

Chapter 6

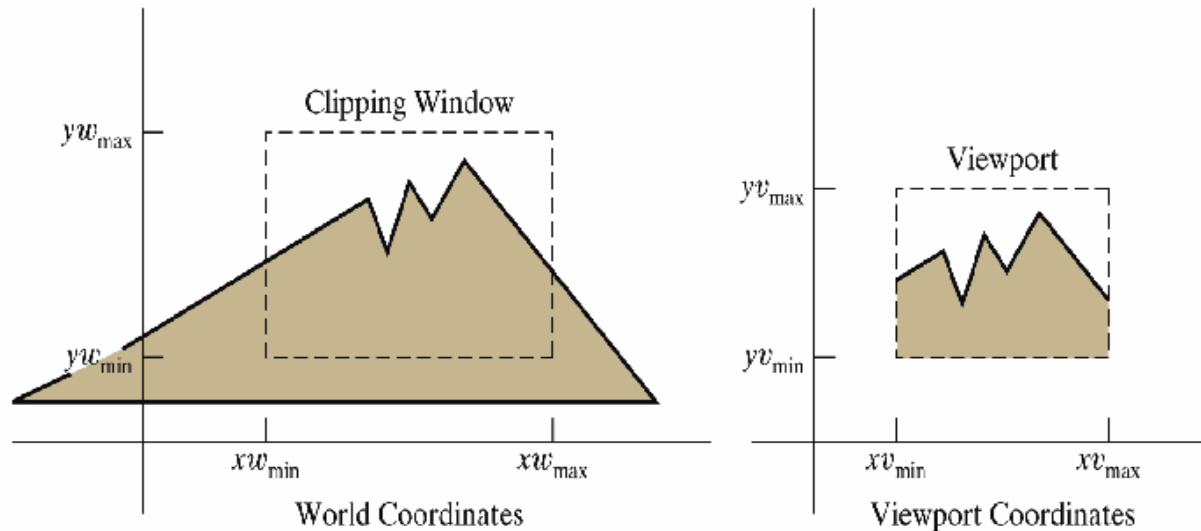
2D Viewing Transformation

2D Viewing Transformation

1. **Definition**
2. 2D Viewing Pipeline
3. Approaches
4. Aspect Ratio

2D Viewing Transformation

- 1. Definitions:** It is defined as a process for displaying views of a two-dimensional picture on an output device:
- Specify which parts of the object to display (clipping window, or world window, or viewing window)
 - Where on the screen to display these parts (view port).



2D Viewing Transformation

- **World Co – Ordinate System** is a right handed Cartesian coordinate system in which picture is actually defined.
- **Physical Device Co – Ordinate System** is a coordinate system that correspond to output device or work stations where image to be displayed. E.g. with our monitor it is a Left handed Cartesian coordinate system.
- **Normalized Co – Ordinate System** It is a right handed coordinate system in which display area of virtual display device correspond to the unit square (1x1).

2D Viewing Transformation

- **Window** or Clipping window is the selected section of a scene that is displayed on a display window. It is a finite region from World Coordinate System.
- **View port** is the window where the object is viewed on the output device. It is a finite region from Device Coordinate System.

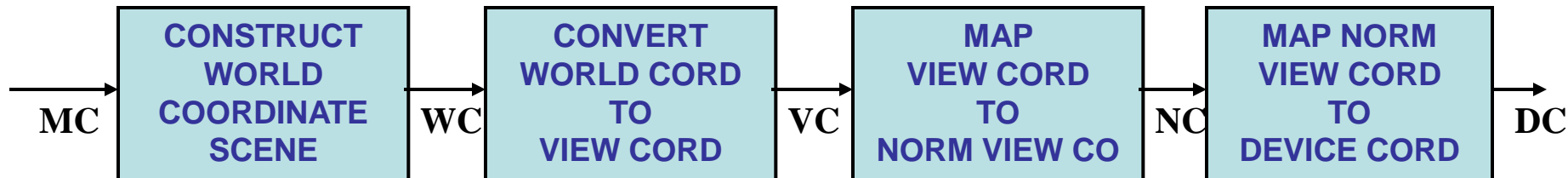
2D Viewing Transformation

1. Definition
- 2. 2D Viewing Pipeline**
3. Approaches
4. Aspect Ratio

2D Viewing Transformation

2. 2D viewing pipeline

- Construct world-coordinate scene using modeling-coordinate transformations
- Convert world-coordinates to viewing coordinates
- Transform viewing-coordinates to normalized-coordinates (ex: between 0 and 1, or between -1 and 1)
- Map normalized-coordinates to device-coordinates.



2D Viewing Transformation

1. Definition
2. 2D Viewing Pipeline
- 3. Approaches**
4. Aspect Ratio

2D Viewing Transformation

3. Approaches

- Two main approaches to 2D viewing Transformation are

1. **Direct Approach**

2. Normalized Approach

2D Viewing Transformation

3.1. Direct Approach

- Mapping the clipping window into a view port which may be normalized
- Its involves
 - translating the origin of the clipping window to that of the view port
 - scaling the clipping window to the size of the view port
- We can drive it by three methods

2D Viewing Transformation

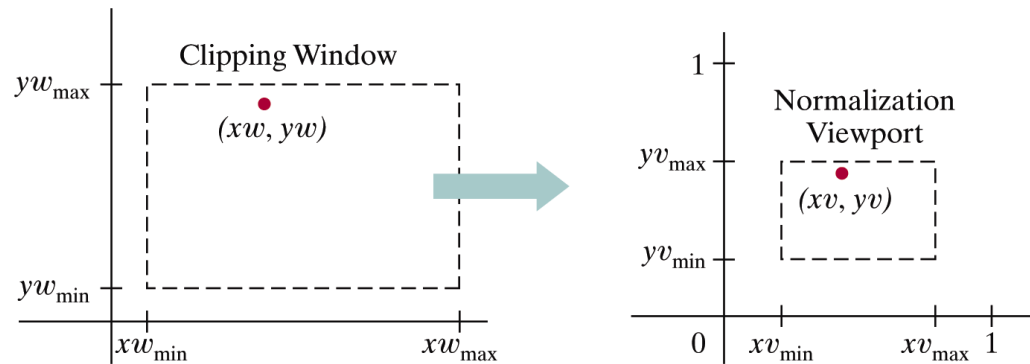


Figure 6-7

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.

(from Donald Hearn and Pauline Baker)

2D Viewing Transformation

Method 1: Let $P(x_w, y_w)$ be any point in the window, which is to be mapped to $P'(x_v, y_v)$ in the associated view port. The two steps required are:

1. Translation T_v that takes $(x_{w_{\min}}, y_{w_{\min}})$ to $(x_{v_{\min}}, y_{v_{\min}})$, where
$$v = (x_{v_{\min}} - x_{w_{\min}})I + (y_{v_{\min}} - y_{w_{\min}})J$$

2. Scaling by following scaling factors about point $(x_{v_{\min}}, y_{v_{\min}})$

$$s_x = \frac{x_{v_{\max}} - x_{v_{\min}}}{x_{w_{\max}} - x_{w_{\min}}} \quad \text{and} \quad s_y = \frac{y_{v_{\max}} - y_{v_{\min}}}{y_{w_{\max}} - y_{w_{\min}}}$$

2D Viewing Transformation

$$\begin{aligned}
 V &= S_{sx, sy, (xv_{\min}, yv_{\min})} \cdot T_v \\
 &= \begin{bmatrix} s_x & 0 & -xv_{\min} s_x + xv_{\min} \\ 0 & s_y & -yv_{\min} s_y + yv_{\min} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & xv_{\min} - xw_{\min} \\ 0 & 1 & yv_{\min} - yw_{\min} \\ 0 & 0 & 1 \end{bmatrix} \\
 &= \begin{bmatrix} s_x & 0 & -s_x * xw_{\min} + xv_{\min} \\ 0 & s_y & -s_y * yw_{\min} + yv_{\min} \\ 0 & 0 & 1 \end{bmatrix}
 \end{aligned}$$

Method

2D Viewing Transformation

Method 2: We get the same result if we perform

$$\begin{aligned} V &= T_{(xv_{\min}, yv_{\min})} S_{sx, sy} \cdot T_{(-xw_{\min}, -yw_{\min})} \\ &= \begin{bmatrix} 1 & 0 & xv_{\min} \\ 0 & 1 & yv_{\min} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -xw_{\min} \\ 0 & 1 & -yw_{\min} \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} s_x & 0 & -s_x * xw_{\min} + xv_{\min} \\ 0 & s_y & -s_y * yw_{\min} + yv_{\min} \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

2D Viewing Transformation

Method 3: Let P (x_w, y_w) be any point in the window, which is to be mapped to P' (x_v, y_v) in the associated view port. To maintain the same relative placement in the view port as in window, we require that

$$\frac{x_v - x_{v_{\min}}}{x_{v_{\max}} - x_{v_{\min}}} = \frac{x_w - x_{w_{\min}}}{x_{w_{\max}} - x_{w_{\min}}} \quad \text{and}$$

$$\frac{y_v - y_{v_{\min}}}{y_{v_{\max}} - y_{v_{\min}}} = \frac{y_w - y_{w_{\min}}}{y_{w_{\max}} - y_{w_{\min}}}$$

2D Viewing Transformation

Which means

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

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

Put these equations in the matrix form.

2D Viewing Transformation

3.2 Normalized Approach

- Mapping the clipping window into a normalized square
- It involves
 - transforming the clipping window into a normalized square
 - Then clipping in normalized coordinates (ex: -1 to 1) or (ex: 0 to 1)
 - Then transferring the scene description to a view port specified in screen coordinates.
 - Finally positioning the view port area in the display window.
- Thus $V = W.N$, where N is a transformation that maps window to normalized view port and W maps Normalized points to view port.

2D Viewing Transformation

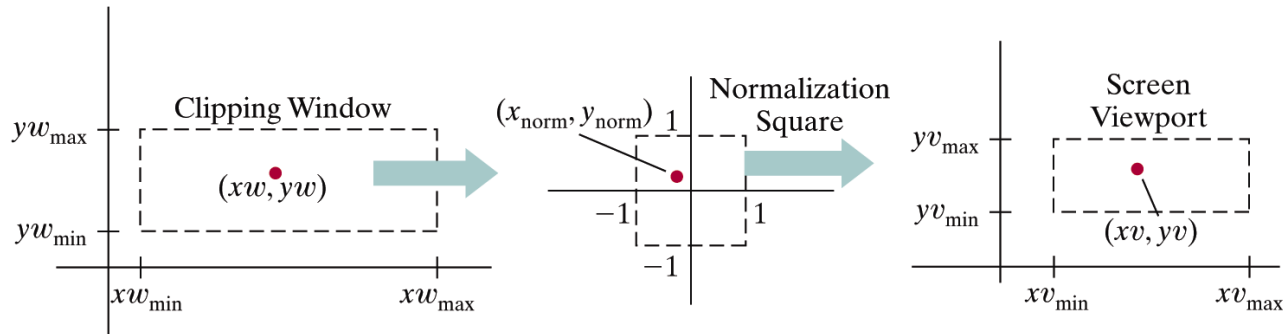


Figure 6-8

A point (x_w, y_w) in a clipping window is mapped to a normalized coordinate position $(x_{\text{norm}}, y_{\text{norm}})$, then to a screen-coordinate position (x_v, y_v) in a viewport. Objects are clipped against the normalization square before the transformation to viewport coordinates.

(from Donald Hearn and Pauline Baker)

2D Viewing Transformation

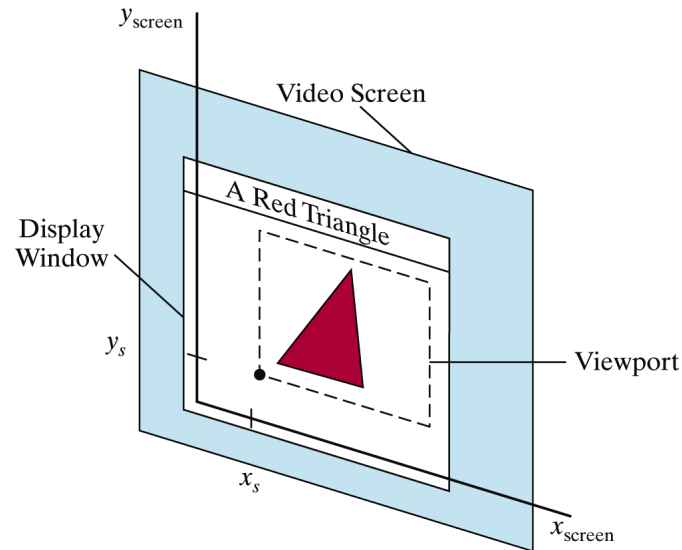


Figure 6-9

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

(from Donald Hearn and Pauline Baker)

Exercise 1

Find the normalization transformation that maps a window defined by (1,1) to (3,5) on to

1. The view port that is entire normalized device.

$$V = \begin{bmatrix} 1/2 & 0 & -1/2 \\ 0 & 1/4 & -1/4 \\ 0 & 0 & 1 \end{bmatrix}$$

2. Has lower left corner (0,0) and upper right corner as (1/2,1/2)

$$V = \begin{bmatrix} 1/4 & 0 & -1/4 \\ 0 & 1/8 & -1/8 \\ 0 & 0 & 1 \end{bmatrix}$$

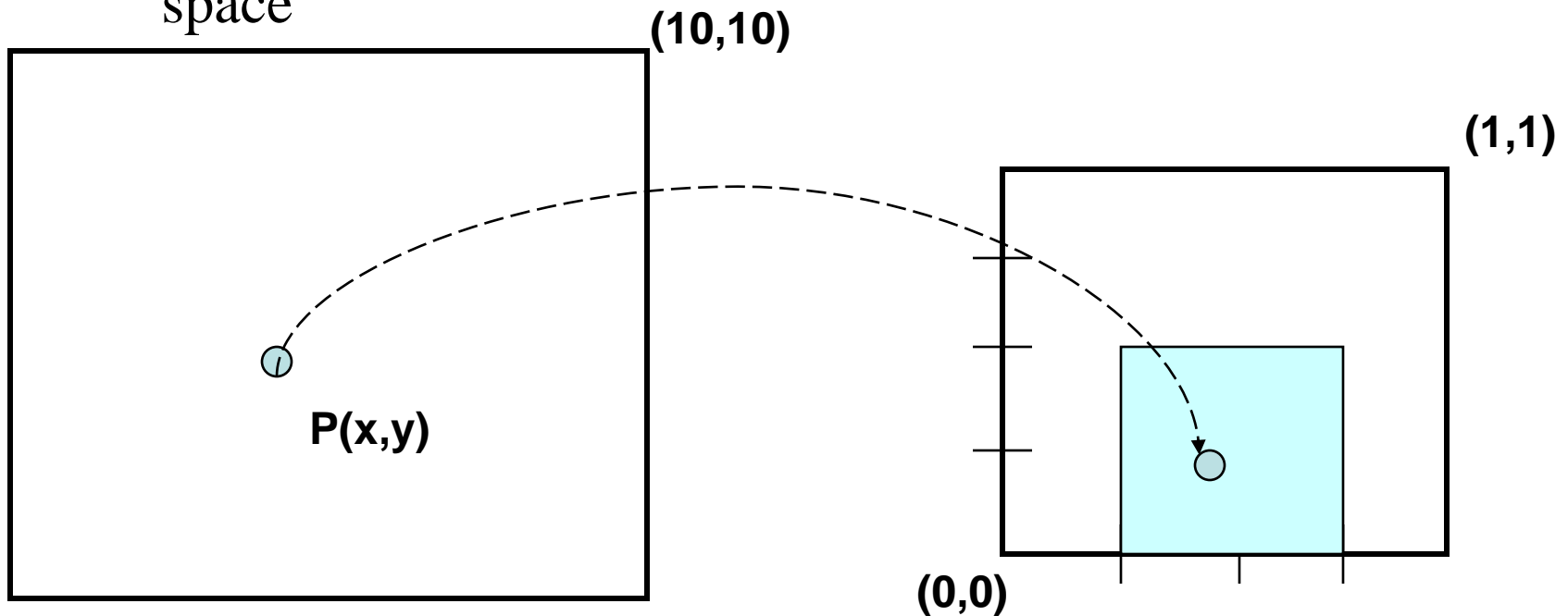
Exercise 2

Find the complete viewing transformation that

1. First maps a window defined by $(1,1)$ to $(10,10)$ on to a view port of size $(1/4,0)$ to $(3/4,1/2)$ in normalized device space
2. Then maps a window of $(1/4,1/4)$ to $(1/2,1/2)$ in normalized device space to view port of $(1,1)$ to $(10,10)$

Exercise 2

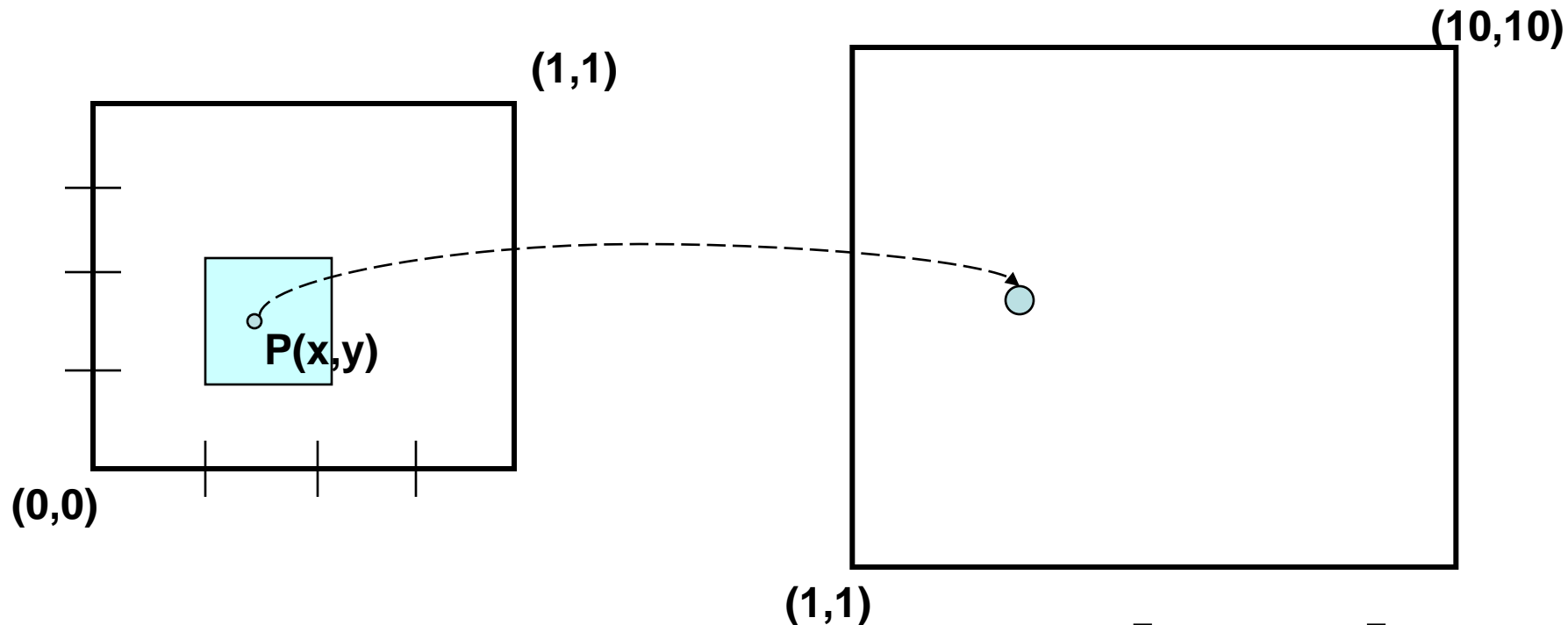
First maps a window defined by (1,1) to (10,10) on to a view port of size (1/4,0) to (3/4,1/2) in normalized device space



$$N = \begin{bmatrix} 1/18 & 0 & 7/36 \\ 0 & 1/18 & -1/18 \\ 0 & 0 & 1 \end{bmatrix} \quad 23$$

Exercise 2

Then maps a window of $(1/4, 1/4)$ to $(1/2, 1/2)$ in normalized device space to view port of $(1, 1)$ to $(10, 10)$



$$W = \begin{bmatrix} 36 & 0 & -8 \\ 0 & 36 & -8 \\ 0 & 0 & 1 \end{bmatrix}$$

$$V = W.N = \begin{bmatrix} 2 & 0 & -1 \\ 0 & 2 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

Exercise 2

1. First maps a window defined by (1,1) to (10,10) on to a view port of size(1/4,0) to (3/4,1/2) in normalized device space

$$N = \begin{bmatrix} 1/18 & 0 & 7/36 \\ 0 & 1/18 & -1/18 \\ 0 & 0 & 1 \end{bmatrix}$$

2. Then maps a window of (1/4,1/4) to (1/2,1/2) in normalized device space to view port of (1,1) to (10,10)

$$W = \begin{bmatrix} 36 & 0 & -8 \\ 0 & 36 & -8 \\ 0 & 0 & 1 \end{bmatrix} \quad V = W.N = \begin{bmatrix} 2 & 0 & -1 \\ 0 & 2 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

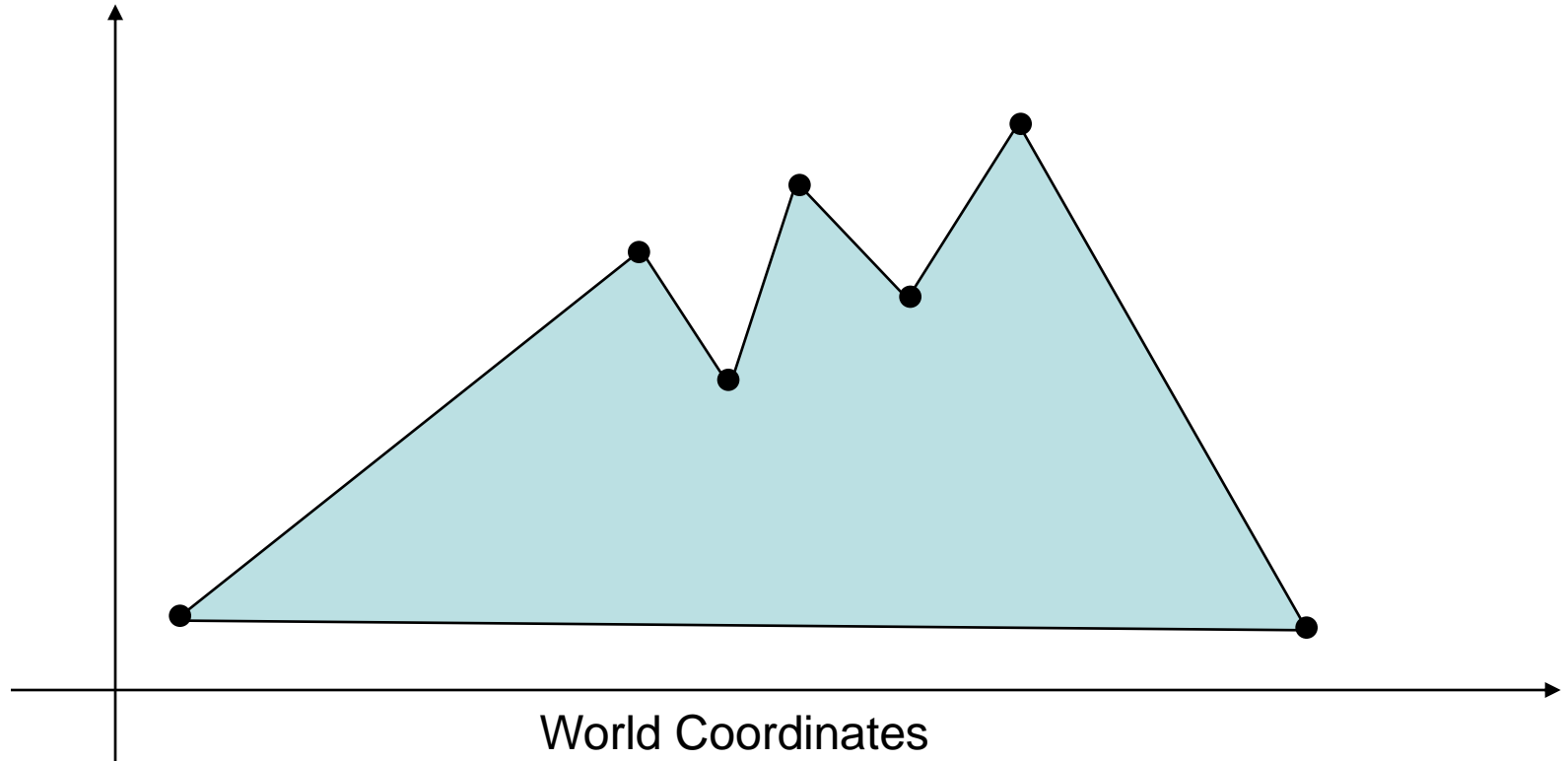
2D Clipping

- 1. Introduction**
2. Point Clipping
3. Line Clipping
4. Polygon/Area Clipping
5. Text Clipping

2D Clipping

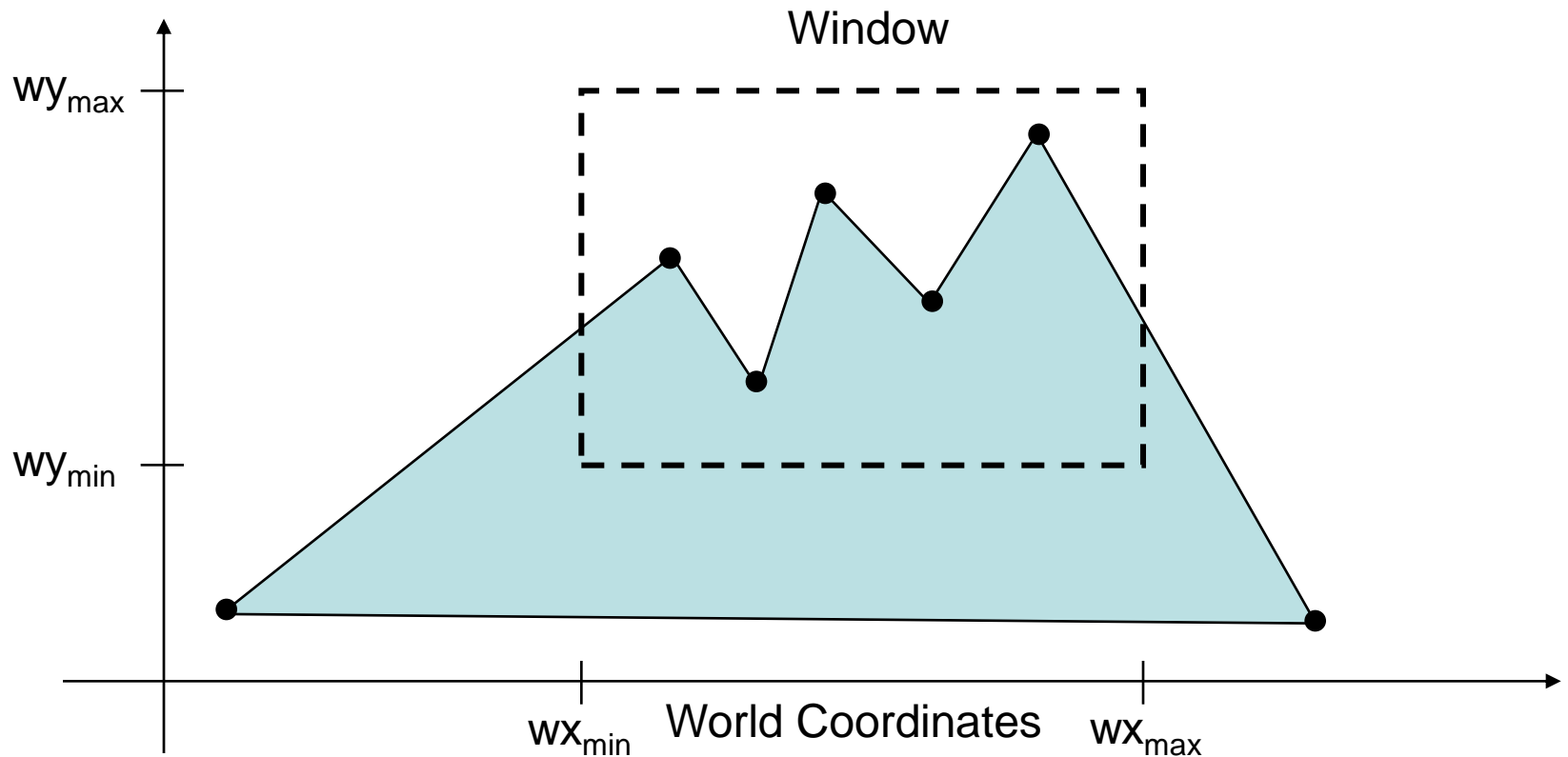
1. Introduction:

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



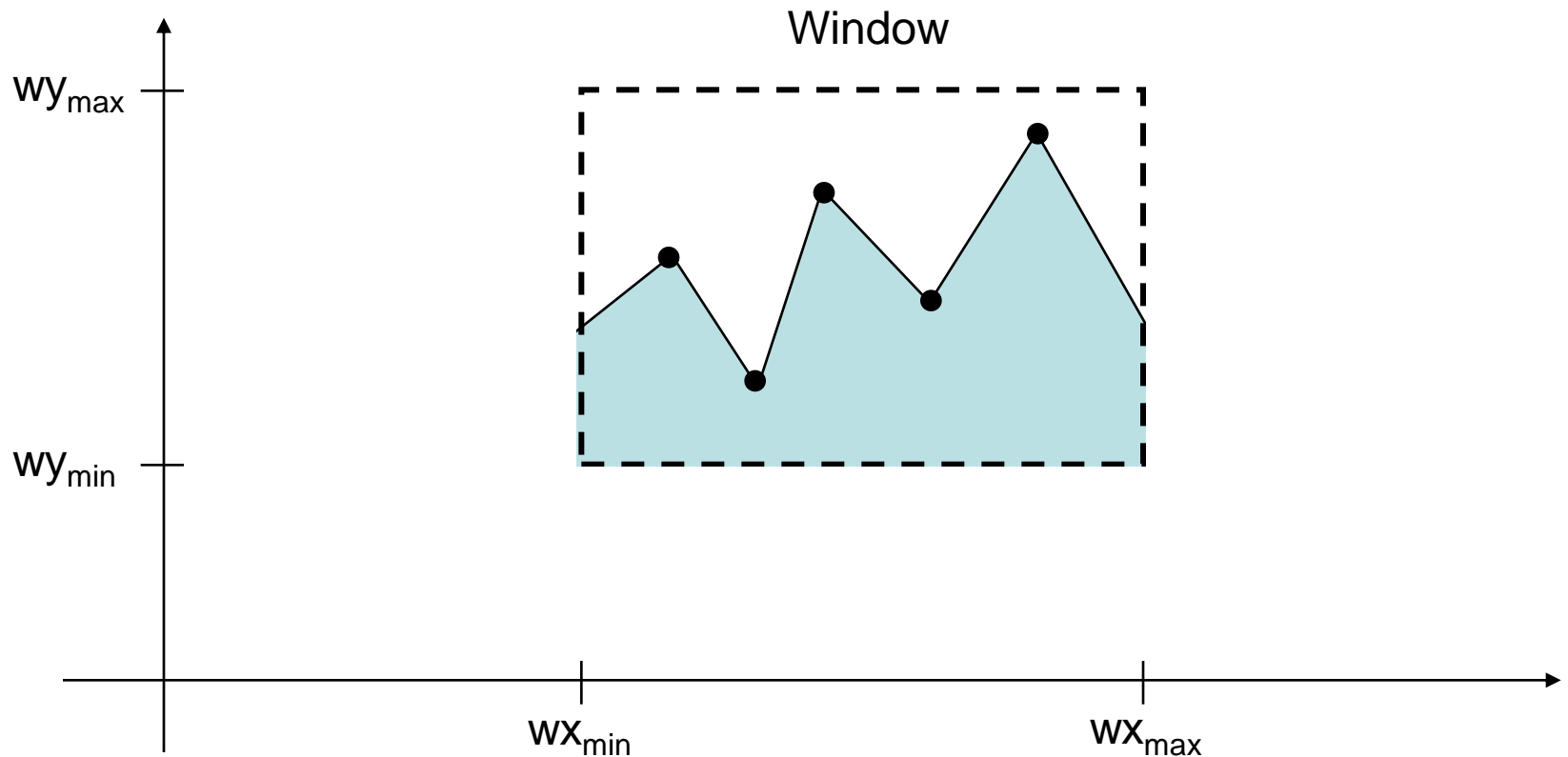
2D Clipping

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



2D Clipping

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

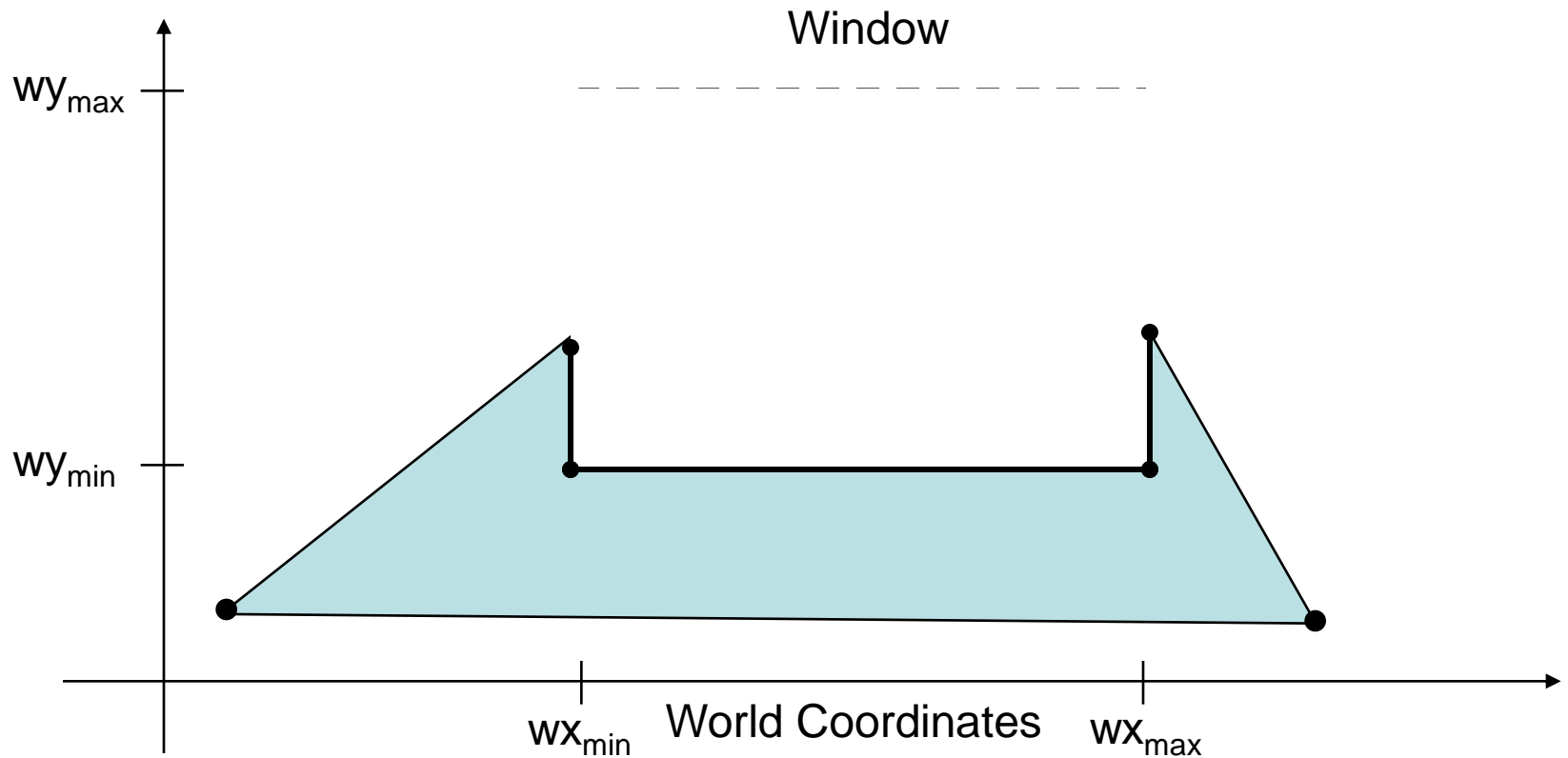


2D Clipping

1.1 Definitions:

- *Clipping* is the process of determining which elements of the picture lie inside the window and are visible.
- *Shielding* is the reverse operation of clipping where window act as the block used to abstract the view.
- By default, the “*clip window*” is the entire canvas
 - not necessary to draw outside the canvas
 - for some devices, it is damaging (plotters)
- Sometimes it is convenient to restrict the “clip window” to a smaller portion of the canvas
 - partial canvas redraw for menus, dialog boxes, other obscuration

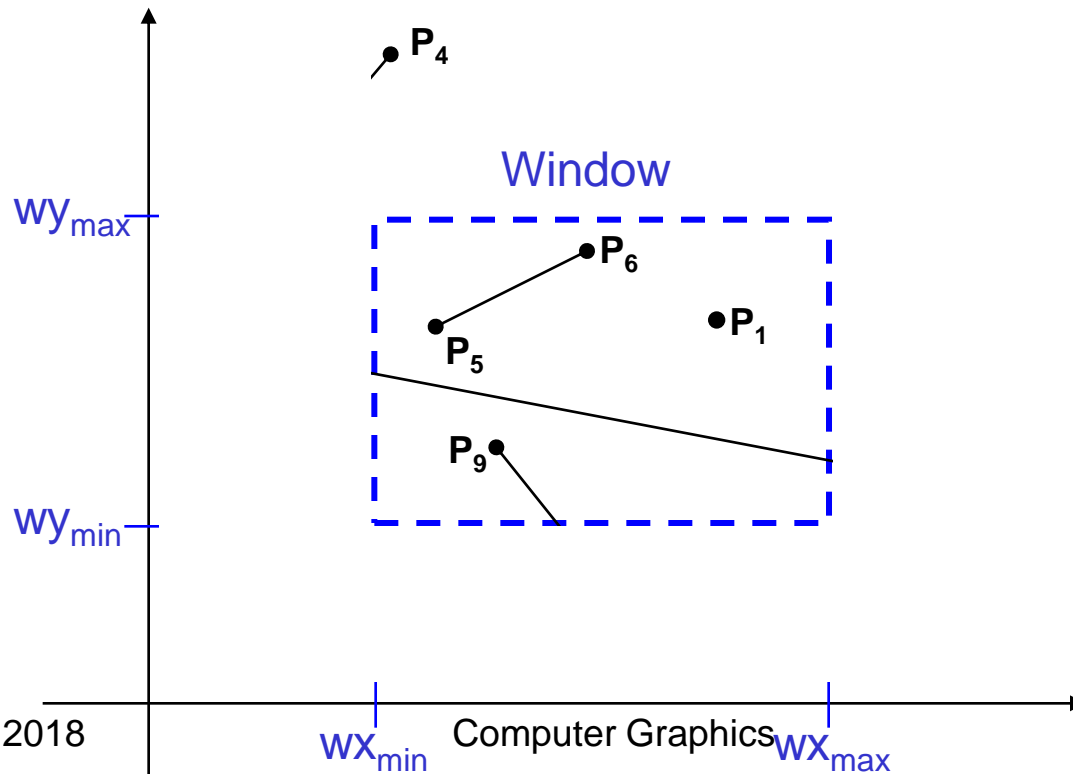
2D Clipping



2D Clipping

1.2 Example:

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



2D Clipping

1.3 Applications:

- Extract part of a defined scene for viewing.
- Drawing operations such as erase, copy, move etc.
- Displaying multi view windows.
- Creating objects using solid modeling techniques.
- Anti-aliasing line segments or object boundaries.
- Identify visible surfaces in 3D views.

2D Clipping

1.4 Types of clipping:

- Three types of clipping techniques are used depending upon when the clipping operation is performed

a. Analytical clipping

- Clip it before you scan convert it
- used mostly for lines, rectangles, and polygons, where clipping algorithms are simple and efficient

2D Clipping

b. Scissoring

- Clip it during scan conversion
- a brute force technique
 - scan convert the primitive, only write pixels if inside the clipping region
 - easy for thick and filled primitives as part of scan line fill
 - if primitive is not much larger than clip region, most pixels will fall inside
 - can be more efficient than analytical clipping.

2D Clipping

c. Raster Clipping

- Clip it after scan conversion
- render everything onto a temporary canvas and copy the clipping region
 - wasteful, but simple and easy,
 - often used for text

2D Clipping

1.5 Levels of clipping:

- Point Clipping
- Line Clipping
- Polygon Clipping
- Area Clipping
- Text Clipping
- Curve Clipping

2D Clipping

1. Introduction
- 2. Point Clipping**
3. Line Clipping
4. Polygon/Area Clipping
5. Text Clipping

Point Clipping

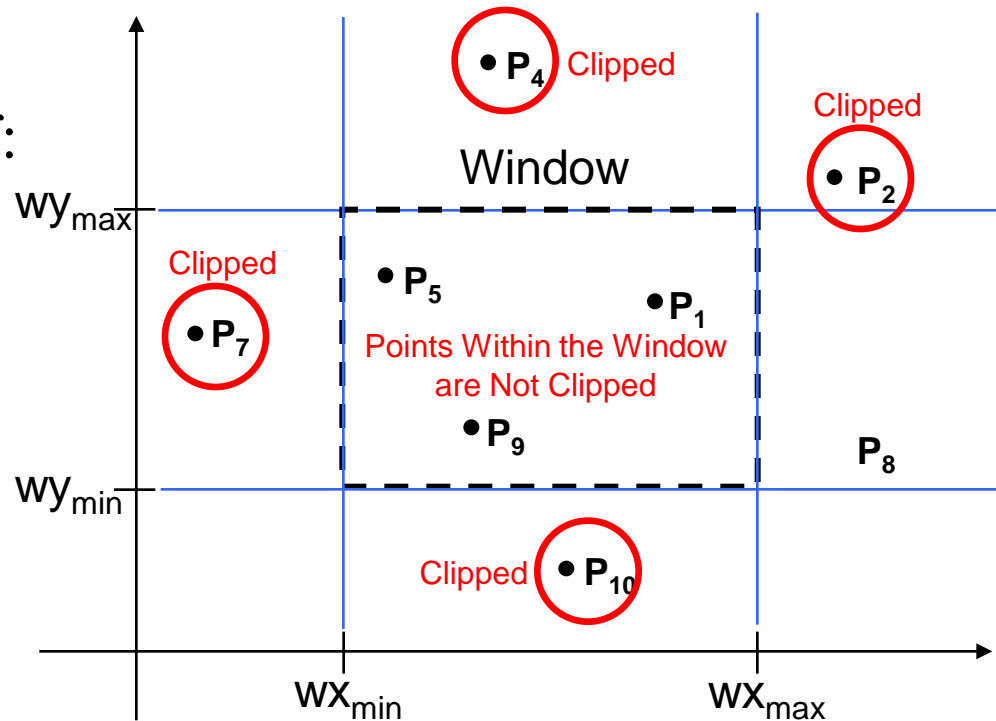
- Simple and Easy
- a point (x,y) is not clipped if:

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

&

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

- otherwise it is clipped

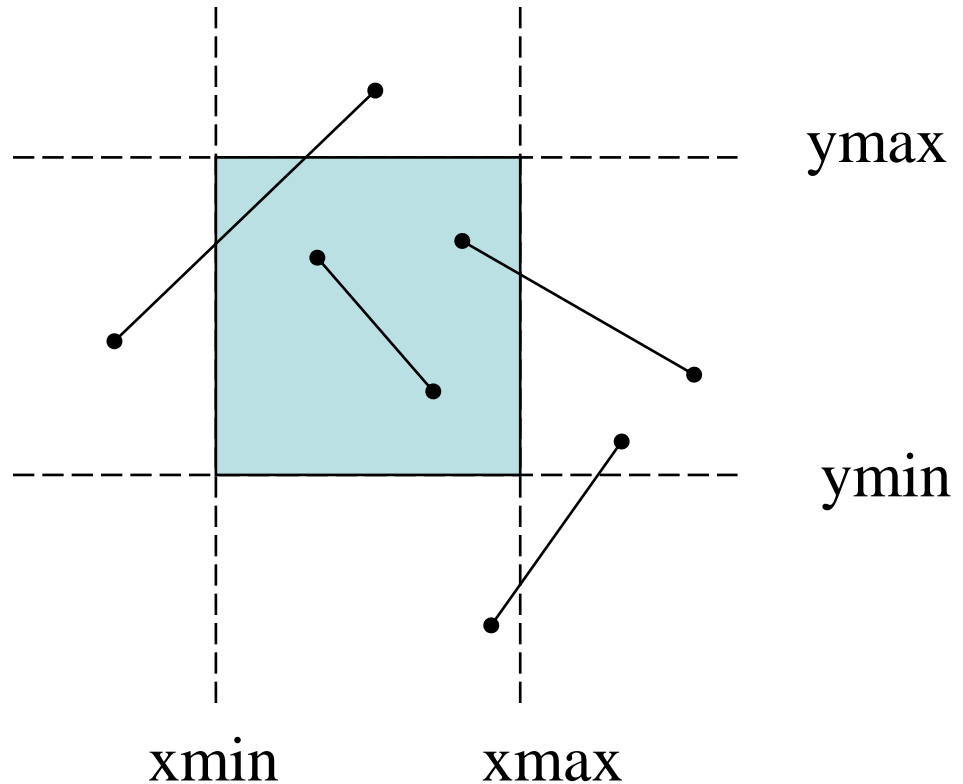


2D Clipping

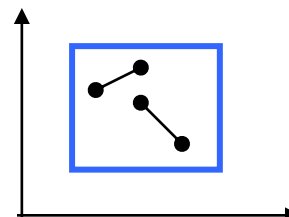
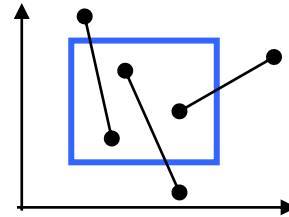
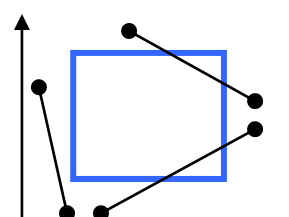
1. Introduction
2. Point Clipping
- 3. Line Clipping**
4. Polygon/Area Clipping
5. Text Clipping

Line Clipping

- It is Harder than point clipping
- We first examine the endpoints of each line to see if they are in the window or not
 - Both endpoints inside, line trivially accepted
 - One in and one out, line is partially inside
 - Both outside, might be partially inside
 - What about trivial cases?



Line Clipping

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!	

2D Line Clipping Algorithms

1. Analytical Line Clipping
- 2. Cohen Sutherland Line Clipping**
3. Liang Barsky Line Clipping

Cohen-Sutherland Line Clipping

- 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 is something of a mystery – can anybody find out who he was?



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 Line Clipping

- Two phases Algorithm

Phase I: Identification Phase

All line segments fall into one of the following categories

1. Visible: Both endpoints lies inside
2. Invisible: Line completely lies outside
3. Clipping Candidate: A line neither in category 1 or 2

Phase II: Perform Clipping

Compute intersection for all lines that are candidate for clipping.

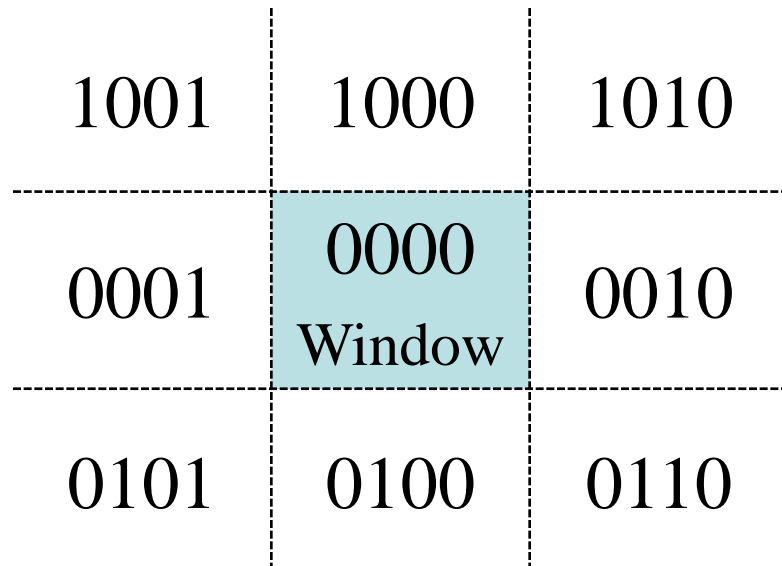
Cohen-Sutherland Line Clipping

Phase I: Identification Phase: 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

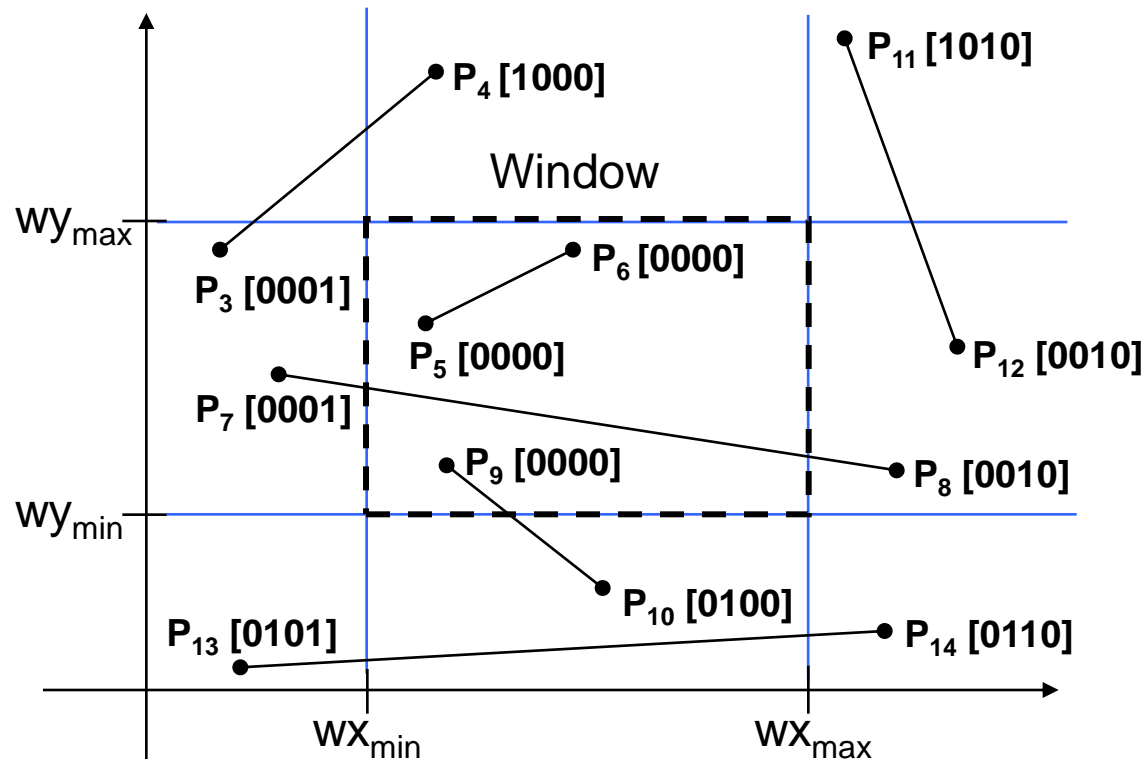
3	2	1	0
above	below	right	left

Region Code Legend



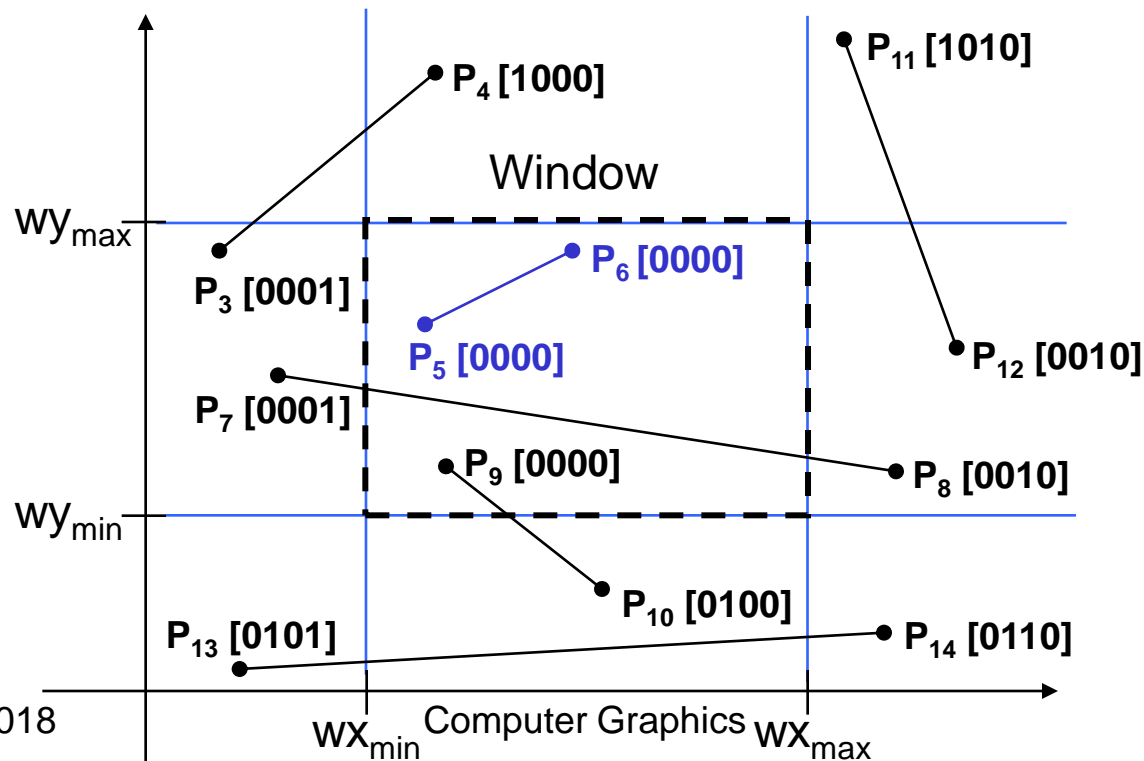
Cohen-Sutherland Line Clipping

Every end-point is labelled with the appropriate region code



Cohen-Sutherland Line Clipping

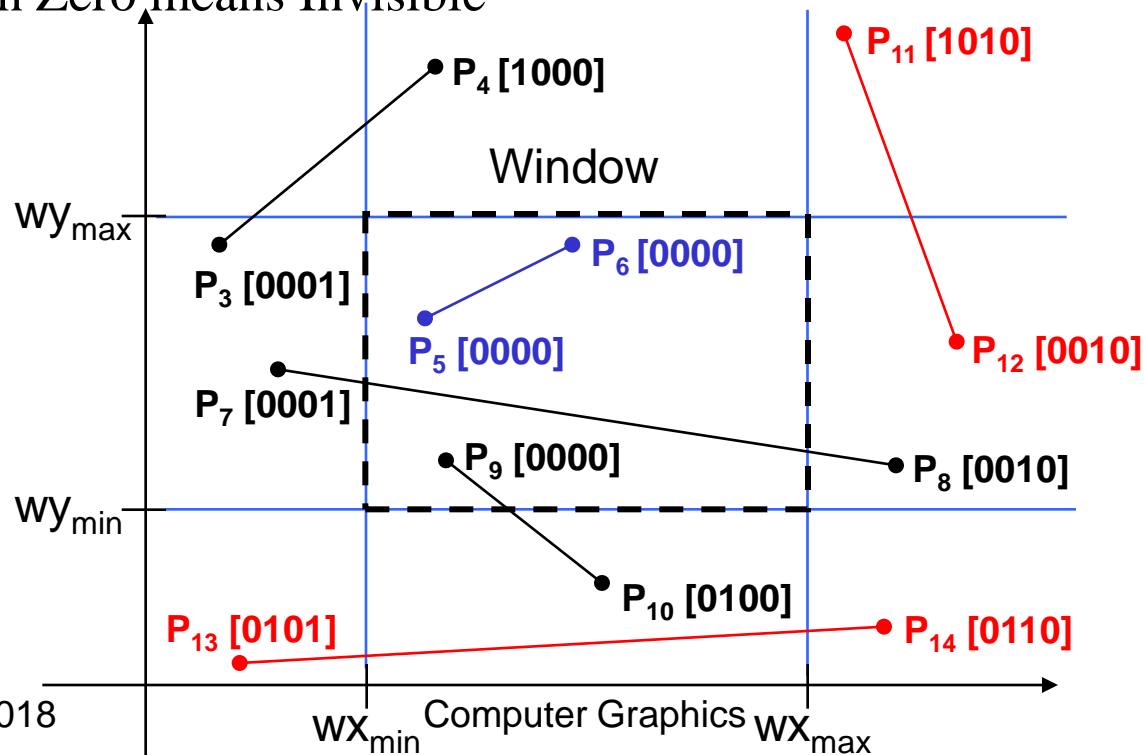
Visible Lines: Lines completely contained within the window boundaries have region code [0000] for both end-points so are not clipped



Cohen-Sutherland Line Clipping

Invisible Lines: Any line with a common set bit in the region codes of both end-points can be clipped completely

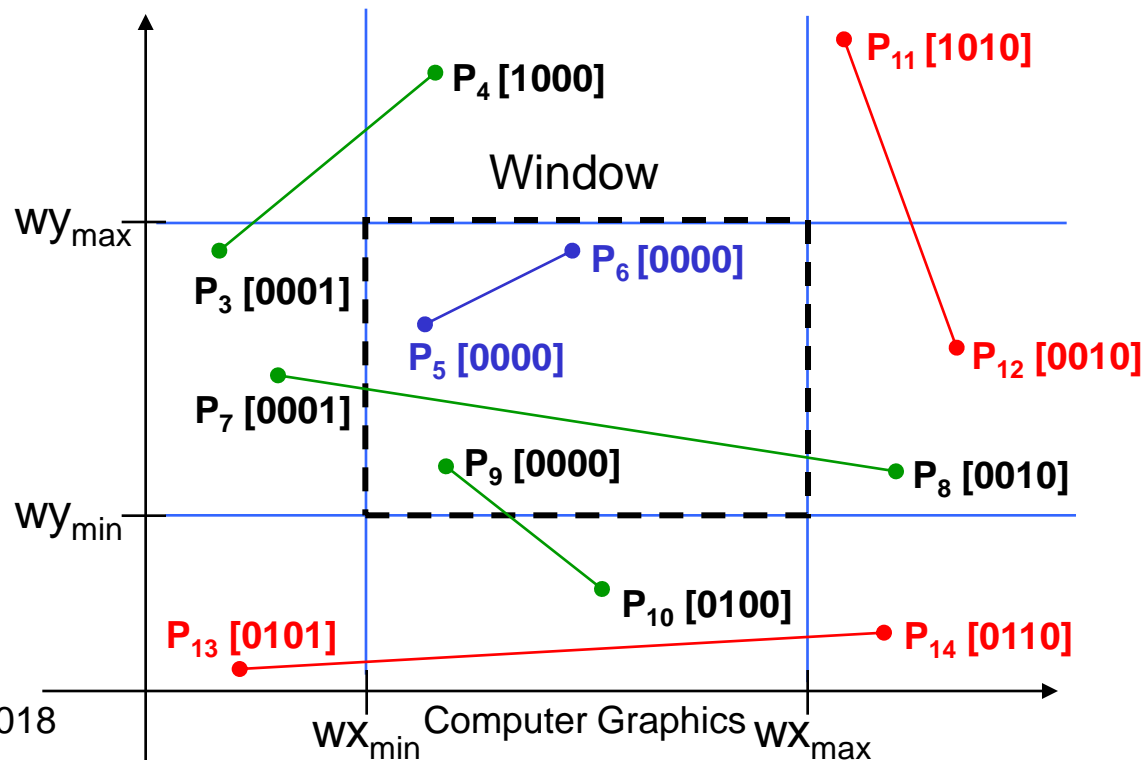
- The AND operation can efficiently check this
- Non Zero means Invisible



Cohen-Sutherland Line Clipping

Clipping Candidates: Lines that cannot be identified as completely inside or outside the window may or may not cross the window interior. These lines are processed in Phase II.

- If AND operation result in 0 the line is candidate for clipping



Cohen-Sutherland Line Clipping

Assigning Codes

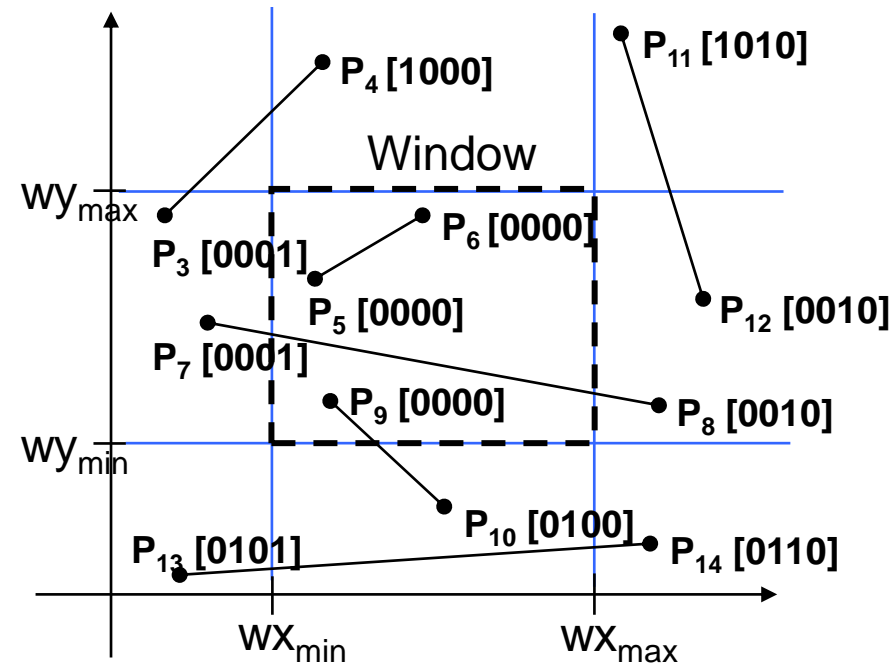
- Let point (x,y) is be given code $b_3b_2b_1b_0$:

bit 3 = 1 if $wy_{max} - y \leq 0$

bit 2 = 1 if $y - wy_{min} \leq 0$

bit 1 = 1 if $wx_{max} - x \leq 0$

bit 0 = 1 if $x - wx_{min} \leq 0$



Cohen-Sutherland Clipping Algorithm

Phase II: Clipping Phase: Lines that are in category 3 are now 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 Line Clipping

- 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}

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

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

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

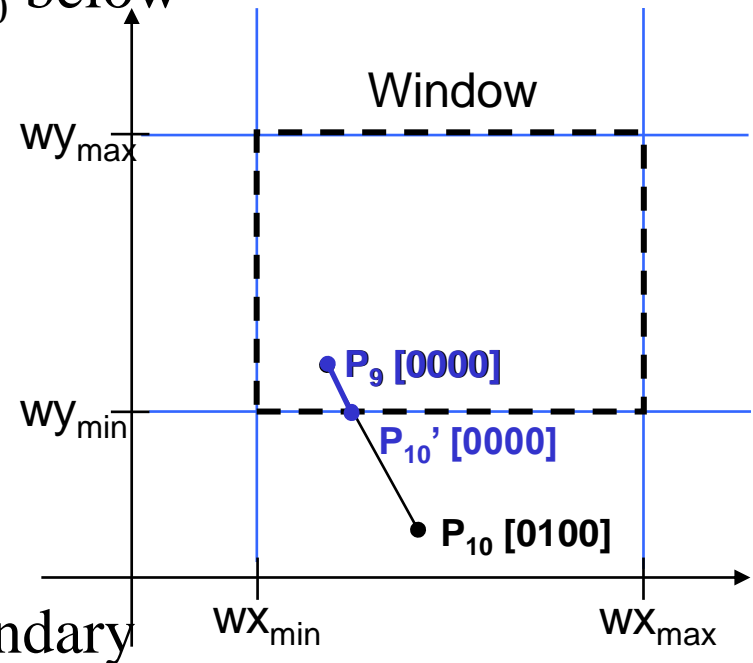
Cohen-Sutherland Line Clipping

- 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 endpoints
 - If one of these is a 1 and the other is a 0 then the line crosses the boundary.

Cohen-Sutherland Line Clipping

Example1: Consider the line P_9 to P_{10} below

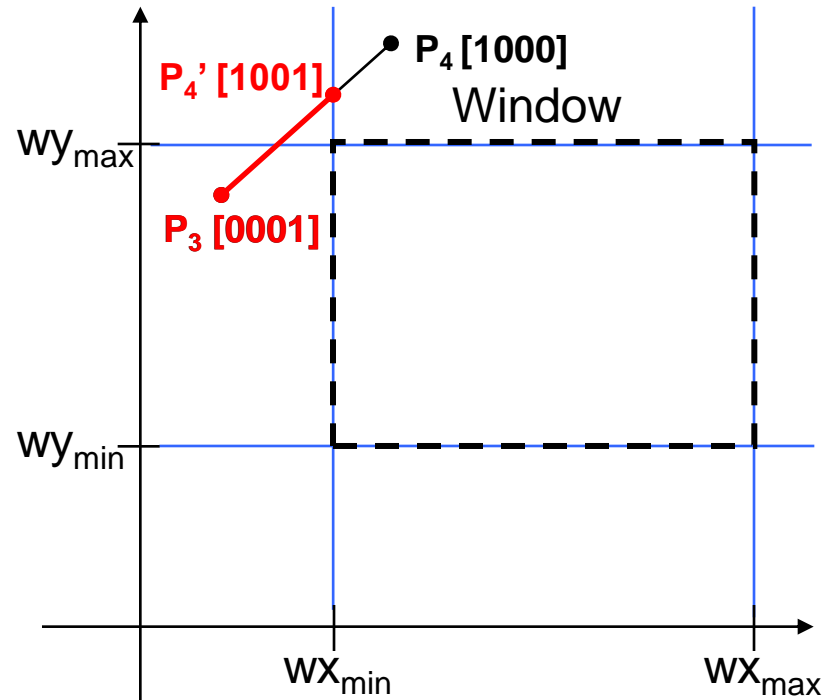
- Start at P_{10}
- From the region codes of the two end-points we know the line doesn't cross the left or right boundary
- Calculate the intersection of the line with the bottom boundary to generate point P_{10}'
- The line P_9 to P_{10}' is completely inside the window so is retained



Cohen-Sutherland Line Clipping

Example 2: 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'

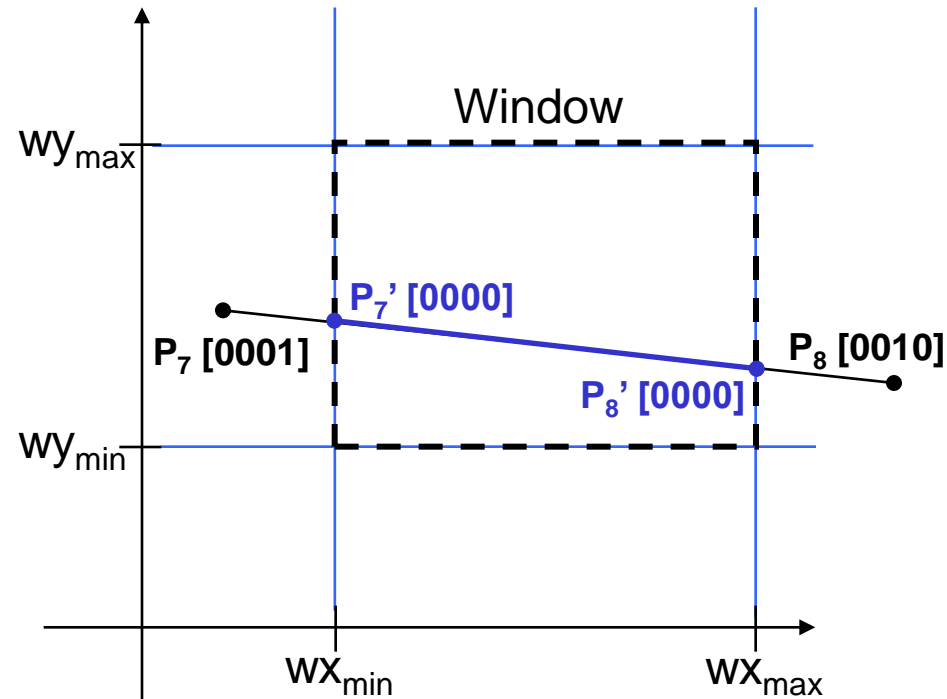


- The line P_3 to P_4' is completely outside the window so is clipped

Cohen-Sutherland Line Clipping

Example 3: Consider the line P_7 to P_8 below

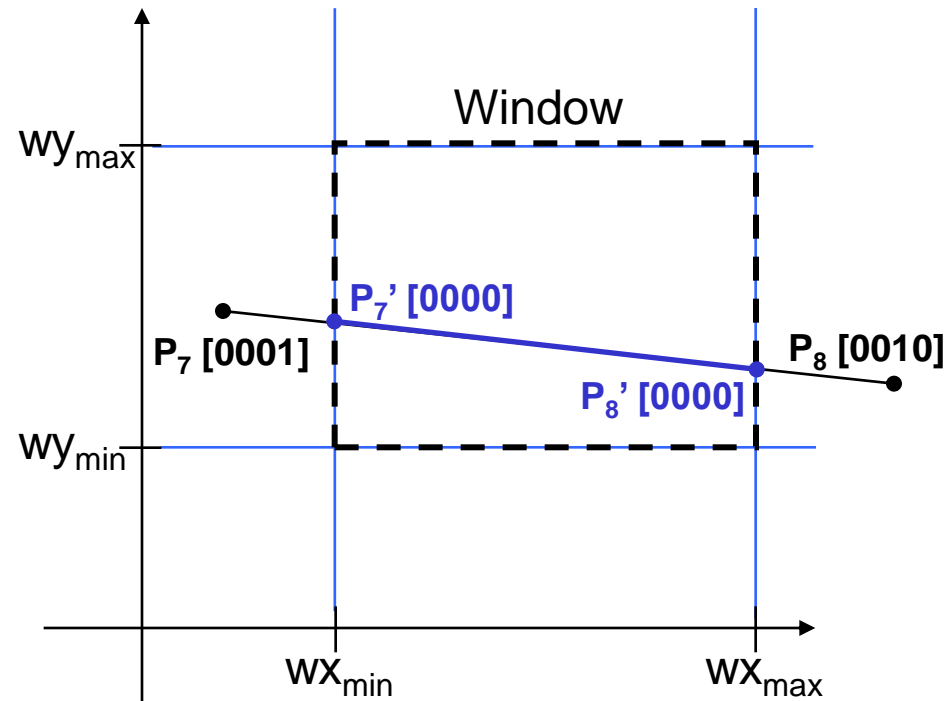
- Start at P_7
- From the two region codes of the two end-points we know the line crosses the left boundary so calculate the intersection point to generate P_7'



Cohen-Sutherland Line Clipping

Example 4: 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



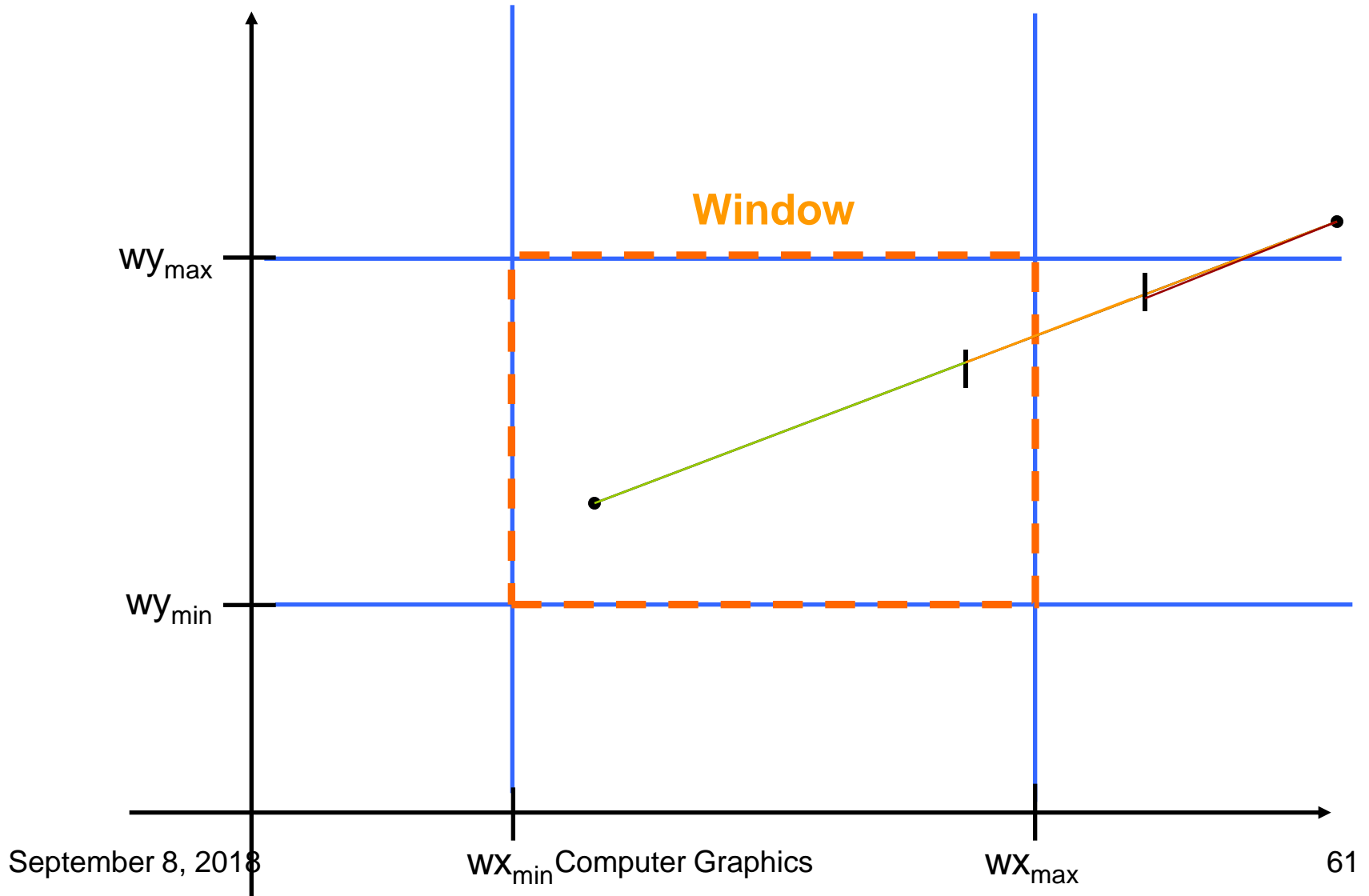
Cohen-Sutherland Line Clipping

Mid-Point Subdivision Method

– Algorithm

1. Initialise the list of lines to all lines
2. Classify lines as in *Phase I*
 - i. Assign 4 point bit codes to both end points $a_3a_2a_1a_0$ and $b_3b_2b_1b_0$
 - ii. If $(a_3a_2a_1a_0 = b_3b_2b_1b_0 = 0)$ Line in category 1
 - iii. If $(a_3a_2a_1a_0) \text{ AND } (b_3b_2b_1b_0) \neq 0$ Line in category 2
 - iv. If $(a_3a_2a_1a_0) \text{ AND } (b_3b_2b_1b_0) = 0$ Line in category 3
3. Display all lines from the list in category 1 and remove;
4. Delete all lines from the list in category 2 as they are invisible;
5. Divide all lines of category 3 are into two smaller segments at mid-point (x_m, y_m) where $x_m = (x_1 + x_2)/2$ and $y_m = (y_1 + y_2)/2$
6. Remove the original line from list and enter its two newly created segments.
7. Repeat step 2-5 until list is null.

Cohen-Sutherland Line Clipping



Cohen-Sutherland Line Clipping

Mid-Point Subdivision Method

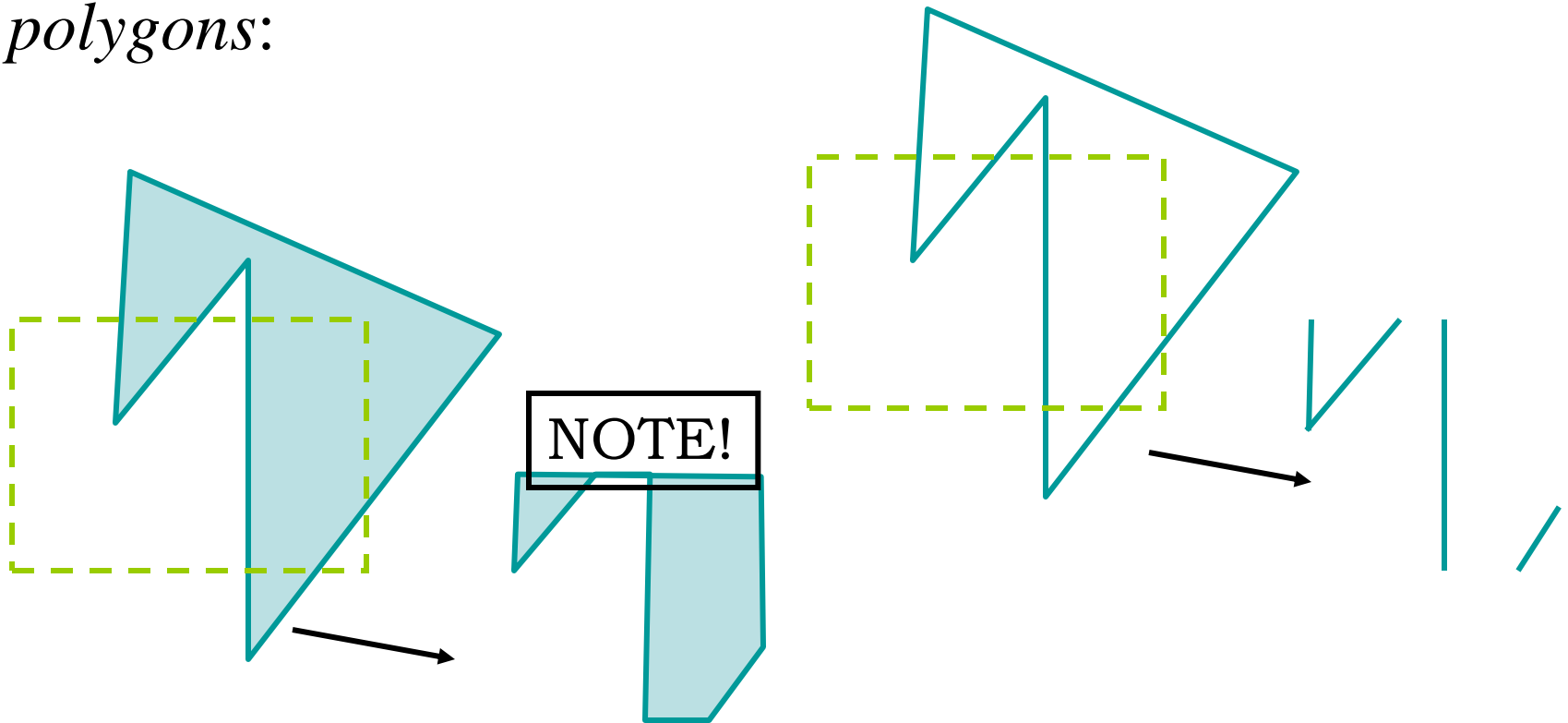
- Integer Version
- Fast as Division by 2 can be performed by simple shift right operation
- For $N \times N$ max dimension of line number of subdivisions required $\log_2 N$.
- Thus a 1024×1024 raster display require just 10 subdivisions.....

2D Clipping

1. Introduction
2. Point Clipping
3. Line Clipping
- 4. Polygon / Area Clipping**
5. Text Clipping

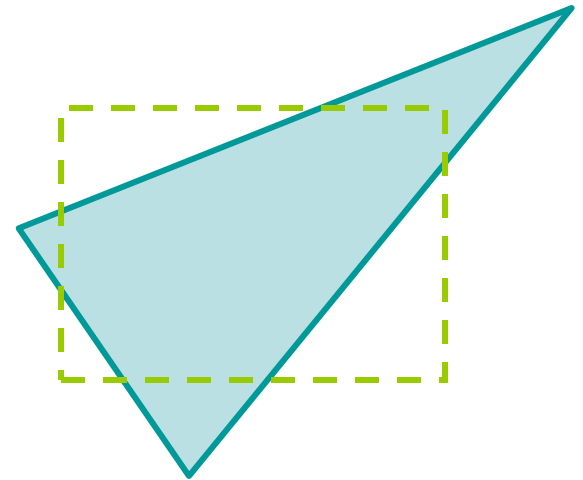
Polygon Clipping

- Note the difference between clipping *lines* and *polygons*:



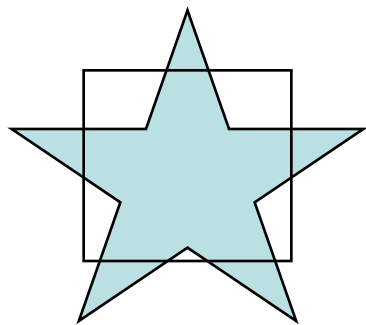
Polygon Clipping

- Some difficulties:
 - Maintaining correct inside/outside
 - Variable number of vertices
 - Handle screen corners correctly

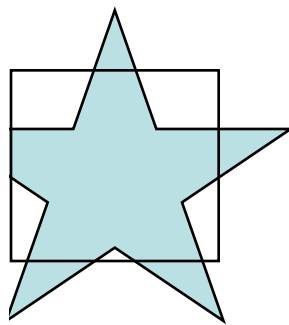


Sutherland-Hodgman Area Clipping

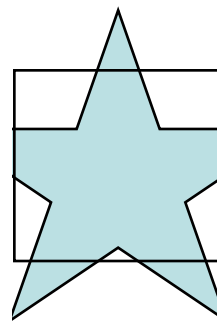
- A technique for clipping areas developed by Sutherland & Hodgman
- Put simply the polygon is clipped by comparing it against each boundary in turn



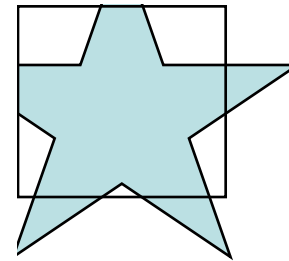
Original Area



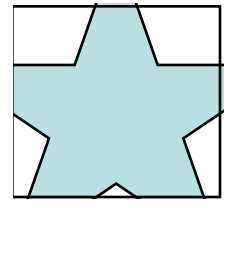
Clip Left



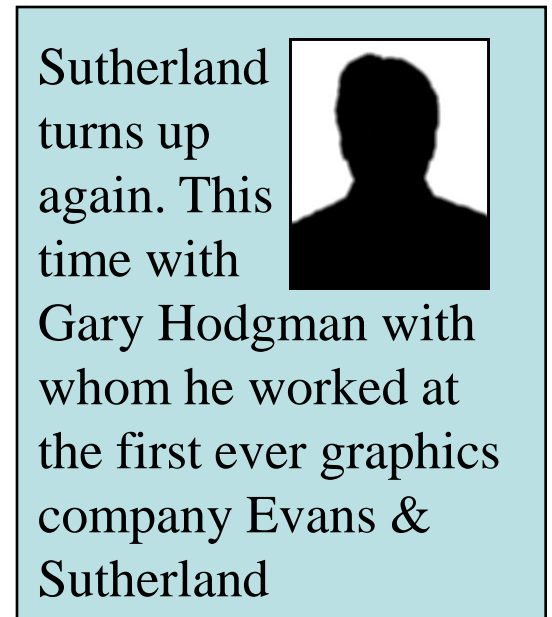
Clip Right



Clip Top



Clip Bottom



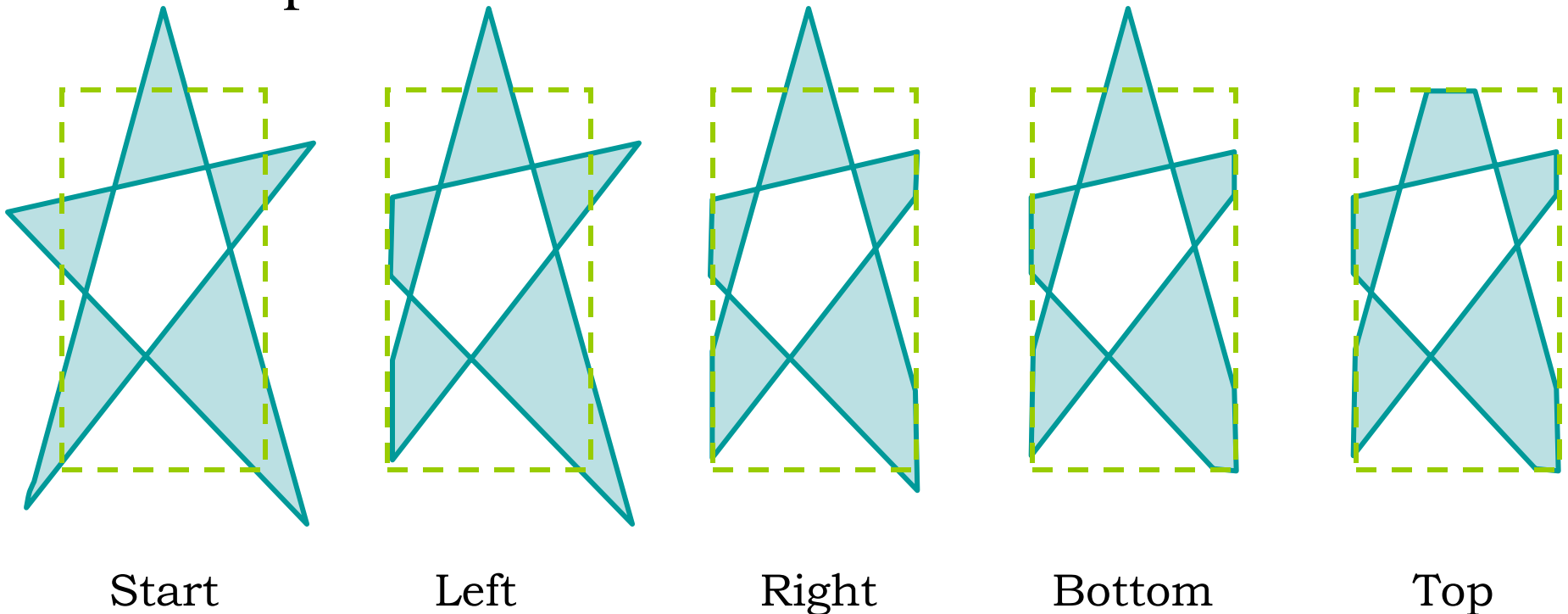
Sutherland-Hodgeman Polygon Clipping

1. Basic Concept:

- Simplify via separation
- Clip whole polygon against one edge
 - Repeat with output for other 3 edges
 - Similar for 3D
- You can create intermediate vertices that get thrown out

Sutherland-Hodgeman Polygon Clipping

- Example

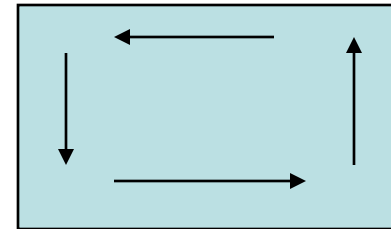


Note that the point one of the points added when clipping on the right gets removed when we clip with bottom

Sutherland-Hodgeman Polygon Clipping

2. Algorithm:

Let (P_1, P_2, \dots, P_N) be the vertex list of the Polygon to be clipped and E be the edge of *positively oriented, convex clipping window*.



We clip each edge of the polygon in turn against each window edge E , forming a new polygon whose vertices are determined as follows:

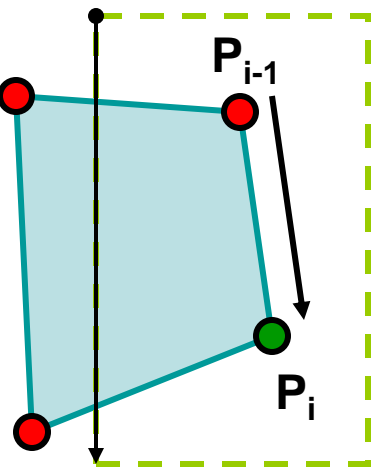
Sutherland-Hodgeman Polygon Clipping

Four cases

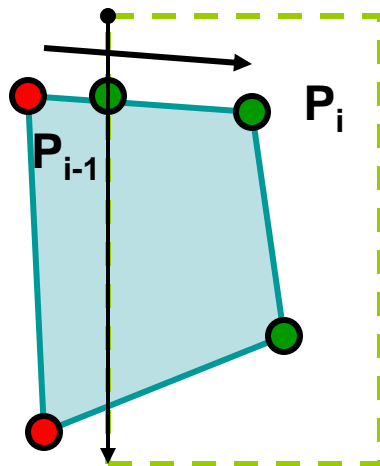
1. **Inside:** If both P_{i-1} and P_i are to the left of window edge vertex then P_i is placed on the output vertex list.
2. **Entering:** If P_{i-1} is to the right of window edge and P_i is to the left of window edge vertex then intersection (I) of $P_{i-1} P_i$ with edge E and P_i are placed on the output vertex list.
3. **Leaving:** If P_{i-1} is to the left of window edge and P_i is to the right of window edge vertex then only intersection (I) of $P_{i-1} P_i$ with edge E is placed on the output vertex list.
4. **Outside:** If both P_{i-1} and P_i are to the right of window edge nothing is placed on the output vertex list.

Sutherland-Hodgeman Polygon Clipping

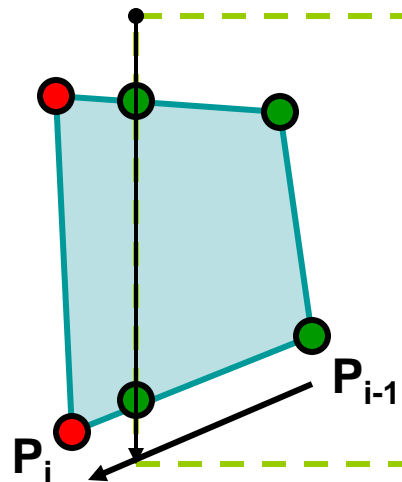
Creating New Vertex List



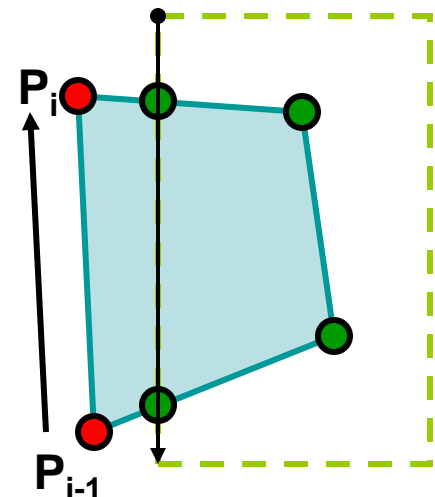
in \rightarrow in
save ending vert
Inside
(1 output)



out \rightarrow in
save new clip vert
and ending vert
Entering
(2 outputs)



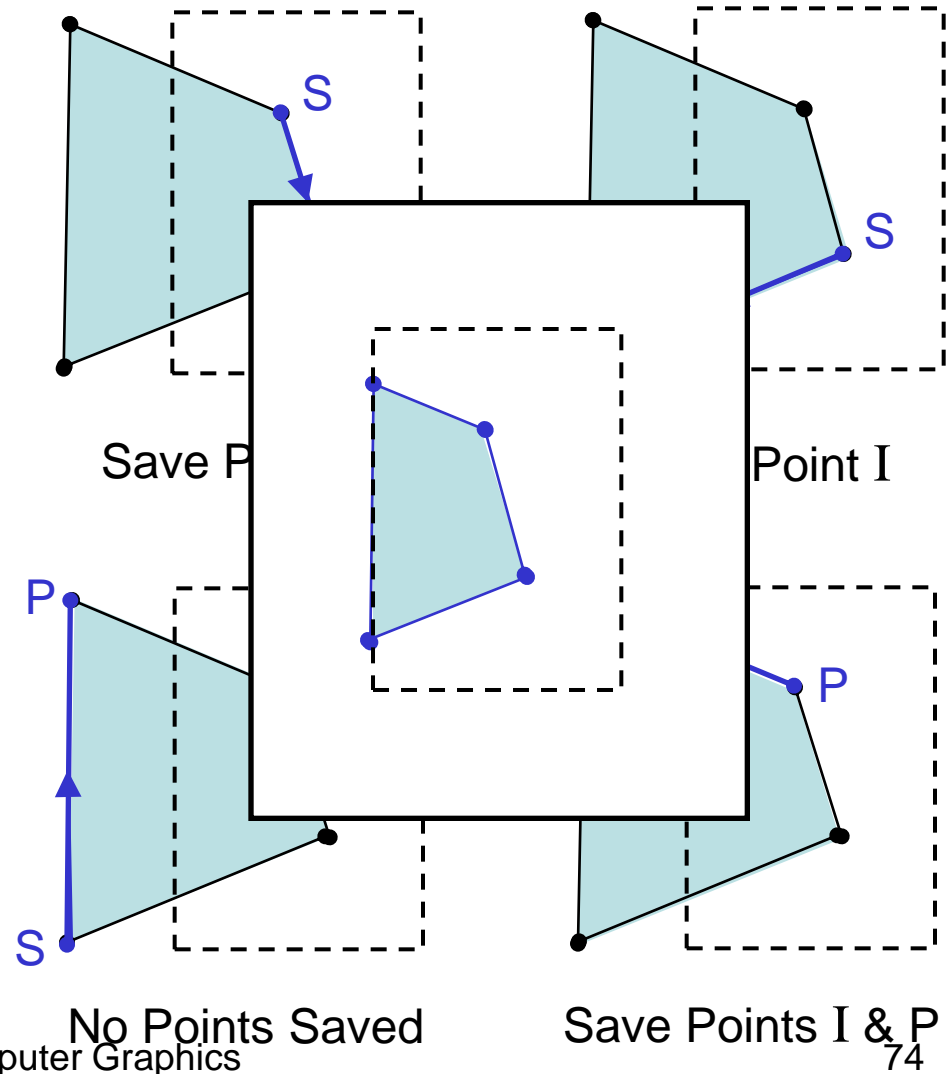
in \rightarrow out
save new clip vert
Leaving
(1 output)



out \rightarrow out
save nothing
Outside
(0 output)

Sutherland-Hodgman Polygon Clipping

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



Sutherland-Hodgeman Polygon Clipping

Inside/Outside Test:

Let $P(x,y)$ be the polygon vertex which is to be tested against edge E defined from $A(x_1, y_1)$ to $B(x_2, y_2)$. Point P is to be said to the left (inside) of E or AB iff

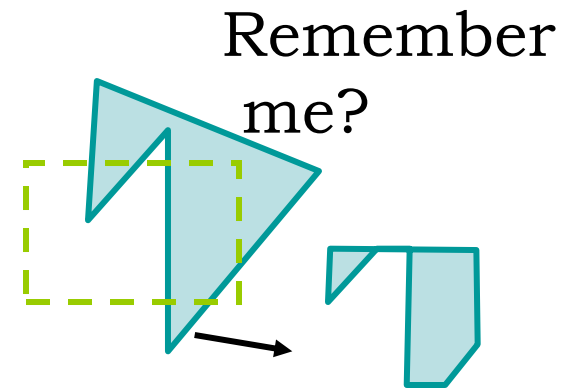
$$\frac{y - y_1}{y_2 - y_1} - \frac{x - x_1}{x_2 - x_1} > 0$$

$$\text{or } C = (x_2 - x_1)(y - y_1) - (y_2 - y_1)(x - x_1) > 0$$

otherwise it is said to be the right/Outside of edge E

Weiler-Atherton Polygon Clipping

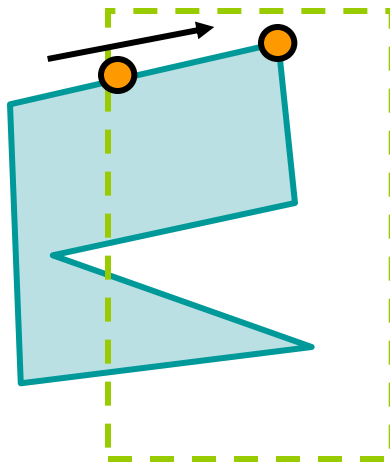
- Problem with Sutherland-Hodgeman:
 - Concavities can end up linked



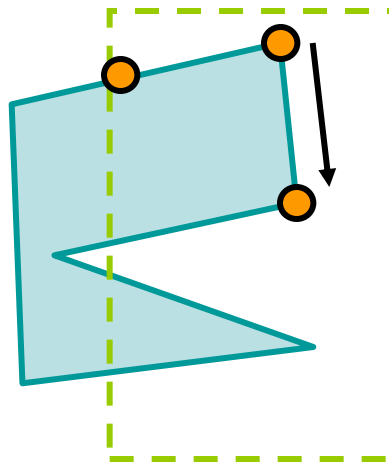
- Weiler-Atherton creates separate polygons in such cases

Weiler-Atherton Polygon Clipping

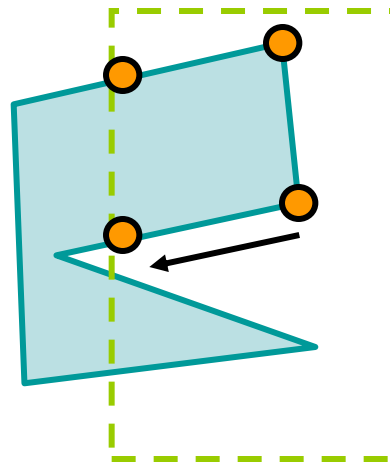
- Example



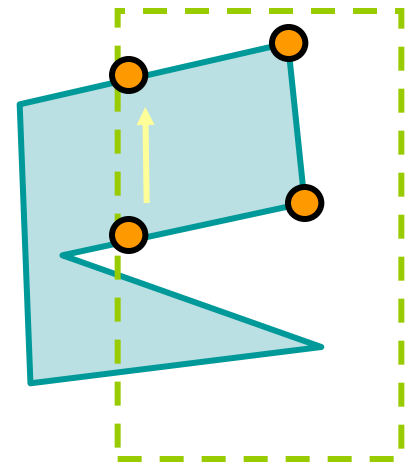
add clip pt.
and end pt.



add end pt.



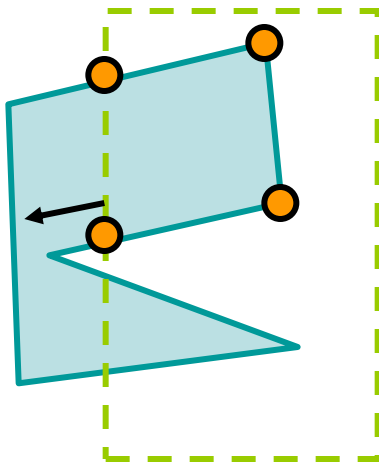
add clip pt.
cache old dir.



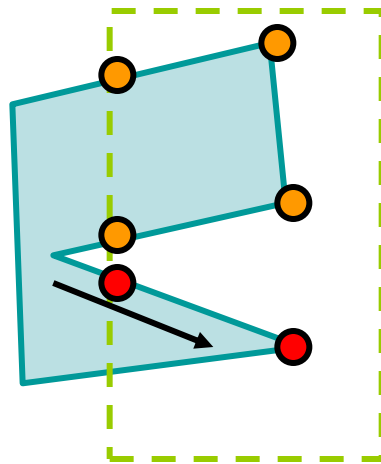
follow clip edge until
a) new crossing
found
b) reach pt. already
added

Weiler-Atherton Polygon Clipping

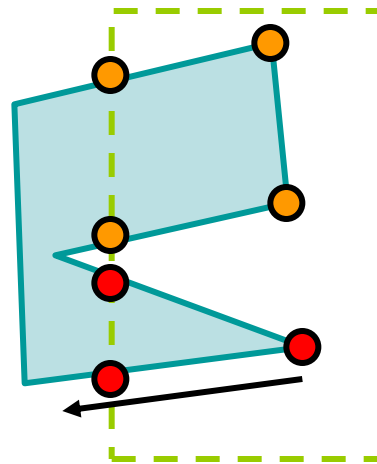
- Example (cont)



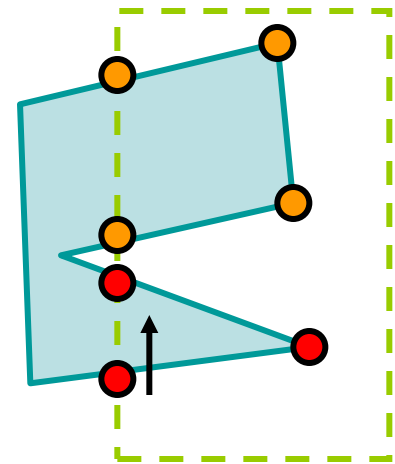
continue from
cached location



add clip pt.
and end pt.



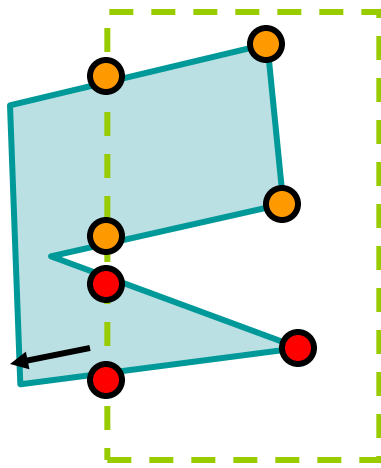
add clip pt.
cache dir.



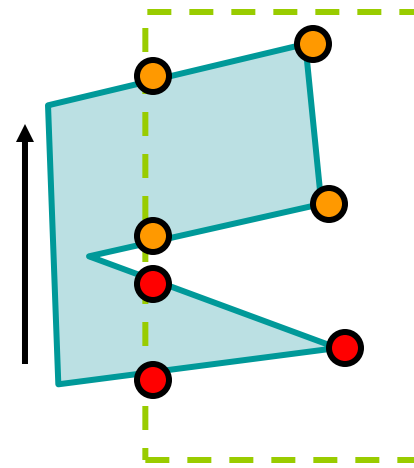
follow clip edge until
a) new crossing
found
b) reach pt. already
added

Weiler-Atherton Polygon Clipping

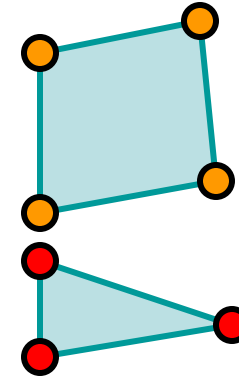
- Example (concluded)



continue from
cached location



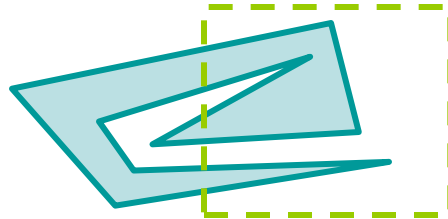
nothing added
finished



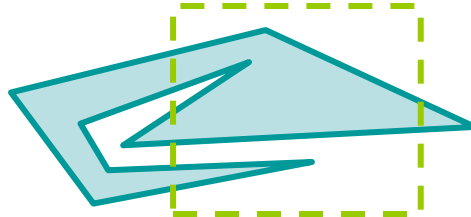
Final result:
Two *unconnected*
polygons

Weiler-Atherton Polygon Clipping

- Difficulties:
 - What if the polygon re-crosses edge?



- How many “cached” crosses?



- Your geometry step must be able to *create* new polygons
 - Instead of 1-in-1-out