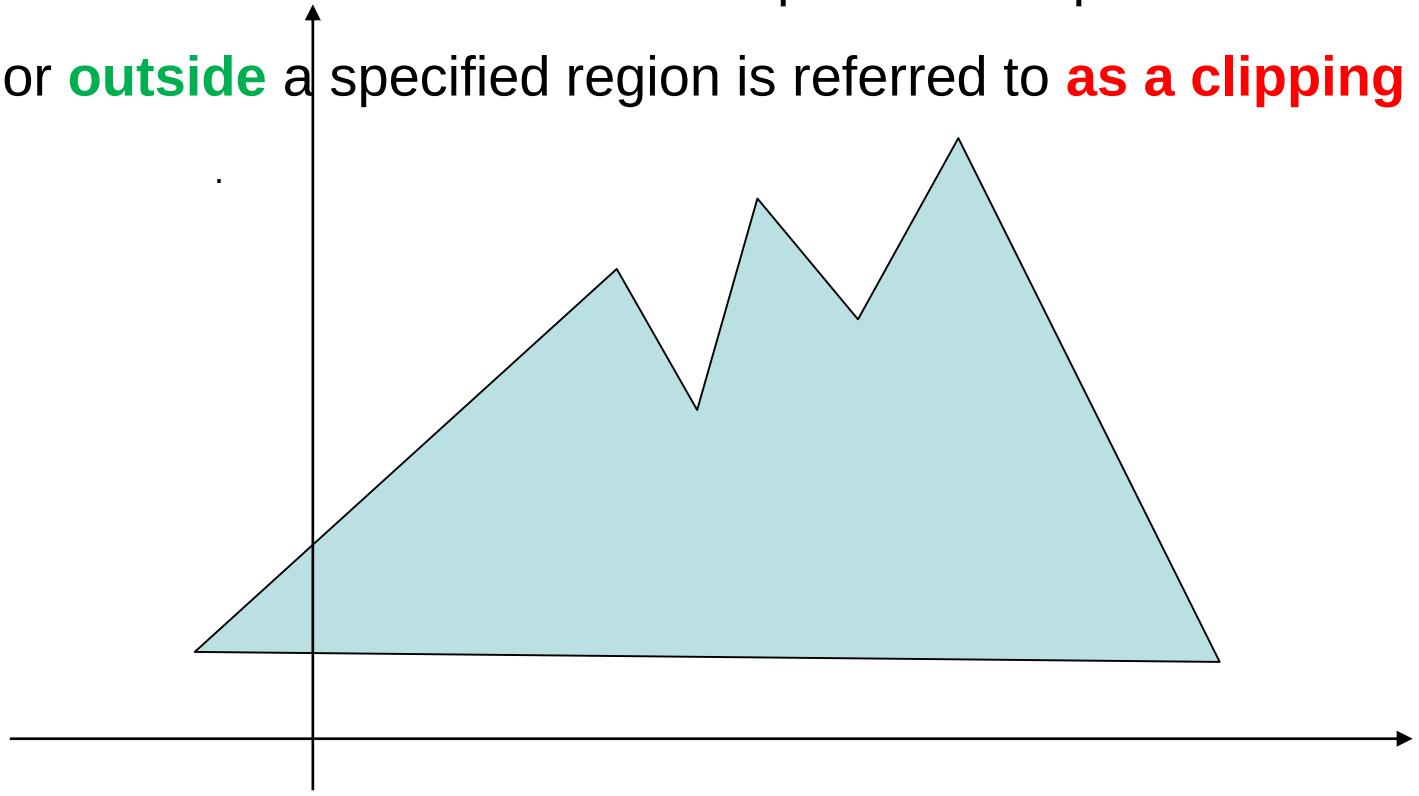
tarza.abdullah@su.edu.krd

Contents

- Clipping
  - **Clipping concept**
  - **Point clipping**
  - **Line clipping**
  - **Cohen Sutherland line clipping algorithm**

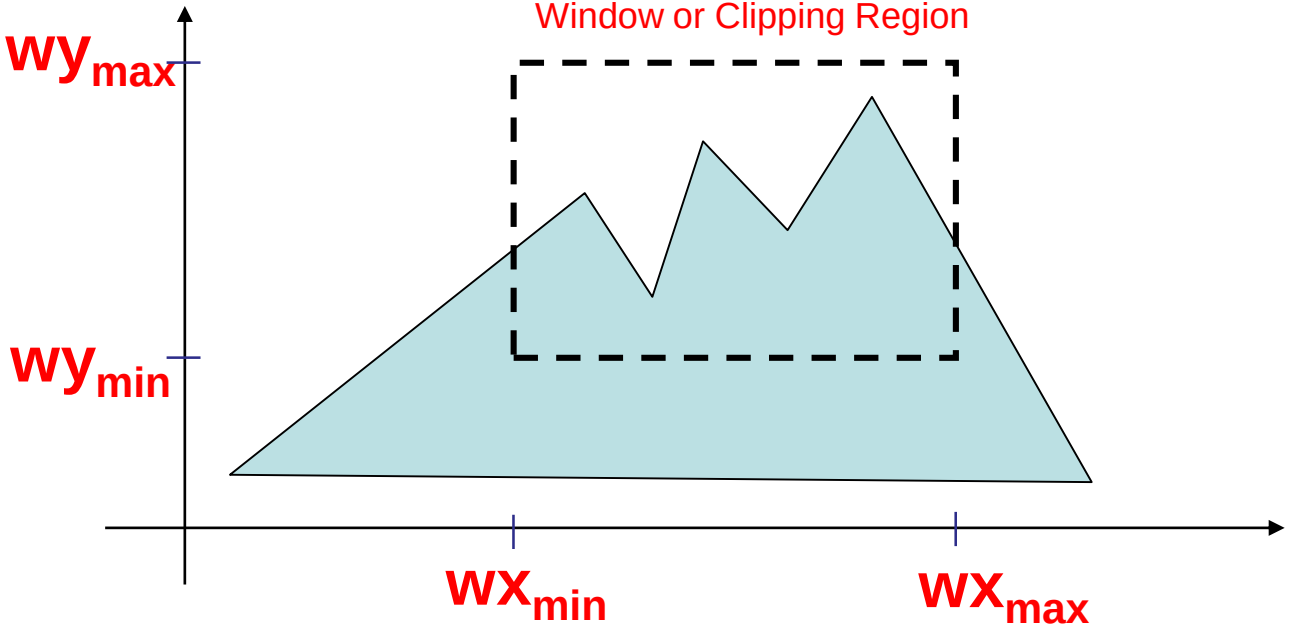
Clipping Concept

Any **procedure** that **eliminates** those portion of a picture that are either **inside** or **outside** a specified region is referred to **as a clipping algorithm.**



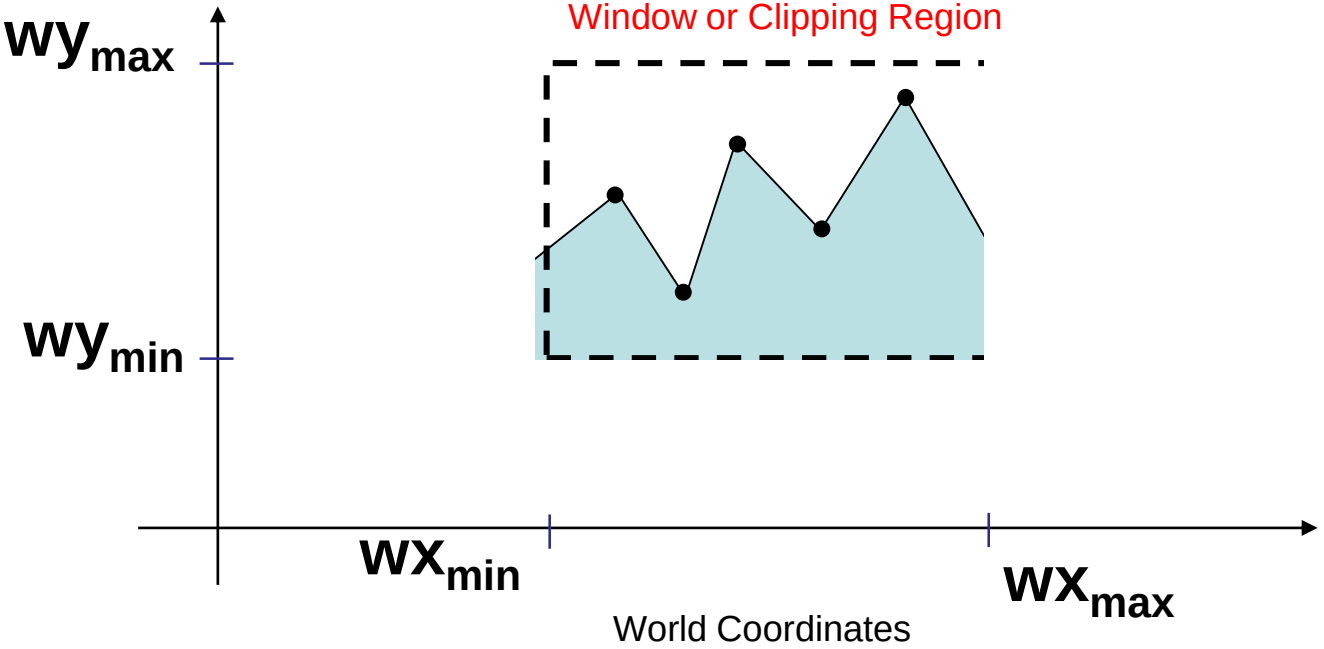
Clipping Concept (cont…)

When we display a scene, only the objects within a particular window are displayed



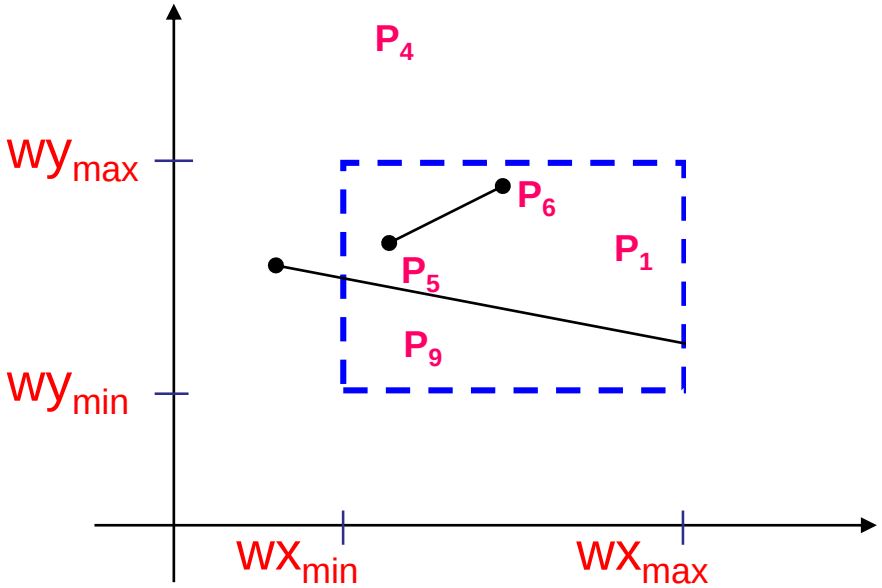
Clipping Concept (cont…)

Because drawing things to a display takes time, we *clip* everything outside the window



Clipping Concept (cont…)

For the image below consider **which lines** and **points** should be **kept** and which ones should be **clipped?????**



Point Clipping

Assuming that the **clip window** is a **rectangle** in standard position, we save a point P = (x, y) **for display** if the following inequalities are satisfied:

$$wx_{min} \leq x \leq wx_{max} \text{ AND } wy_{min} \leq y \leq wy_{max}$$

The **point** is either **inside** the view pane (window) **or outside** the view pane.

Example: Let us have a view pane (window). The coordinates of the window are:

(xwmin, xwmax) - For X-axis of the window

(ywmin, ywmax) - For Y-axis of the window

Point Clipping(cont…)

Steps of Point Clipping:

Step 1: First, we set the value of xw_{min} and xw_{max} , and Yw_{min} and Yw_{max}coordinates for the window

Step 2: Set the coordinates of a given point P(x,y).

Step 3: Check the condition of $(wx_{min} \leq x \leq wx_{max}$ **AND** $wy_{min} \leq y \leq wy_{max})$

Step 4: If

Point coordinates lie between the **(xw_{min}, xw_{max})** and **(yw_{min}, yw_{max})**

Then

{Display the point in the view pane**}**

Else

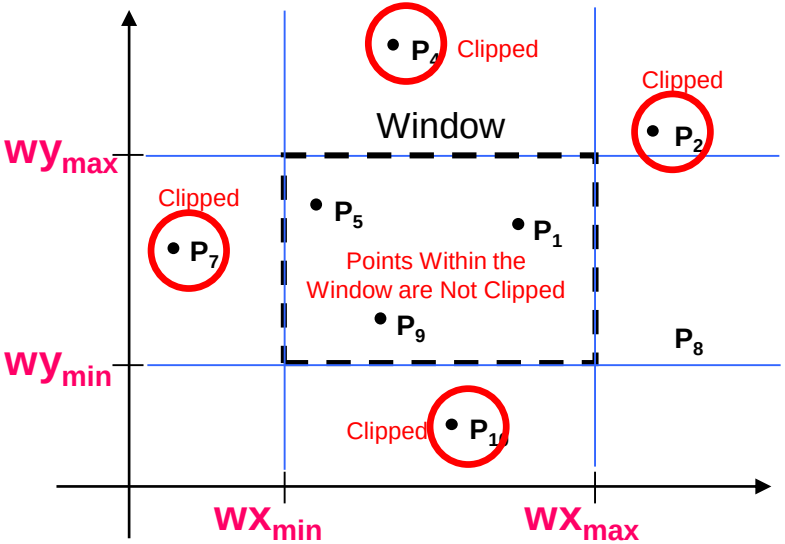
{Remove the point**}**

Step 5: Stop.

Point Clipping(cont…)

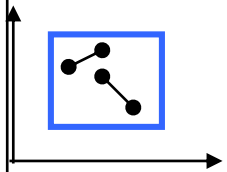
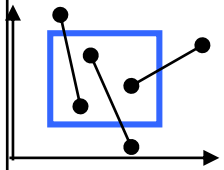
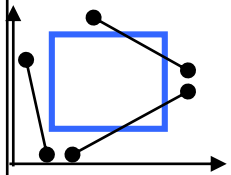
Easy - a point (x,y) is **not** clipped if:

$$wx_{min} \le x \le wx_{max} \textbf{ AND } wy_{min} \le y \le wy_{max}$$



Line Clipping

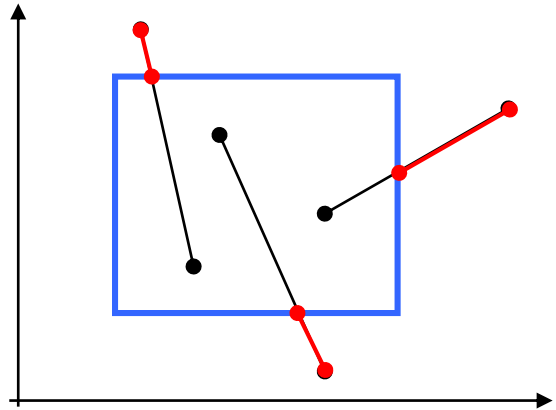
Harder - examine the **end-points** of each line to see if they are in the window or not

Situation	Solution	Example
Both end-points inside the window	Don't clip	
One end-point inside the window, one outside	Must clip	
Both end-points outside the window	Reject	

Brute force line clipping

Brute force line clipping can be performed as follows:

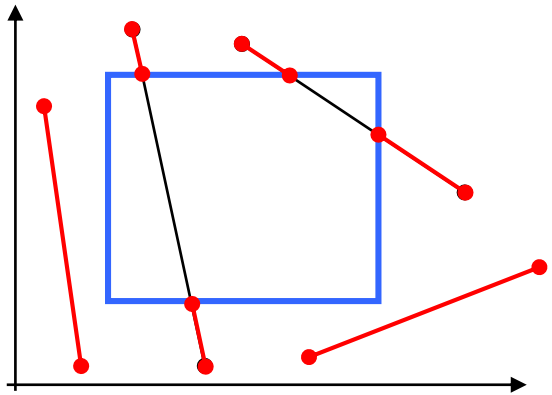
- Don’t clip lines with both end-points within the window



- For lines with one end-point inside the window and one end-point outside, calculate the intersection point (using the equation of the line) and clip from this point out

Brute Force Line Clipping (cont…)

- For lines with **both end-points outside the window** **test** the line for **intersection** with all of the window boundaries, and clip appropriately



- However, calculating line intersections is **computationally expensive.**

Because a scene can contain **so many lines,** the brute force approach to clipping is too much slow

Cohen-Sutherland line Clipping Algorithm

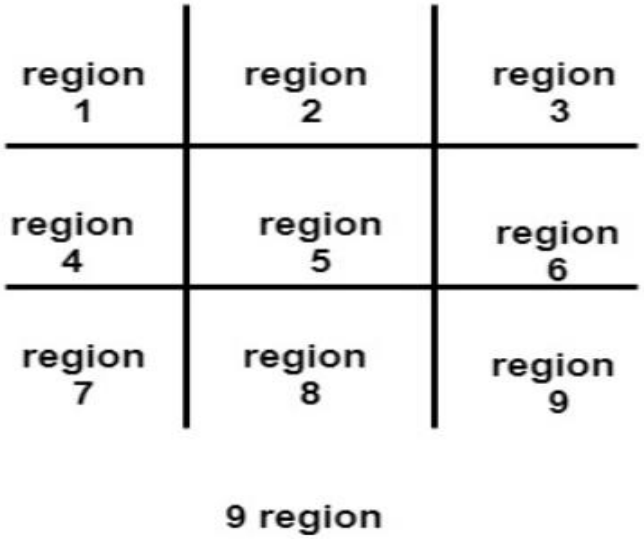
An efficient line clipping algorithm

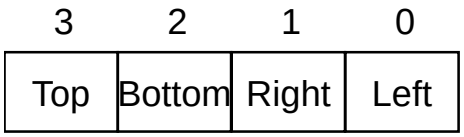
The key advantage of the algorithm is that it **vastly reduces** the number of **line intersections** that must be calculated.

Cohen-Sutherland

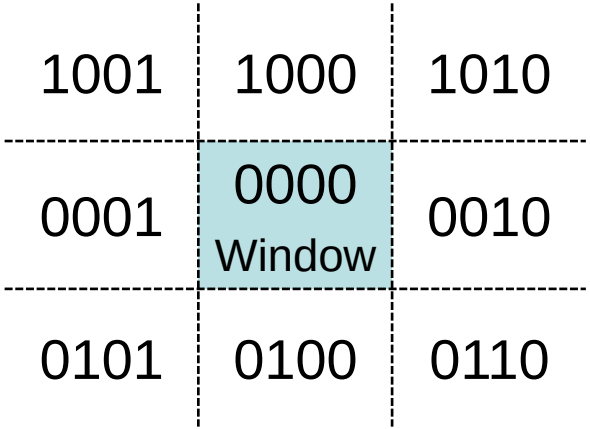
The space is **divided** into regions based on the window boundaries

- Each region has a unique four bit region code
- **Region codes** indicate **the position of the regions** with respect to the window



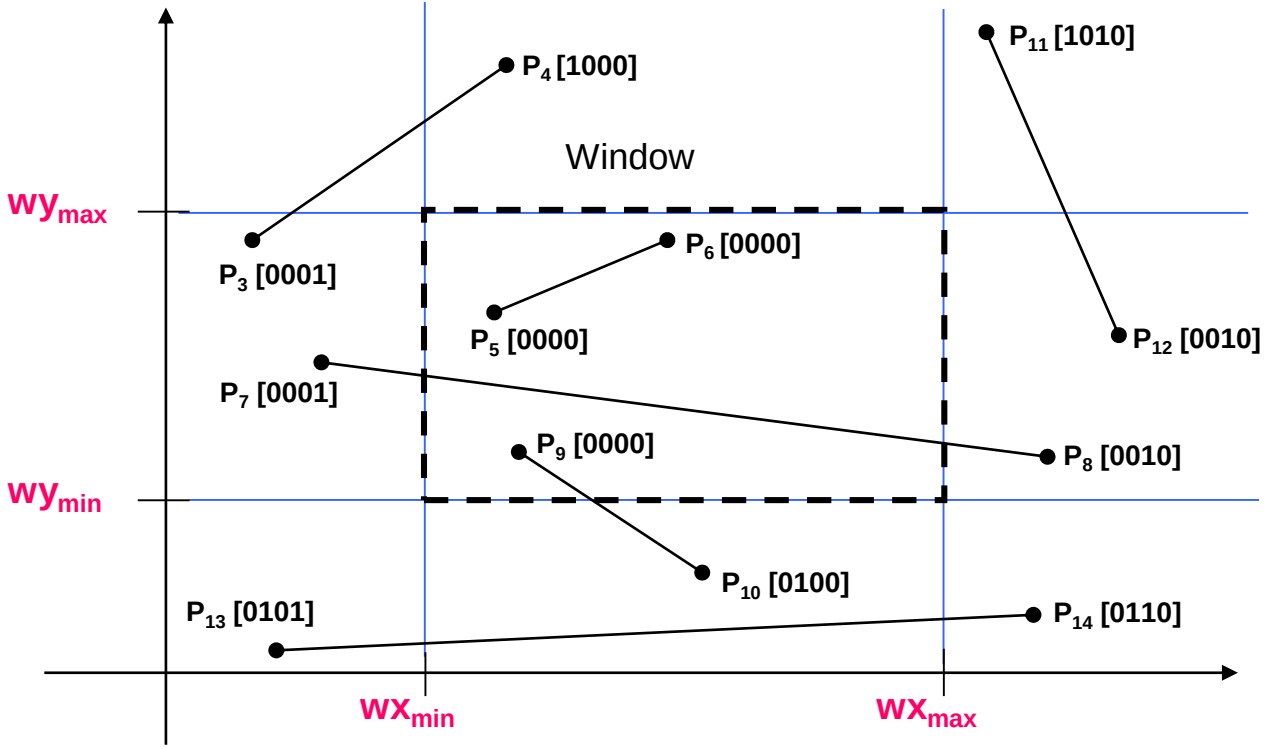


Region Code Legend



Cohen-Sutherland: Labelling

Every **end-point** is labelled with the appropriate **region code**



Cohen-Sutherland Algorithm

Step 1 − Assign a **region code** for each endpoints.

Step 2 − **If** both endpoints have a region code **0000** then **accept** this line.

Step 3 − **Else,** perform the logical **AND** operation for both region codes.

Step 3.1 − **If** the result is **not 0000,** then **reject** the line.

Step 3.2 − **Else** you need clipping.

Step 3.2.1 − Choose **an endpoint** of the line that is **outside** the window.

Step 3.2.2 − Find the **intersection point** at the window boundary based on region code

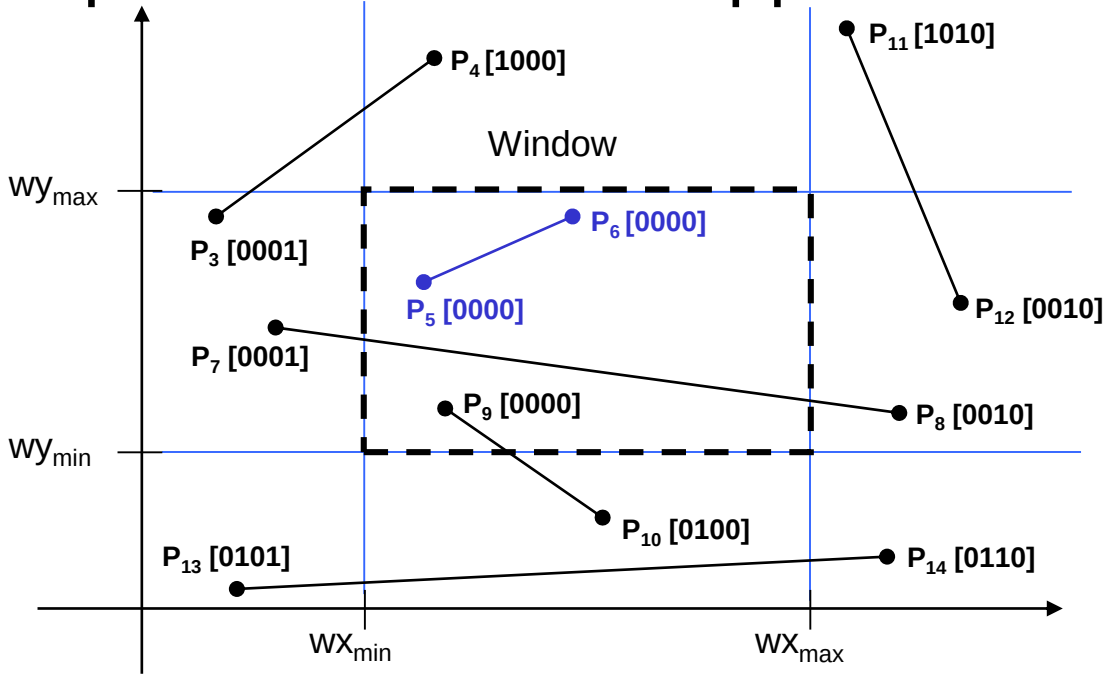
Step 3.2.3 − **Replace endpoint** with the **intersection point** and **update** the region code.

Step 3.2.4 − **Repeat step 2** until we find a clipped line either trivially **accepted** or trivially **rejected.**

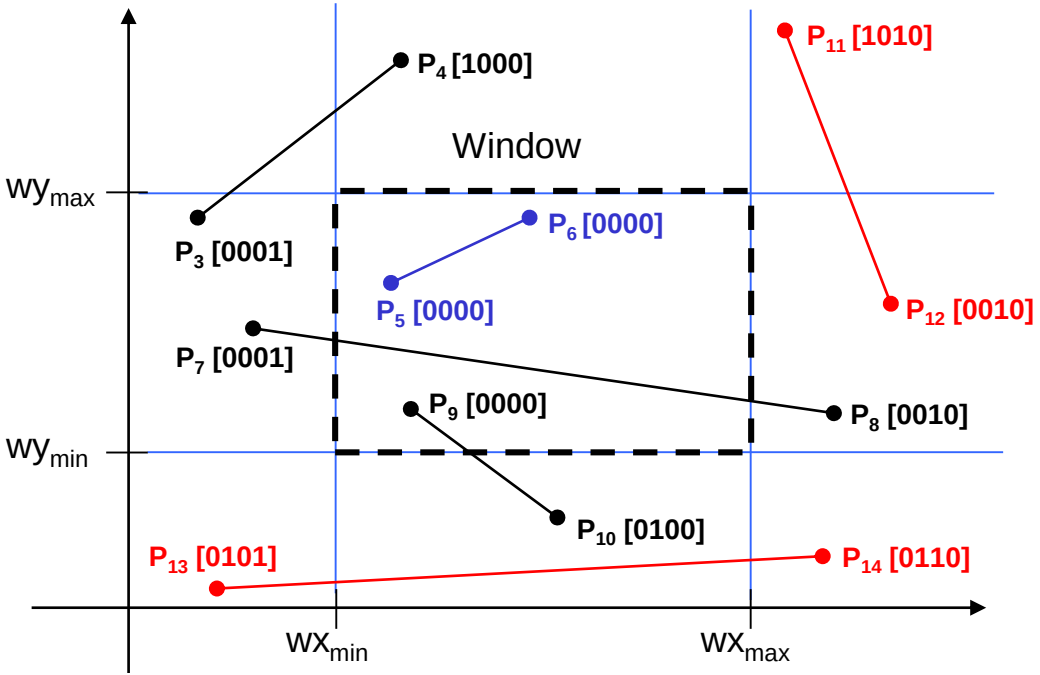
Step 4 − Repeat step 1 for other lines.

Cohen-Sutherland: Lines In The Window

Lines completely contained the region code [0000] for both end-points so are not clipped.



Cohen-Sutherland: Lines Outside The Window



Cohen-Sutherland Examples

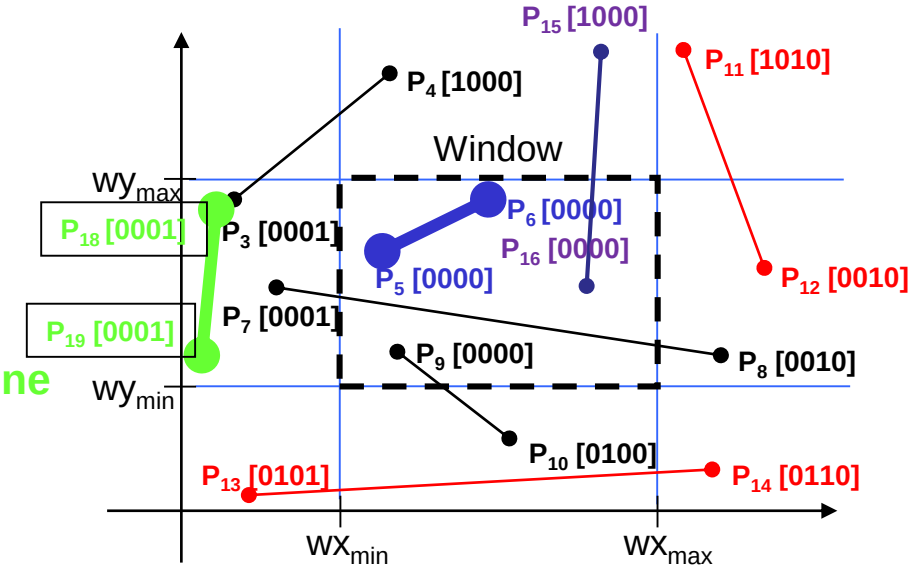
Consider the line P_{18} to P_{19}

P_{18} =0001➔Non Zero

P_{19} =0001➔Non Zero

AND➔0001➔Non Zero

Completely outside the window,so reject the line



Consider the line **P_5 to P_6**

P_5 =0000➔ Zero

P_6 =0000➔ Zero

AND➔0000➔Zero, which indicates that the line **P_5 & P_6** are completely inside the window and no clipping is required, so accept the line.

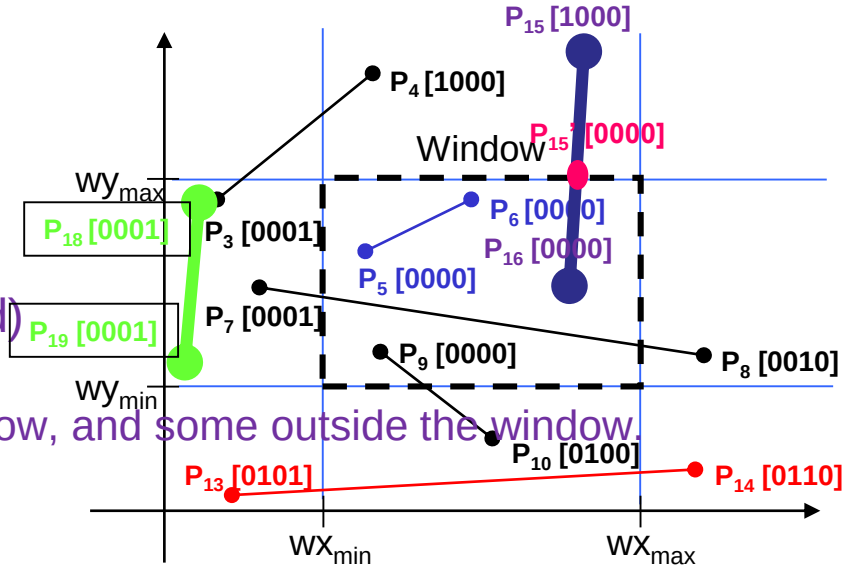
Cohen-Sutherland Examples

Consider the line P_{15} to P_{16}

P_{15} =1000➔Non Zero

P_{16} =0000➔ Zero

AND➔0000➔ Zero (clipping is required)



Some portion of the line is inside the clipping window, and some outside the window.

Find the **intersection point $P_{15'}$**

$P_{15'}$ =0000➔ Zero

P_{16} =0000➔ Zero

AND➔0000➔Zero(accept $P_{15'}$ & P_{16} and clip P_{15} & P_{16})

Cohen-Sutherland Examples

Consider the line P_7 to P_8

P_7 =0010➔Non Zero

P_8 =0100➔ Non Zero

AND➔0000➔ Zero (So, clipping is required, you have to perform clipping two times)

Line **P_7 to $P_{7'}$** is clipped

$P_{7'}$ =0000➔ Zero

P_8 =0100➔ Non Zero

AND➔0000➔ Zero (clipping is required)

Line **P_8 to $P_{8'}$** is clipped

$P_{8'}$ =0000➔ Zero

$P_{7'}$ =0000➔ Zero

AND➔0000➔ Zero (accept the line)