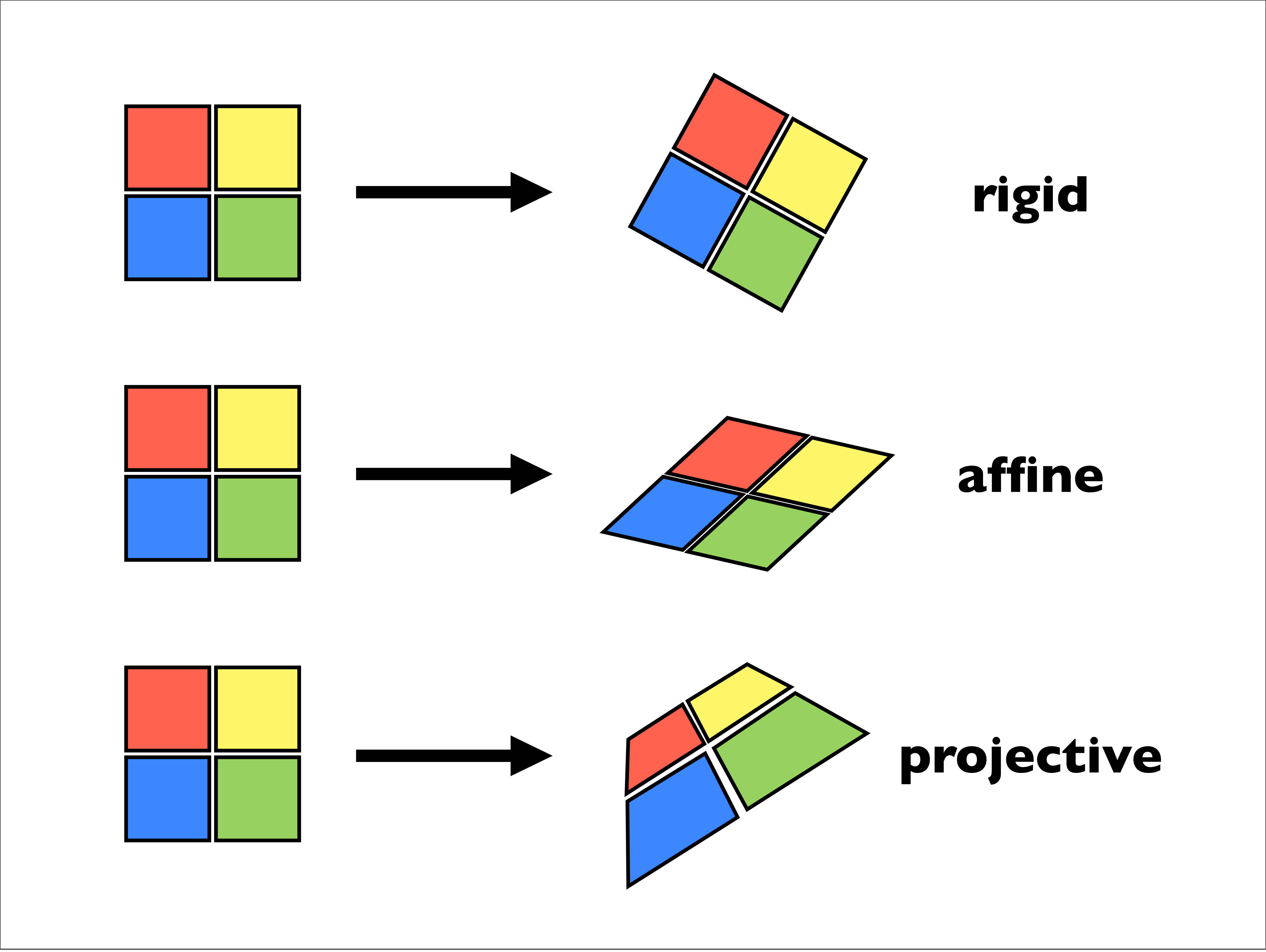
CS130 : Computer Graphics

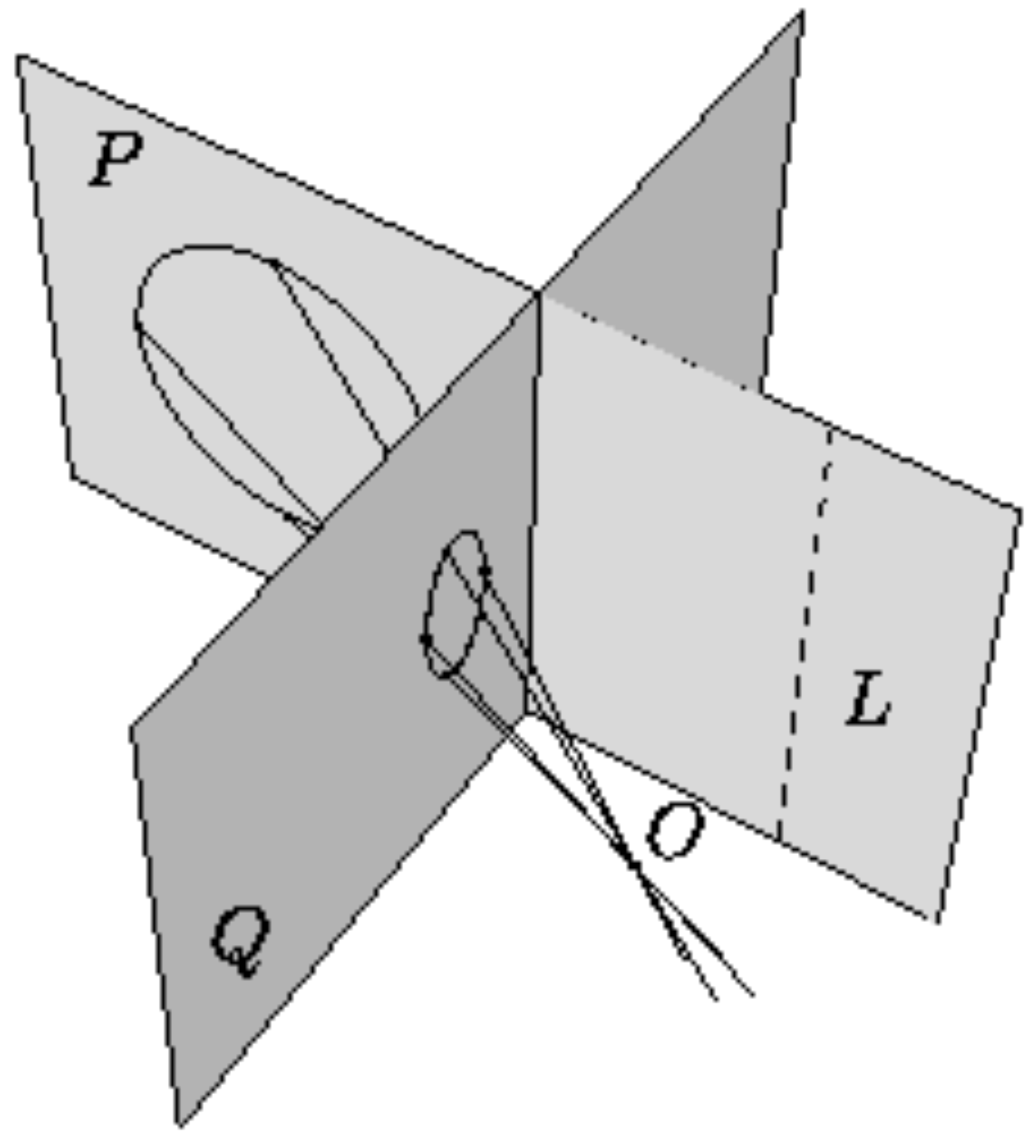
Lecture 10: Perspective Viewing

Tamar Shinar
Computer Science & Engineering
UC Riverside



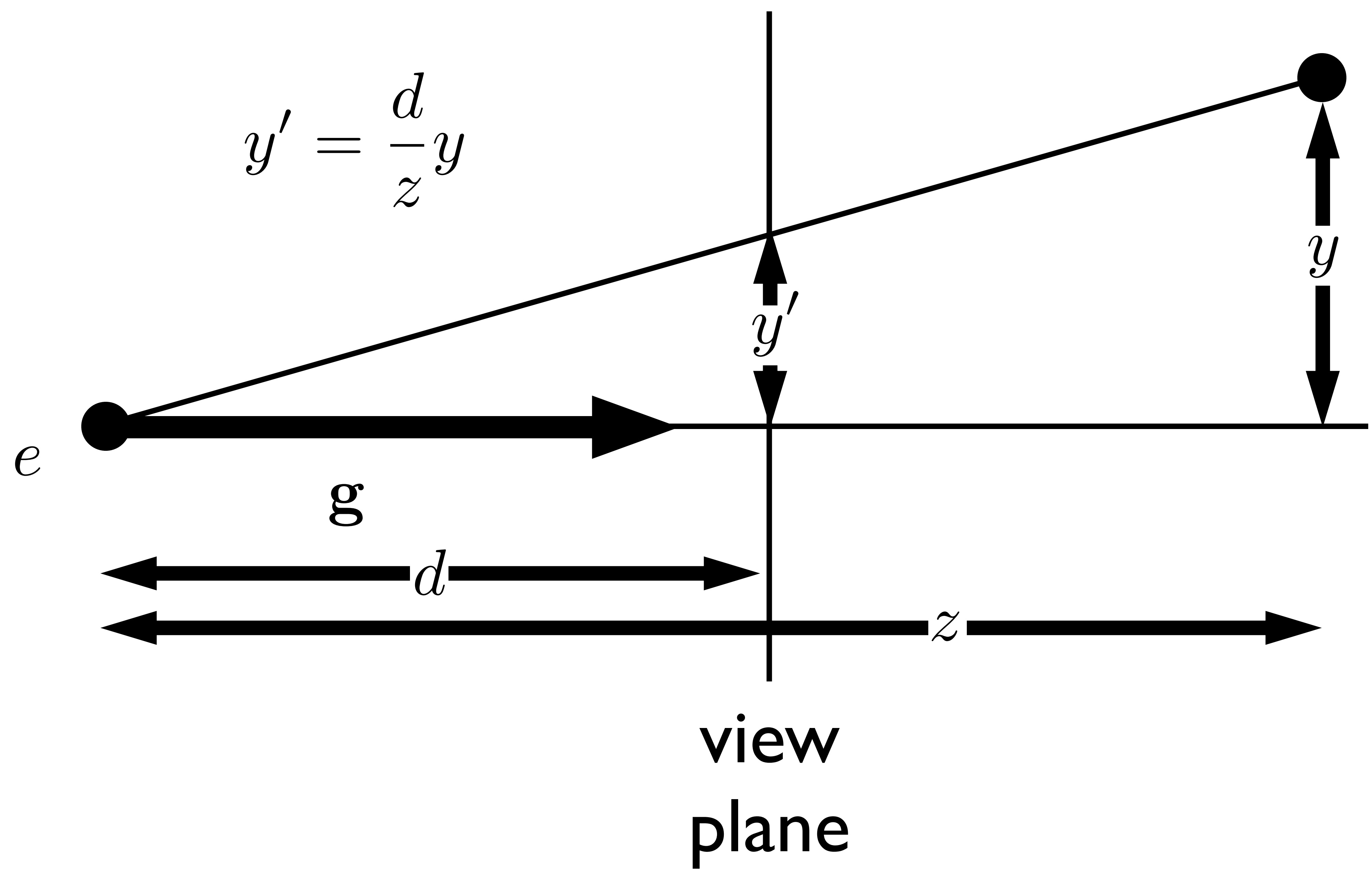
rigid – translation and rotation only – parallel lines and angles are preserved
affine – scaling, shear, translation, rotation – parallel lines preserved, angles **not** preserved
projective – parallel lines and angles **not** preserved

Projective Transformations



[geom.uiuc.edu]

Projective Transformations



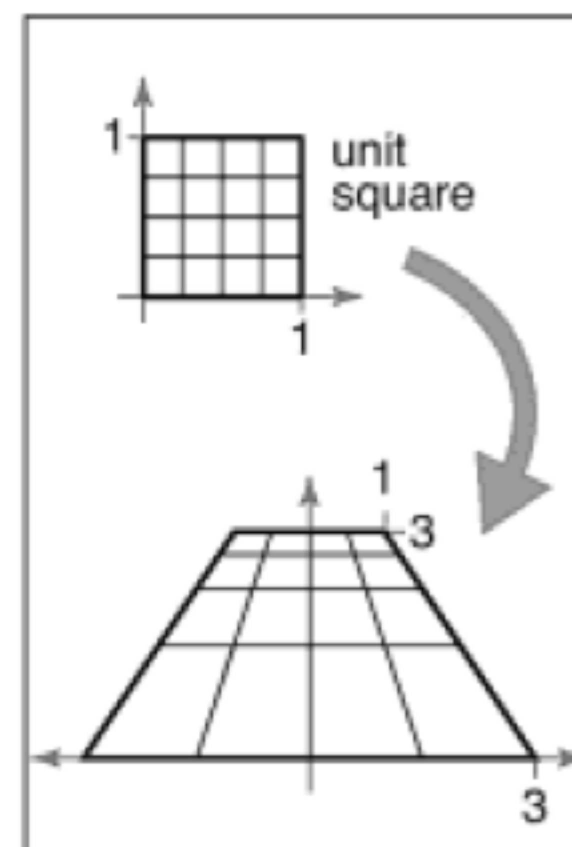
note that the height, **y'**, in **camera space** is proportion to y and inversely proportion to z. We want to be able to specify such a transformation with our **4x4 matrix machinery**

Projective Transformations

$$\begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ w \end{pmatrix} \rightarrow \quad x = \frac{\tilde{x}}{w} \quad y = \frac{\tilde{y}}{w} \quad z = \frac{\tilde{z}}{w}$$

<u>Example:</u>

$$M = \begin{pmatrix} 2 & 0 & -1 \\ 0 & 3 & 0 \\ 0 & \frac{2}{3} & \frac{1}{3} \end{pmatrix}$$



[Shirley, Marschner]

Note: this makes our homogeneous representation for unique only **up to a constant**

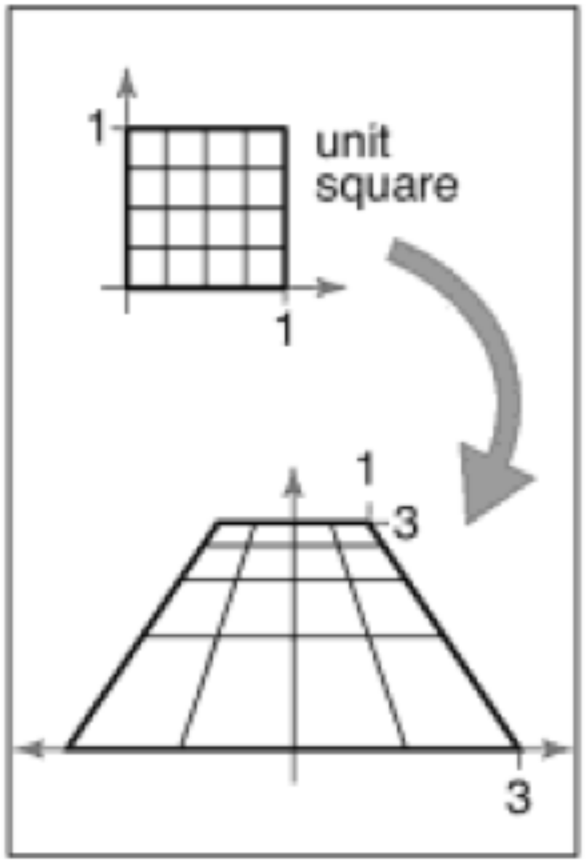
Projective Transformations

$$\begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ w \end{pmatrix} \rightarrow \quad \begin{aligned} x &= \frac{\tilde{x}}{w} \\ y &= \frac{\tilde{y}}{w} \\ z &= \frac{\tilde{z}}{w} \end{aligned}$$

We can now implement perspective projection!

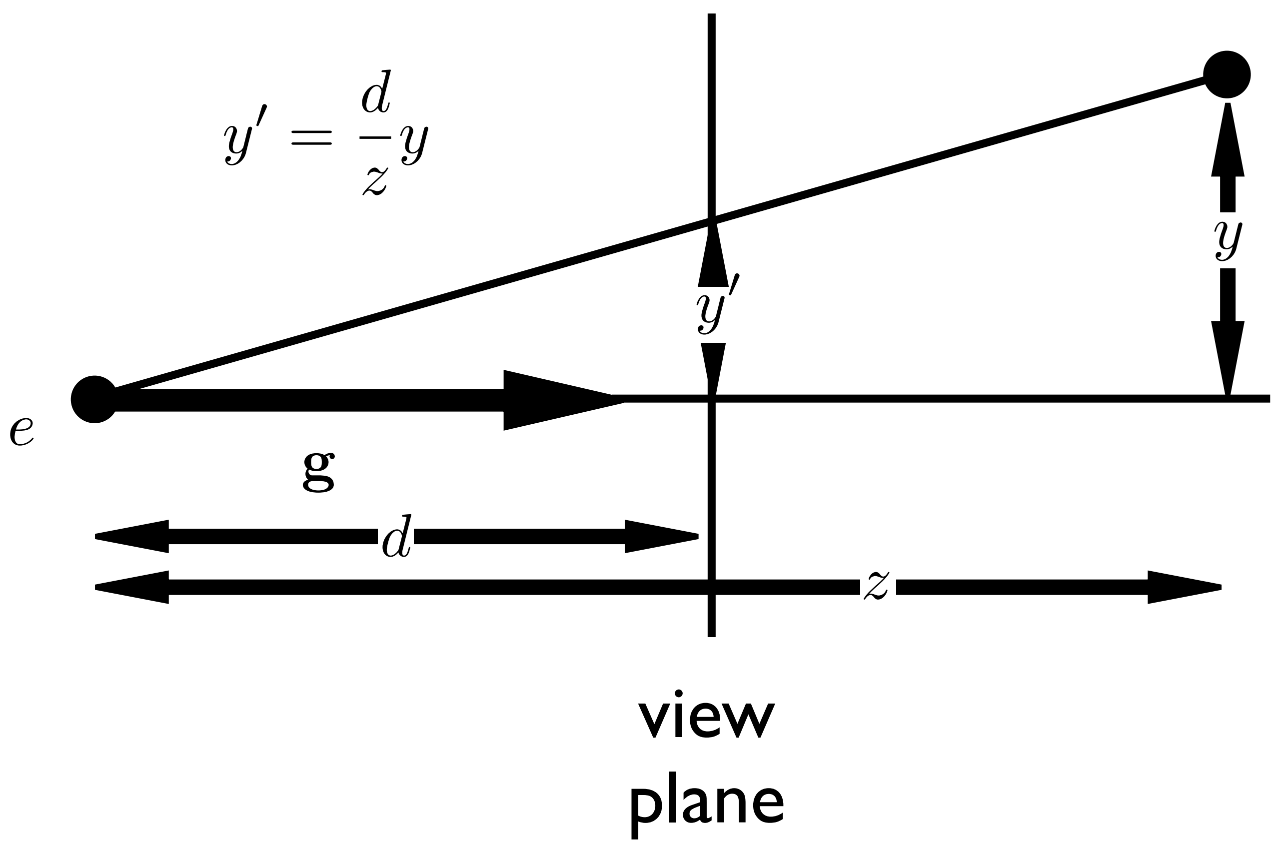
<u>Example:</u>

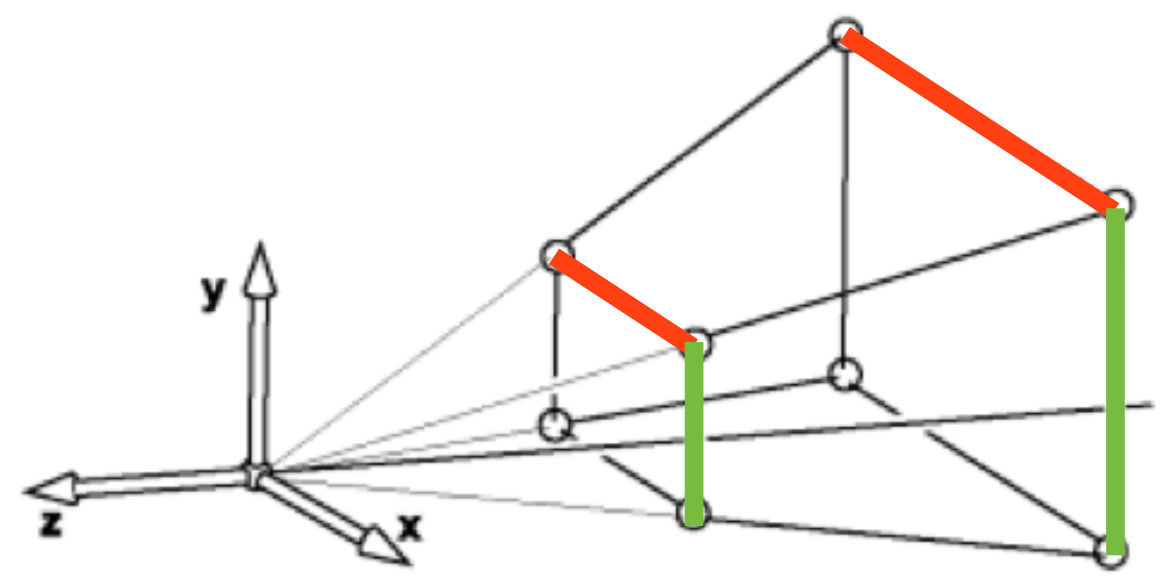
$$M = \begin{pmatrix} 2 & 0 & -1 \\ 0 & 3 & 0 \\ 0 & \frac{2}{3} & \frac{1}{3} \end{pmatrix}$$



[Shirley, Marschner]

Perspective Projection





both x and y get multiplied by d/z

[Shirley, Marschner]

note that both x and y will be transformed

Simple perspective projection

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ z/d \end{pmatrix} \Rightarrow \begin{cases} x' = \frac{d}{z}x \\ y' = \frac{d}{z}y \\ z' = \frac{d}{z}z = d \end{cases}$$

This achieves a simple perspective projection onto the view plane z = d

but we've lost all information about z!

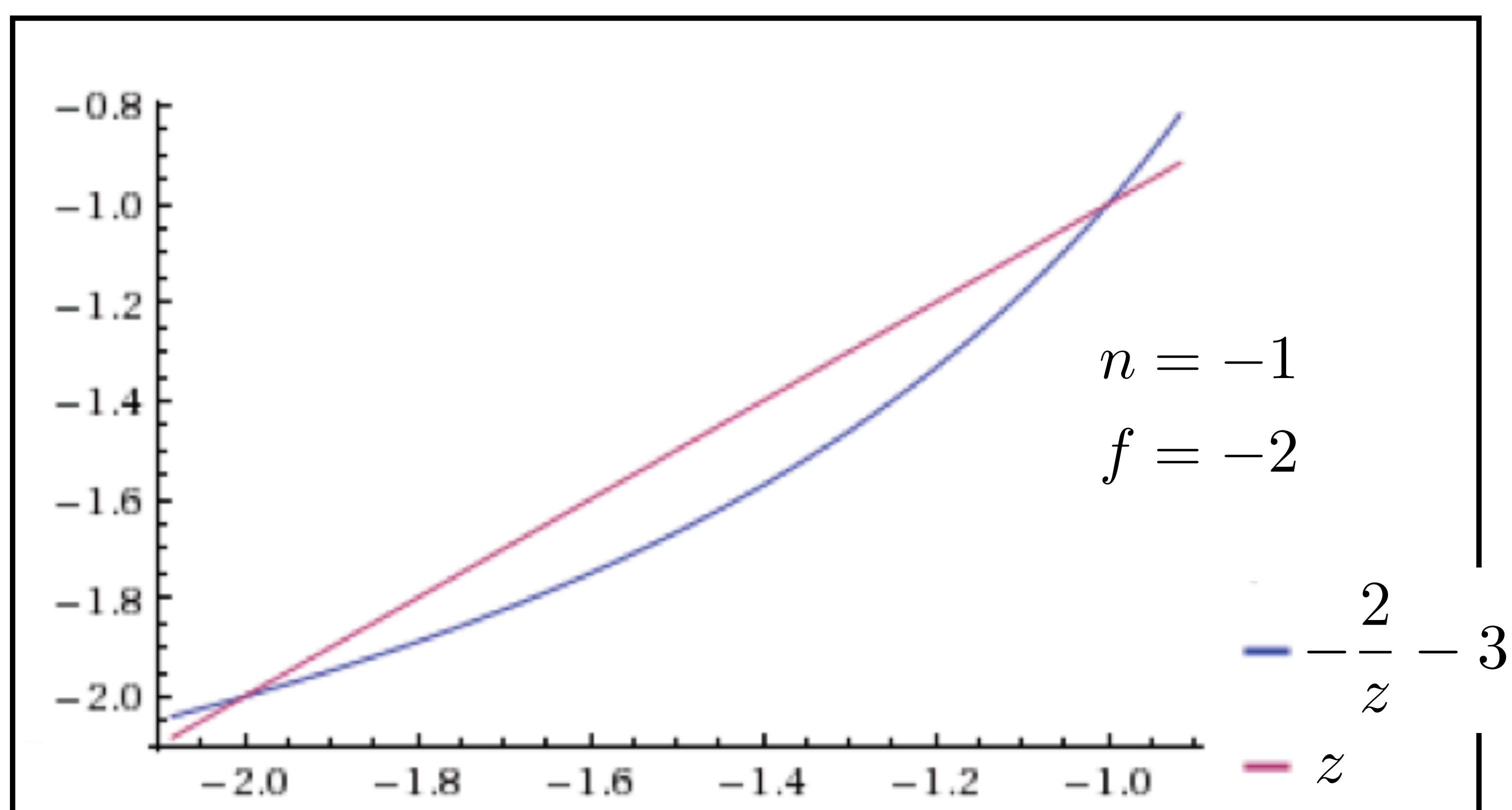
<whiteboard>

This simple projection matrix won't suffice. We need to preserve z information for later hidden surface removal.

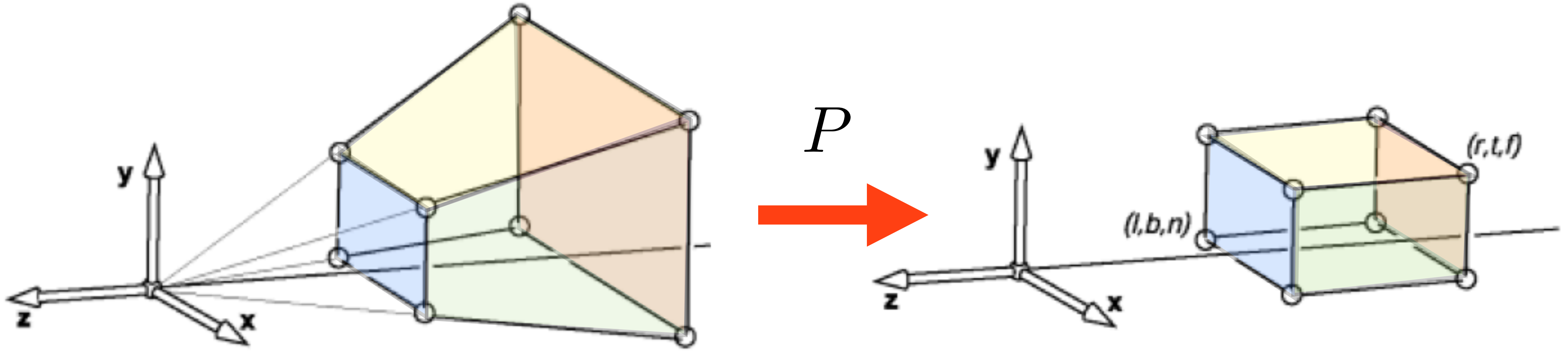
Perspective Projection

$$P = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad z' = (n+f) - \frac{nf}{z}$$

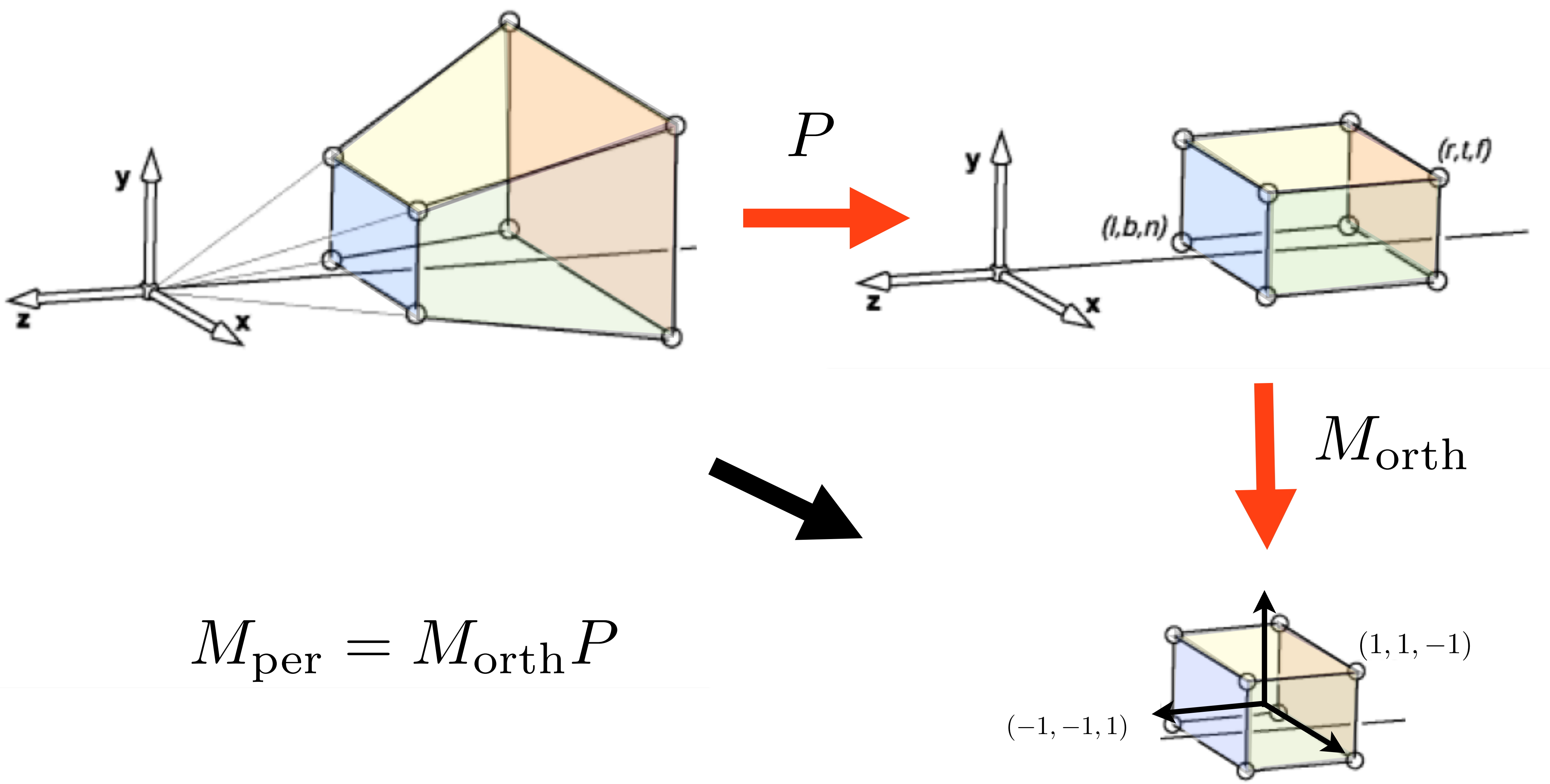
Example:



The perspective transformation does not preserve **z** completely, but it preserves **z = n, f** and is **monotone** (preserves ordering) with respect to z

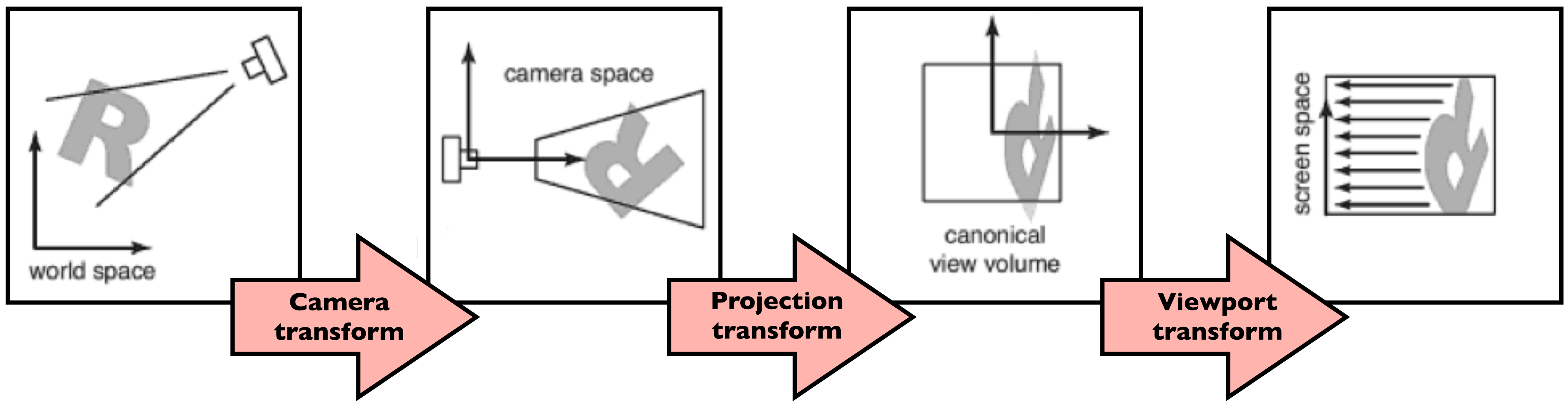


So far we've mapped the view frustrum to a rectangular box. This rectangular box has the same near face as the view frustrum. The far face has been mapped down to the far face of the box. This mapping is given by P.



We need a second mapping to get our points into the canonical view volume. This second mapping is just a mapping from one box to another. So it's given by an orthographic mapping, M_orth. The final perspective transformation is the composition of P and M_orth.

Line drawing algorithm



construct M_{vp} M_{cam}

construct M_{per}

$M = M_{vp} M_{per} M_{cam}$

for each line segment (a_i, b_i) do

$\quad \mathbf{p} = M\mathbf{a}_i$

$\quad \mathbf{q} = M\mathbf{b}_i$

drawline $(x_p/w_p, y_p/w_p, x_q/w_p, y_q/w_p)$

draw lines specified in <u>world</u> space

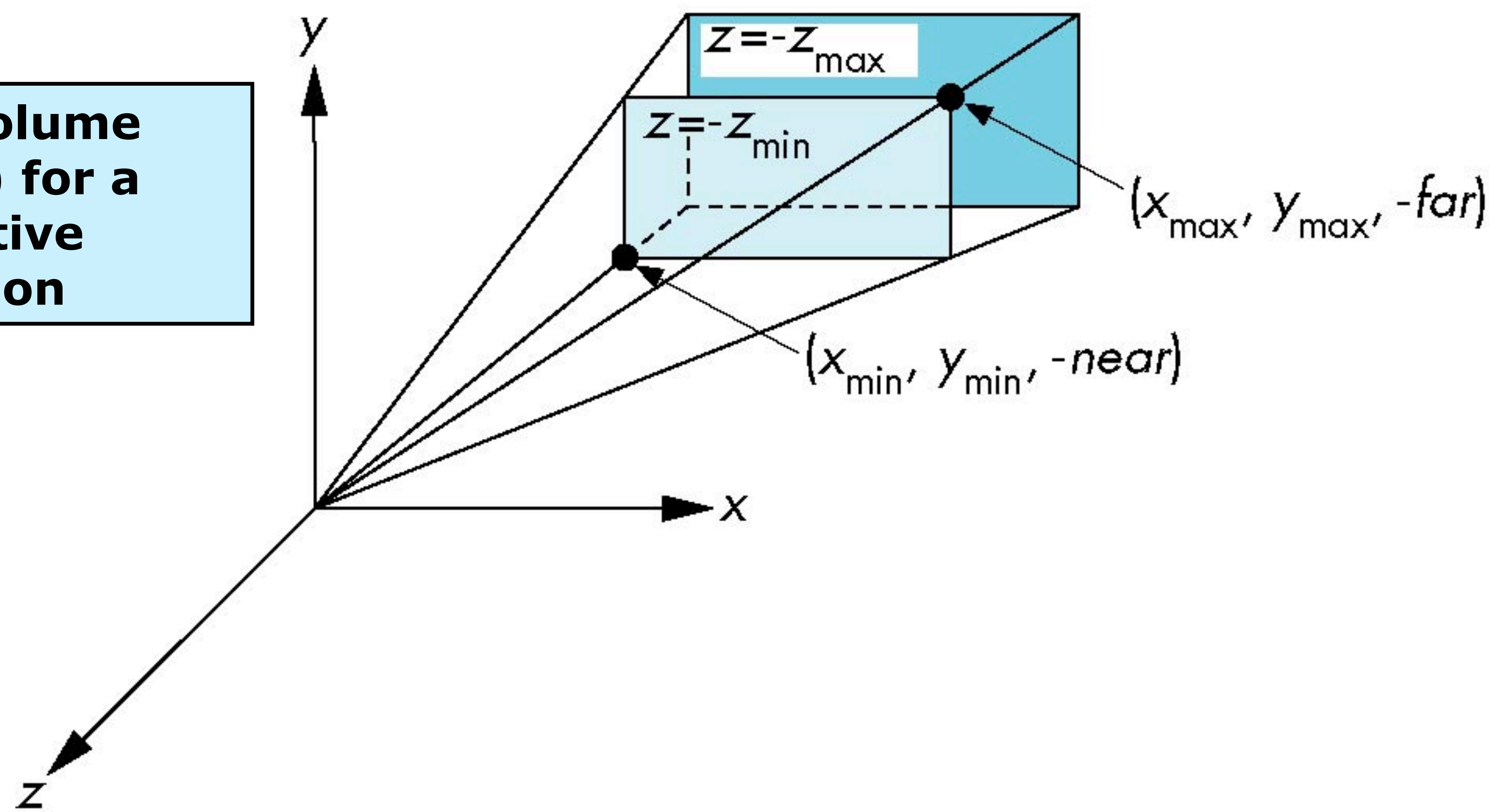
[Shirley, Marschner]

Note the two lines that have changed: 1. we put our perspective transformation into the overall transformation matrix M. 2. When we call the drawline function, we have to divide the x and y coordinates by the w coordinate.

OpenGL Perspective Viewing

```
glFrustum(xmin,xmax,ymin,ymax,near,far)
```

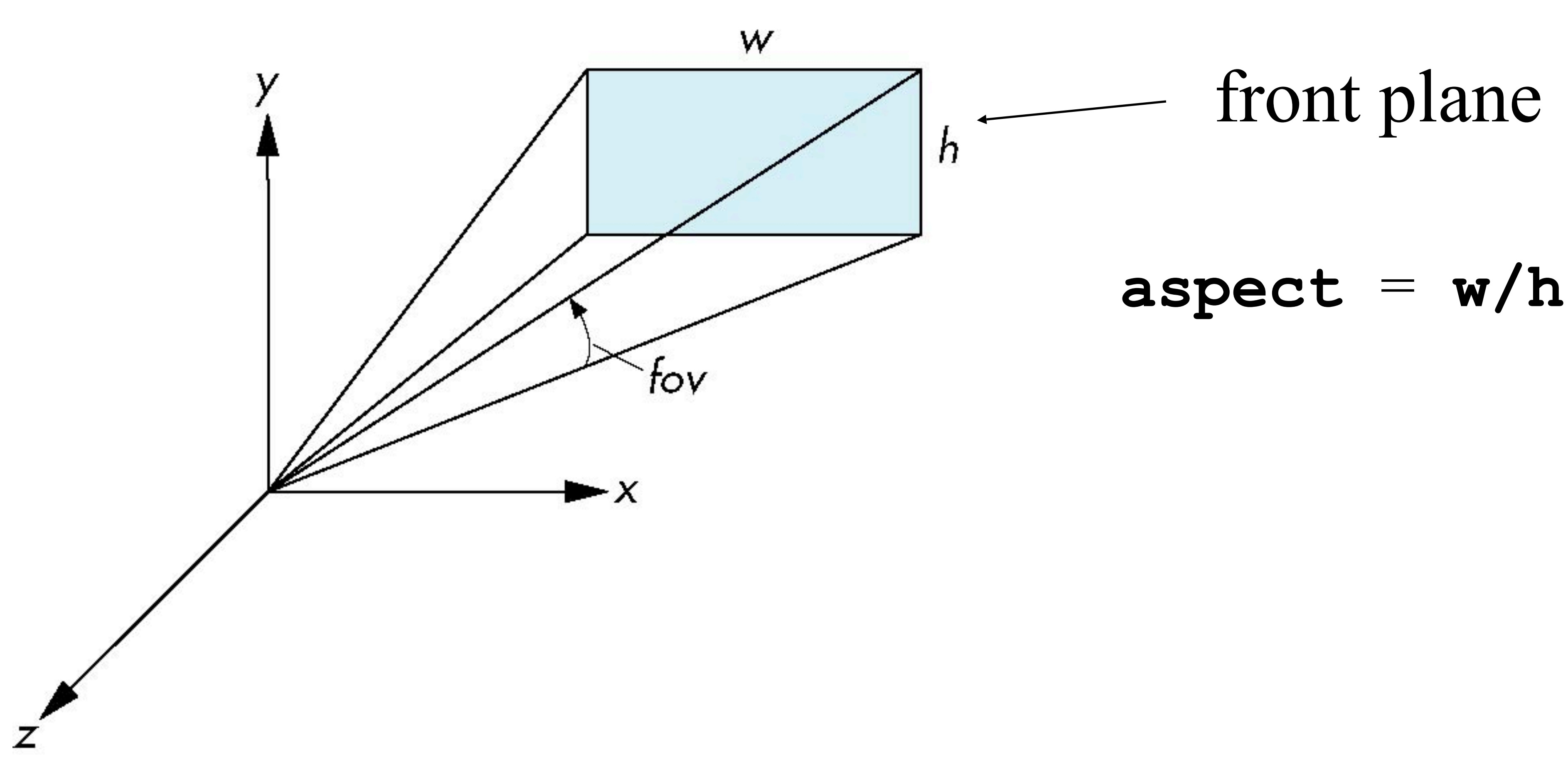
Clipping volume (frustrum) for a perspective projection



Here’s how you set up a perspective view in OpenGL. Note that near and far are both negative, but you pass their absolute values to OpenGL.

Using Field of View

With **`glFrustum`** it is often difficult to get the desired view **`gluPerpective(fovy, aspect, near, far)`** often provides a better interface



Sometimes it's more convenient to just give an angle, the field-of-view, and an aspect ratio, instead of l, r, t, b. The glu library provides such a function. It will figure out l, r, t, b, and call glFrustrum for you.