CS130 : Computer Graphics

Lecture 10: Perspective Viewing (cont.)

Tamar Shinar
Computer Science & Engineering
UC Riverside

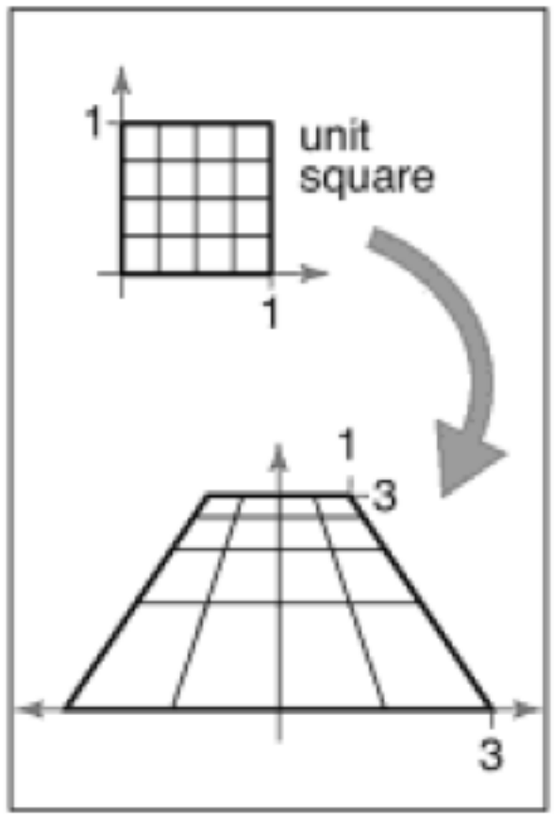
Projective Transformations

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

We can now implement perspective projection!

Example:

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

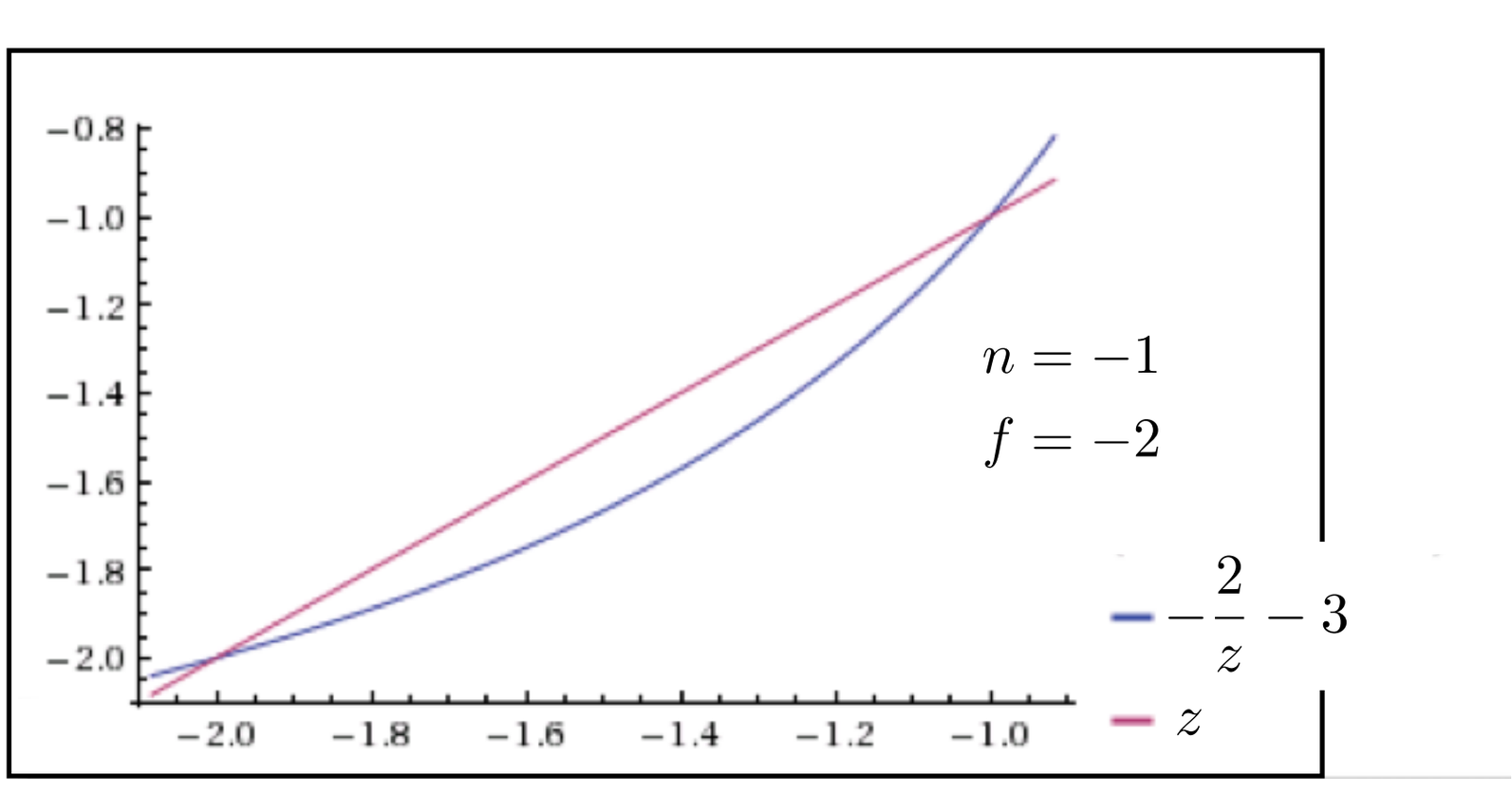


[Shirley, Marschner]

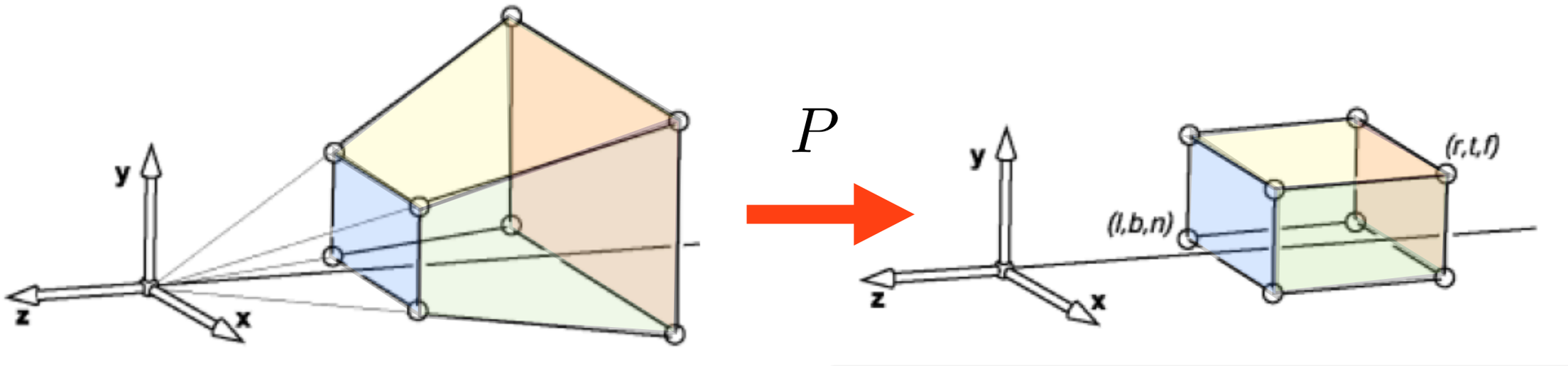
Perspective Projection

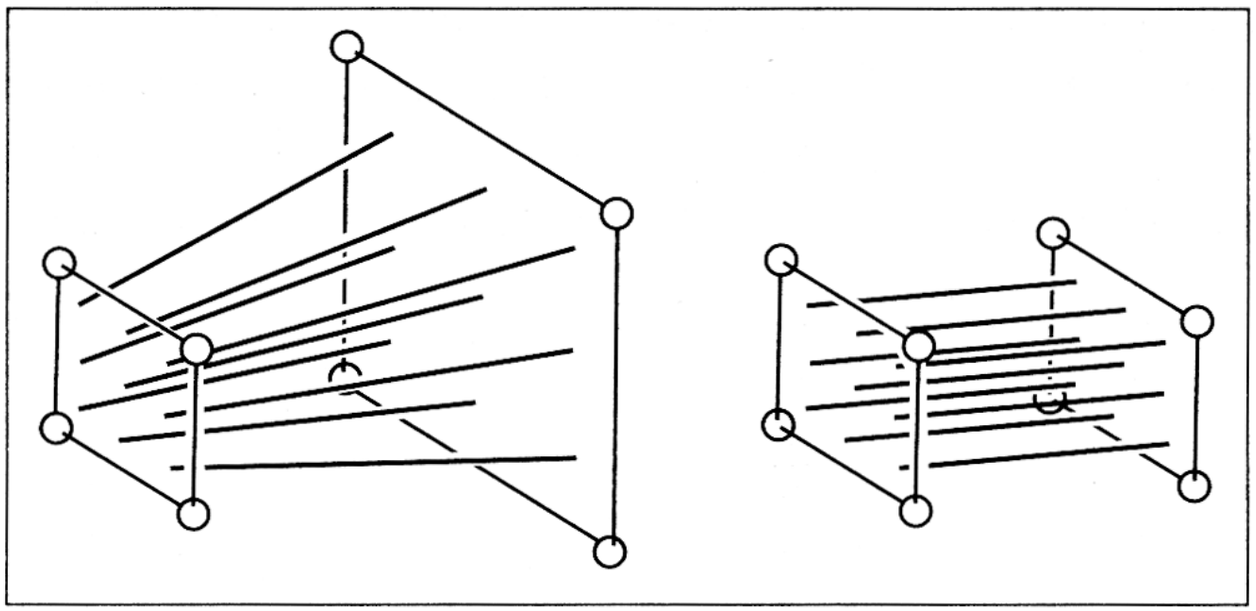
$$P = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad z' = (n+f) - \frac{nf}{z}$$

Example:



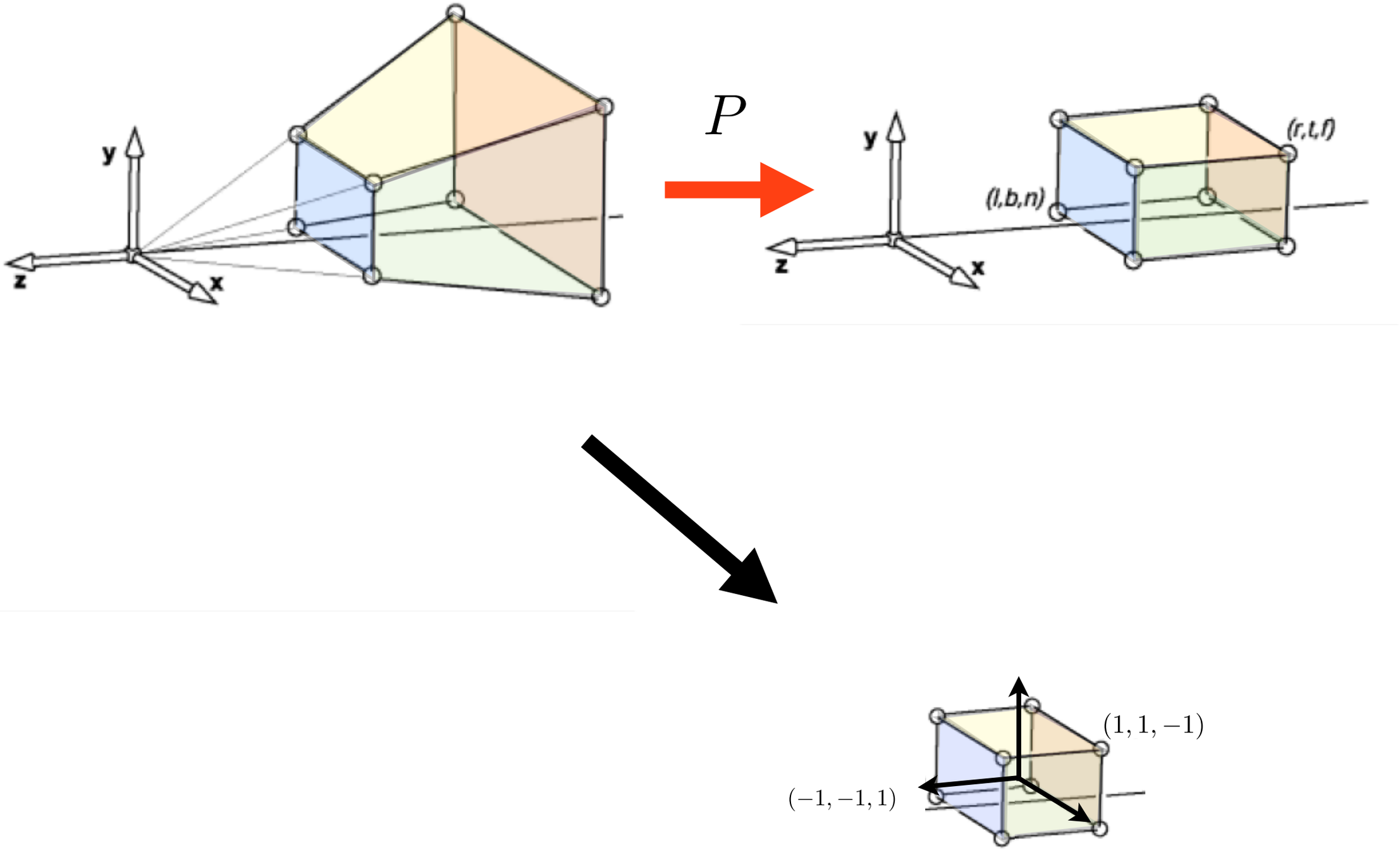
The perspective transformation does not preserve **z** completely, but it preserves **z = n, f** and is **monotone** (preserves ordering) with respect to z



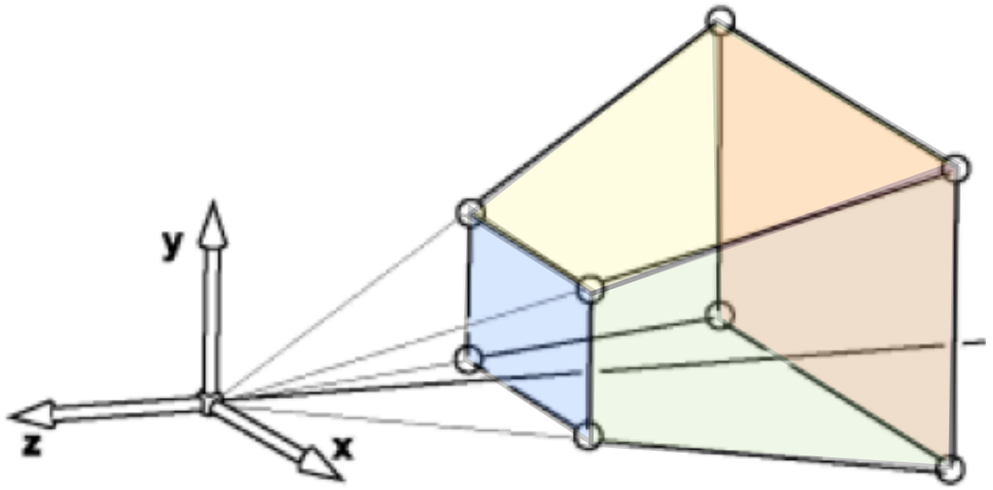


[Shirley, Marschner]

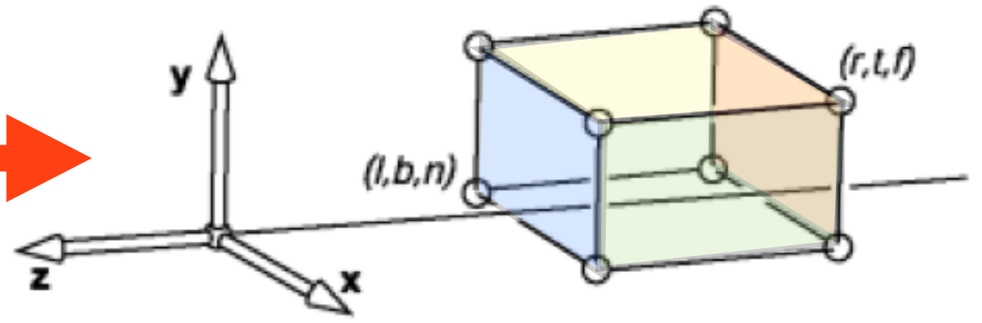
So far we've mapped the view frustrum to a rectangular box. This rectangular box has the same near face as the view frustrum. The far face has been mapped down to the far face of the box. This mapping is given by P. The bottom figure shows how lines in the view frustrum get mapped to the rect. box.



We’re not quite done yet thought, because the projection transform should map the view frustrum to the canonical view volume.

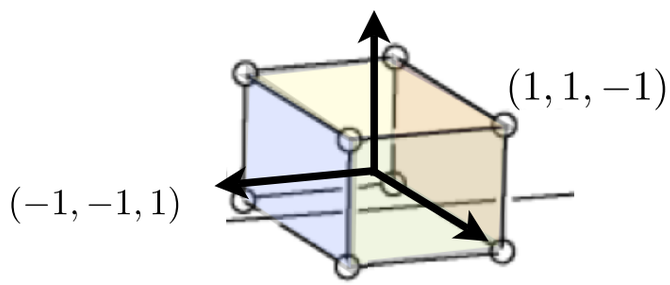


P



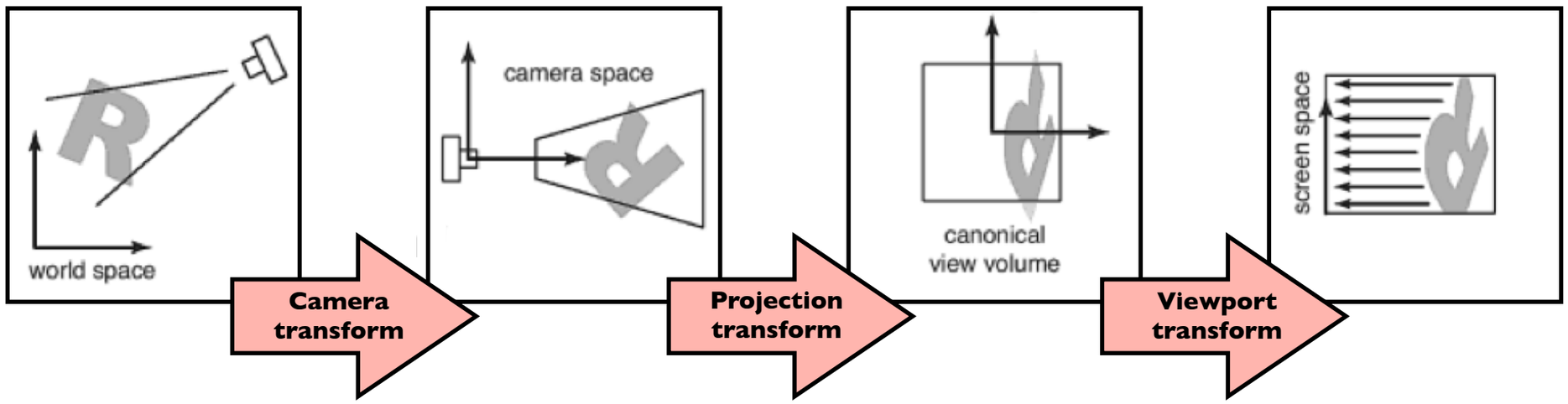
M_{orth}

$$M_{\text{per}} = M_{\text{orth}} P$$



We need a second mapping to get our points into the canonical view volume. This second mapping is 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

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

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

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

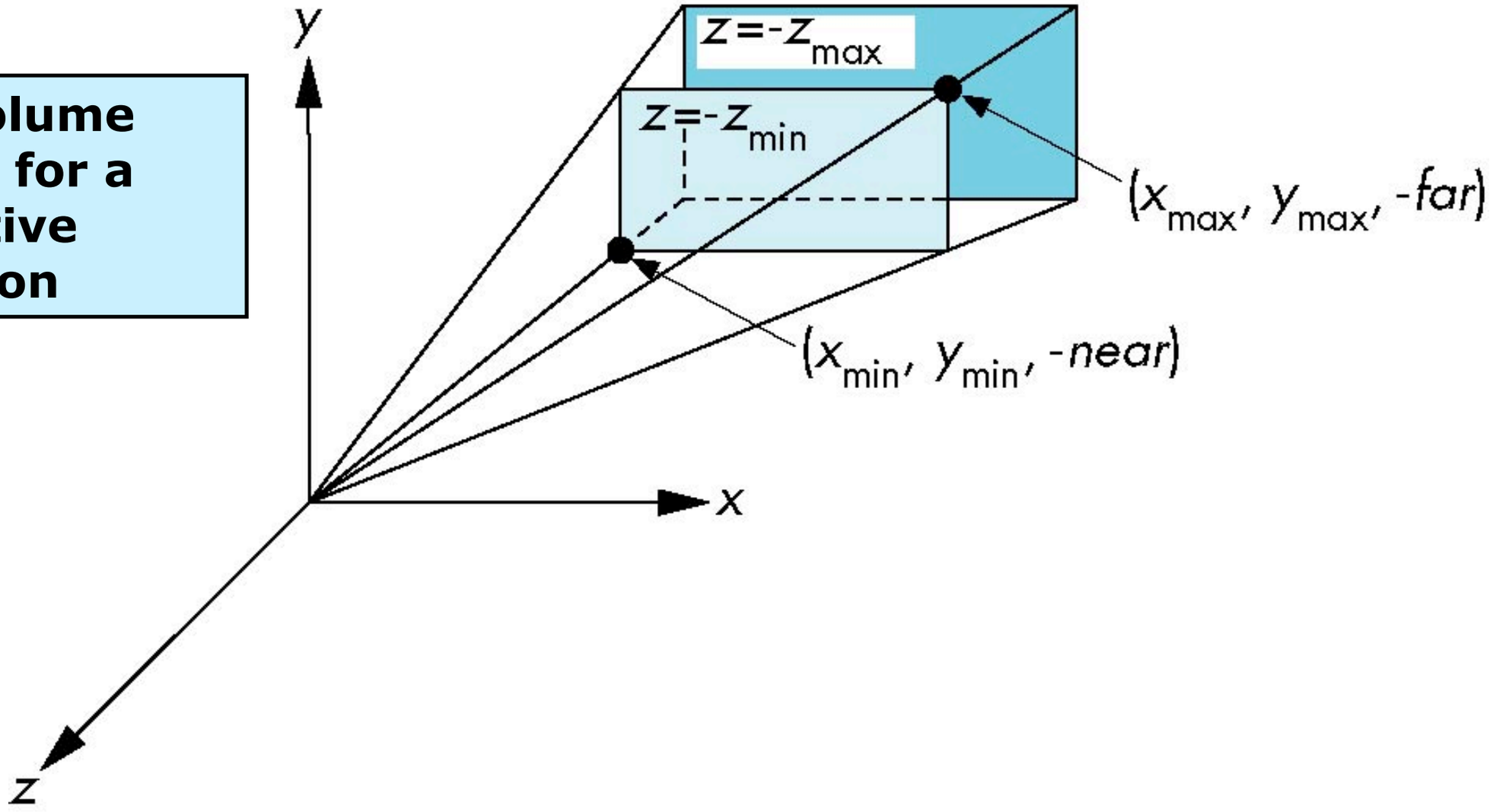
[Shirley, Marschner]

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)`

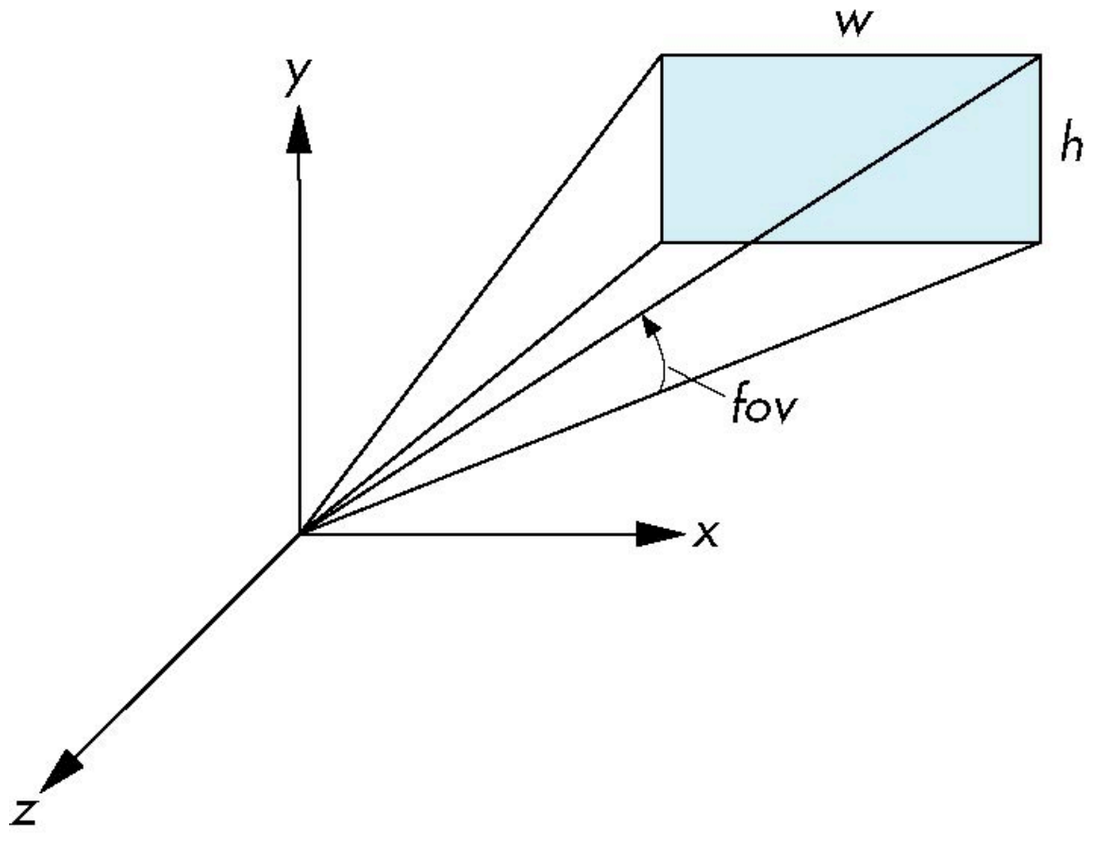
Clipping volume (frustrum) for a perspective projection



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. Another difference with OpenGL is that it uses a left-handed coordinates system after the camera transformations.

Using Field of View

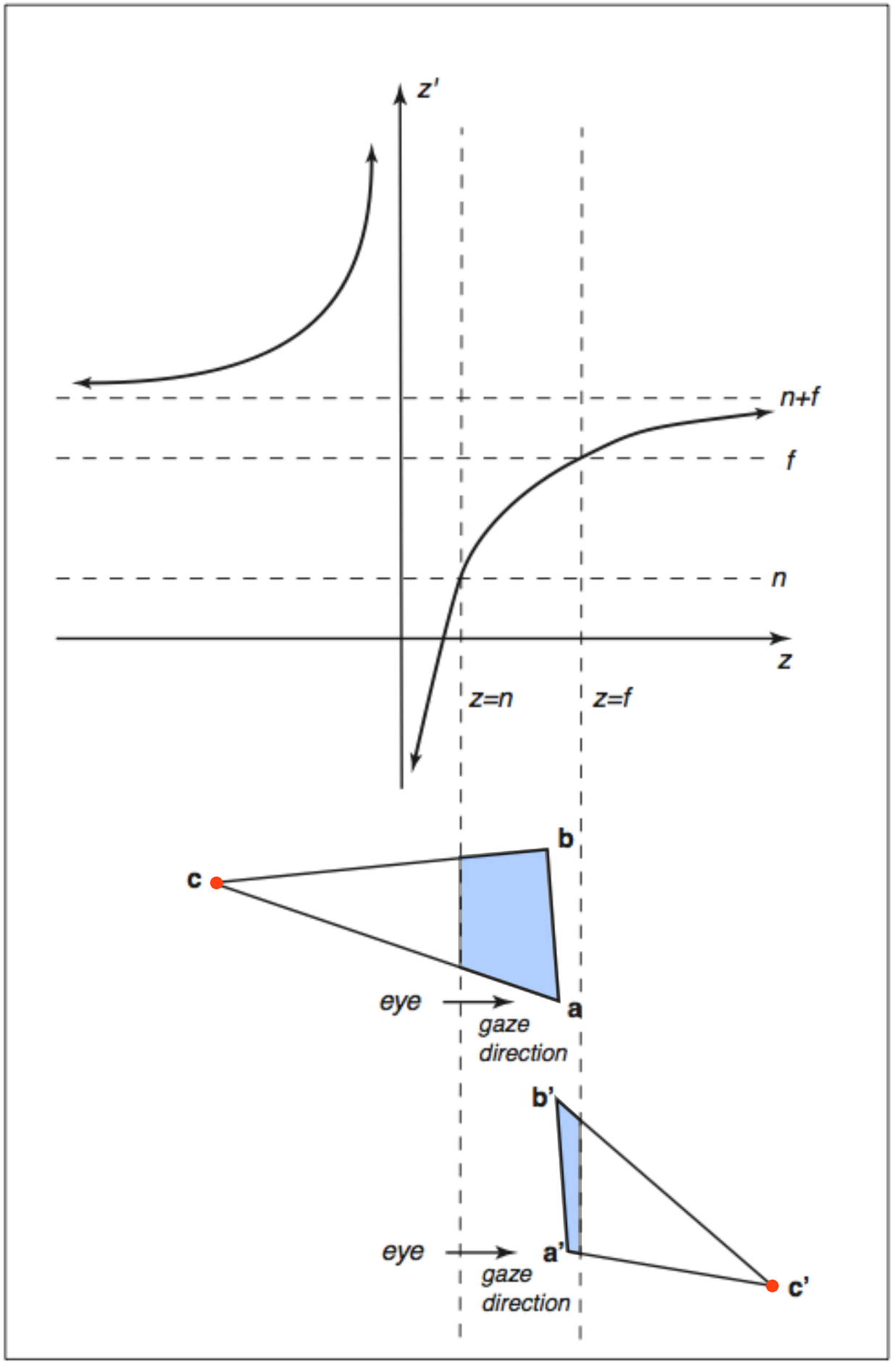
With **`glFrustum`** it is often difficult to get the desired view **`gluPerpective(fovy, aspect, near, far)`** often provides a better interface



front plane

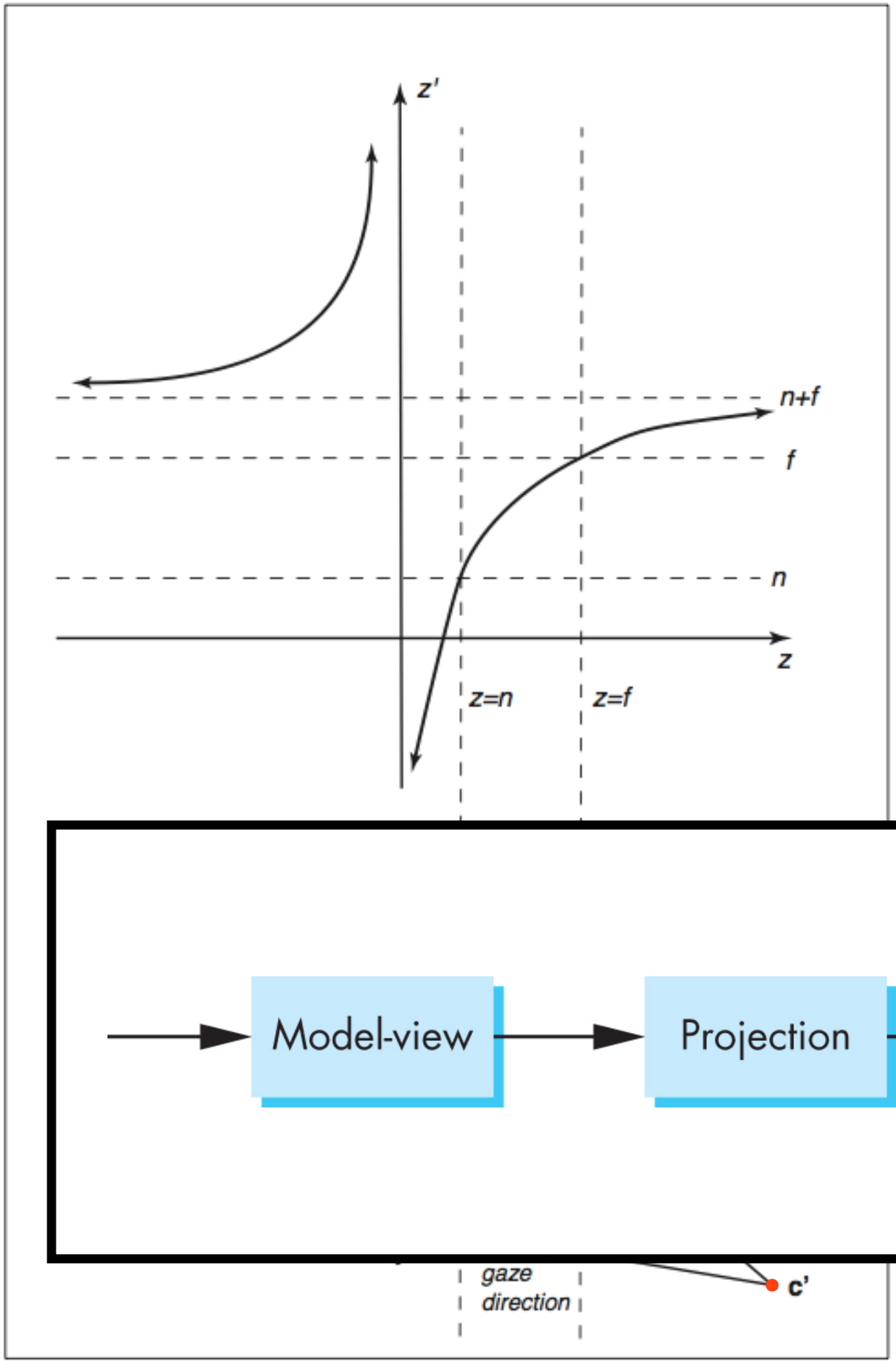
`aspect = w/h`

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.



[Shirley, Marschner]

Clipping after the perspective transformation can cause problems



[Shirley, Marschner]

OpenGL clips **after** projection and **before** perspective division

$$-w \leq x \leq w$$

$$-w \leq y \leq w$$

$$-w \leq z \leq w$$