Computer Graphics

Lecture 20

Projective Transformations

Perspective Viewing

Announcements

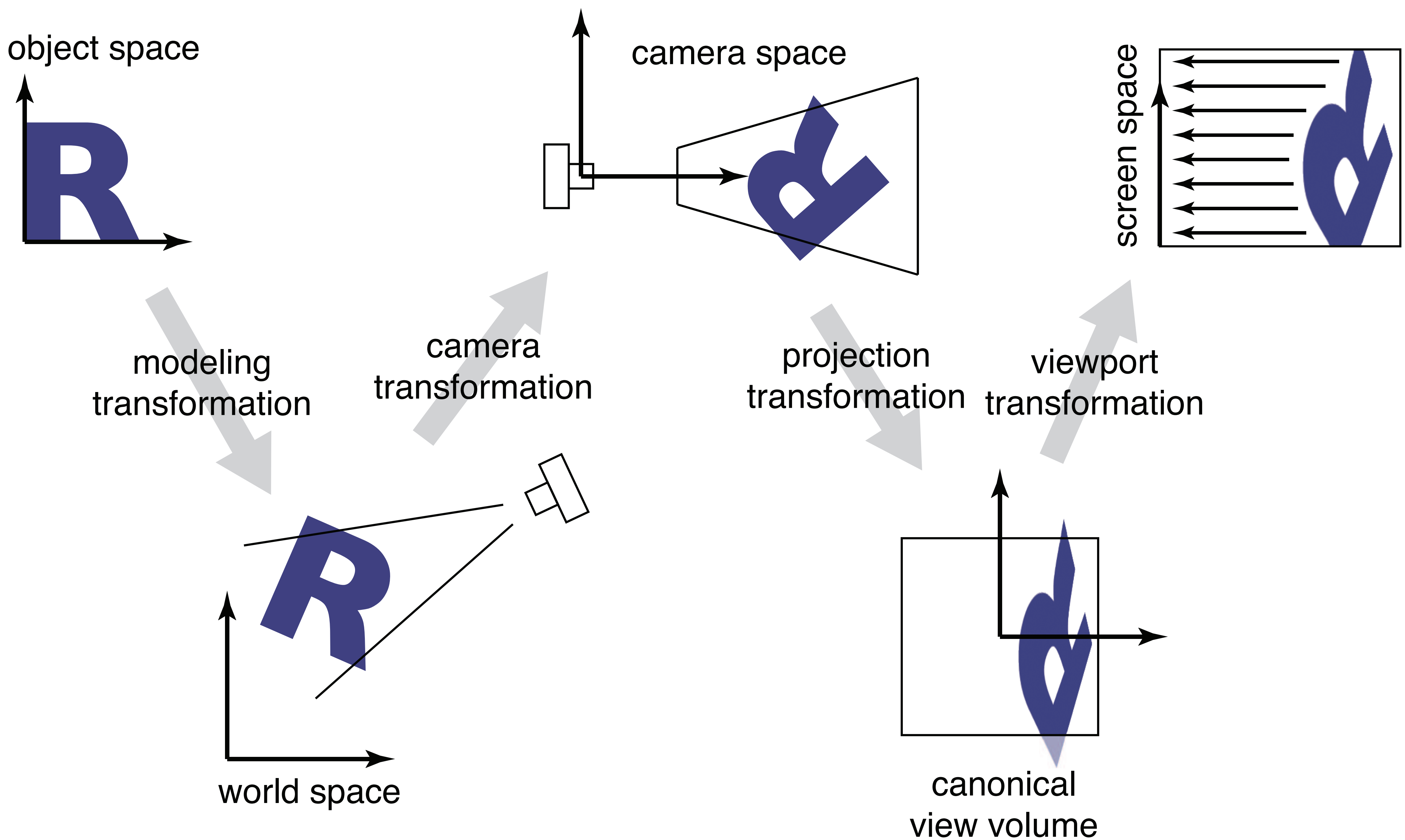
- Final project details and logistics available.
  - Assignments on Canvas
  - Writeup on the course webpage
- Near term:
  - Form groups by 1 week from now, with a (possibly vague) topic idea
  - Submit proposal a week from Friday
- HW3 due Monday
- A3 out today; done individually; due Nov 9th; shorter than A1/2

Goals

- Know how to interpret homogeneous coordinates when the fourth coordinate $w \neq 1$
- Know how to derive the perspective projection matrix.
- Know how the perspective projection matrix fits into the larger object-order transformation pipeline.

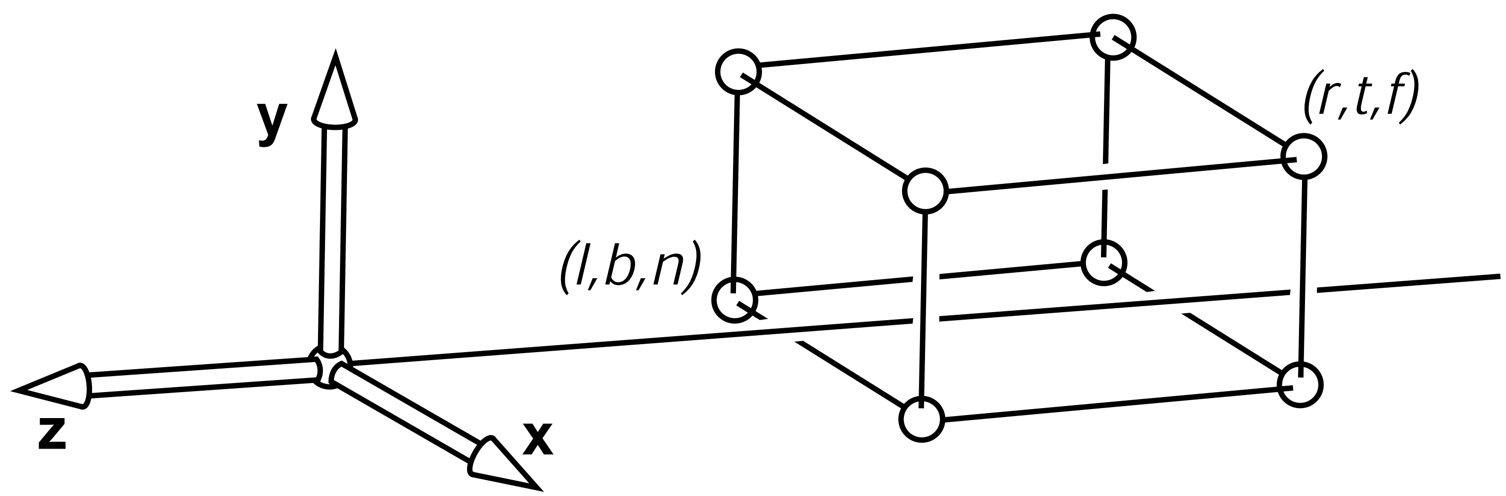
Viewing Transformations: Overview

A standard sequence of transforms to go from **object (model) space** to **screen (image) space**



Last time: Orthographic Camera

- Rays were already parallel to the z axis, so we only had to fiddle with scales.

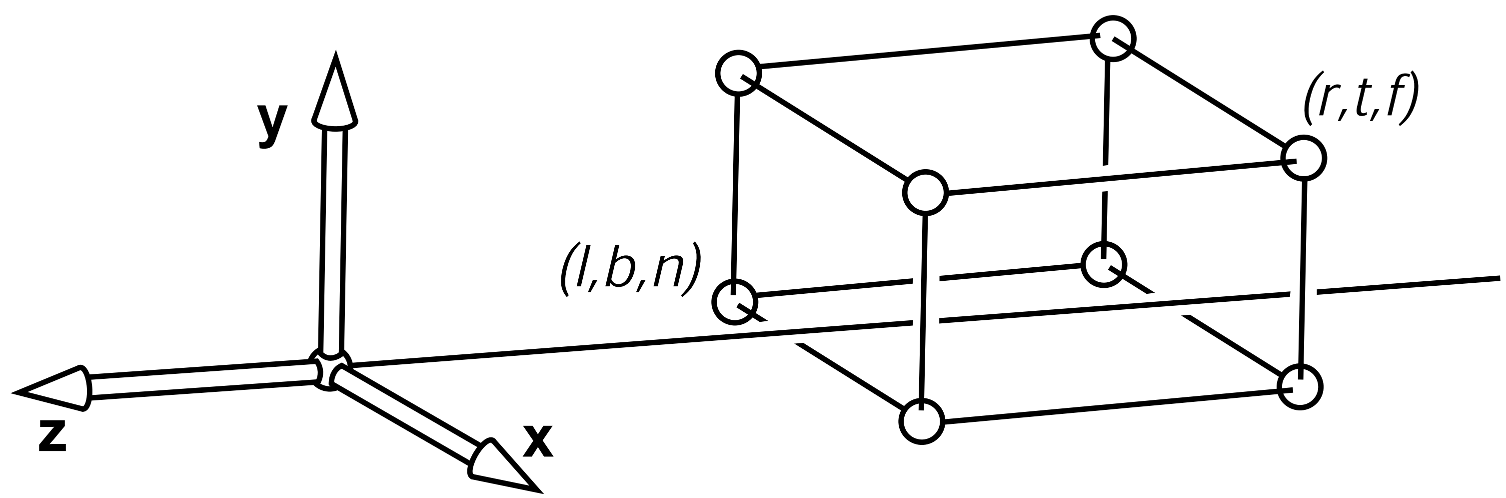


- Introduced near and far *clipping planes*
  - Excuse: throw away stuff behind the camera and too far away
  - Real reason: limit the range of possible depths
    (we'll need this later)

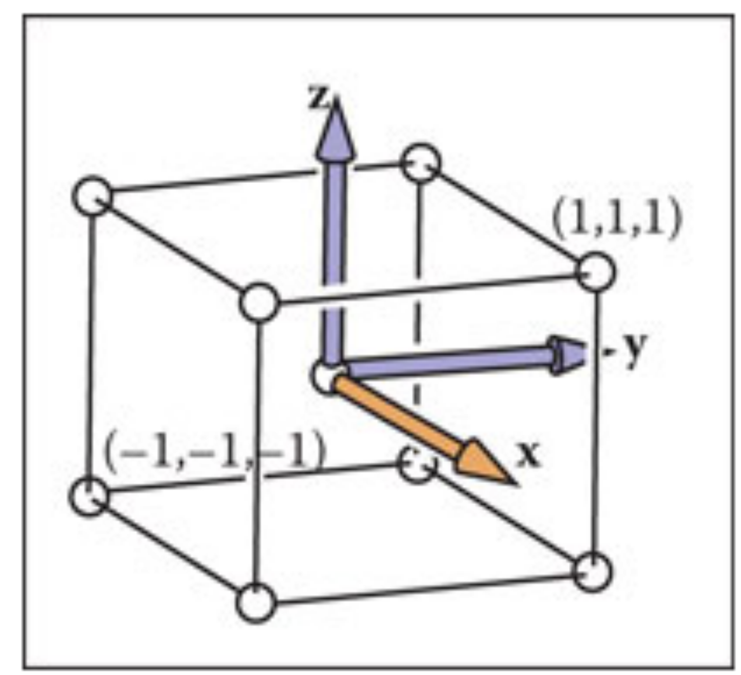
Orthographic Projection

- The result of our hard work:

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



$\rightarrow$ $\mathbf{M}_{\text{orth}}$

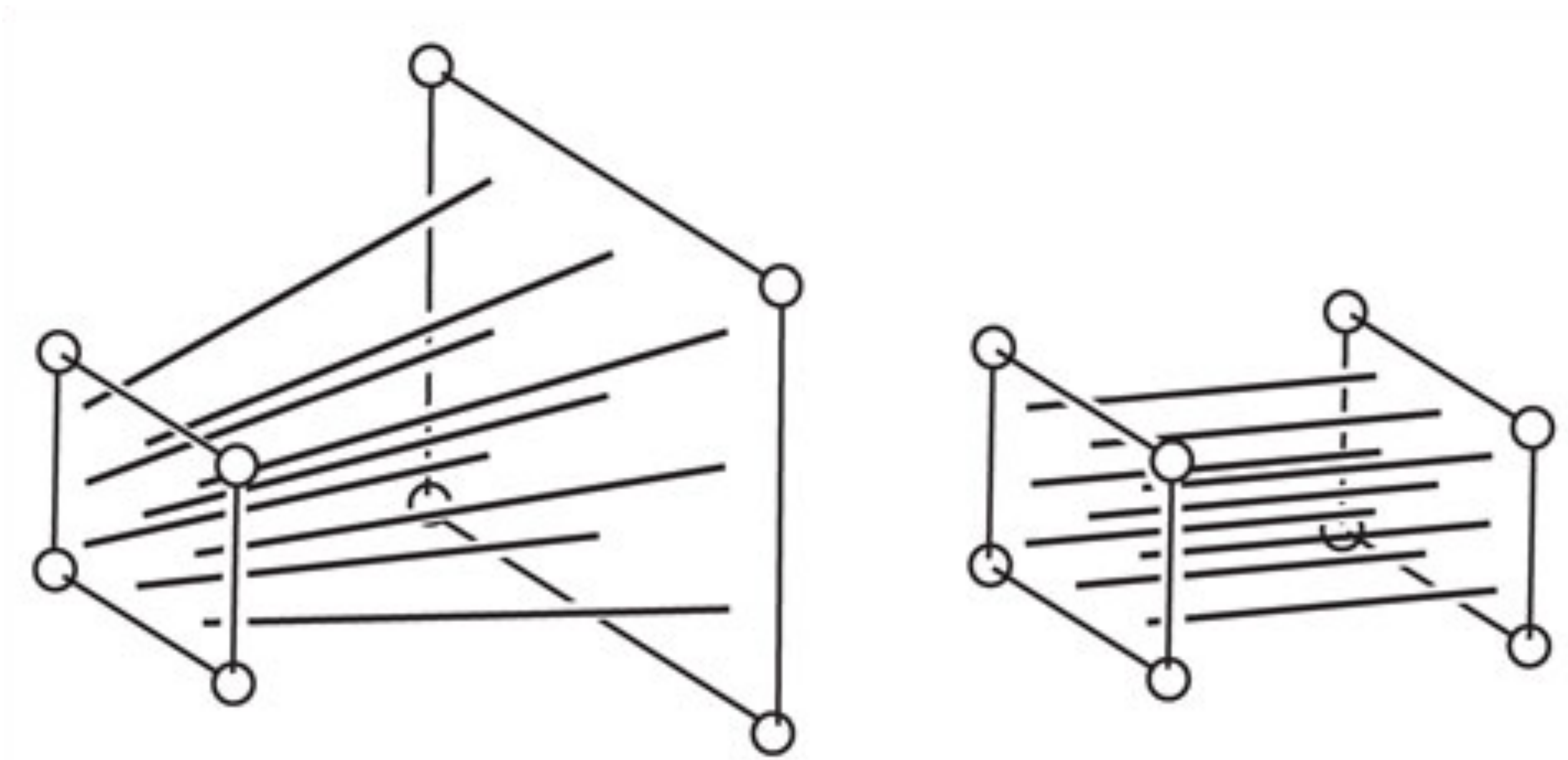


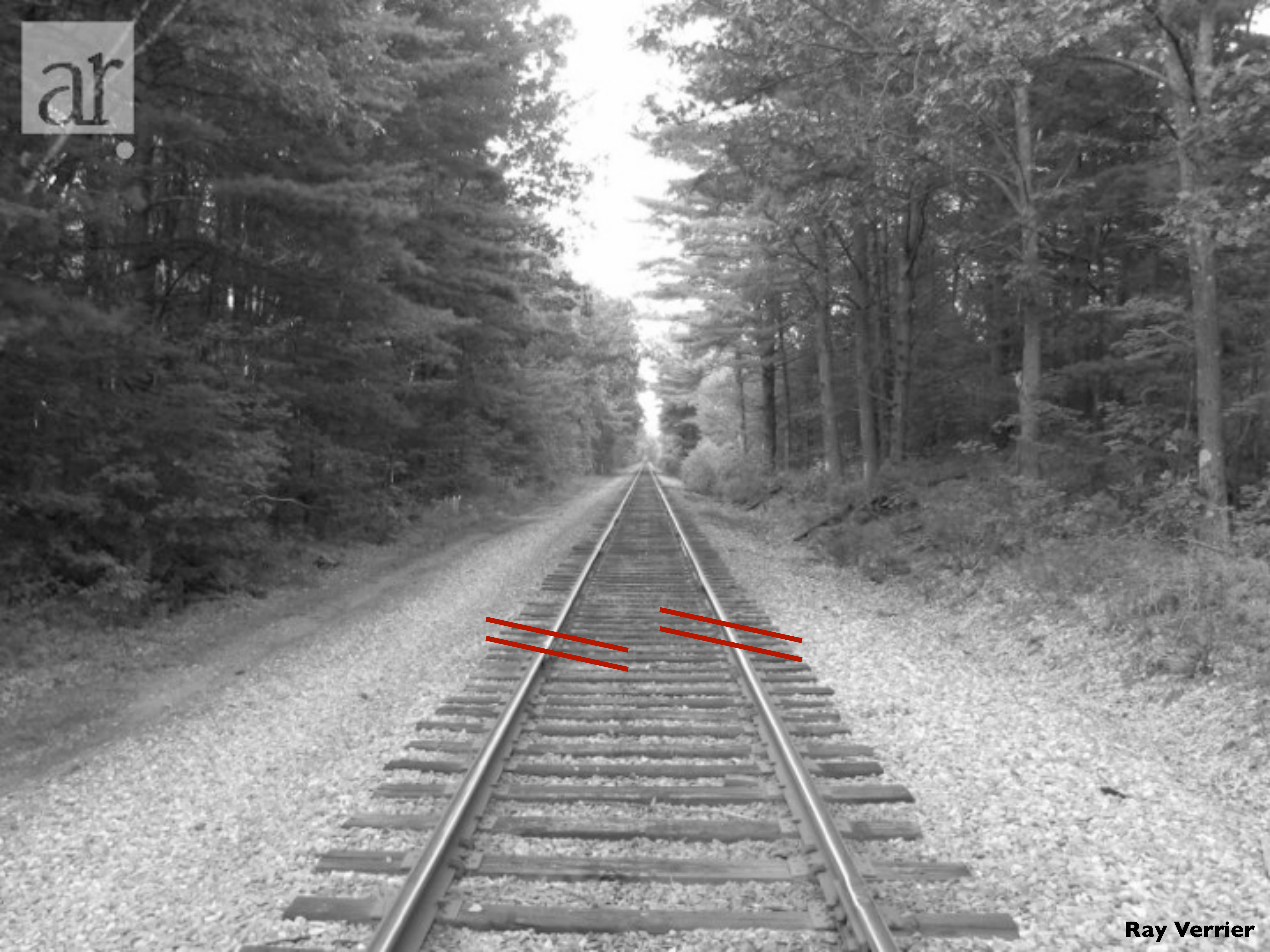
Camera Coordinates

Normalized Device Coordinates

Perspective Projection

- In a perspective camera, we have to warp space in a more dramatic way.




Ray Verrier

Perspective Projection

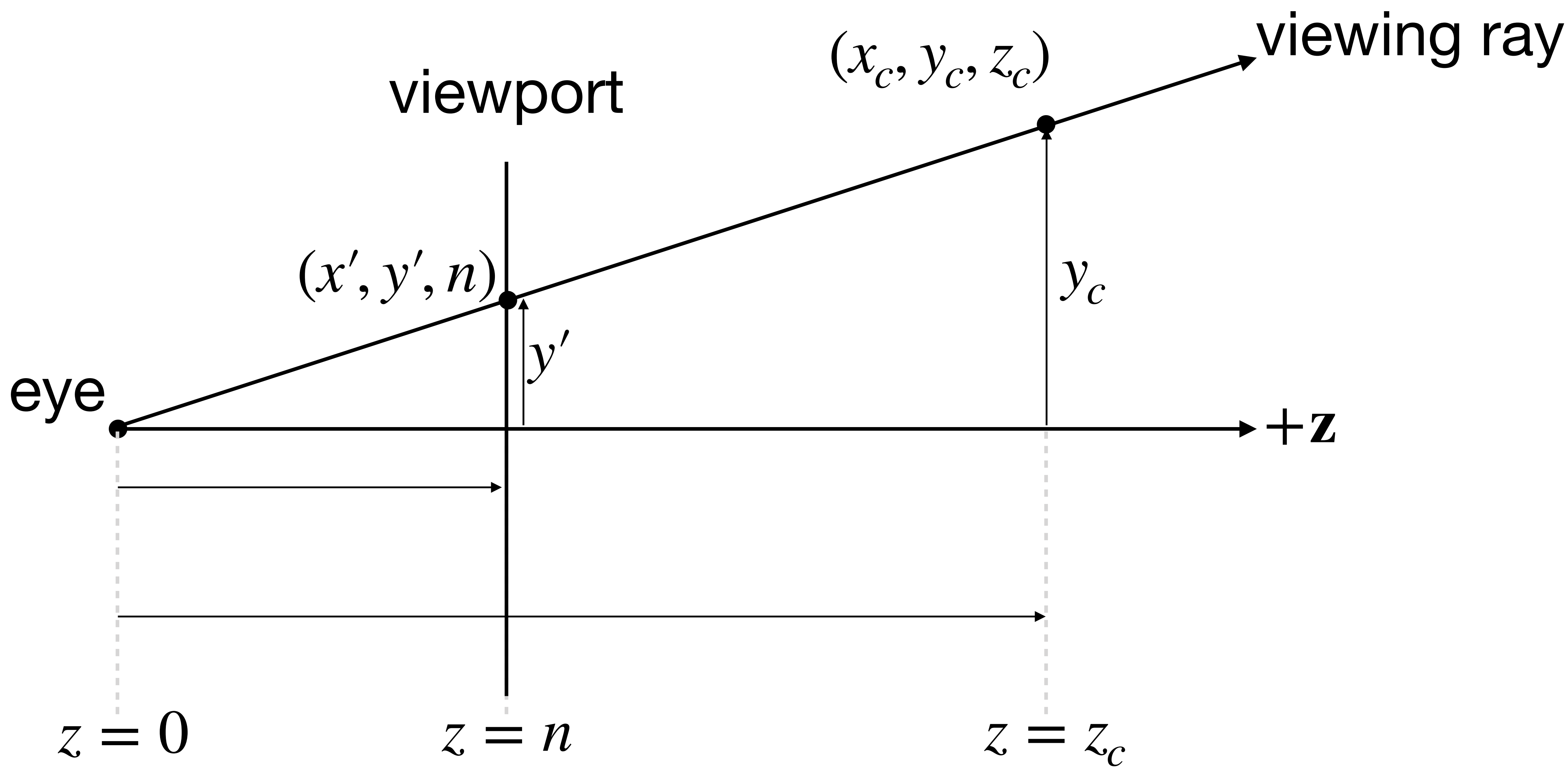
- In a perspective camera, we have to warp space in a more dramatic way.
- Demo: https://www.cs.cornell.edu/courses/cs4620/2019fa/demos/view_explore/view_explore.html
- Recall: linear and affine transformations preserve parallelism.

  We don't have the tools for the job!

Perspective Projection

Exercise:

Find y', the y coordinate of the point where (x_c, y_c, z_c) projects onto the viewport.



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*

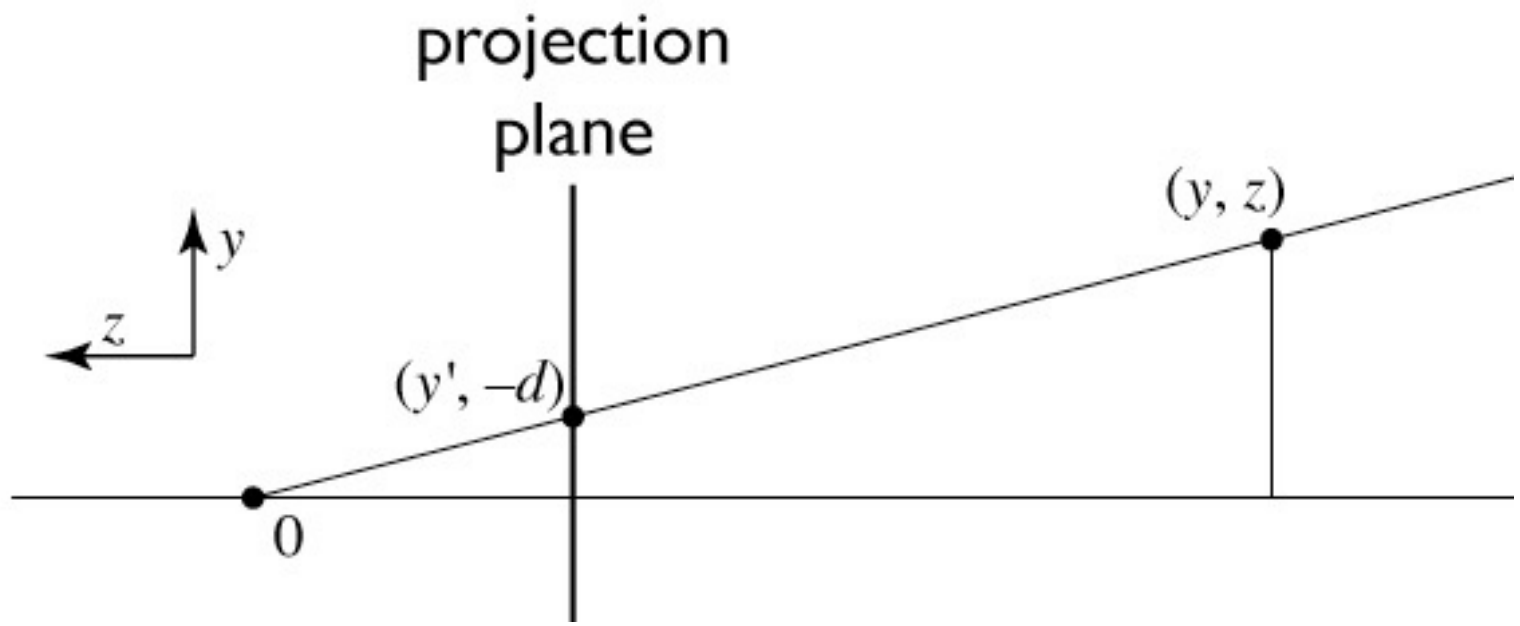
$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \sim \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix}$$

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:

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

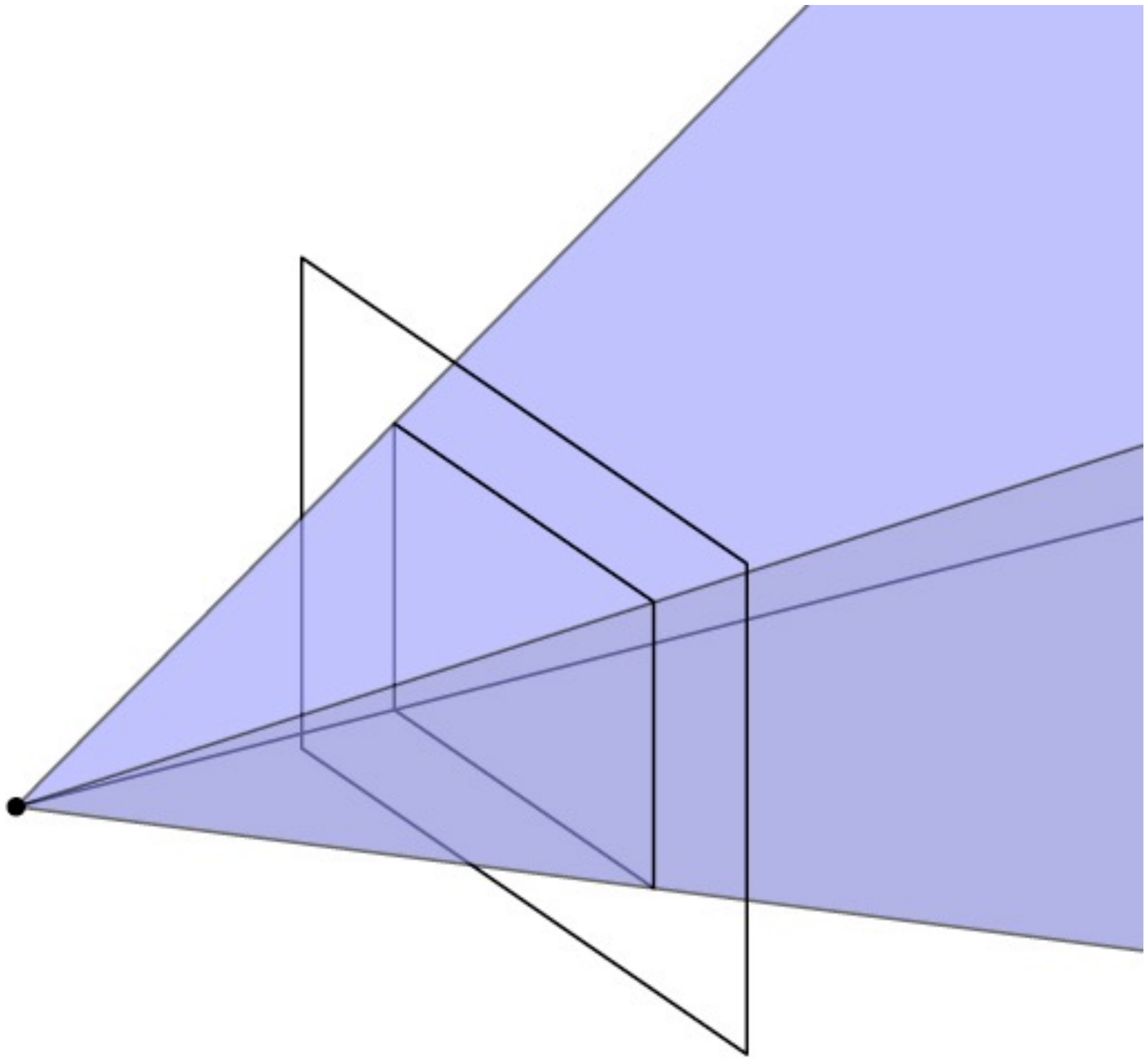
What can projective transformations do?

- Map a quadrilateral to another quadrilateral.
- https://iis.uibk.ac.at/public/piater/courses/demos/homography/homography.xhtml
  - This demo seems to be broken in Firefox, but works in Safari (did not test on Chrome)

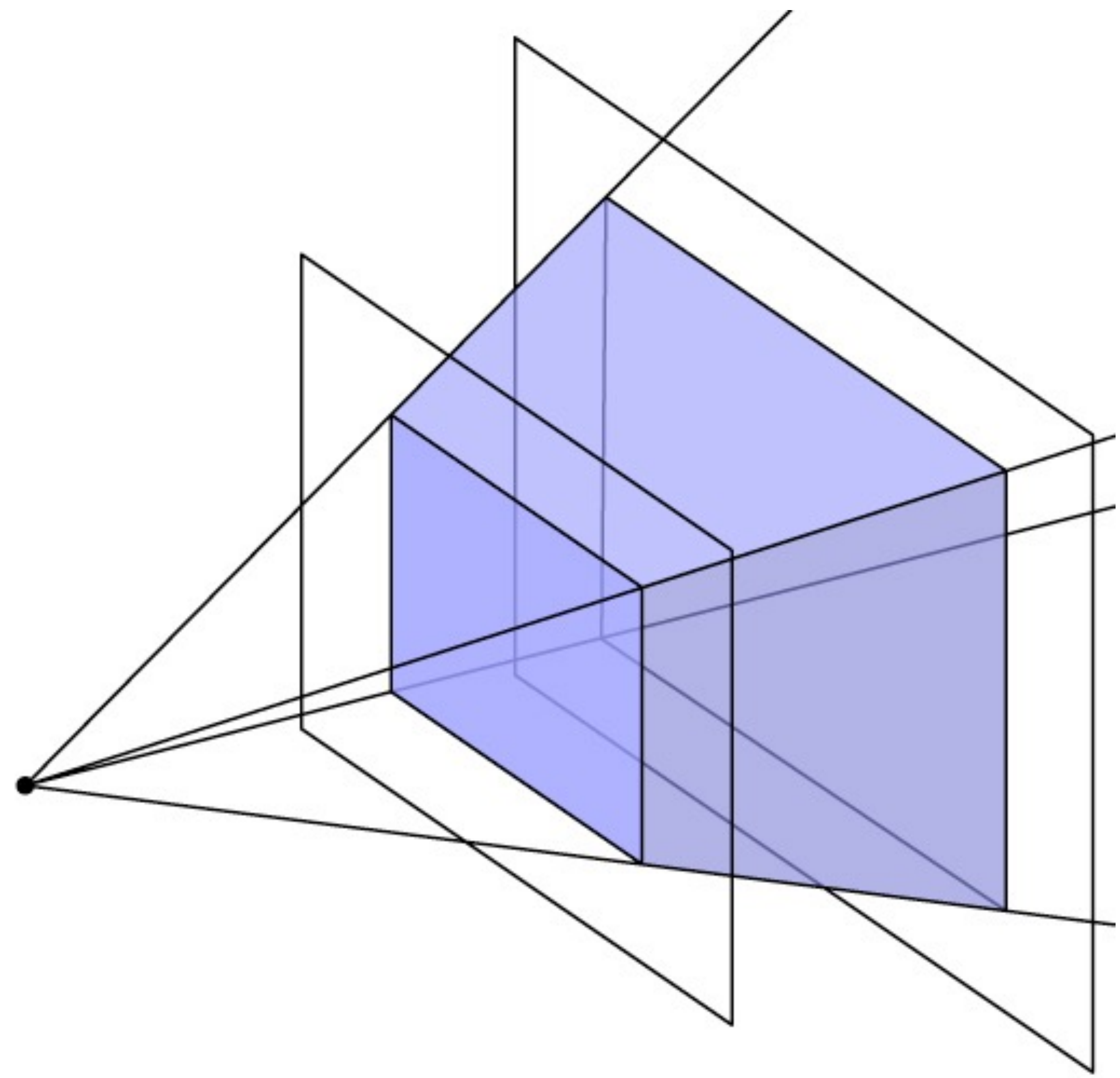
What can projective transformations do?

- Map a quadrilateral to another quadrilateral.
- https://iis.uibk.ac.at/public/piater/courses/demos/homography/homography.xhtml
- Aside: line segments still map to line segments, so we can still do wireframe rendering.

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.

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

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

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

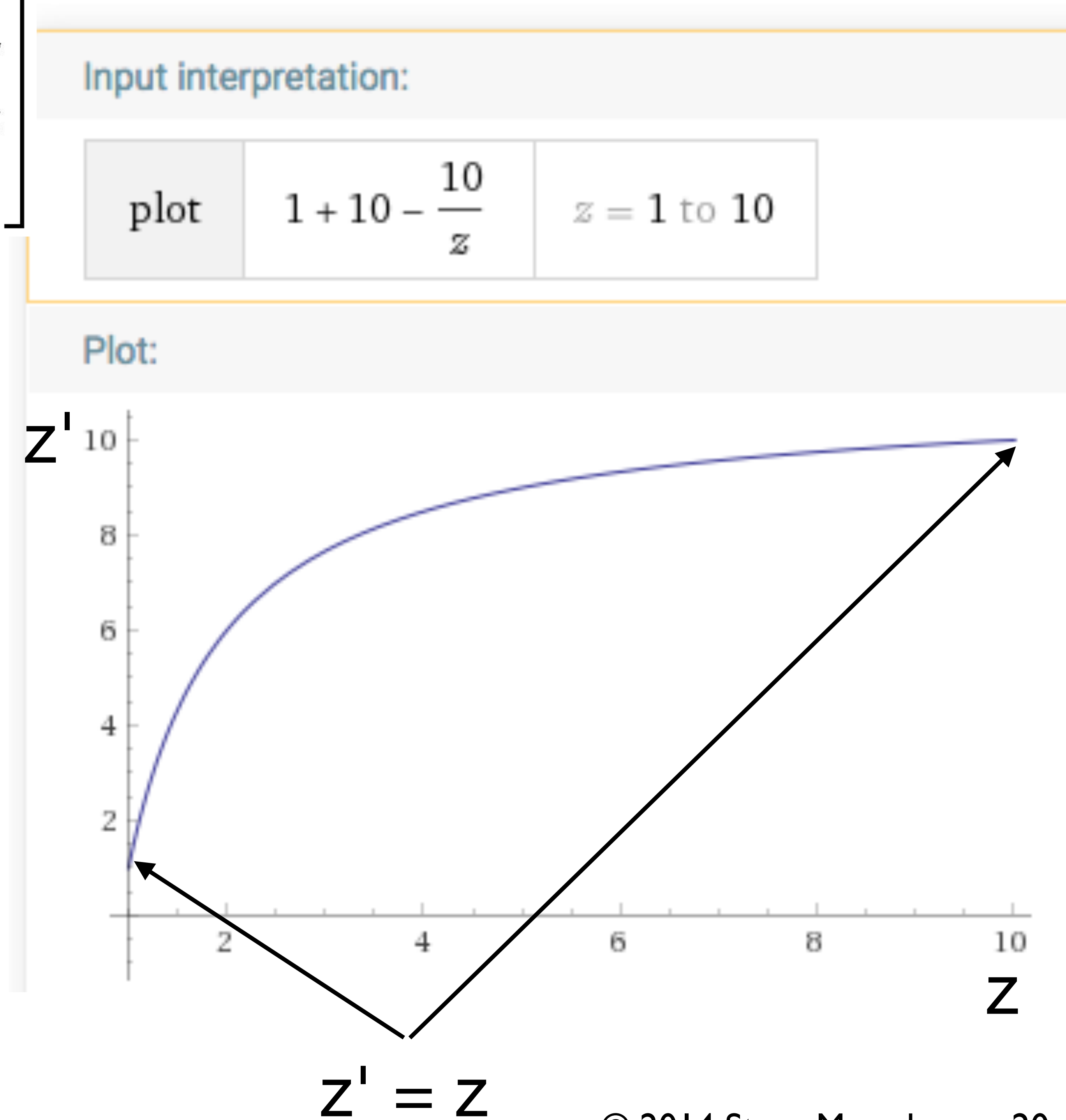
$$\text{result: } a = -(n + f) \text{ and } b = nf \text{ (try it)}$$

Carrying depth through perspective

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

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

Example:
n=1, f=10



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.

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

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

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

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

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}

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

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