CS230 : Computer Graphics

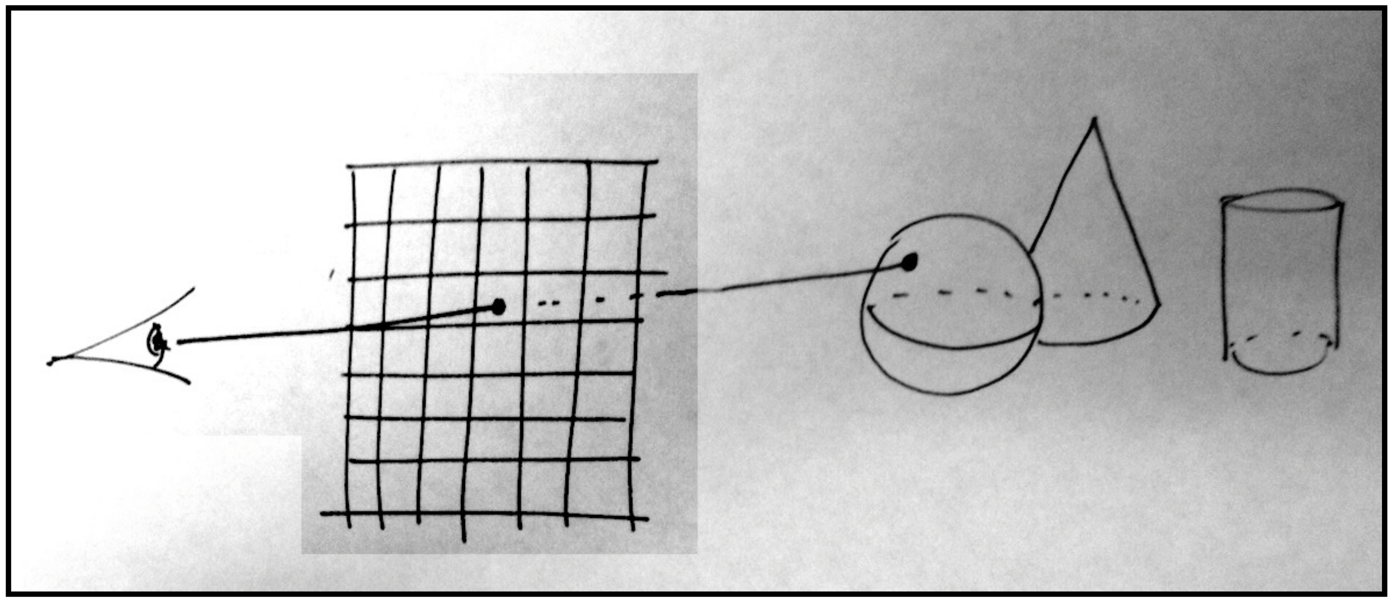
Lecture 6: Viewing Transformations

Tamar Shinar
Computer Science & Engineering
UC Riverside

Rendering approaches

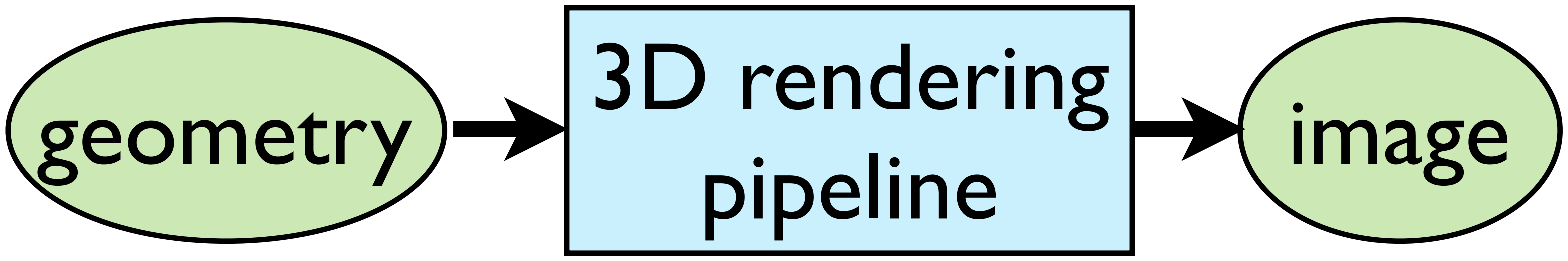
1. image-oriented

foreach pixel ...



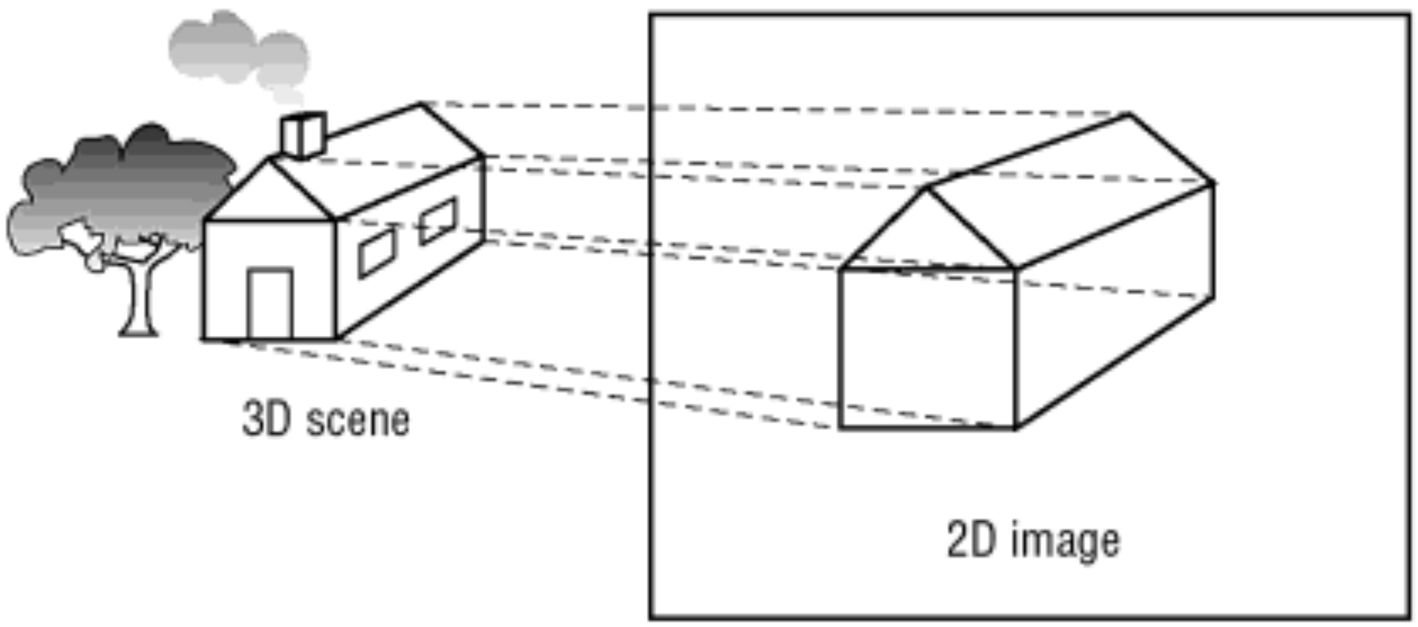
2. object-oriented

foreach object ...



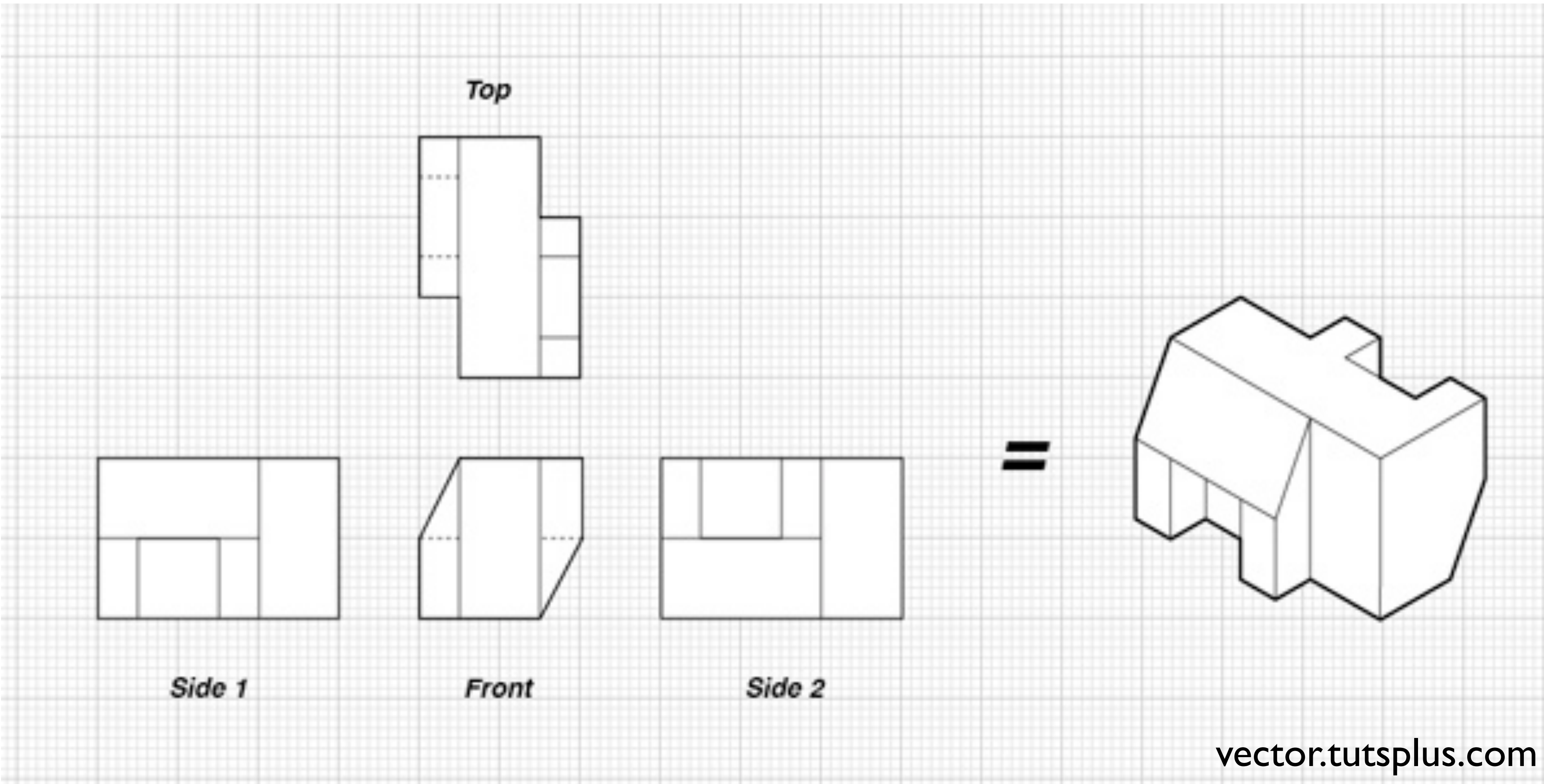
object-oriented rendering – e.g., OpenGL graphics pipeline, Renderman (REYES)
task: figure out where a point in the geometry will land on the final image pixels

Projection: map 3D scene to 2D image



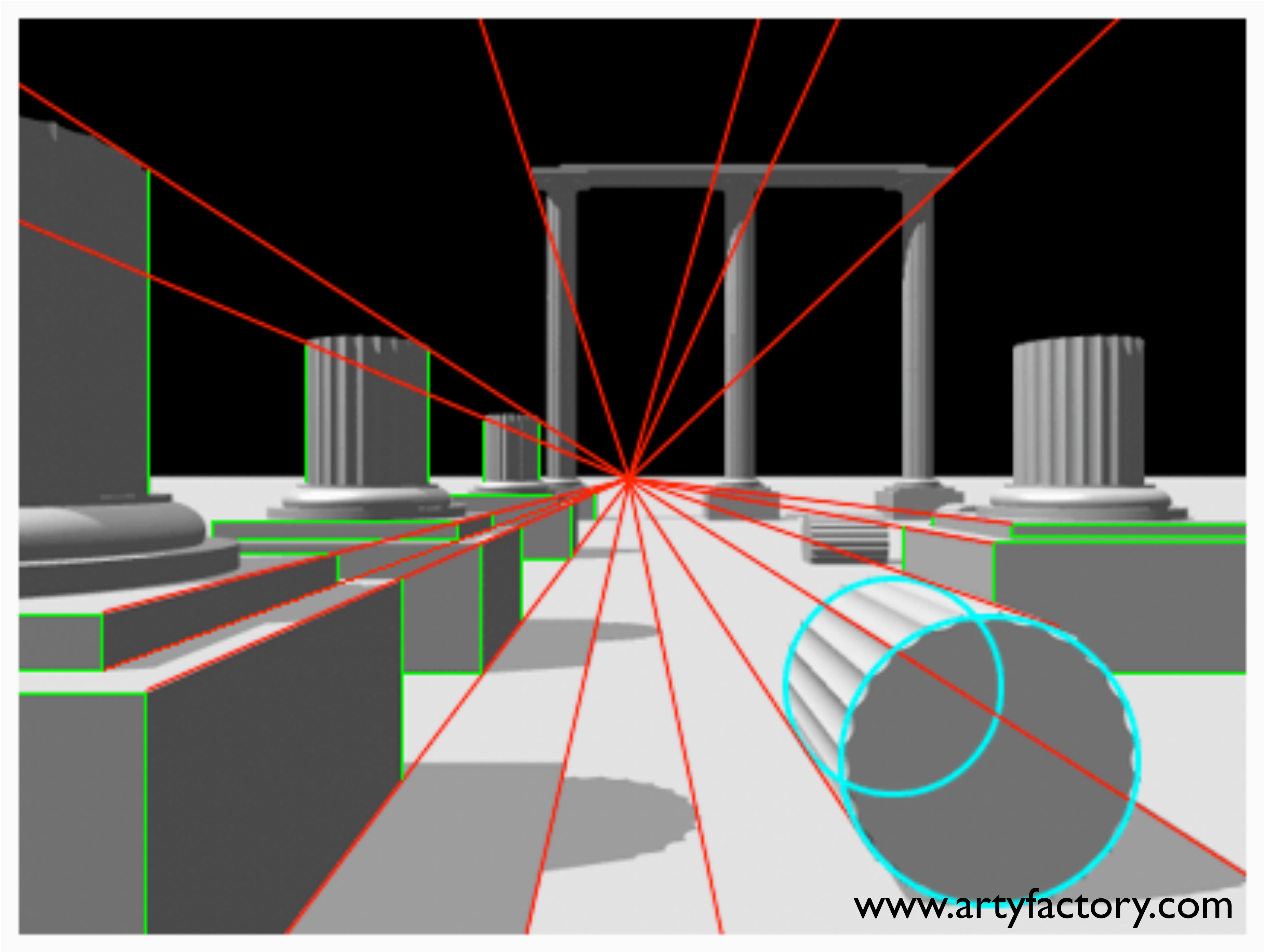
OpenGL Super Bible, 5th Ed.

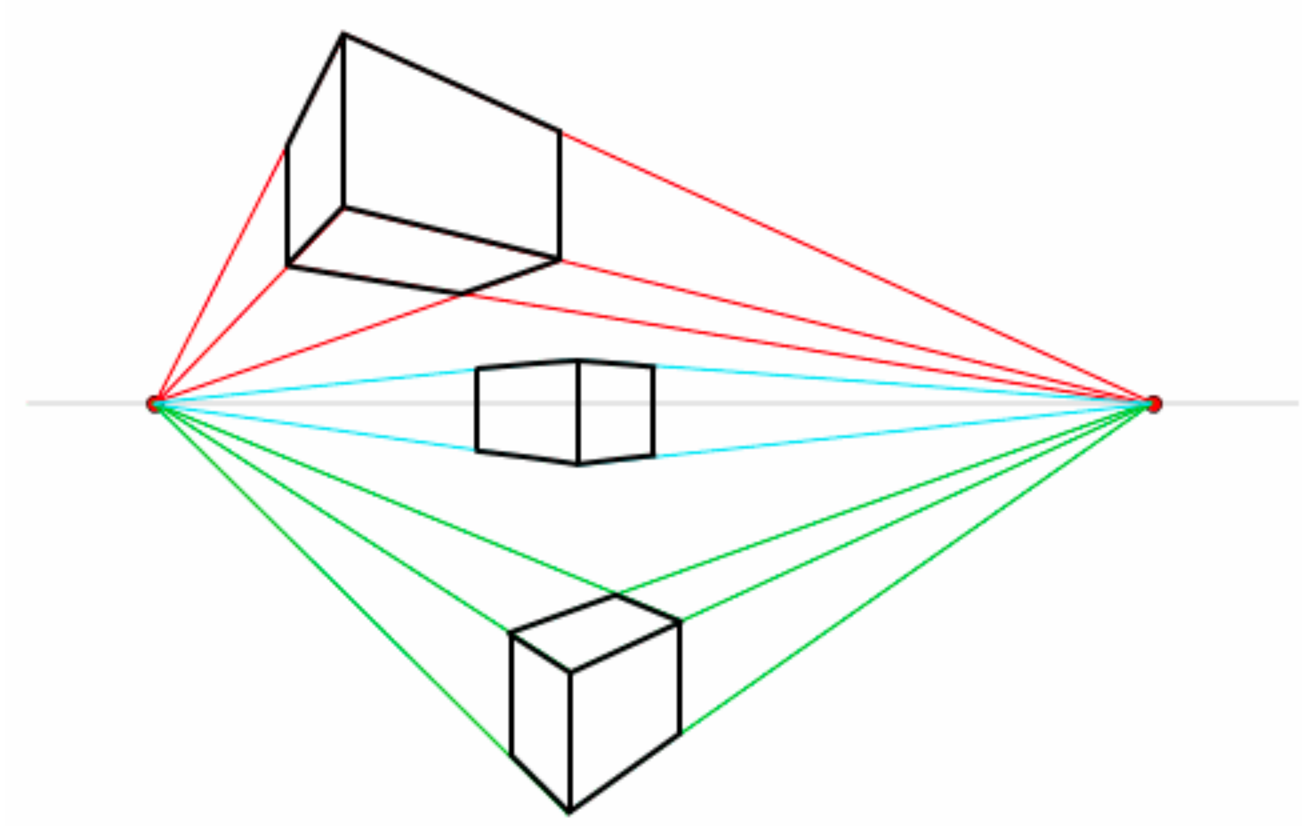
Orthographic projection



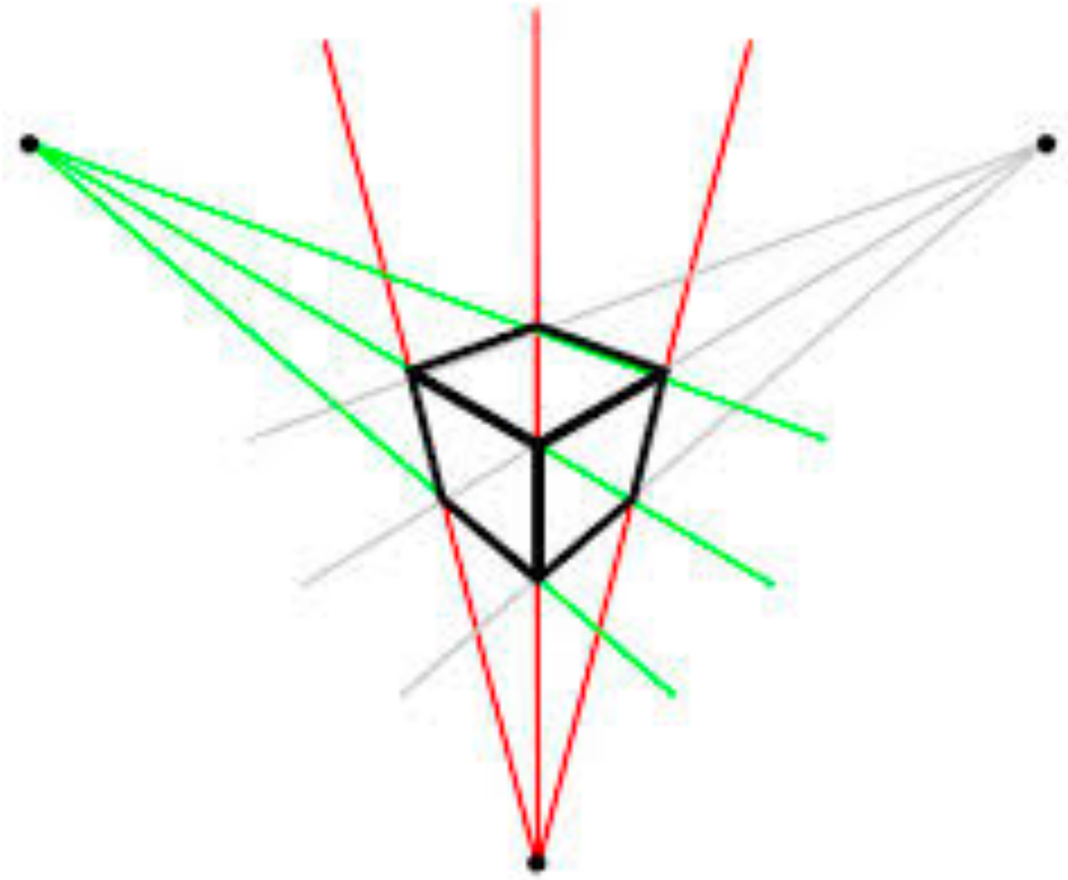
Orthographic, or parallel projection
- parallel lines appear parallel (unlike perspective proj.)
- equal length lines appear equal length (unlike perspective proj.)

Perspective projection



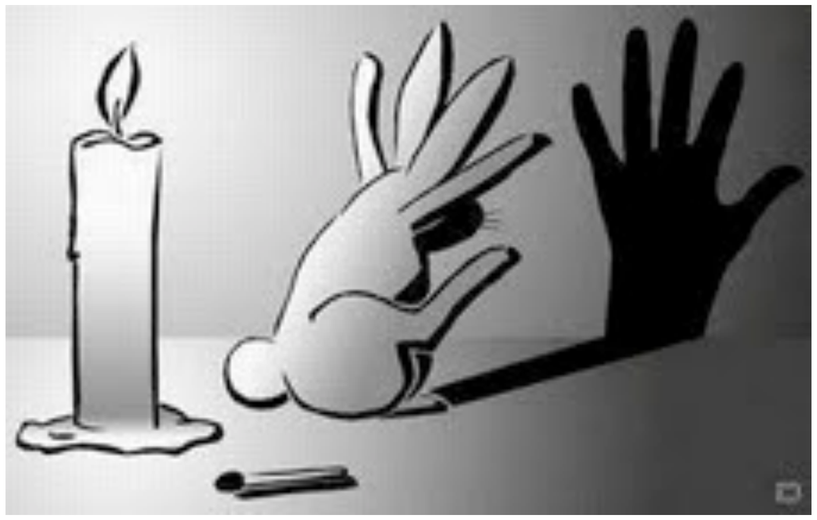


two-point perspective

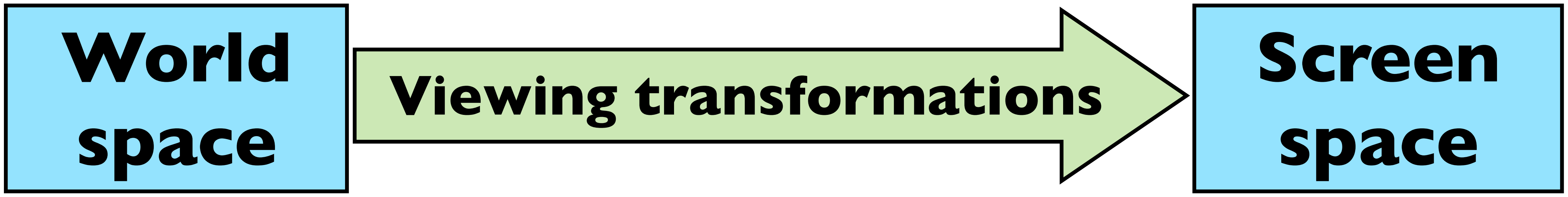


three-point perspective

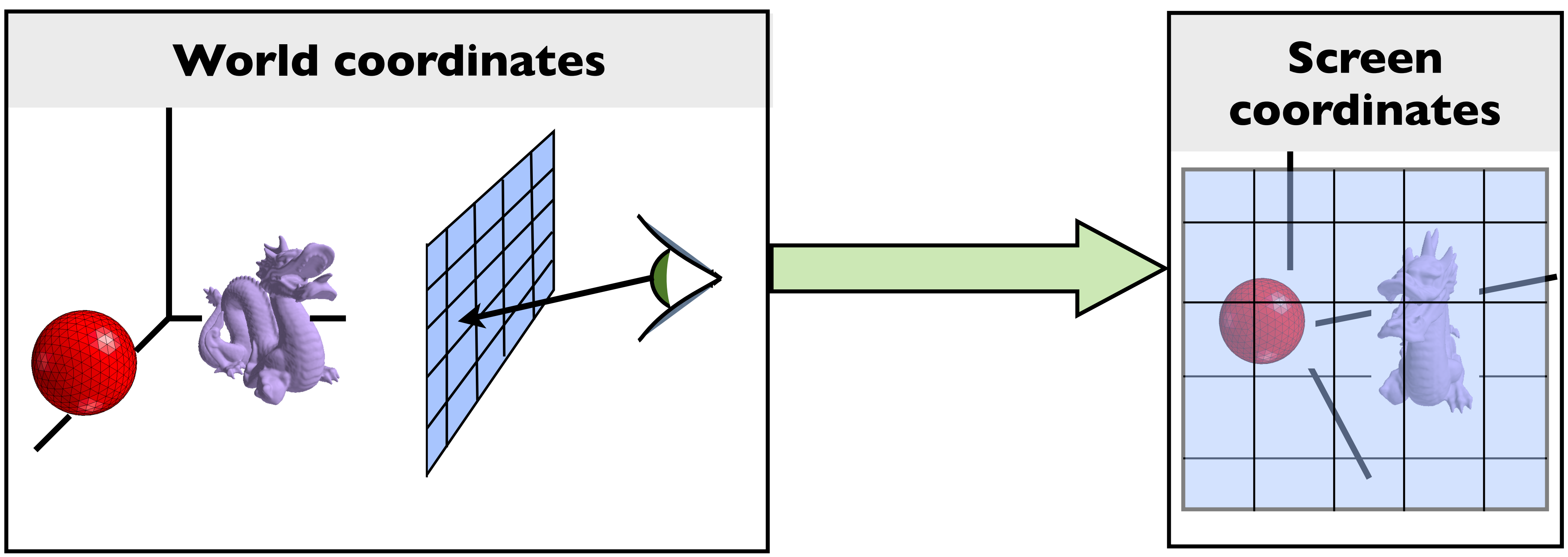
Viewing Transformations



Viewing transformations

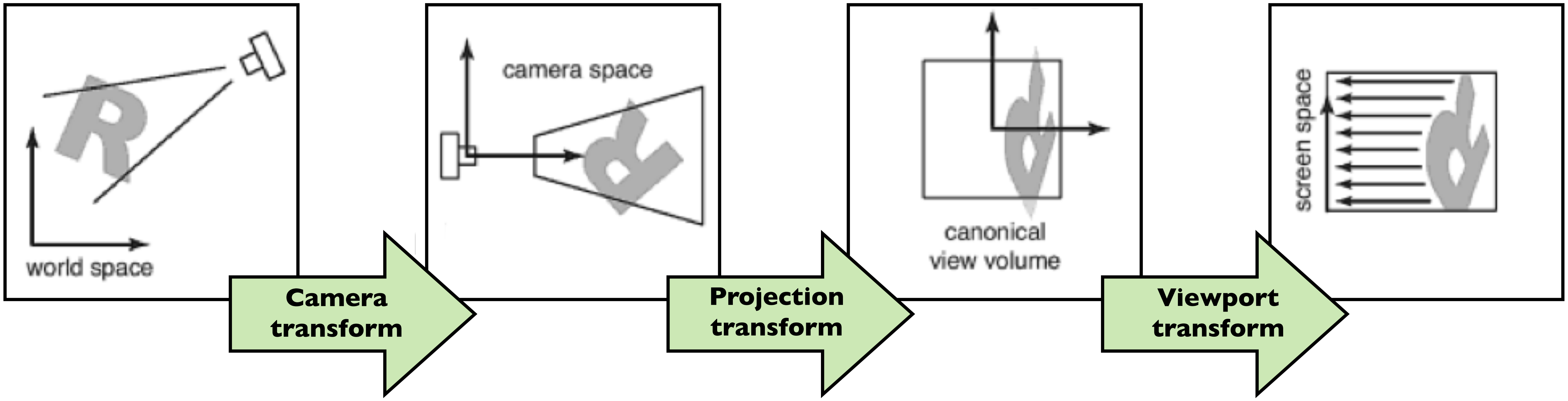


- Map points from their 3D locations to their positions in a 2D view



The viewing transformation also project any point along pixel's viewing ray back to the pixel's position in **screen (or image) space**

Decomposition of viewing transforms



- rigid body transformation
- place camera at origin

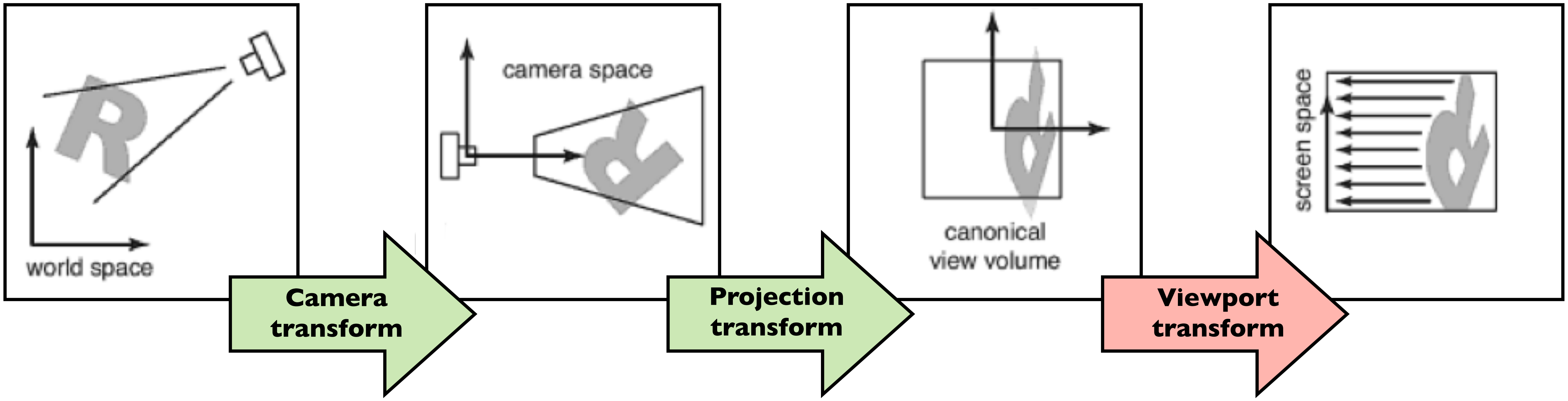
- x, y, z in [-1,1]
- depends on type of projection

- map to pixel coordinates

Viewing transforms depend on: camera position and orientation, type of projection, field of view, image resolution

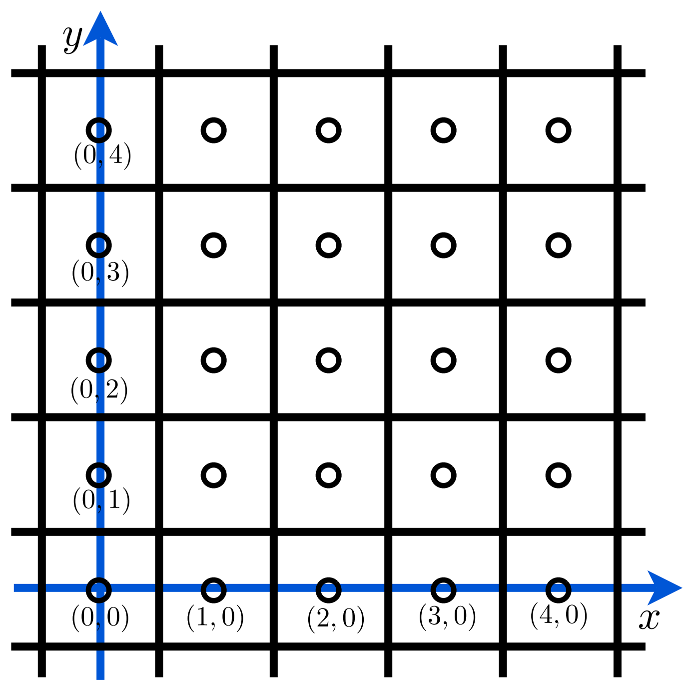
there are several names for these spaces: “camera space” = “eye space”, “canonical view volume” = “clip space”= “normalized device coordinates”, “screen space=pixel coordinates” and for the transforms: “camera transformation” = “viewing transformation”

Viewport transform

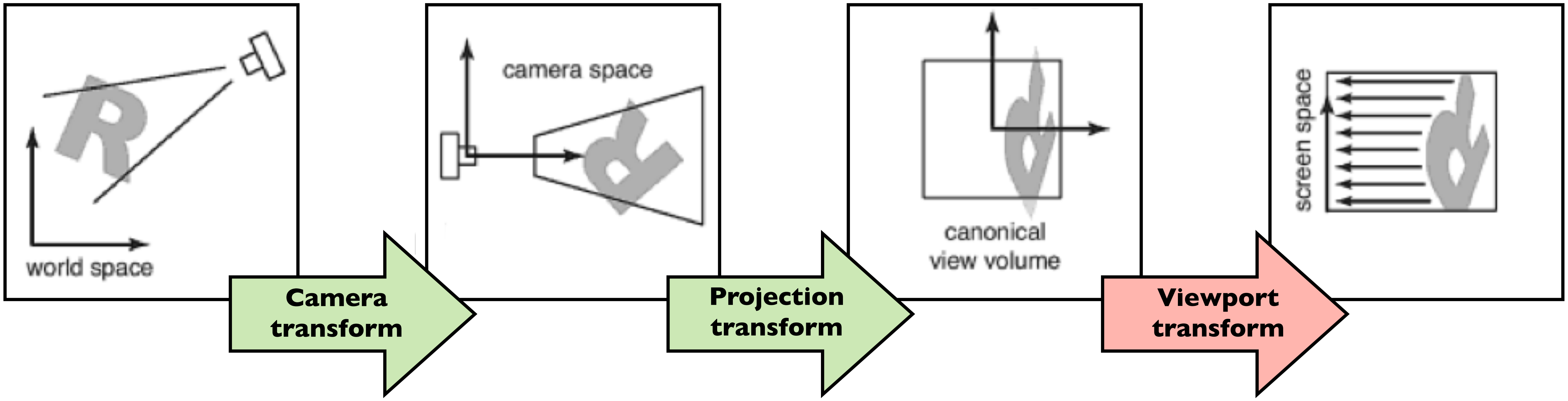


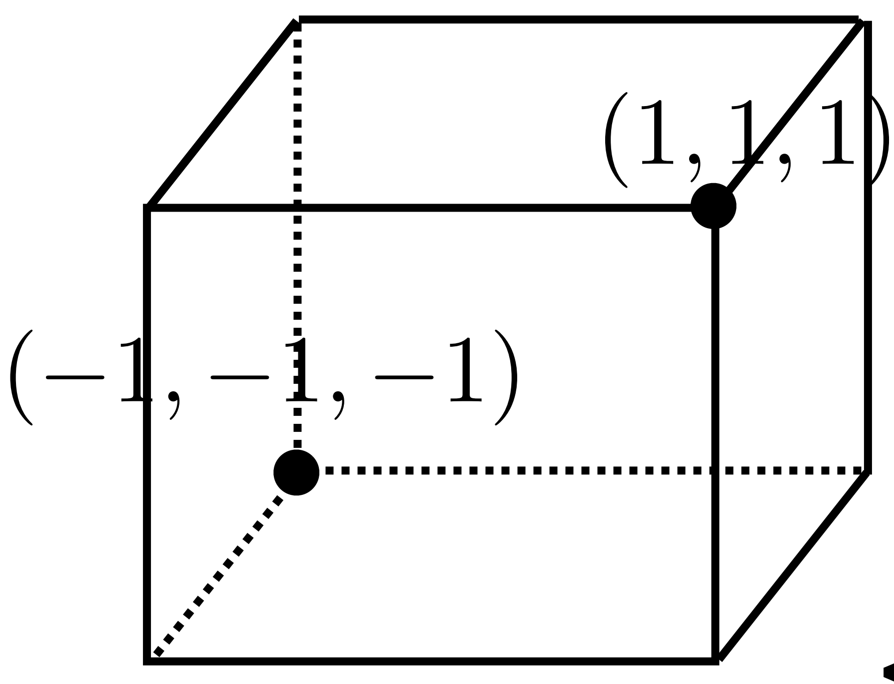
$$(x, y, z) \to (x', y', z')$$

$$(x, y, z) \in [-1, 1]^3 \qquad \begin{aligned} x' &\in [-.5, n_x - .5] \\ y' &\in [-.5, n_y - .5] \\ z' &\in [-1, 1] \end{aligned}$$



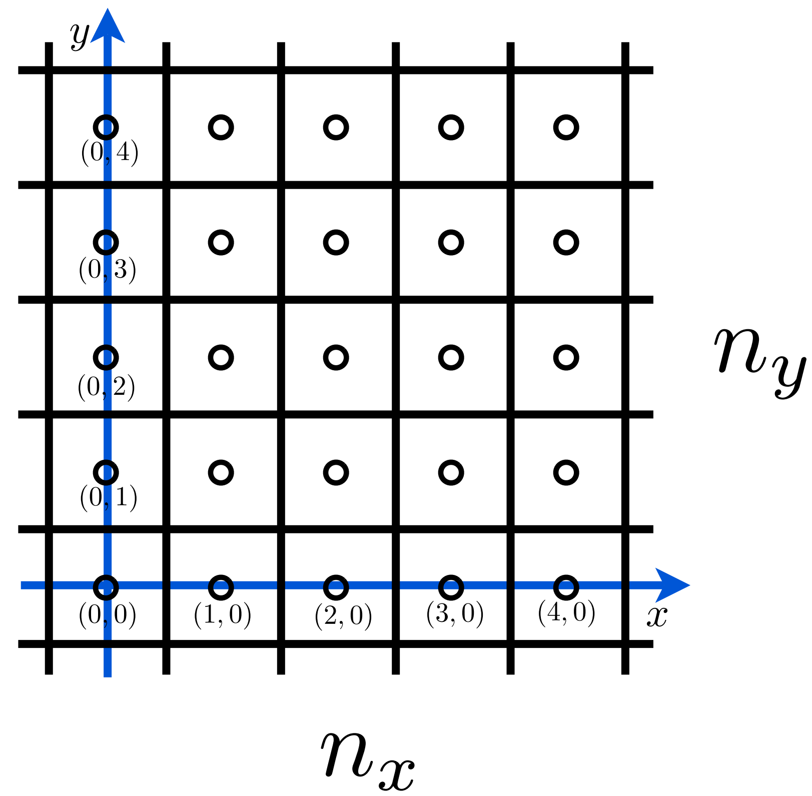
Viewport transform



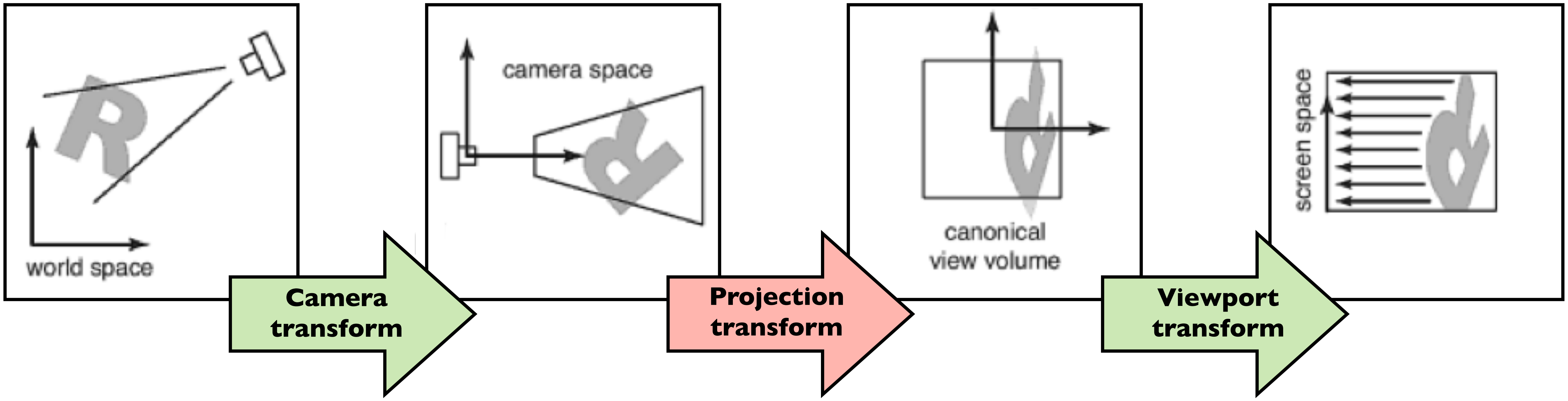


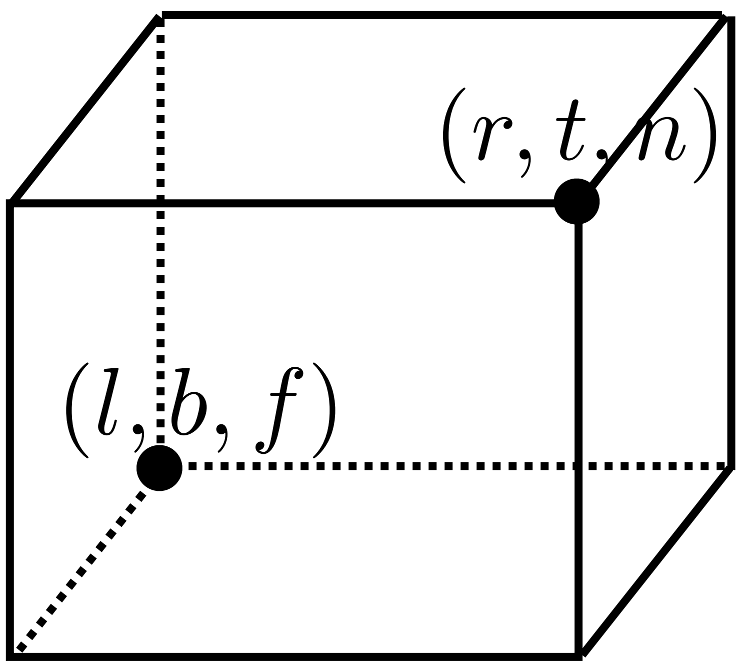
M_{vp}

<whiteboard>

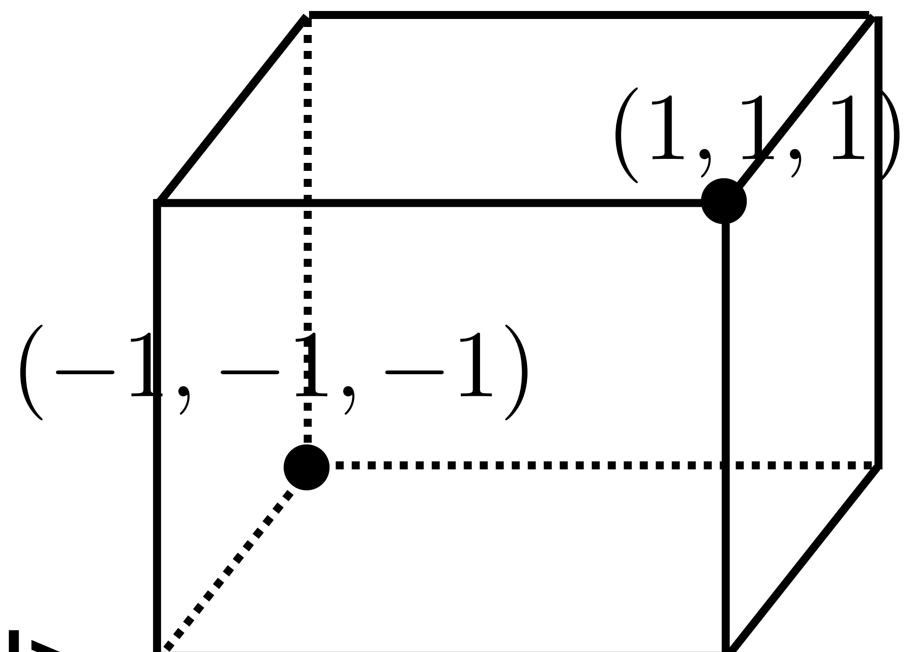


Orthographic Projection Transform



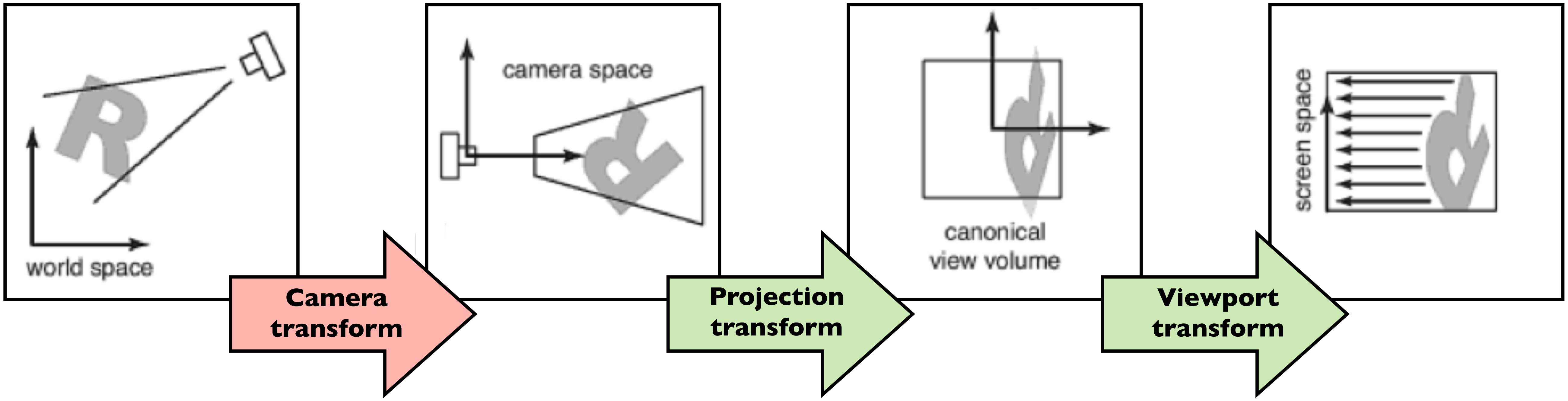


M_{orth}



<whiteboard>

Camera Transform



Camera Transform

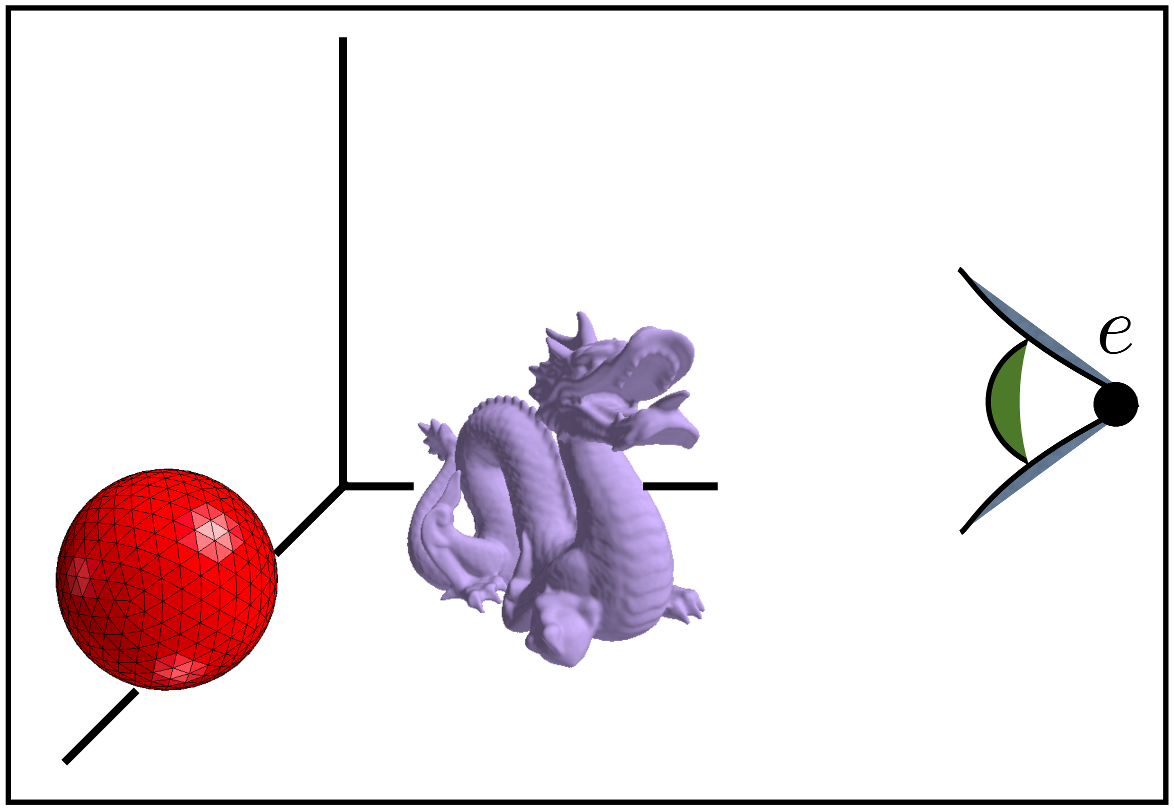
How do we specify the camera configuration?

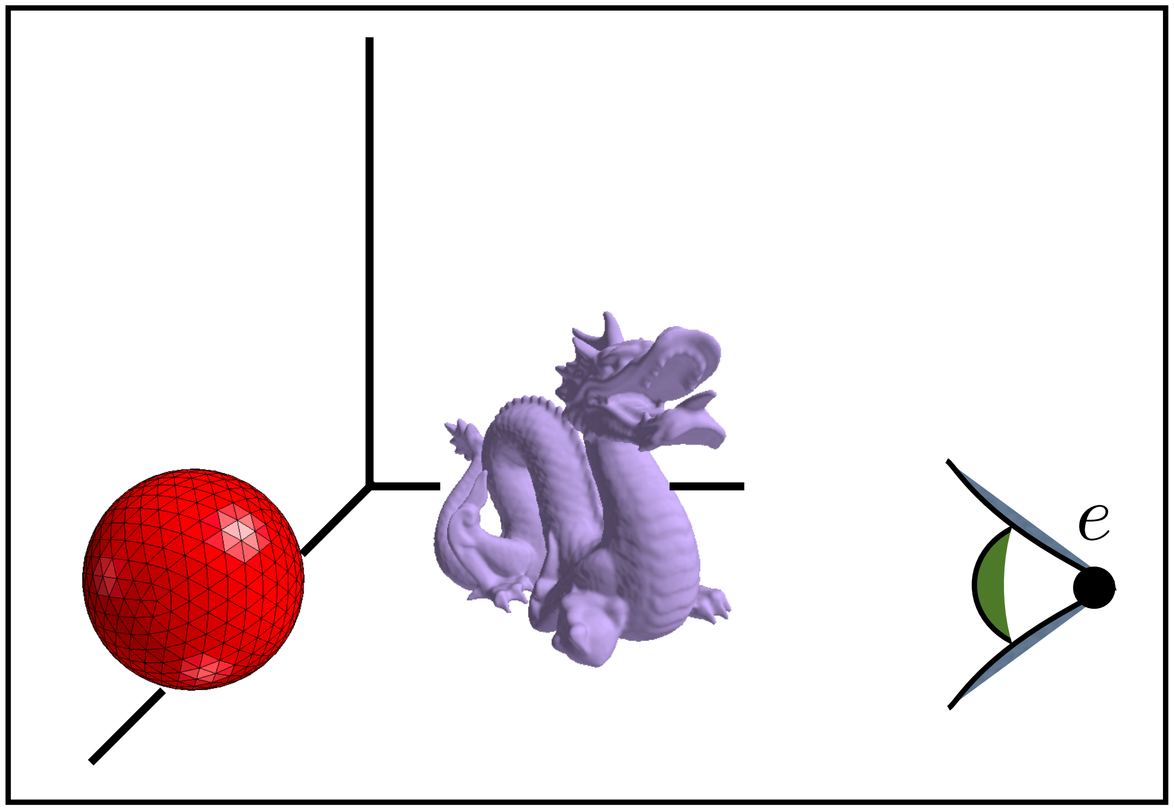
(orthogonal case)

Camera Transform

How do we specify the camera configuration?

eye position

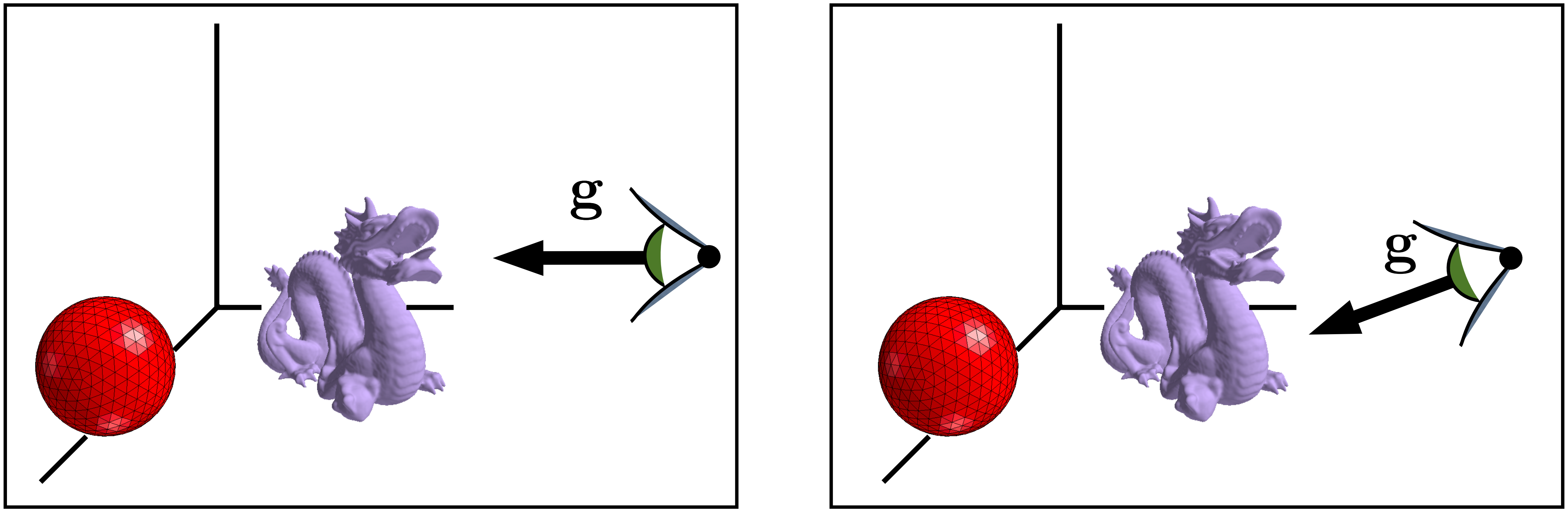




Camera Transform

How do we specify the camera configuration?

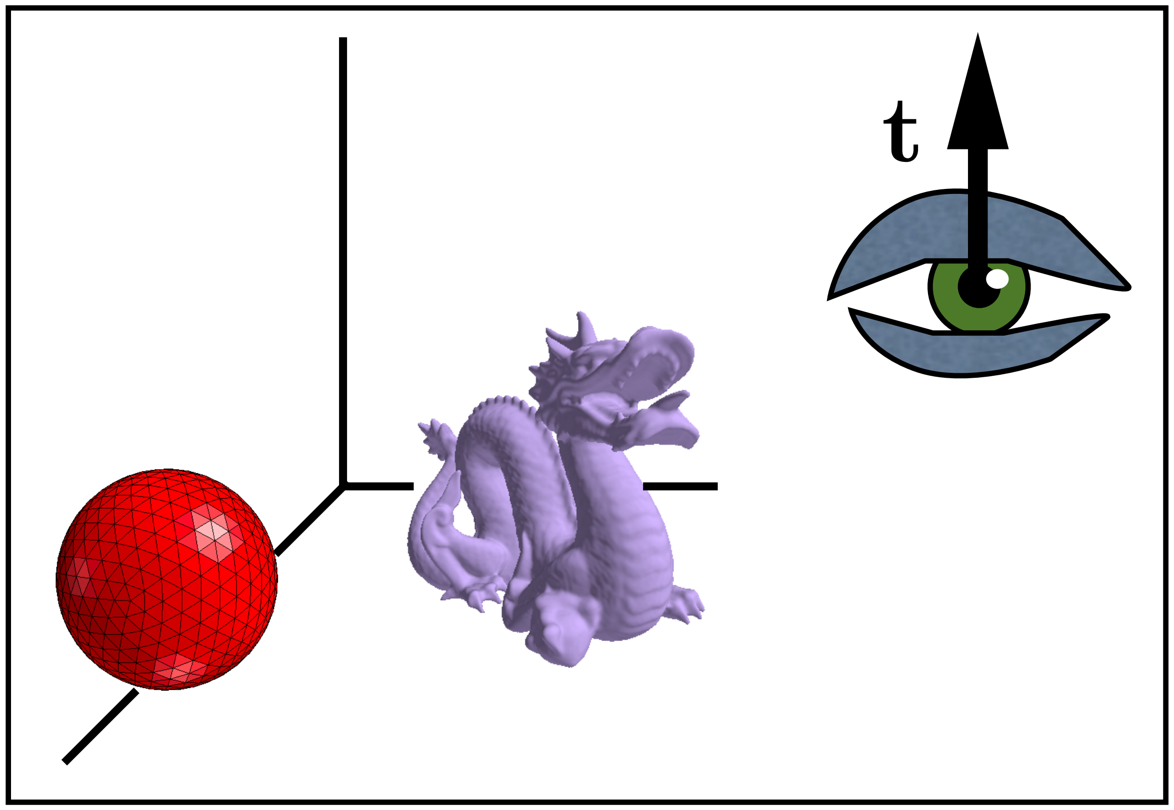
gaze direction

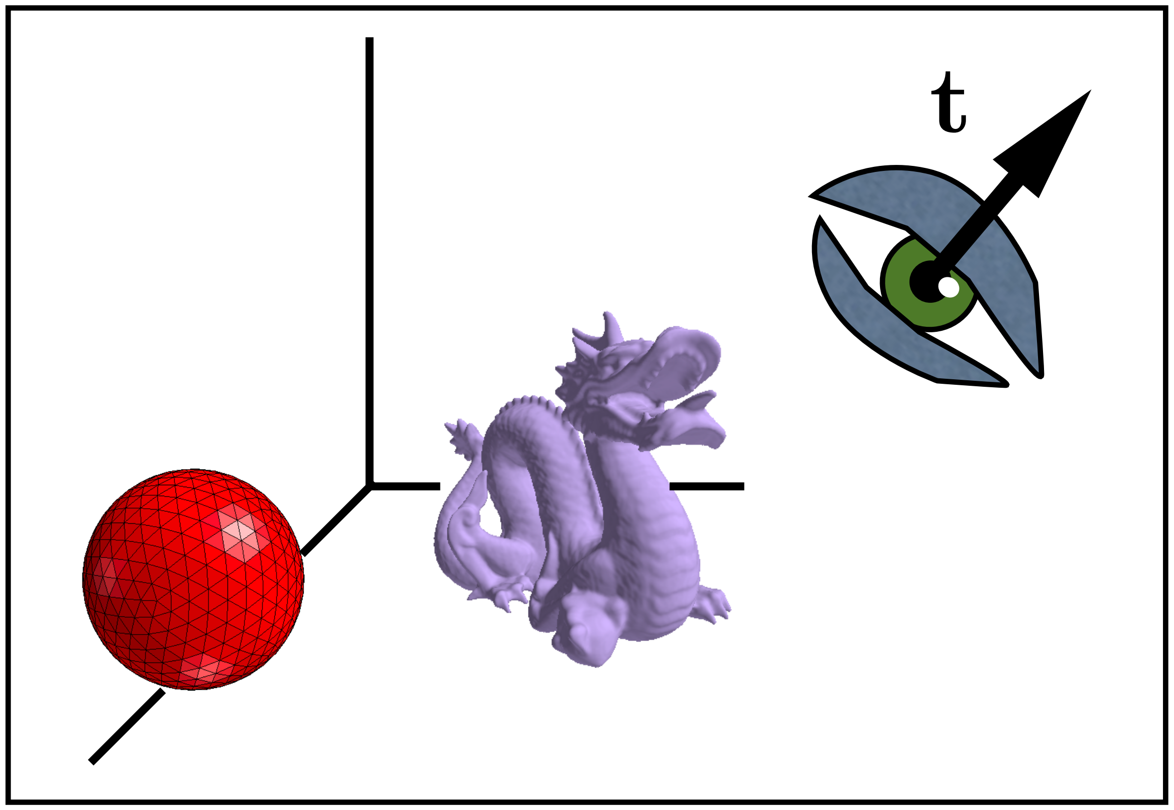


Camera Transform

How do we specify the camera configuration?

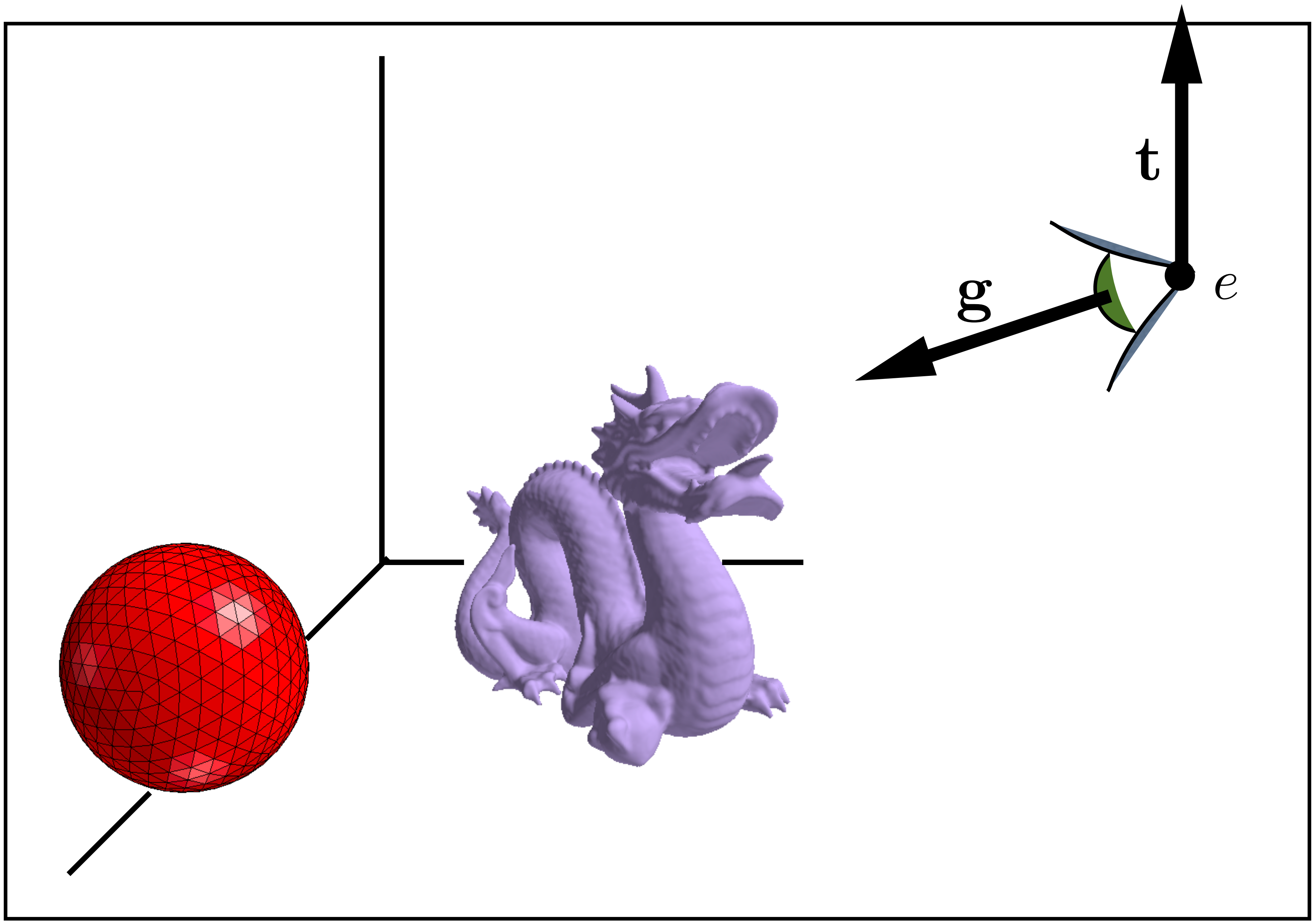
up vector



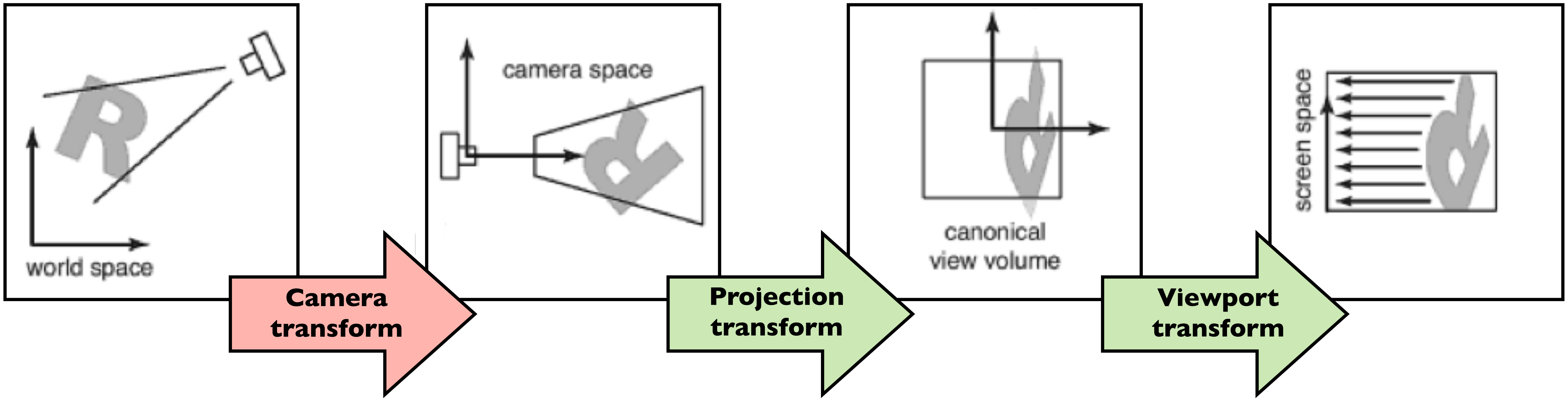


Camera Transform

How do we specify the camera configuration?



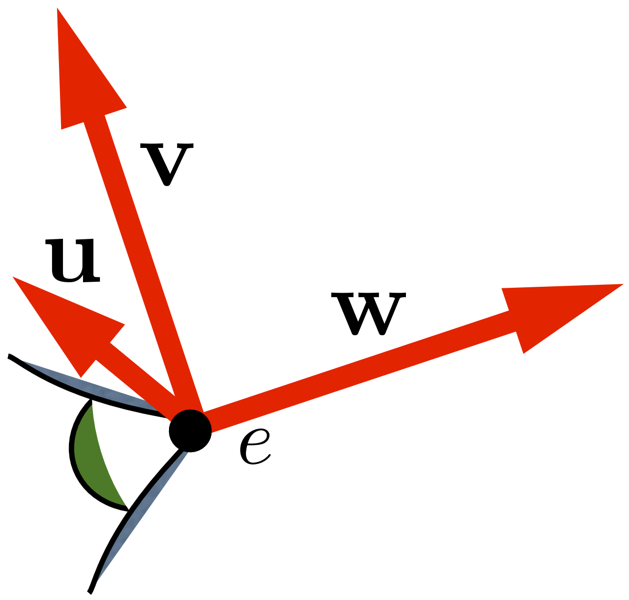
Camera Transform

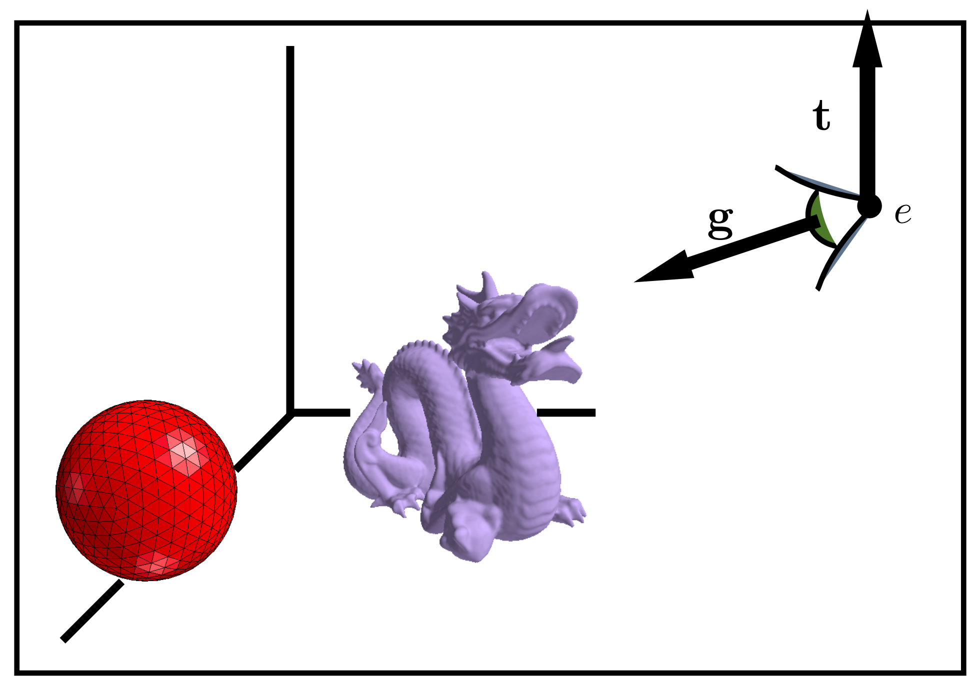


$$\mathbf{w} = -\frac{\mathbf{g}}{||\mathbf{g}||}$$

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

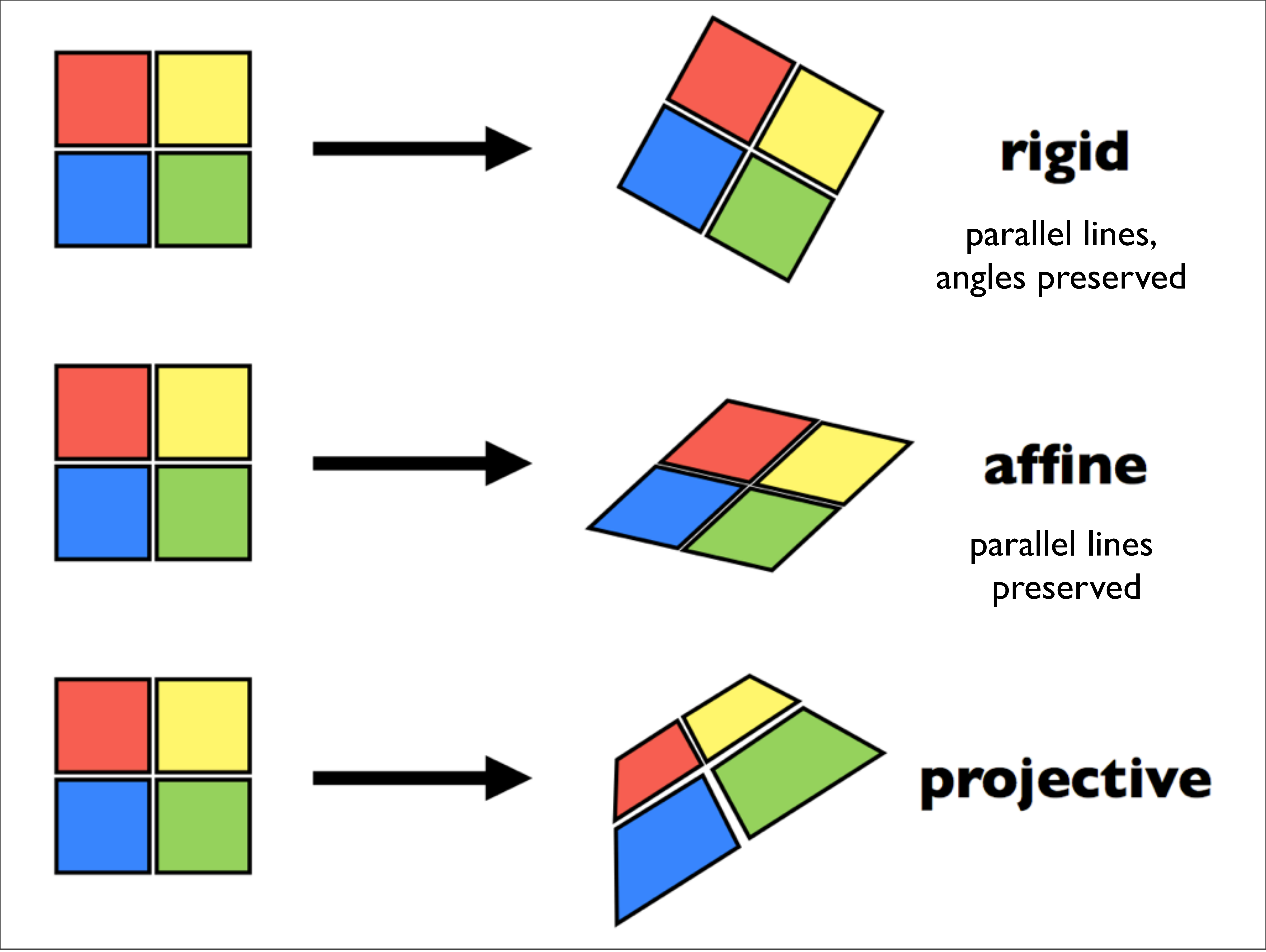
$$\mathbf{v} = \mathbf{w} \times \mathbf{u}$$





M_{cam} <whiteboard>

Perspective Viewing

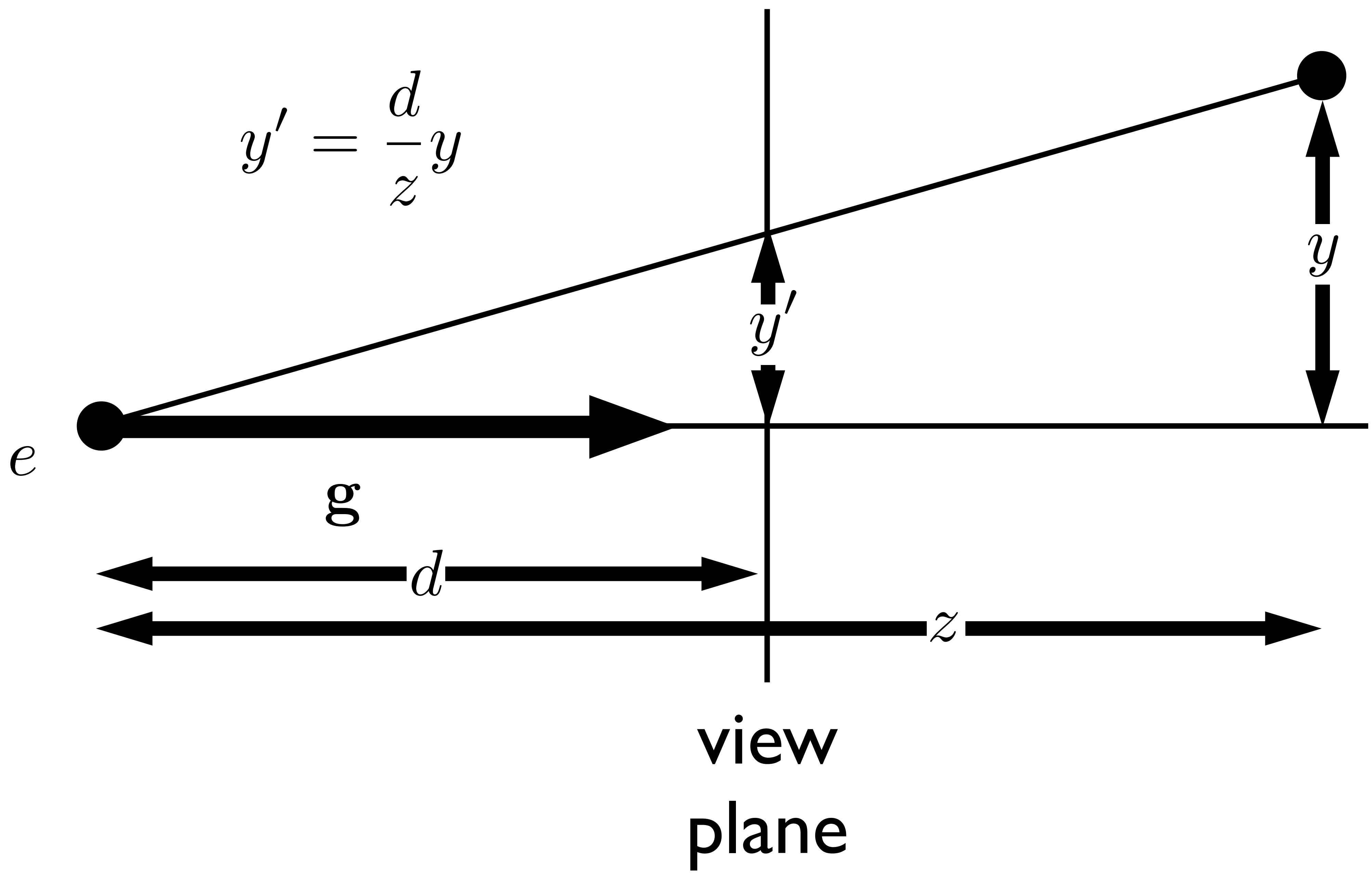


rigid – translation and rotation only – parallel lines and angles are preserved

affine – scaling, shear, translation, rotation – parallel lines preserved, angles **not** preserved

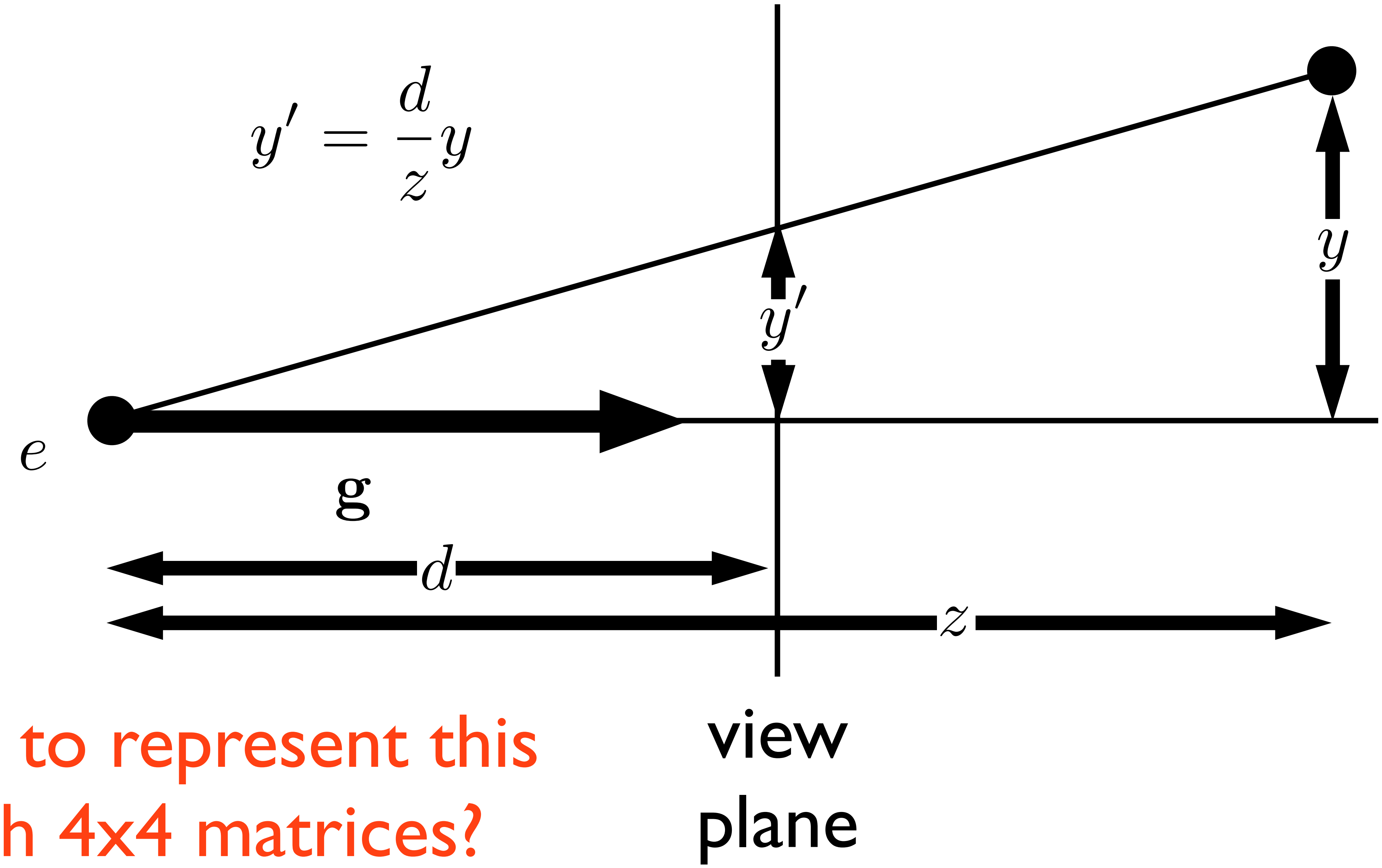
projective – parallel lines and angles **not** preserved

Projective Transformations



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

Projective Transformations



How to represent this with 4x4 matrices?

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

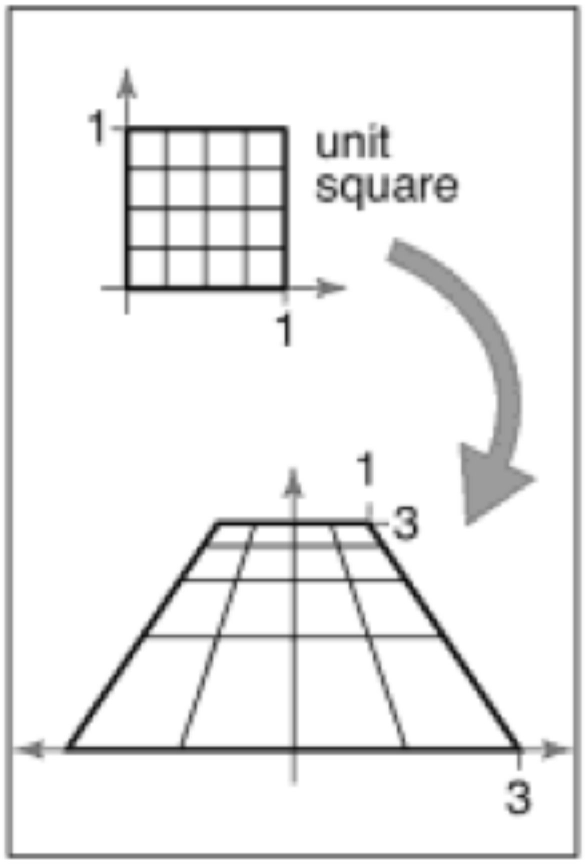
Projective Transformations

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

Use 4th coordinate as the denominator

Example:

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



[Shirley, Marschner]

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

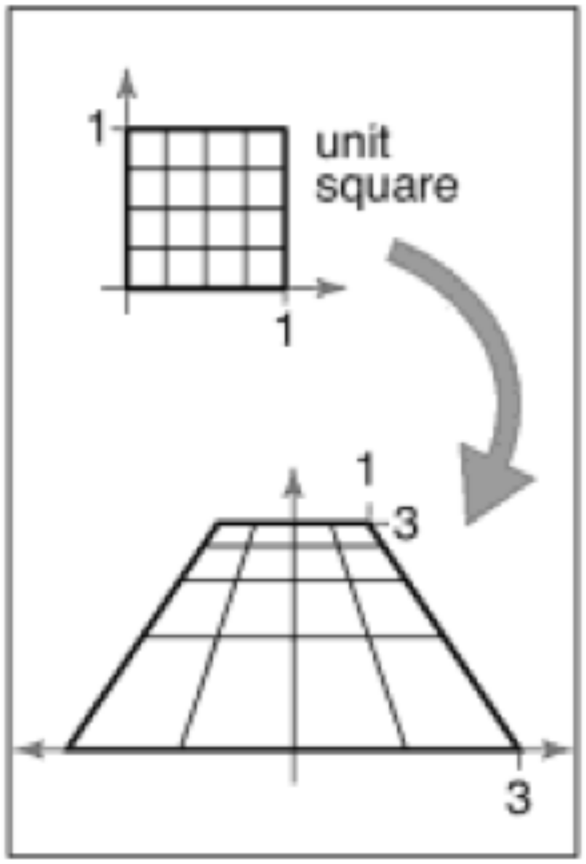
Projective Transformations

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

We can now implement perspective projection!

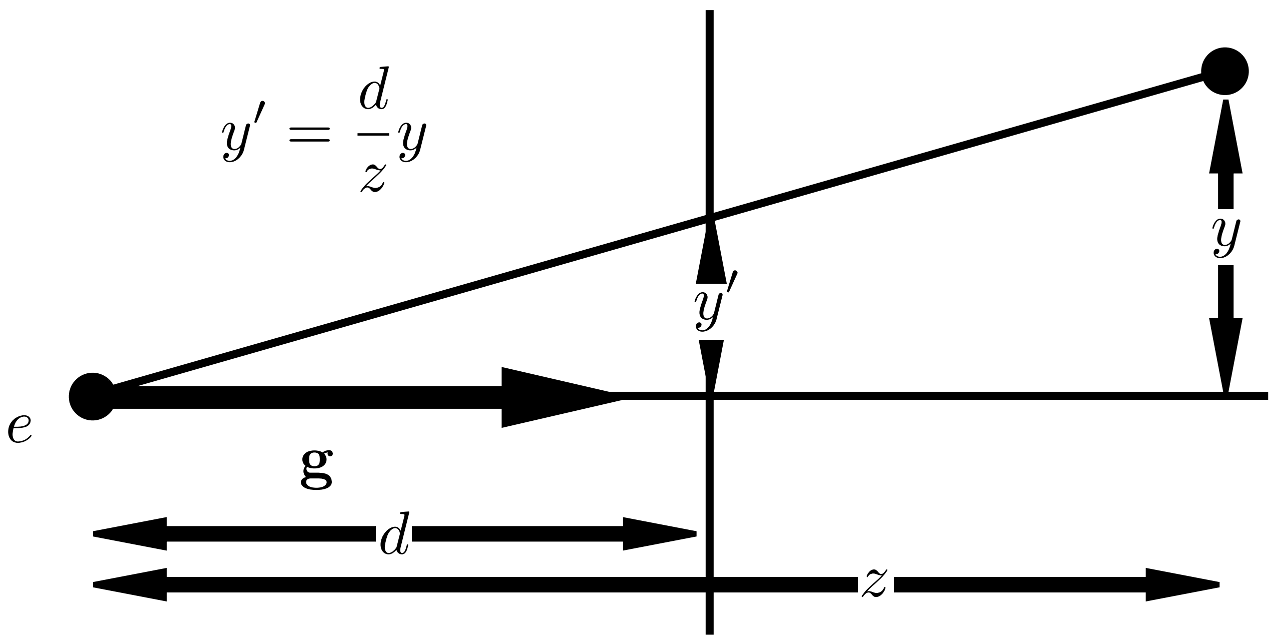
<u>Example:</u>

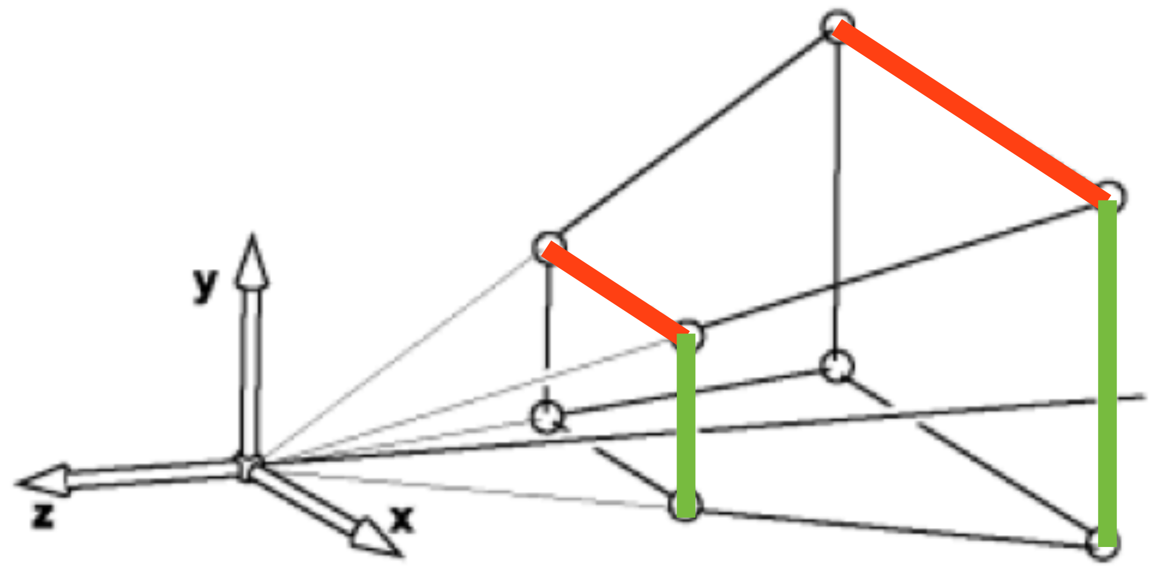
$$M = \begin{pmatrix} 2 & 0 & -1 \\ 0 & 3 & 0 \\ 0 & \frac{2}{3} & \frac{1}{3} \end{pmatrix}$$



[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

$\boxed{\text{but we've lost all information about z!}}$

<whiteboard>

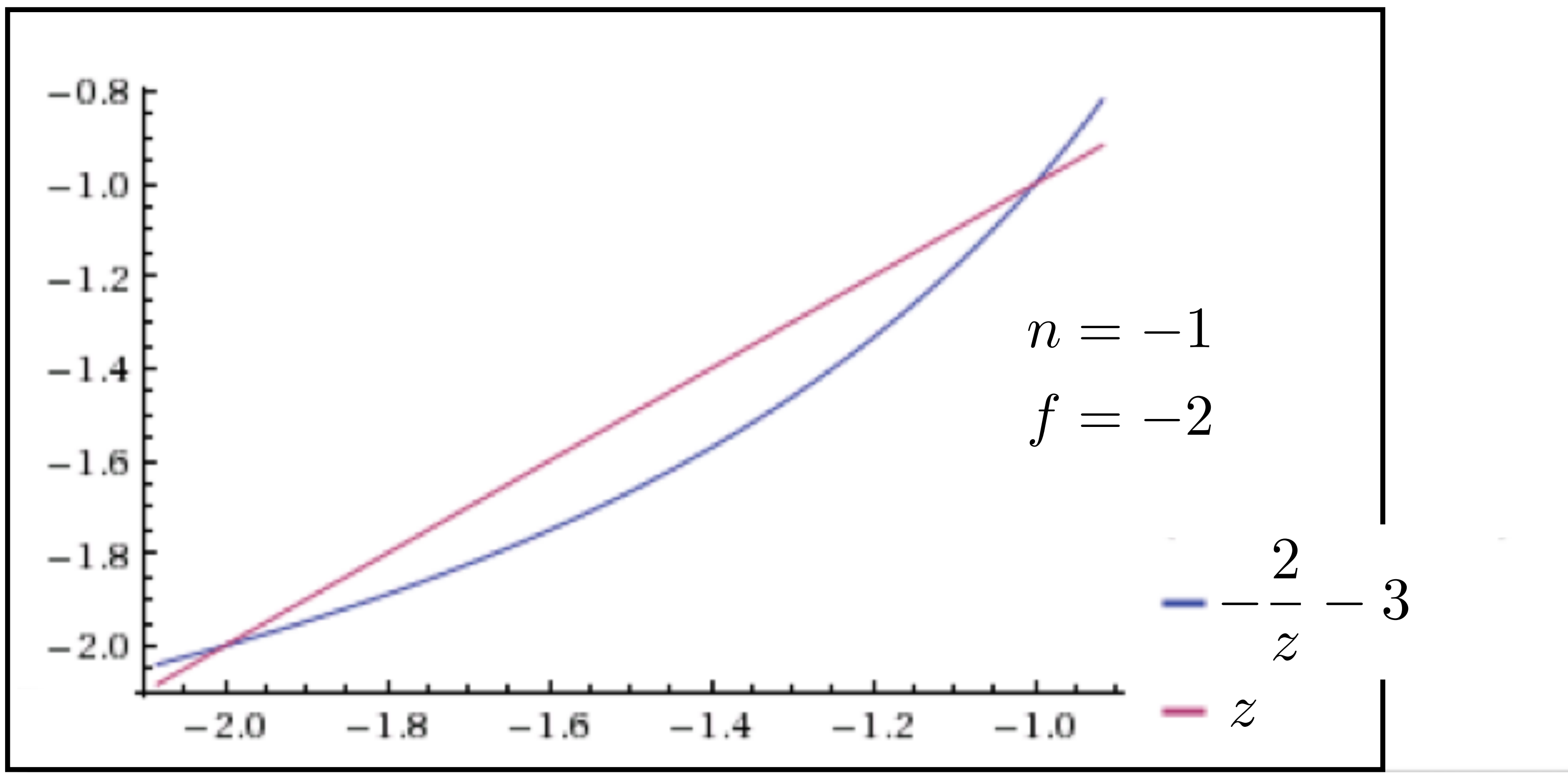
This simple projection matrix won't suffice. We need to preserve z information for later hidden surface removal.

whiteboard: derive P

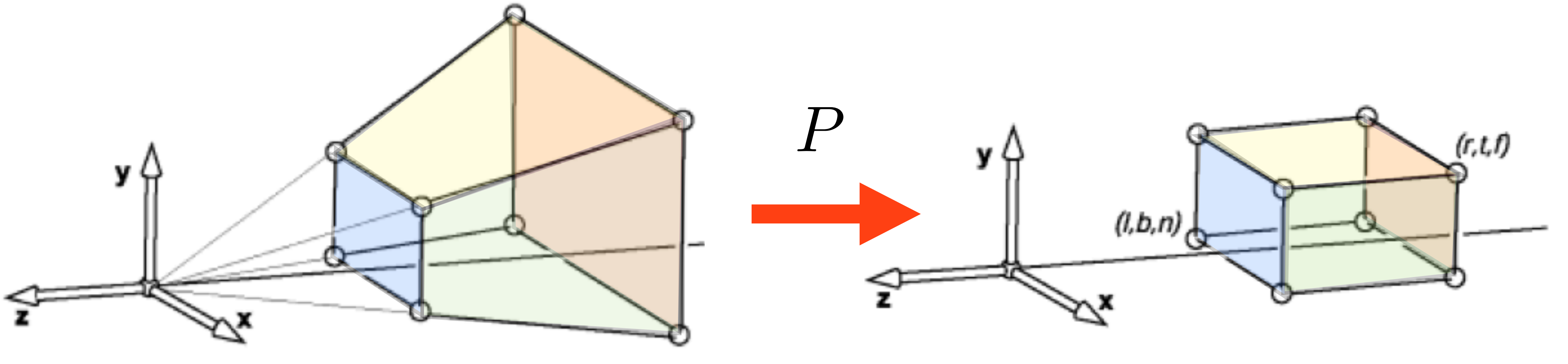
Perspective Projection

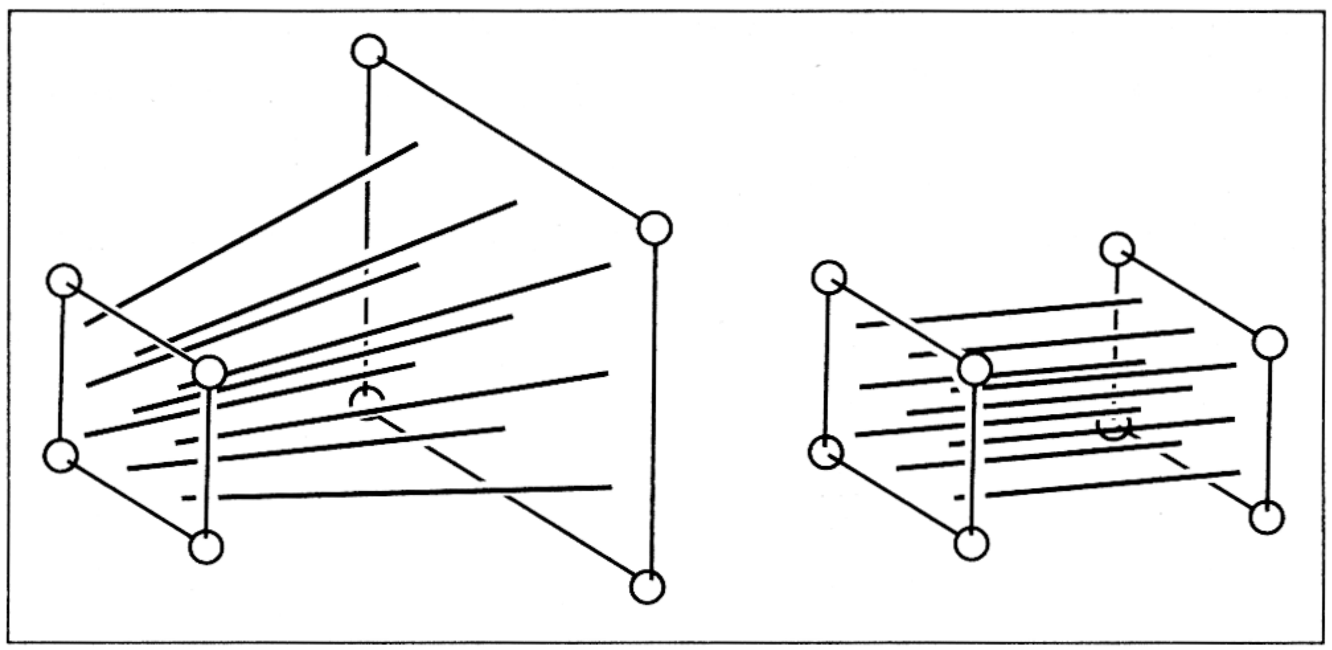
$$P = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad z' = (n+f) - \frac{nf}{z}$$

Example:



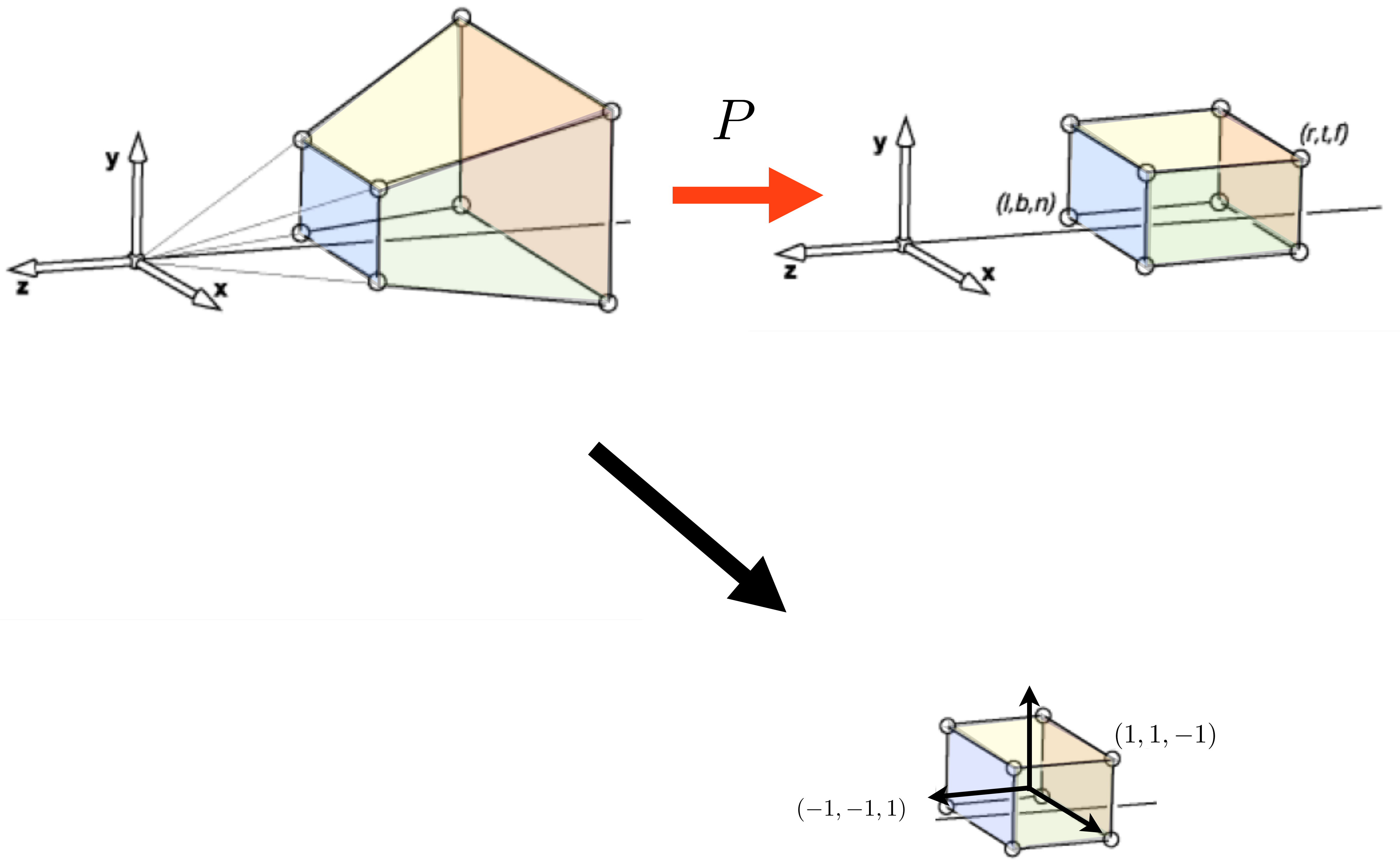
The perspective transformation does not preserve **z** completely, but it preserves **z = n, f** and is **monotone** (preserves ordering) with respect to z



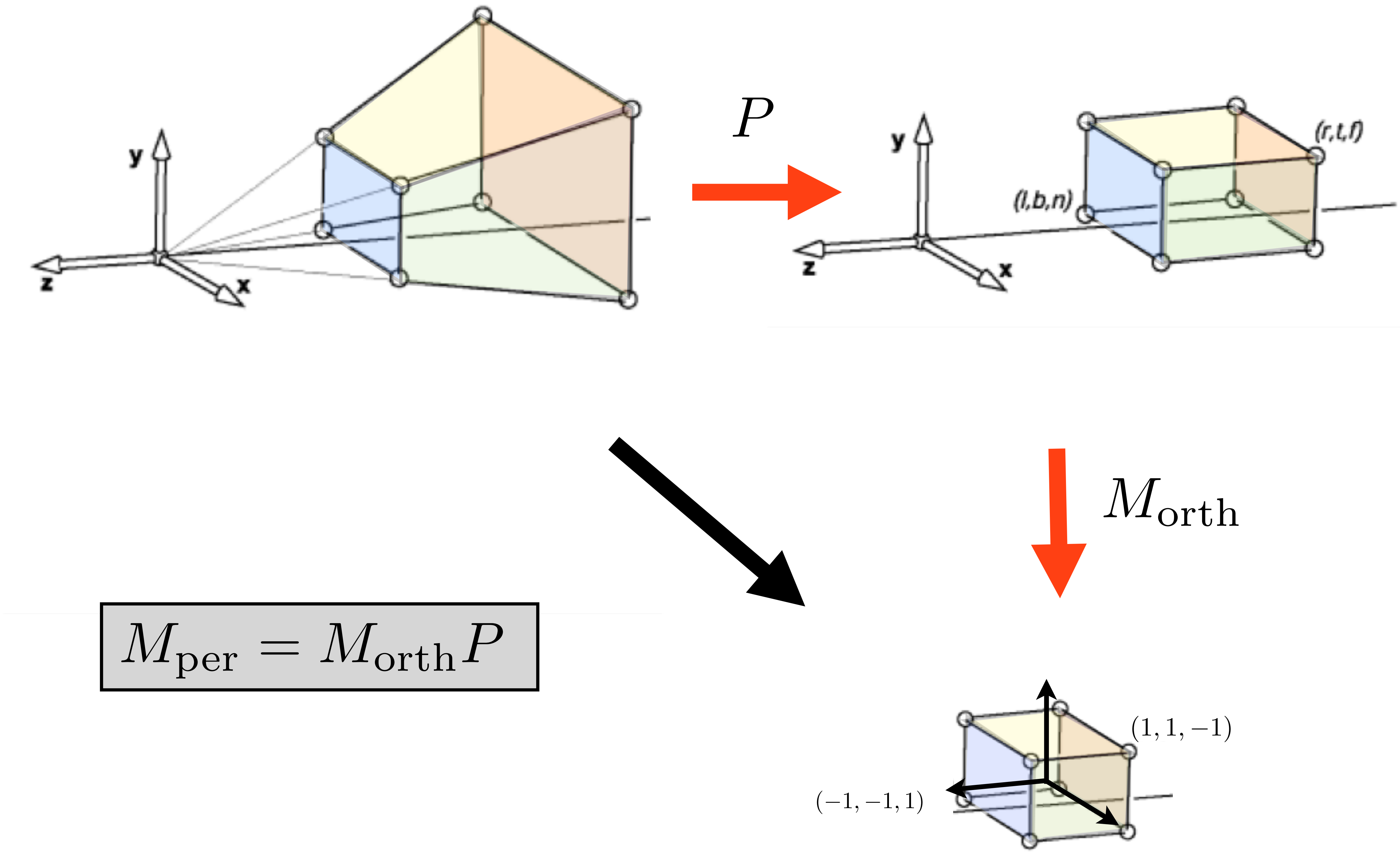


[Shirley, Marschner]

So far we've mapped the view frustum to a rectangular box. This rectangular box has the same near face as the view frustum. 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 frustum get mapped to the rect. box.



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



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.