Computer Graphics

- 3D Viewing

Junjie Cao @ DLUT
Spring 2019
http://jjcao.github.io/ComputerGraphics/

Pleasure may come from illusion, but happiness can come only of reality.

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
- Inverting the ray tracing process requires division for the perspective case

Viewing transformations

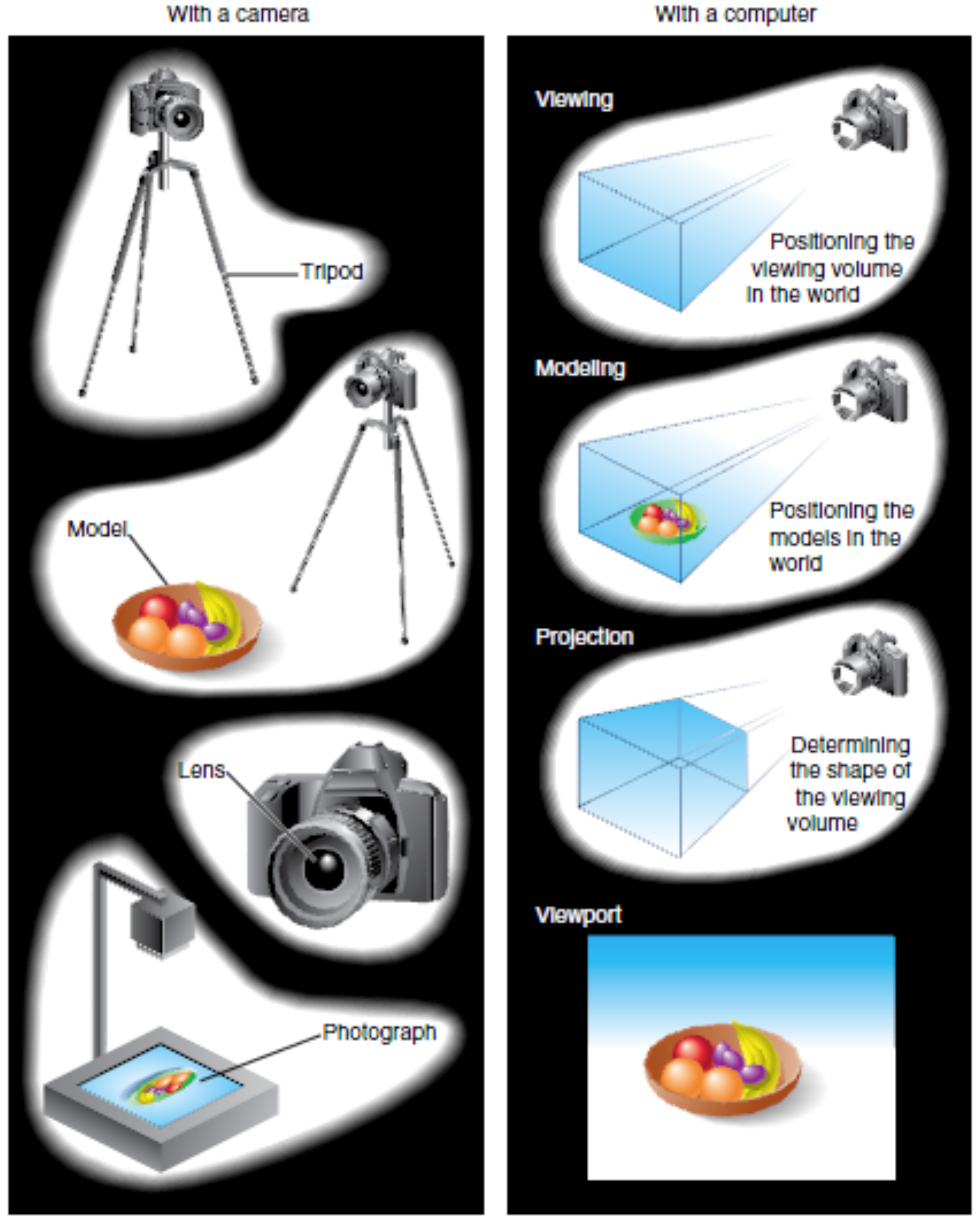
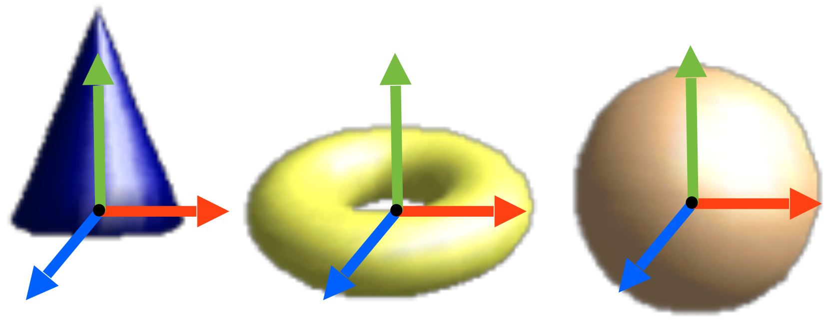


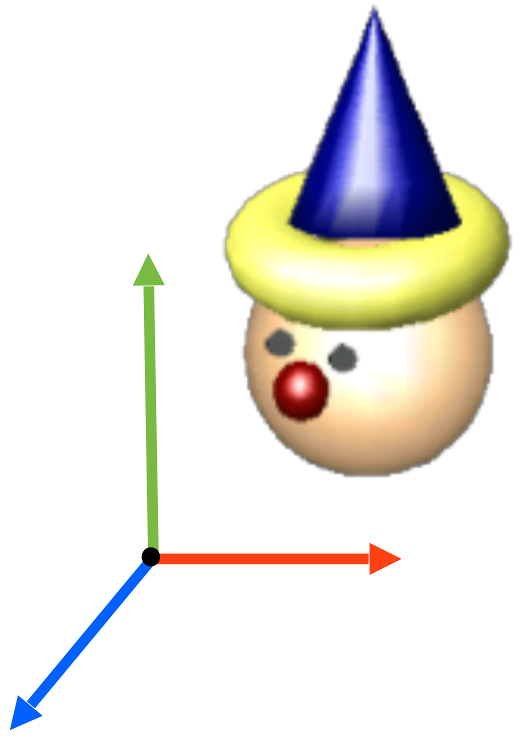
Figure 3-1 The Camera Analogy

World Coordinates → Modeling | Camera | Projection | Viewport → Screen Space (pixels)

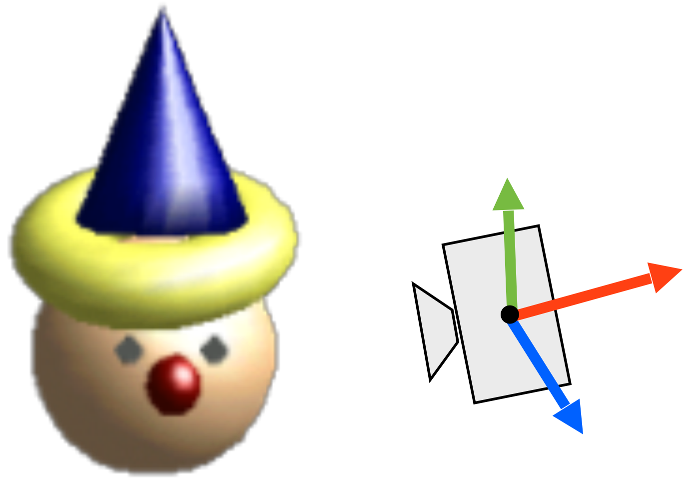
Coordinate Systems



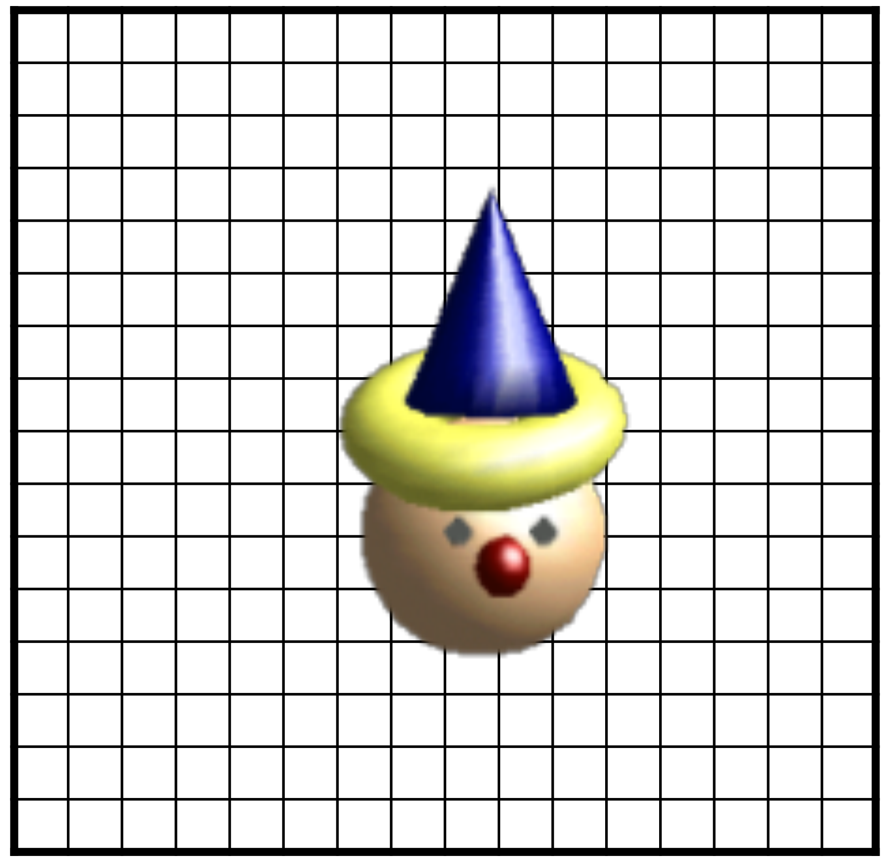
object coordinates



world coordinates



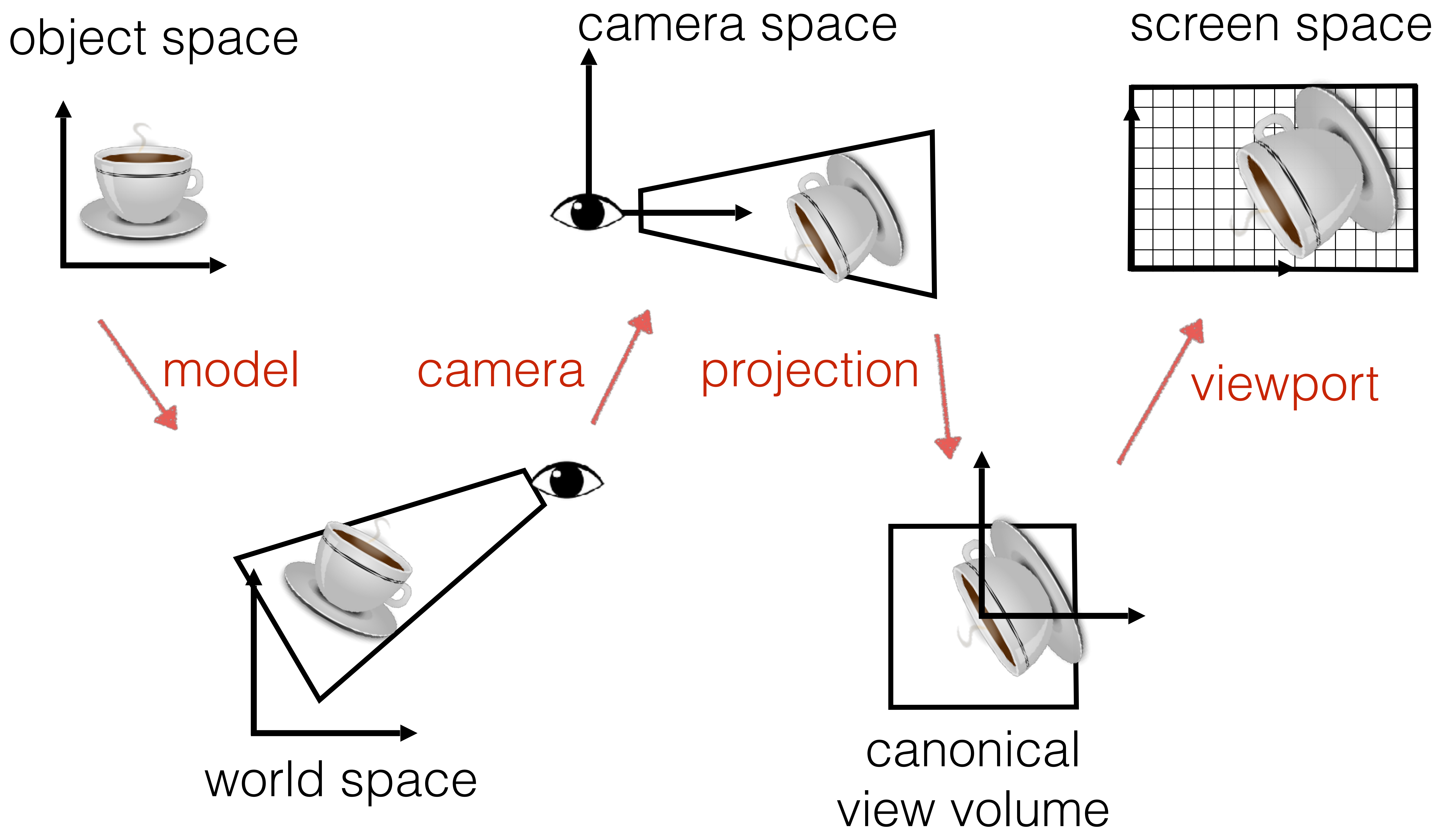
camera coordinates



screen coordinates

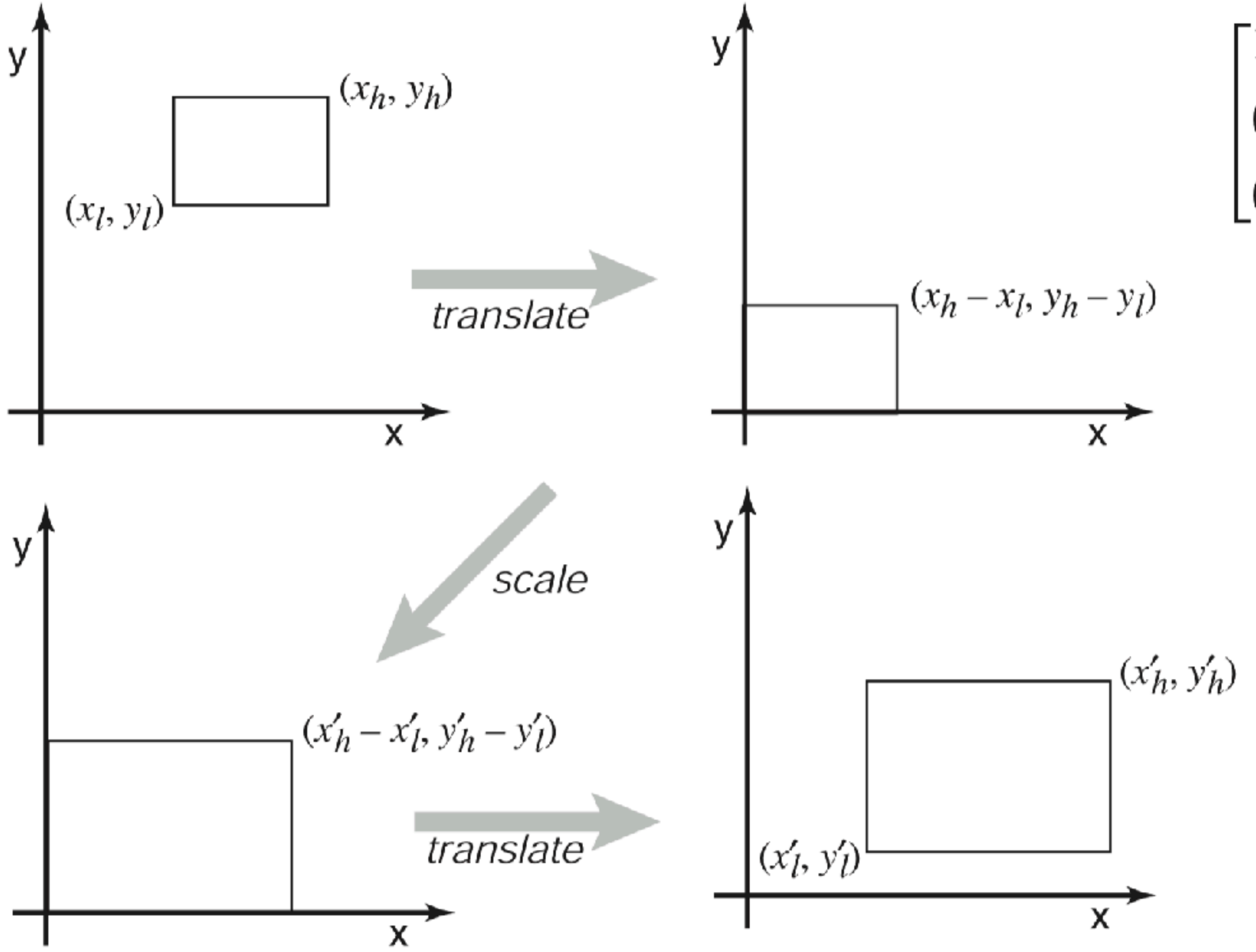
Image Copyright: Mark Pauly

Viewing Transformation



Windowing transforms

- **take one axis-aligned rectangle or box to another**

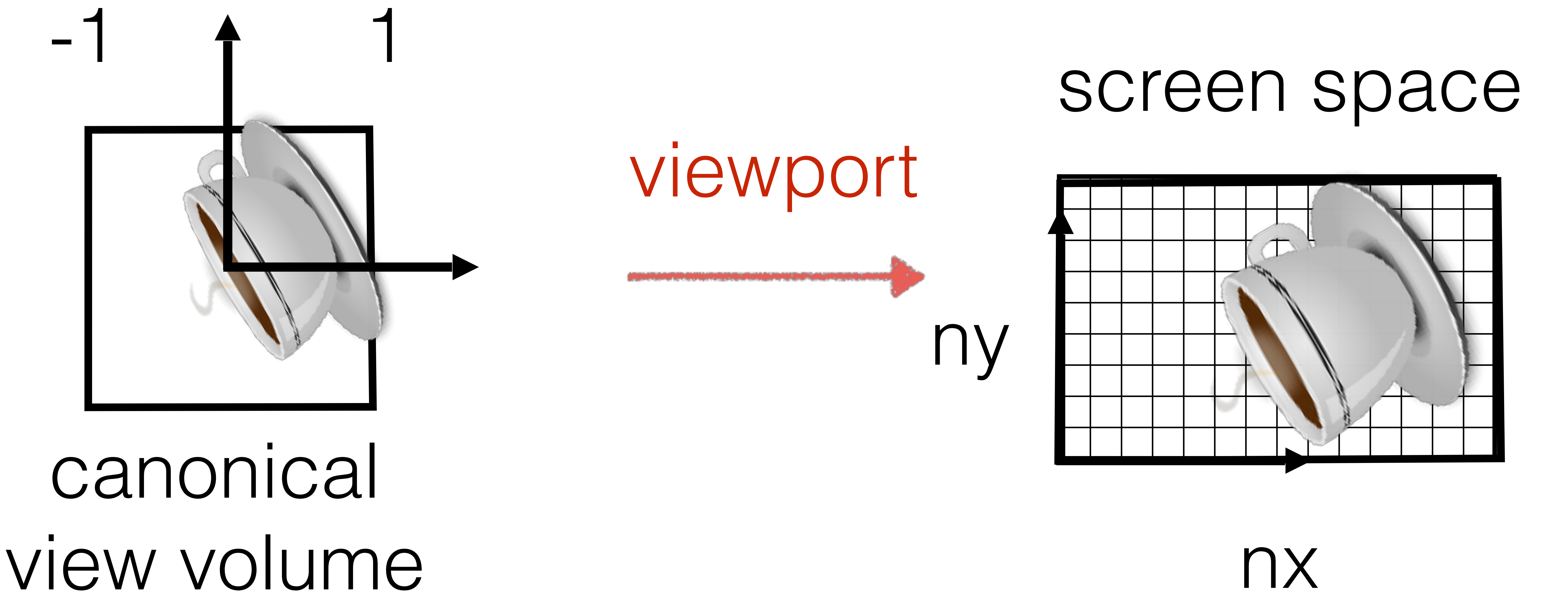


$$\begin{bmatrix}1&0&x'_l\\0&1&y'_l\\0&0&1\end{bmatrix}\begin{bmatrix}\frac{x'_h-x'_l}{x_h-x_l}&0&0\\0&\frac{y'_h-y'_l}{y_h-y_l}&0\\0&0&1\end{bmatrix}\begin{bmatrix}1&0&-x_l\\0&1&-y_l\\0&0&1\end{bmatrix}$$

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

Viewport transformation

It is a simple windowing transform

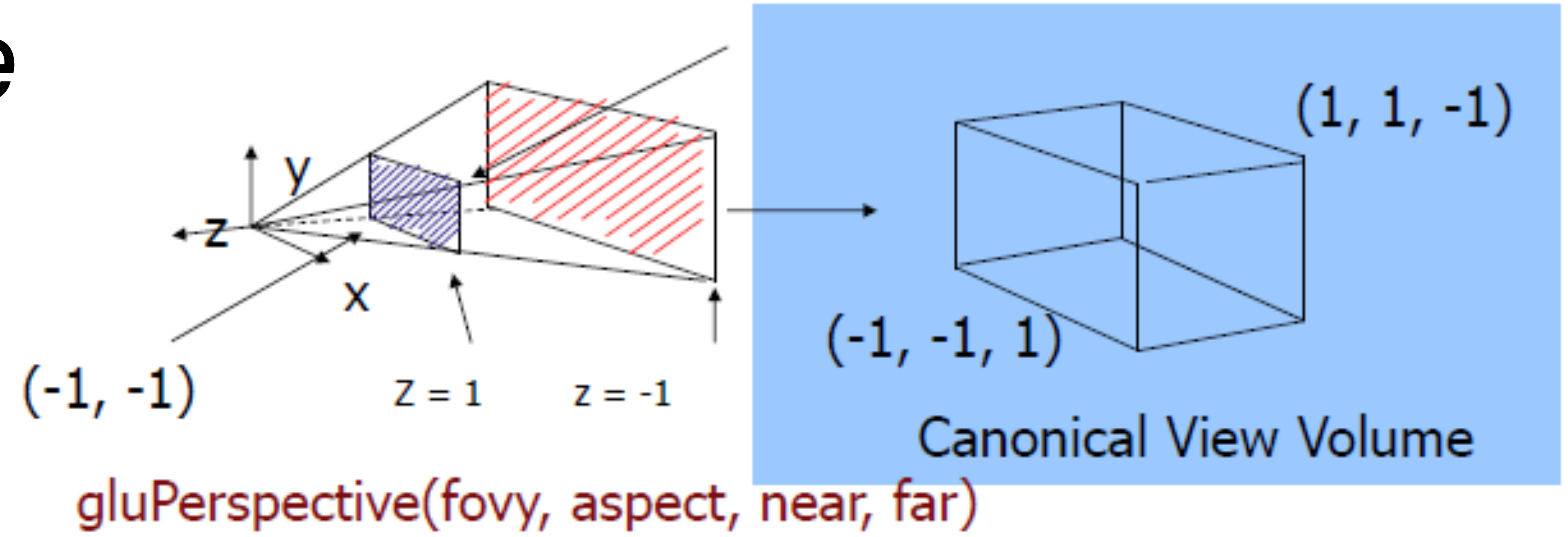


$$\begin{bmatrix} x_{screen} \\ y_{screen} \\ 1 \end{bmatrix} = \begin{bmatrix} nx/2 & 0 & \frac{n_x-1}{2} \\ 0 & ny/2 & \frac{n_y-1}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{canonical} \\ y_{canonical} \\ 1 \end{bmatrix}$$

How does it look in 3D?

Canonical view volume to screen space

- a restricted case: the *canonical view volume*



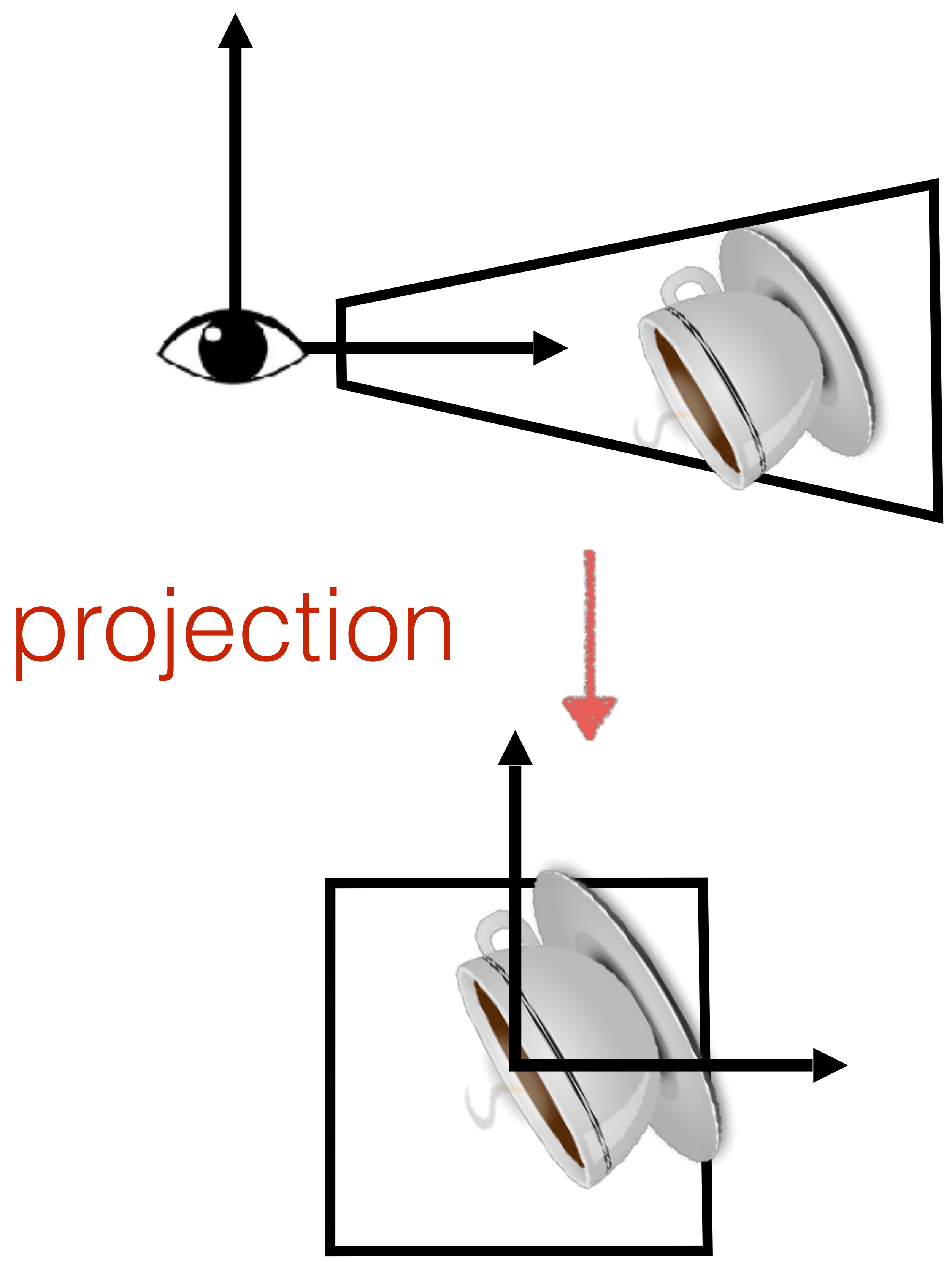
- ***coordinates in it are called "normalized device coordinates" (NDC)***

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

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

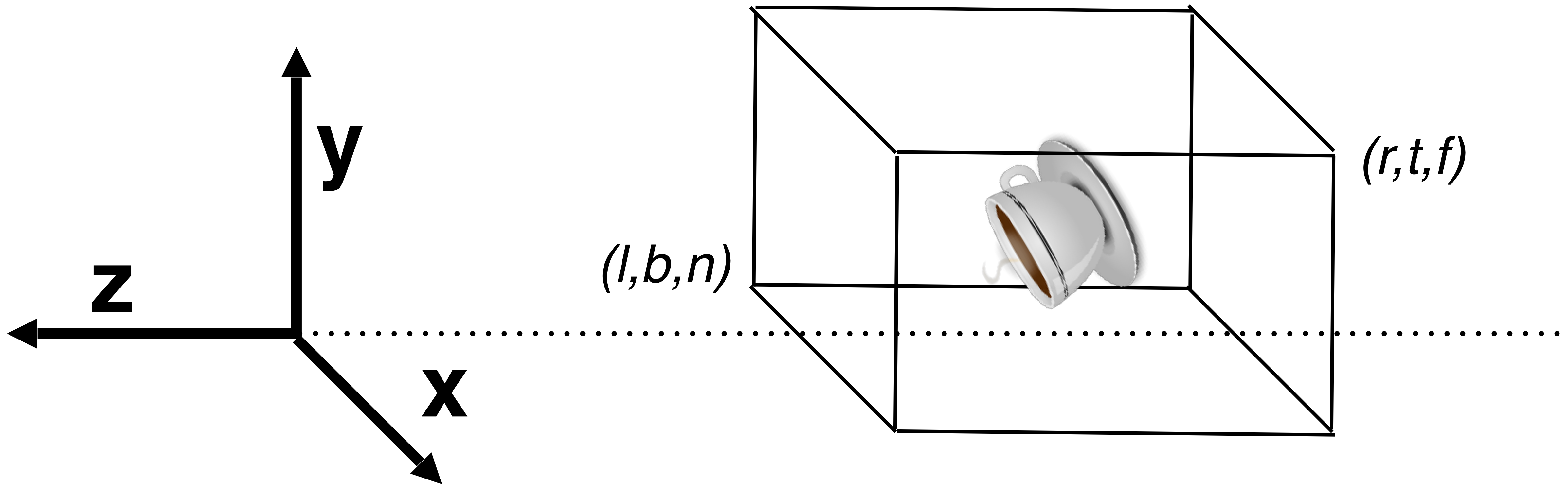
Orthographic Projection

camera space



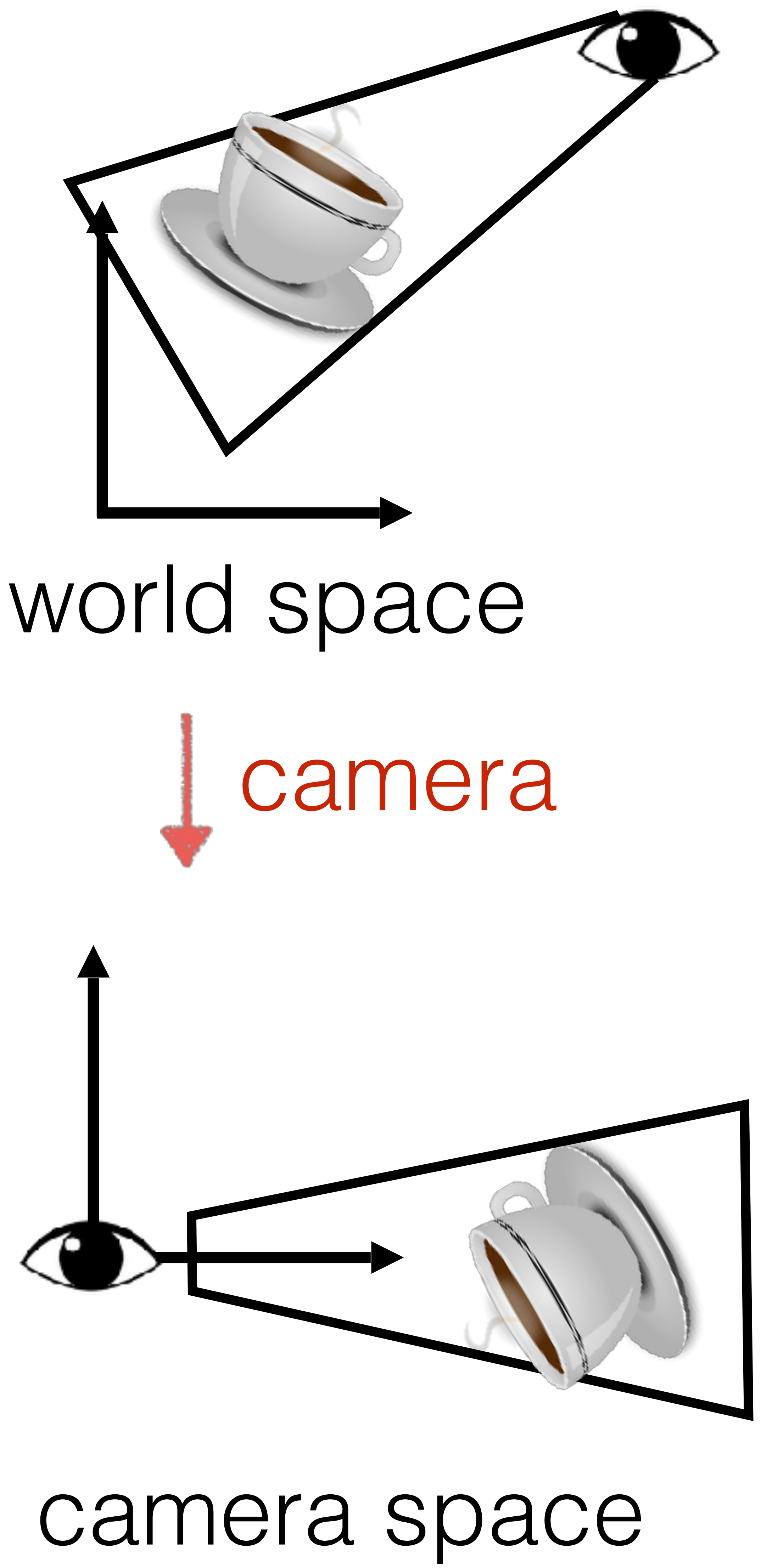
canonical
view volume

It is also a windowing transform

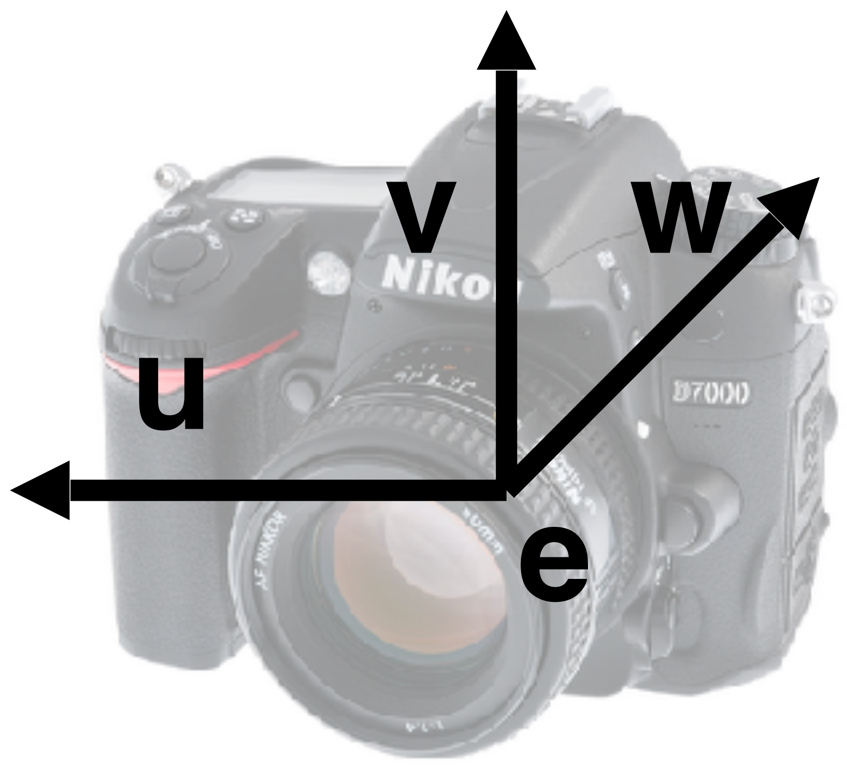


$$\mathbf{M}_{orth} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Camera Transformation



1. Construct the camera reference system given:
   1. The eye position **e**
   2. The gaze direction **g**
   3. The view-up vector **t**

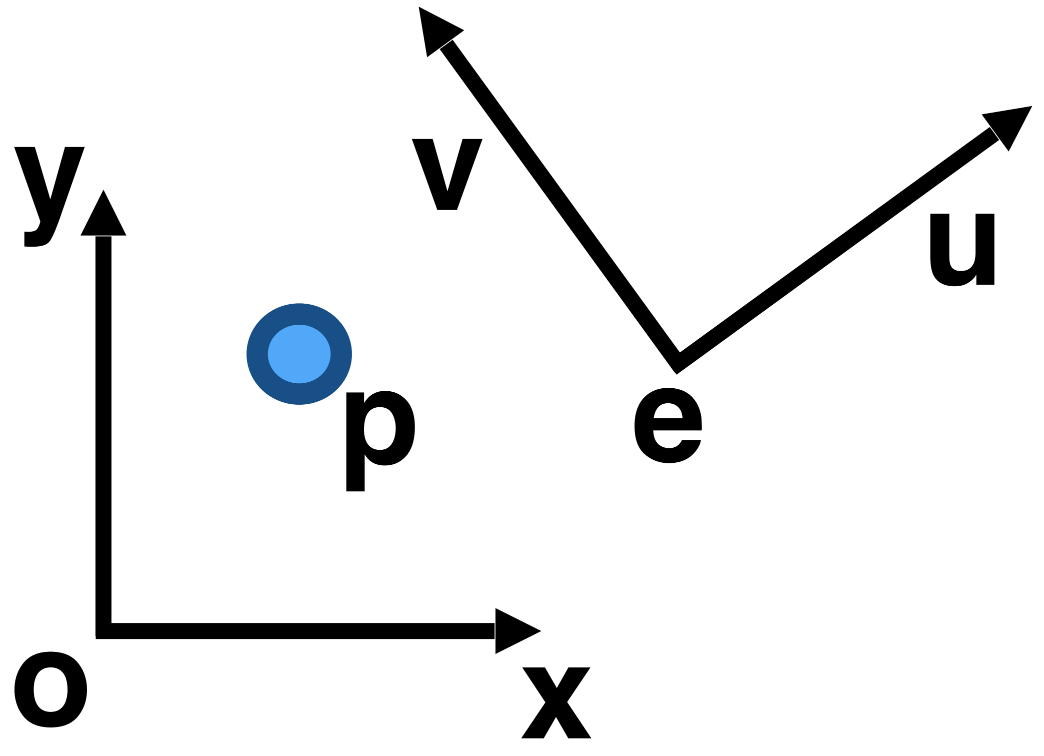


$$\mathbf{w} = -\frac{\mathbf{g}}{||\mathbf{g}||}$$

$$\mathbf{u} = \frac{\mathbf{t} \times \mathbf{w}}{||\mathbf{t} \times \mathbf{w}||}$$

$$\mathbf{v} = \mathbf{w} \times \mathbf{u}$$

Change of frame



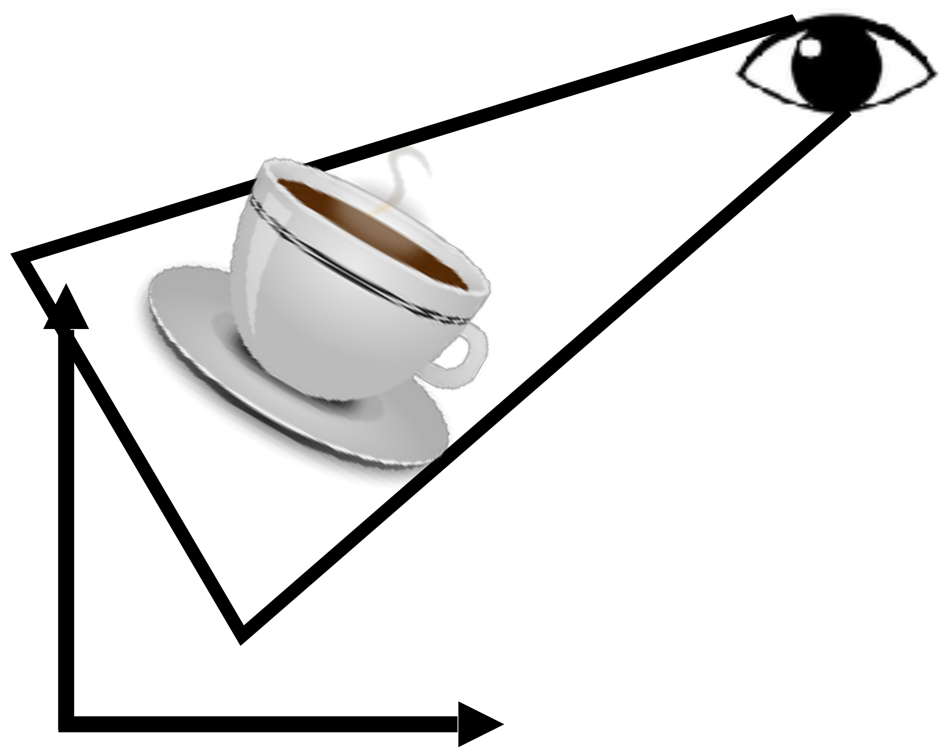
$$\mathbf{p} = (p_x, p_y) = \mathbf{o} + p_x\mathbf{x} + p_y\mathbf{y}$$
$$\mathbf{p} = (p_u, p_v) = \mathbf{e} + p_u\mathbf{u} + p_v\mathbf{v}$$

$$\begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & e_x \\ 0 & 1 & e_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_x & v_x & 0 \\ u_y & v_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_u \\ p_v \\ 1 \end{bmatrix} = \begin{bmatrix} u_x & v_x & e_x \\ u_y & v_y & e_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_u \\ p_v \\ 1 \end{bmatrix}$$

$$\mathbf{p}_{xy} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{e} \\ 0 & 0 & 1 \end{bmatrix} \mathbf{p}_{uv} \qquad \mathbf{p}_{uv} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{e} \\ 0 & 0 & 1 \end{bmatrix}^{-1} \mathbf{p}_{xy}$$

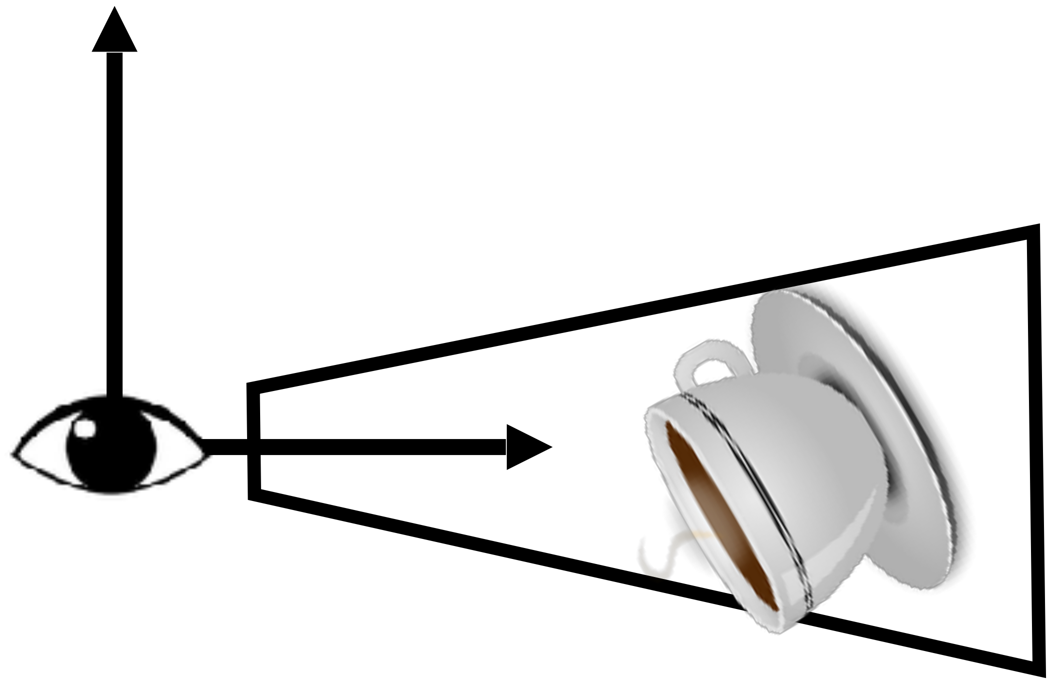
Can you write it directly without the inverse?

Camera Transformation



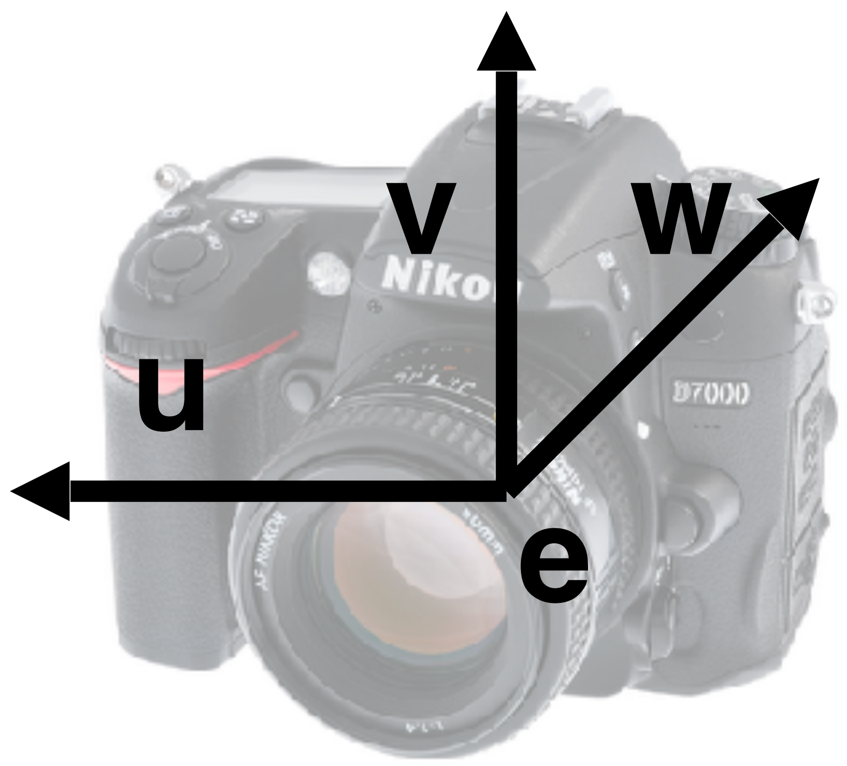
world space

camera



camera space

1. Construct the camera reference system given:
   1. The eye position **e**
   2. The gaze direction **g**
   3. The view-up vector **t**



$$\mathbf{w} = -\frac{\mathbf{g}}{||\mathbf{g}||}$$

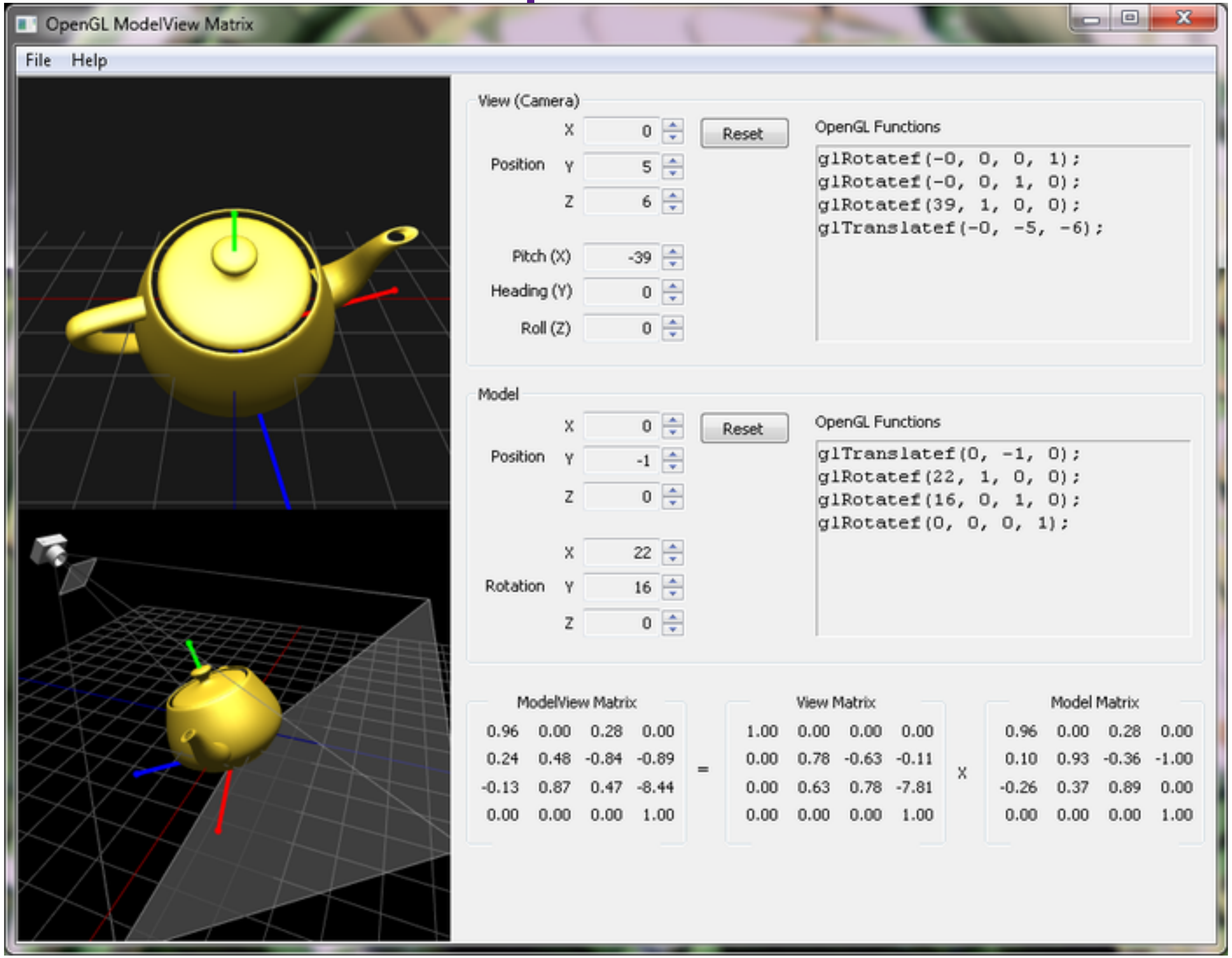
$$\mathbf{u} = \frac{\mathbf{t} \times \mathbf{w}}{||\mathbf{t} \times \mathbf{w}||}$$

$$\mathbf{v} = \mathbf{w} \times \mathbf{u}$$

2. Construct the unique transformations that converts world coordinates into camera coordinates

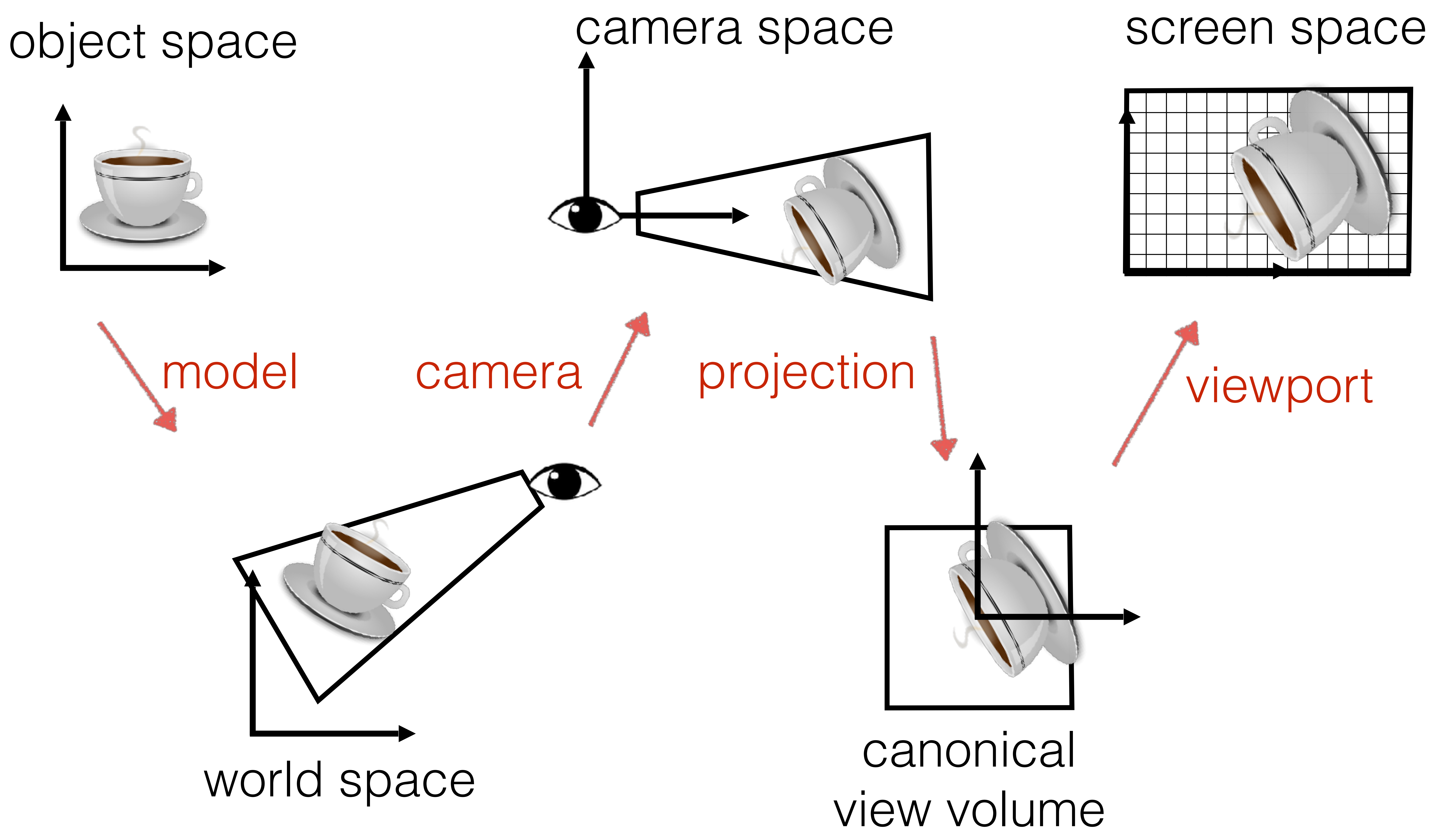
$$\mathbf{M}_{cam} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1}$$

Example: ModelView Matrix



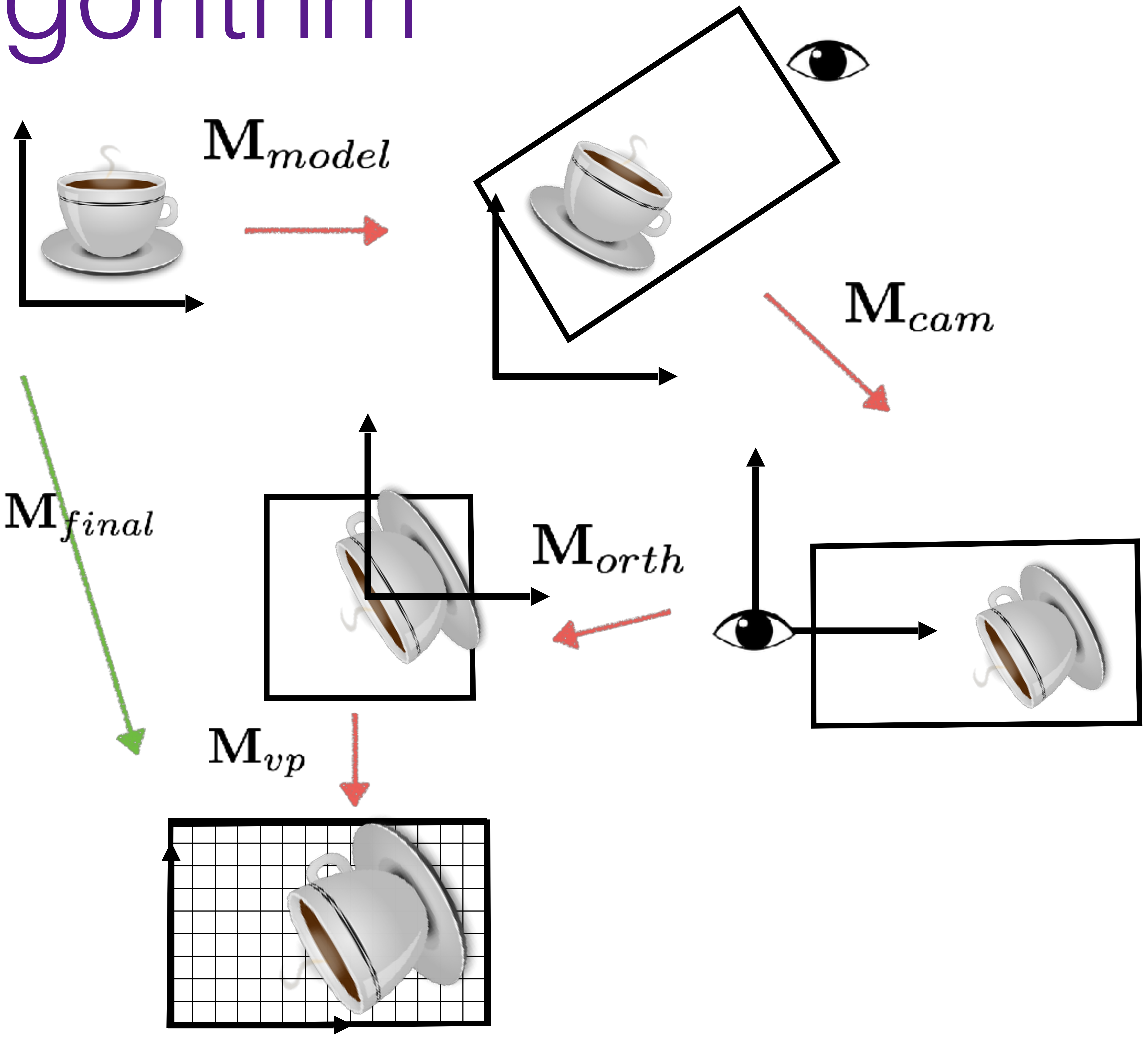
- http://www.songho.ca/opengl/gl_transform.html#example1

Viewing Transformation



Algorithm

- Construct Viewport Matrix $\mathbf{M}_{vp}$
- Construct Projection Matrix $\mathbf{M}_{orth}$
- Construct Camera Matrix $\mathbf{M}_{cam}$
- $\mathbf{M} = \mathbf{M}_{vp}\mathbf{M}_{orth}\mathbf{M}_{cam}$
- For each model
    - Construct Model Matrix $\mathbf{M}_{model}$
    - $\mathbf{M}_{final} = \mathbf{M}\mathbf{M}_{model}$
    - For every point **p** in each primitive of the model
        - $\mathbf{p}_{final} = \mathbf{M}_{final}\mathbf{p}$
    - Rasterize the model



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., $M_{cam} = F_c^{-1}$)**
- **Orthographic projection, M_{orth}**
- **Viewport transform, M_{vp}**

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

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

Rudimentary perspective in cave drawings

Lascaux, France source: Wikipedia

Painting in middle ages: incorrect perspective

- Art in the service of religion
- Perspective abandoned or forgotten

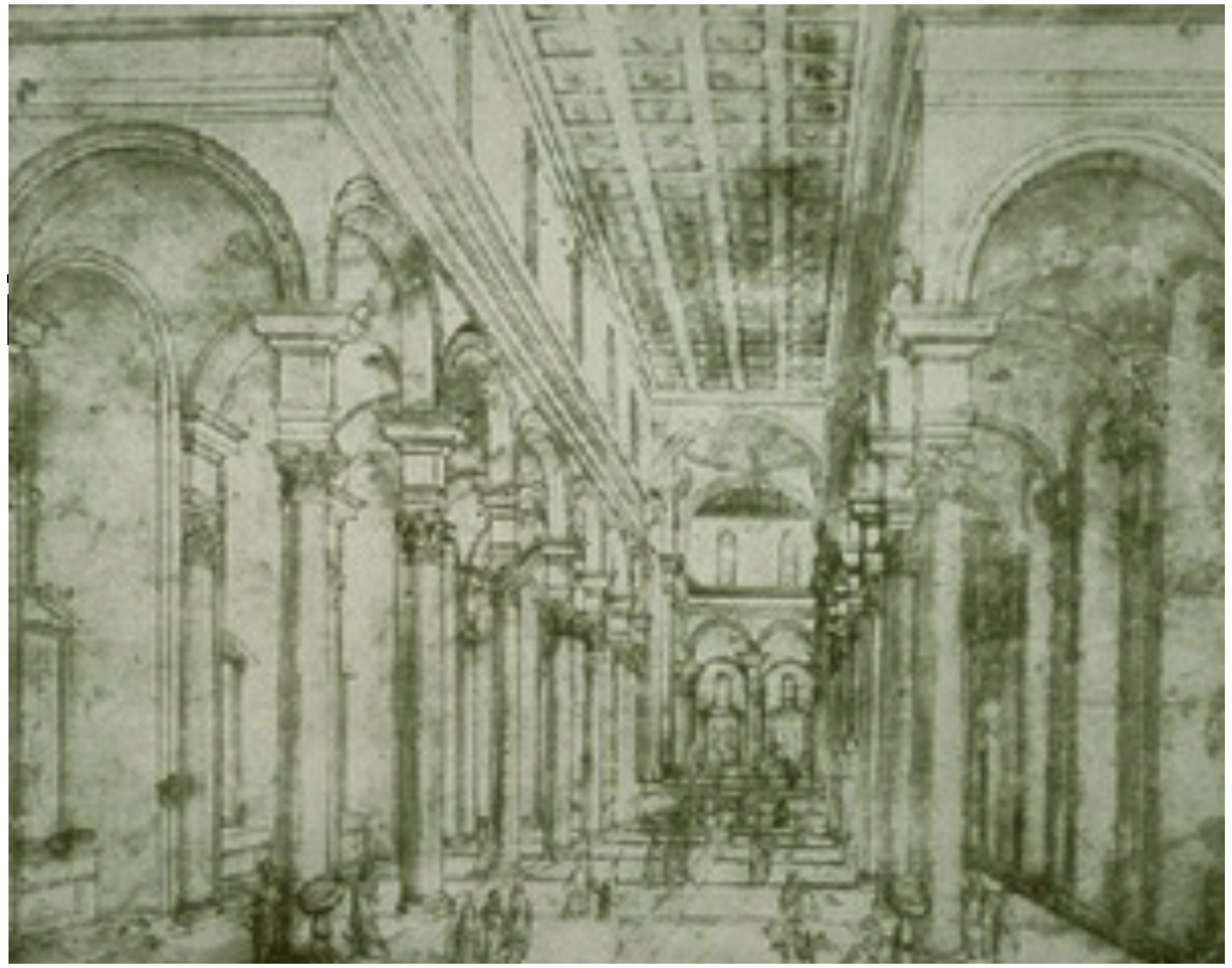
Ottonian manuscript, ca. 1000

8-9th century painting

Renaissance

• ati

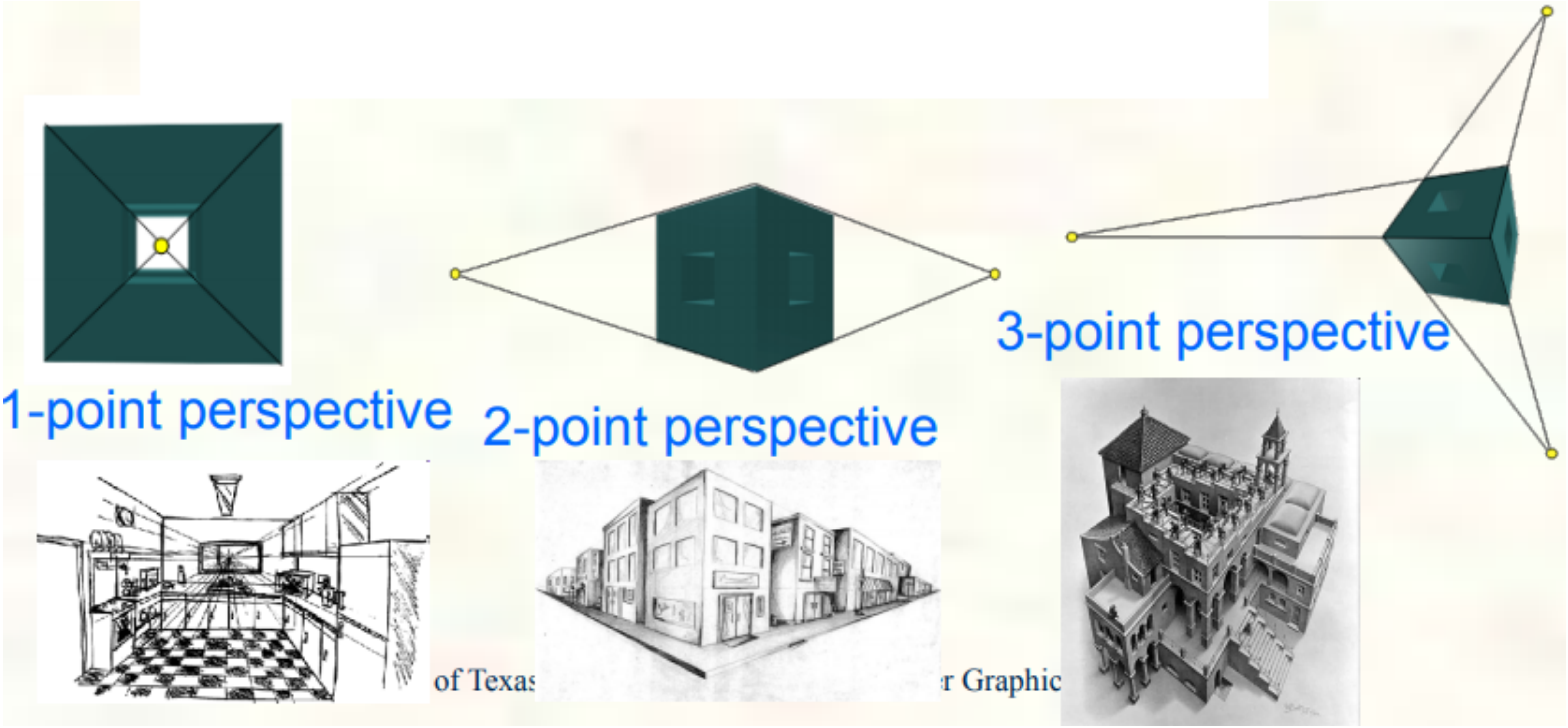
Filippo Brunelleschi Florence, 1415



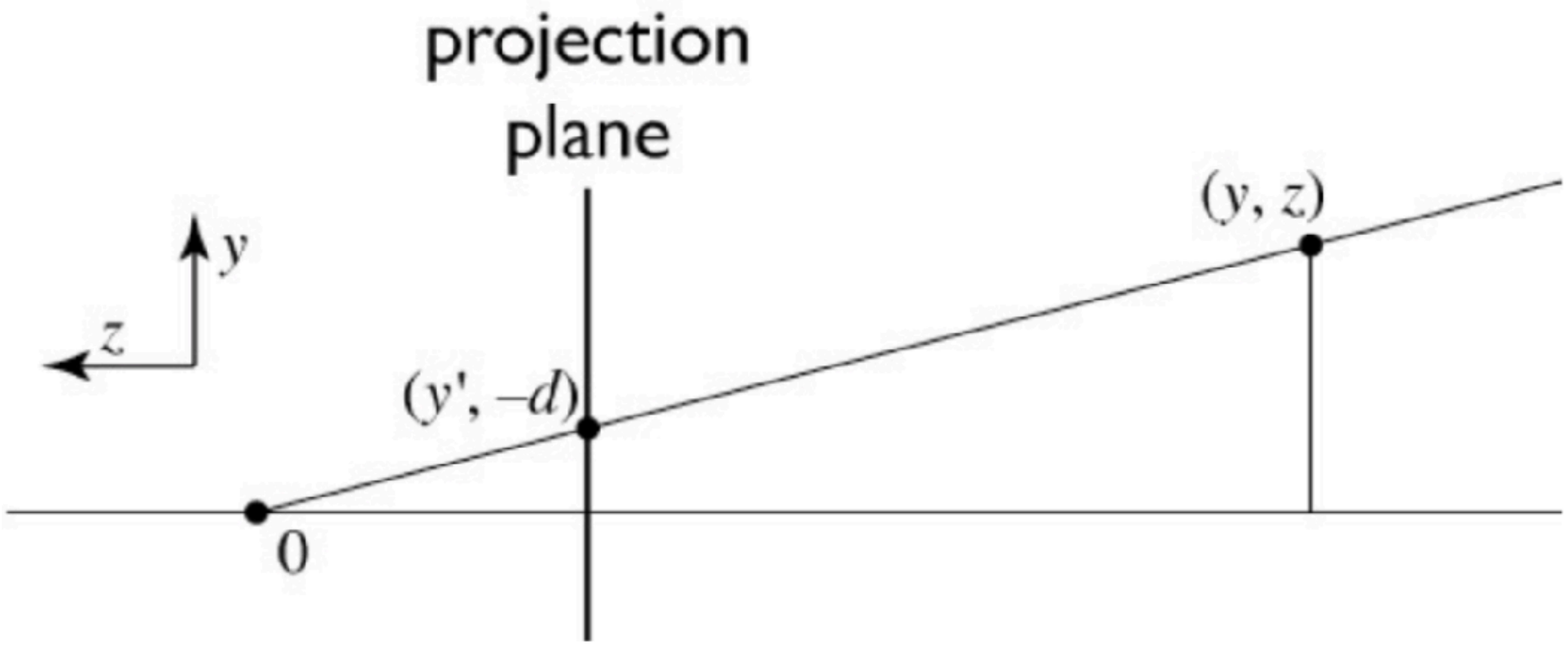
Brunelleschi, elevation of Santo Spirito,1434-83, Florence

Masaccio – The Tribute Money c. 1426-27
Fresco, The Brancacci Chapel, Florence

1-, 2-, and 3-point Perspective



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**

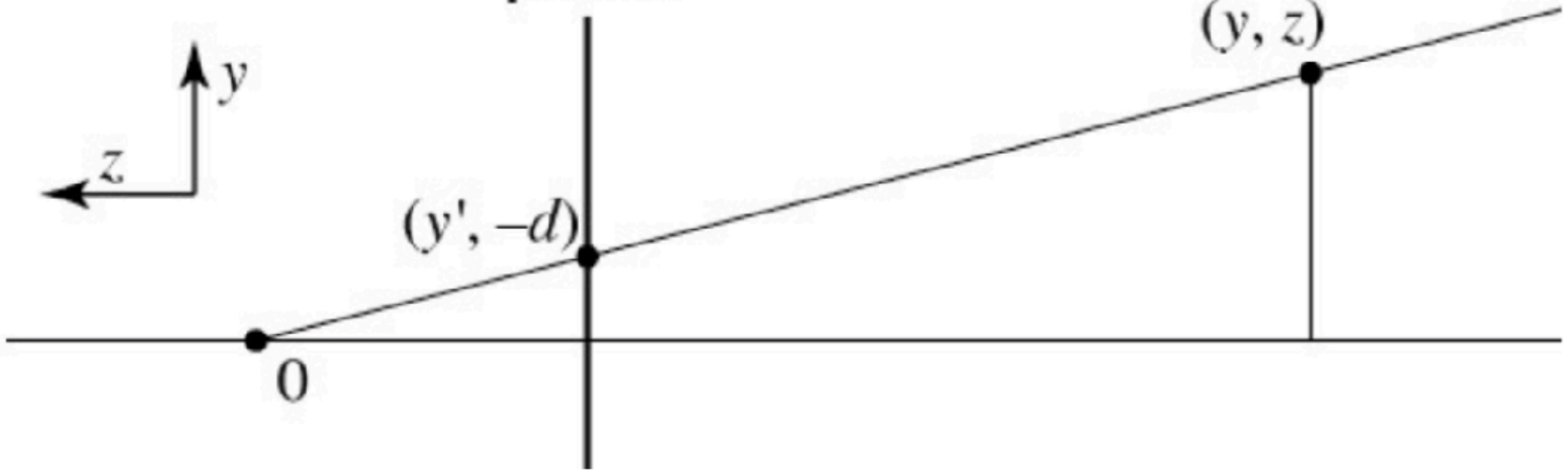
$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \sim \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix}$$

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

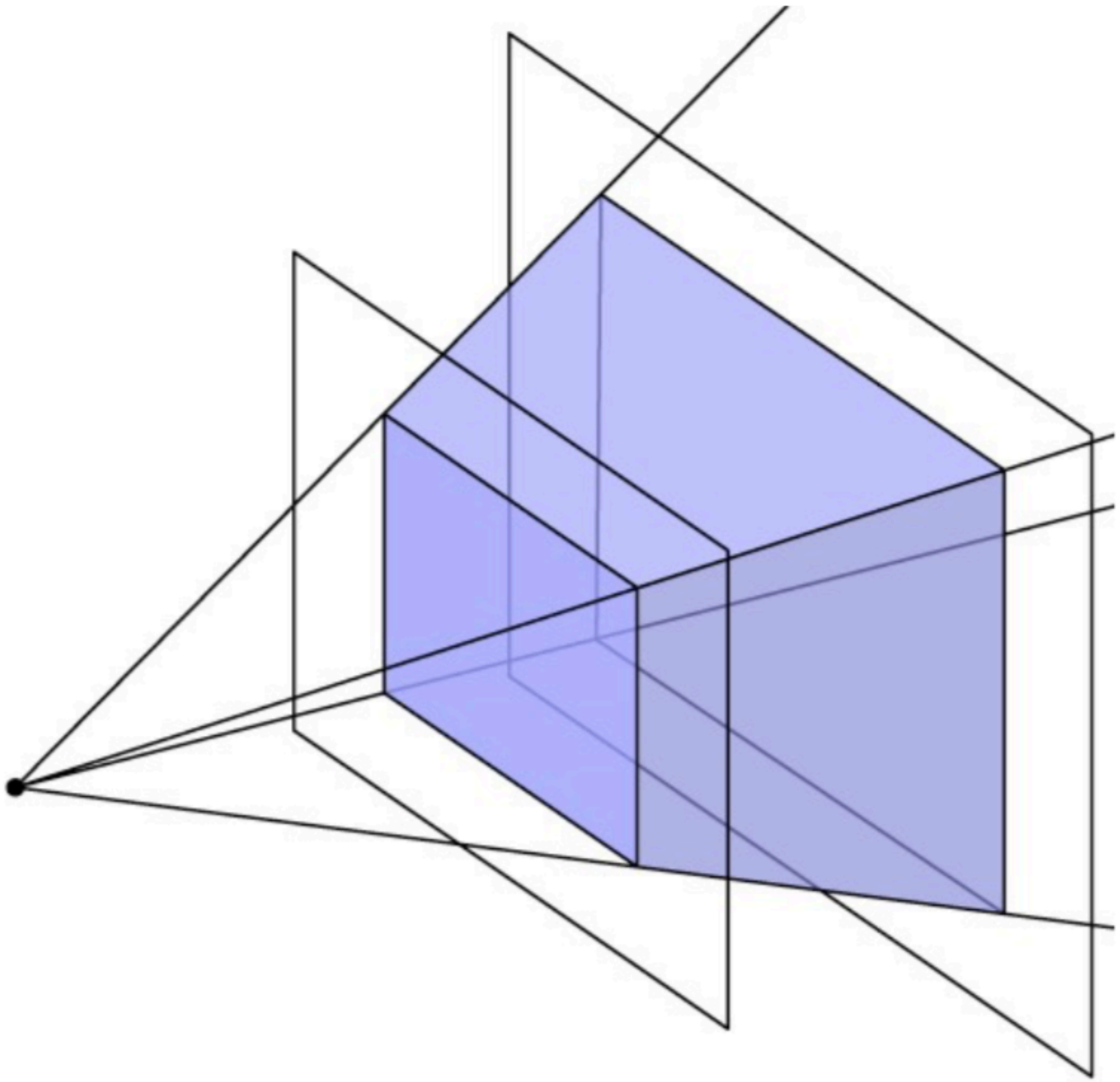
Perspective projection



to implement perspective, just move z to w:

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

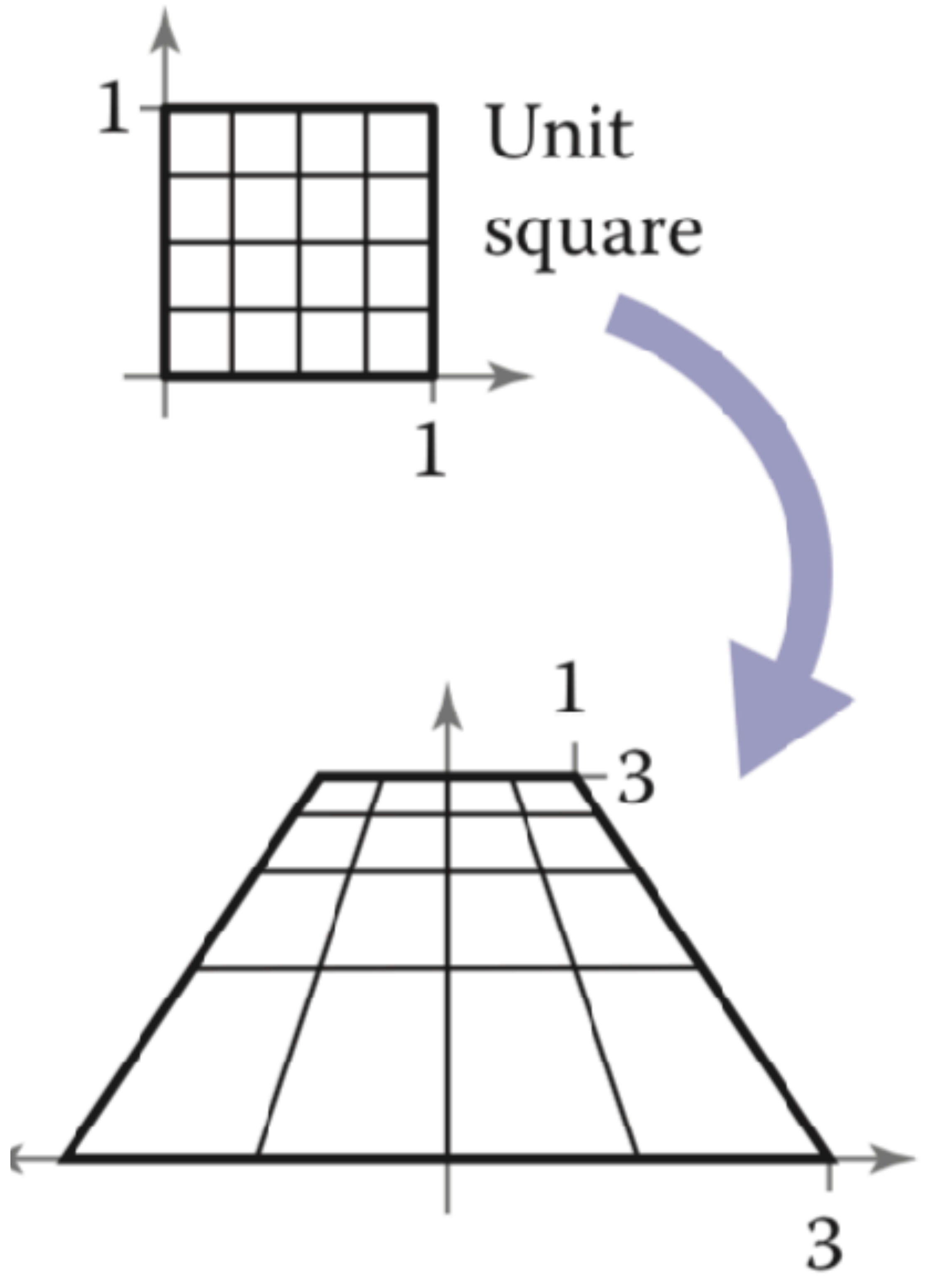
View volume: perspective (clipped)



Carrying depth through perspective

- Perspective 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 $z'(f) = f$.

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

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

want $z'(n) = n$ and $z'(f) = f$

result: $a = -(n + f)$ and $b = nf$ (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**

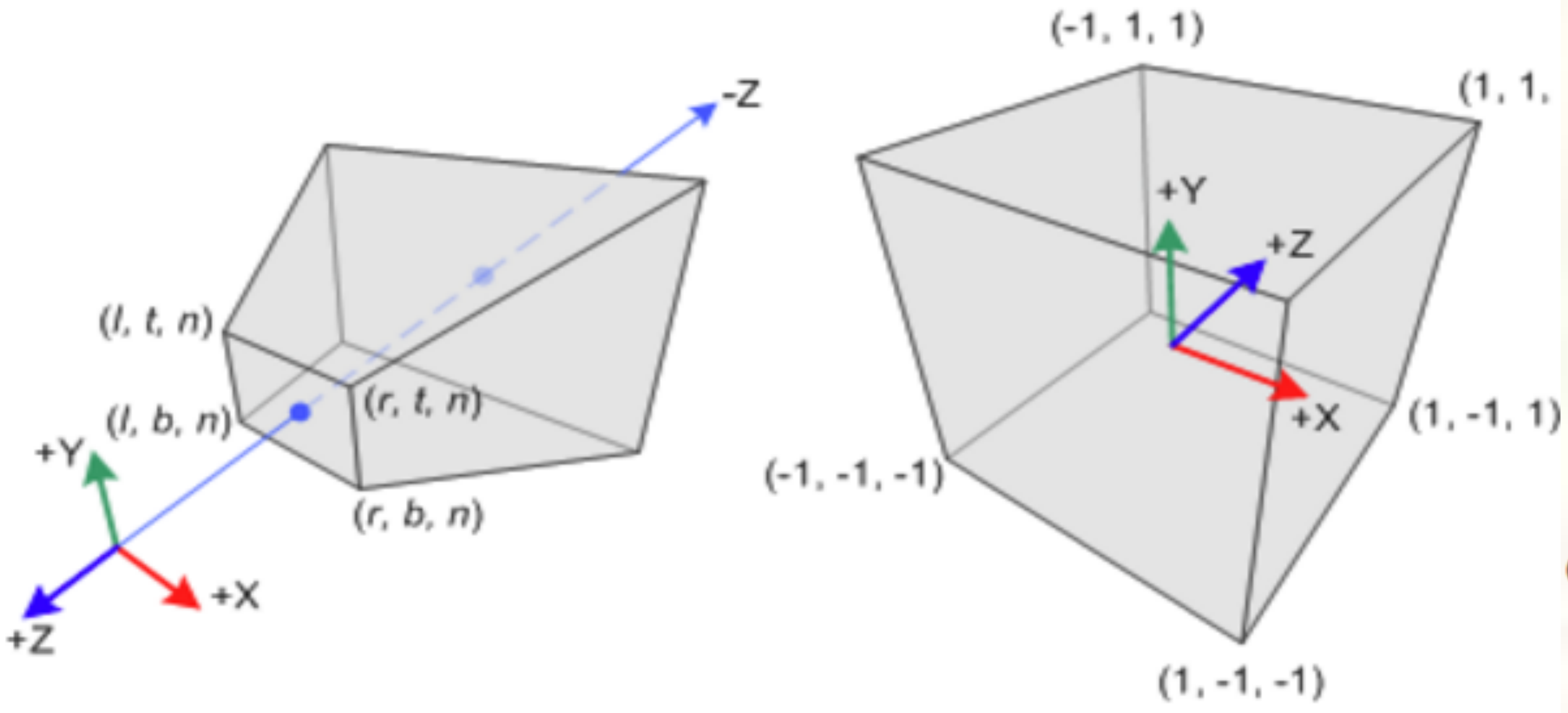
$$\mathbf{M}_{\text{per}} = \mathbf{M}_{\text{orth}}\mathbf{P}$$

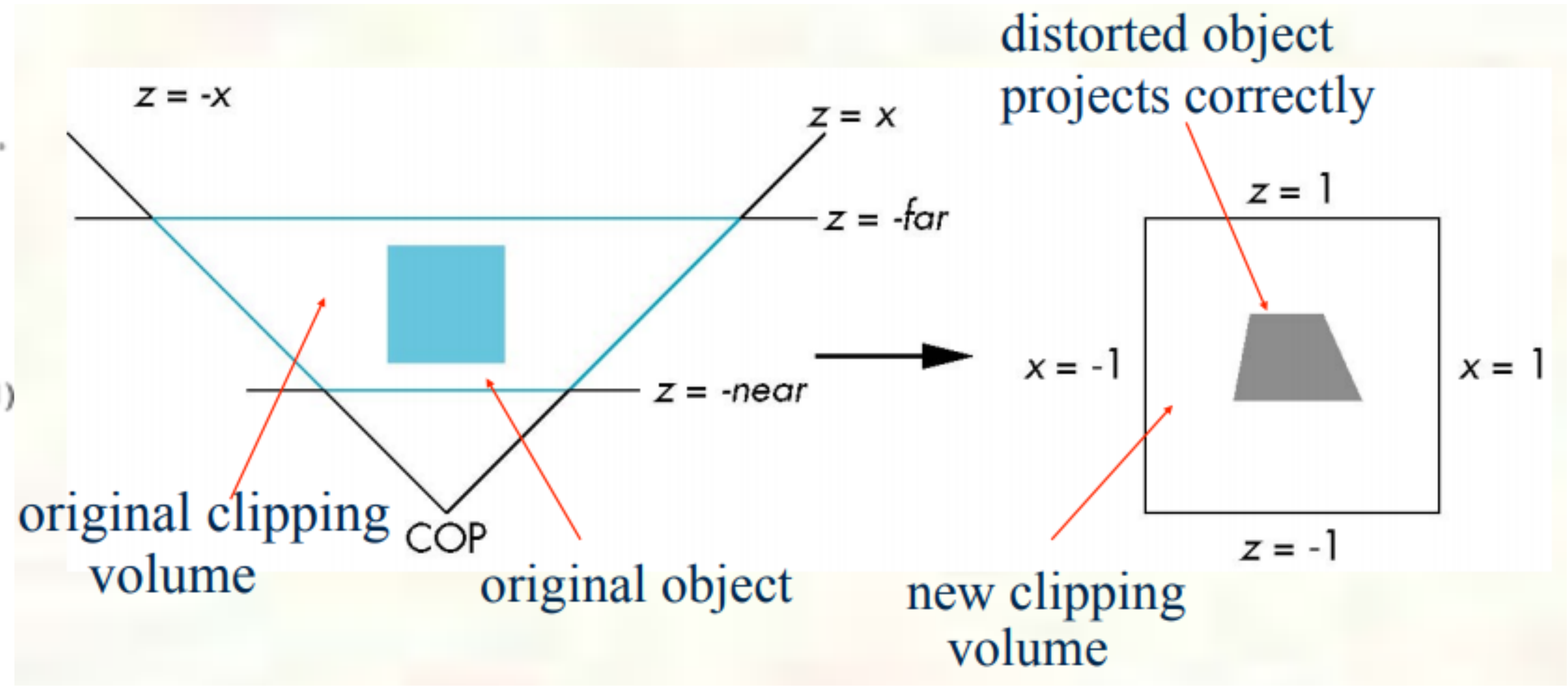
$$= \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

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

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 projection matrix





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

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

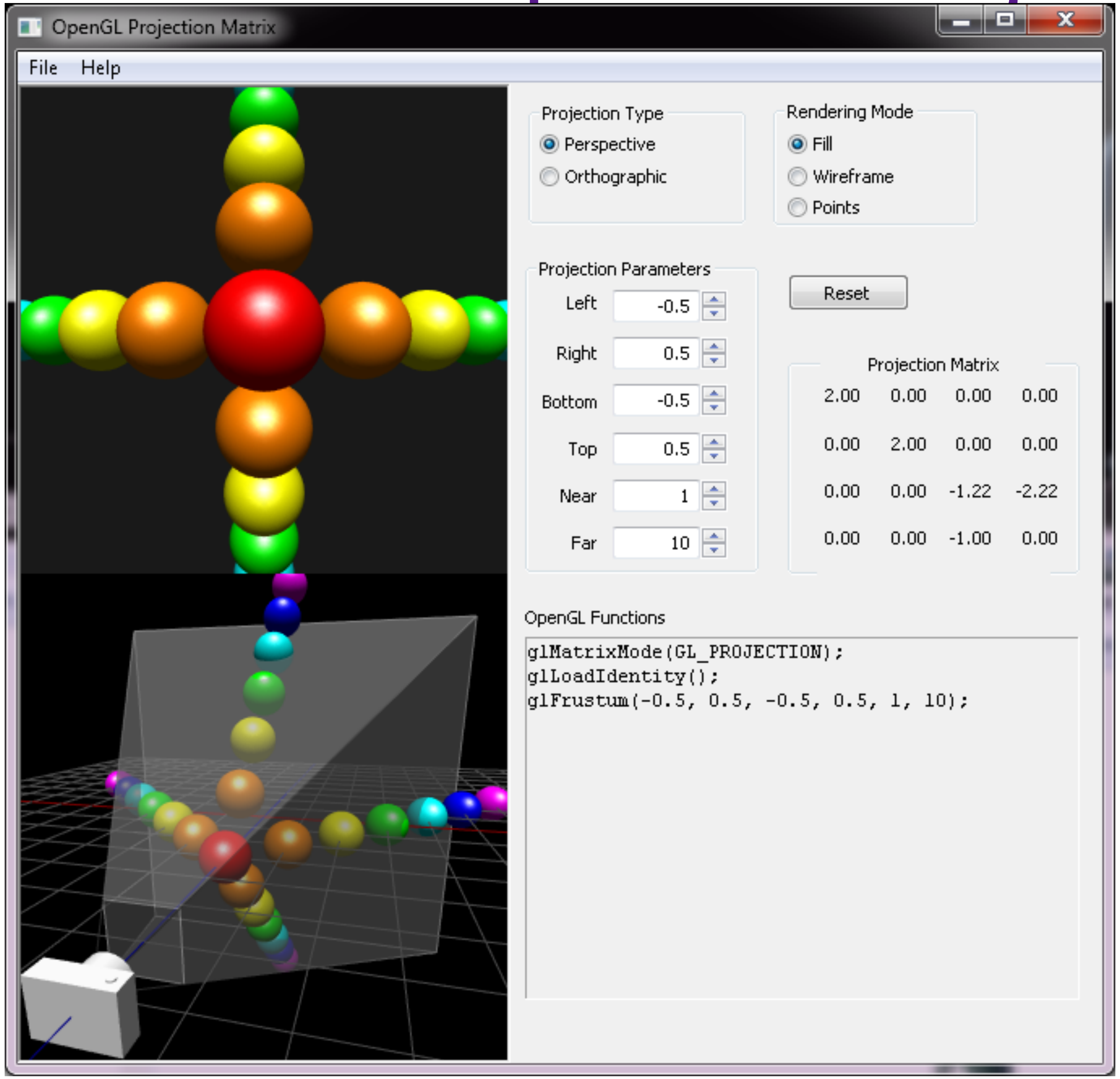
Perspective transformation chain

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

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

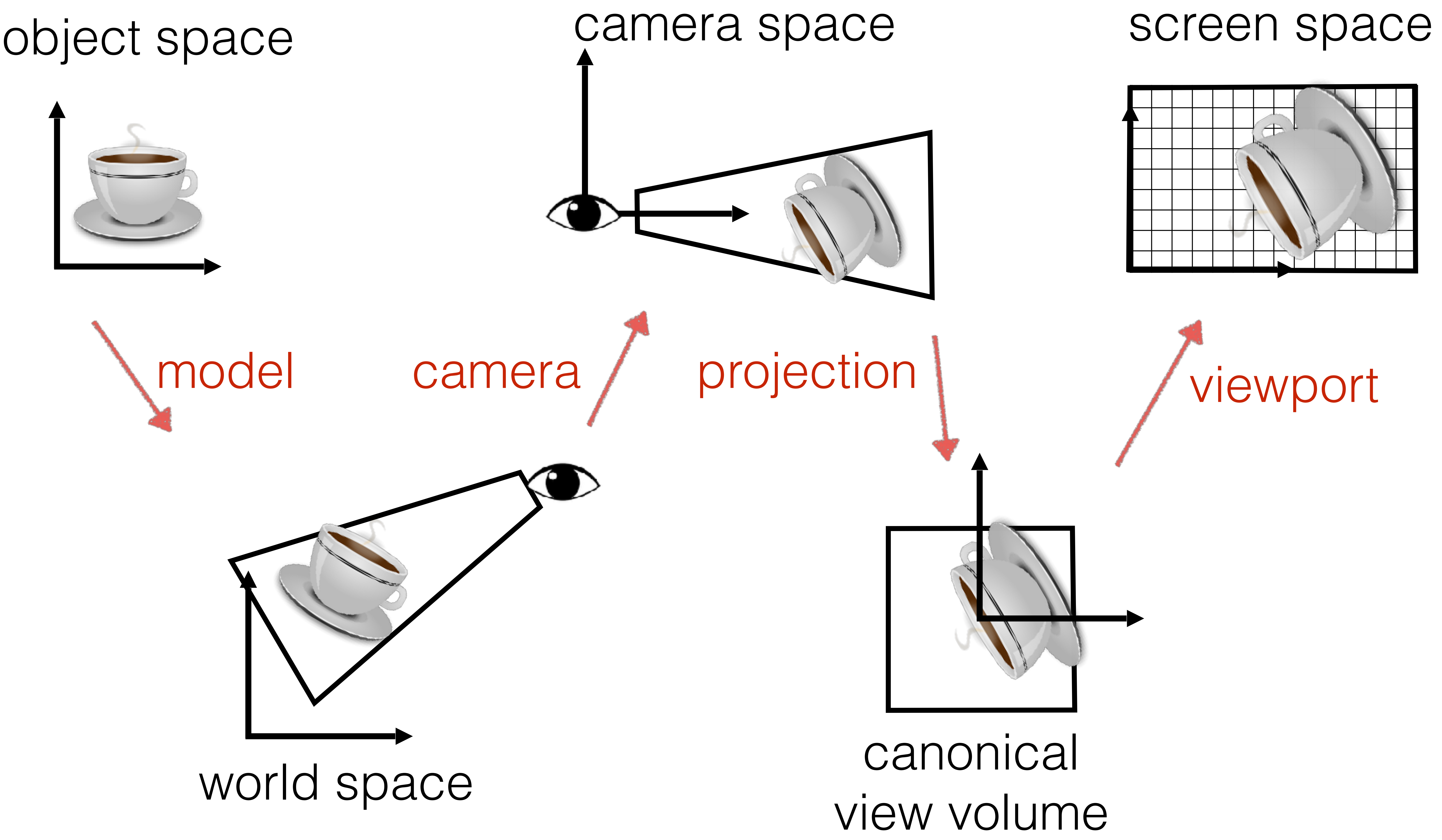
$$\underbrace{\begin{bmatrix} x_s \\ y_s \\ z_c \\ 1 \end{bmatrix}}_{\text{screen space}} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y-1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \underbrace{\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l}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

Example: Projection Matrix



- http://www.songho.ca/opengl/gl_transform.html#projection

Viewing Transformation



References

Fundamentals of Computer Graphics, Fourth Edition
4th Edition **by Steve Marschner, Peter Shirley**

Chapter 7