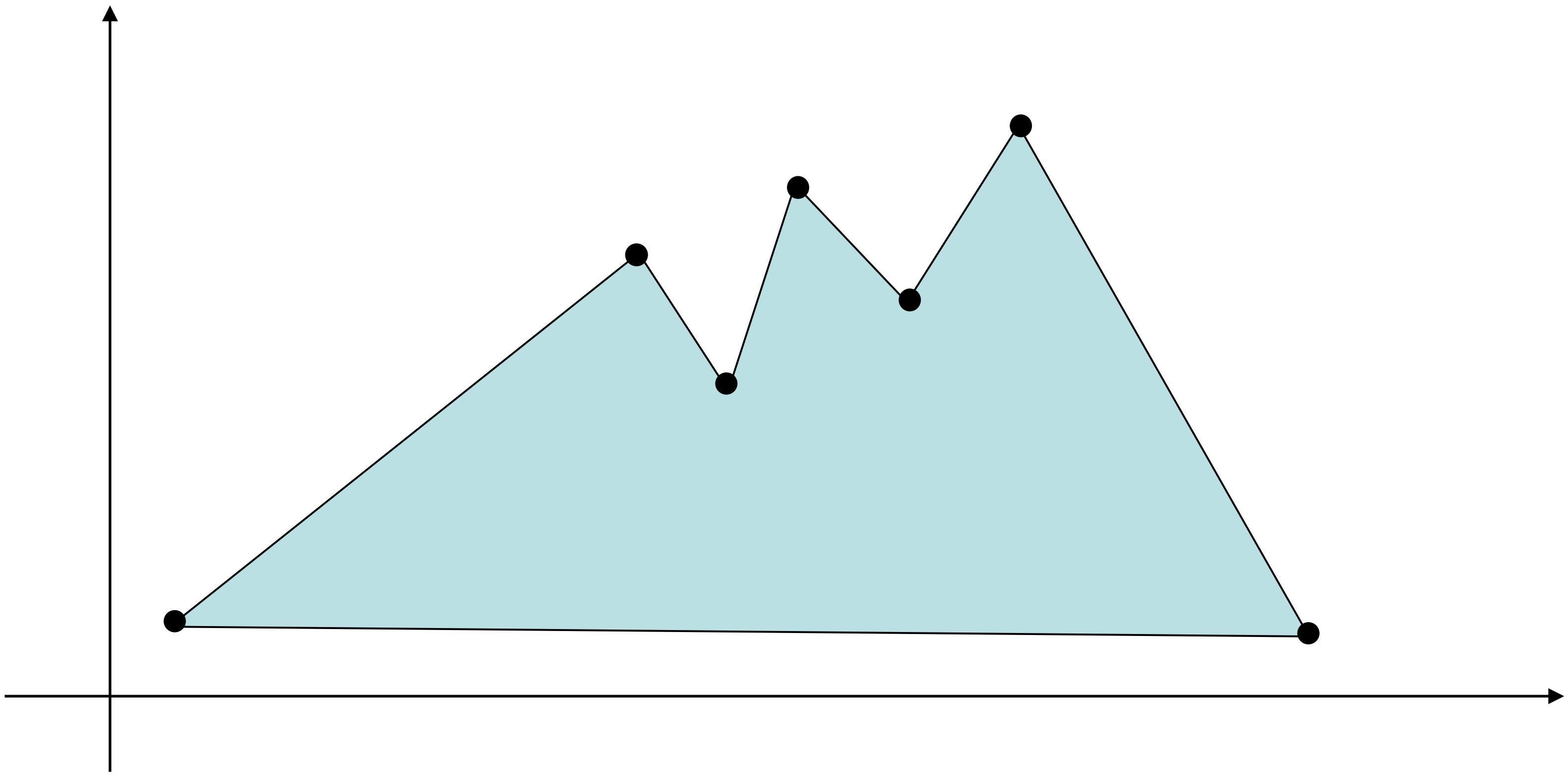
Dr. George Karraz, Ph. D.

Viewing In 2D

Contents

Windowing I

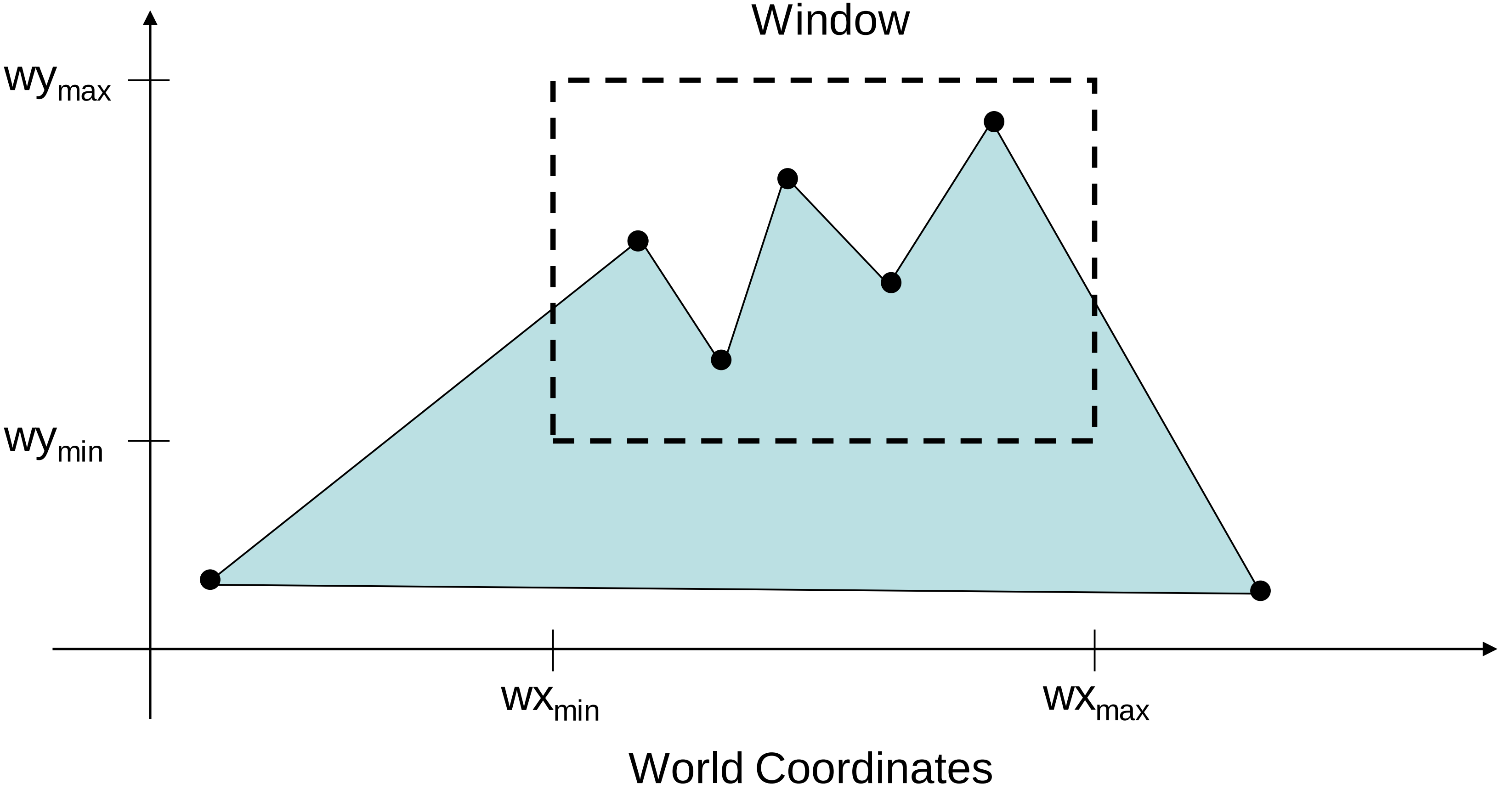
A scene is made up of a collection of objects specified in world coordinates



World Coordinates

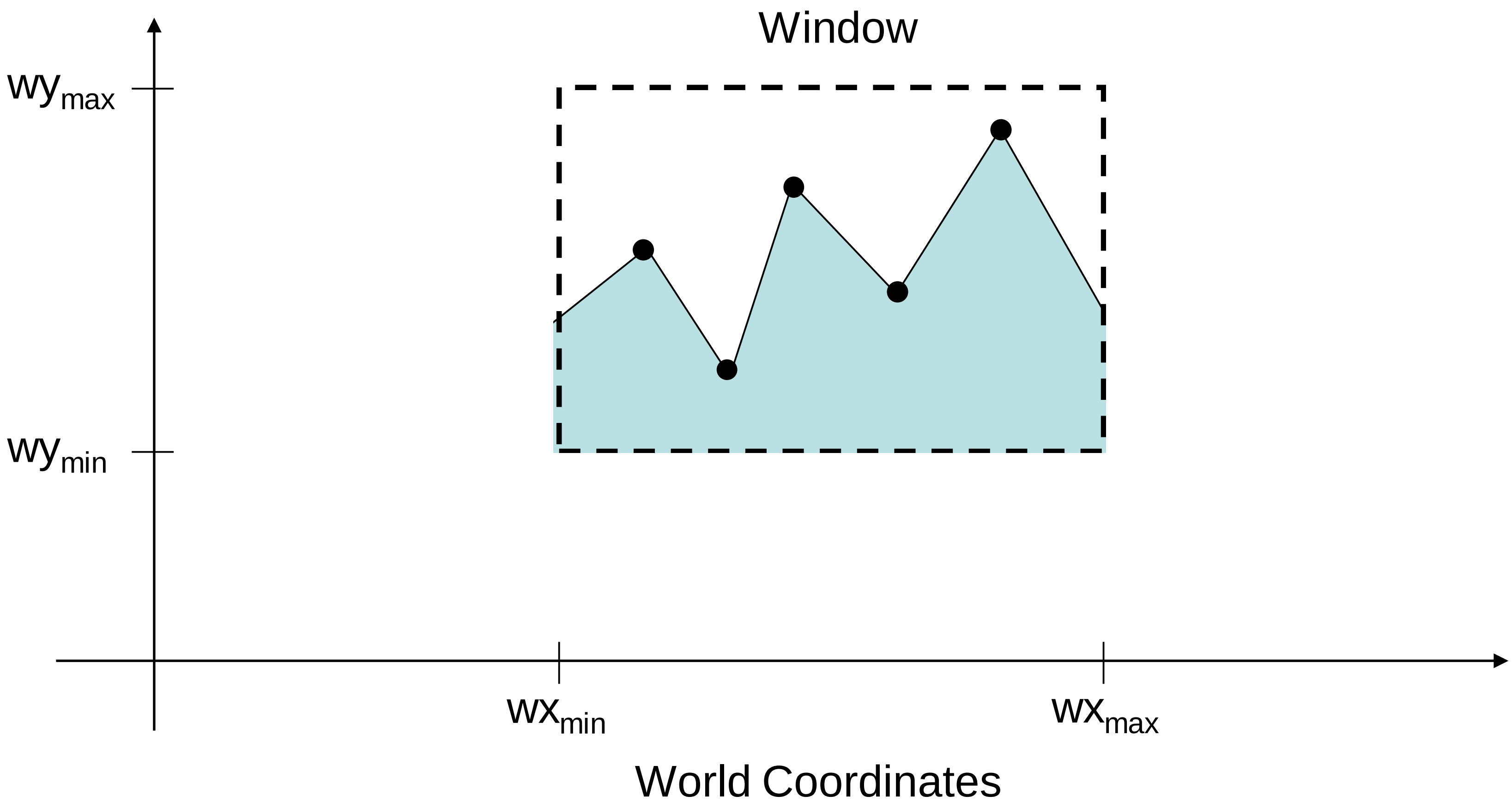
Windowing II

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



Windowing III

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



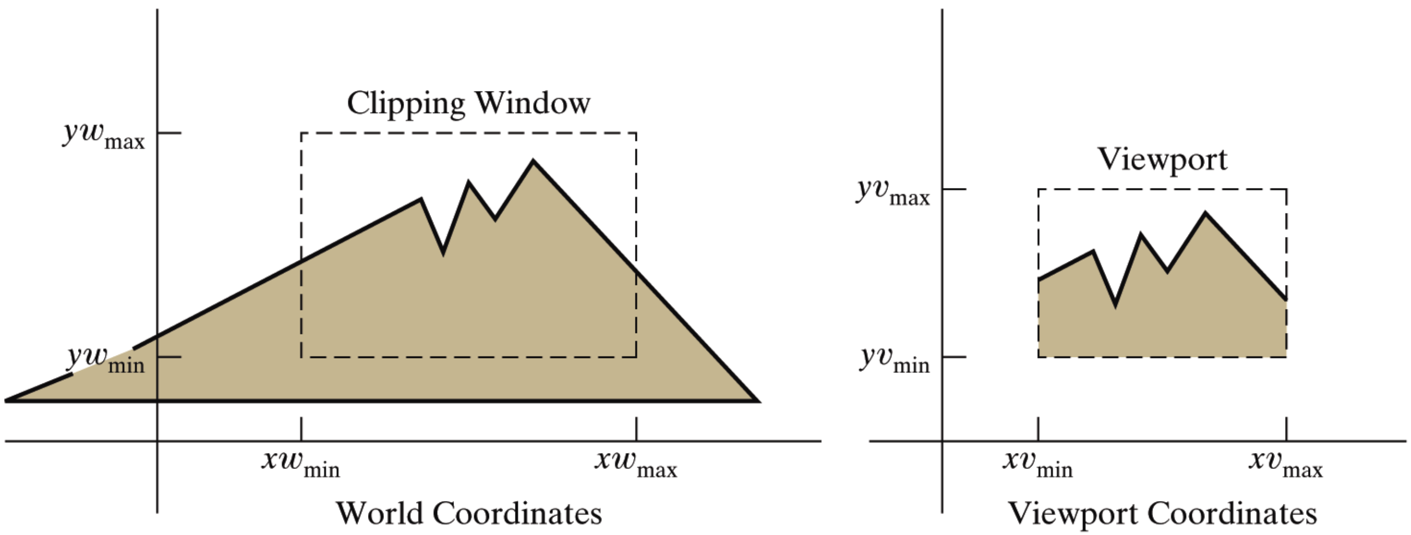


Figure 6-2

A clipping window and associated viewport, specified as rectangles aligned with the coordinate axes.

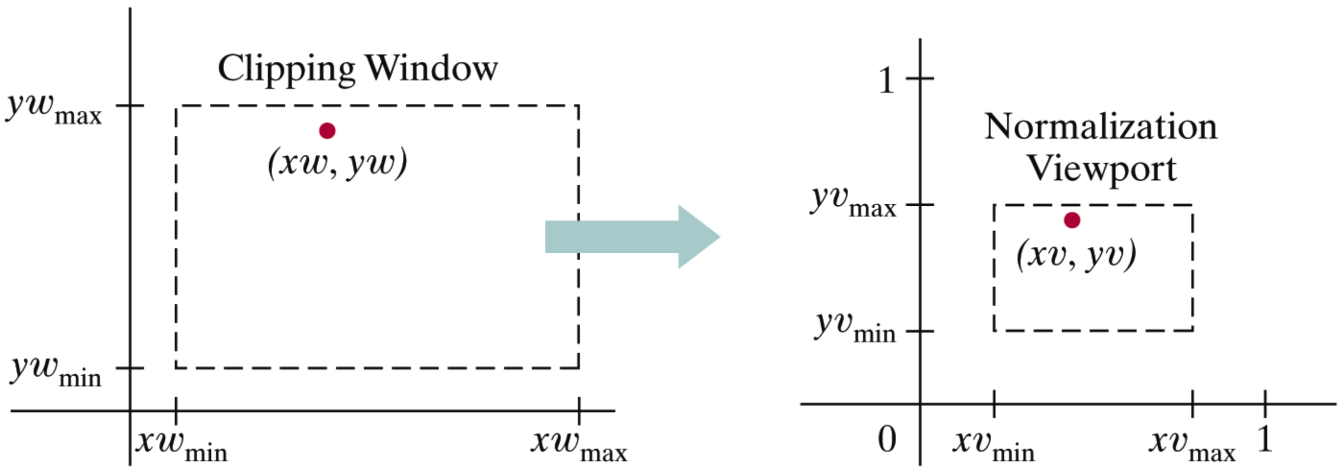


Figure 6-7

A point (xw, yw) in a world-coordinate clipping window is mapped to viewport coordinates (xv, yv), within a unit square, so that the relative positions of the two points in their respective rectangles are the same.

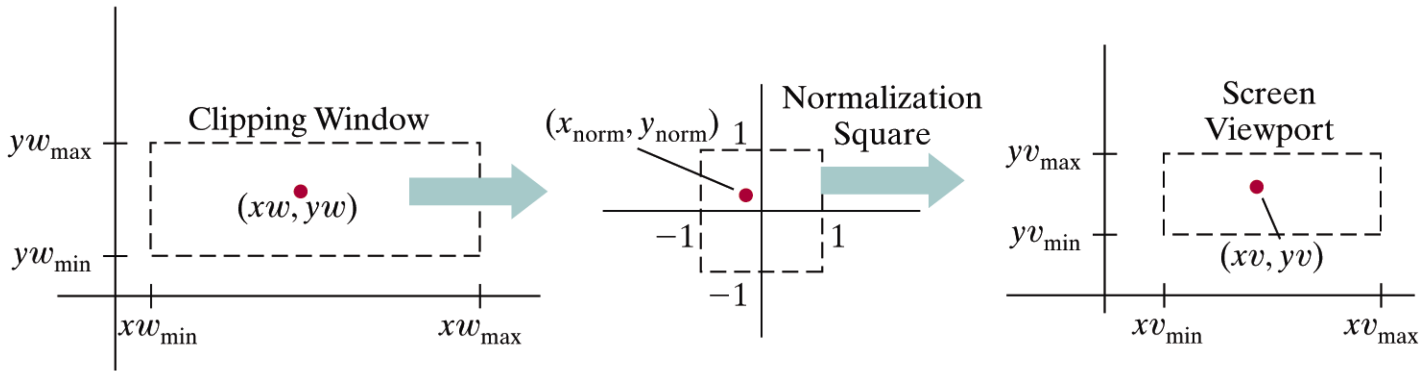


Figure 6-8

A point (xw, yw) in a clipping window is mapped to a normalized coordinate position (x_{norm}, y_{norm}), then to a screen-coordinate position (xv, yv) in a viewport. Objects are clipped against the normalization square before the transformation to viewport coordinates.

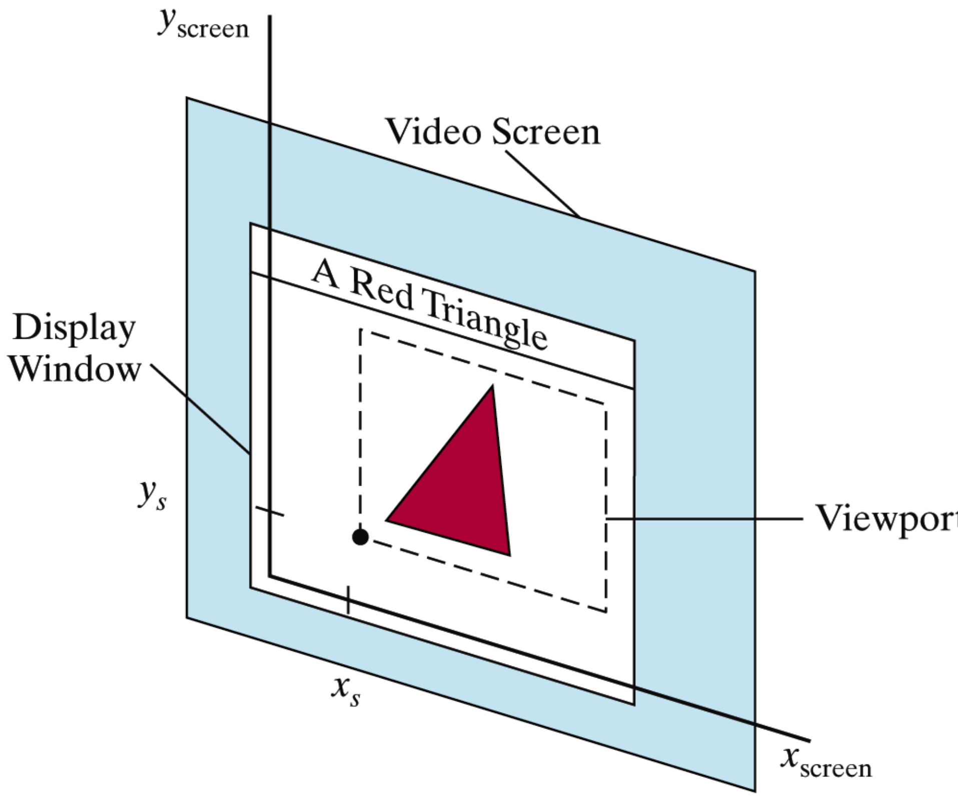


Figure 6-9

A viewport at coordinate position (x_s , y_s) within a display window.

Coordinate Systems

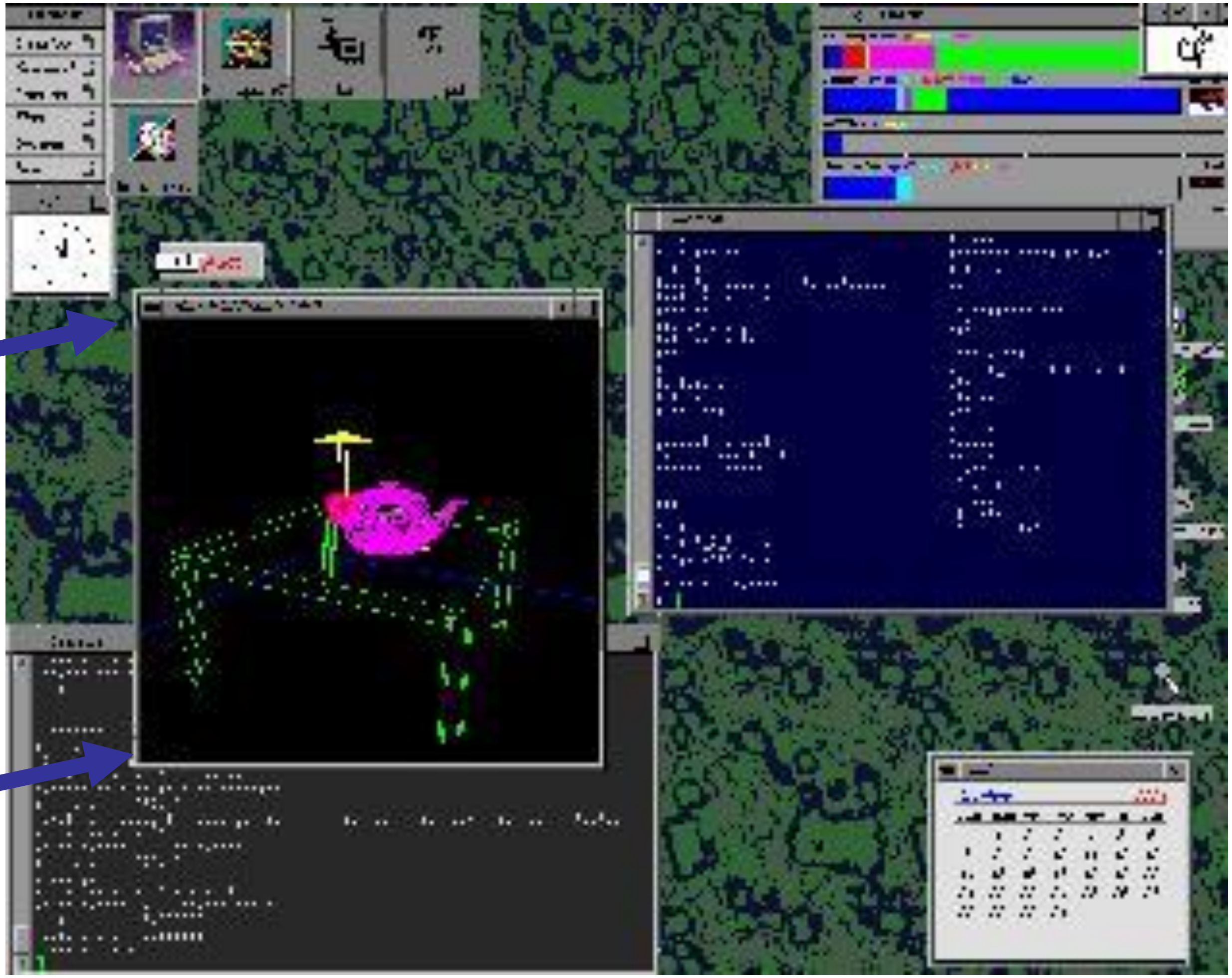
- Screen Coordinate system
- World Coordinate system
- World window
- Viewport
- Window to viewport mapping

Screen Coordinate System

Glut

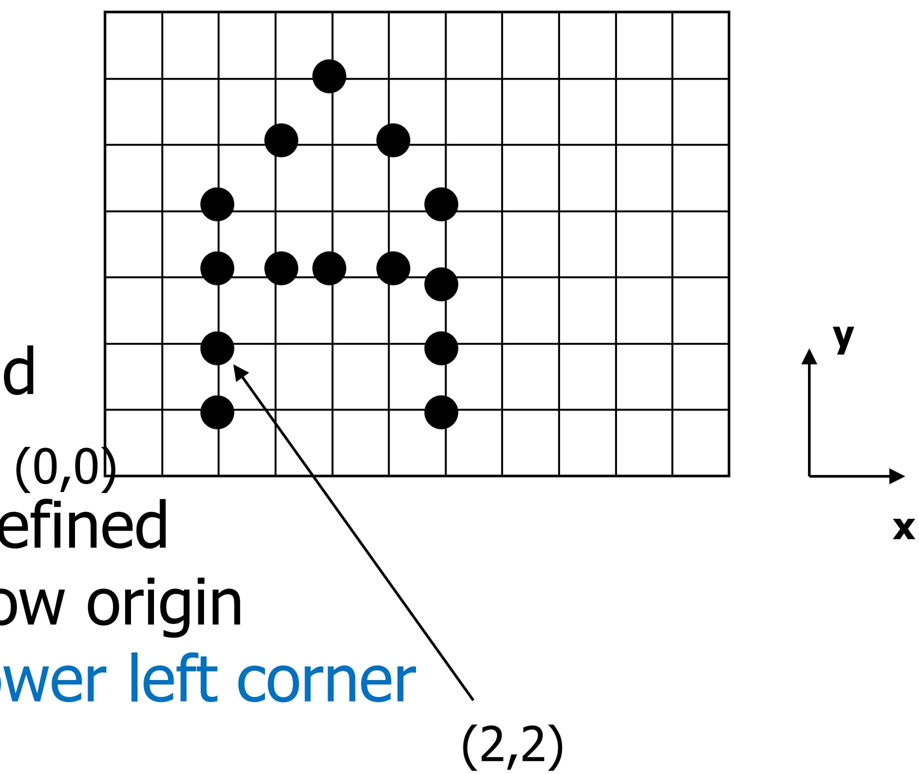
OpenGL

(0,0)



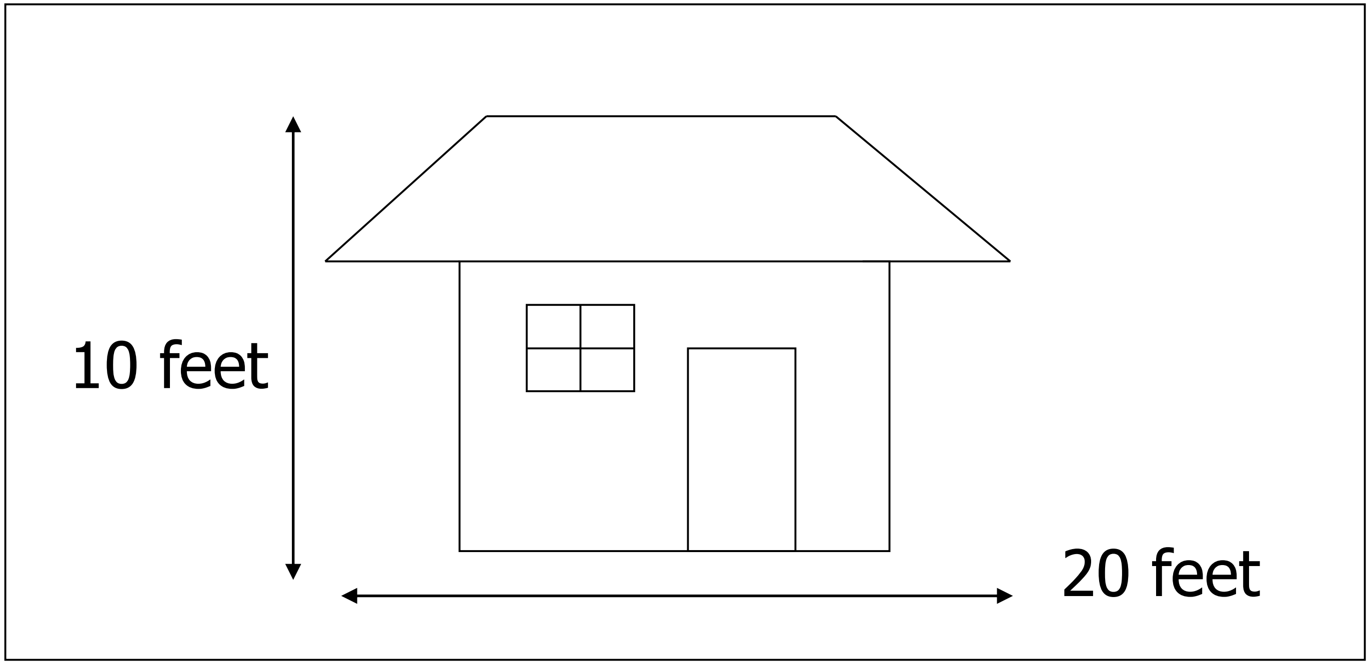
Screen Coordinate System

- 2D Regular Cartesian Grid
- Origin (0,0) at lower left corner
- Horizontal axis – x
  Vertical axis – y
- Pixels are defined at the grid intersections
- This coordinate system is defined relative to the display window origin (OpenGL convention: the lower left corner of the window)

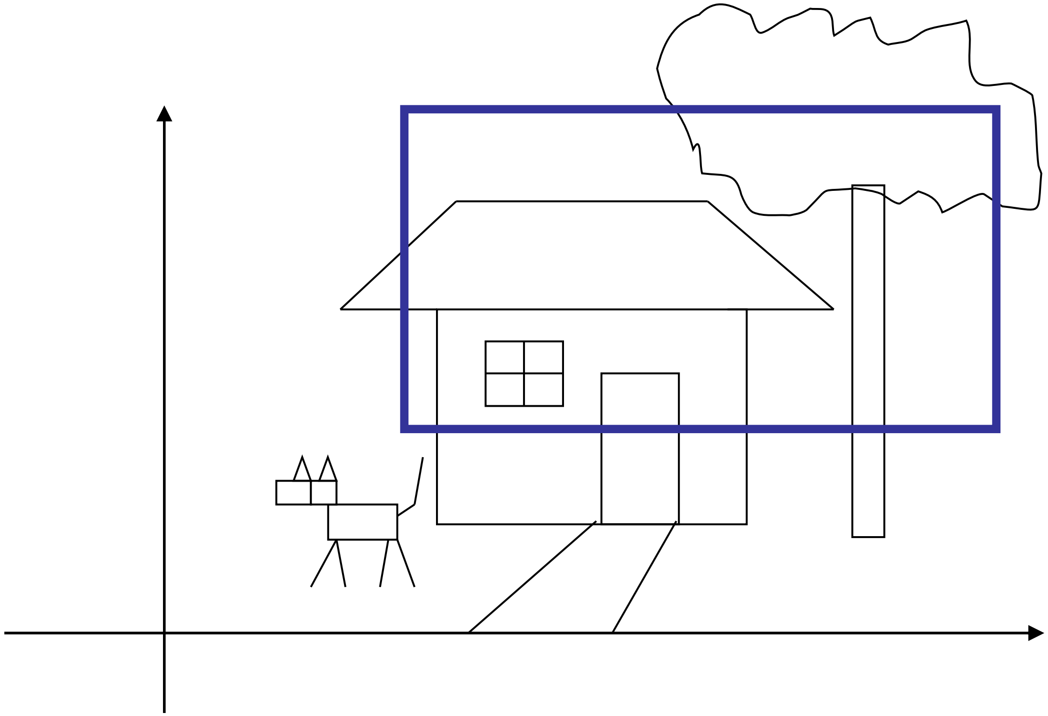


World Coordinate System

- Screen coordinate system is not easy to use

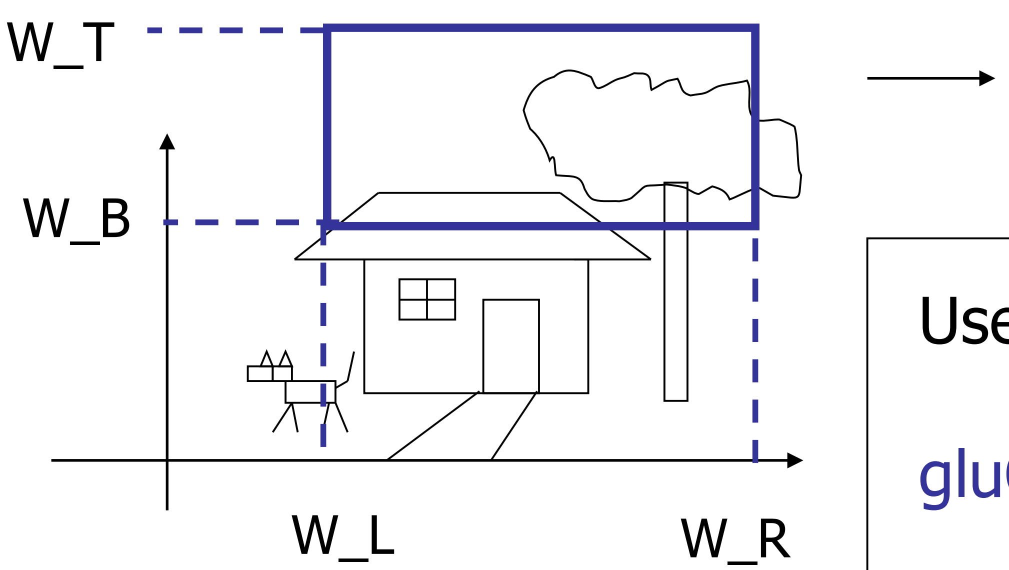


Define a world window



World Window

- World window – a rectangular region in the world that limits our view



Define by

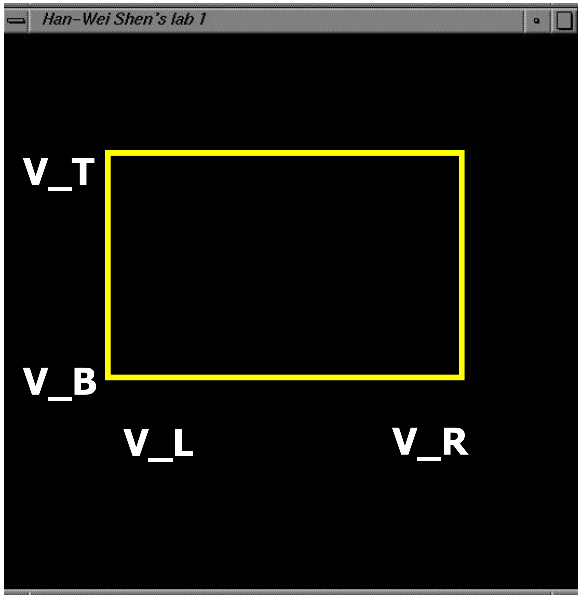
W_L, W_R, W_B, W_T

Use OpenGL command:

gluOrtho2D(left, right, bottom, top)

Viewport

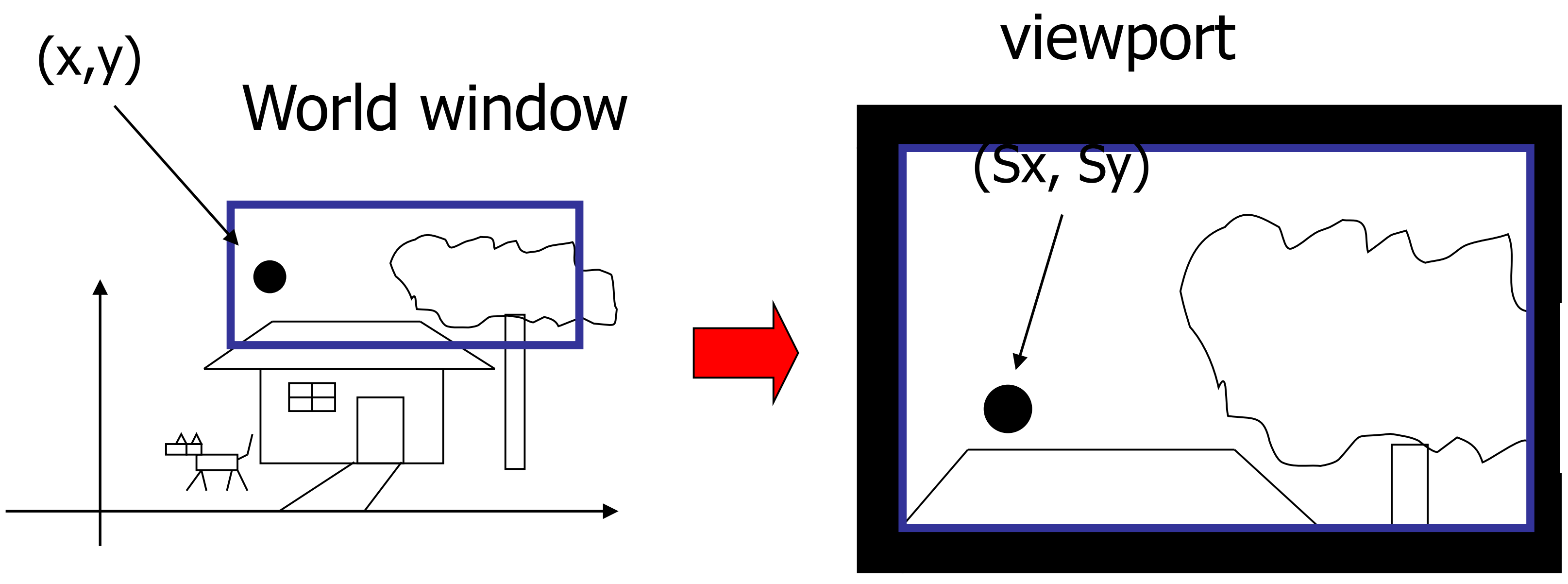
- The rectangular region in the screen that maps to our world window
- Defined in the window's (or control's) coordinate system




```
glViewport(int  left, int  bottom,
                   int  (right-left),
                   int (top-bottom));
```

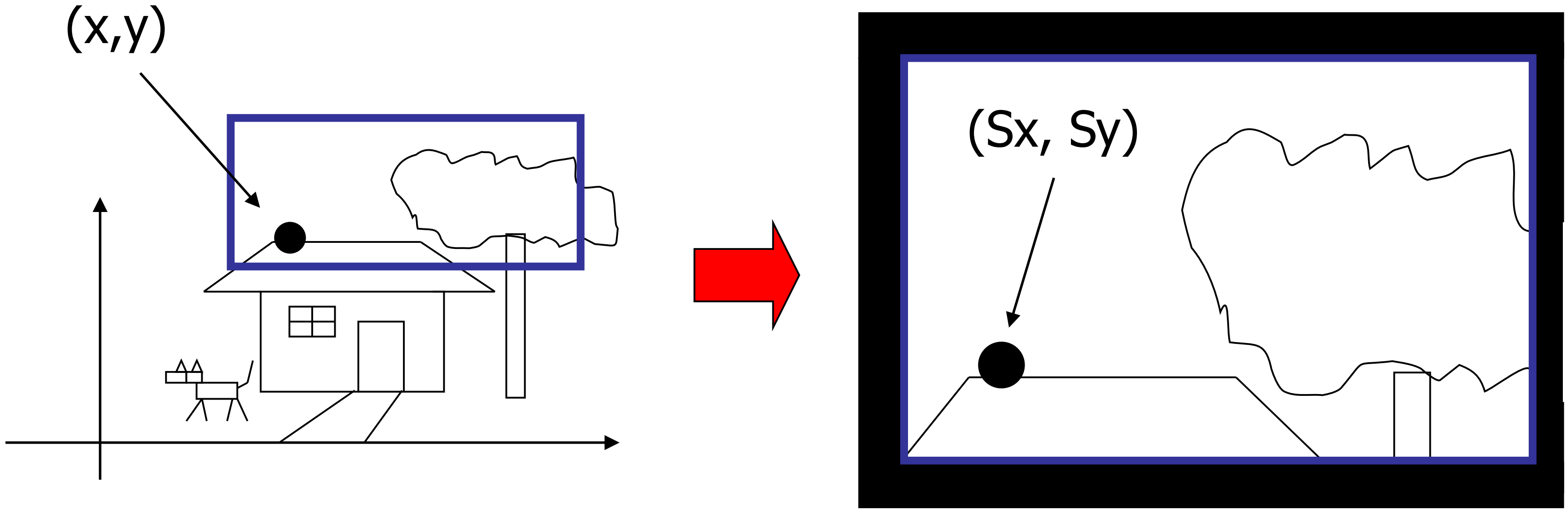
Window to viewport mapping

- The objects in the world window will then be drawn onto the viewport



Window to viewport mapping

- How to calculate (sx, sy) from (x,y)?



Window to viewport mapping

- First thing to remember – you don't need to do it by yourself. OpenGL will do it for you
  - You just need to define the viewport (with glViewport()), and the world window (with gluOrtho2D())
- But we will look 'under the hood'

Also, one thing to remember …

- A practical OpenGL issue
  - Before calling gluOrtho2D(), you need to have the following two lines of code –

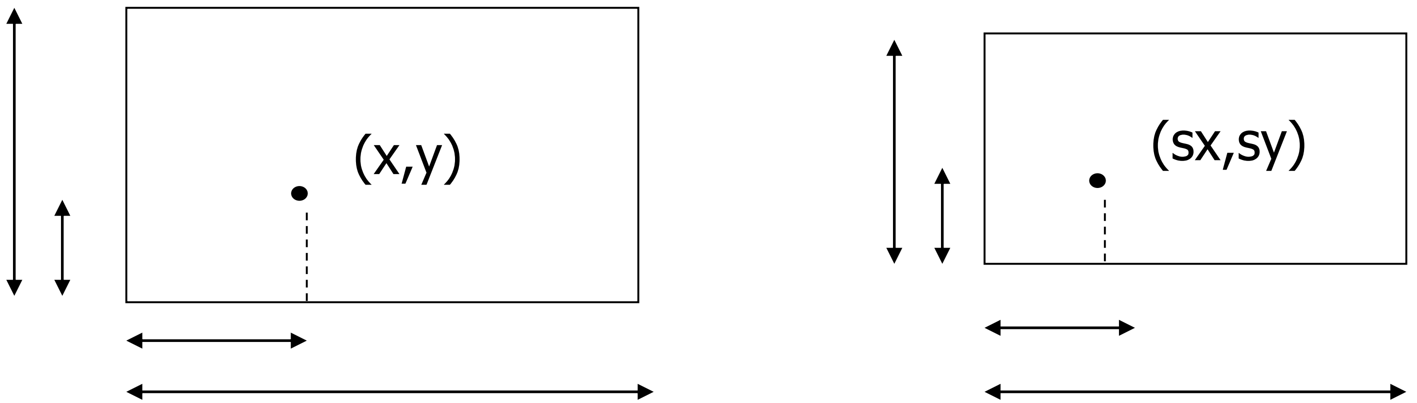
```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();

gluOrtho2D(Left, Right, Bottom, Top);
```

Window to viewport mapping

- Things that are given:
  - The world window (W_L, W_R, W_B, W_T)
  - The viewport (V_L, V_R, V_B, V_T)
  - A point (x,y) in the world coordinate system
- Calculate the corresponding point (sx, sy) in the screen coordinate system

Window to viewport mapping

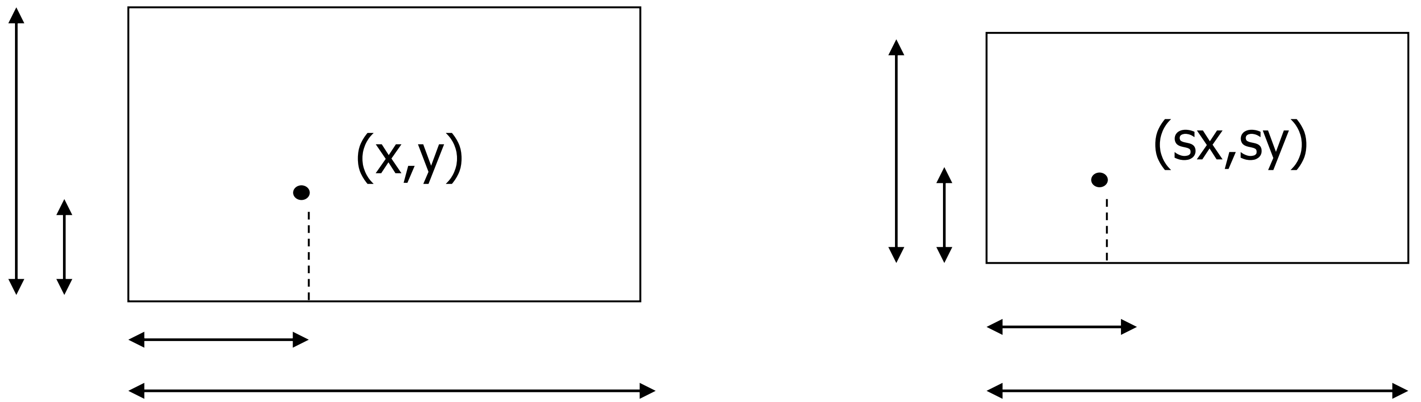
- Basic principle: the mapping should be proportional



$$(x - W_L) / (W_R - W_L) = (sx - V_L) / (V_R - V_L)$$

$$(y - W_B) / (W_T - W_B) = (sy - V_B) / (V_T - V_B)$$

Window to viewport mapping



$$(x - W_L) / (W_R - W_L) = (sx - V_L) / (V_R - V_L)$$

$$(y - W_B) / (W_T - W_B) = (sy - V_B) / (V_T - V_B)$$

$$sx = (x - W_L) * (V_R-V_L)/(W_R-W_L) + V_L$$

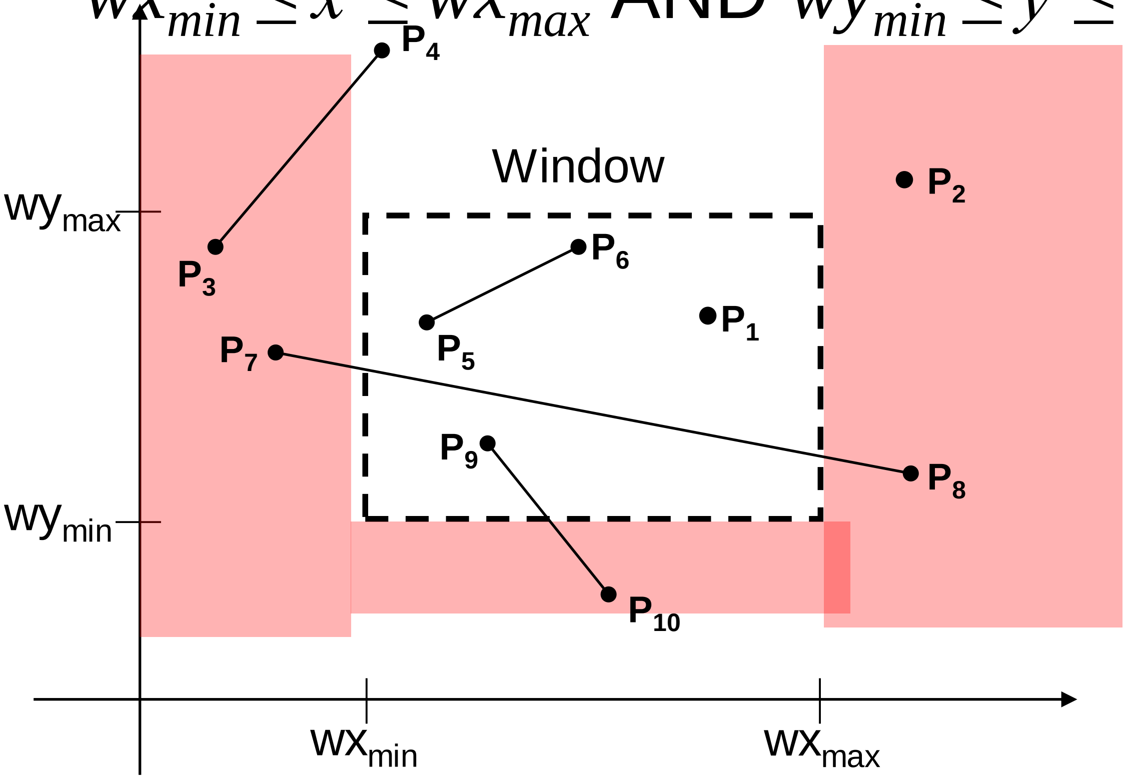
$$sy = (y - W_B) * (V_T-V_B)/(W_T-W_B) + V_B$$

Clipping

Point clipping is easy:

– For point (x,y) the point is not clipped if

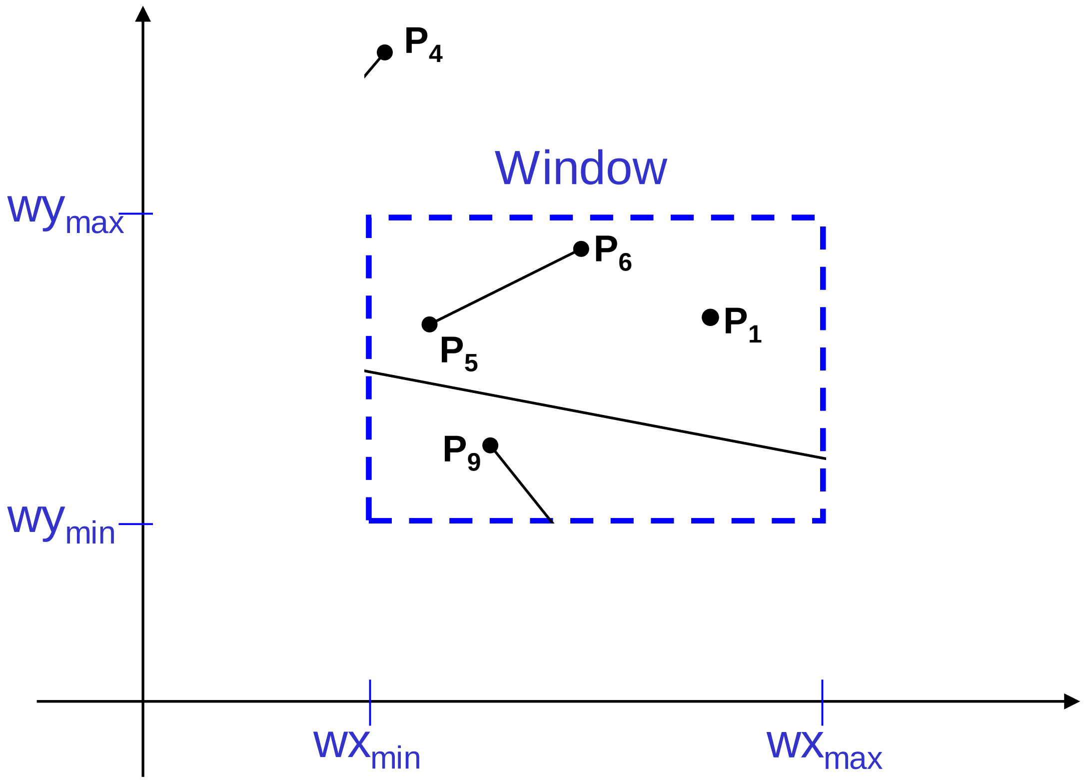
$$wx_{min} \leq x \leq wx_{max} \text{ AND } wy_{min} \leq y \leq wy_{max}$$



Before Clipping

Clipping

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

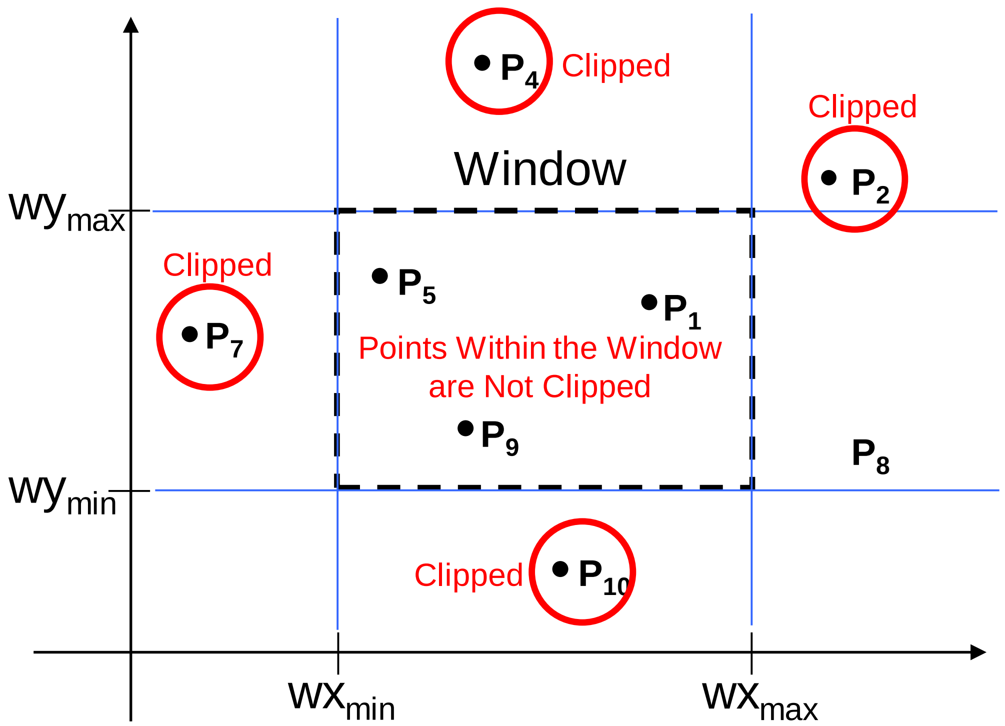


Point Clipping

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

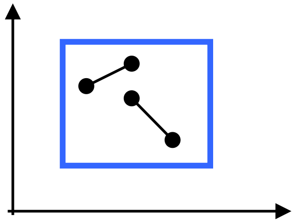
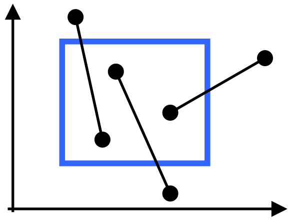
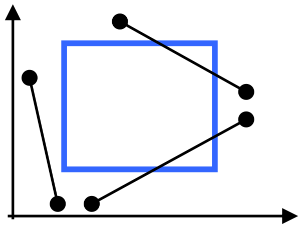
$$wx_{min} \leq x \leq wx_{max} \text{ AND } wy_{min} \leq y \leq wy_{max}$$

otherwise it is clipped



Line Clipping

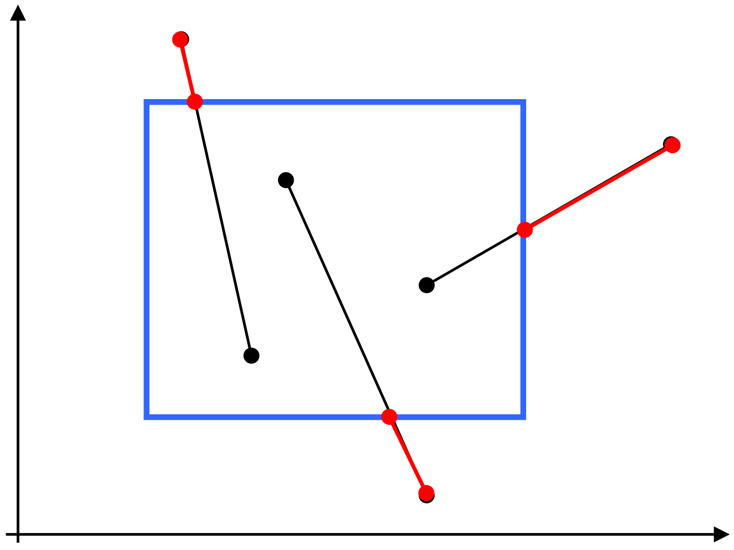
Harder - examine the end-points of each line to see if they are in the window or not

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	Don't know!	

Brute Force Line Clipping

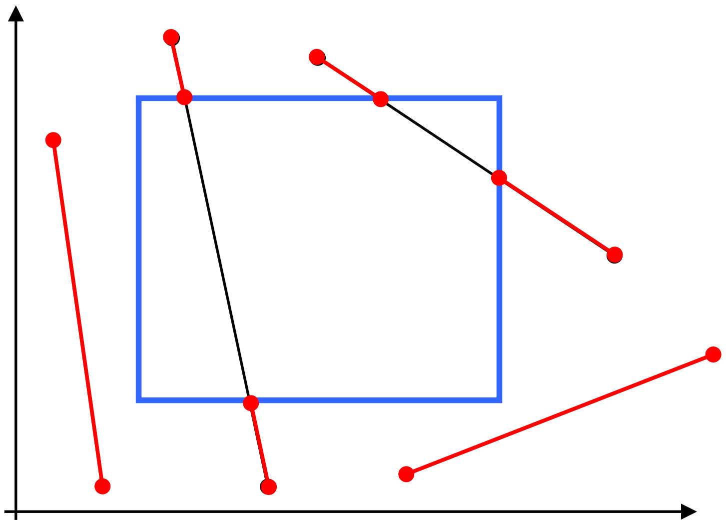
Brute force line clipping can be performed as follows:

- 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 much too slow

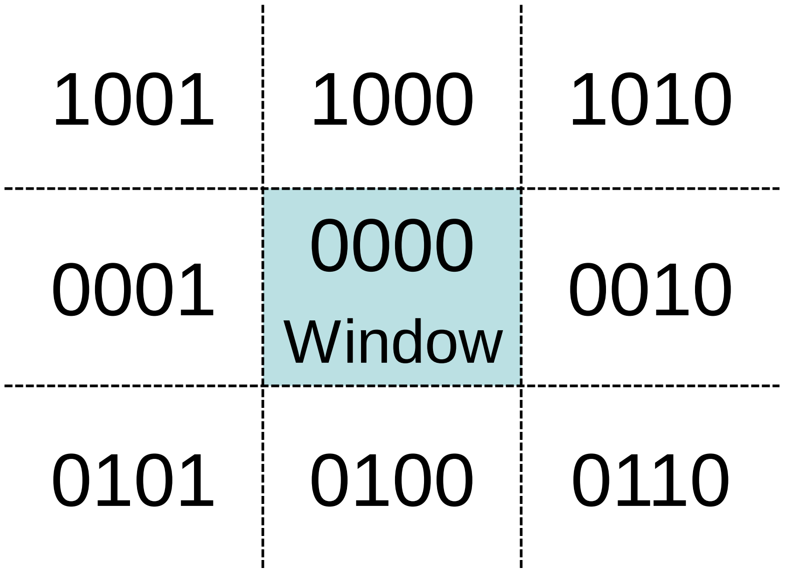
Cohen-Sutherland: World Division

World 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

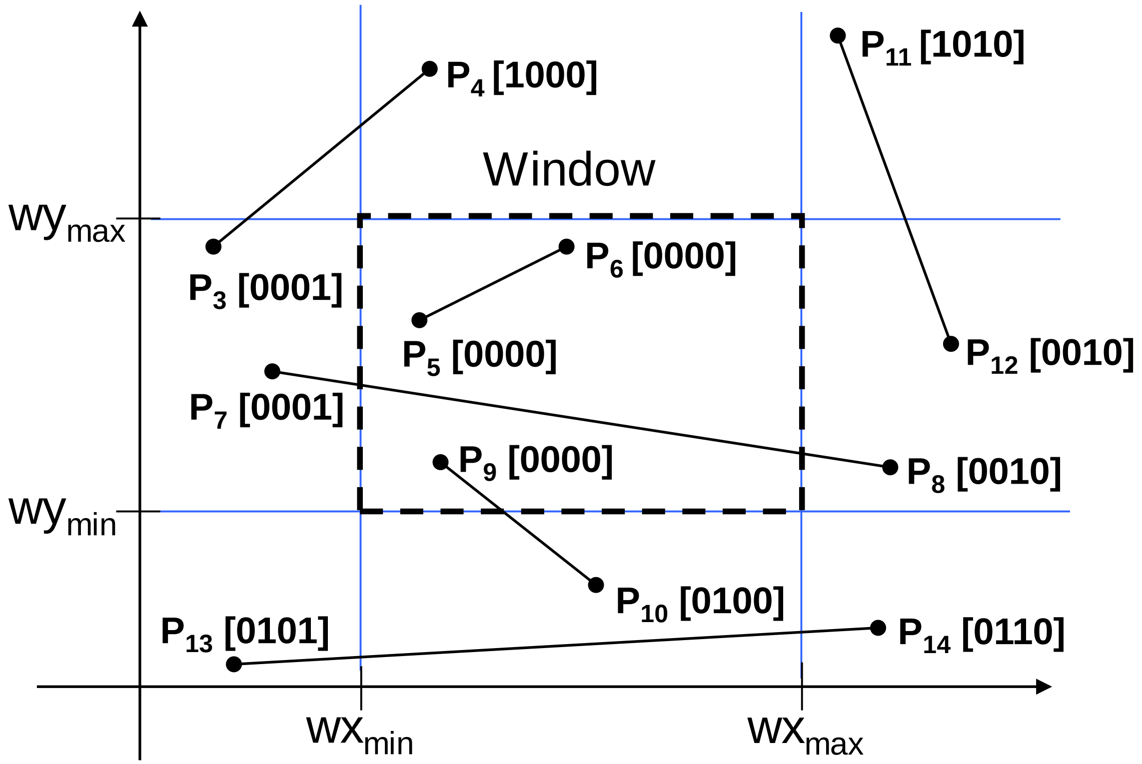
3	2	1	0
above	below	right	left

Region Code Legend



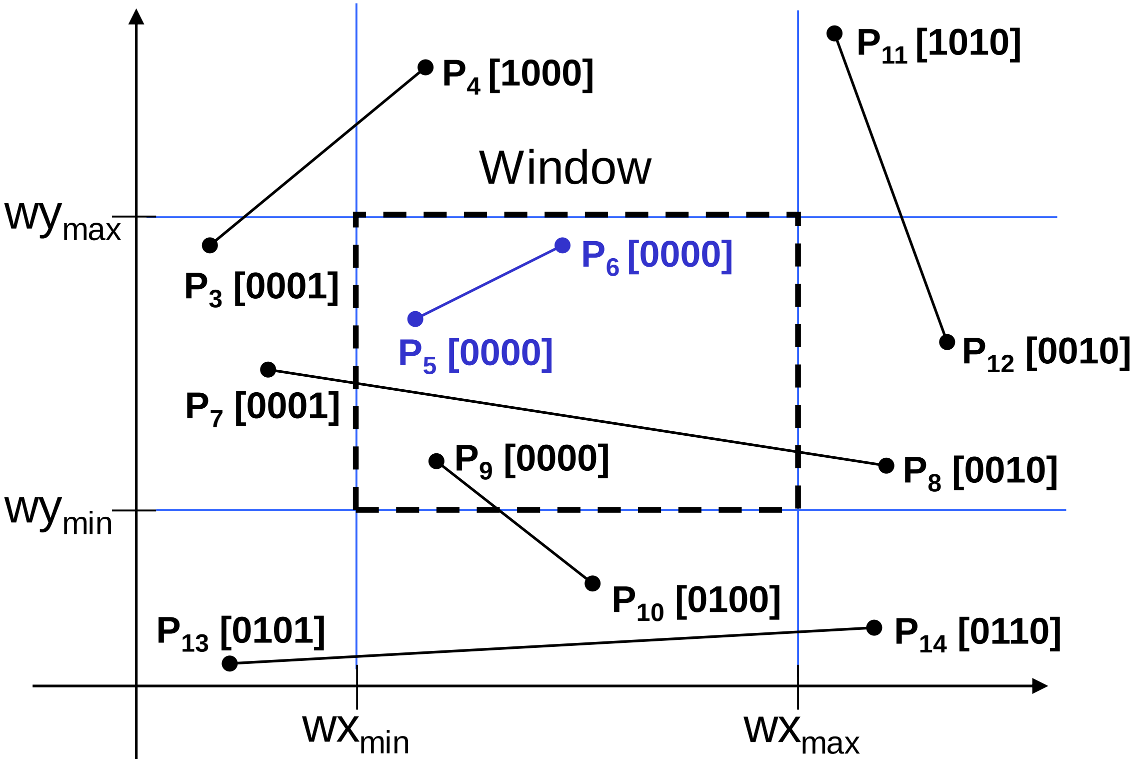
Cohen-Sutherland: Labelling

Every end-point is labelled with the appropriate region code



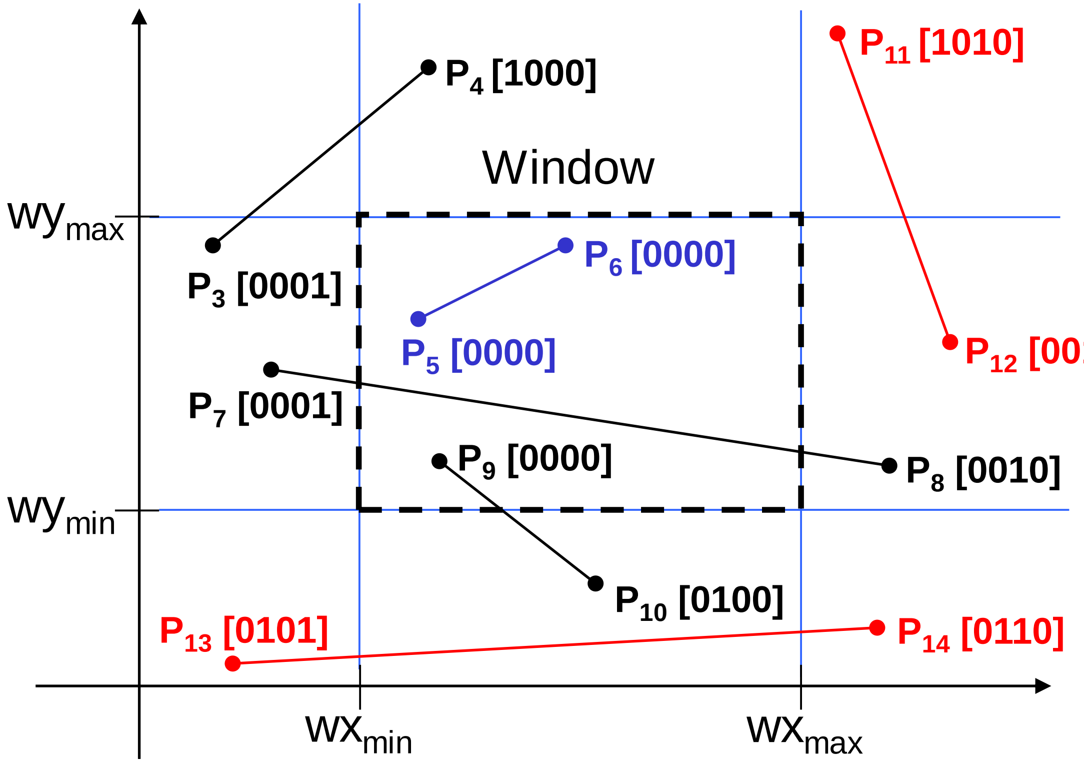
Cohen-Sutherland: Lines In The Window

Lines completely contained within the window boundaries have region code [0000] for both end-points so are not clipped



Cohen-Sutherland: Lines Outside The Window

- Any lines with a common set bit in the region codes of both end-points can be clipped
  - The AND operation can efficiently check this



Cohen-Sutherland: Other Lines

Lines that cannot be identified as completely inside or outside the window may or may not cross the window interior

These lines are processed as follows:

- Compare an end-point outside the window to a boundary (choose any order in which to consider boundaries e.g. left, right, bottom, top) and determine how much can be discarded
- If the remainder of the line is entirely inside or outside the window, retain it or clip it respectively

Cohen-Sutherland: Other Lines (cont…)

- Otherwise, compare the remainder of the line against the other window boundaries
- Continue until the line is either discarded or a segment inside the window is found

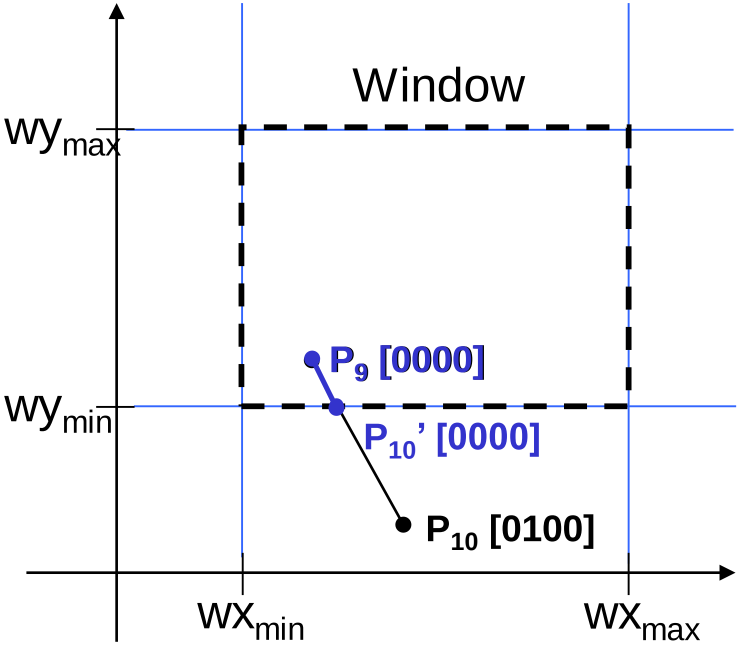
We can use the region codes to determine which window boundaries should be considered for intersection

- To check if a line crosses a particular boundary we compare the appropriate bits in the region codes of its end-points
- If one of these is a 1 and the other is a 0 then the line crosses the boundary

Cohen-Sutherland Examples

Consider the line P_9 to P_{10} below

- Start at P_{10}
- From the region codes of the two end-points we know the line doesn't cross the left or right boundary

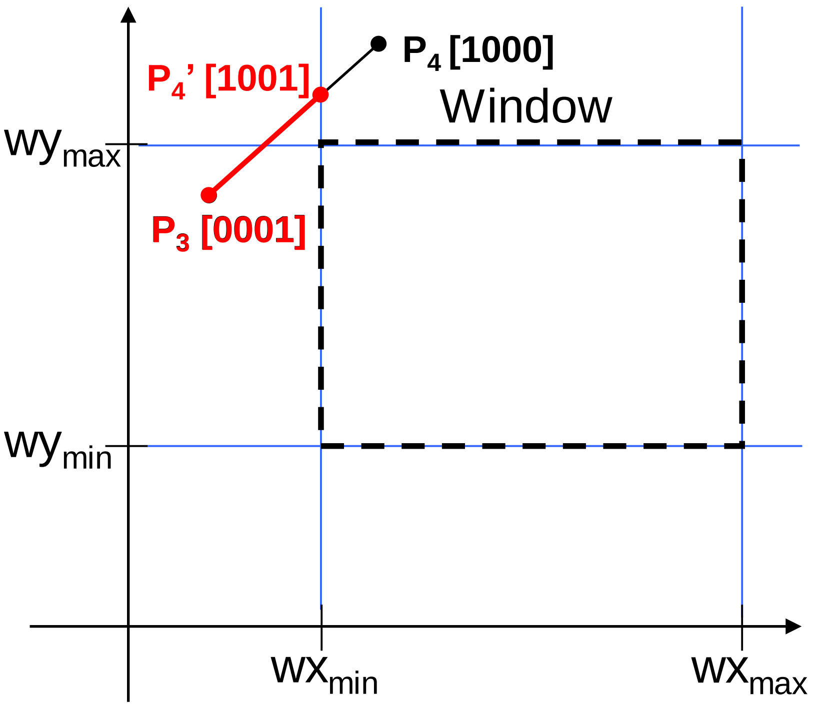


- Calculate the intersection of the line with the bottom boundary to generate point P_{10}'
- The line P_9 to P_{10}' is completely inside the window so is retained

Cohen-Sutherland Examples (cont…)

Consider the line P_3 to P_4 below

- Start at P_4
- From the region codes of the two end-points we know the line crosses the left boundary so calculate the intersection point to generate P_4'

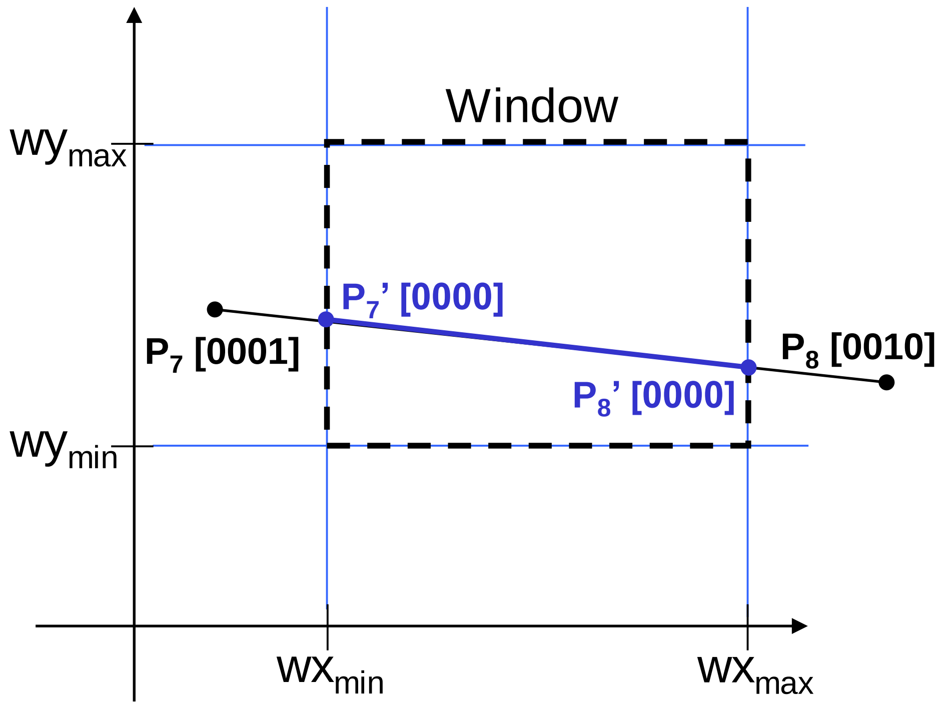


- The line P_3 to P_4' is completely outside the window so is clipped

Cohen-Sutherland Examples (cont…)

Consider the line P_7 to P_8 below

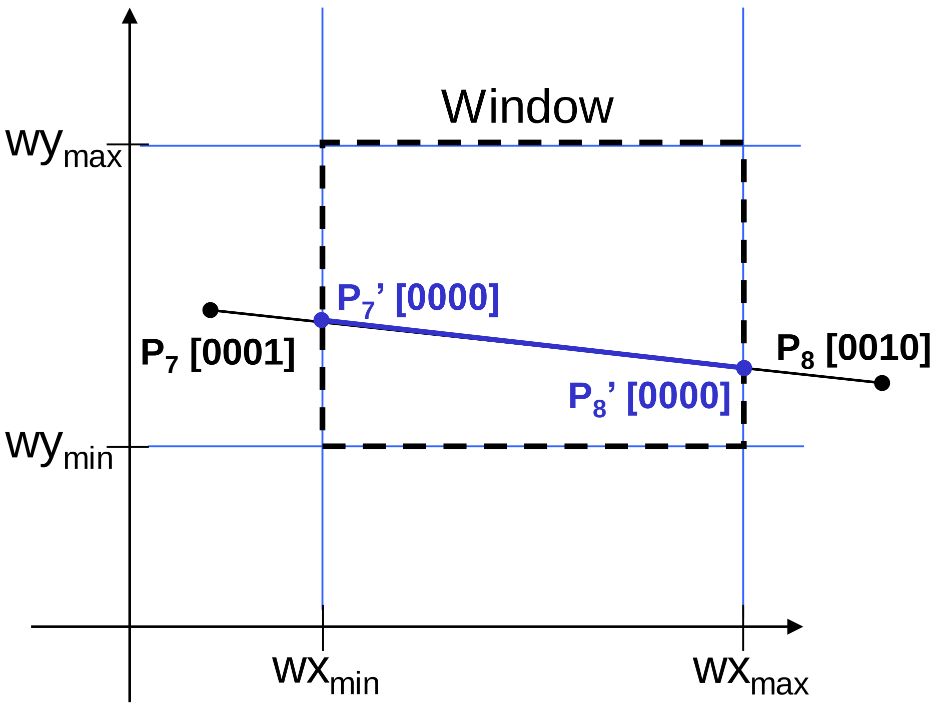
- Start at P_7
- From the two region codes of the two end-points we know the line crosses the left boundary so calculate the intersection point to generate P_7'



Cohen-Sutherland Examples (cont…)

Consider the line P_7' to P_8

- Start at P_8
- Calculate the intersection with the right boundary to generate P_8'
- P_7' to P_8' is inside the window so is retained



Calculating Line Intersections

Intersection points with the window boundaries are calculated using the line-equation parameters

- Consider a line with the end-points (x_1, y_1) and (x_2, y_2)
- The y-coordinate of an intersection with a vertical window boundary can be calculated using:

$$y = y_1 + m\,(x_{boundary} - x_1)$$

where $x_{boundary}$ can be set to either wx_{min} or wx_{max}

Calculating Line Intersections (cont…)

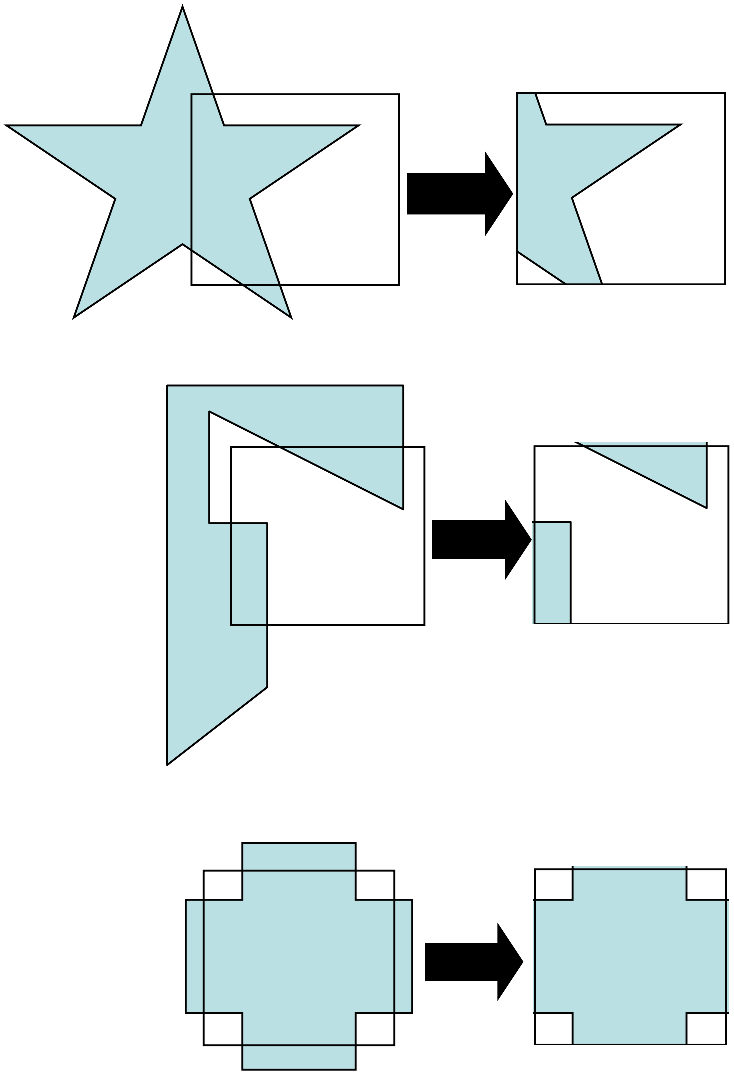
– The x-coordinate of an intersection with a horizontal window boundary can be calculated using:

$$x = x_1 + (y_{\text{boundary}} - y_1) / m$$

where $y_{boundary}$ can be set to either wy_{min} or wy_{max}

– m is the slope of the line in question and can be calculated as $m = (y_2 - y_1) / (x_2 - x_1)$

Area Clipping



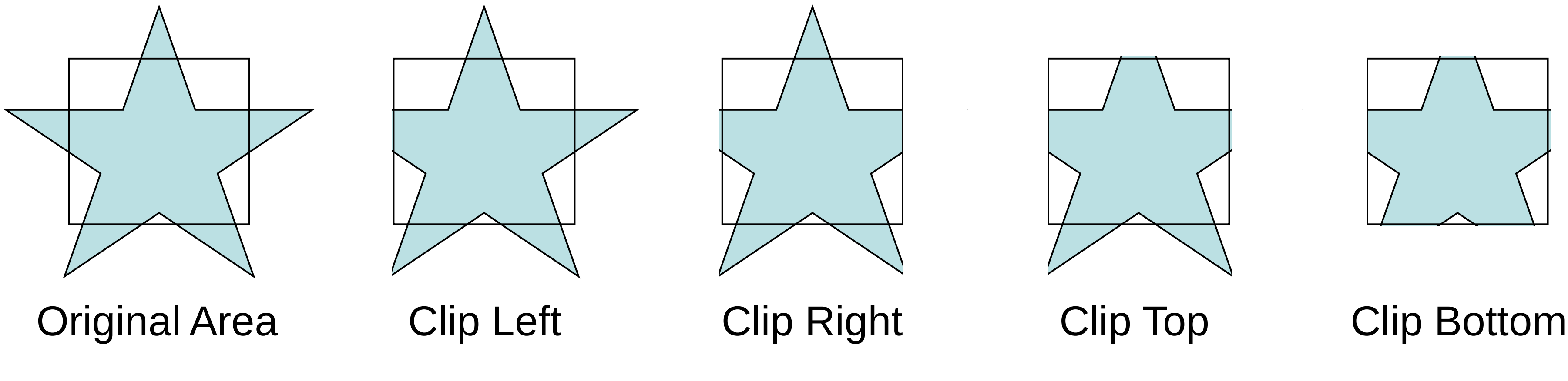
Similarly to lines, areas must be clipped to a window boundary

Consideration must be taken as to which portions of the area must be clipped

Sutherland-Hodgman Area Clipping Algorithm

A technique for clipping areas developed by Sutherland & Hodgman

Put simply the polygon is clipped by comparing it against each boundary in turn



Sutherland-Hodgman Area Clipping Algorithm (cont…)

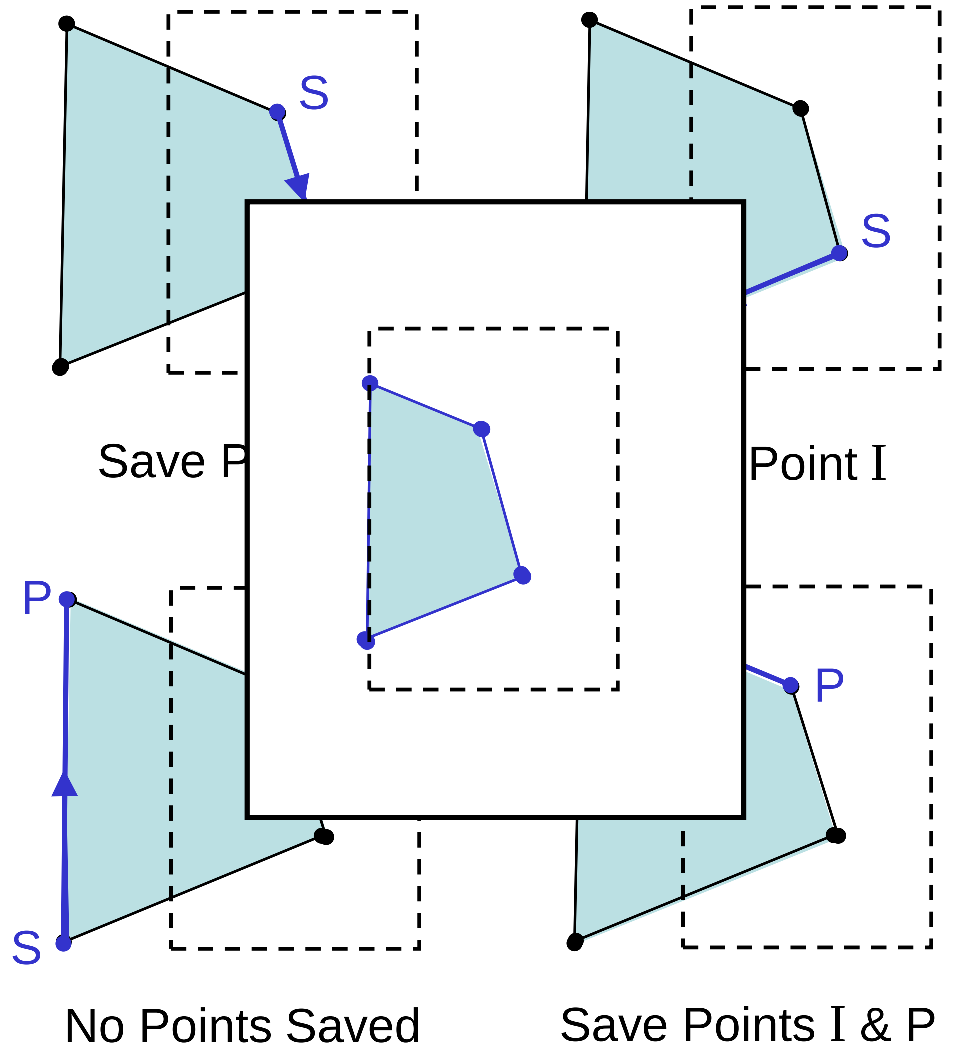
To clip an area against an individual boundary:

- Consider each vertex in turn against the boundary
- Vertices inside the boundary are saved for clipping against the next boundary
- Vertices outside the boundary are clipped
- If we proceed from a point inside the boundary to one outside, the intersection of the line with the boundary is saved
- If we cross from the outside to the inside intersection point and the vertex are saved

Sutherland-Hodgman Example

Each example shows the point being processed (P) and the previous point (S)

Saved points define area clipped to the boundary in question



Summary

Objects within a scene must be clipped to display the scene in a window

Because there are can be so many objects clipping must be extremely efficient

The Cohen-Sutherland algorithm can be used for line clipping

The Sutherland-Hodgman algorithm can be used for area clipping