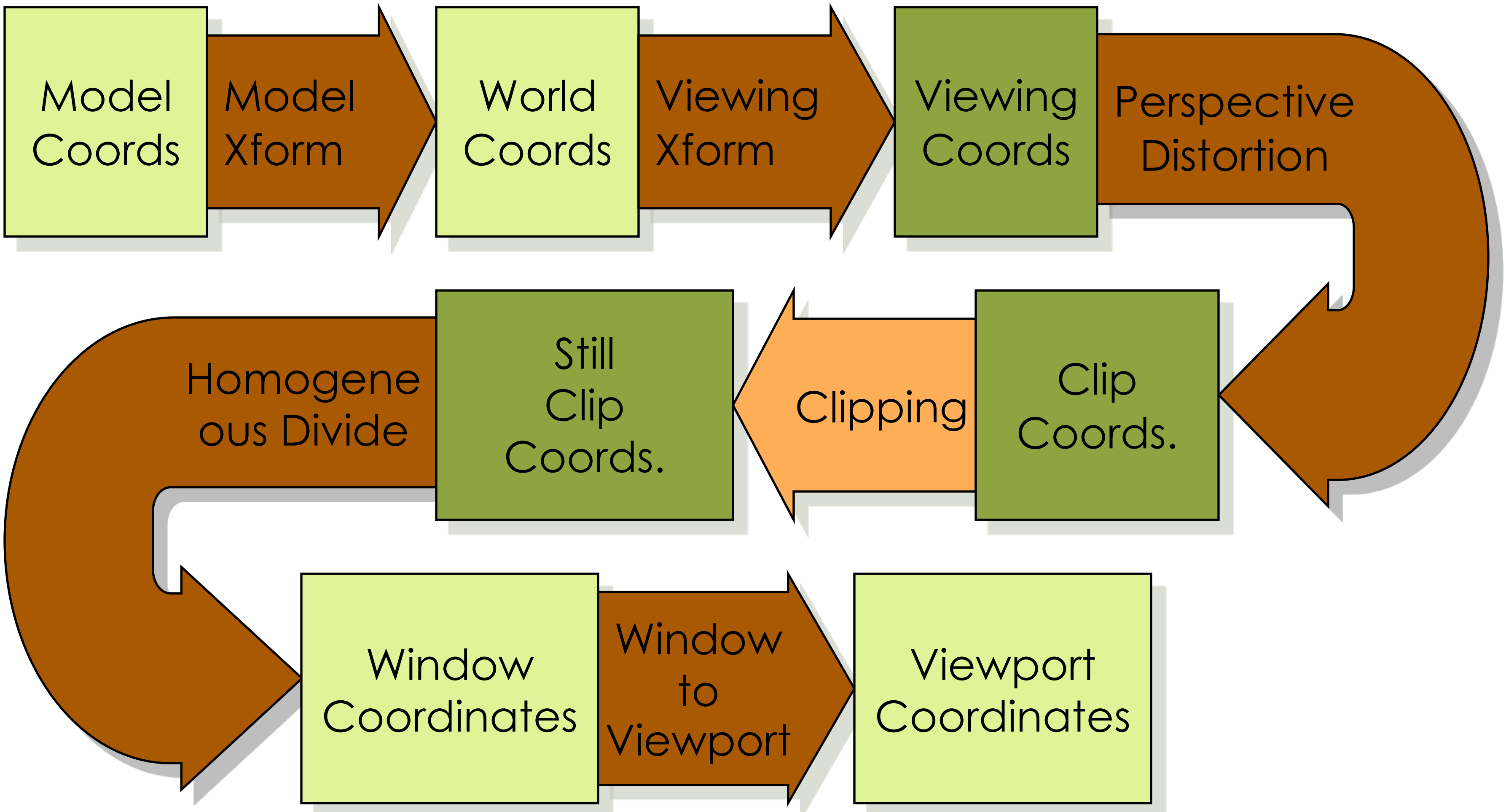
CS 418: Interactive Computer Graphics

Clipping

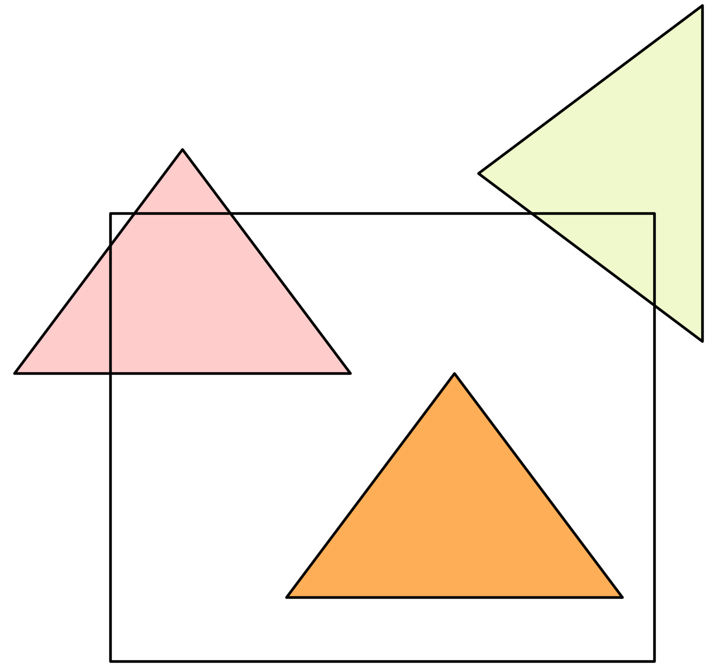
Eric Shaffer

Based on slides by John Hart

Graphics Pipeline

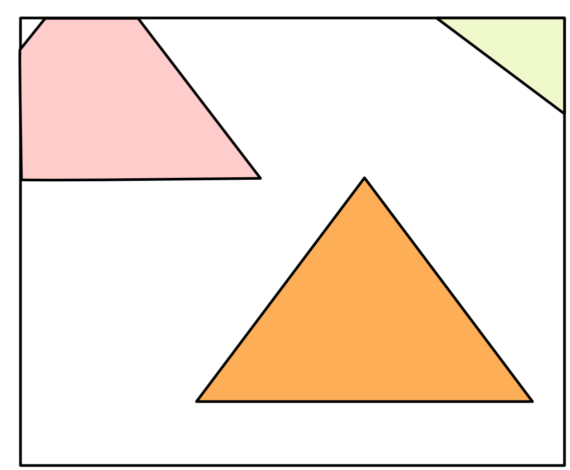


Why Clip?



Why not just transform all triangles to the screen and just ignore pixels off the screen?

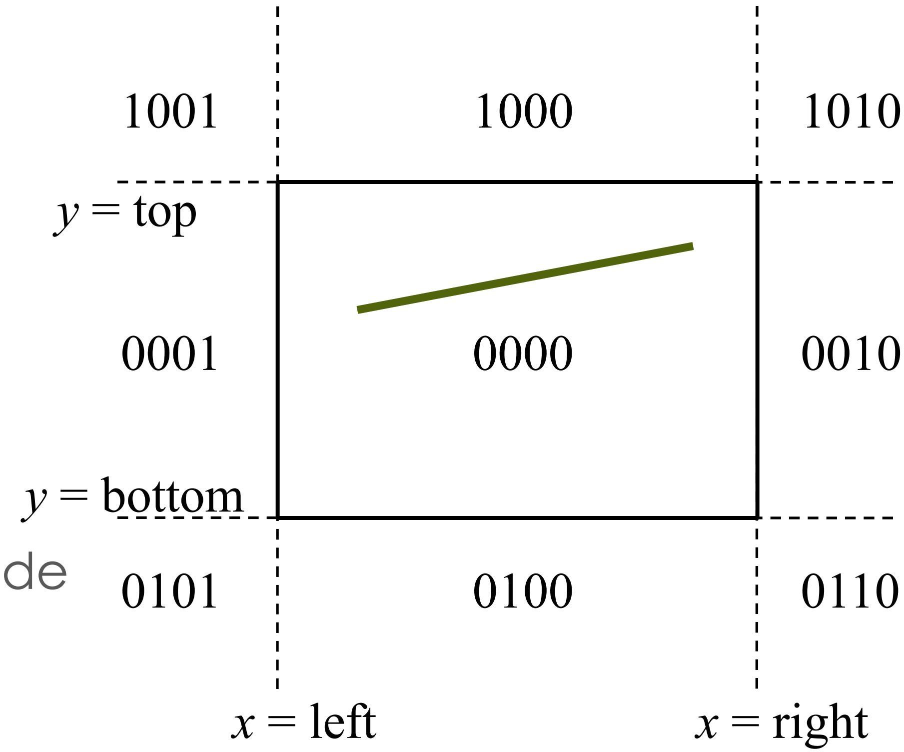
- Takes time to rasterize a triangle
- Very small number of triangles fall within the viewing frustum
- WebGL clips automatically
  - …you don't have to implement clipping
  - You should know how it works



Clipping Happens When?

- Different rasterization engines can make different choices
  - WebGL does it after the vertex shader runs
    - In 3D
    - Before performing division by the homogeneous coordinate
  - Could also be done in 2D, after the division
- We'll look at a 2D clipping algorithm
  - Generalizes to 3D

Outcodes

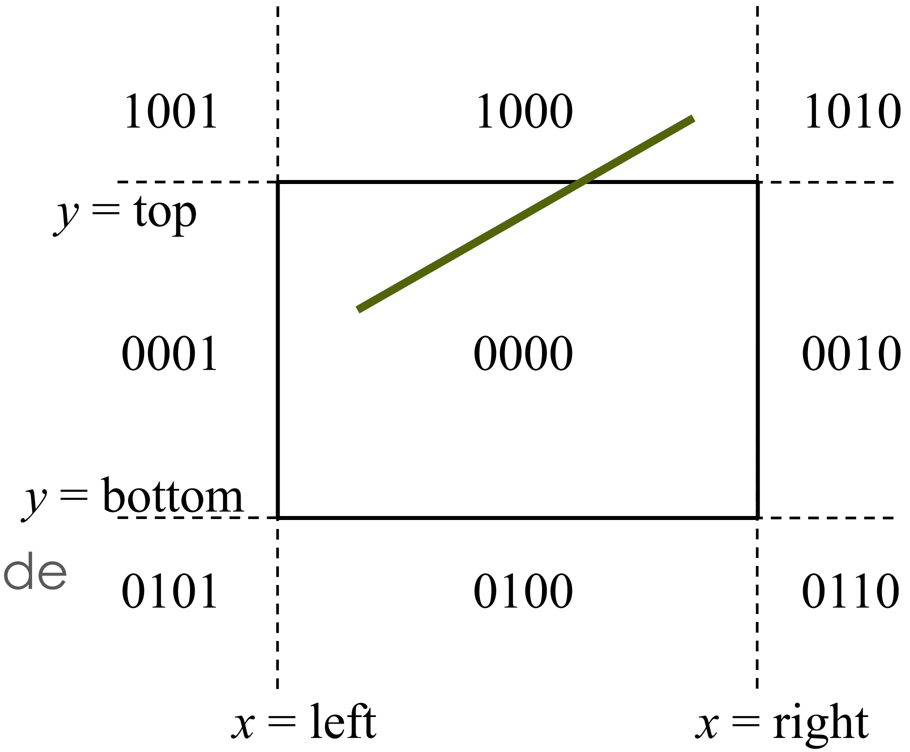


- Cohen-Sutherland
- Assign segment endpoints a bitcode

  $b_3b_2b_1b_0$

  - $b_0 = x < \text{left}$
  - $b_1 = x > \text{right}$
  - $b_2 = y < \text{bottom}$
  - $b_3 = y > \text{top}$
- Let o_0 = outcode(x0,y0), o_1 = outcode(x1,y1)
  - $o_0 = o_1 = 0$: segment visible
  - $o_0 = 0$, $o_1 \neq 0$: segment must be clipped

Outcodes



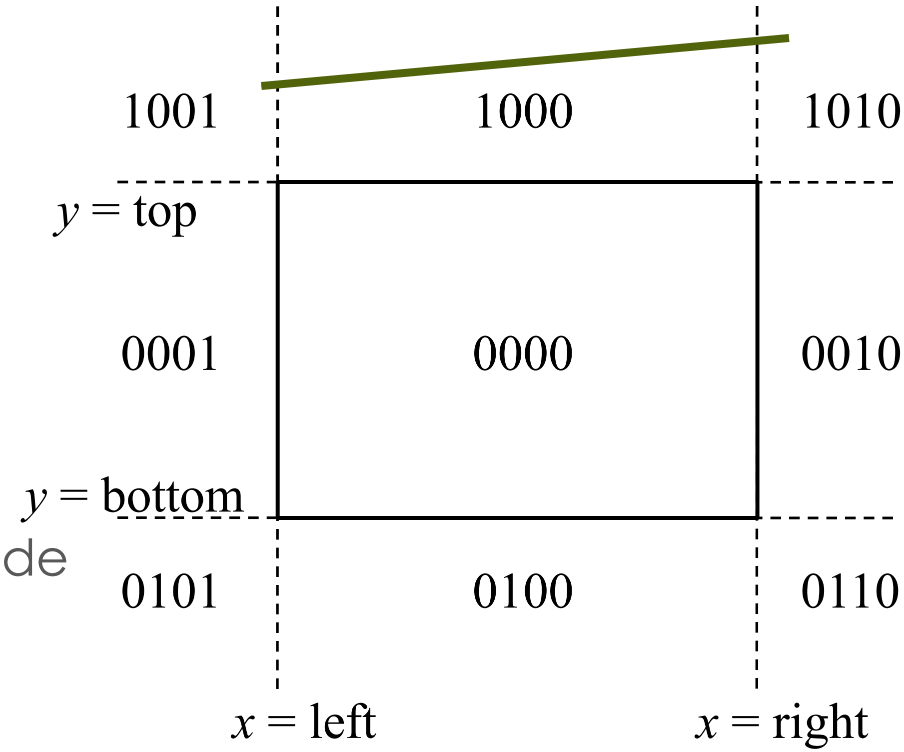
- Cohen-Sutherland
- Assign segment endpoints a bitcode

  $b_3b_2b_1b_0$
  - b_0 = x < left
  - b_1 = x > right
  - b_2 = y < bottom
  - b_3 = y > top
- Let o_0 = outcode(x0,y0), o_1 = outcode(x1,y1)
  - $o_0 = o_1 = 0$: segment visible
  - $o_0 = 0$, $o_1 \neq 0$: segment must be clipped

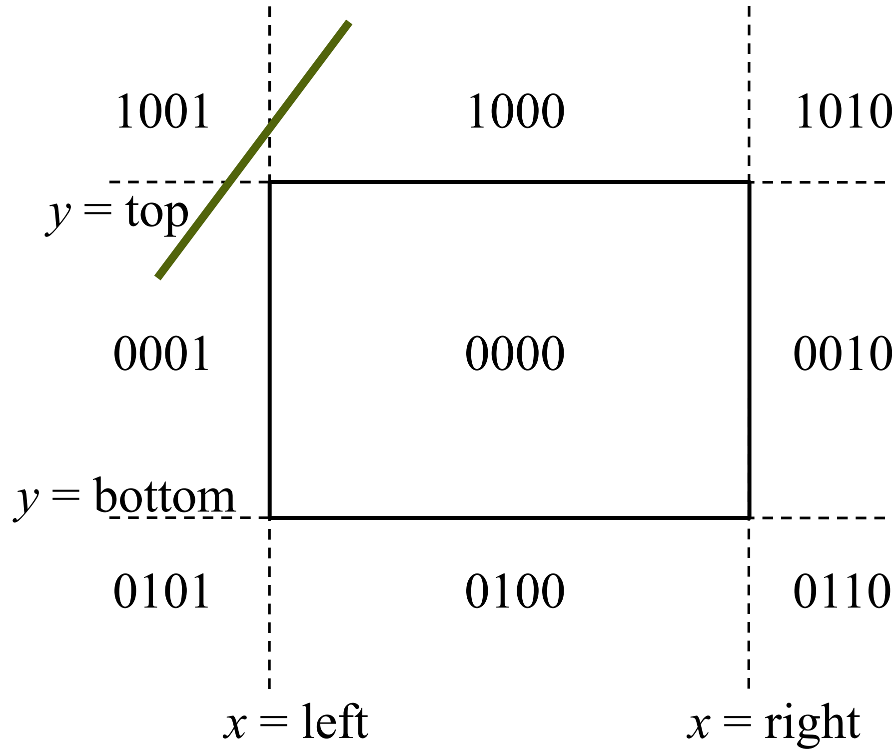
Outcodes

- Cohen-Sutherland
- Assign segment endpoints a bitcode

  $b_3 b_2 b_1 b_0$
  - b_0 = x < left
  - b_1 = x > right
  - b_2 = y < bottom
  - b_3 = y > top
- Let o_0 = outcode(x0,y0), o_1 = outcode(x1,y1)
  - $o_0 = o_1 = 0$: segment visible
  - $o_0 = 0$, $o_1 \neq 0$: segment must be clipped
  - o_0 & $o_1 \neq 0$: segment can be ignored



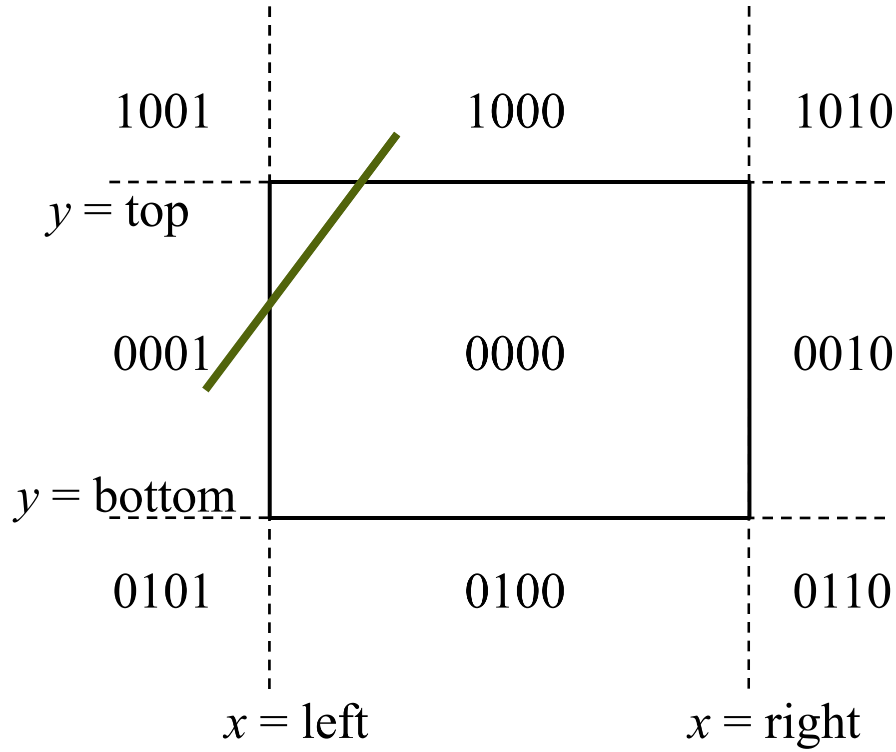
Outcodes



- Cohen-Sutherland
- Assign segment endpoints a bitcode

  $b_3 b_2 b_1 b_0$
  - $b_0 = x < \text{left}$
  - $b_1 = x > \text{right}$
  - $b_2 = y < \text{bottom}$
  - $b_3 = y > \text{top}$
- Let o_0 = outcode(x0,y0), o_1 = outcode(x1,y1)
  - $o_0 = o_1 = 0$: segment visible
  - $o_0 = 0$, $o_1 \neq 0$: segment must be clipped
  - o_0 & $o_1 \neq 0$: segment can be ignored
  - o_0 & $o_1 = 0$: segment might need clipping

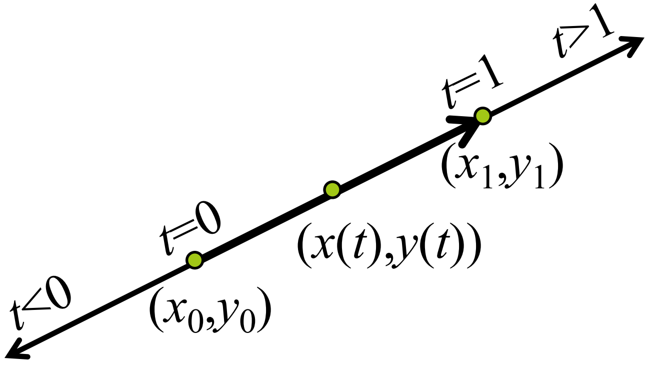
Outcodes



- Cohen-Sutherland
- Assign segment endpoints a bitcode

  $b_3b_2b_1b_0$
  - b_0 = x < left
  - b_1 = x > right
  - b_2 = y < bottom
  - b_3 = y > top
- Let o_0 = outcode(x0,y0), o_1 = outcode(x1,y1)
  - $o_0 = o_1 = 0$: segment visible
  - $o_0 = 0$, $o_1 \neq 0$: segment must be clipped
  - o_0 & $o_1 \neq 0$: segment can be ignored
  - o_0 & $o_1 = 0$: segment might need clipping

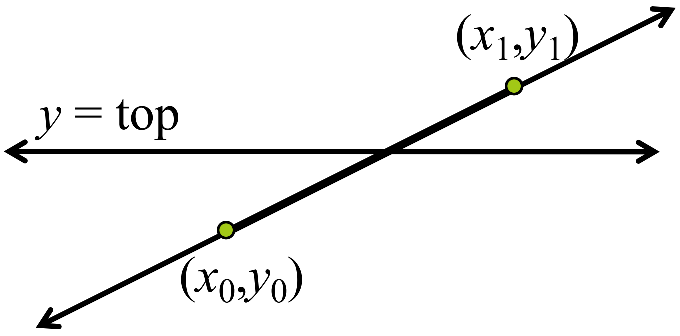
Intersecting Lines



- Parametric representation of a line segment

$$x(t) = x_0 + t\,(x_1 - x_0)$$
$$y(t) = y_0 + t\,(y_1 - y_0)$$

Intersecting Lines



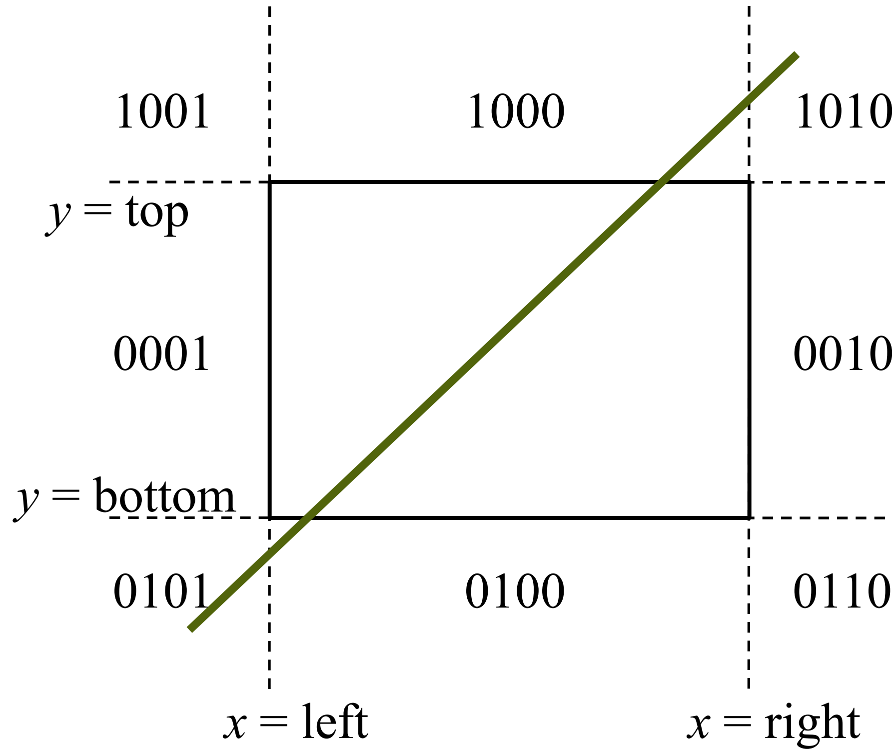
- Parametric representation of a line segment

$$x(t) = x_0 + t\,(x_1 - x_0)$$
$$y(t) = y_0 + t\,(y_1 - y_0)$$

- Plug in clipping window edge to find t

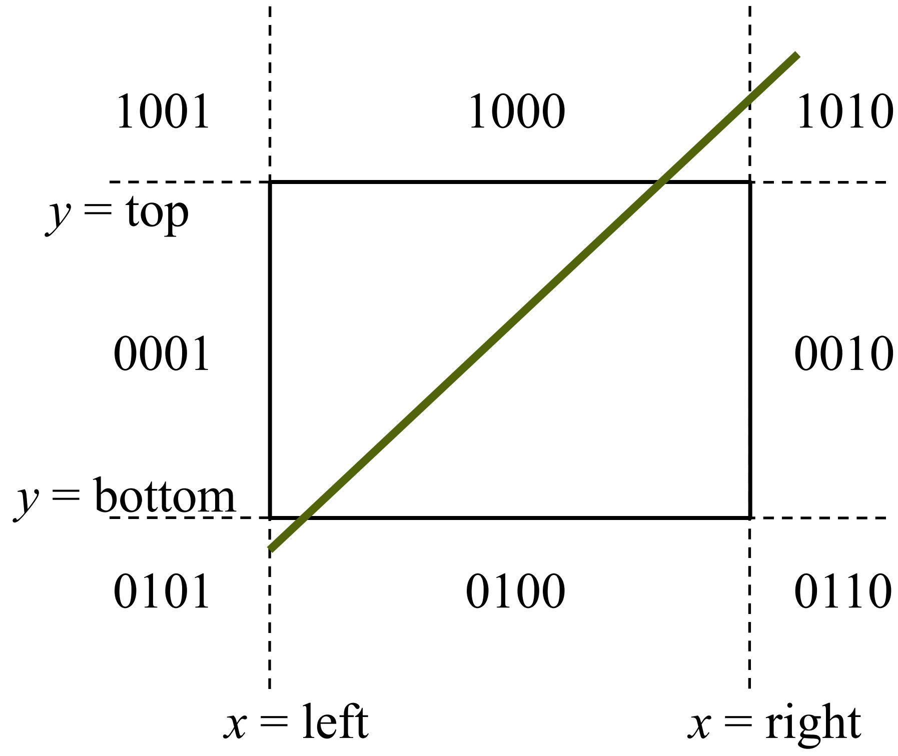
$$\text{top} = y_0 + t\,(y_1 - y_0)$$
$$t = (\text{top} - y_0)/(y_1 - y_0)$$

Cohen-Sutherland Clipping

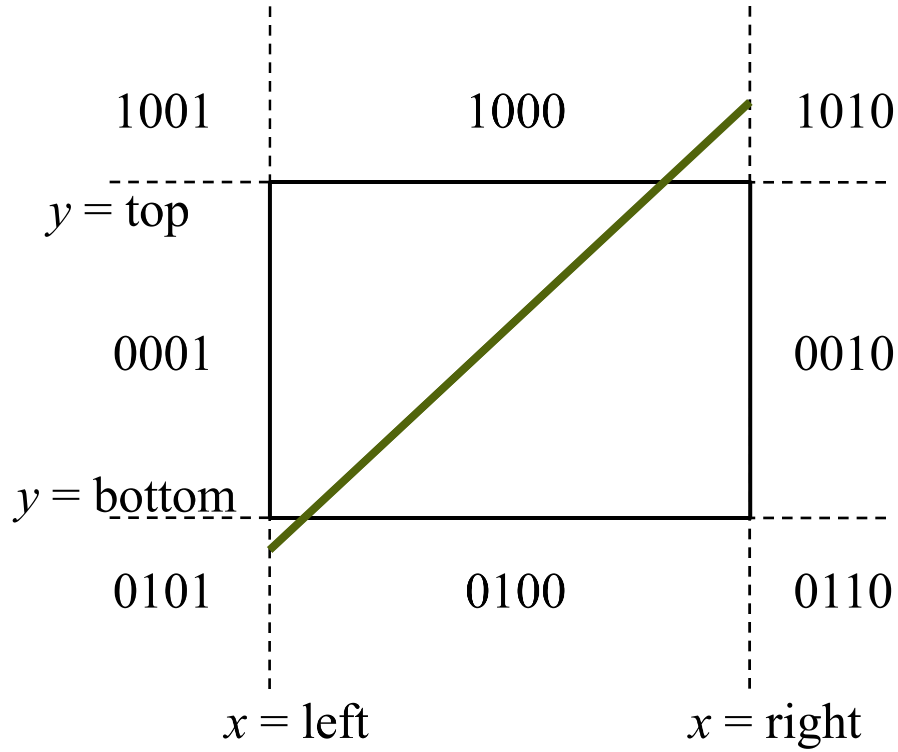


Cohen-Sutherland Clipping

- First clip 0101
- Move (x_0,y_0) to (left,…)

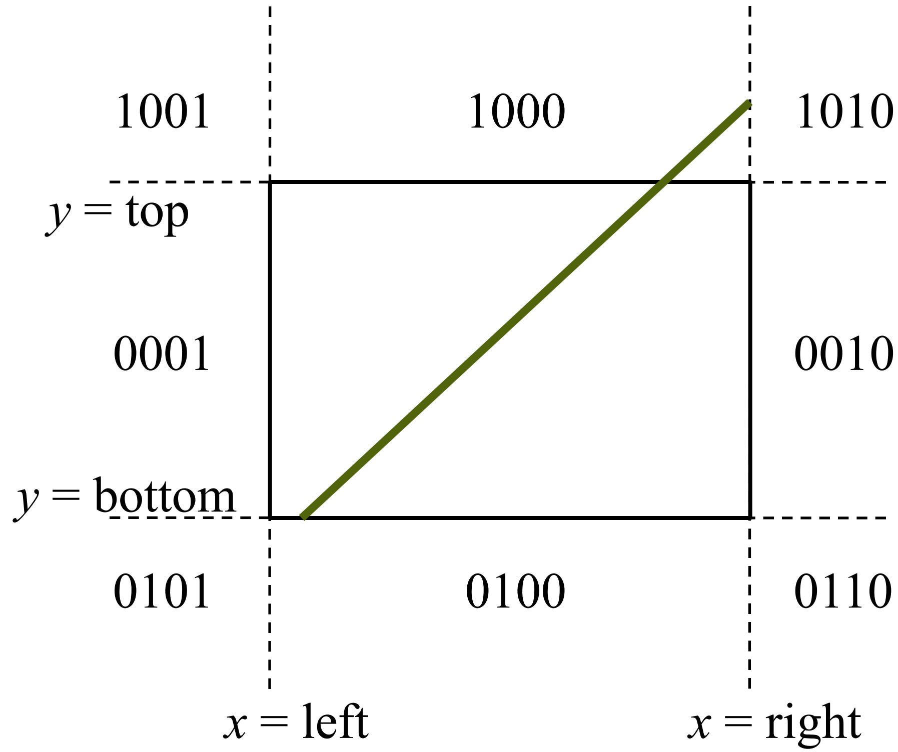


Cohen-Sutherland Clipping



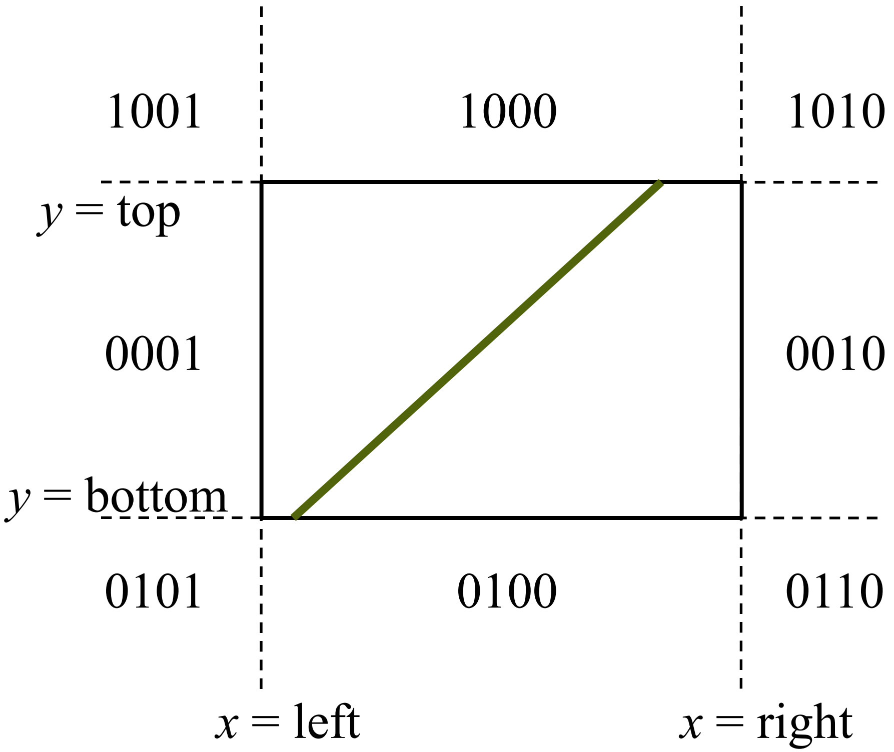
- First clip 0101
- Move (x_0,y_0) to (left,…)
- Then clip 1010
- Move (x_1,y_1) to (right,…)

Cohen-Sutherland Clipping



- First clip 0001
- Move (x_0,y_0) to (left,…)
- Then clip 0010
- Move (x_1,y_1) to (right,…)
- Then clip 0100
- Move (x_0,y_0) again, now to (…,bottom)

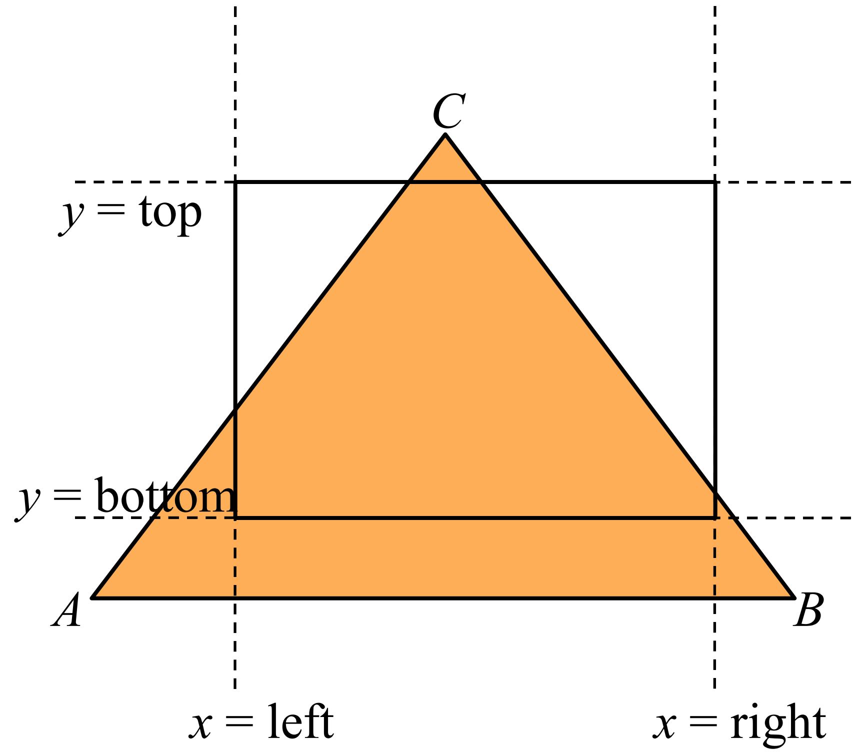
Cohen-Sutherland Clipping



- First clip 0101
- Move (x_0,y_0) to (left,…)
- Then clip 1010
- Move (x_1,y_1) to (right,…)
- Then clip 0100
- Move (x_0,y_0) again, now to (…,bottom)
- Finally clip 1000
- Move (x_1,y_1) again, now to (…,top)

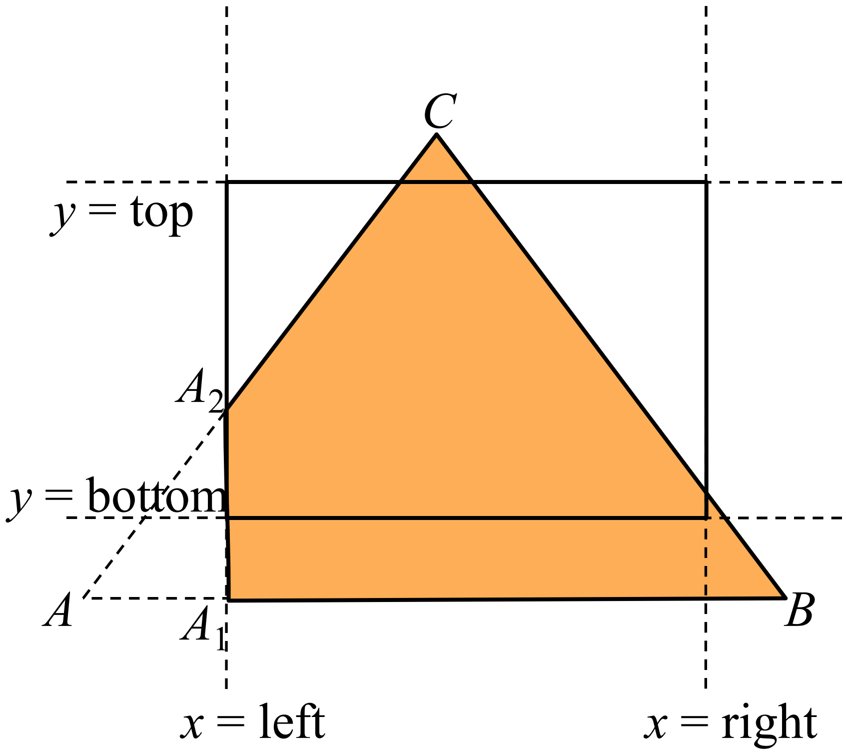
Polygon Clipping

- Sutherland-Hodgman
- Polygon *ABC*



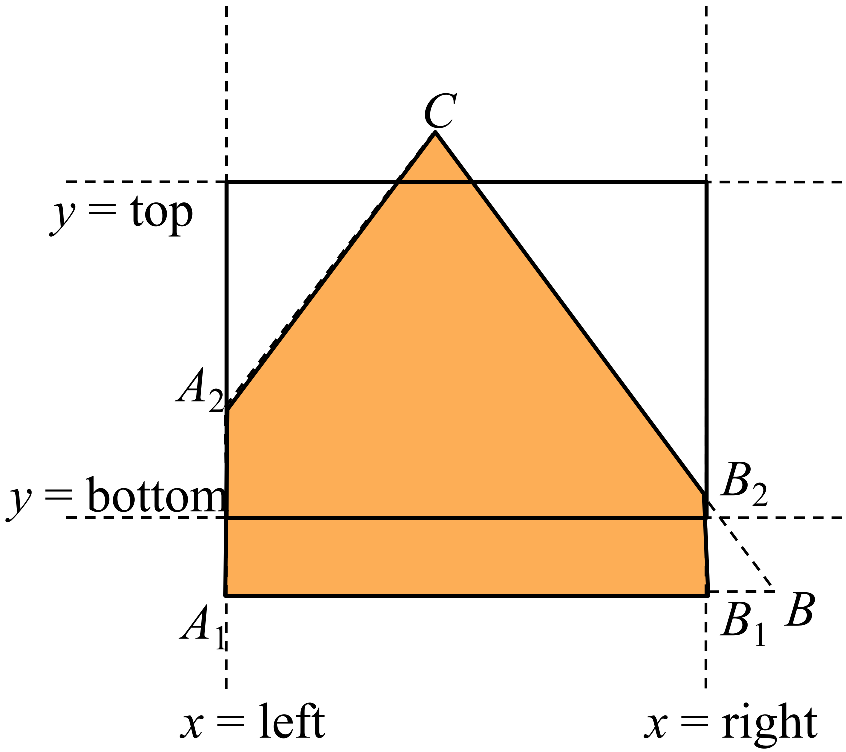
Polygon Clipping

- Sutherland-Hodgman
- Polygon ABC
- Clip left: A_1BCA_2

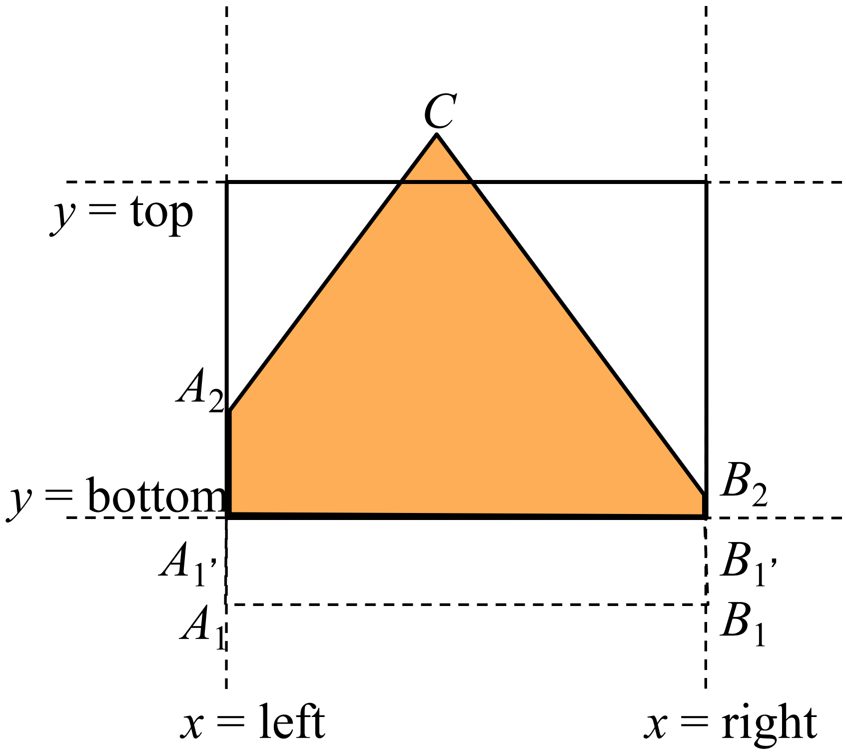


Polygon Clipping

- Sutherland-Hodgman
- Polygon ABC
- Clip left: A_1BCA_2
- Clip right: $A_1B_1B_2CA_2$



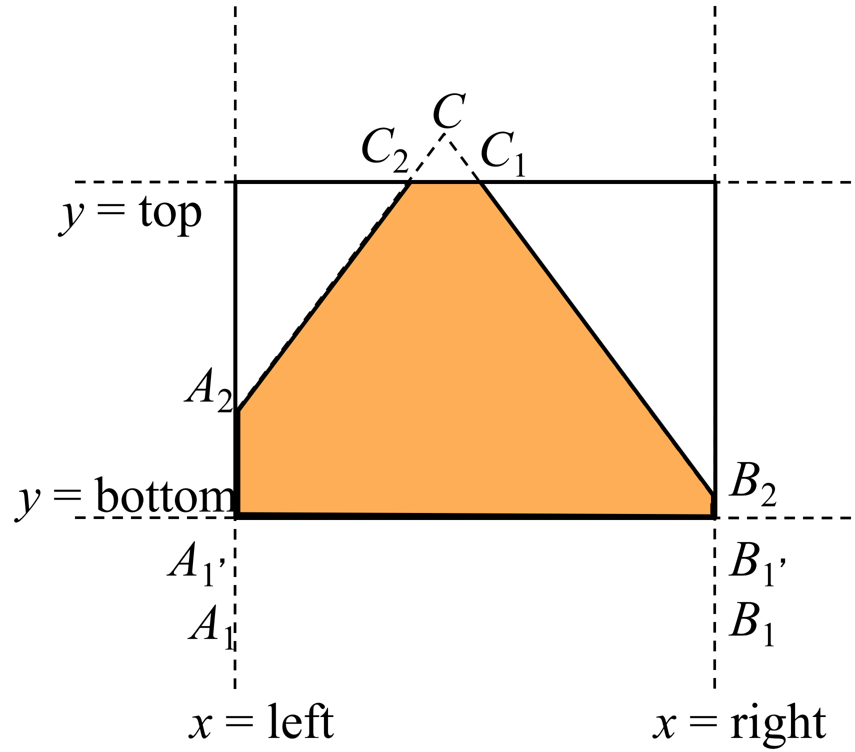
Polygon Clipping



- Sutherland-Hodgman
- Polygon ABC
- Clip left: A_1BCA_2
- Clip right: $A_1B_1B_2CA_2$
- Clip bottom: $A_1'B_1'B_2CA_2$

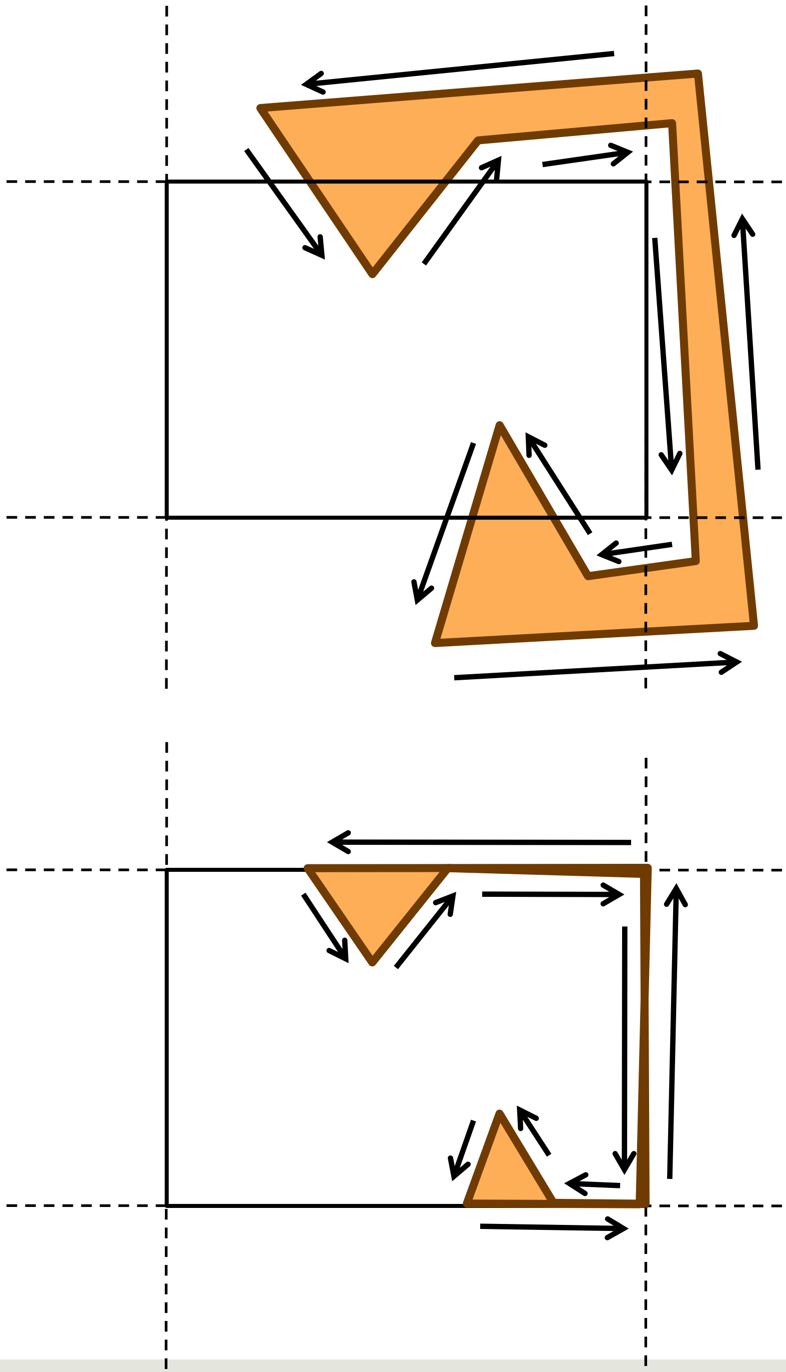
Polygon Clipping

- Sutherland-Hodgman
- Polygon ABC
- Clip left: A_1BCA_2
- Clip right: $A_1B_1B_2CA_2$
- Clip bottom: $A_1B_1'B_2'CA_2$
- Clip top: $A_1B_1'B_2'C_1C_2A_2$



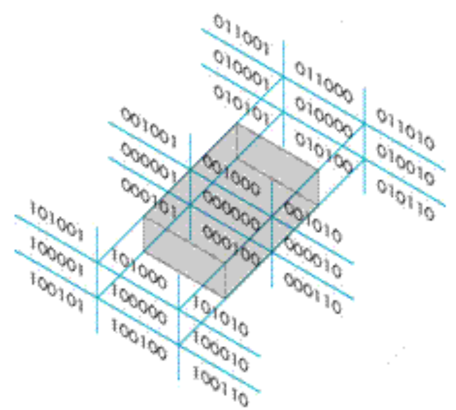
Concave Clipping

- Sutherland-Hodgman
- Clip segments even if they are trivially rejectible (rejectionable?)
- Outputs a single polygon that appears as multiple polygons
- Reversed edges don′t get filled

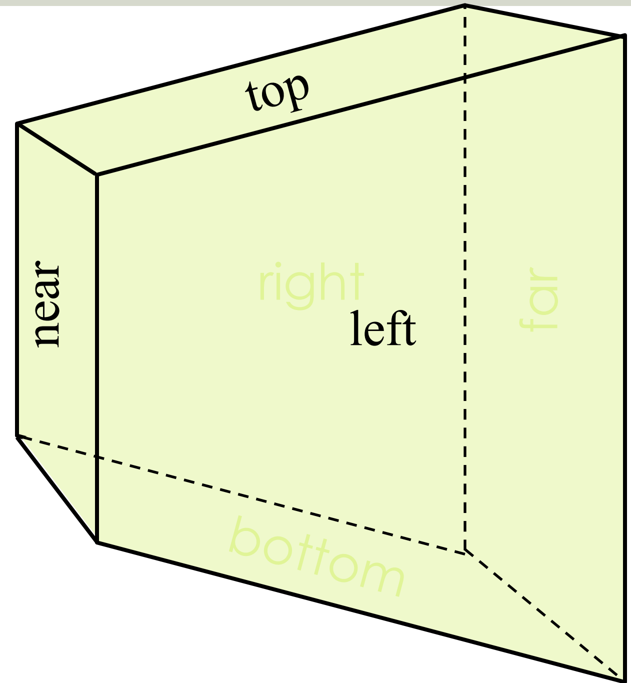


Clipping in 3D

- Clipping can be done in 3D clip coordinates
- Need to be able to compute
  - Which side of a plane a point is on
  - Line segment – Plane intersections
- Can still use Cohen-Sutherland
  - 6-bit outcodes
  - 27 different regions of space

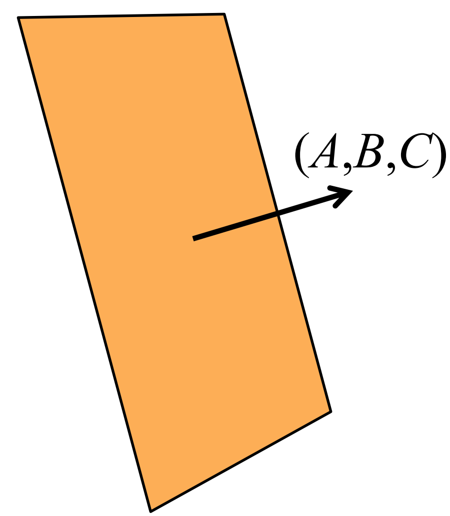


Clipping in 3-D



- Need to keep depth (z-coordinate) of geometry for visible surface detection
- Generalize oriented screen edge to oriented clipping plane $C = (A,B,C,D)$
- Then *any* homogeneous point $P = (x,y,z,w)^T$ classified as
  - "on" if $C\,P = 0$
  - "in" if $C\,P < 0$
  - "out" if $C\,P > 0$

$$\begin{bmatrix} A & B & C & D \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = 0$$



$$Ax + By + Cz + D = 0$$
$$\Updownarrow$$
$$wAx + wBy + wCz + wD = 0$$

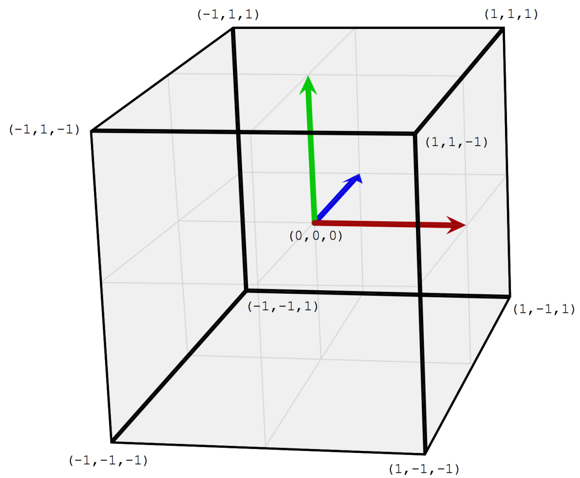
Clipping in 3D

- Plane equation can be rewritten $n \cdot (p - p_0) = 0$
  - n the normal and p_0 is a point on the plan
  - plane is formed by all points p for which equation is true
- For a line defined by points p_1 and p_2
  - parametric equation is $p(t) = (1 - t)p_1 + tp_2$
- You can find the intersection of a plane and line:

$$t = \frac{n \cdot (p_0 - p_1)}{n \cdot (p_2 - p_1)}$$

Clipping in WebGL

- Clipping happens after the vertices leave the vertex shader
  - But before the homogeneous divide
- Everything outside the [-1,+1] cube is discarded or clipped
  - Axis-aligned clipping planes
  - Inside-outside test simpler
    - e.g. is z coordinate > 1?
- Quick review
  - What plane is the projection plane?



Clipspace

Clipping in WebGL

- Everything is orthographically projected to z=0 plane
- Remember – the viewing transformation and projection transformation move the geometry you want to see into the WebGL view volume
  - The eyepoint in the view volume image below is not meaningful
    - Things “behind” the eye will be visible