



FLAME
UNIVERSITY

EVERLASTING
learning

FUNDAMENTALS OF COMPUTER GRAPHICS (CSIT304)

2D VIEWING AND CLIPPING

CHIRANJOY CHATTOPADHYAY

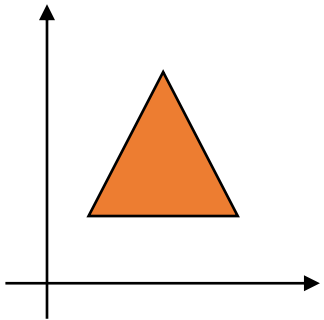
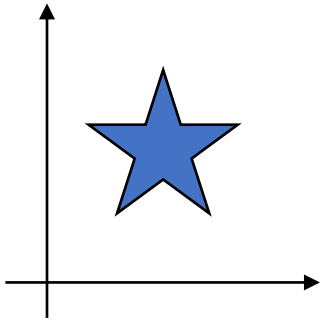
Associate Professor,
FLAME School of Computation and Data Science

2D VIEWING

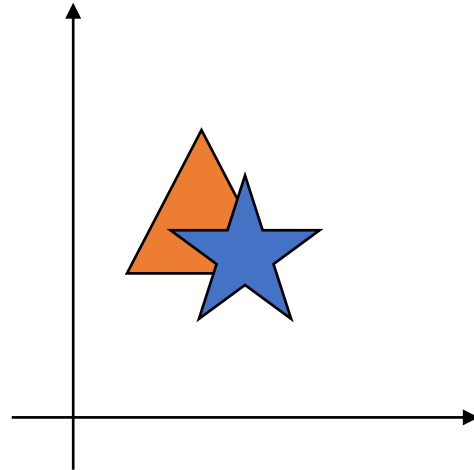
INTRODUCTION

- We see in real life through a small window or the viewfinder of a camera.
- A computer-generated image often depicts a partial view of a large scene.
- Objects are placed into a world coordinate system (WCS).
- A rectangular window with its **edges parallel to the axes of the WCS** is used to select the portion of the scene for which an image is to be generated.
- A Sub-area of the total pixel area.

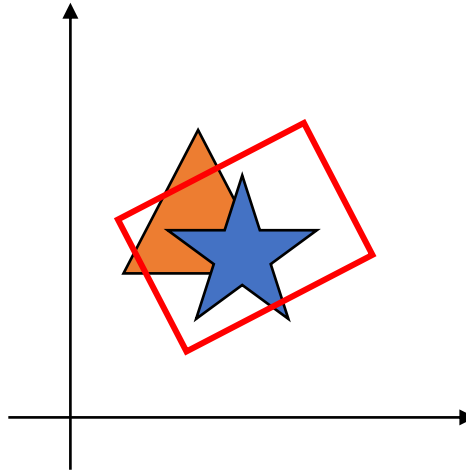
ILLUSTRATION



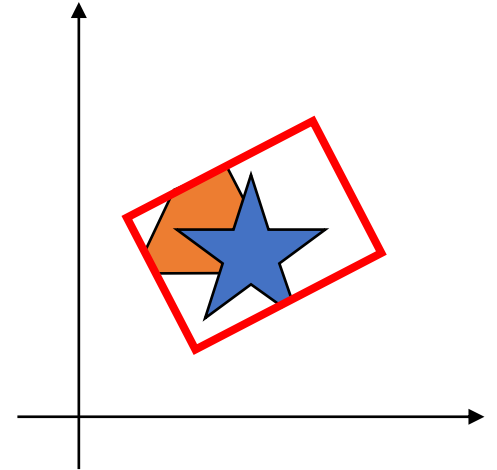
**Modeling
Coordinates**



**World
Coordinates**



**View
Coordinates**



**Device
Coordinates**

WINDOW

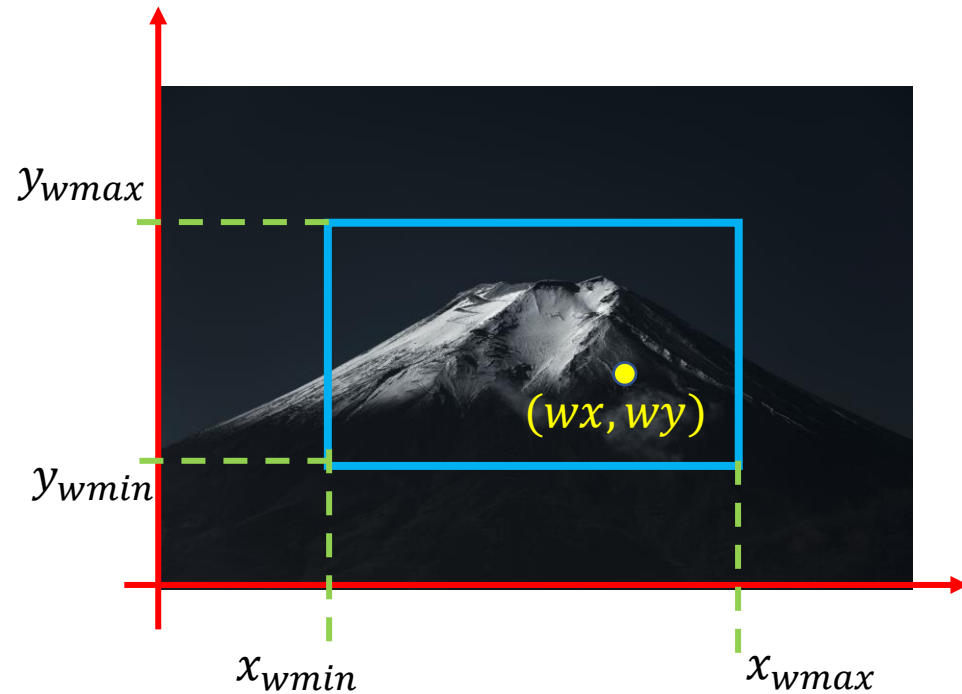
- A world-coordinate area selected for display is called a window or clipping window.
- The section of the 2D scene that is selected for viewing.
- The window defines what is to be viewed.

VIEWPORT

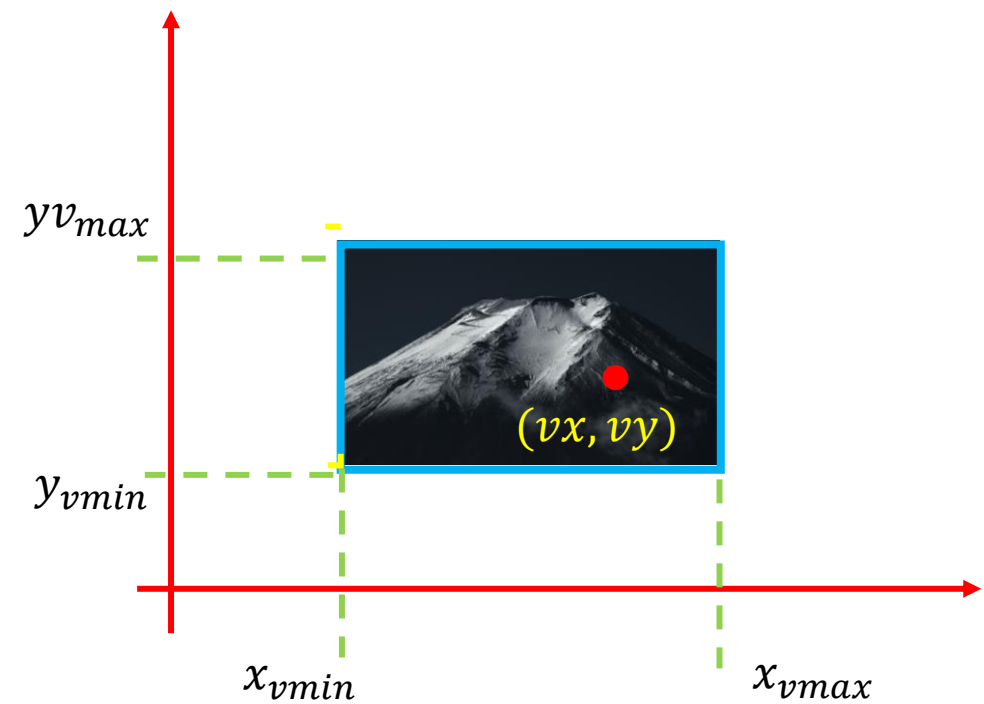
- An area on a display device to which a window is mapped is called a viewport.
- The viewport indicates where on an output device selected part will be displayed.

ILLUSTRATION

How to convert / transform a point in the world coordinate (wx, wy) to a point in the view coordinate (vx, vy) ?




**World
Coordinates**



**View-port
Coordinates**

WINDOW TO VIEWPORT TRANSFORMATION

To maintain the same relative placement of the point in the viewport as in the window

$$\frac{wx - xw_{min}}{xw_{max} - xw_{min}} = \frac{vx - xv_{min}}{xv_{max} - xv_{min}}$$
$$\frac{yx - yw_{min}}{yw_{max} - yw_{min}} = \frac{vy - yv_{min}}{yv_{max} - yv_{min}}$$

$$vx = ?$$
$$vy = ?$$

$$vx = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}} (wx - xw_{min}) + xv_{min}$$

$$vy = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}} (yx - yw_{min}) + yv_{min}$$

WINDOW TO VIEWPORT TRANSFORMATION

- Translate the window to the origin $[(wx - xw_{min}), (wy - yw_{min})]$
- Scale it to the size of the viewport

$$\left[\frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}, \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}} \right]$$

- Translate scaled window to the position of the viewport $[+xv_{min}, +yv_{min}]$

TRANSFORMATION MATRIX

Window is parallel to the coordinate axis. What if it is not?

$$\begin{bmatrix} vx \\ vy \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & xv_{min} \\ 0 & 1 & yv_{min} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}} & 0 & 0 \\ 0 & \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -xw_{min} \\ 0 & 1 & -yw_{min} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} wx \\ wy \\ 1 \end{bmatrix}$$

V

T

S

T

W

2D CLIPPING

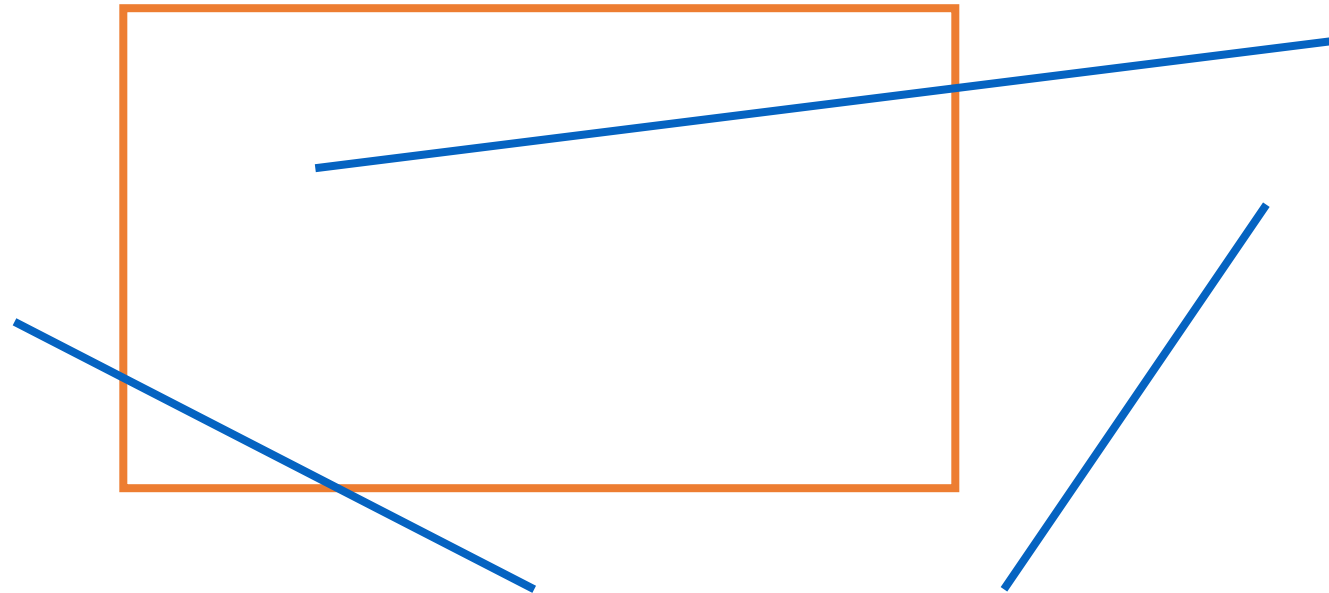
OBJECTIVE

After completing this lecture the students will be able to

- Explain and solve numerical problems of 2D Clipping of
 - Point
 - Line
 - Polygon

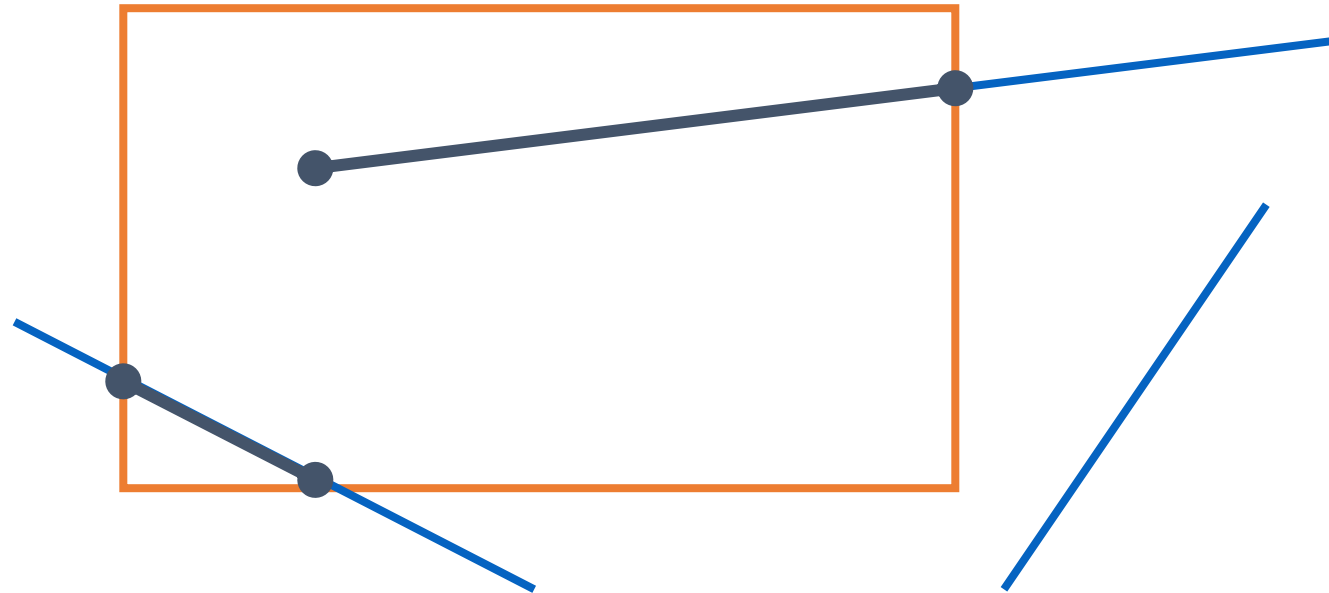
CLIPPING

- We've been assuming that all primitives (lines, triangles, polygons) lie entirely within the *viewport*
- In general, this assumption will not hold



CLIPPING

- Analytically calculating the portions of primitives within the viewport



WHY CLIP?

- Bad idea to rasterize outside of framebuffer bounds
- Also, don't waste time scan converting pixels outside window

CLIPPING

- The naïve approach to clipping lines:

for each line segment

for each edge of viewport

find intersection points

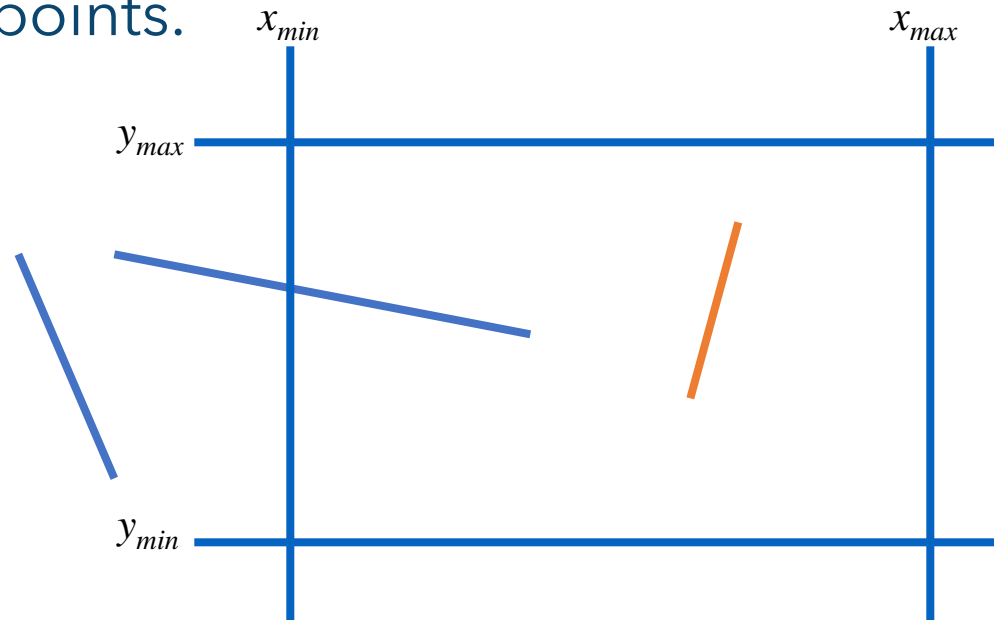
pick “nearest” point

if anything is left, draw it

- *What do we mean by “nearest”?*

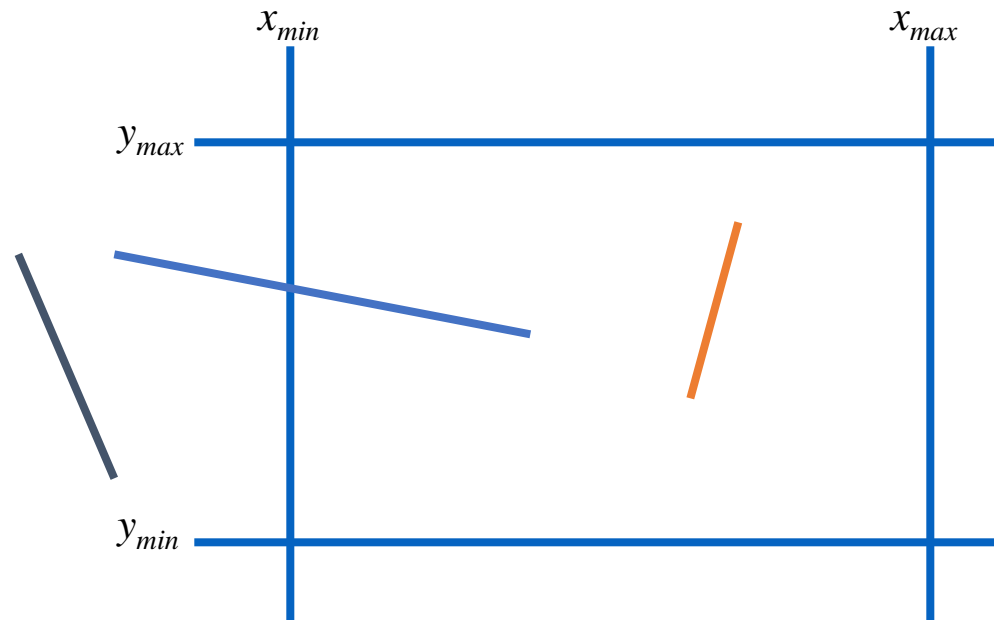
TRIVIAL ACCEPTS

- Big optimization: trivial accept/ rejects
- *How can we quickly determine whether a line segment is entirely inside the viewport?*
- A: test both endpoints.



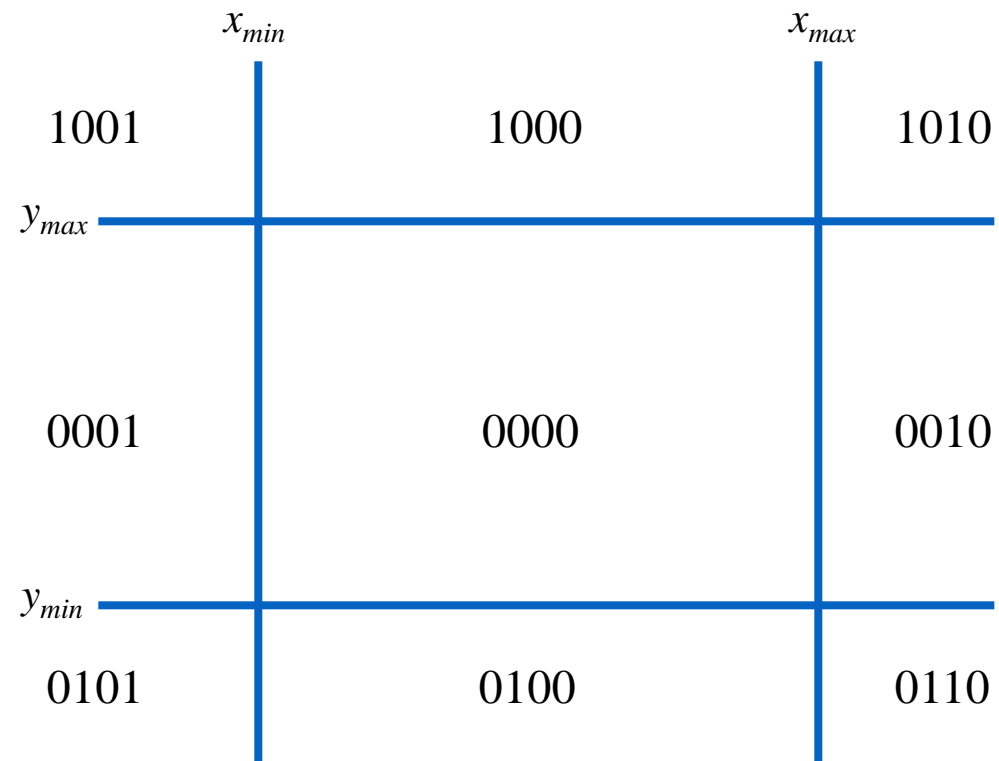
TRIVIAL REJECTS

- *How can we know a line is outside viewport?*
- A: if both endpoints on wrong side of same edge, can trivially reject line



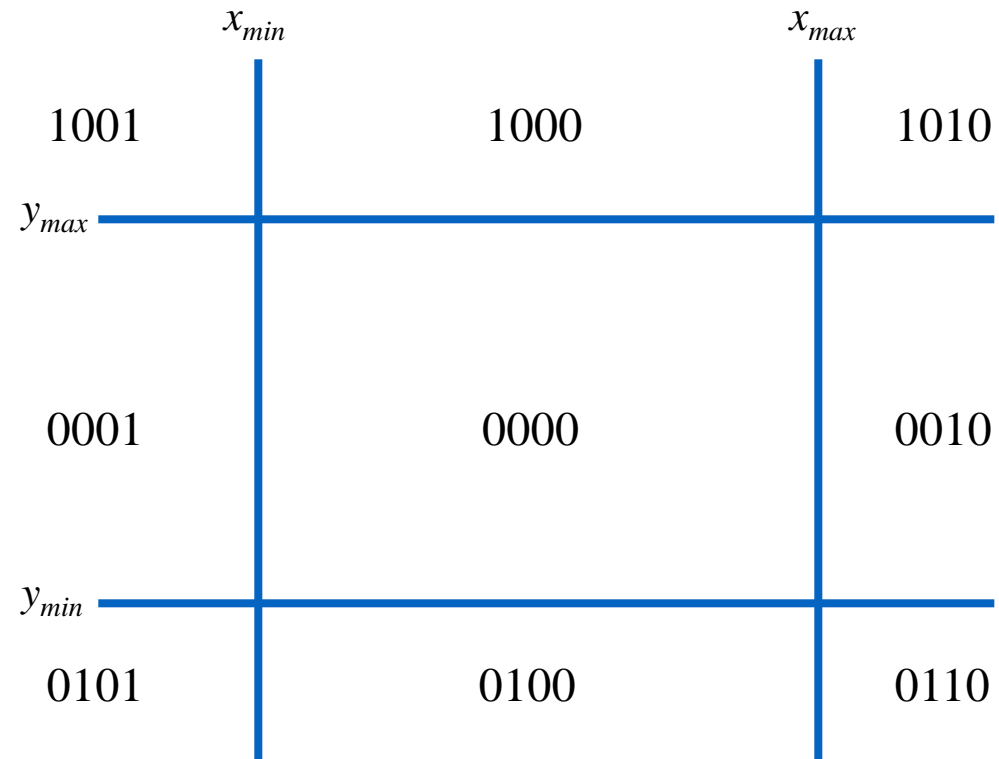
COHEN-SUTHERLAND LINE CLIPPING

- Divide view plane into regions defined by viewport edges
- Assign each region a 4-bit **outcode**:



COHEN-SUTHERLAND LINE CLIPPING

- *To what do we assign outcodes?*
- *How do we set the bits in the outcode?*
- *How do you suppose we use them?*

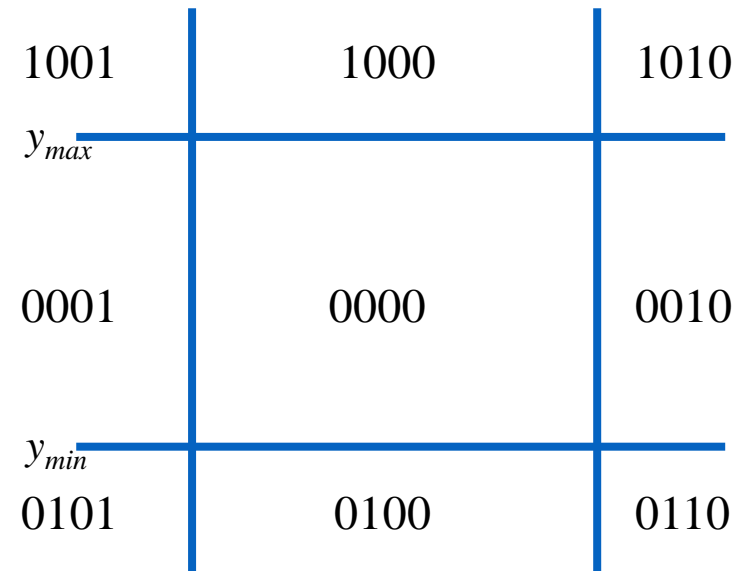


COHEN-SUTHERLAND LINE CLIPPING

- Set bits with simple tests

$x > x_{\max}$ $y < y_{\min}$ etc.

- Assign an outcode to each vertex of line
 - If both outcodes = 0, trivial accept
 - bitwise AND vertex outcodes together
 - If result $\neq 0$, trivial reject



DEFINING OUTCODES

- For each endpoint, define an outcode (TBRL) **$b_0b_1b_2b_3$**

$b_0 = 1$ if $y > y_{\max}$, 0 otherwise

$b_1 = 1$ if $y < y_{\min}$, 0 otherwise

$b_2 = 1$ if $x > x_{\max}$, 0 otherwise

$b_3 = 1$ if $x < x_{\min}$, 0 otherwise

1001	1000	1010	$y = y_{\max}$
0001	0000	0010	
0101	0100	0110	$y = y_{\min}$
$x = x_{\min}$		$x = x_{\max}$	

Alternatively, $b_0 = \text{sign}(y - y_{\max})$, ...

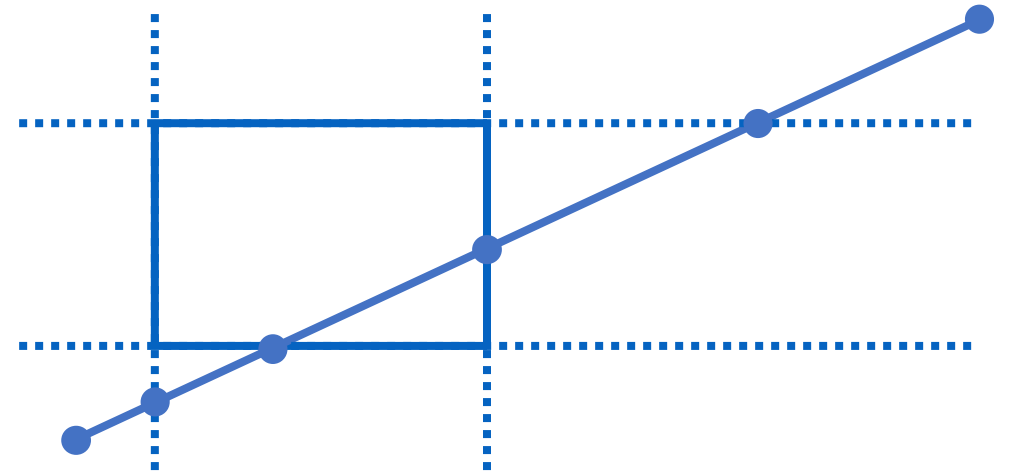
- Outcodes divide space into 9 regions
- Computation of outcode requires at most 4 subtractions

COHEN-SUTHERLAND LINE CLIPPING

- If line cannot be trivially accepted or rejected, subdivide so that one or both segments can be discarded
- Pick an edge that the line crosses (**how?**)
- Intersect line with edge (**how?**)
- Discard portion on wrong side of edge and assign **outcode** to **new vertex**
- Apply trivial accept/ reject tests; repeat if necessary

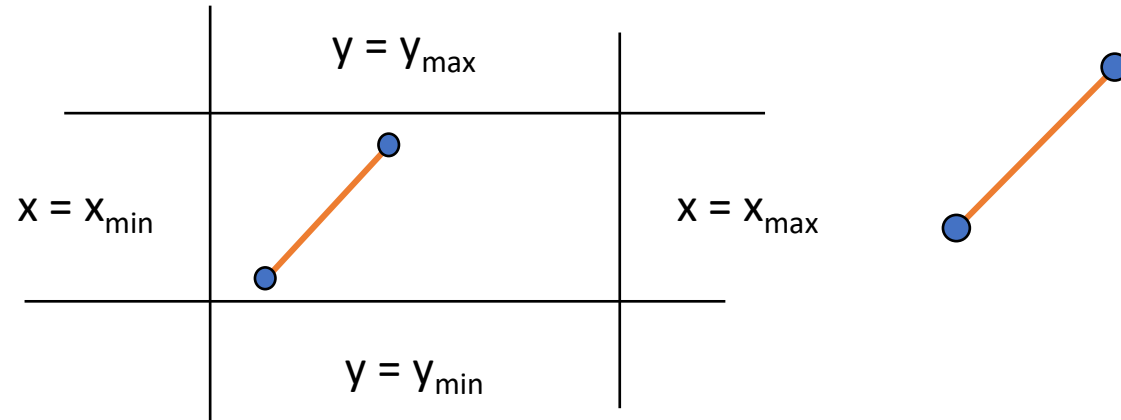
COHEN-SUTHERLAND LINE CLIPPING

- **Outcode tests** and line-edge intersects are quite fast
- But some lines require multiple iterations:
 - Clip top
 - Clip left
 - Clip bottom
 - Clip right
- Fundamentally more efficient algorithms:
 - Cyrus-Beck uses parametric lines
 - Liang-Barsky optimizes this for upright volumes



THE CASES

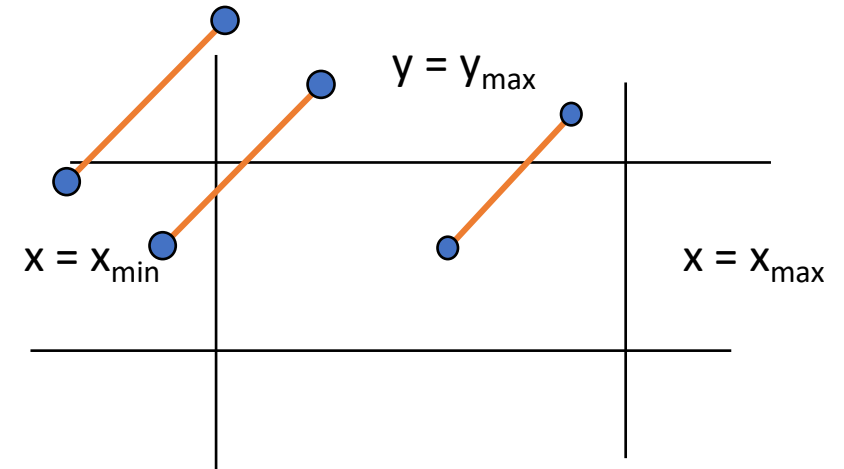
- Case 1: both endpoints of line segment inside all four lines
 - Draw (accept) line segment as is



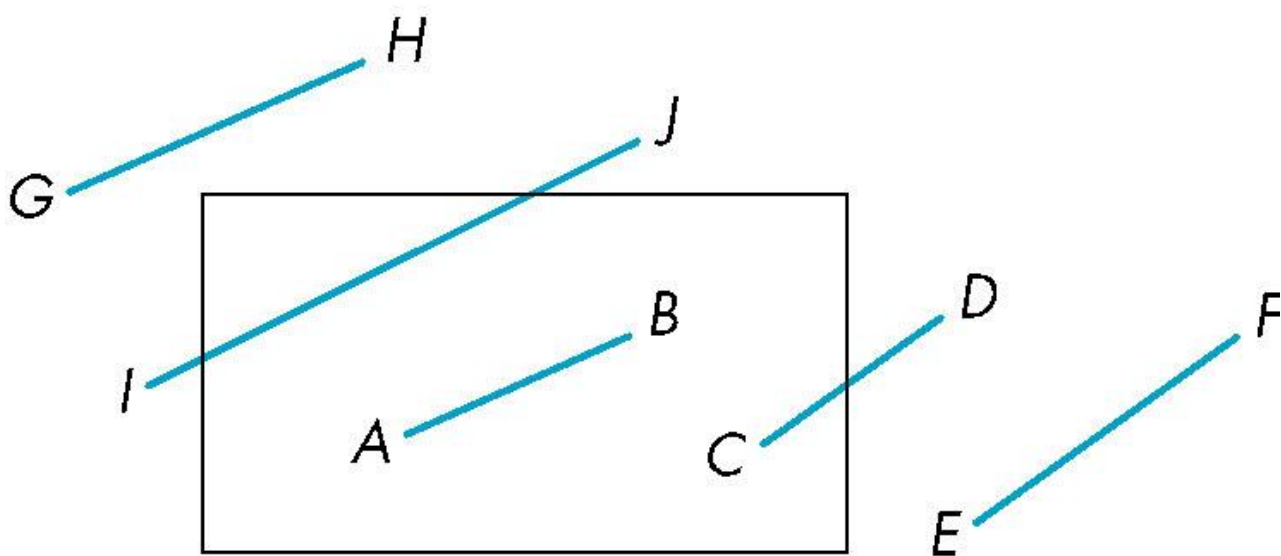
- Case 2: both endpoints outside all lines and on same side of a line
 - Discard (reject) the line segment

THE CASES

- Case 3: One endpoint inside, one outside
 - Must do at least one intersection
- Case 4: Both outside
 - May have part inside
 - Must do at least one intersection



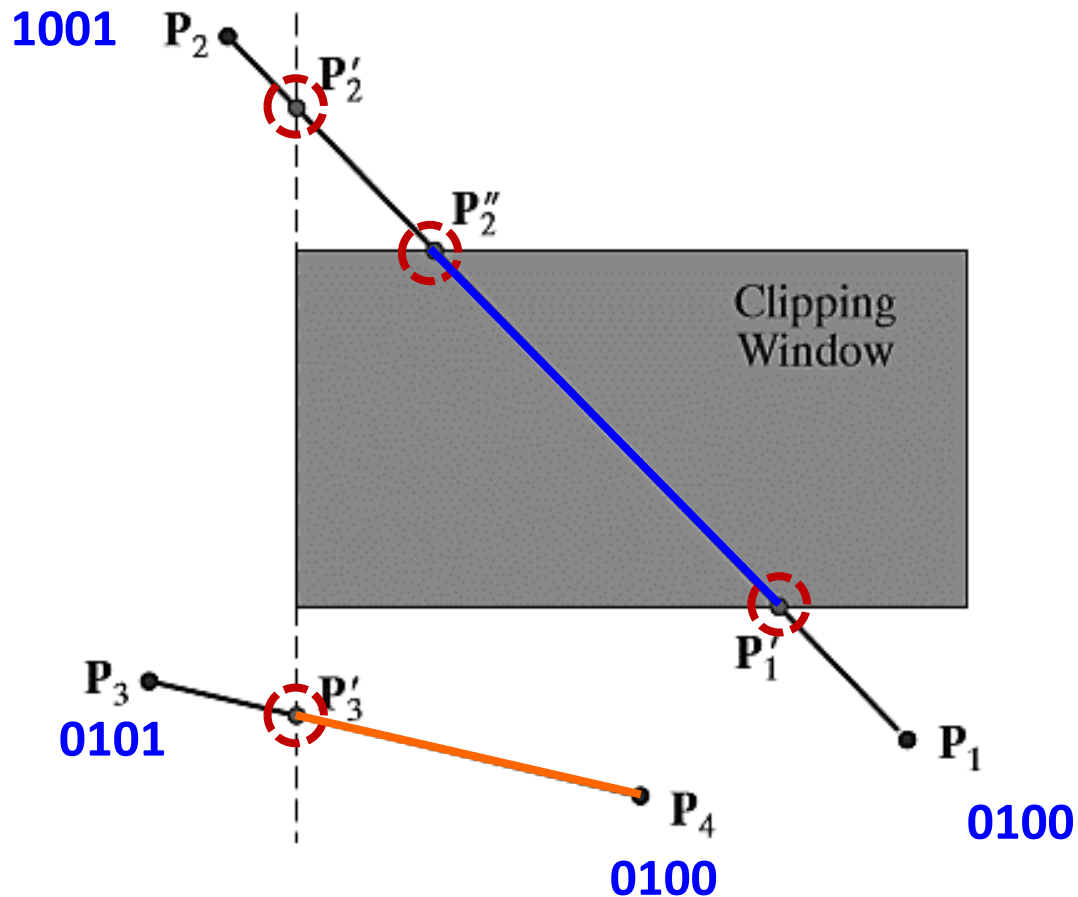
EXAMPLE



Point	T	B	R	L
A	0	0	0	0
B	0	0	0	0
C	0	0	0	0
D	0	0	1	0
E	0	0	1	0
F	0	0	1	0
G	0	0	0	1
H	1	0	0	0
I	0	0	0	1
J	1	0	0	0

EXAMPLE (INTERSECTION)

- Select the extended boundary line
- Intersect with the candidate boundaries



If $b_0 == 1$
Intersect with $y=y_{\max}$

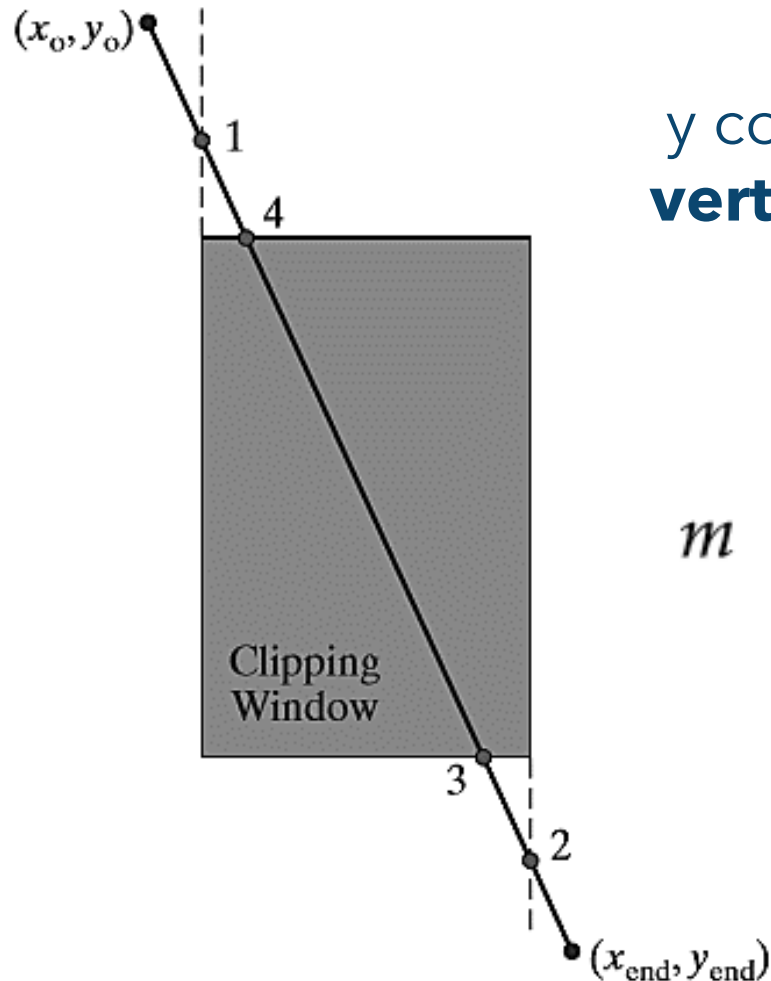
If $b_1 == 1$
Intersect with $y=y_{\min}$

If $b_3 == 1$
Intersect with $x=x_{\max}$

If $b_4 == 1$
Intersect with $x=x_{\min}$

MORE EXAMPLE

Use the slope intercept form of the line equation



y coordinate of the intersection point with a **vertical clipping** border line

$$y = y_0 + m(x - x_0)$$

$$m = (y_{end} - y_0) / (x_{end} - x_0)$$

$$x = xW_{min} \text{ OR } xW_{max}$$

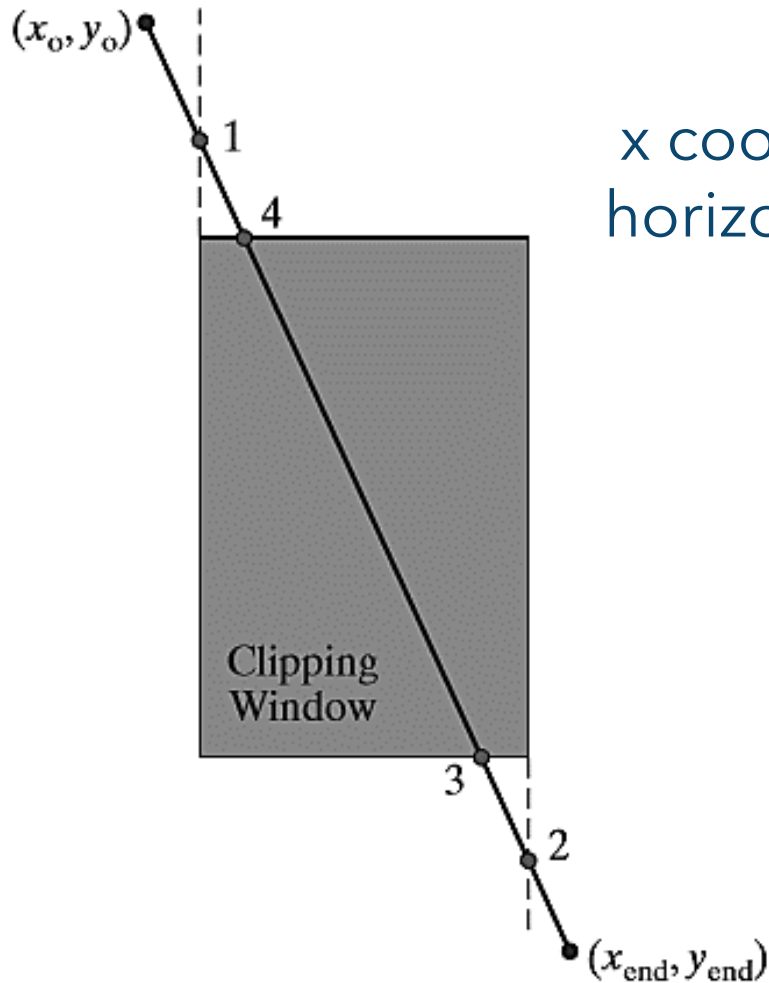
MORE EXAMPLE

Use the slope intercept form of the line equation

x coordinate of the intersection point with a horizontal clipping border line

$$x = x_0 + \frac{y - y_0}{m}$$

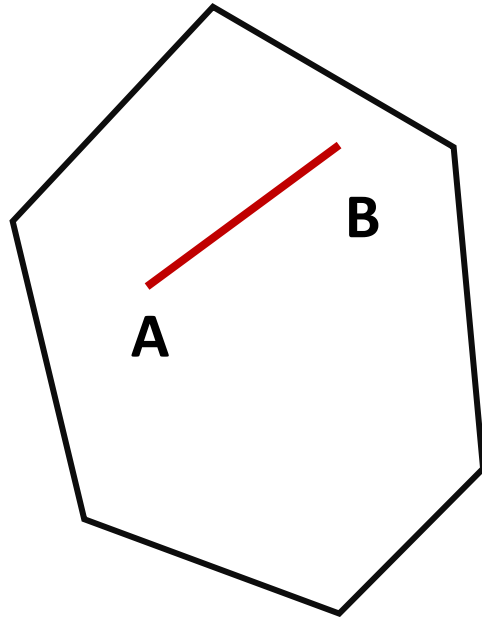
$$y = yw_{min} \text{ or } yw_{max}$$



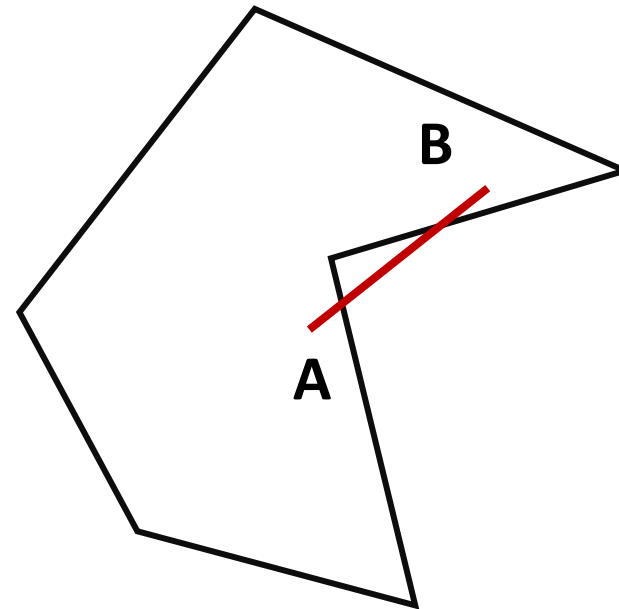
CLIPPING POLYGONS

- We know how to clip a single line segment
 - How about a polygon in 2D?
 - How about in 3D?
- Clipping polygons is more complex than clipping the individual lines
 - Input: polygon
 - Output: polygon, or nothing
- ***When can we trivially accept/reject a polygon as opposed to the line segments that make up the polygon?***

CONVEX POLYGON CLIPPING WINDOW



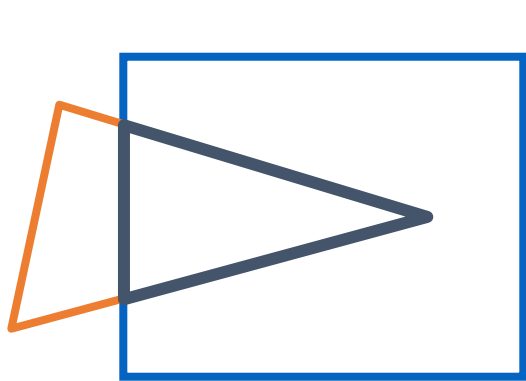
Convex



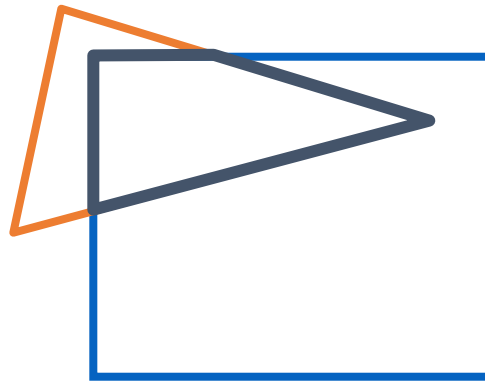
Non-Convex

WHY IS CLIPPING HARD?

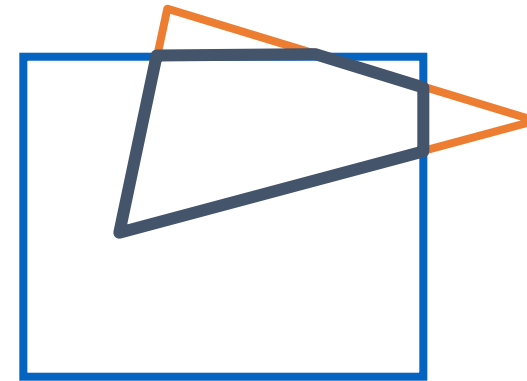
- ***What happens to a triangle during clipping?***
- Possible outcomes:



Triangle → triangle



Triangle → quad

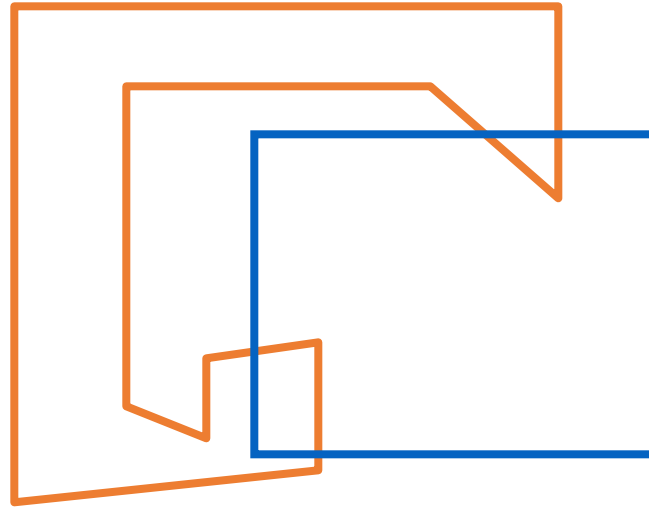


Triangle → 5-gon

How many sides can a clipped triangle have?

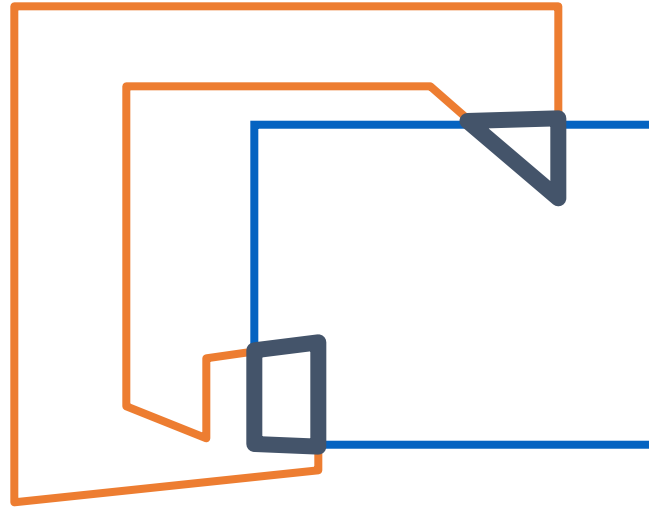
WHY IS CLIPPING HARD?

- A really tough case:



WHY IS CLIPPING HARD?

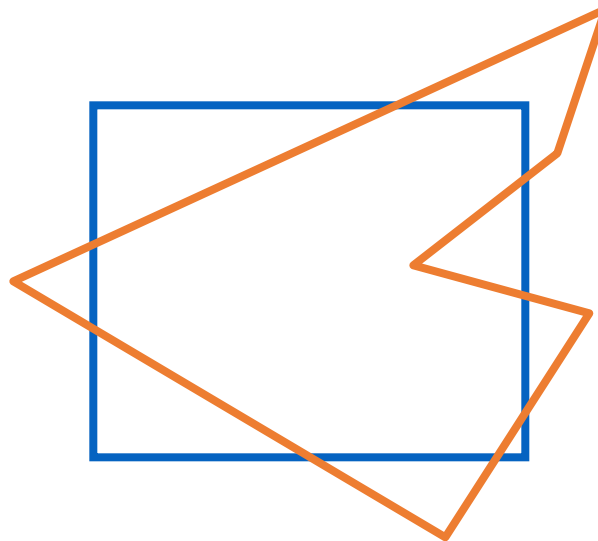
- A really tough case:



concave polygon → multiple polygons

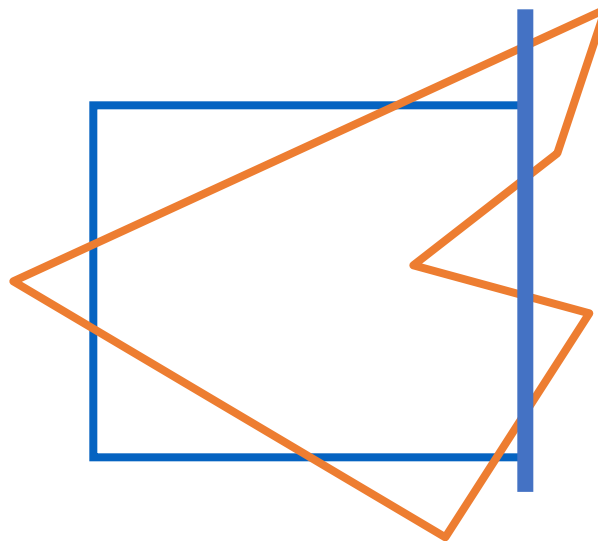
SUTHERLAND-HODGMAN CLIPPING

- Basic idea:
 - Consider each edge of the viewport individually
 - Clip the polygon against the edge equation
 - After doing all planes, the polygon is fully clipped



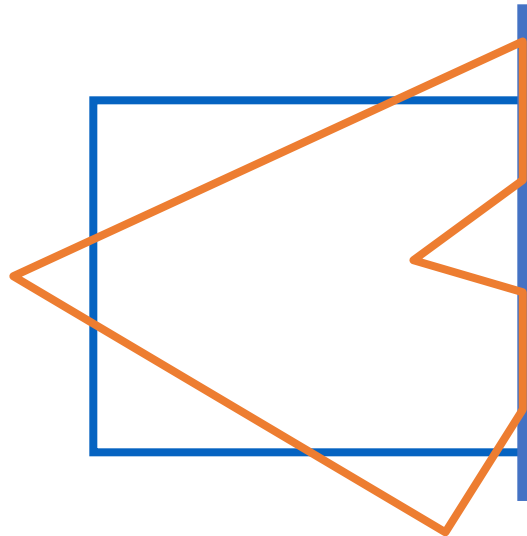
SUTHERLAND-HODGMAN CLIPPING

- Basic idea:
 - Consider each edge of the viewport individually
 - Clip the polygon against the edge equation
 - After doing all planes, the polygon is fully clipped



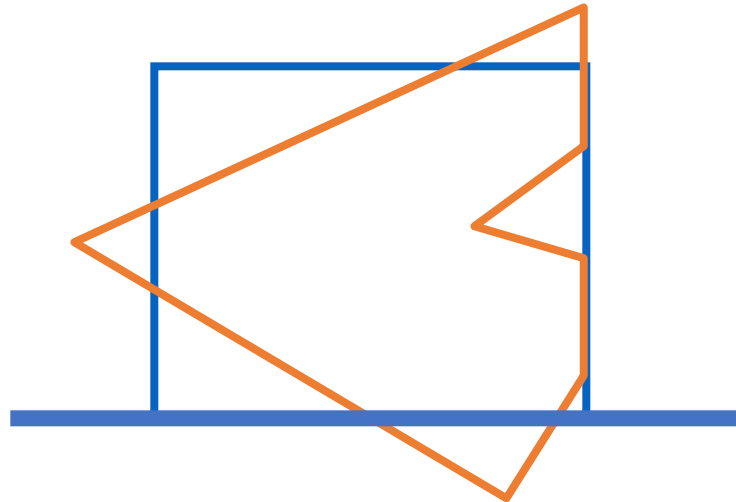
SUTHERLAND-HODGMAN CLIPPING

- Basic idea:
 - Consider each edge of the viewport individually
 - Clip the polygon against the edge equation
 - After doing all planes, the polygon is fully clipped



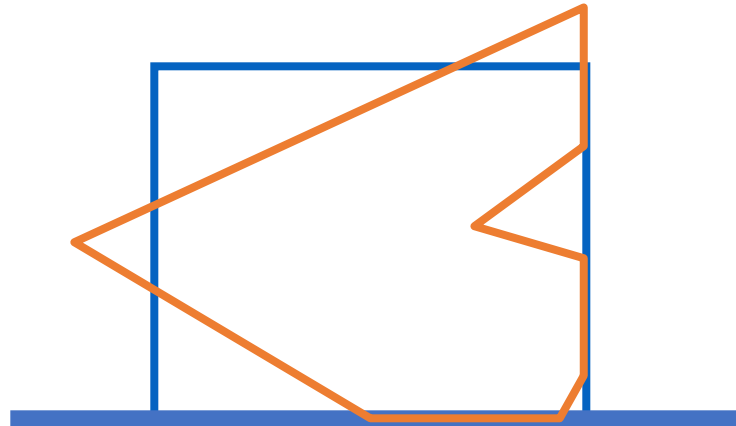
SUTHERLAND-HODGMAN CLIPPING

- Basic idea:
 - Consider each edge of the viewport individually
 - Clip the polygon against the edge equation
 - After doing all planes, the polygon is fully clipped



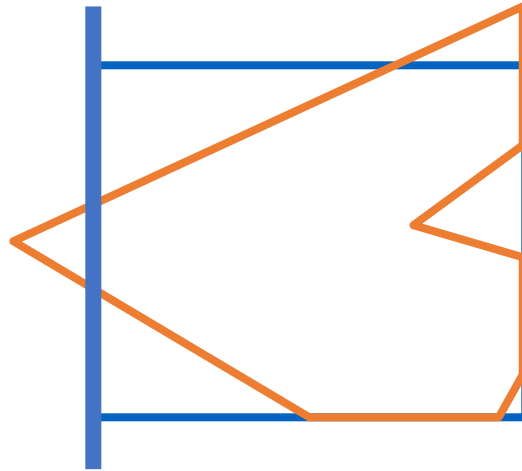
SUTHERLAND-HODGMAN CLIPPING

- Basic idea:
 - Consider each edge of the viewport individually
 - Clip the polygon against the edge equation
 - After doing all planes, the polygon is fully clipped



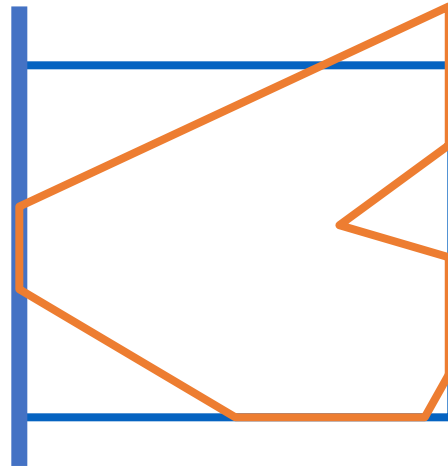
SUTHERLAND-HODGMAN CLIPPING

- Basic idea:
 - Consider each edge of the viewport individually
 - Clip the polygon against the edge equation
 - After doing all planes, the polygon is fully clipped



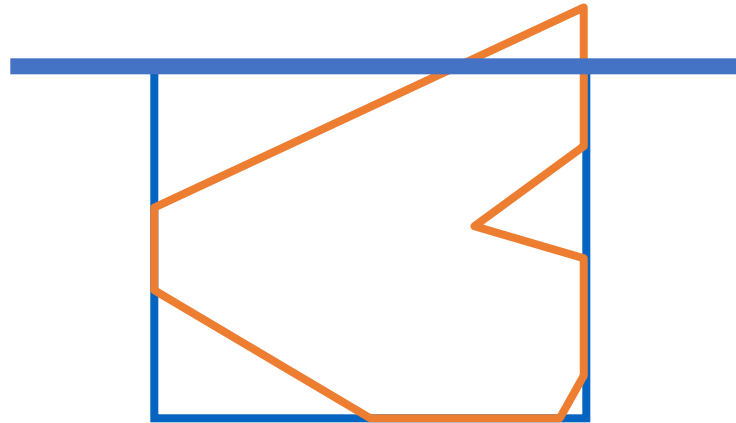
SUTHERLAND-HODGMAN CLIPPING

- Basic idea:
 - Consider each edge of the viewport individually
 - Clip the polygon against the edge equation
 - After doing all planes, the polygon is fully clipped



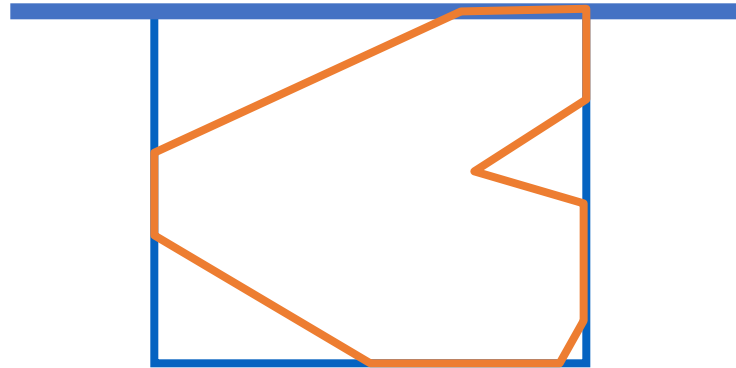
SUTHERLAND-HODGMAN CLIPPING

- Basic idea:
 - Consider each edge of the viewport individually
 - Clip the polygon against the edge equation
 - After doing all planes, the polygon is fully clipped



SUTHERLAND-HODGMAN CLIPPING

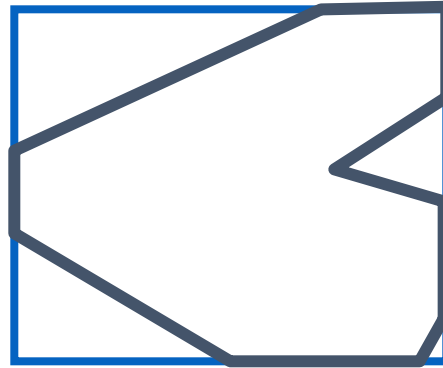
- Basic idea:
 - Consider each edge of the viewport individually
 - Clip the polygon against the edge equation
 - After doing all planes, the polygon is fully clipped



SUTHERLAND-HODGMAN CLIPPING

- Basic idea:
 - Consider each edge of the viewport individually
 - Clip the polygon against the edge equation
 - After doing all planes, the polygon is fully clipped

Will this work for non-rectangular clip regions?



What would 3-D clipping involve?

SUTHERLAND-HODGMAN CLIPPING

- Input/output for algorithm:
 - Input: list of polygon vertices in order
 - Output: list of clipped polygon vertices consisting of old vertices (maybe) and new vertices (maybe)
- Note: this is exactly what we expect from the clipping operation against each edge
- This algorithm generalizes to 3-D

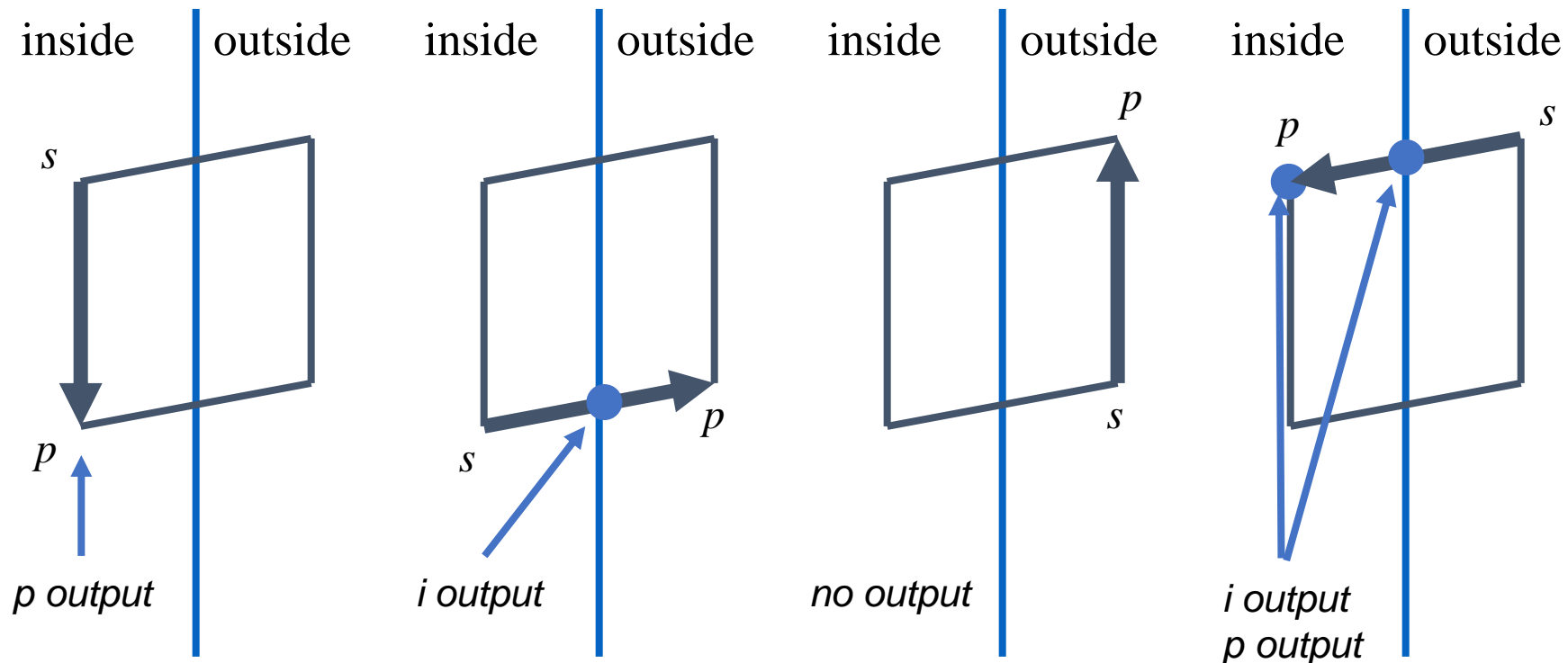
SUTHERLAND-HODGMAN CLIPPING

- We need to be able to create clipped polygons from the original polygons
- Sutherland-Hodgman basic routine:
 - Go around polygon one vertex at a time
 - Current vertex has position ***p***
 - Previous vertex had position ***s***, and it has been added to the output if appropriate

SUTHERLAND-HODGMAN CLIPPING

- Edge from s to p takes one of four cases:

(**Blue** line can be a line or a plane)



SUTHERLAND-HODGMAN CLIPPING

1. s inside plane and p inside plane

- Add p to output
- Note: **s has already been added**

2. s inside plane and p outside plane

- Find intersection point i
- Add i to output

1. s outside plane and p outside plane

- Add nothing

2. s outside plane and p inside plane

- Find intersection point i
- Add i to output, followed by p

PSEUDO-CODE

1. Initialize:

1. Initialize the output polygon with the vertices of the input polygon.

2. Clip Against Each Clipping Window Edge:

1. For each edge of the clipping window (xmin, ymin, xmax, ymax), do the following steps:
 1. Set the input polygon to the current output polygon.
 2. Clear the output polygon.
 3. Loop through each consecutive pair of vertices (v1, v2) in the input polygon.

3. Process Each Vertex:

1. For each vertex v in the input polygon, do the following:
 1. If v is inside the clipping window, add it to the output polygon.
 2. If the previous vertex was outside the clipping window and the current vertex is inside, add the intersection point of the edge and the clipping window to the output polygon.

4. Handle Wraparound:

1. Handle the wraparound case by connecting the last and first vertices of the polygon.

5. Repeat for Each Clipping Window Edge:

1. Repeat steps 2-4 for each edge of the clipping window.

6. Final Result:

1. The final result is the clipped polygon, stored in the output polygon.

ANIMATION





FLAME
UNIVERSITY

EVERLASTING
learning

THANK YOU