3D Viewing

CS 4620 Lecture 9

Viewing, backward and forward

- **So far have used the backward approach to viewing**
  - start from pixel
  - ask what part of scene projects to pixel
  - explicitly construct the ray corresponding to the pixel
- **Next will look at the forward approach**
  - start from a point in 3D
  - compute its projection into the image
- **Central tool is matrix transformations**
  - combines seamlessly with coordinate transformations used to position camera and model
  - ultimate goal: single matrix operation to map any 3D point to its correct screen location.

Forward viewing

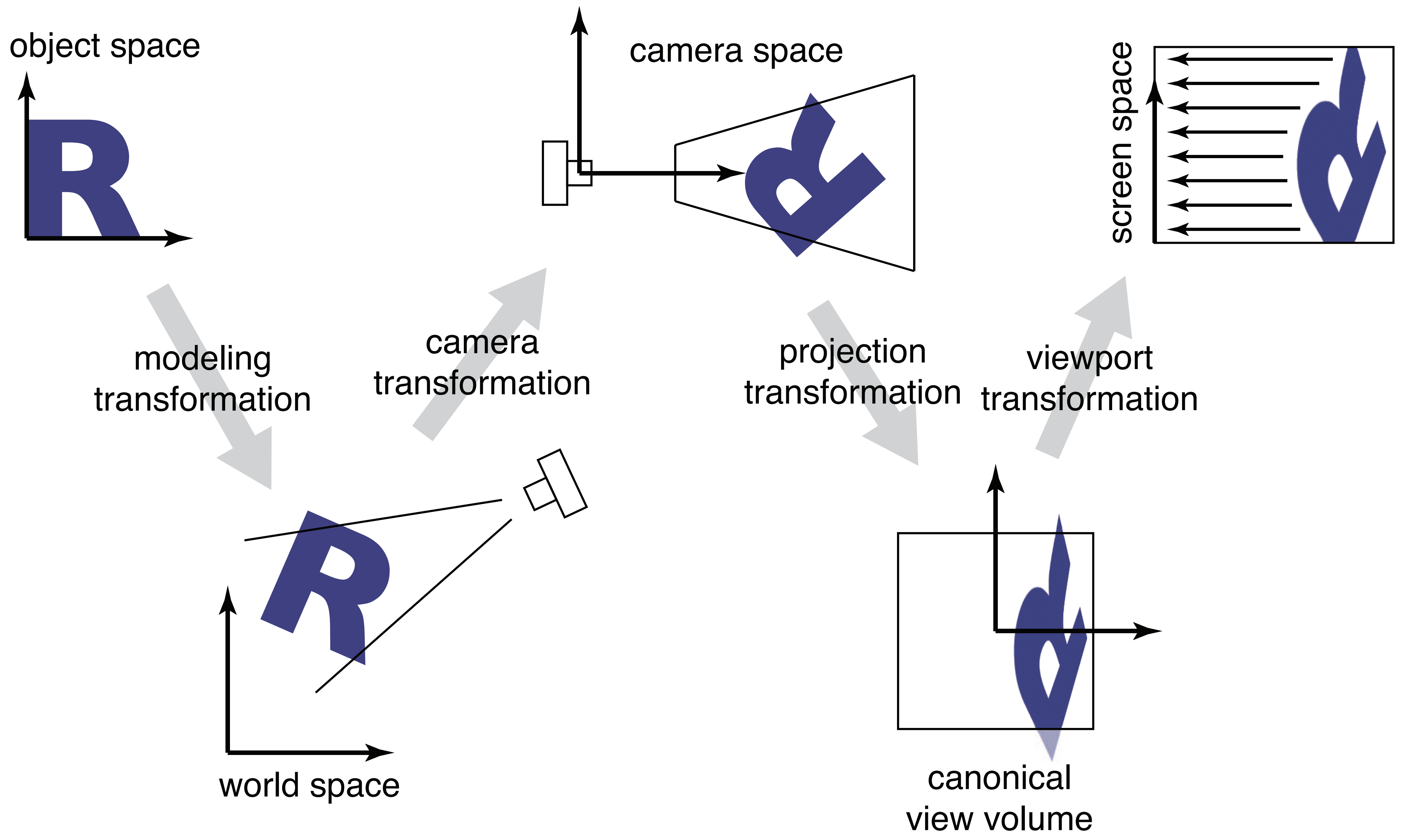
- **Would like to just invert the ray generation process**
- **Problem 1: ray generation produces rays, not points in scene**
- **Inverting the ray tracing process requires division for the perspective case**

Mathematics of projection

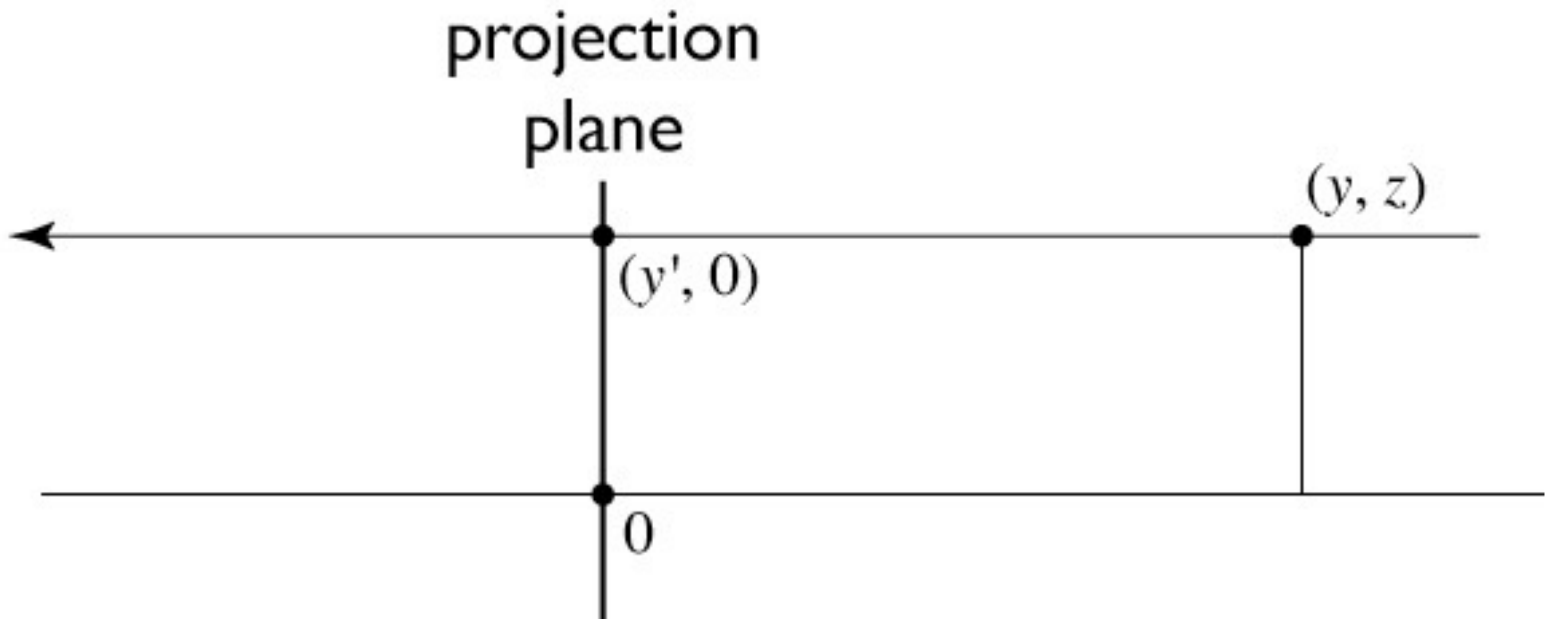
- **Always work in eye coords**
  - assume eye point at **0** and plane perpendicular to ***z***
- **Orthographic case**
  - a simple projection: just toss out ***z***
- **Perspective case: scale diminishes with** *z*
  - and increases with ***d***

Pipeline of transformations

- **Standard sequence of transforms**



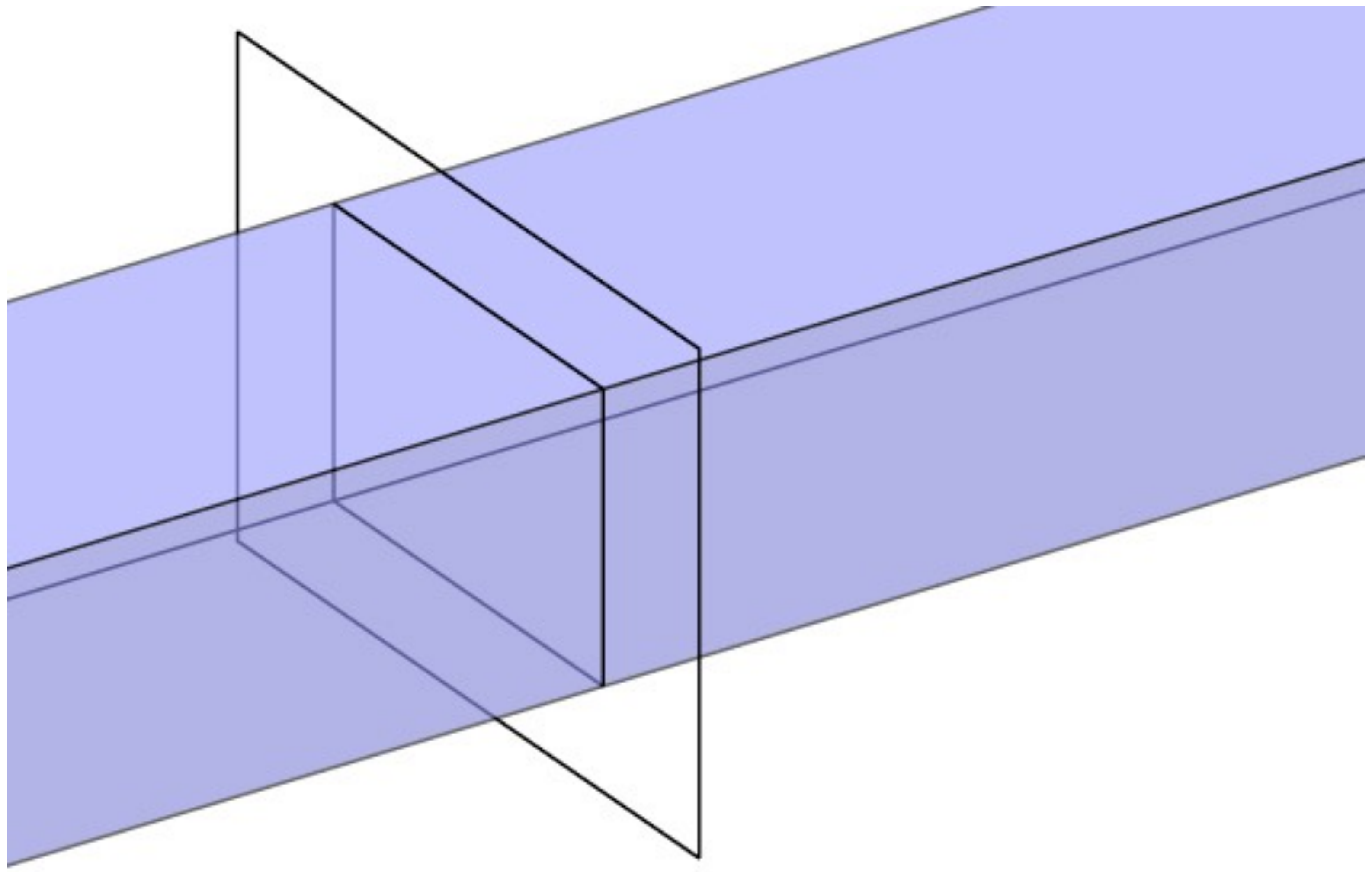
Parallel projection: orthographic



to implement orthographic, just toss out z:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

View volume: orthographic



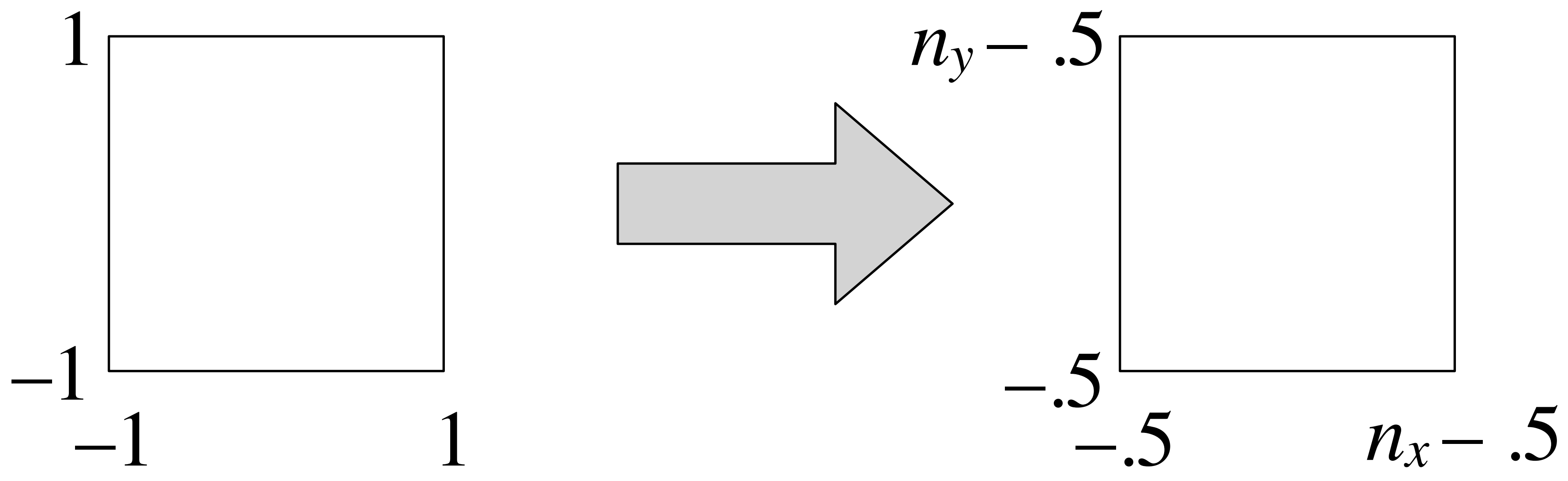
Viewing a cube of size 2

- **Start by looking at a restricted case: the *canonical view volume***
- **It is the cube $[-1,1]^3$, viewed from the *z* direction**
- **Matrix to project it into a square image in $[-1,1]^2$ is trivial:**

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

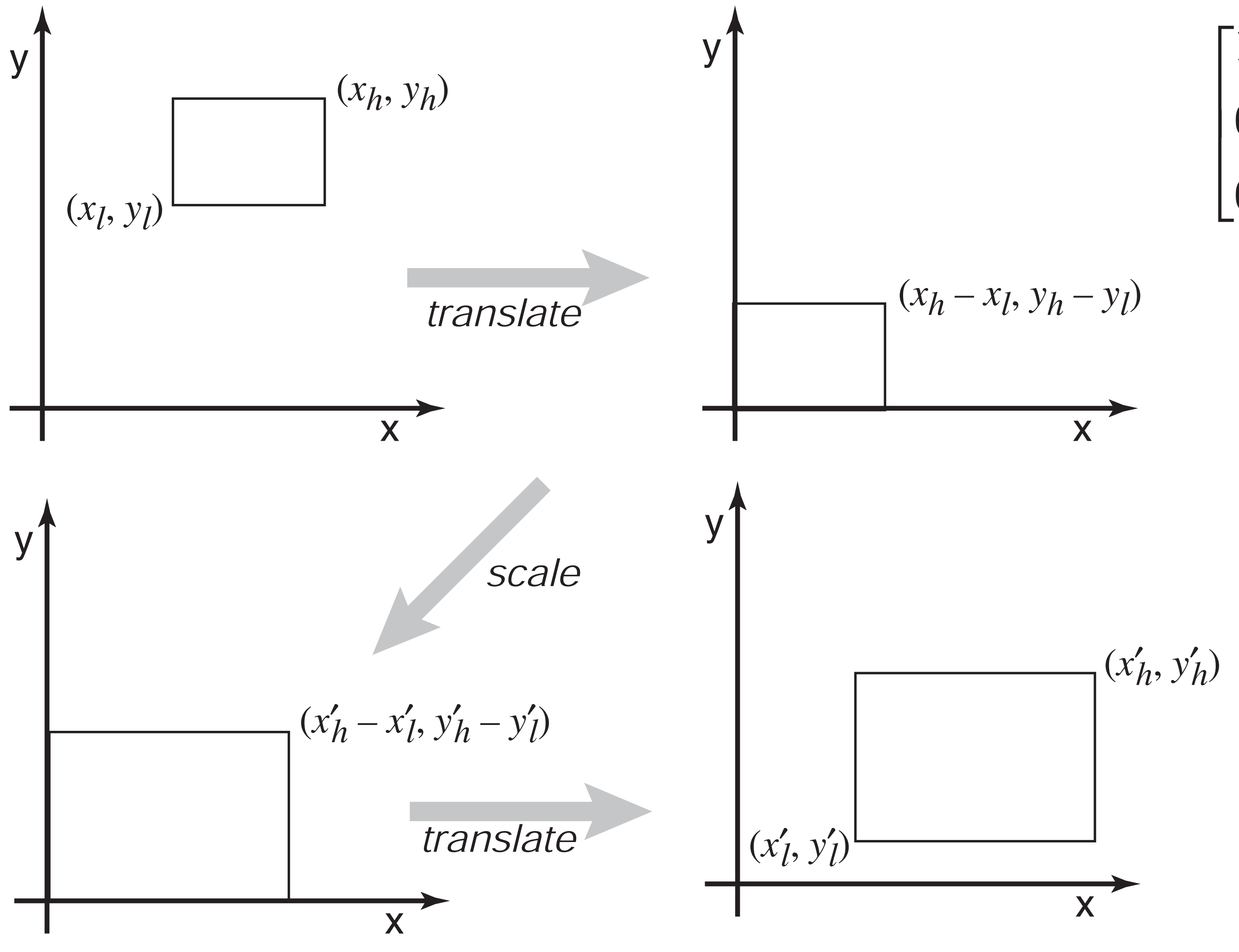
Viewing a cube of size 2

- **To draw in image, need coordinates in pixel units, though**
- **Exactly the opposite of mapping (*i*,*j*) to (*u*,*v*) in ray generation**
  - ...and exactly the same as a texture lookup



Windowing transforms

- **This transformation is worth generalizing: take one axis-aligned rectangle or box to another**
  - a useful, if mundane, piece of a transformation chain

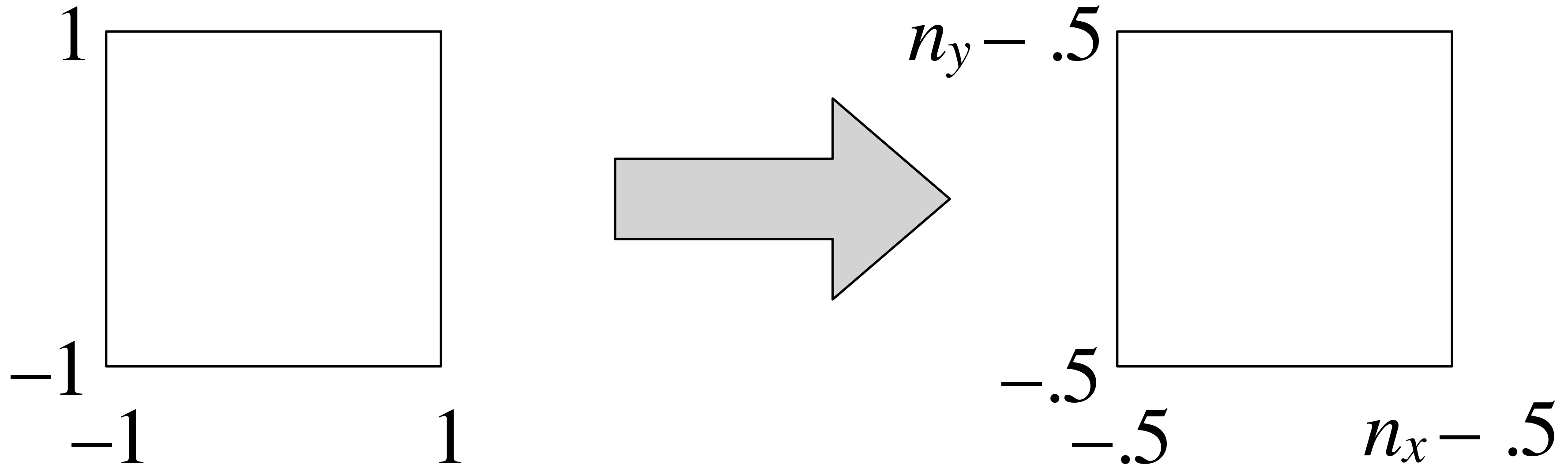


$$\begin{bmatrix} 1 & 0 & x'_l \\ 0 & 1 & y'_l \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{x'_h - x'_l}{x_h - x_l} & 0 & 0 \\ 0 & \frac{y'_h - y'_l}{y_h - y_l} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_l \\ 0 & 1 & -y_l \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \frac{x'_h - x'_l}{x_h - x_l} & 0 & \frac{x'_l x_h - x'_h x_l}{x_h - x_l} \\ 0 & \frac{y'_h - y'_l}{y_h - y_l} & \frac{y'_l y_h - y'_h y_l}{y_h - y_l} \\ 0 & 0 & 1 \end{bmatrix}$$

[Shirley3e f. 6-16; eq. 6-6]

Viewport transformation



$$\begin{bmatrix} x_{\text{screen}} \\ y_{\text{screen}} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{n_x}{2} & 0 & \frac{n_x - 1}{2} \\ 0 & \frac{n_y}{2} & \frac{n_y - 1}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\text{canonical}} \\ y_{\text{canonical}} \\ 1 \end{bmatrix}$$

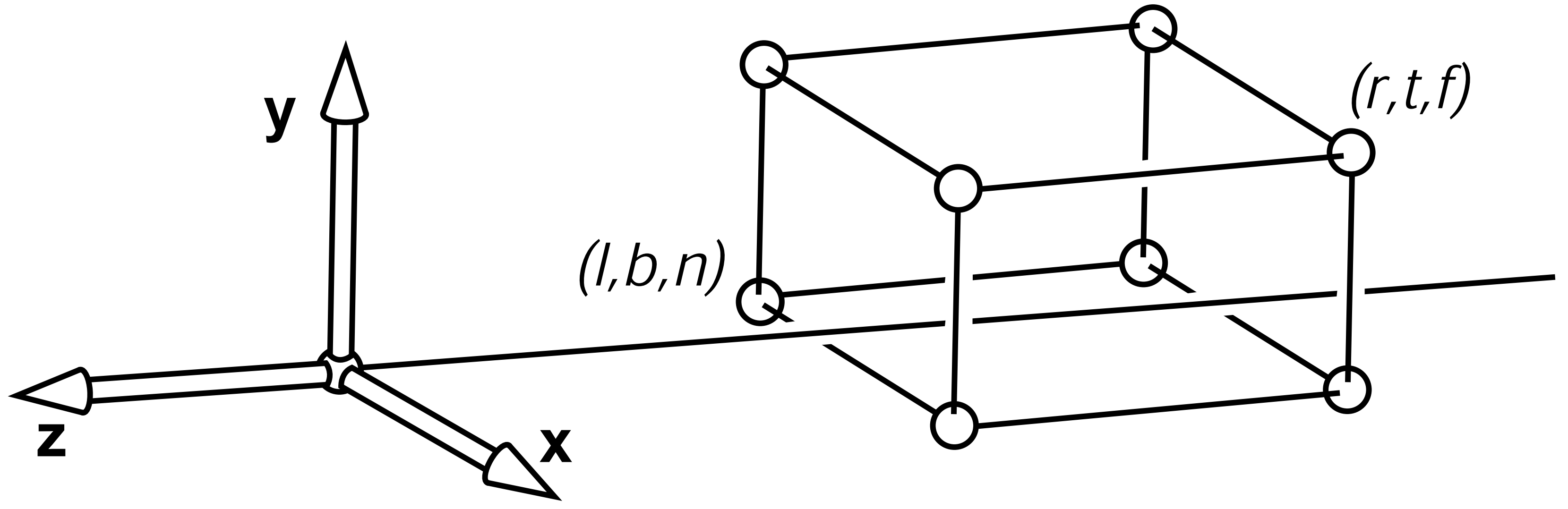
Viewport transformation

- **In 3D, carry along *z* for the ride**
  - one extra row and column

$$\mathbf{M}_{\mathrm{vp}} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x - 1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y - 1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Orthographic projection

- **First generalization: different view rectangle**
  - retain the minus-*z* view direction



  - specify view by left, right, top, bottom
  - also near, far

Orthographic projection

- **We can implement this by mapping the view volume to the canonical view volume.**
- **This is just a 3D windowing transformation!**

$$\begin{bmatrix} \frac{x'_h - x'_l}{x_h - x_l} & 0 & 0 & \frac{x'_l x_h - x'_h x_l}{x_h - x_l} \\ 0 & \frac{y'_h - y'_l}{y_h - y_l} & 0 & \frac{y'_l y_h - y'_h y_l}{y_h - y_l} \\ 0 & 0 & \frac{z'_h - z'_l}{z_h - z_l} & \frac{z'_l z_h - z'_h z_l}{z_h - z_l} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{M}_{\text{orth}} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

caution: differences from traditional OpenGL standard! Here, *n* and *f* are negative; near is +1 in the canonical view volume; and both eye space and clip space have right handed coordinates.

Locating the camera

- **In constructing viewing rays we used the equation**

$$\mathbf{o} = \mathbf{e}$$

$$\mathbf{d} = -d\mathbf{w} + u\mathbf{u} + v\mathbf{v}$$

  – this can be seen as transforming the ray (**0**, (*u*, *v*, –*d*)) by the linear transformation:

$$F_c = \begin{bmatrix} | & | & | & | \\ \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{e} \\ | & | & | & | \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{o} = F_c \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad \mathbf{d} = F_c \begin{bmatrix} u \\ v \\ -d \\ 0 \end{bmatrix}$$

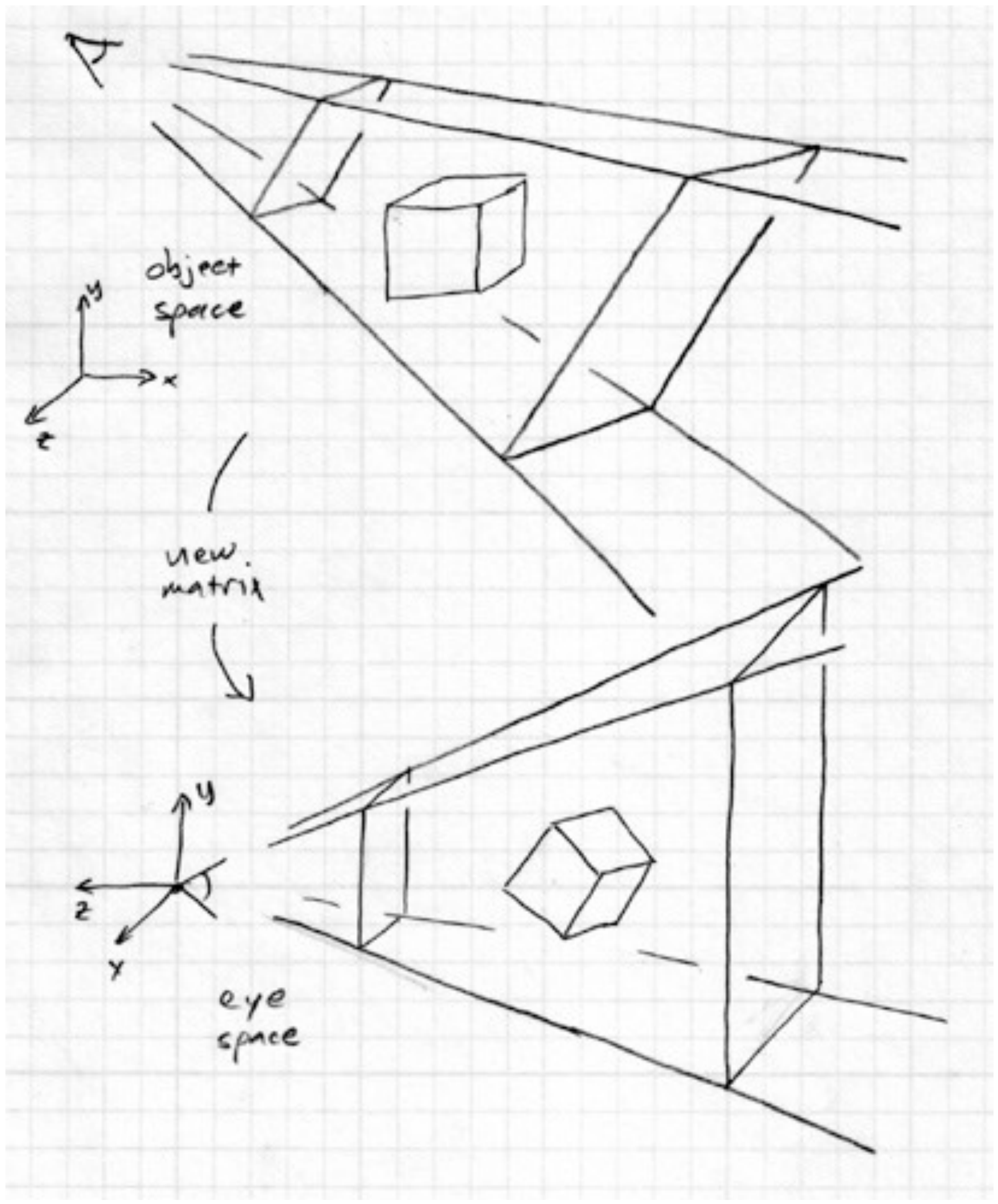
  – in this interpretation, we first constructed the ray in eye space, then transformed it to world space.

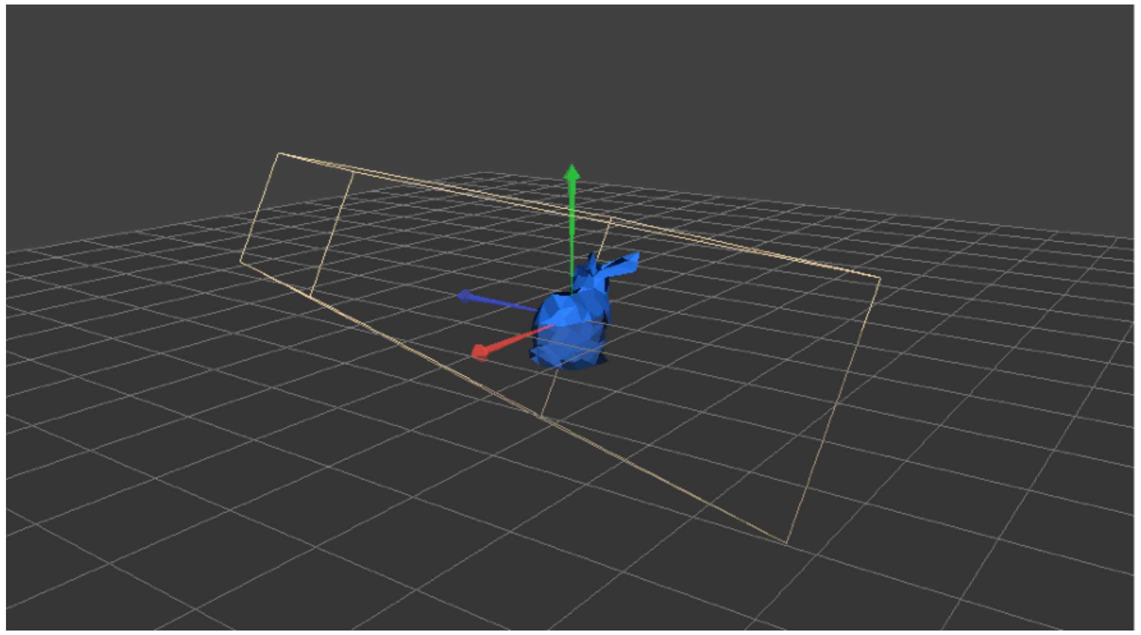
Camera and modeling matrices

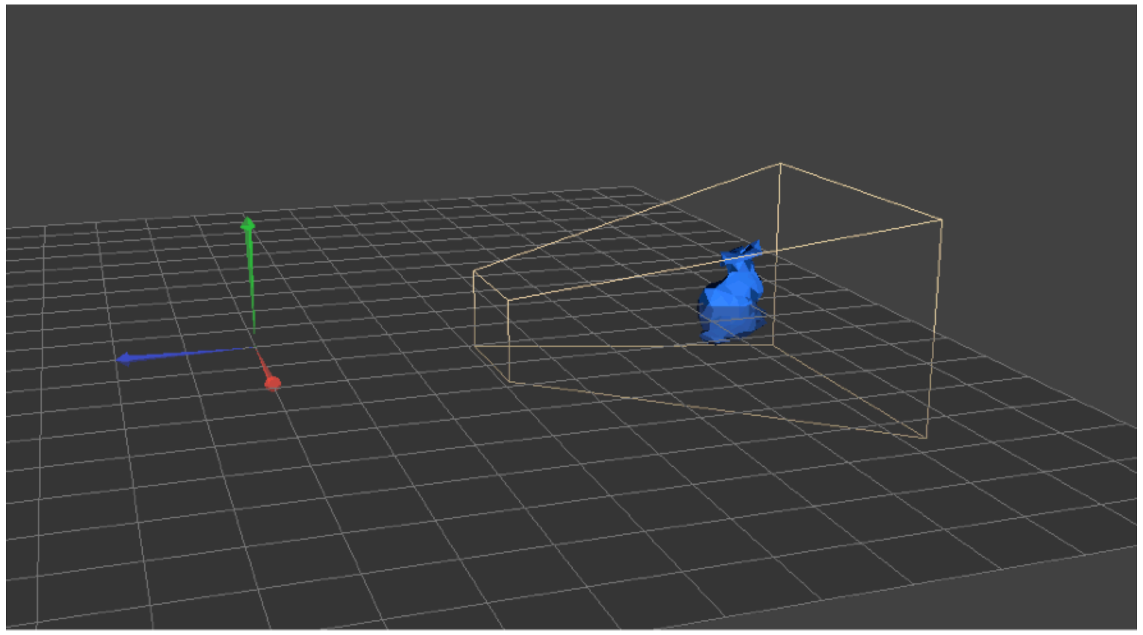
- **The preceding transforms start from eye coordinates**
  – before we apply those we need to transform into that space
- **Transform from world (canonical) to eye space is traditionally called the** *viewing matrix*
  – it is the canonical-to-frame matrix for the camera frame
  – that is, F_c^{-1}
- **Remember that geometry would originally have been in the object's local coordinates; transform into world coordinates is called the** *modeling matrix*, M_m
- **Note many programs combine the two into a** *modelview* **matrix and just skip world coordinates**

Viewing transformation

Demo







the camera matrix rewrites all coordinates in eye space

Orthographic transformation chain

- **Start with coordinates in object's local coordinates**
- **Transform into world coords (modeling transform, M_m)**
- **Transform into eye coords (camera xf., $M_{\text{cam}} = F_c^{-1}$)**
- **Orthographic projection, M_{orth}**
- **Viewport transform, M_{vp}**

$$\mathbf{p}_s = \mathbf{M}_{\text{vp}}\mathbf{M}_{\text{orth}}\mathbf{M}_{\text{cam}}\mathbf{M}_{\text{m}}\mathbf{p}_o$$

$$\begin{bmatrix} x_s \\ y_s \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y-1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \mathbf{M}_{\text{m}} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$

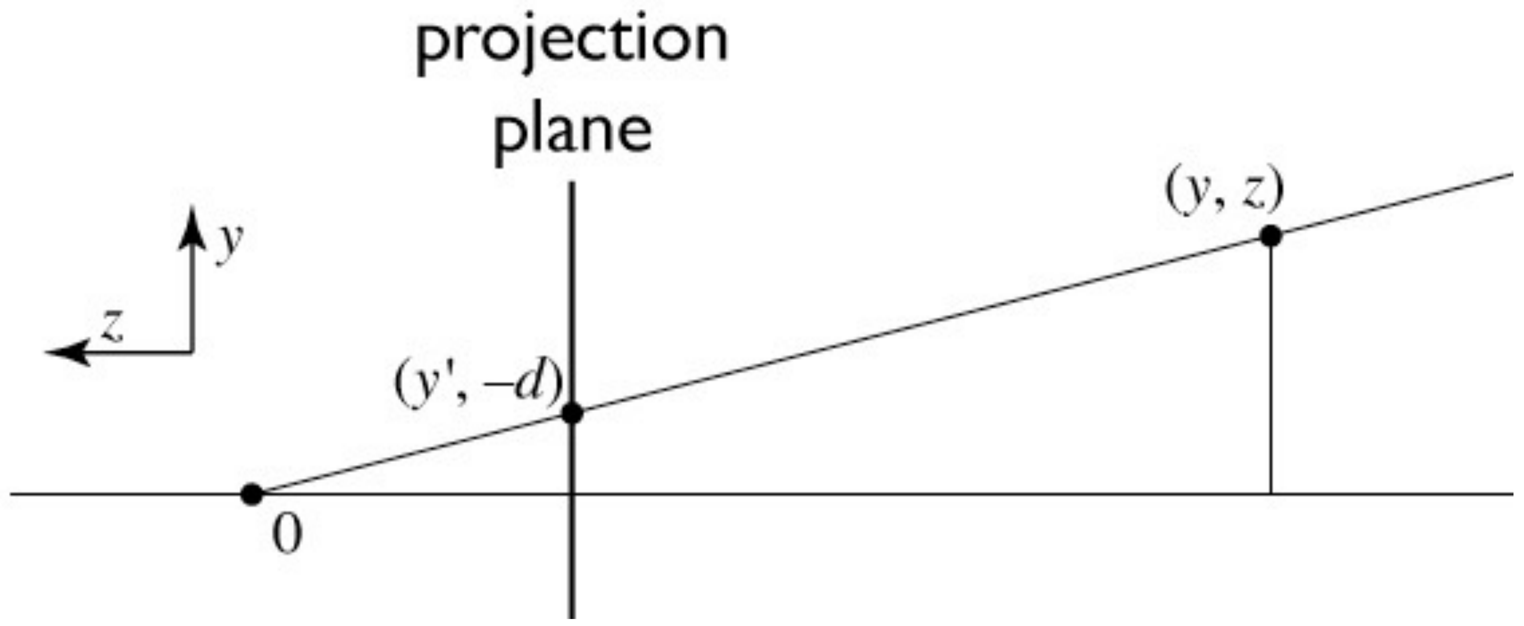
Clipping planes

- **In object-order systems we always use at least two *clipping planes* that further constrain the view volume**
  - near plane: parallel to view plane; things between it and the viewpoint will not be rendered
  - far plane: also parallel; things behind it will not be rendered
- **These planes are:**
  - partly to remove unnecessary stuff (e.g. behind the camera)
  - but really to constrain the range of depths

    (we'll see why later)

Ray Verrier

Perspective projection



similar triangles:

$$\frac{y'}{d} = \frac{y}{-z}$$

$$y' = -dy/z$$

Homogeneous coordinates revisited

- **Perspective requires division**
  - that is not part of affine transformations
  - in affine, parallel lines stay parallel
    - therefore not vanishing point
    - therefore no rays converging on viewpoint
- **“True” purpose of homogeneous coords: projection**

Homogeneous coordinates revisited

- **Introduced *w* = 1 coordinate as a placeholder**

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

  - used as a convenience for unifying translation with linear

- **Can also allow arbitrary *w***

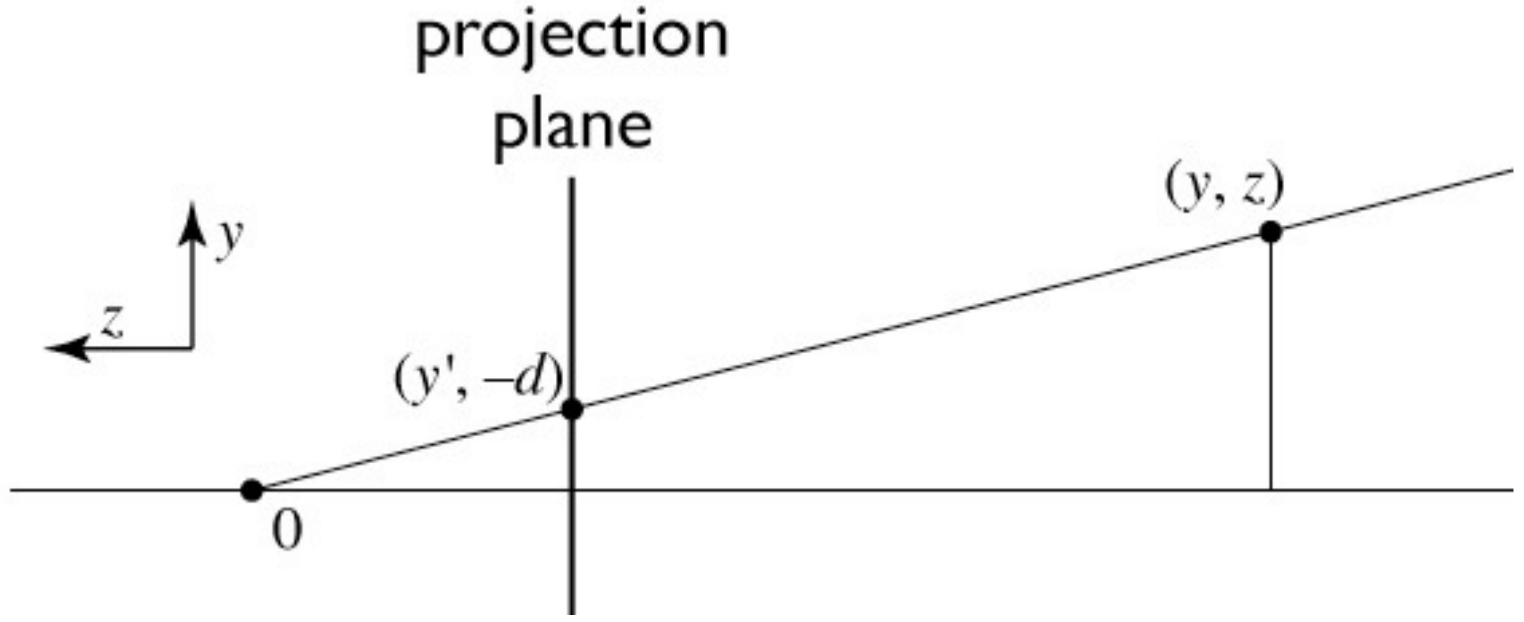
$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \sim \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix}$$

Implications of *w*

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \sim \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix}$$

- **All scalar multiples of a 4-vector are equivalent**
- **When *w* is not zero, can divide by *w***
  - therefore these points represent "normal" affine points
- **When *w* is zero, it's a point at infinity, a.k.a. a direction**
  - this is the point where parallel lines intersect
  - can also think of it as the vanishing point
- **Digression on projective space**

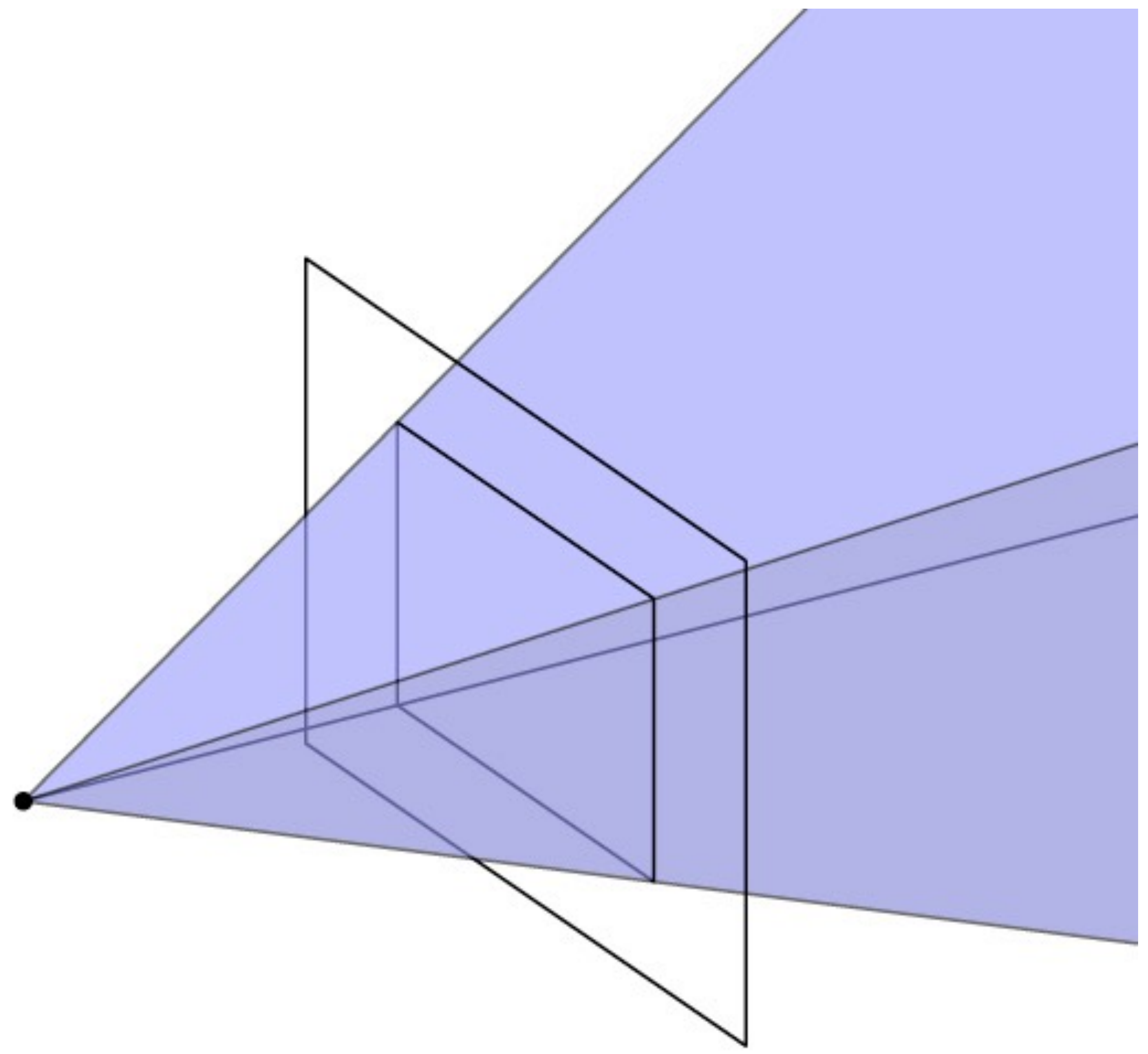
Perspective projection



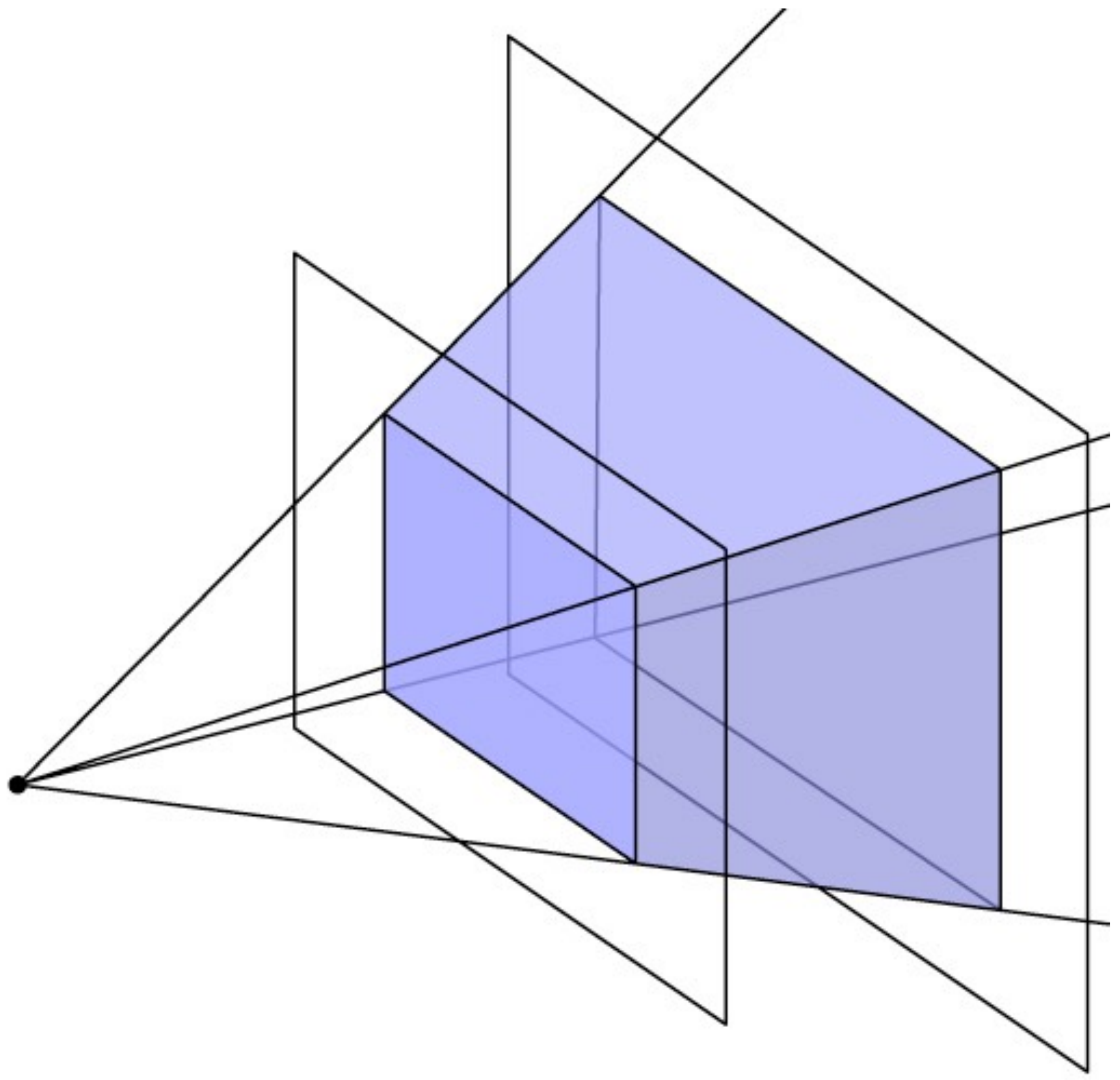
to implement perspective, just move z to w:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -dx/z \\ -dy/z \\ 1 \end{bmatrix} \sim \begin{bmatrix} dx \\ dy \\ -z \end{bmatrix} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

View volume: perspective



View volume: perspective (clipped)



Carrying depth through perspective

- **Perspective has a varying denominator—can't preserve depth!**
- **Compromise: preserve depth on near and far planes**

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} \sim \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ -z \end{bmatrix} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

– that is, choose a and b so that *z*'(*n*) = *n* and *z*'(*f*) = *f*.

$$\tilde{z}(z) = az + b$$

$$z'(z) = \frac{\tilde{z}}{-z} = \frac{az+b}{-z}$$

$$\text{want } z'(n) = n \text{ and } z'(f) = f$$

$$\text{result: } a = -(n+f) \text{ and } b = nf \text{ (try it)}$$

Official perspective matrix

- **Use near plane distance as the projection distance**
  - i.e., $d = -n$
- **Scale by –1 to have fewer minus signs**
  - scaling the matrix does not change the projective transformation

$$\mathbf{P} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Perspective projection matrix

- **Product of perspective matrix with orth. projection matrix**

$$\mathbf{M}_{\text{per}} = \mathbf{M}_{\text{orth}}\mathbf{P}$$

$$= \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{l+r}{l-r} & 0 \\ 0 & \frac{2n}{t-b} & \frac{b+t}{b-t} & 0 \\ 0 & 0 & \frac{f+n}{n-f} & \frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

caution: differences from traditional OpenGL standard! Here, *n* and *f* are negative; near is +1 in the canonical view volume; and both eye space and clip space have right handed coordinates.

Perspective transformation chain

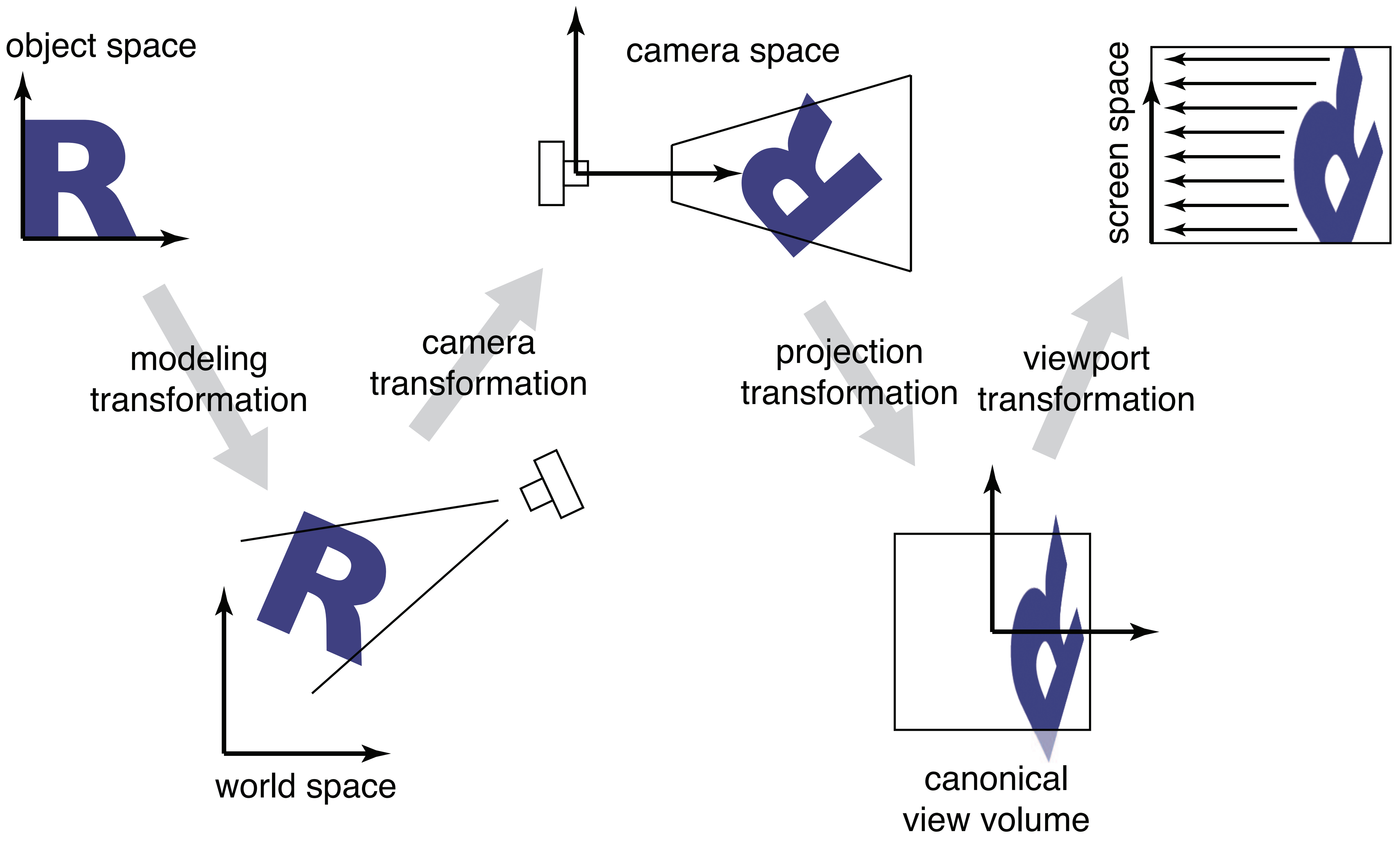
- **Transform into world coords (modeling transform, *M_m*)**
- **Transform into eye coords (camera xf., *$M_{\text{cam}} = F_c^{-1}$*)**
- **Perspective matrix, *P***
- **Orthographic projection, *M_{orth}***
- **Viewport transform, *M_{vp}***

$$\mathbf{p}_s = \mathbf{M}_{\text{vp}}\mathbf{M}_{\text{orth}}\mathbf{P}\mathbf{M}_{\text{cam}}\mathbf{M}_{\text{m}}\mathbf{p}_o$$

$$\begin{bmatrix} x_s \\ y_s \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y-1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix} \mathbf{M}_{\text{cam}}\mathbf{M}_{\text{m}} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$

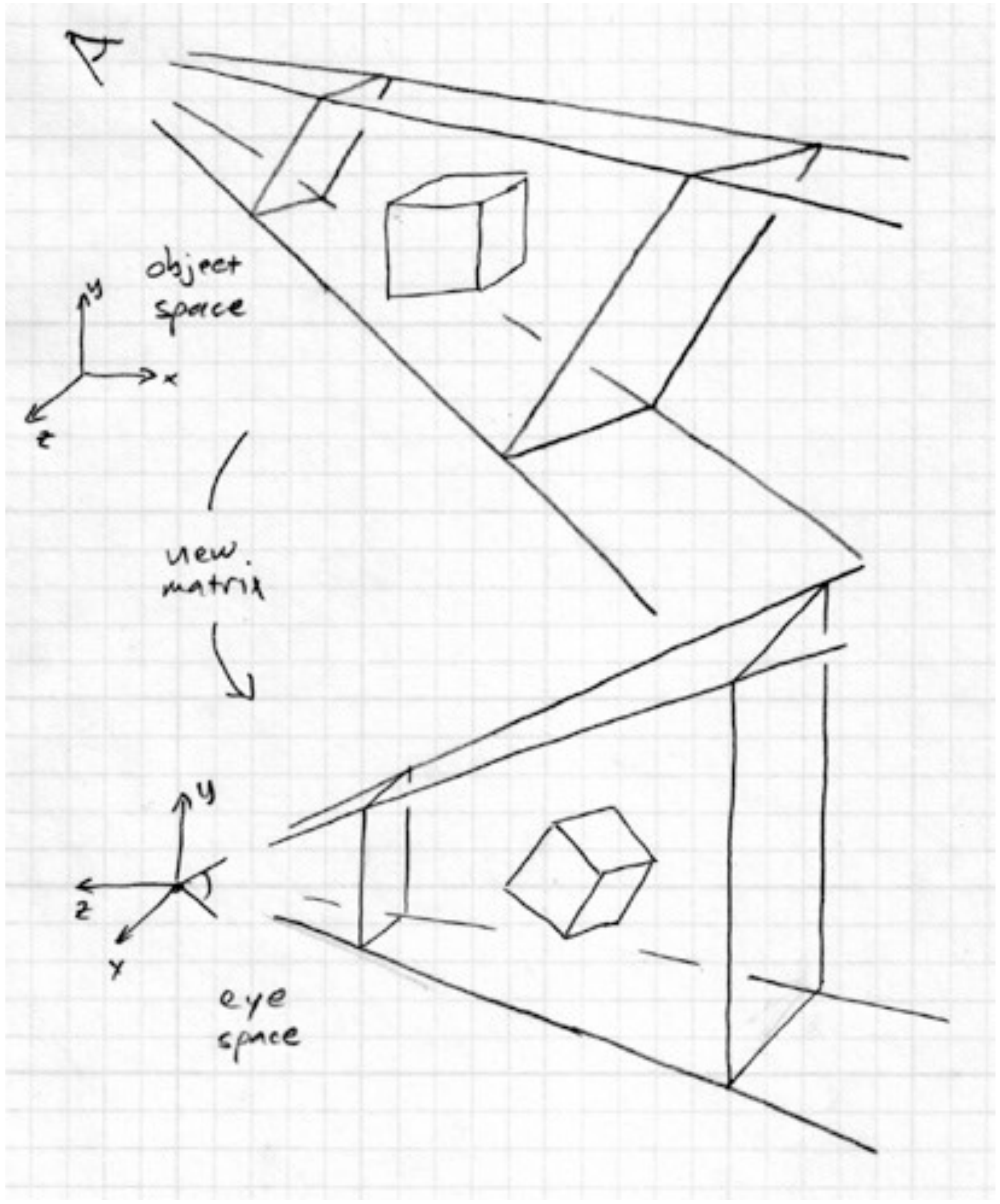
Pipeline of transformations

- **Standard sequence of transforms**



Transformations for perspective

Demo



object space
y
x
z
view matrix
y
z
x
eye space