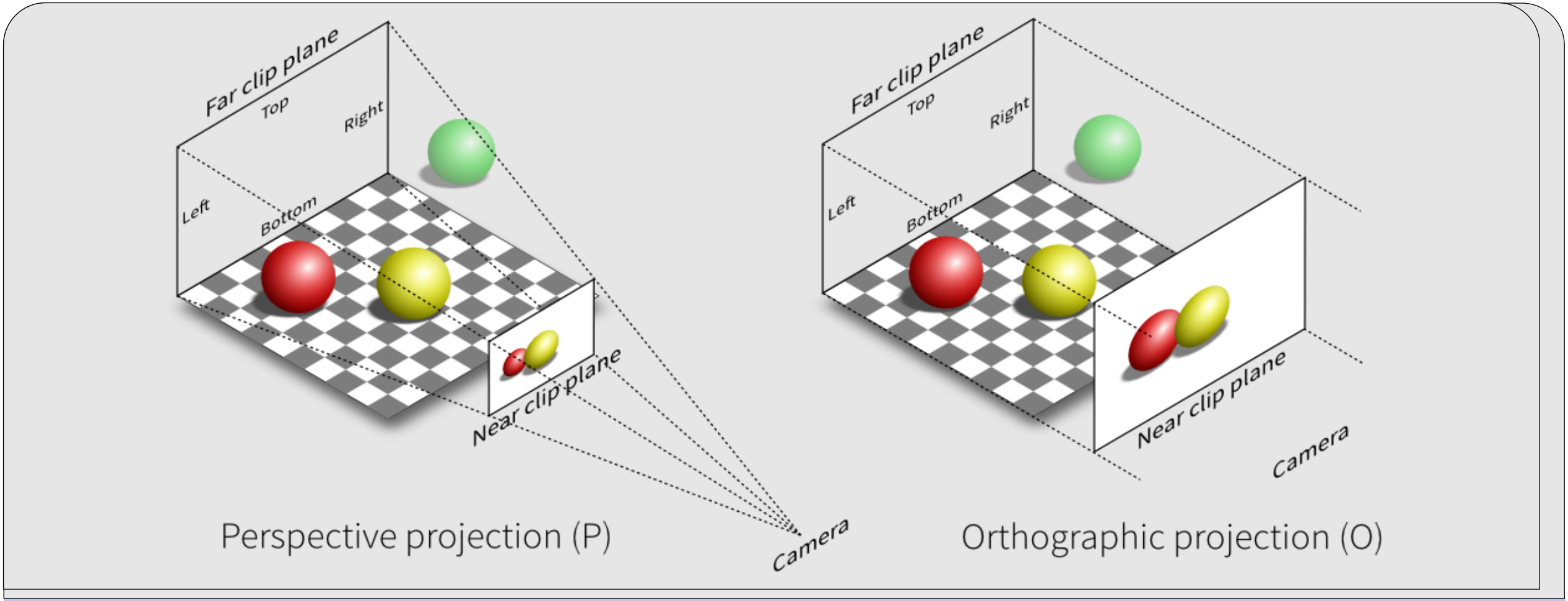


Computer Graphics

Clipping

Andreas Aristidou
andarist@ucy.ac.cy
http://www.andreasaristidou.com

Midterm Exam

MARCH 2022

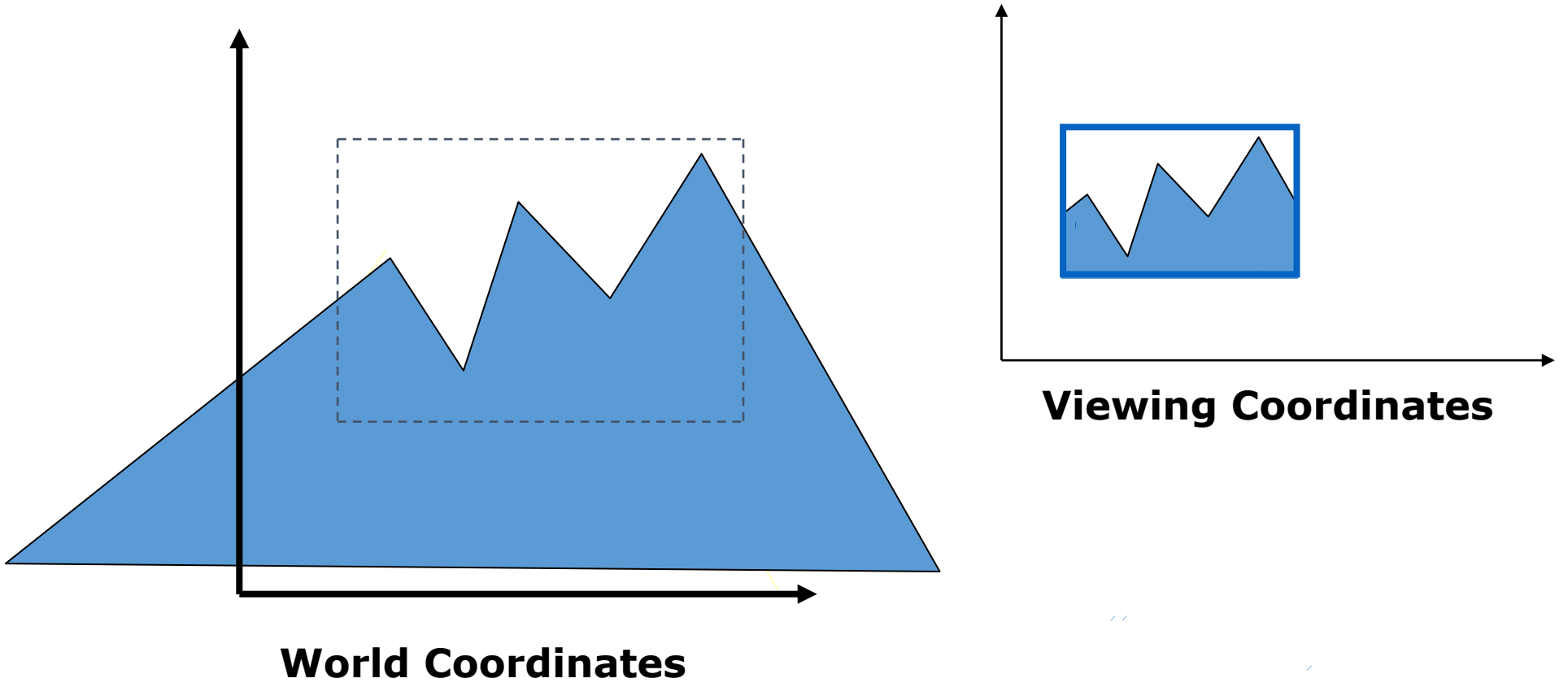
Sun	Mon	Tue	Wed	Thu	Fri	Sat
27	28	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	1	2

Two-dimensional view: *Summary*

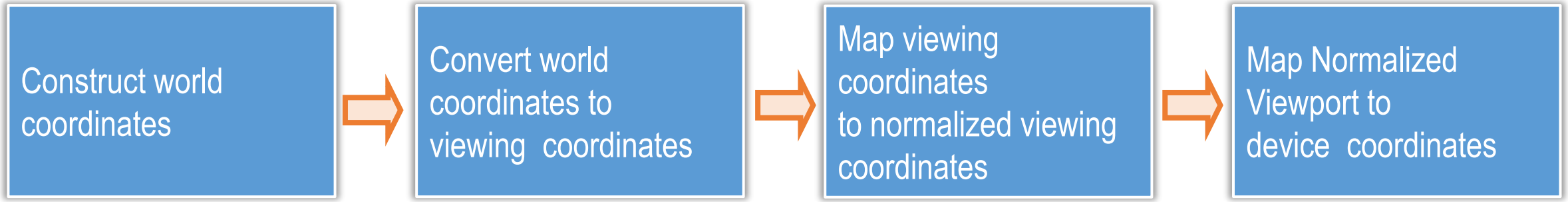
- An area that is selected for viewing is called a window.
  - *The window indicates what it is that we will see.*
- An area on a device to which the viewing window is assigned (e.g. monitor) is called **viewport**.
  - *The **viewport** indicates where the view will take place.*

Two-dimensional view: *Summary*

- Mapping an area taken from the global coordinate system on a device is called a **projection transformation**
-

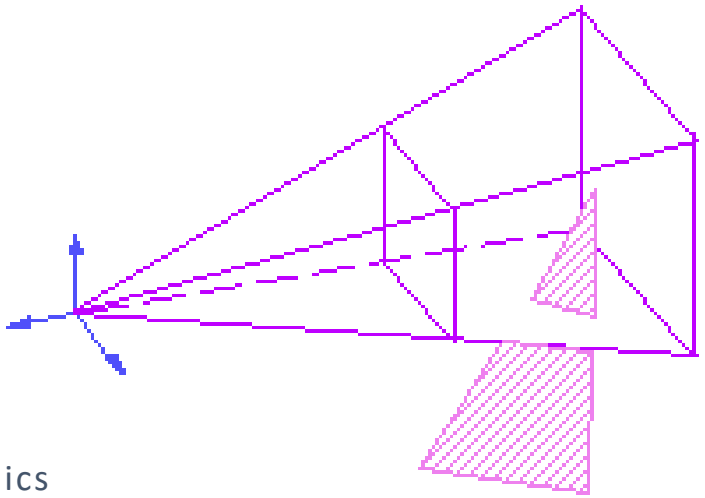


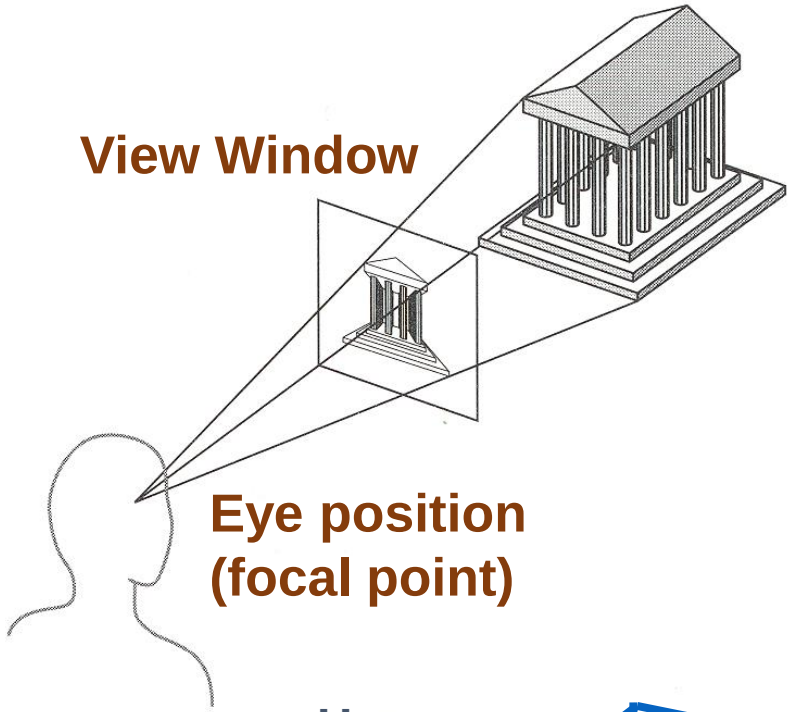
2D view transformation: *Summary*

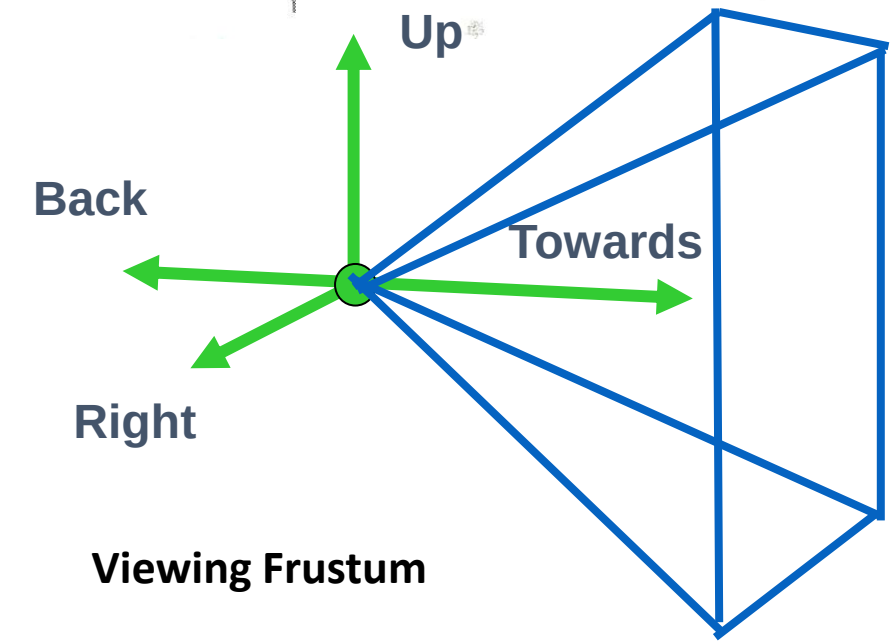


Clipping

- It is the process of finding the exact part of the polygon that is in the field of view (view volume)
- To maintain consistency, clipping a polygon should result in a polygon, not a sequence of partially detached lines
- We will first look at a 2D solution and then extend it to 3D
-





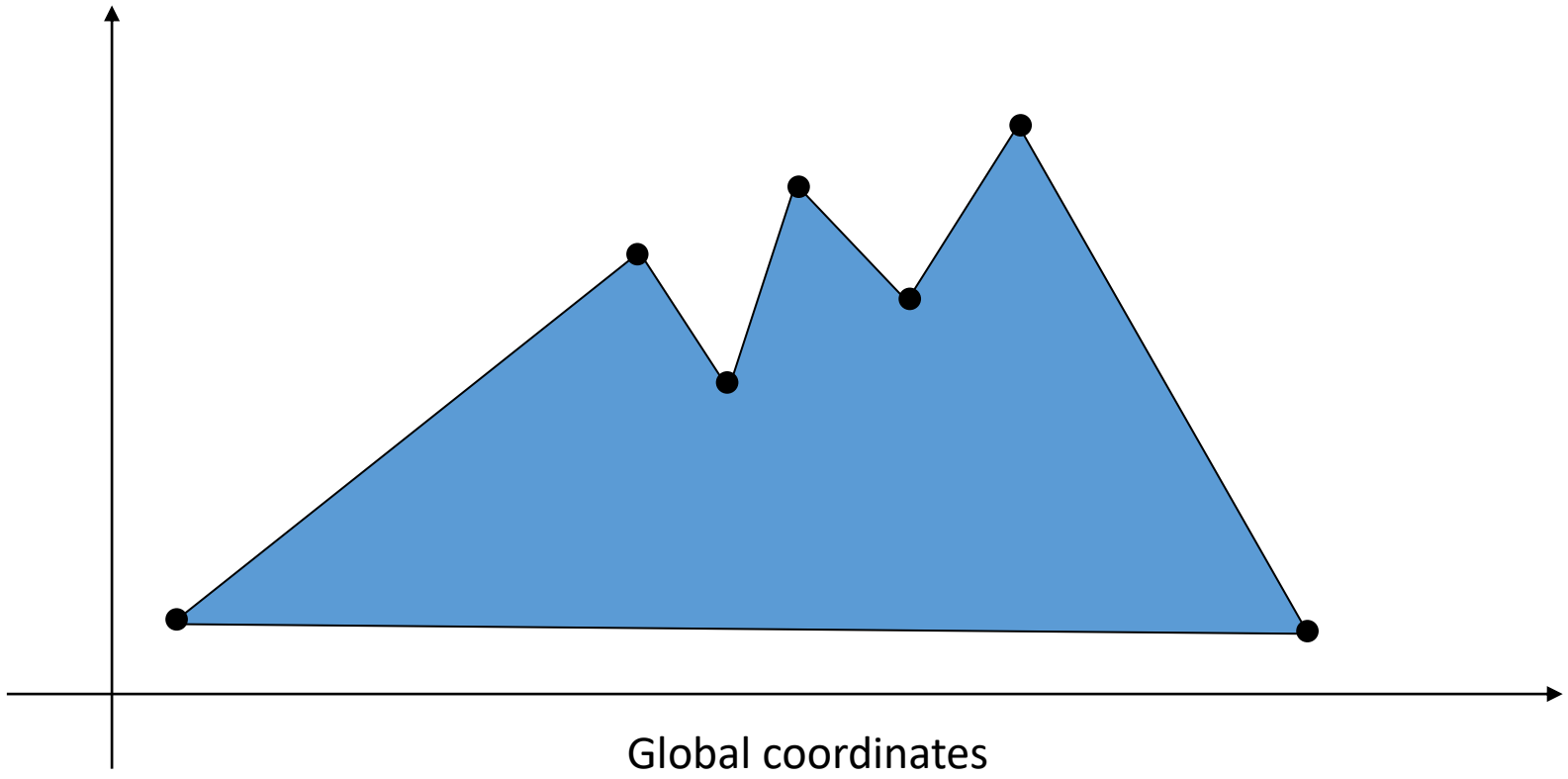


Why clipping?

- Clipping removes objects that will not be visible from the scene
- The point of this is to remove computational effort

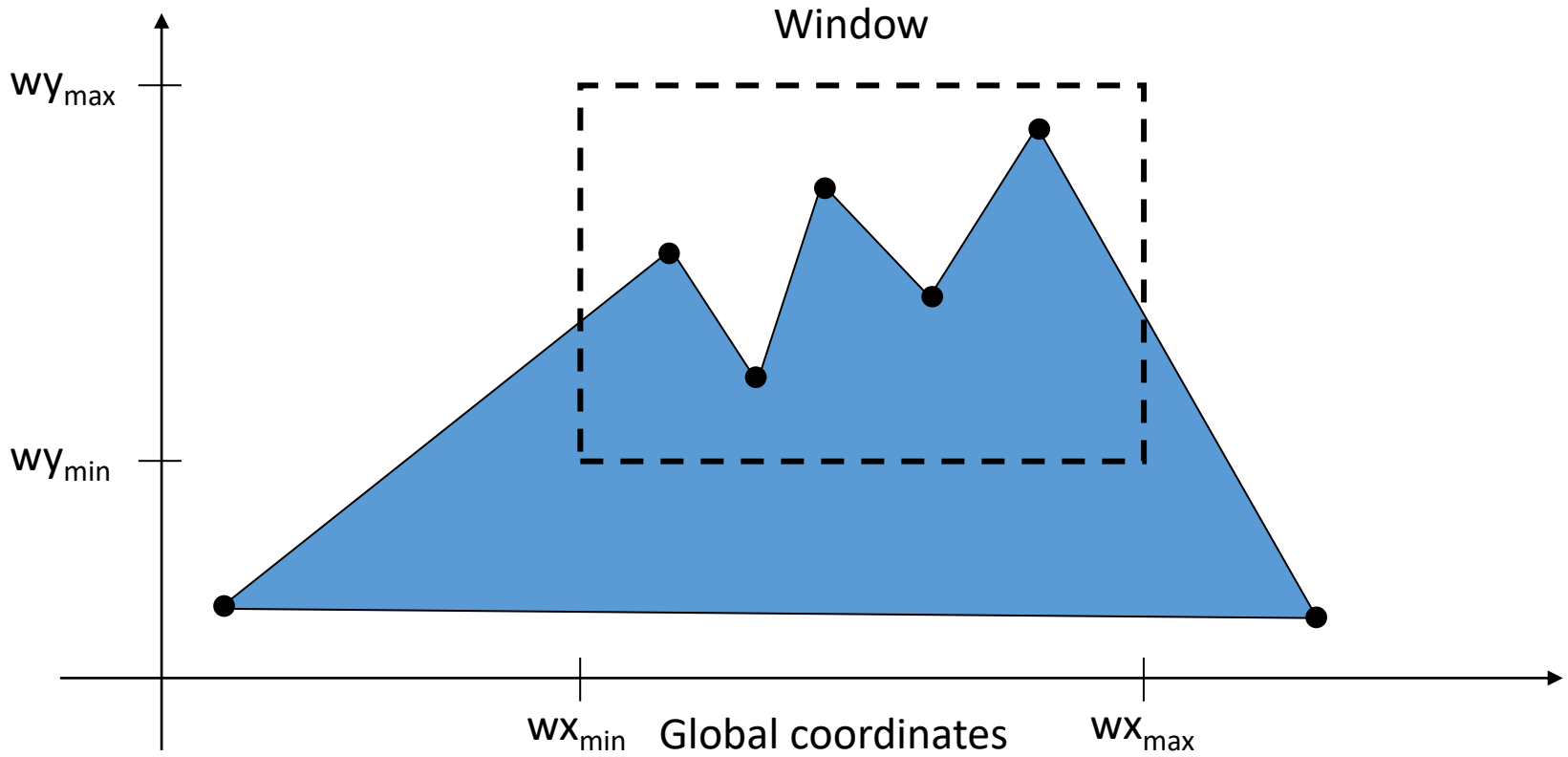
Windowing I

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



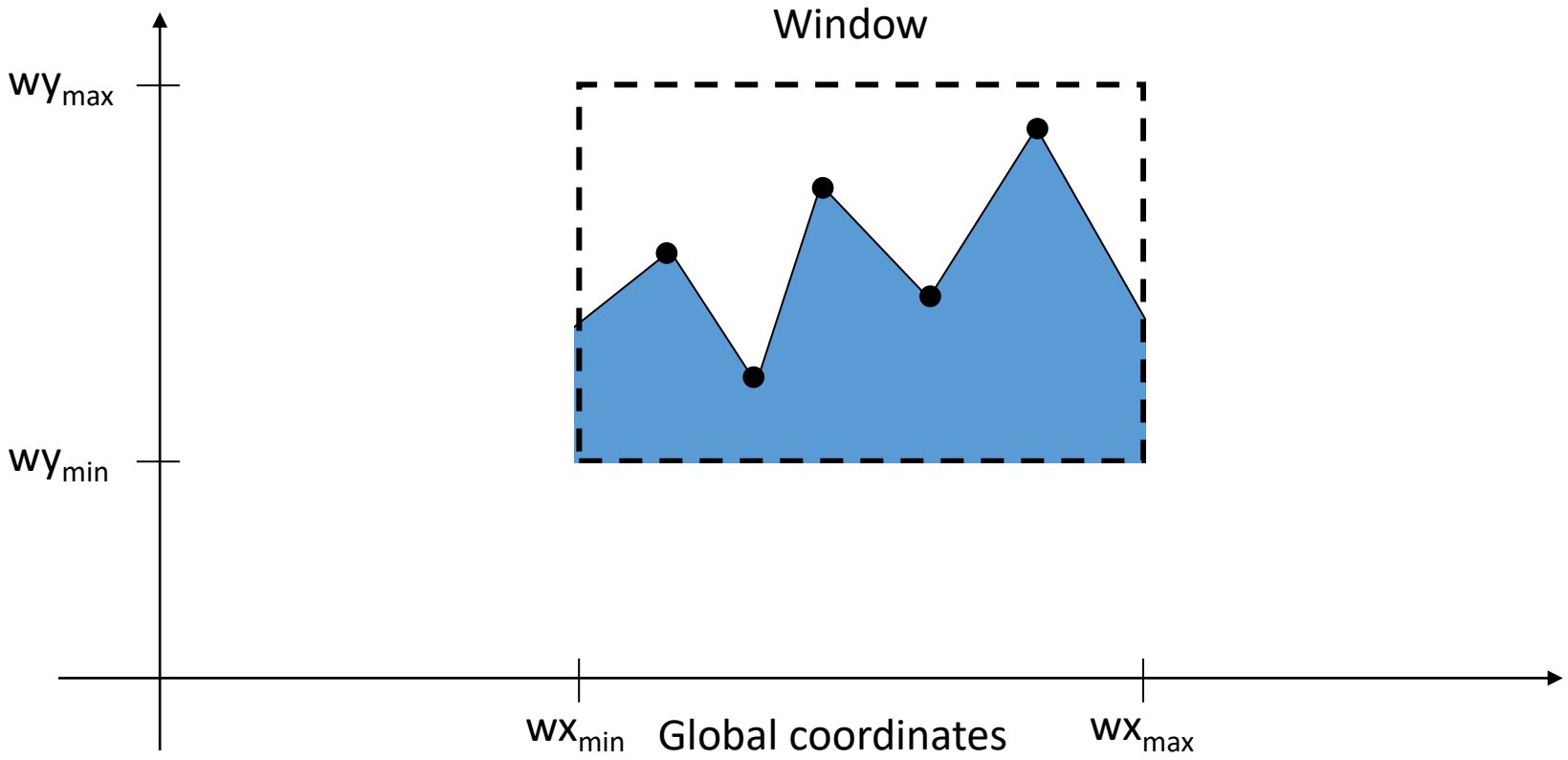
Windowing II

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



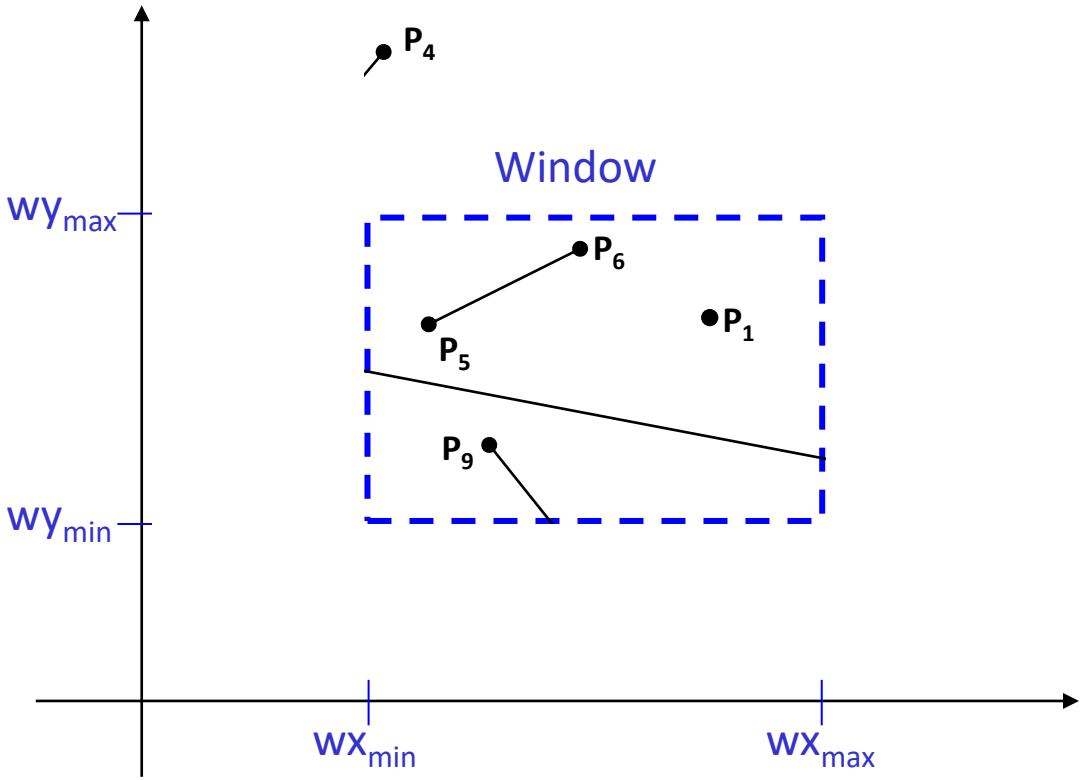
Windowing III

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



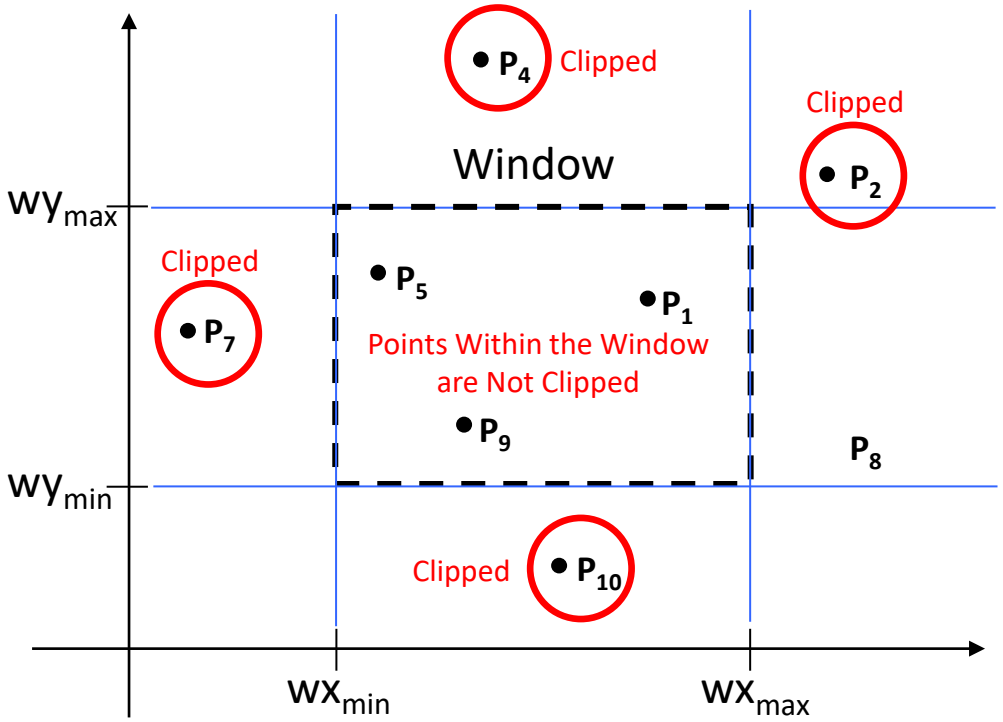
Clipping

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



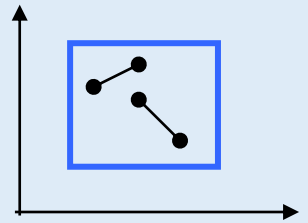
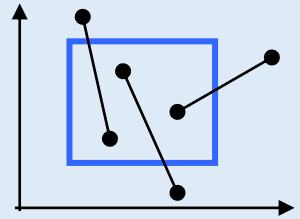
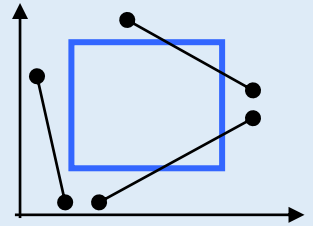
Point Clipping

- Easy - a point (x,y) is not clipped if:
- $wx_{min} \leq x \leq wx_{max}$ KAI $wy_{min} \leq y \leq wy_{max}$
- otherwise it is clipped



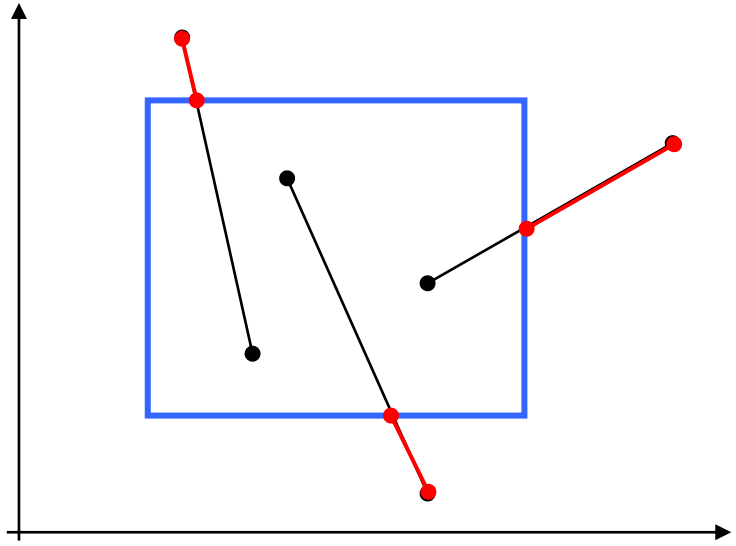
Line Clipping

- 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!	

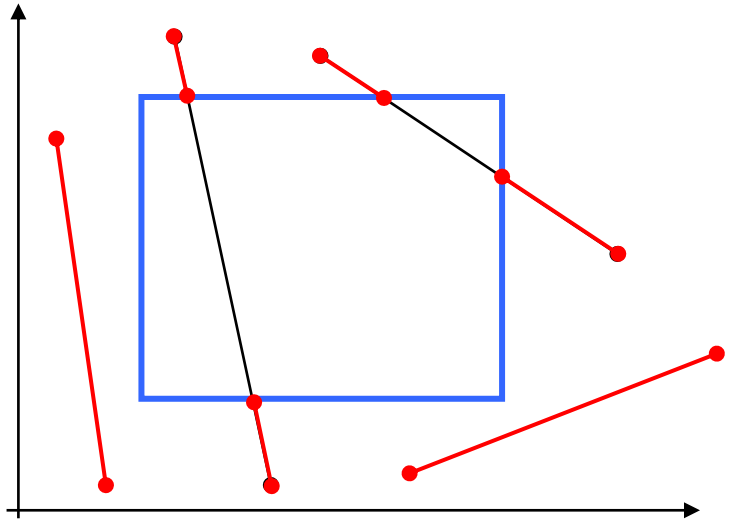
Brute Force Line Clipping

- 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

- Brute force line clipping can be performed as follows:
  - 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…

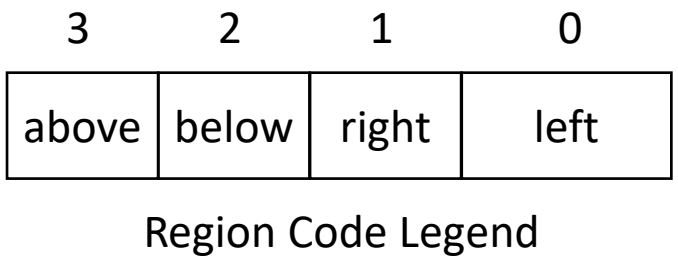
Cohen-Sutherland Clipping Algorithm

- 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

Dr. Ivan E. Sutherland co-developed the Cohen-Sutherland clipping algorithm. Sutherland is a graphics giant and includes amongst his achievements the invention of the head mounted display.

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

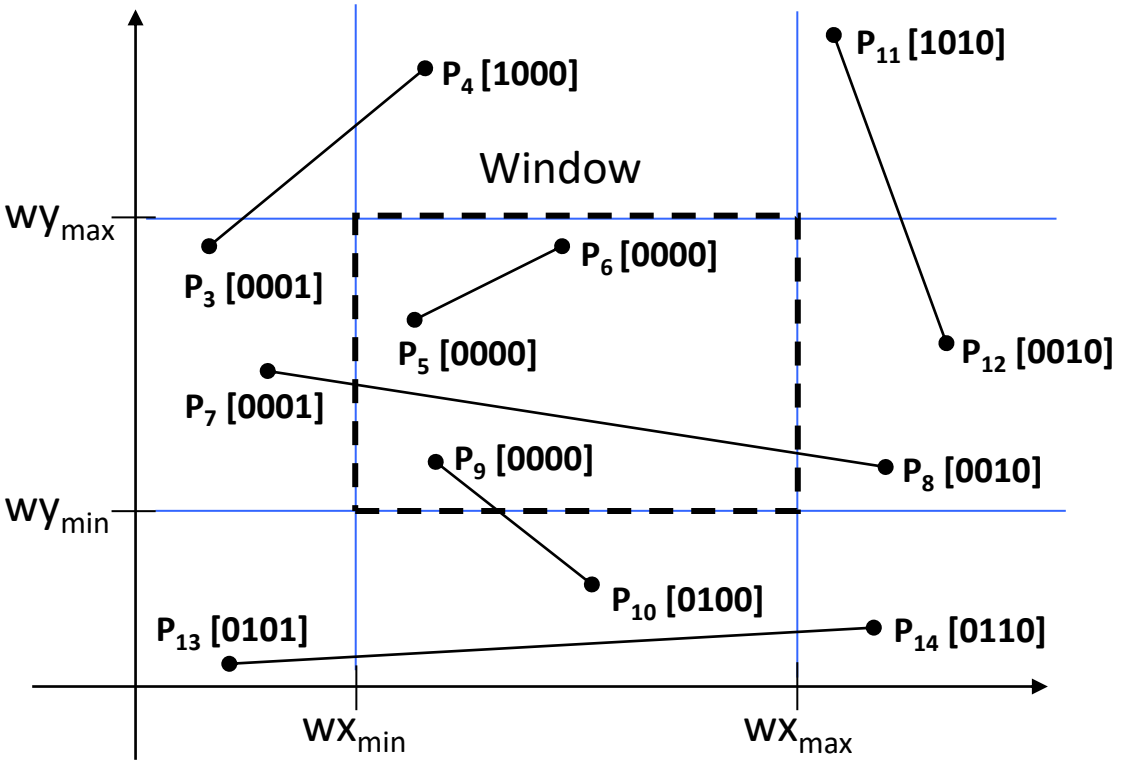


Region Code Legend

1001	1000	1010
0001	0000 Window	0010
0101	0100	0110

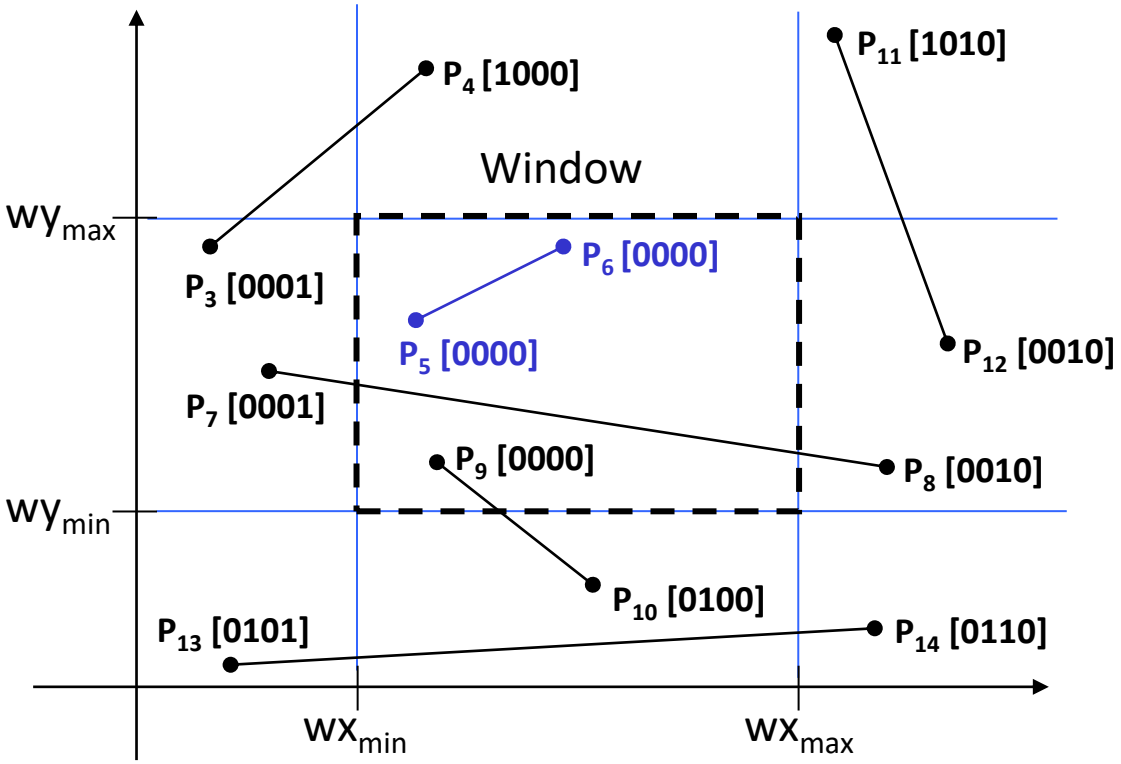
Cohen-Sutherland: Labelling

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



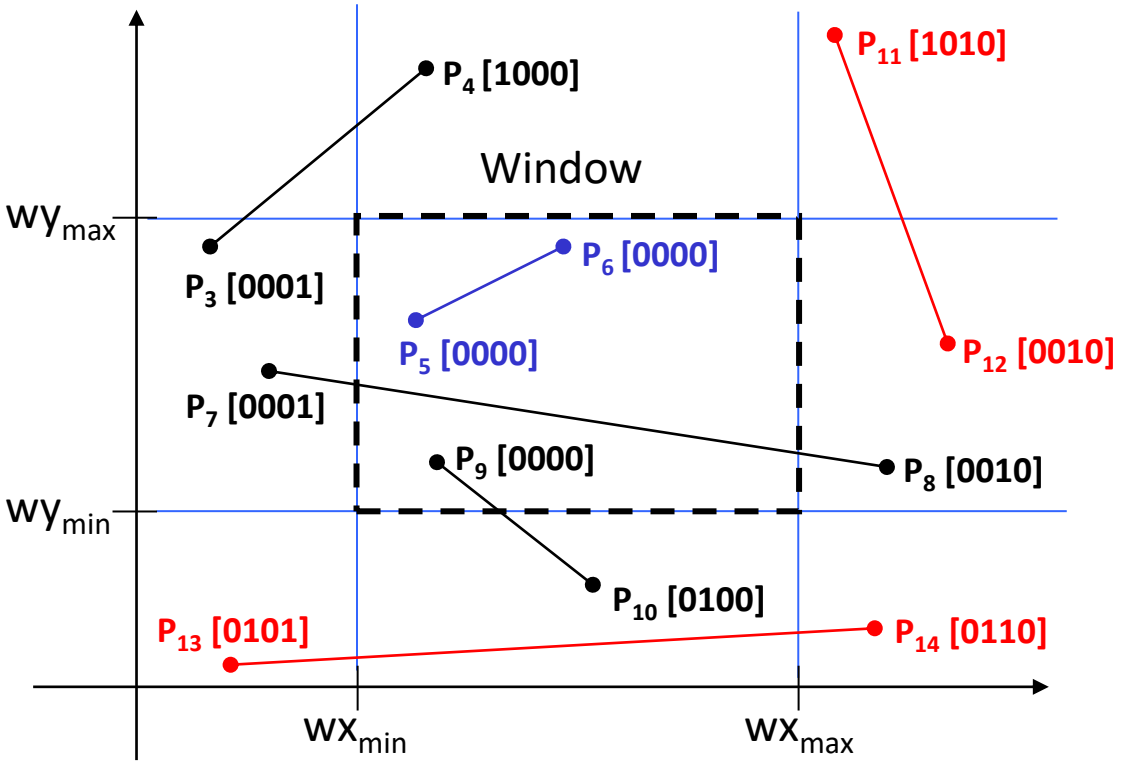
Cohen-Sutherland: Lines in The Window

- 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

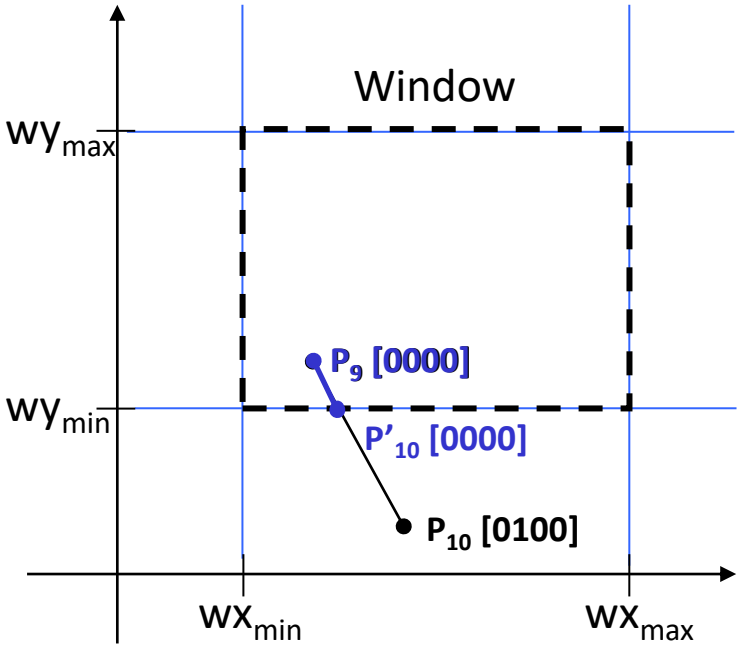
- Lines that cannot be identified as completely inside or outside the window may or may not cross the window interior
- These lines are processed as follows:
  - Compare an end-point outside the window to a boundary (choose any order in which to consider boundaries e.g. left, right, bottom, top) and 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
  - 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

Cohen-Sutherland: Other Lines

- 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

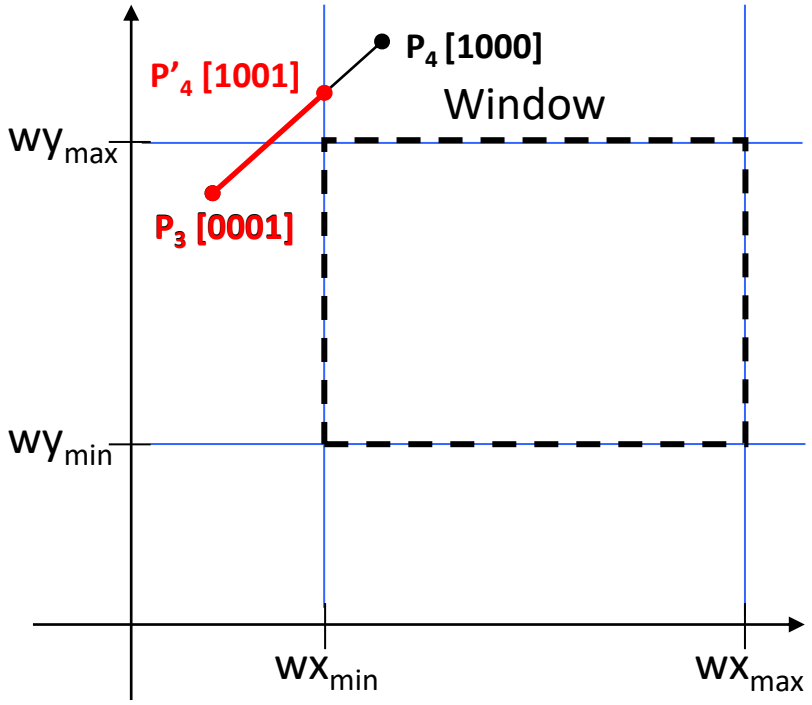
Cohen-Sutherland: Examples

- 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



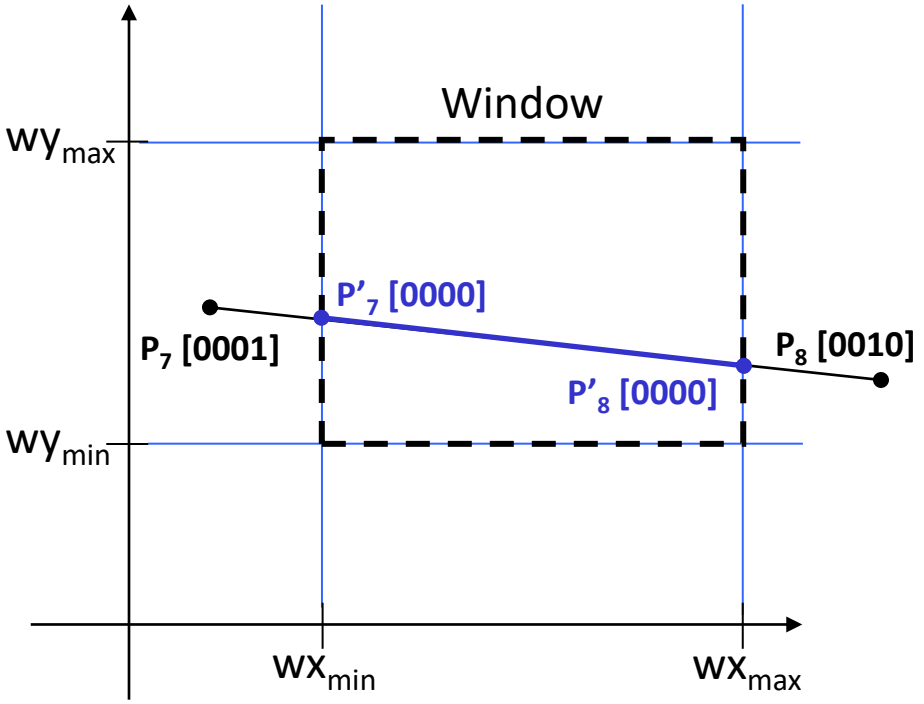
Cohen-Sutherland: Examples

- 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



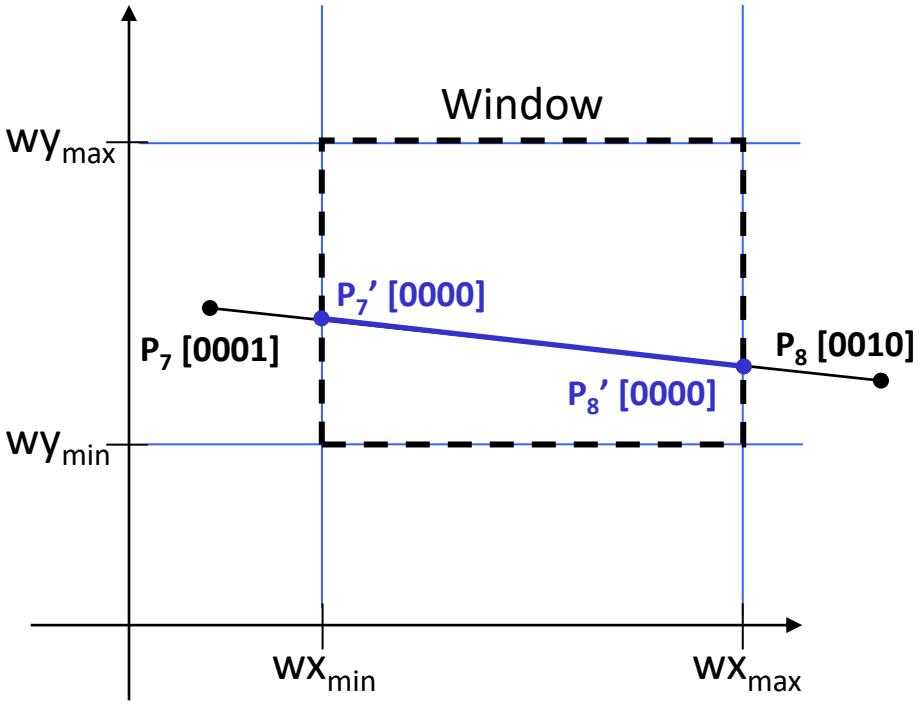
Cohen-Sutherland: Examples

- 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

- 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

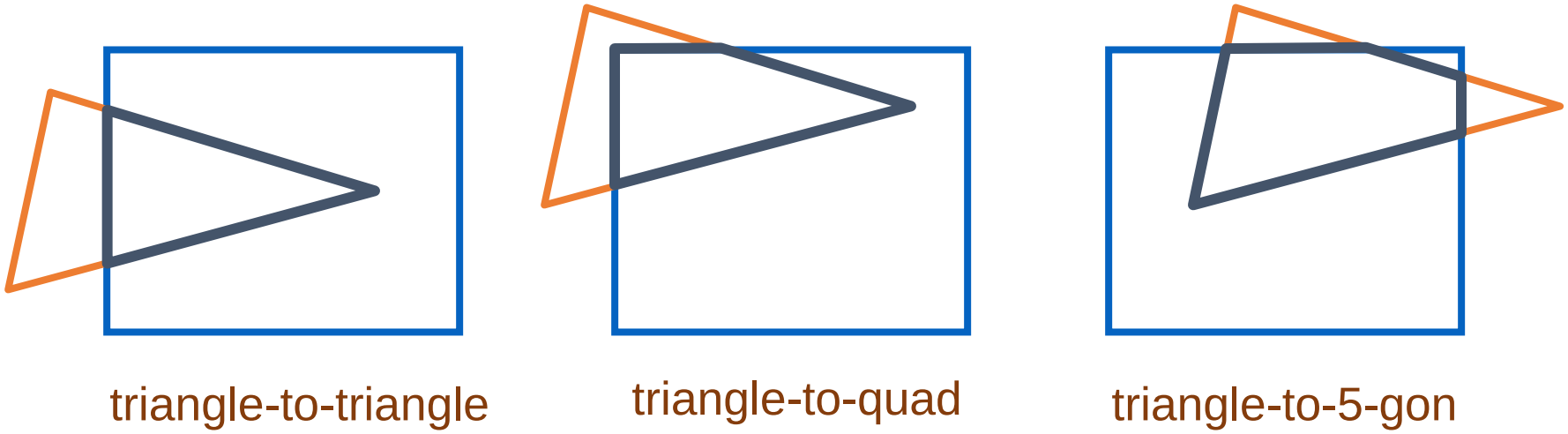
- The x-coordinate of an intersection with a horizontal window boundary can be calculated using:

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

  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 $m = (y_2 - y_1) / (x_2 - x_1)$

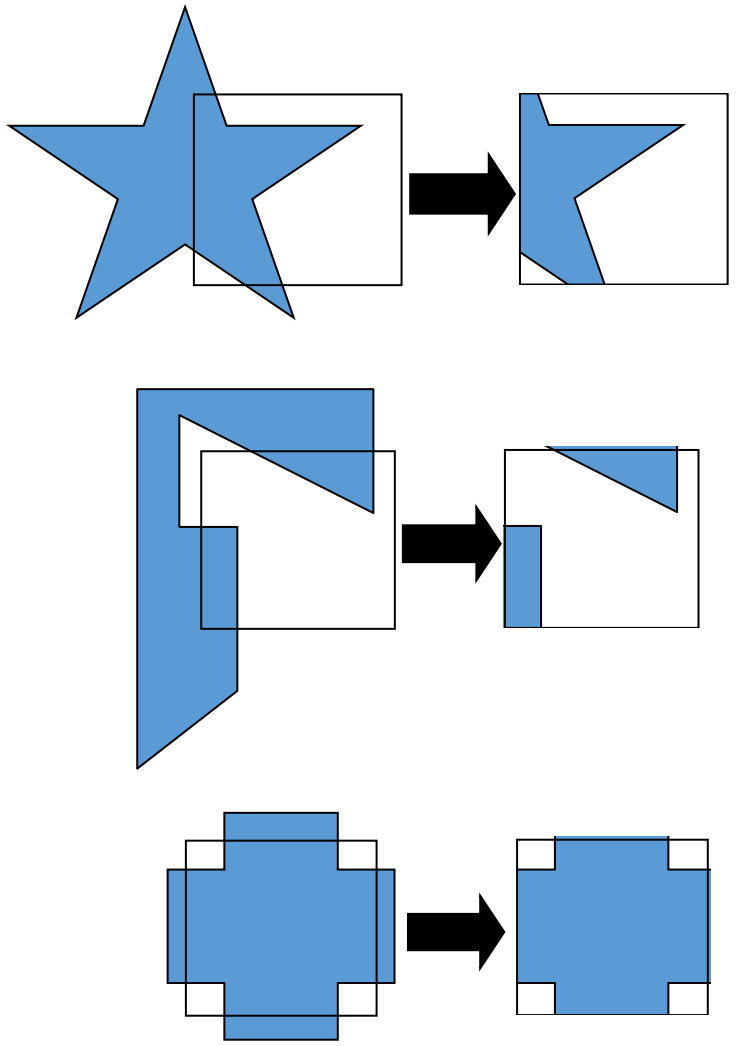
Why is clipping difficult?

- What can happen to a triangle after clipping?
- Possible results



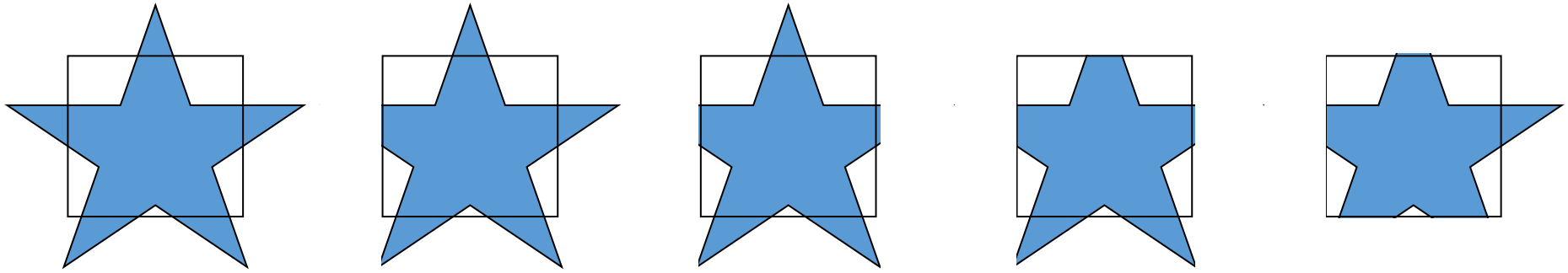
Area Clipping

- 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

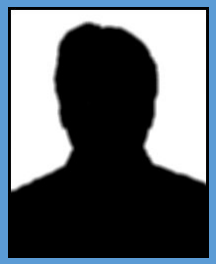


Sutherland-Hodgman: Area Clipping Algorithm

- A technique for clipping areas developed by Sutherland & Hodgman
- Put simply the polygon is clipped by comparing it against each boundary in turn



Original Area | Clip Left | Clip Right | Clip Top | Clip Bottom



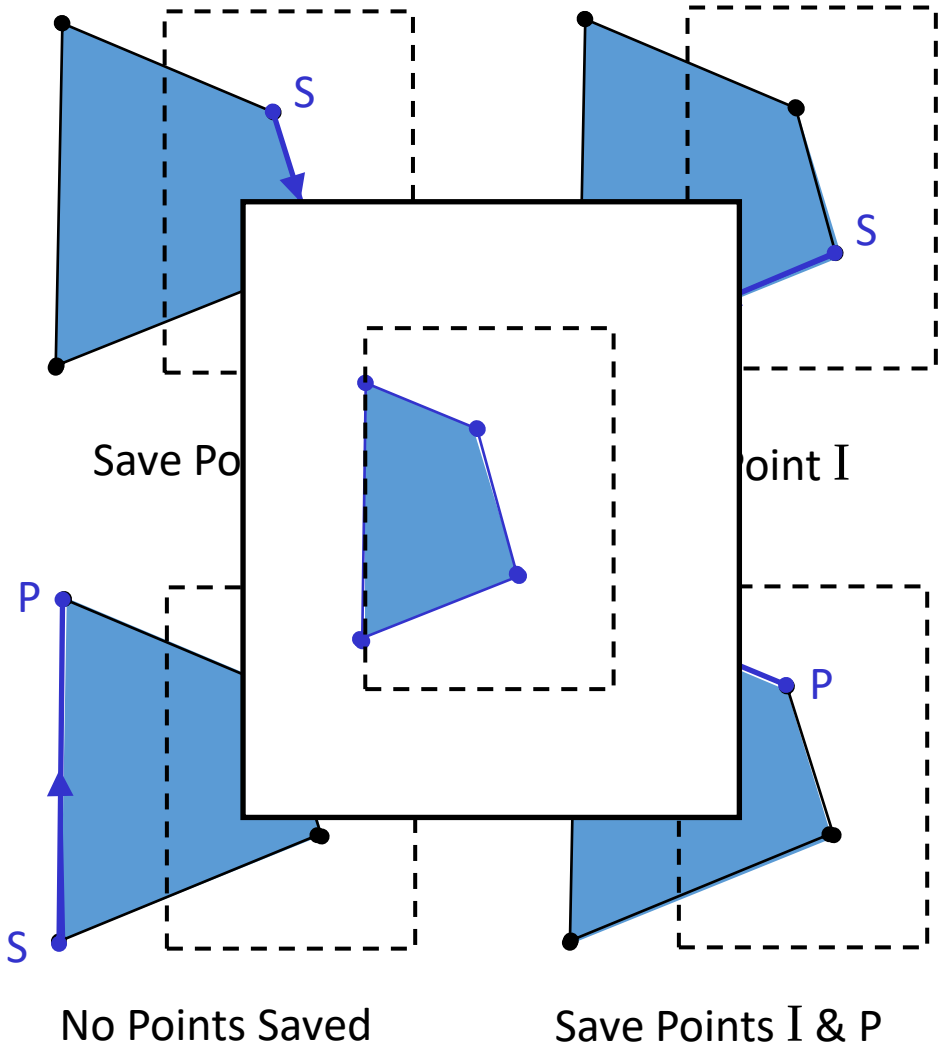
Sutherland turns up again. This time with Gary Hodgman with whom he worked at the first ever graphics company Evans & Sutherland

Sutherland-Hodgman: Area Clipping Algorithm

- 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

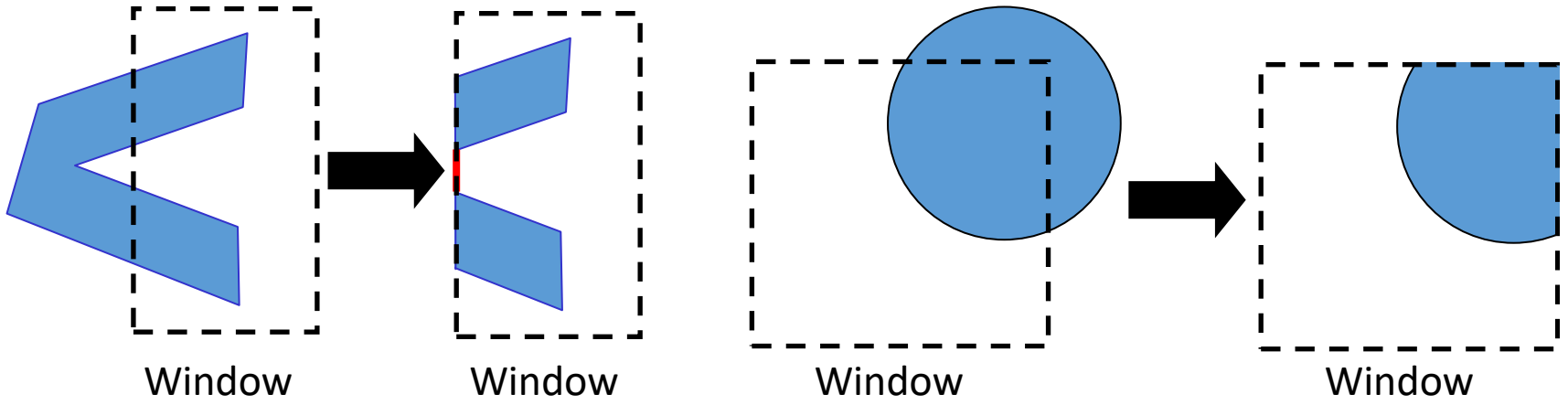
Sutherland-Hodgman: Παράδειγμα

- Each example shows the point being processed (P) and the previous point (S)
- Saved points define area clipped to the boundary in question



Other Area Clipping Concerns

- Clipping concave areas can be a little more tricky as often superfluous lines must be removed
  - When we have non-convex polygons then the above algorithm can produce polygons with coincidental edges
    - This is ok for rendering but maybe not for other applications (e.g. shadows)
    - The Weiler-Atherton algorithm produces separate polygons for each visible piece of polygon



- Clipping curves requires more work
  - For circles we must find the two intersection points on the window boundary

Clipping polygons in 3D

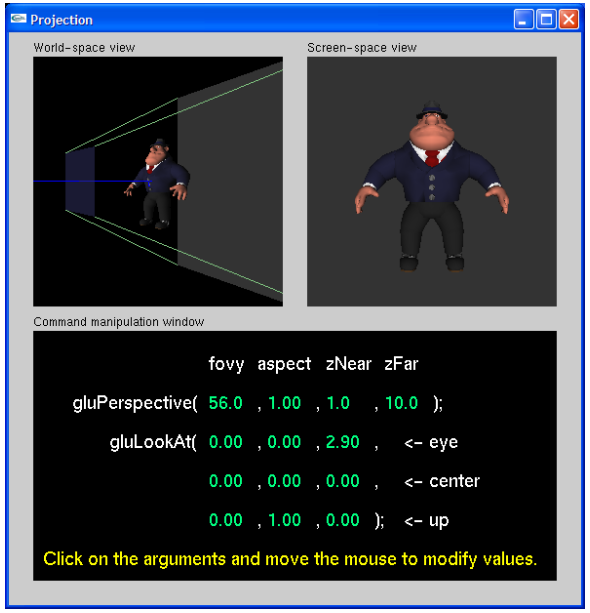
- Similar to the case in two dimensions, we divide the world into regions
  - This time we use a 6-bit region code to give us 27 different region codes
  - The bits in these regions codes are as follows:

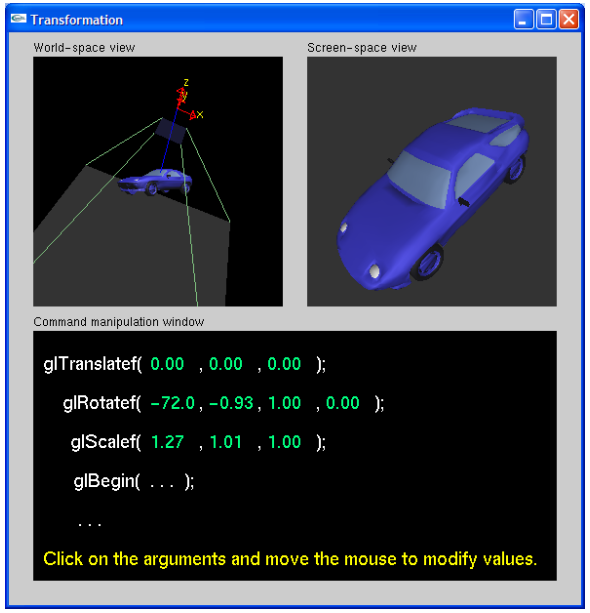
Summary

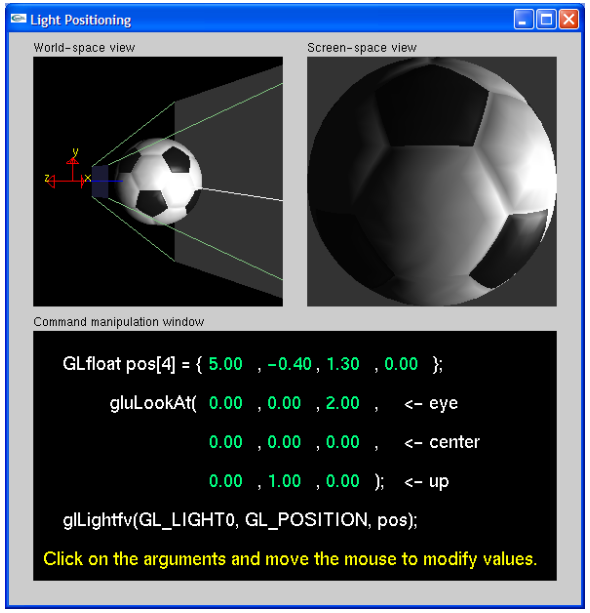
- Just like the case in two dimensions, clipping removes objects that will not be visible from the scene
- The point of this is to remove computational effort
- 3-D clipping is achieved in two basic steps
  - Discard objects that can’t be viewed
    - i.e. objects that are behind the camera, outside the field of view, or too far away
  - Clip objects that intersect with any clipping plane

Nate Robins' OpenGL Tutorials

- Nate Robins has a number of great OpenGL tutorial applications posted on his website



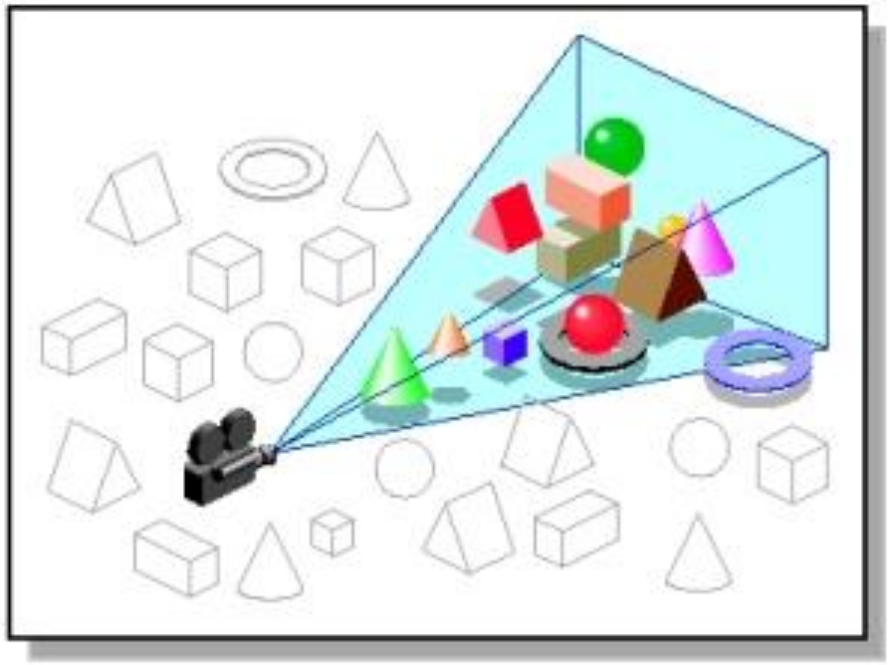




Nate Robin's OpenGL Tutorials available at: http://www.xmission.com/~nate/tutors.html

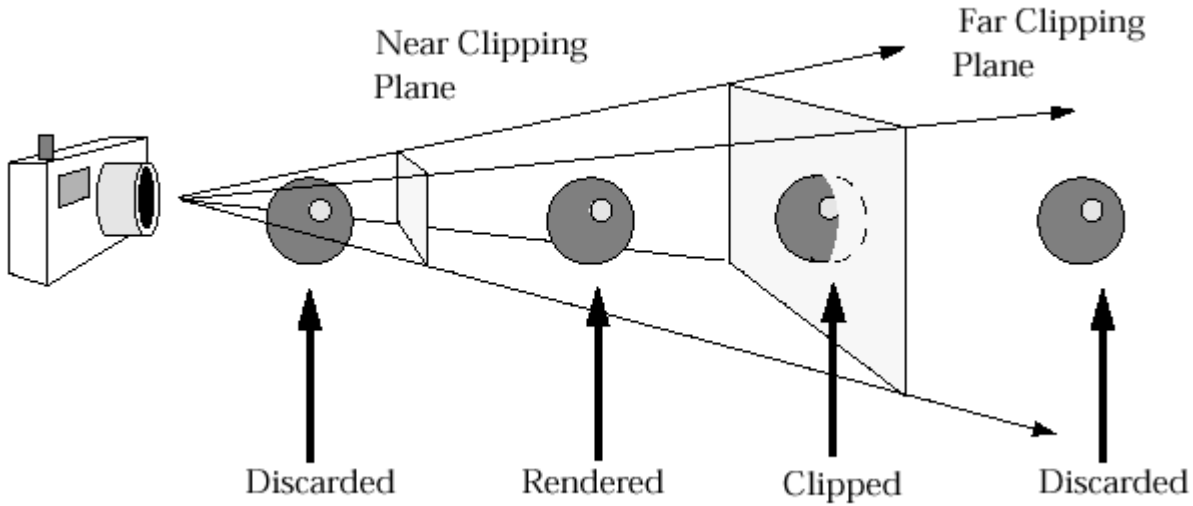
Clipping Objects

- Discarding objects that cannot possibly be seen involves comparing an objects bounding box/sphere against the dimensions of the view volume
  - Can be done before or after projection



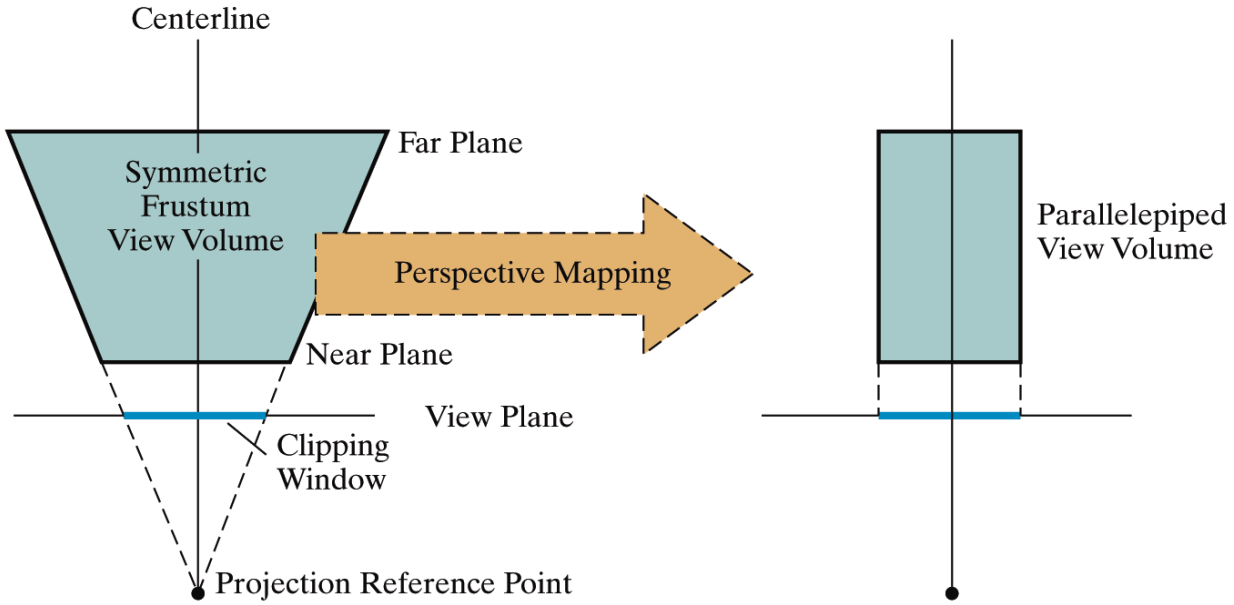
Clipping Objects

- Objects that are partially within the viewing volume need to be clipped – just like the 2D case



The clipping volume

- After the perspective transformation is complete the frustum shaped viewing volume has been converted to a parallelopiped - remember we preserved all z coordinate depth information



When do we clip?

- We perform clipping after the projection transformation and normalisation are complete
- So, we have the following:

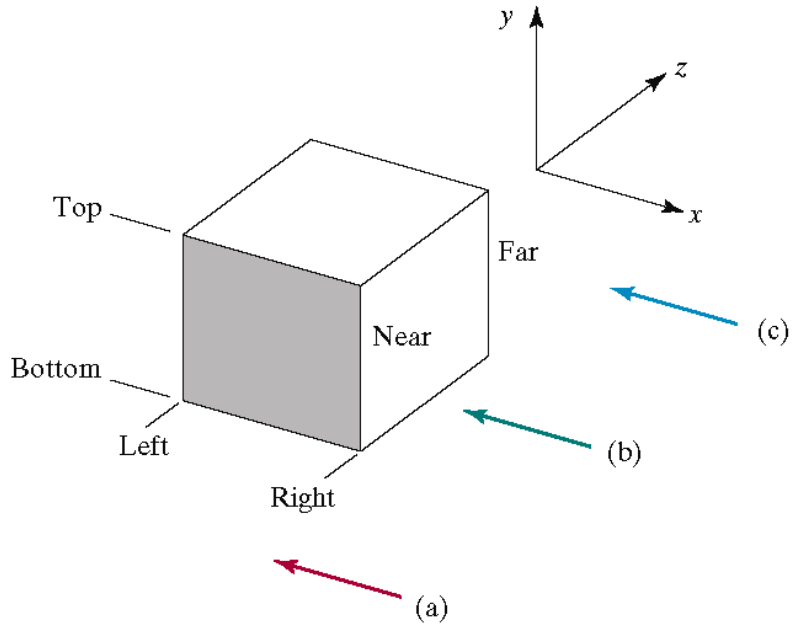
$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} = M \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Διαίρεση του κόσμου

- Similar to the case in two dimensions, we divide the world into regions
- This time we use a 6-bit region code to give us 27 different region codes
- The bits in these regions codes are as follows:

bit 6	bit 5	bit 4	bit 3	bit 2	bit 1
Far	Near	Top	Bottom	Right	Left

Region Codes



011001	011000	011010
010001	010000	010010
010101	010100	010110

Region Codes
In Front of Near Plane
(a)

001001	001000	001010
000001	000000	000010
000101	000100	000110

Region Codes
Between Near and Far Planes
(b)

101001	101000	101010
100001	100000	100010
100101	100100	100110

Region Codes
Behind Far Plane
(c)

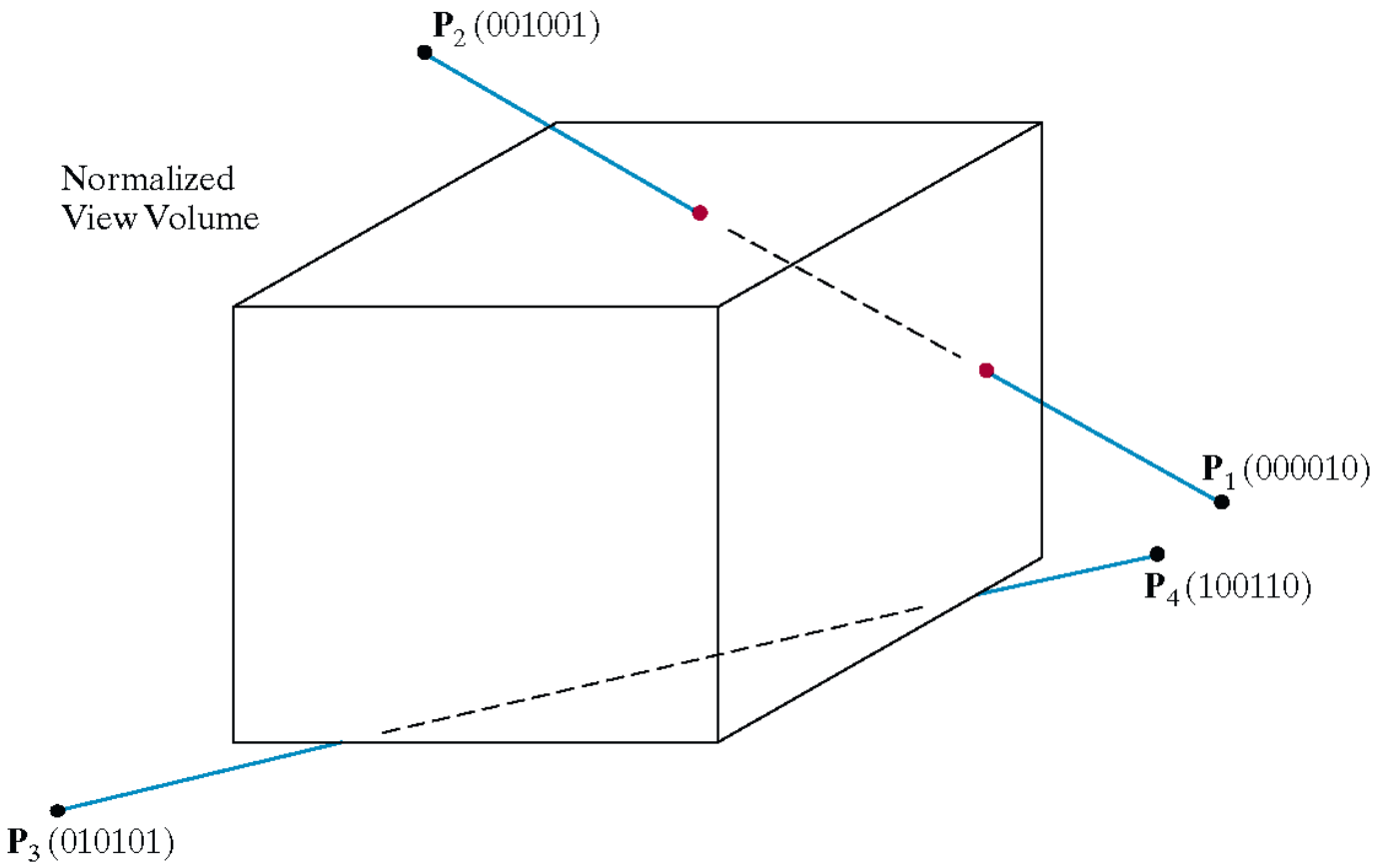
Point Clipping

- Point clipping is trivial so we won’t spend any time on it
  - If the point belongs to the [000000], then it is visible

Line Clipping

- To clip lines we first label all end points with the appropriate region codes
- We can trivially accept all lines with both end-points in the [000000] region
- We can trivially reject all lines whose end points share a common bit in any position
  - This is just like the 2 dimensional case as these lines can never cross the viewing volume
  - In the example that follows the line from P_3[010101] to P_4[100110] can be rejected

Line Clipping



The Equation Of The Line For 3D Clipping

- For clipping equations for three dimensional line segments are given in their parametric form
- For a line segment with end points $P_1(x1_h, y1_h, z1_h, h1)$ and $P_2(x2_h, y2_h, z2_h, h2)$ the parametric equation describing any point on the line is:

$$P = P_1 + (P_2 - P_1)u \qquad 0 \le u \le 1$$

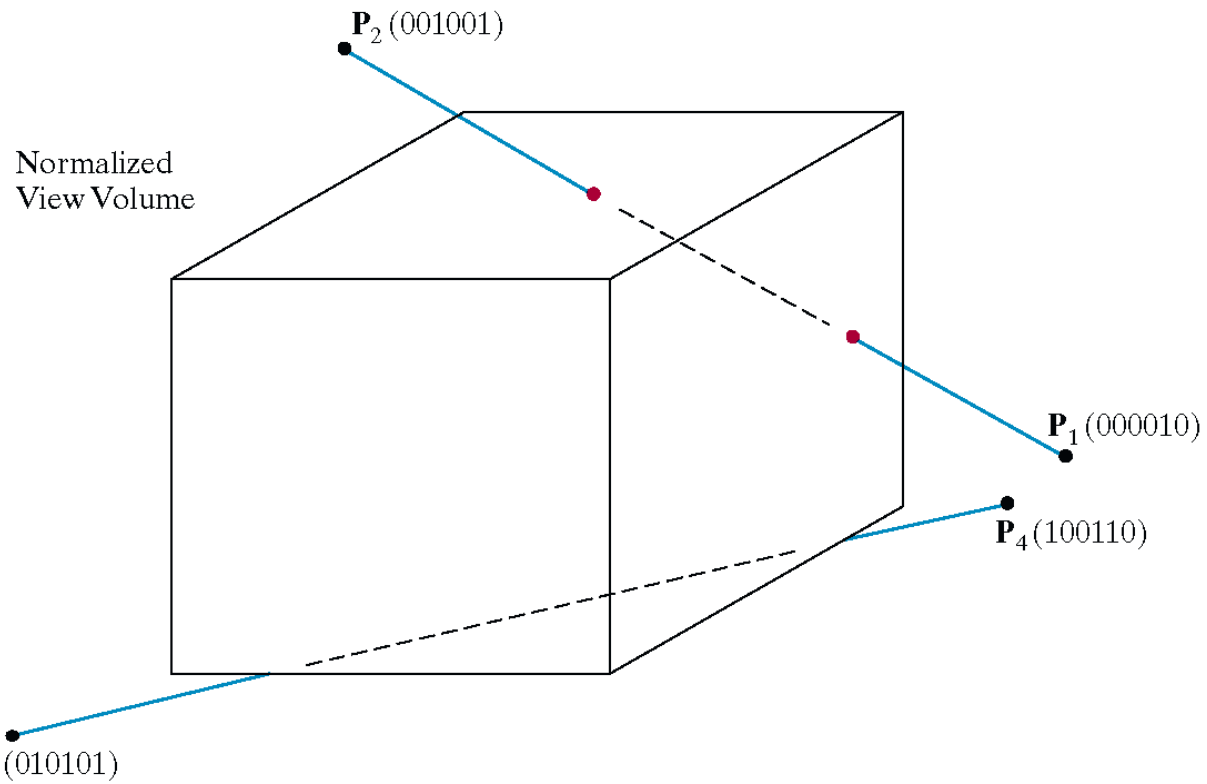
The Equation Of The Line For 3D Clipping

- From this parametric equation of a line we can generate the equations for the homogeneous coordinates:

$$x_h = x_{1,h} + (x_{2,h} - x_{1,h})u$$
$$y_h = y_{1,h} + (y_{2,h} - y_{1,h})u$$
$$z_h = z_{1,h} + (z_{2,h} - z_{1,h})u$$
$$h = h_1 + (h_2 - h_1)u$$

3D Line Clipping Example

- Consider the line P_1[000010] to P_2[001001]
- Because the lines have different values in bit 2 we know the line crosses the right boundary



3D Line Clipping Example

- Since the right boundary is at $x = 1$ we now know the following holds:

$$x_p = \frac{x_h}{h} = \frac{x_{1,h} + (x_{2,h} - x_{1,h})u}{h_1 + (h_2 - h_1)u} = 1$$

- which we can solve for u as follows:

$$u = \frac{x_{1,h} - h_1}{(x_{1,h} - h_1) - (x_{2,h} - h_2)}$$

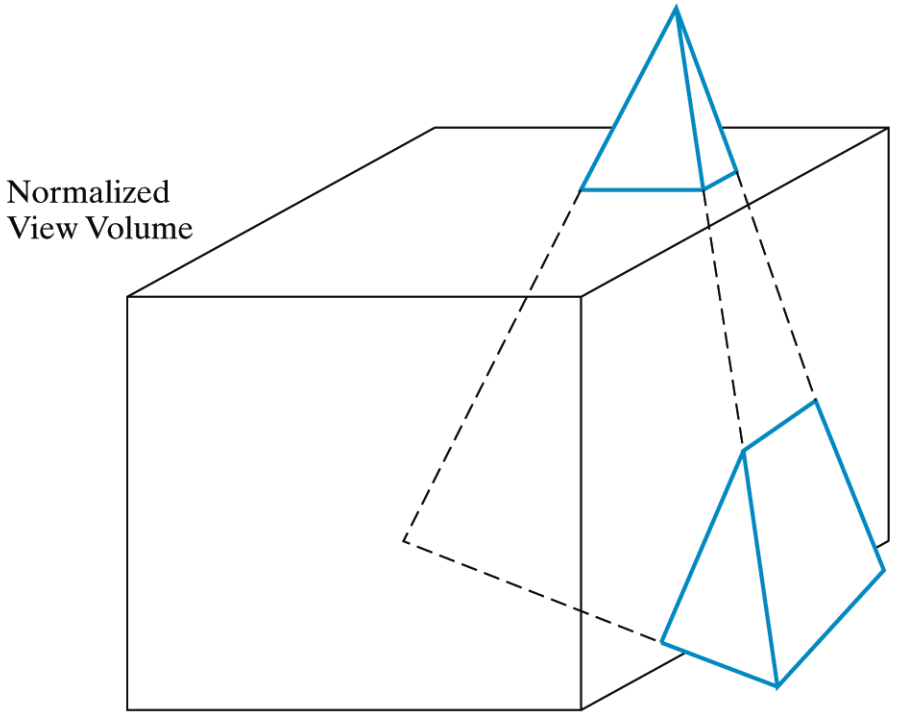
- using this value for u we can then solve for y_p and z_p similarly

3D Line Clipping Example

- When then simply continue as per the two dimensional line clipping algorithm

3D Polygon Clipping

- However the most common case in 3D clipping is that we are clipping graphics objects made up of polygons



3D Polygon Clipping

- In this case we first try to eliminate the entire object using its bounding volume
- Next we perform clipping on the individual polygons using the Sutherland-Hodgman algorithm we studied previously

Games and Clipping Planes

- Sometimes in a game you can position the camera in the right spot that the front of an object gets clipped, letting you see inside of it.
- Video games use various techniques to avoid this glitch. One technique is to have objects that are very close to the near clip plane fade out before they get cut off, as can be seen below
- This technique gives a clean look while solving the near clipping problem (the wooden fence fades out as the camera gets too close to it, allowing you to see the wolf behind it).

Screenshots from the game, *Okami*

Games and Clipping Planes

- Ever played a video game and all of a sudden some object pops up in the background (e.g., a tree in a racing game)? That's an object coming inside the far clip plane.
- Old solution: add fog in the distance. A classic example, Turok: Dinosaur Hunter
- Modern solution (e.g. Stone Giant), dynamic level of detail: mesh detail increases when you get closer to it in the game

- Thanks to fast hardware and level of detail algorithms, we can push the *far* plane back now and fog is much less prevalent