Homogeneous Coordinates

A way of representing data

Representing n−d space by n+1 dimensions

1. representing big integer numbers

   for example,

   16bit word for an integer between
   −32768 and 32767

   How to represent a number > 32767 ?

   a position [60000, y, z]

   homogeneous coordindates:

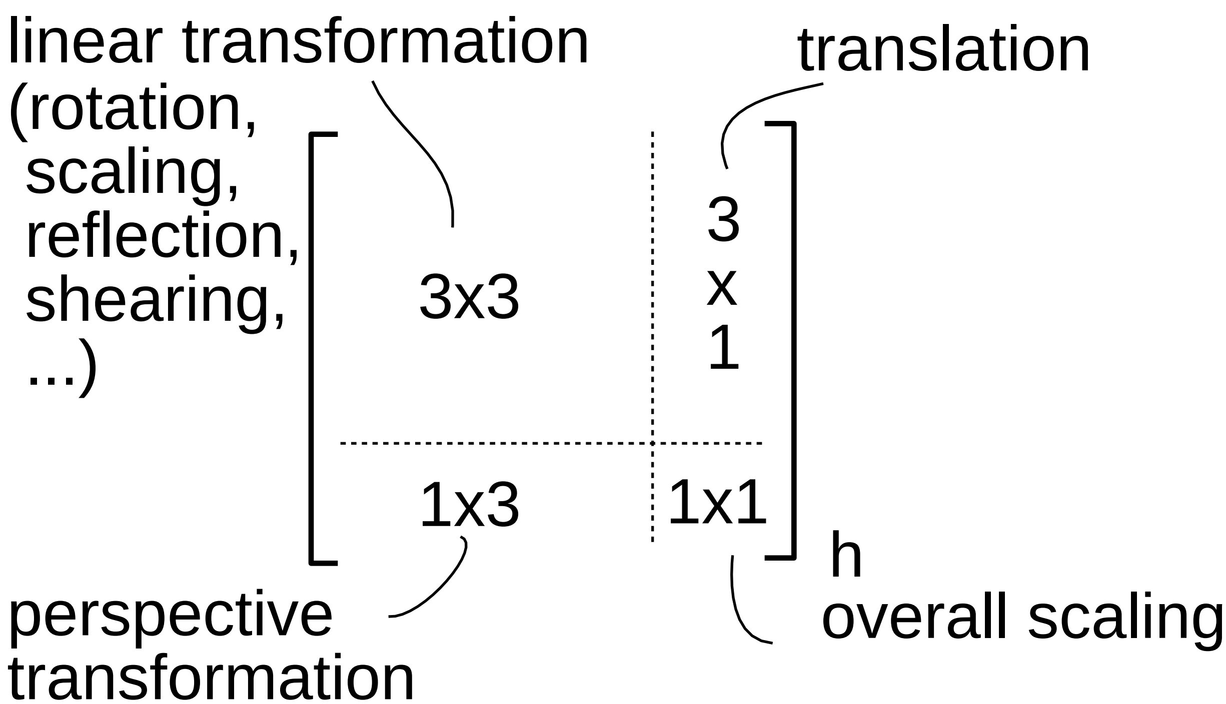
   [30000, y/2, z/2, 1/2]

<u>Homogeneous Coordinates</u> (cont'd)

2. defining an object and its transformation

- distinguish between a vector and a point
- modify the position of the origin of the coordinate system

  there is no room in the 3x3 matrix to specify translation!

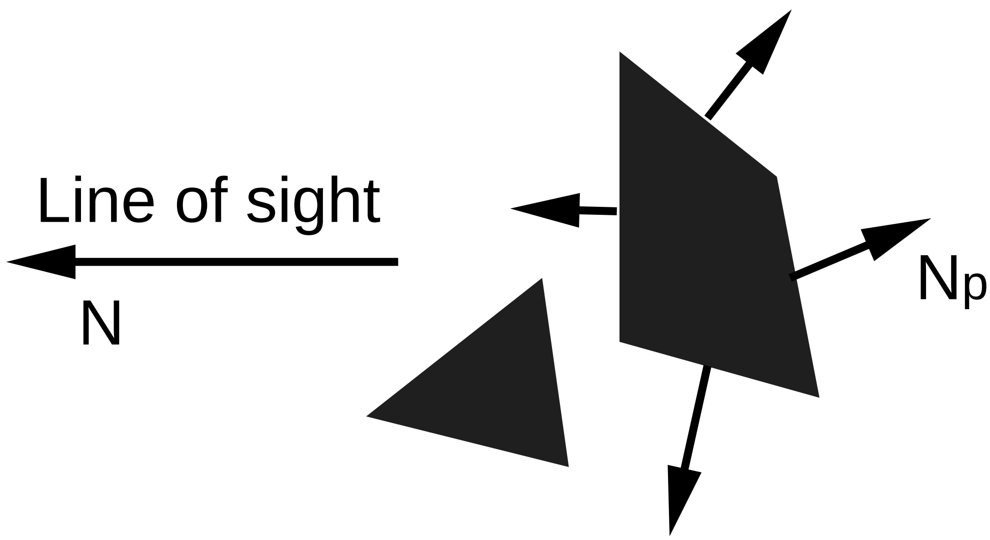


- there is no unique homegeneous coordinate representation!
- h = 0 ?

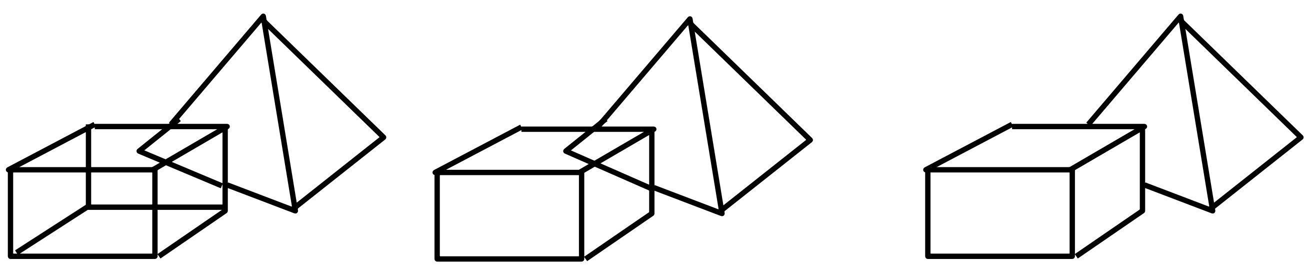
3D View Space

Operations

1. Culling/Back Face Elimination



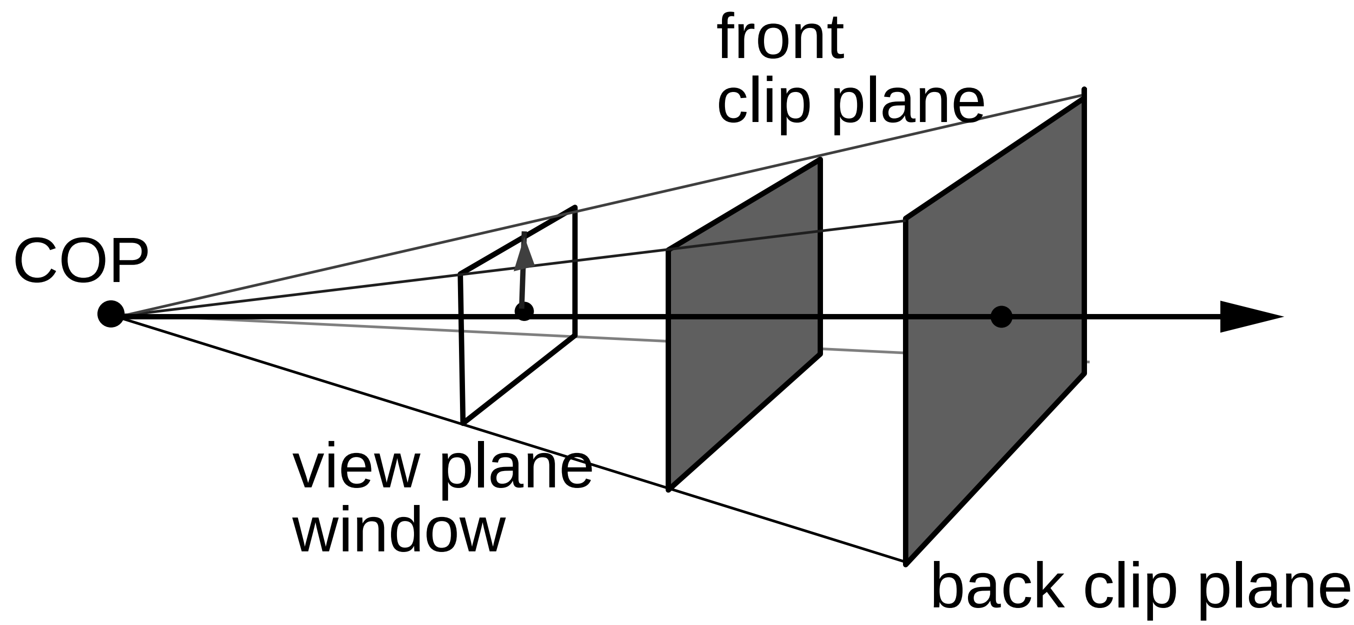
Visibility = $N_p \bullet N > 0$

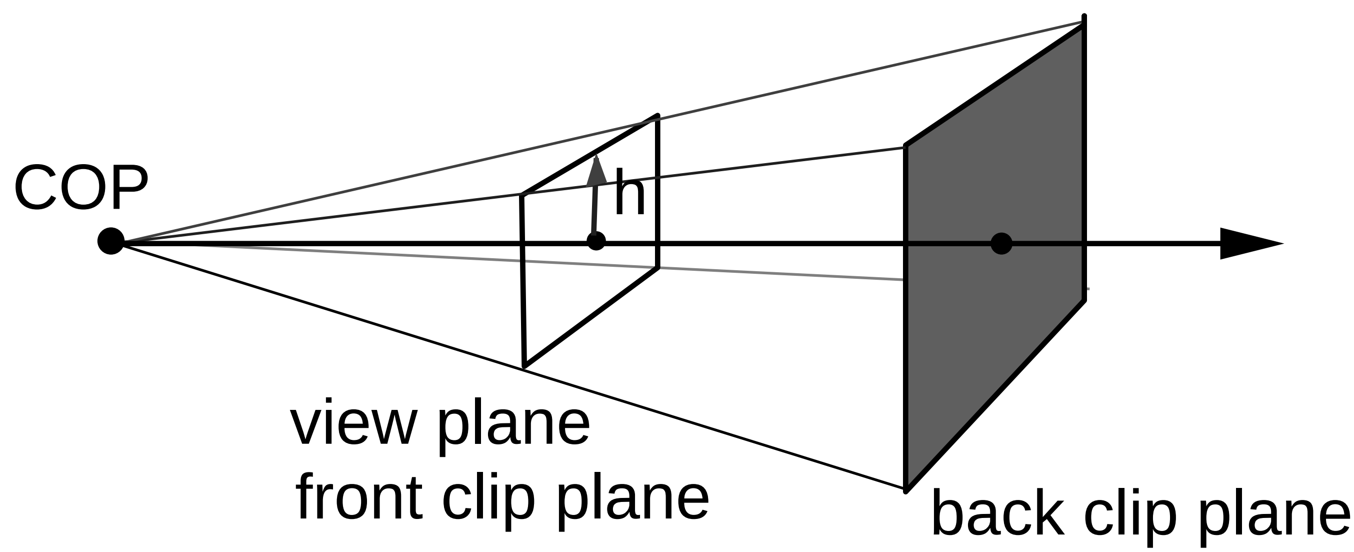


2. View Volume Clipping

3. Hidden Surface Removal

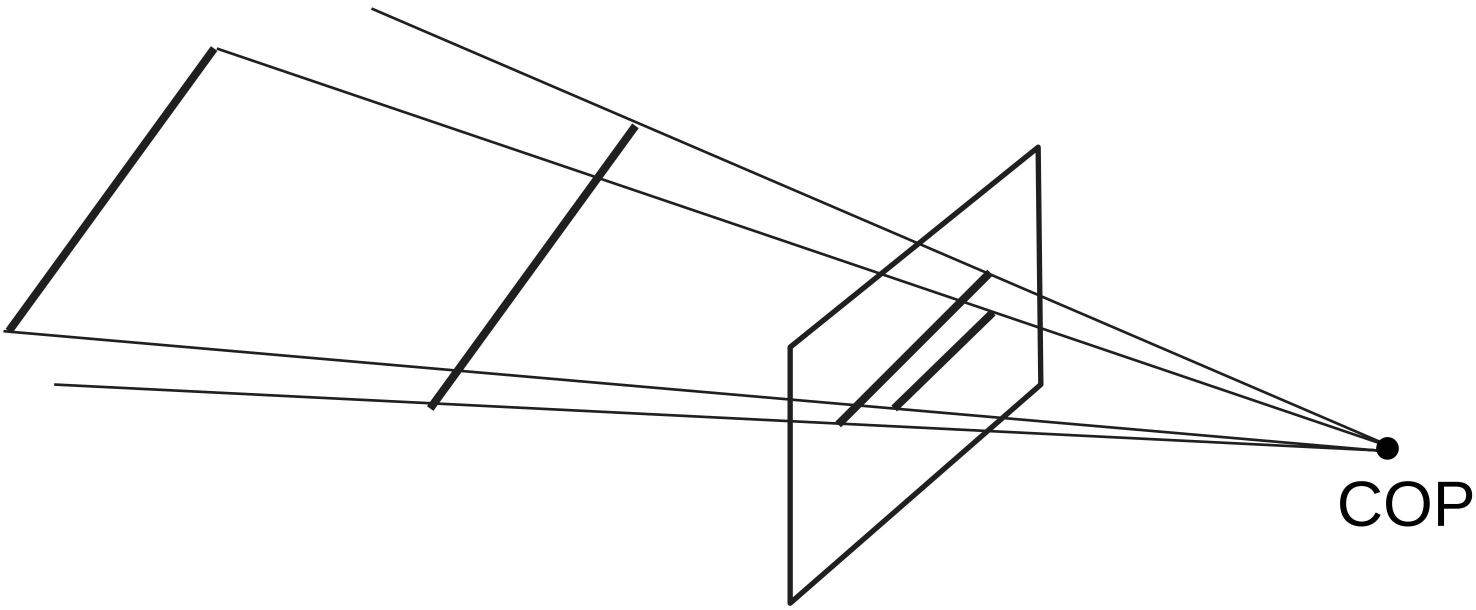
View Volume (View Frustum)



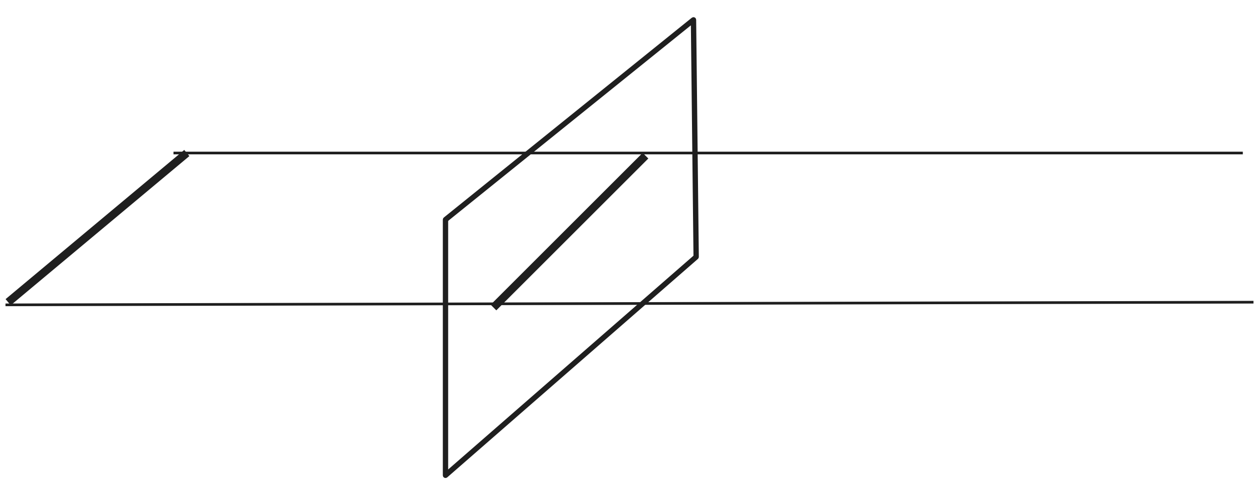


Perspective Projection

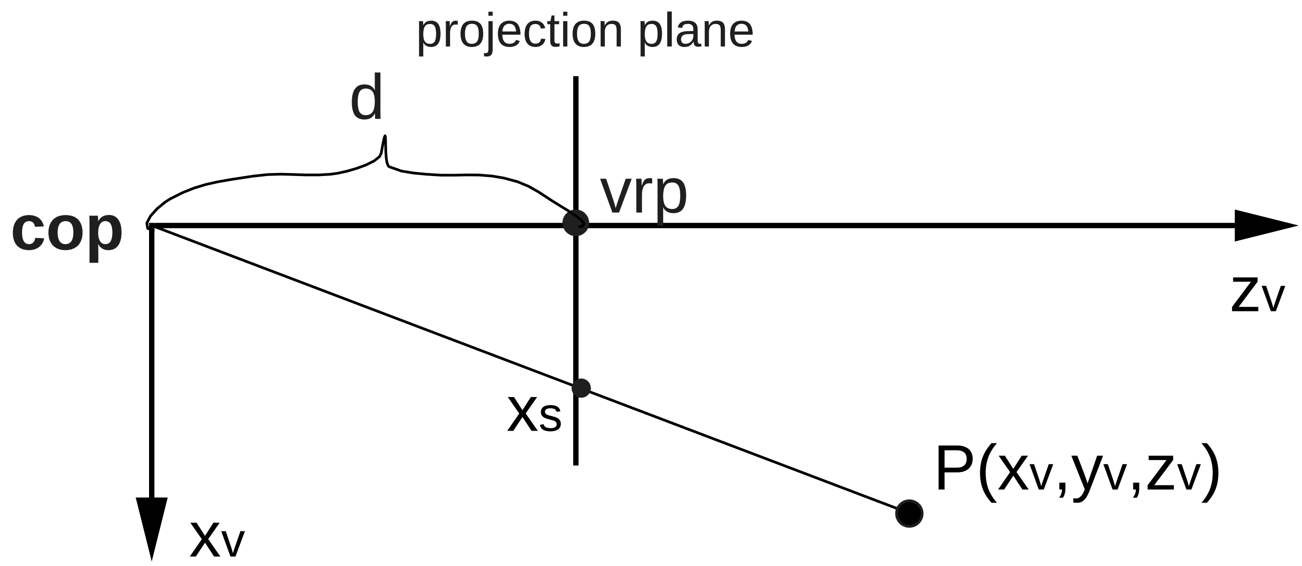
- foreshortening
- line width is not preserved, so are angles
- irreversible

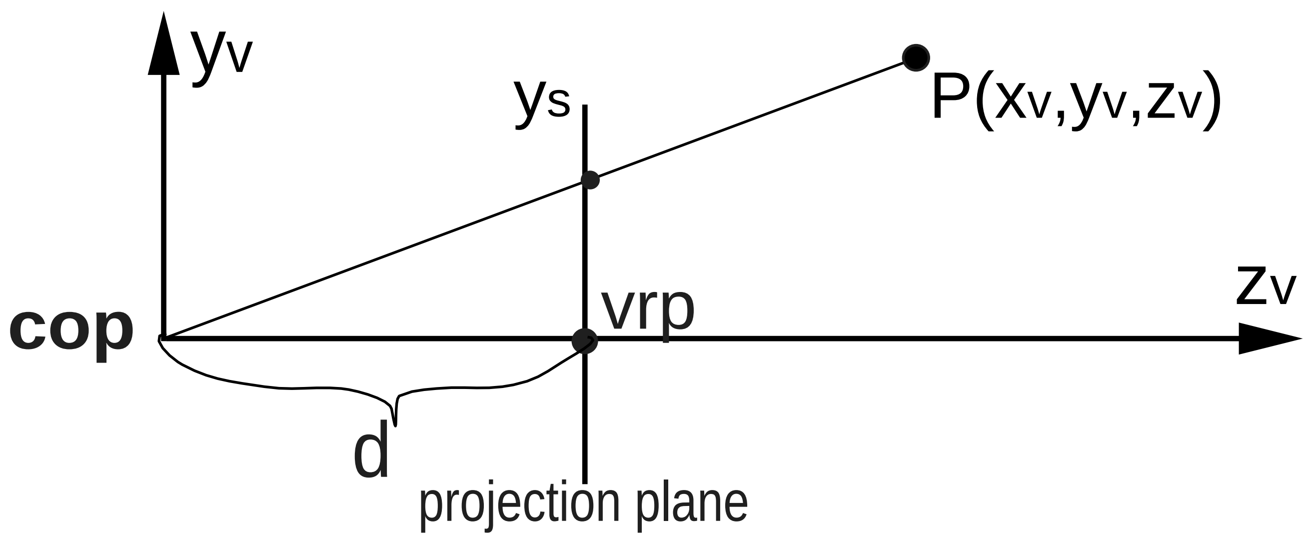


What happens if we move COP to infinitely far away?



3D Screen Space





$$\frac{x_s}{d} = \frac{x_v}{z_v} \qquad \frac{y_s}{d} = \frac{y_v}{z_v}$$

$$x_s = \frac{d}{z_v} x_v \qquad y_s = \frac{d}{z_v} y_v$$

Non−linear!!

$$x_s = \frac{d}{z_v} x_v \qquad y_s = \frac{d}{z_v} y_v$$

Expressed in Homogeneous Coordindates

$$\begin{bmatrix} X \\ Y \\ Z \\ w \end{bmatrix} = M \begin{bmatrix} x_v \\ y_v \\ z_v \\ 1 \end{bmatrix} \qquad \text{where} \qquad M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

that is, $$\begin{bmatrix} X \\ Y \\ Z \\ w \end{bmatrix} = \begin{bmatrix} x_v \\ y_v \\ z_v \\ z_v/d \end{bmatrix}$$

after perspective divide

$$\begin{bmatrix} x_s \\ y_s \\ z_s \\ 1 \end{bmatrix} = \begin{bmatrix} X/w \\ Y/w \\ Z/w \\ w/w \end{bmatrix} = \begin{bmatrix} x_v/w \\ y_v/w \\ d \\ 1 \end{bmatrix}$$

For perspective projection:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

For parallel projection:

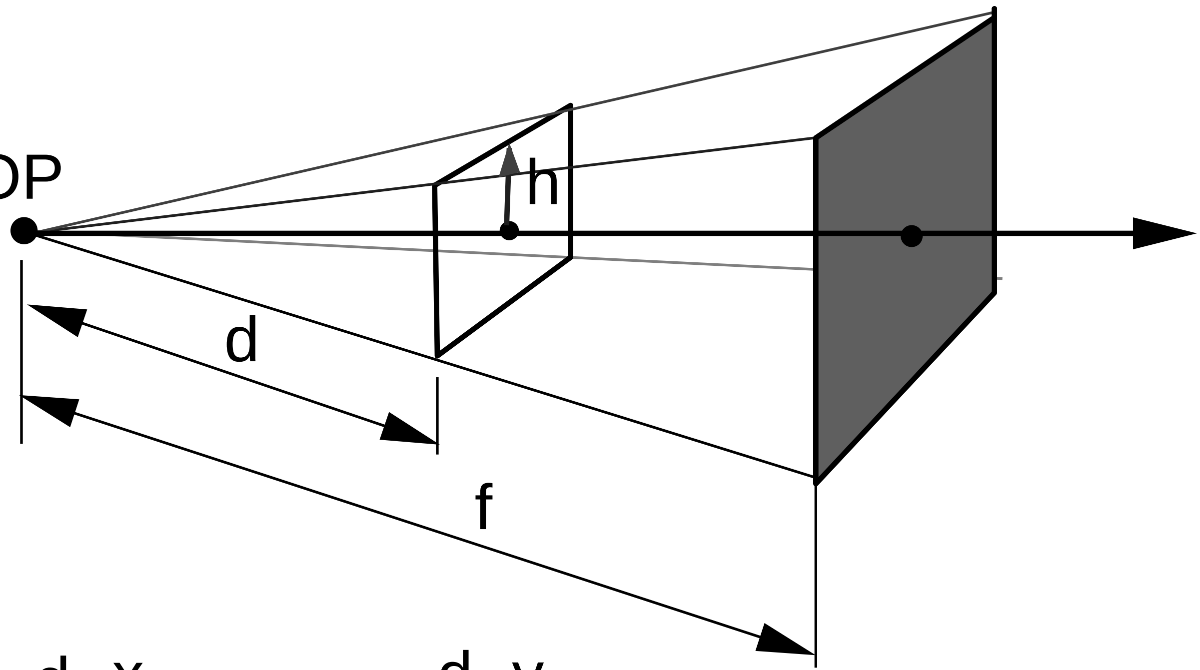
$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$x_s = x_v$
$y_s = y_v$
$z_s = 0$ (with projection plan at z=0)

An overall transformatin from World to Screen space can be expressed as:

$$\boxed{P' = M_{proj} M_{view} P}$$

View Volume and **Depth**



$$x_s = \frac{d}{h}\frac{x_v}{z_v} \qquad y_s = \frac{d}{h}\frac{y_v}{z_v}$$

$(-1 < x_s, y_s < 1)$

$$z_s = ? \qquad w = z_v$$

$$= f\,\frac{1 - (d/z_v)}{f - d}$$

$$M_{proj} = \begin{bmatrix} d/h & 0 & 0 & 0 \\ 0 & d/h & 0 & 0 \\ 0 & 0 & f/(f-d) & -df/(f-d) \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

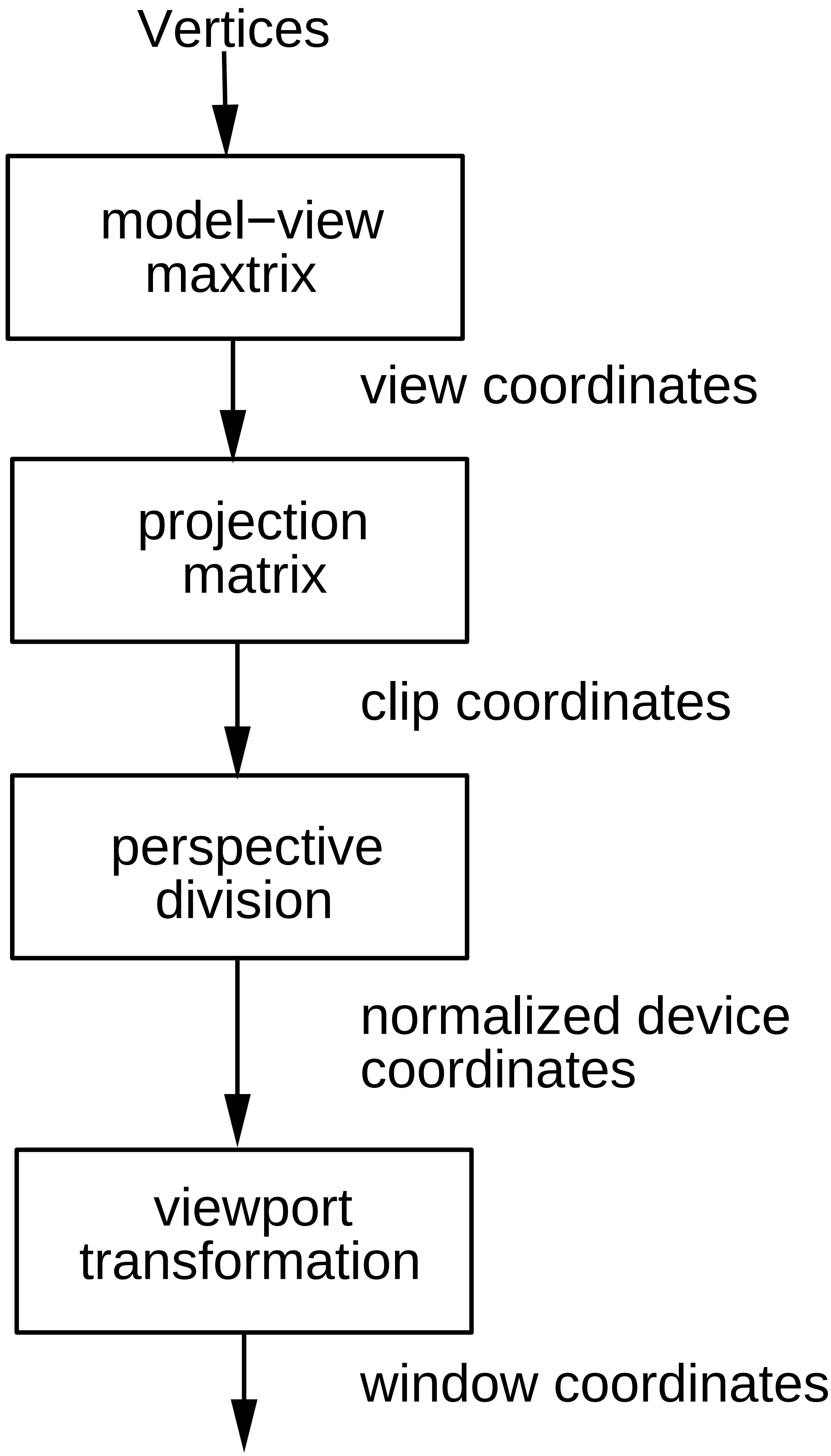
$$x_s = \frac{d}{h} x_v$$

$$y_s = \frac{d}{h} y_v$$

$$z_s = fz_v/(f-d) - df(f-d)$$

$$w = z_v$$

Stages of Vertex Transformation



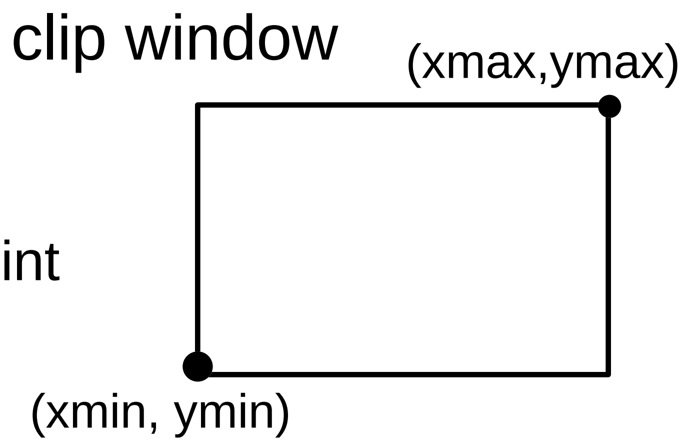
<u>Clipping</u>

1. Point clipping
2. Line segment clipping
3. Polygon clipping
4. Clipping in three dimensions

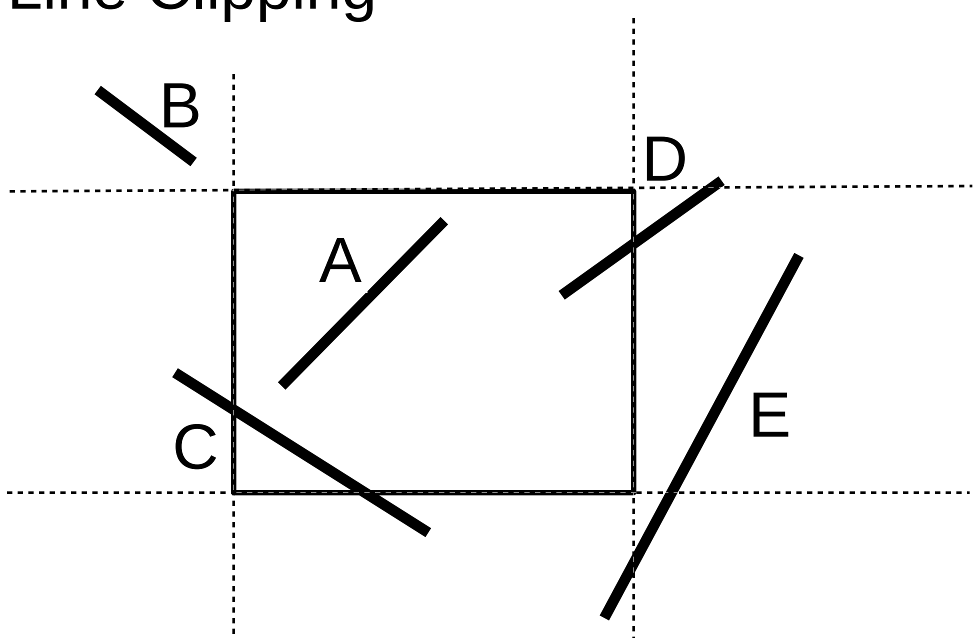
Point Clipping

$x_{min} \le x \le x_{max}$
$y_{min} \le y \le y_{max}$

Otherwise, do not draw the point



Line Clipping



Brute Force Line Clipping

Compute intersections of line with every window boundary −> expensive

Cohen−Sutherland Algorithm

− try to avoid intersection calculations as much as possible
− based on calculation of binary region codes for each end point of the line

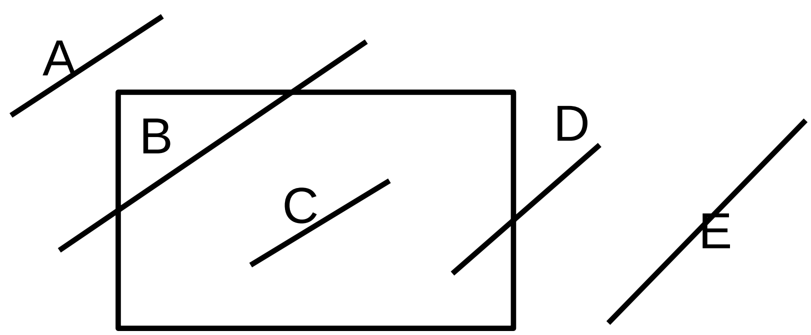
1001	1000	1010
0001	0000	0010
0101	0100	0110

bit 1: left edge
bit 2: right edge
bit 3: bottom edge
bit 4: top edge

bit 1

Calculation of region codes:

bit 1 = 1 if x < xmin
bit 2 = 1 if x > xmax
bit 3 = 1 if y < ymin
bit 4 = 1 if y > ymax

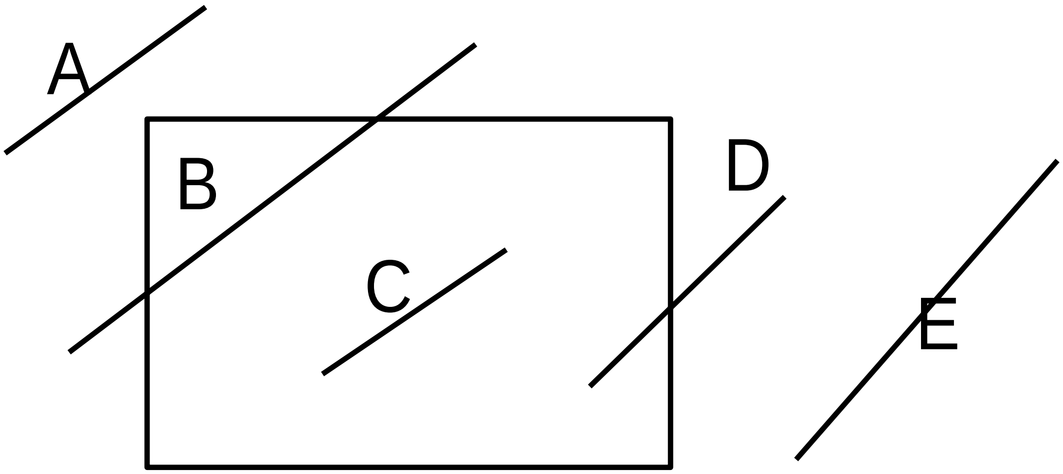


Trivial accept or reject:

if (code (P0) | code(P1)) = 0 −> accept   (e.g., C)

if (code (P0) & code(P1)) !=0 −> reject   (e.g., E)

bitwise AND



$Code(P0) \& Code(P1) \neq 0$

E 0010 0010

$Code(P0) = Code(P1) = 0$

C 0000 0000

$Code(P0) \& Code(P1) = 0$

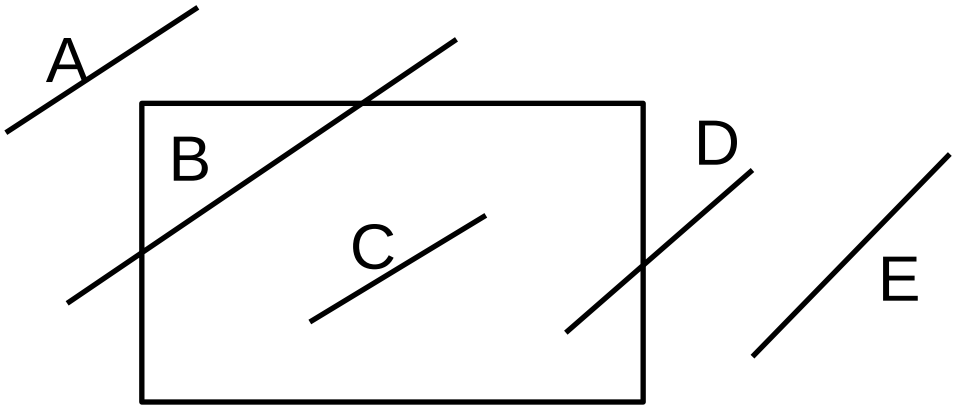
B 0001 1000

A 0001 1000

$Code(P0) \neq 0$, $Code(P1) = 0$; or vice versa

D 0000 0010

Cohen−Sutherland Algorithm (cont'd)



Algorithm:

1. Compute the binary region code for P0 and P1
2. Loop

- *Test for trivial reject or accept (exit?)* (C,E)

- *If (**code(P0)!=0 and code(P1)=0**)*

  One point is inside and the other is outside.
  The line must be shortened. (D)
  One or two intersections must be computed.

  else

  *if (**code(P0)&code(P1)=0**)*

  Both endpoints are outside. (A,B)
  Intersect with one of the sides of the window,
  and check the code of the resulting point

Liang−Barsky Line Clipping

− for parametric lines

− clip a 2d line against to a rectangle
or an arbitrary convex polygon in the plane

− can be generalized to 3d

Parametric equation of a line:

$$x = (1-u)x_1 + ux_2$$
$$y = (1-u)y_1 + uy_2 \qquad \text{where } 0 \leq u \leq 1$$

or

$$x = x_1 + u\,dx$$
$$y = y_1 + u\,dy \qquad \text{where } dx = x_2 - x_1,\ dy = y_2 - y_1$$

$$x_{min} \leq x_1 + u\,dx \leq x_{max}$$
$$y_{min} \leq y_1 + u\,dy \leq y_{max}$$

each of the 4 inequalities can be expressed as

$u p_k \leq q_k$ for $k=1,2,3,4$

where

$$p_1 = -dx, \quad q_1 = x_1 - x_{min}$$
$$p_2 = dx, \quad q_2 = x_{max} - x_1$$
$$p_3 = -dy, \quad q_3 = y_1 - y_{min}$$
$$p_4 = dy, \quad q_4 = y_{max} - y_1$$

Liang−Barsky Line Clipping (cont'd)

$p_1 = -dx, \quad q_1 = x_1 - x_{min}$
$p_2 = dx, \quad q_2 = x_{max} - x_1$
$p_3 = -dy, \quad q_3 = y_1 - y_{min}$
$p_4 = dy, \quad q_4 = y_{max} - y_1$

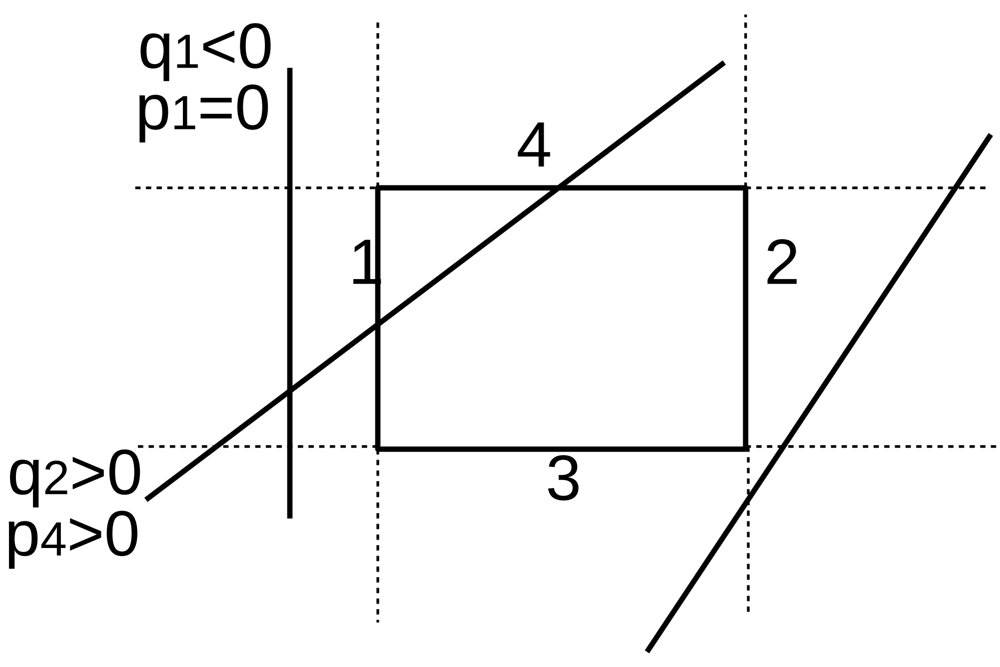
$p_k = 0$ the line is parallel to one of the clipping boundaries

$q_k < 0$ the line is completely outside the boundary

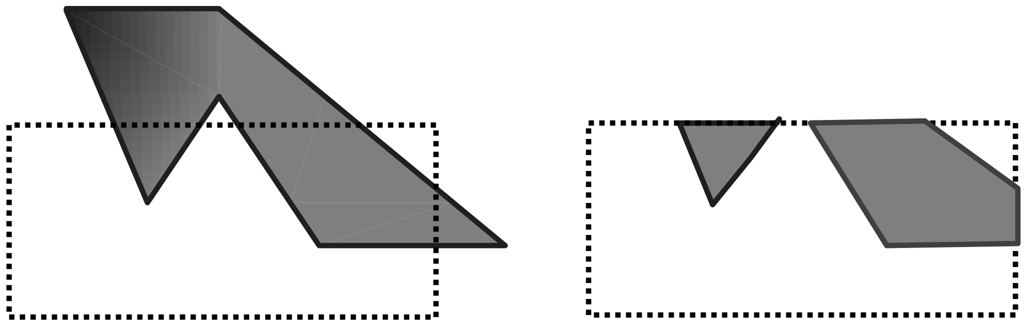
$p_k > 0$ the line proceeds from the inside to the outside

$q_k \geq 0$ the line is inside the parallel clipping boundary

.....

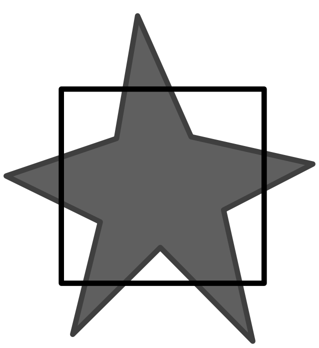


Polygon Clipping

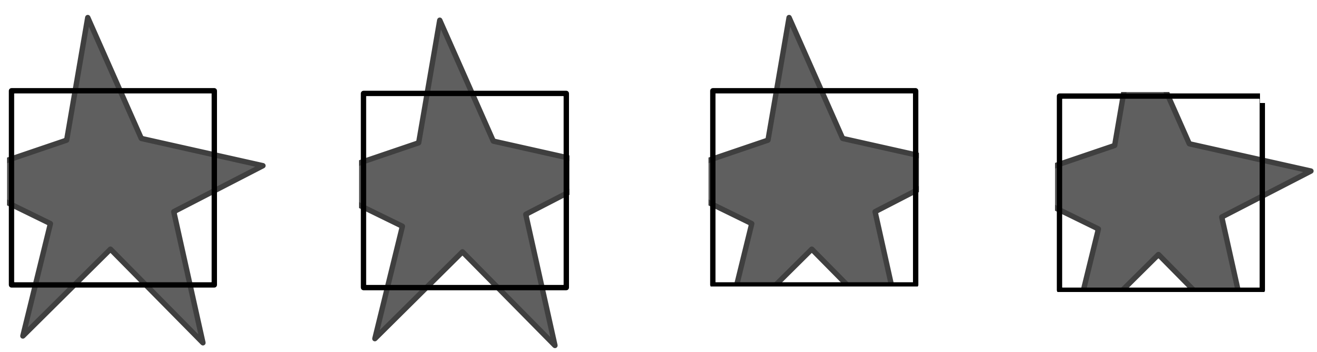


The output of a polygon clipper should be a sequence of vertices that defines the clipped polygon boundaries

Sutherland−Hodgeman Polygon Clipping

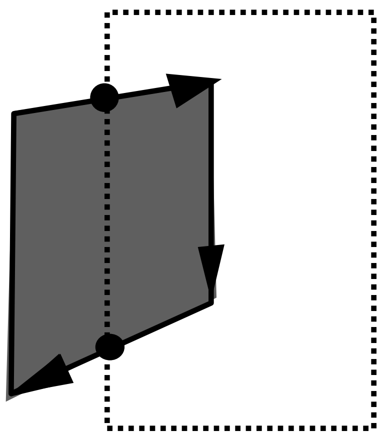


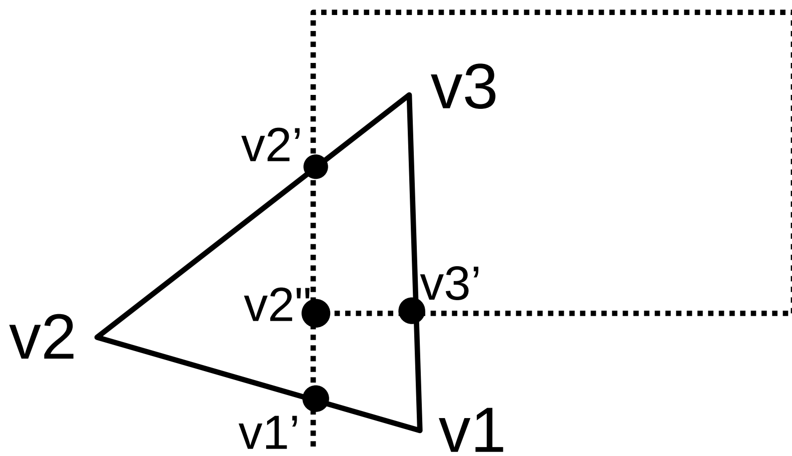
- clip the polygon to each of the window boundaries in succession
- edge and vertex defintions of the polygon are updated accordingly



4 cases when processing vertices in sequence:

1. out −> in : save intersection point and current point
2. in −> in : save current point
3. in −> out : save intersection point
4. out −> out : save none





→ Left → Right → Bottom → Top →

v1	v1	v1		
v2	v1’	v1’		
v3	v2’, v3	v2’, v3	v2'', v2’	v2'', v2’
			v3	v3
			v3’	v3’

Concave case: