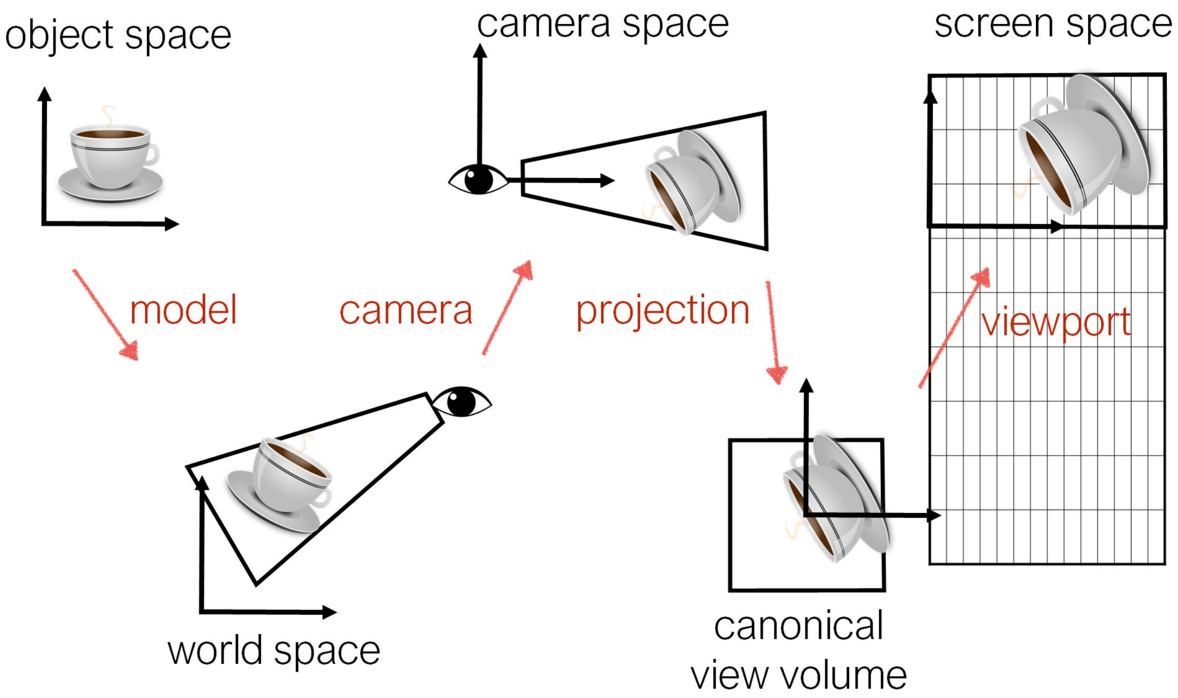
Rendering Pipeline 1

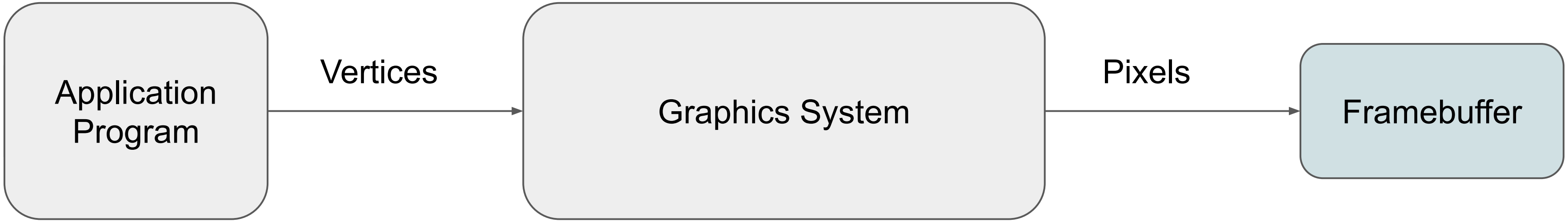
CS 425: Computer Graphics 1

UIC Computer Science

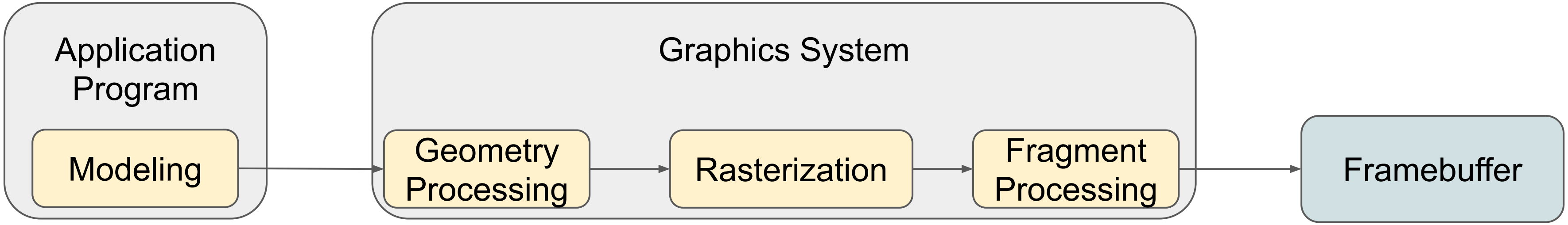
Recap: Transformations



Graphics System



Graphics Pipeline



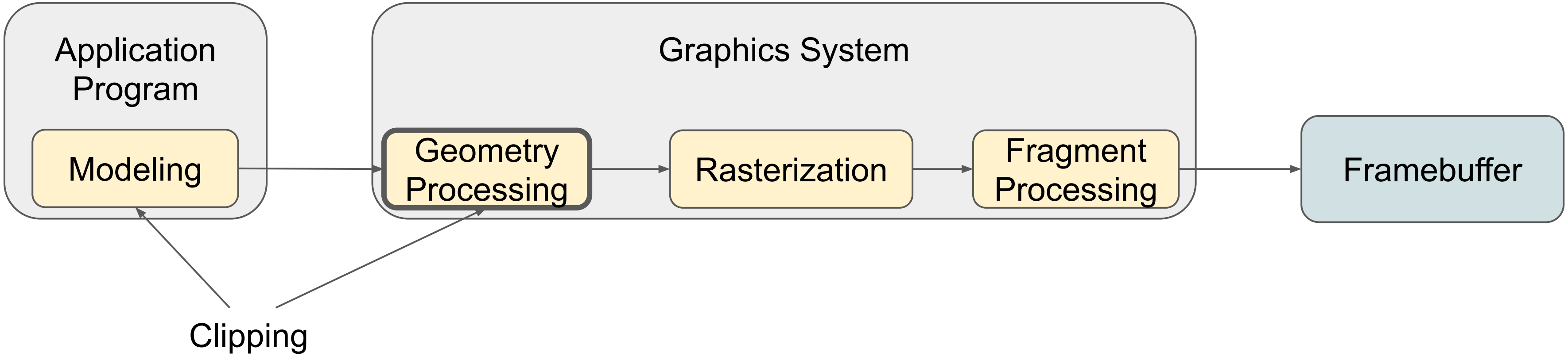
Geometry Processing

- Change of representation (We saw this in the last class)
- Projection
- Primitive Assembly
- Clipping
- Shading

Geometry Processing

- Projection:
  - Transforming to a normalized view volume.
  - View volume will be a cube of size 2, centered at origin.
  - Eliminates dependency on application specifications as well as display device specifications.
- Primitive Assembly:
  - Grouping of vertices to form objects (primitives).
  - Done before clipping.
- Clipping
- Shading
  - Assignment of colors to vertices and determining how the intermediate vertices are to be colored.

Clipping

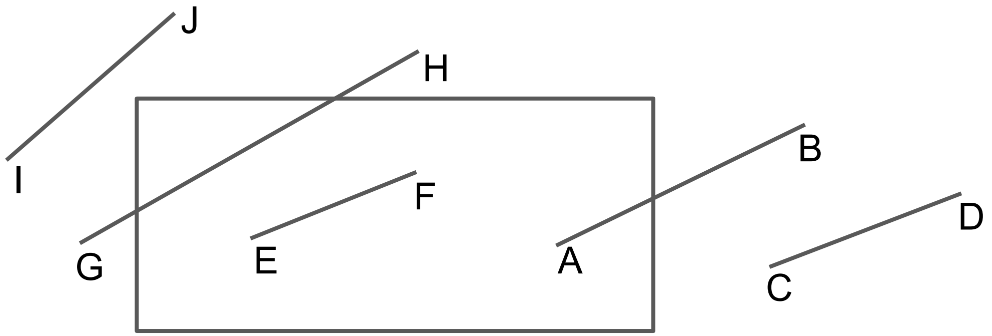

Application Program
Modeling
Graphics System
Geometry Processing
Rasterization
Fragment Processing
Framebuffer
Clipping

Clipping

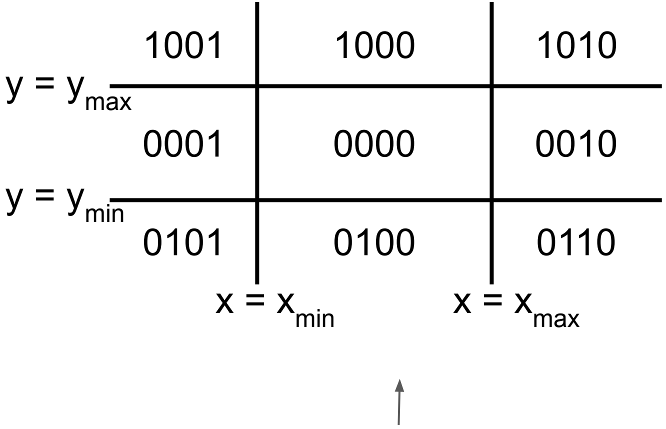
- Rasterization is expensive. Optimize what needs to be rasterized.
- Brute force approach: Clip every primitive against all the four sides of the view window (or six sides of the view volume)
- Computing intersections can be expensive since these calculations involve floating point divisions.

Clipping: Line segments

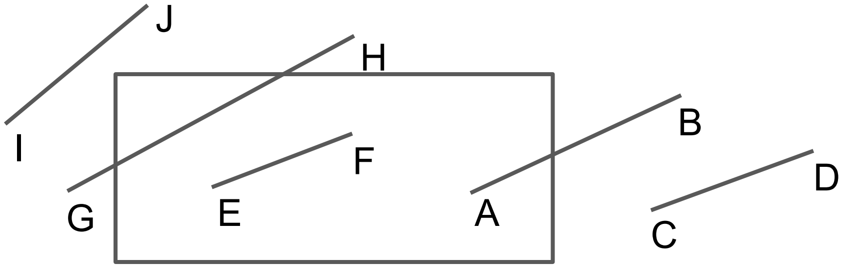
Cohen-Sutherland Line Clipping



Clipping: Cohen-Sutherland Line Clipping



$\mathbf{b_0 b_1 b_2 b_3}$



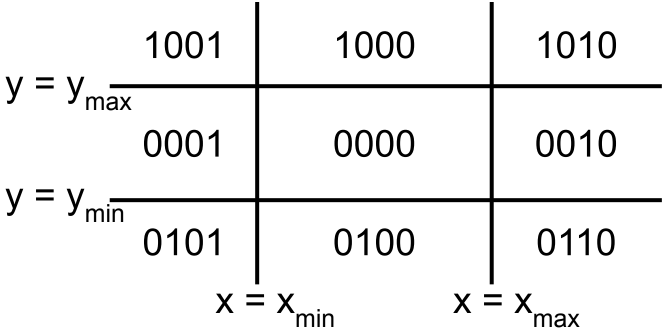
$$b_0 = \begin{cases} 1 & \text{if } y > y_{max} \\ 0 & \text{otherwise} \end{cases}$$

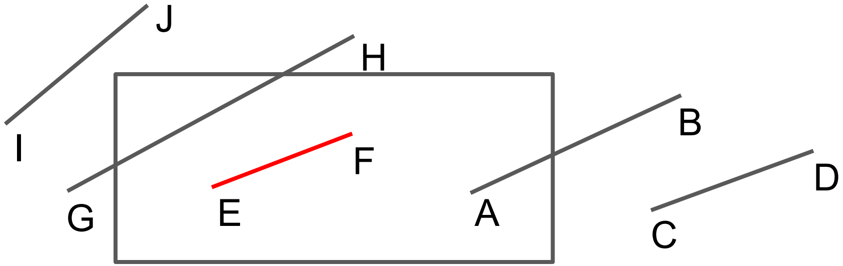
Clipping: Cohen-Sutherland Line Clipping

```
outcode(x, y):
    out = 0000
    if (y > y_max)
        out = out | 1000
    else if (y < y_min)
        out = out | 0100
    if (x > x_max)
        out = out | 0010
    else if (x < x_min)
        out = out | 0001
    return out
```

Clipping: Cohen-Sutherland Line Clipping

Case1:



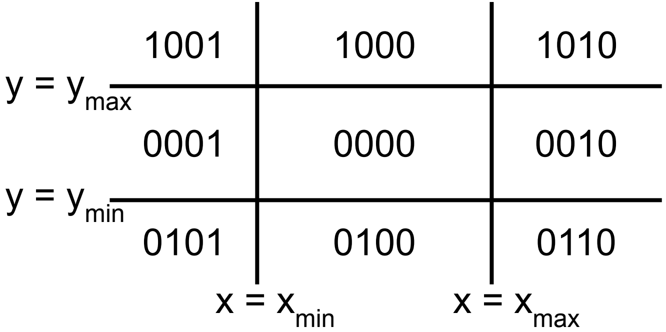


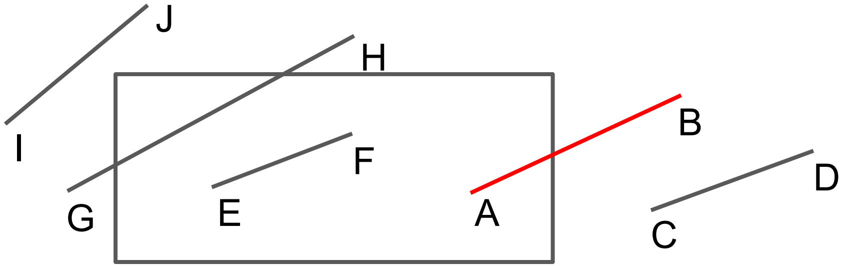
$o_1 = \text{outcode}(x_1, y_1)$
$o_2 = \text{outcode}(x_2, y_2)$

$\mathbf{o_1 = o_2 = 0}$

Clipping: Cohen-Sutherland Line Clipping

Case2:



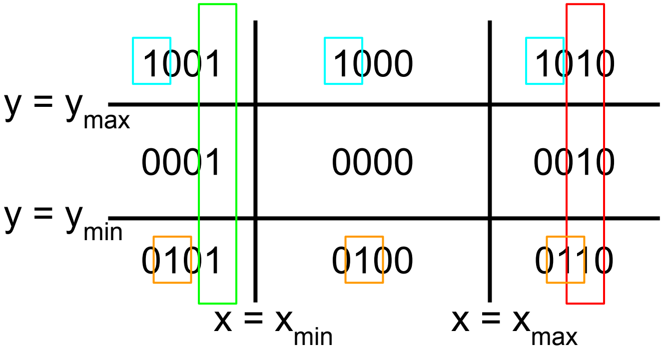


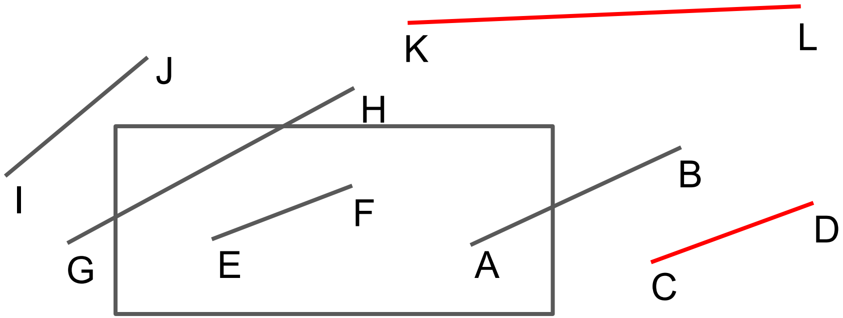
$o_1 = \text{outcode}(x_1, y_1)$
$o_2 = \text{outcode}(x_2, y_2)$

$o_1 \neq 0$ and $o_2 = 0$ or vice versa

Clipping: Cohen-Sutherland Line Clipping

Case3:



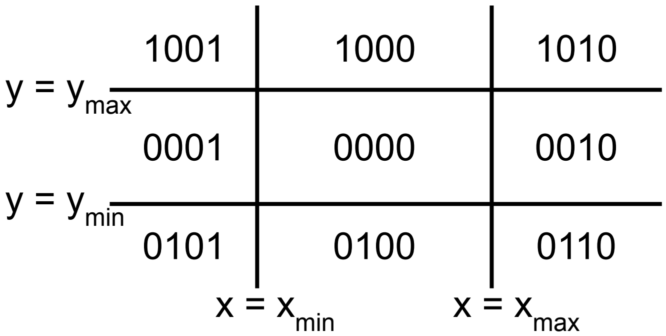


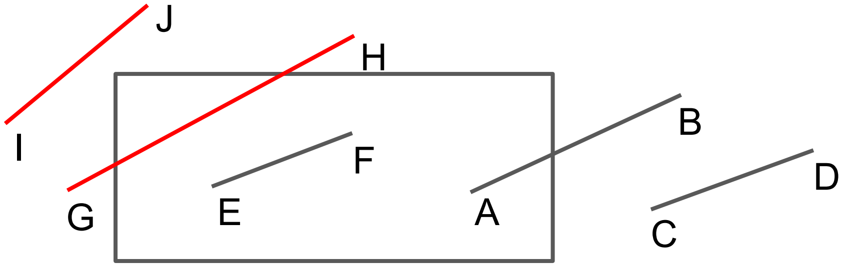
o_1 = outcode(x_1, y_1)
o_2 = outcode(x_2, y_2)

$\mathbf{o_1 \,\&\, o_2 \neq 0}$

Clipping: Cohen-Sutherland Line Clipping

Case4:





o_1 = outcode(x_1, y_1)
o_2 = outcode(x_2, y_2)

$\mathbf{o_1 \;\&\; o_2 = 0}$

Clipping: Cohen-Sutherland Line Clipping

Intersection calculation:

- $y = mx + h$
- $(y - y_1) = ((y_2 - y_1) / (x_2 - x_1)) * (x - x_1)$

Disadvantages:

1. Algorithm is recursively applied (till the clipped result can be accepted or rejected).
2. Vertical lines will have to be treated as a special case.

Clipping: Liang-Barsky Line Clipping

Algorithm makes use of parametric form of line.

Suppose:

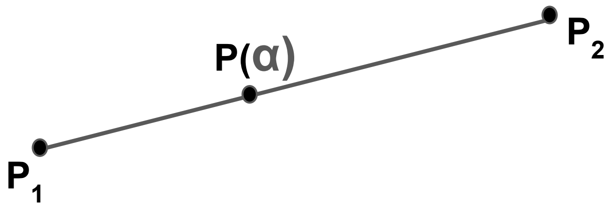
$$\mathbf{P_1 = [x_1, y_1]^T \text{ and } P_2 = [x_2, y_2]^T.}$$

Then,

$$\mathbf{P(\alpha) = (1-\alpha)P_1 + \alpha P_2}$$

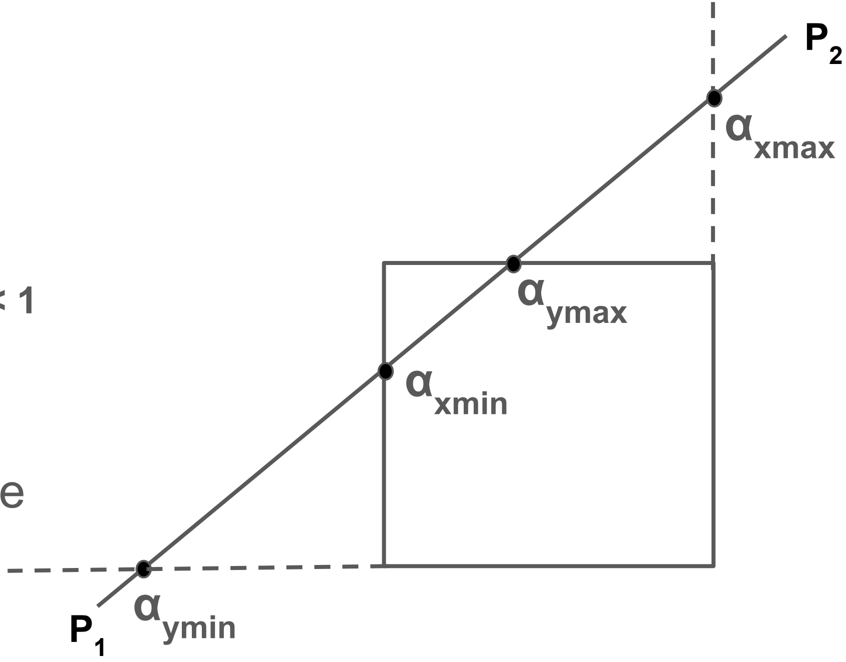
Or

$$\mathbf{x(\alpha) = (1-\alpha)x_1 + \alpha x_2,\ y(\alpha) = (1-\alpha)y_1 + \alpha y_2}$$



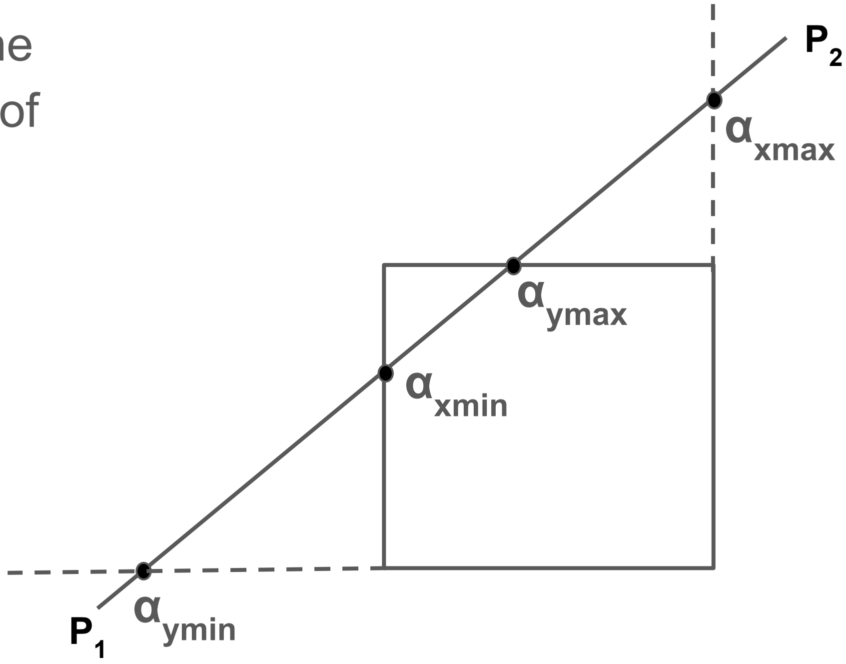
Clipping: Liang-Barsky Line Clipping

- Compute the four **α** values corresponding to four sides of the clipping window
- Compare them to find the order.
  - In this case $\mathbf{0 < \alpha_{ymin} < \alpha_{xmin} < \alpha_{ymax} < \alpha_{xmax} < 1}$
- Between two alpha values corresponding to vertical extremes if there is a horizontal alpha value (or vice versa) it indicates the line passes through the clip window.



Clipping: Liang-Barsky Line Clipping

- Any **α** value that is < 0 or > 1 implies the intersection happens on the extension of the actual line segment.



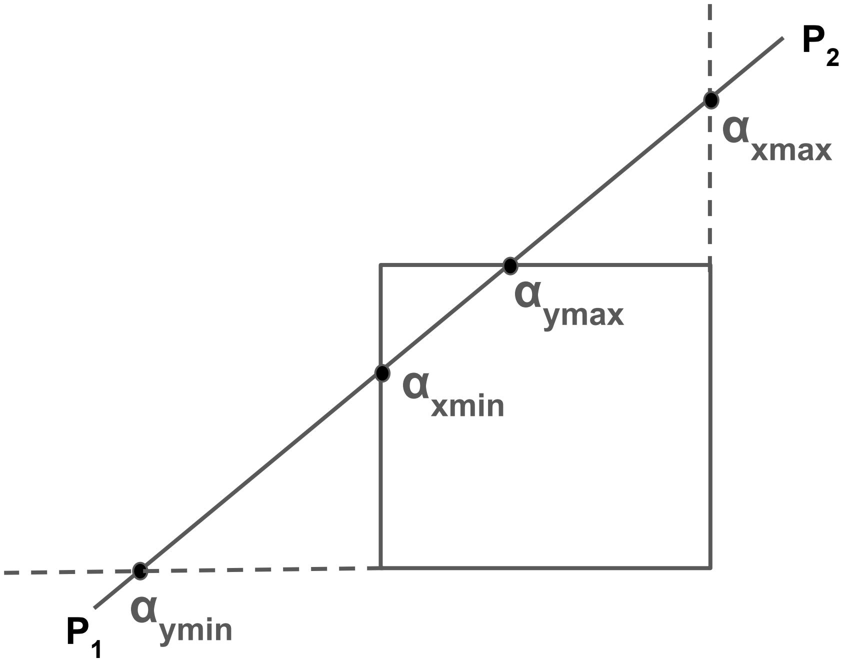
Clipping: Liang-Barsky Line Clipping

Computing alpha value:

$$\alpha_{ymin} = (y_{min} - y_1) / (y_2 - y_1)$$

or

$$\Delta y \, \alpha_{ymin} = \Delta y_{min}$$



Clipping

- Polygons (convex) can be clipped by successively clipping the edges against the clipping window.
- Concave polygons need to be tessellated (broken into smaller convex polygons) and the constituent polygons can be clipped as above.
- The clipping algorithms discussed extend naturally to 3 dimensions.