Computer Graphics

Lecture #7

Viewing

Recall: Homogeneous Coordinates

- To put this into one system of linear equations, we increase the dimensionality, adding a component w = 1 for vectors

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & x_t \\ m_{21} & m_{22} & y_t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11}x + m_{12}y + x_t \\ m_{21}x + m_{22}y + y_t \\ 1 \end{bmatrix}$$

- Implements a linear transformation followed by a translation (x_t,y_t)

- These transformations are called **affine transformations**:

    - Like linear transformations, they keep straight lines straight and parallel lines parallel, but they do not preserve the origin

Recall: Coordinate Systems

- Points in space can be represented using an origin position and a set of orthogonal basis vectors:

Recall: Coordinate Systems

- Points in space can be represented using an origin position and a set of orthogonal basis vectors:

$$\mathbf{p} = \left(x_p, y_p\right) \equiv \mathbf{0} + x_p\mathbf{x} + y_p\mathbf{y}$$

Recall: Coordinate Systems

- Points in space can be represented using an origin position and a set of orthogonal basis vectors:

$$\mathbf{p} = (x_p, y_p) \equiv \mathbf{0} + x_p \mathbf{x} + y_p \mathbf{y} \qquad \mathbf{p} = (u_p, v_p) \equiv \mathbf{e} + u_p \mathbf{u} + v_p \mathbf{v}$$

Recall: Coordinate Systems

- Points in space can be represented using an origin position and a set of orthogonal basis vectors:

$$\mathbf{p} = (x_p, y_p) \equiv \mathbf{0} + x_p \mathbf{x} + y_p \mathbf{y} \qquad \mathbf{p} = (u_p, v_p) \equiv \mathbf{e} + u_p \mathbf{u} + v_p \mathbf{v}$$

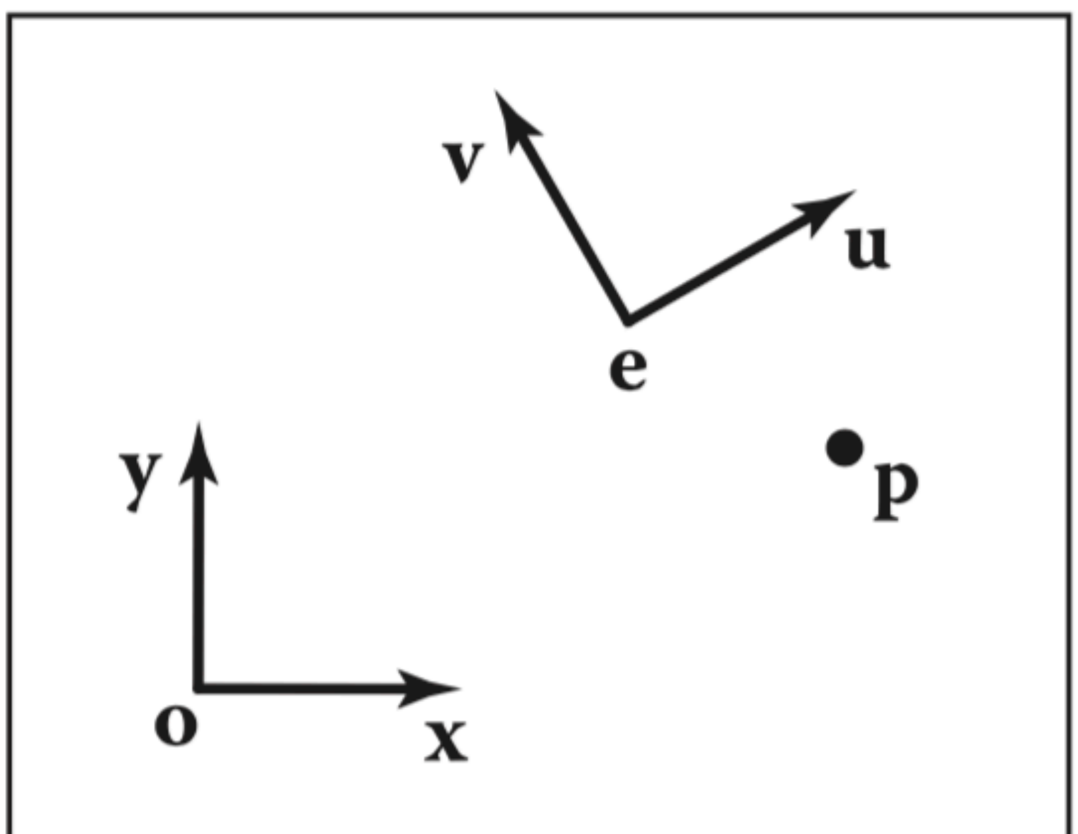
- Any point can be described in either coordinate system

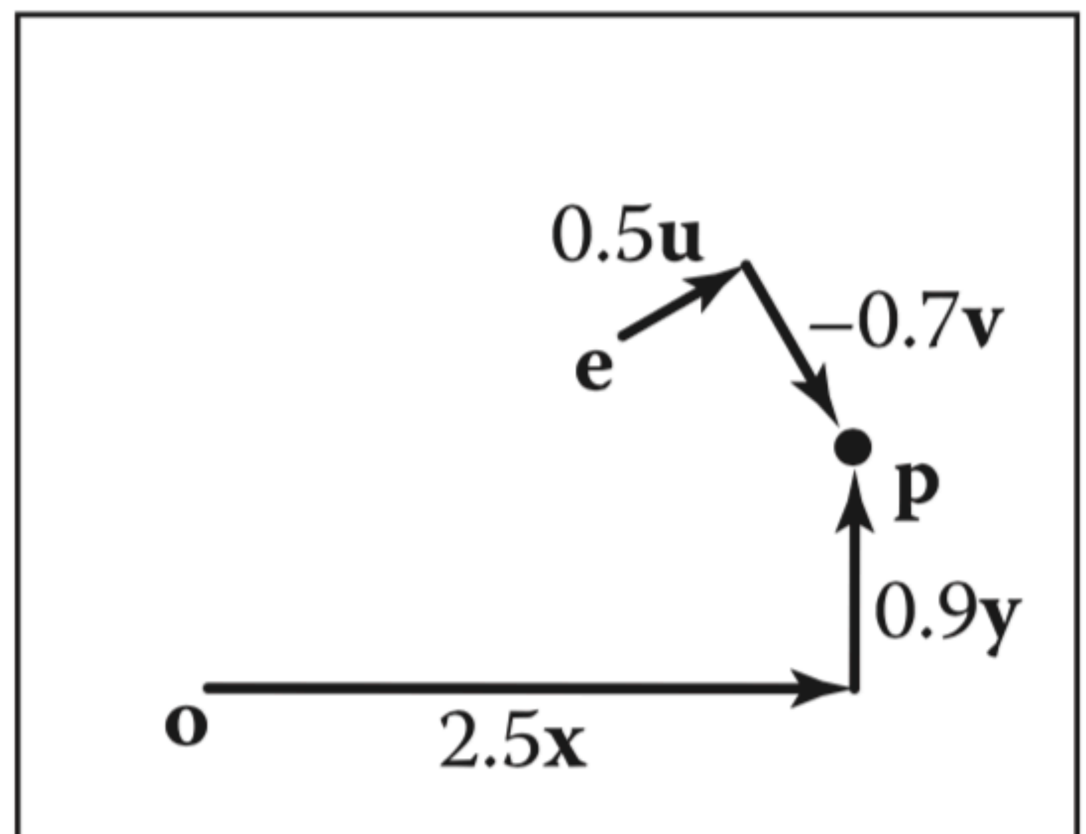
Recall: Coordinate Systems

- Points in space can be represented using an origin position and a set of orthogonal basis vectors:

$$\mathbf{p} = (x_p, y_p) \equiv \mathbf{0} + x_p\mathbf{x} + y_p\mathbf{y} \qquad \mathbf{p} = (u_p, v_p) \equiv \mathbf{e} + u_p\mathbf{u} + v_p\mathbf{v}$$

- Any point can be described in either coordinate system





Recall: Matrices for Converting Coordinate Systems

- Using homogenous coordinates and affine transformations, we can convert between coordinate systems:

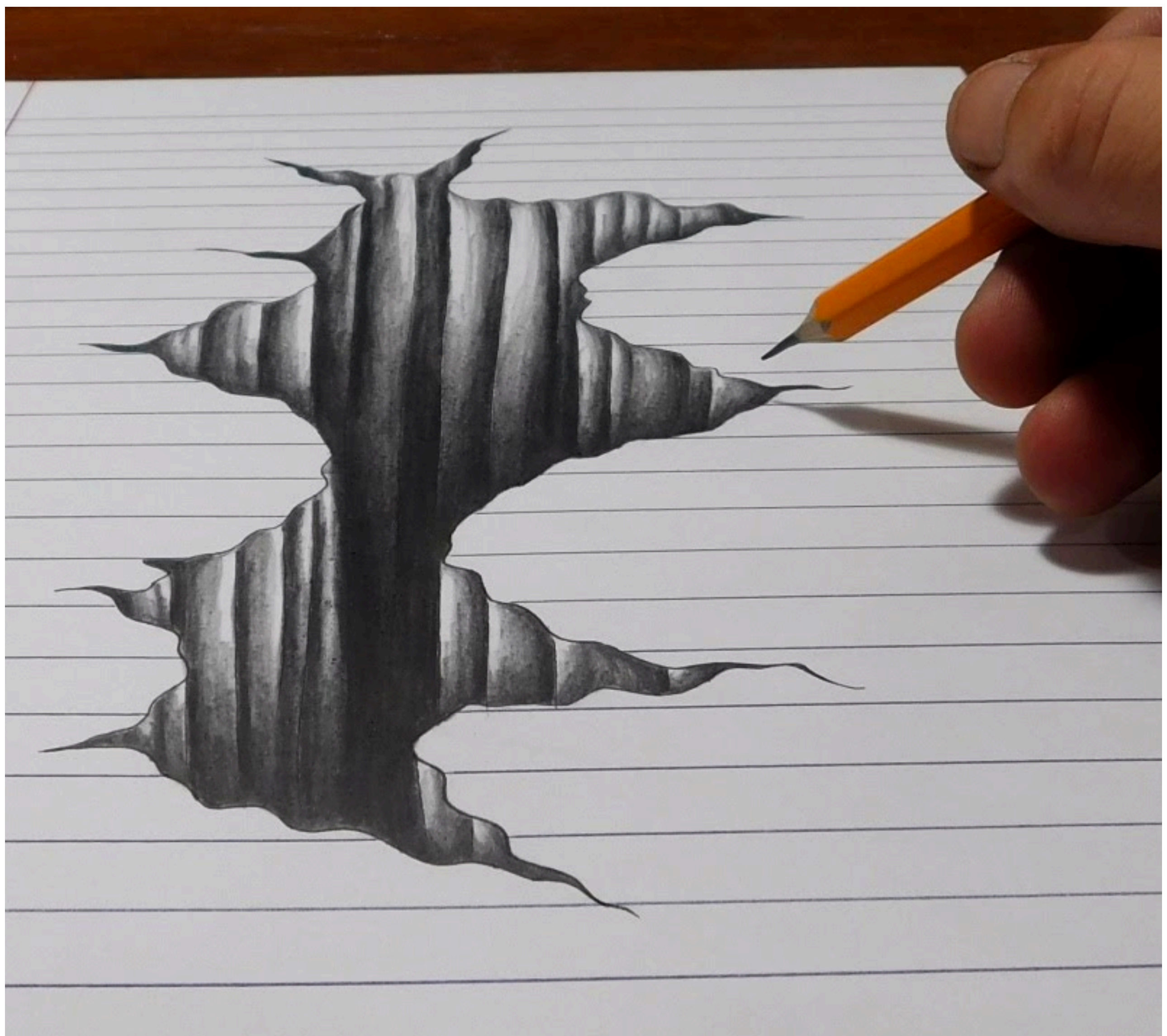
$$\begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_e \\ 0 & 1 & y_e \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_u & x_v & 0 \\ y_u & y_v & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_p \\ v_p \\ 1 \end{bmatrix} = \begin{bmatrix} x_u & x_v & x_e \\ y_u & y_v & y_e \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_p \\ v_p \\ 1 \end{bmatrix}$$

- More generally, any arbitrary coordinate system transform:
- $$\mathbf{P}_{uv} = \begin{bmatrix} \mathbf{x}_{uv} & \mathbf{y}_{uv} & \mathbf{o}_{uv} \\ 0 & 0 & 1 \end{bmatrix} \mathbf{p}_{xy}$$

Viewing

Recall: Two Ways to Think About How We Make Images

- Drawing
- Photography



Recall: Two Ways to Think About Rendering

- Object-Ordered
- Decide, for every object in the scene, its contribution to the image

- Image-Ordered
- Decide, for every pixel in the image, its contribution from every object

Recall: Two Ways to Think About Rendering

- Object-Ordered

TODAY

- Decide, for every object in the scene, its contribution to the image

- Image-Ordered
- Decide, for every pixel in the image, its contribution from every object

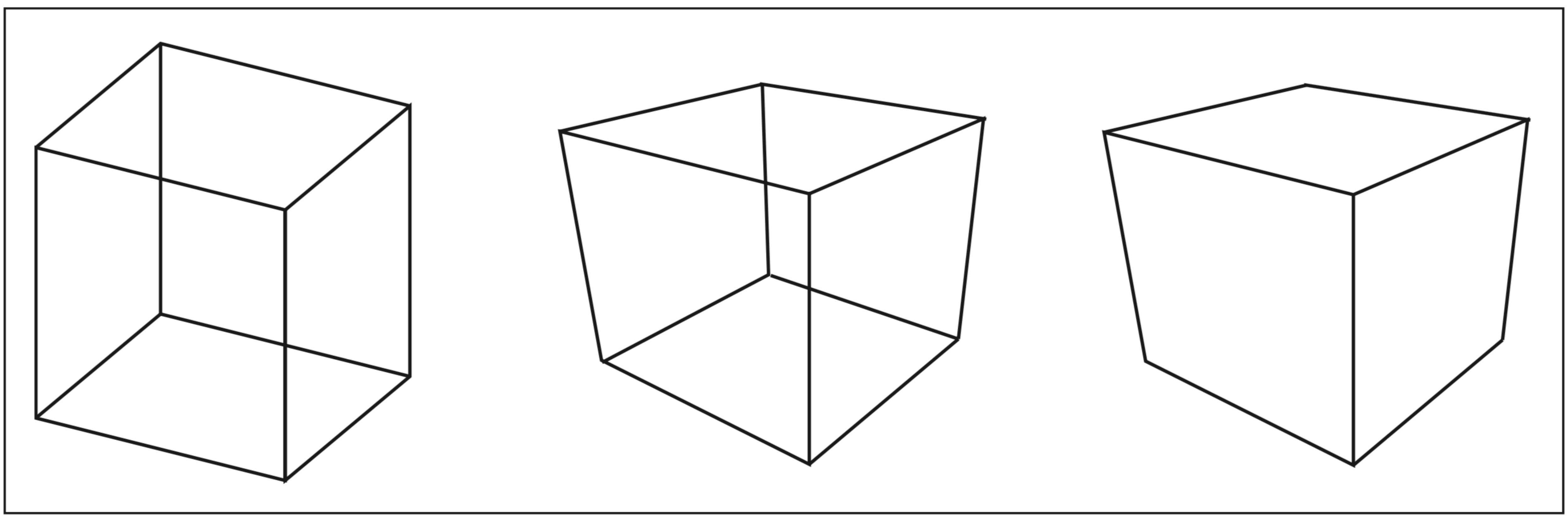
View Transformations

Using Transformations for Rendering

- Idea for today: Matrices can be used to move objects from 3D spaces to the 2D space of an image
  - Broadly, this reduction of dimensions is called **viewing transformation**
  - We will compose multiple matrix-based transformations to rethink cameras

Drawing by Transformation

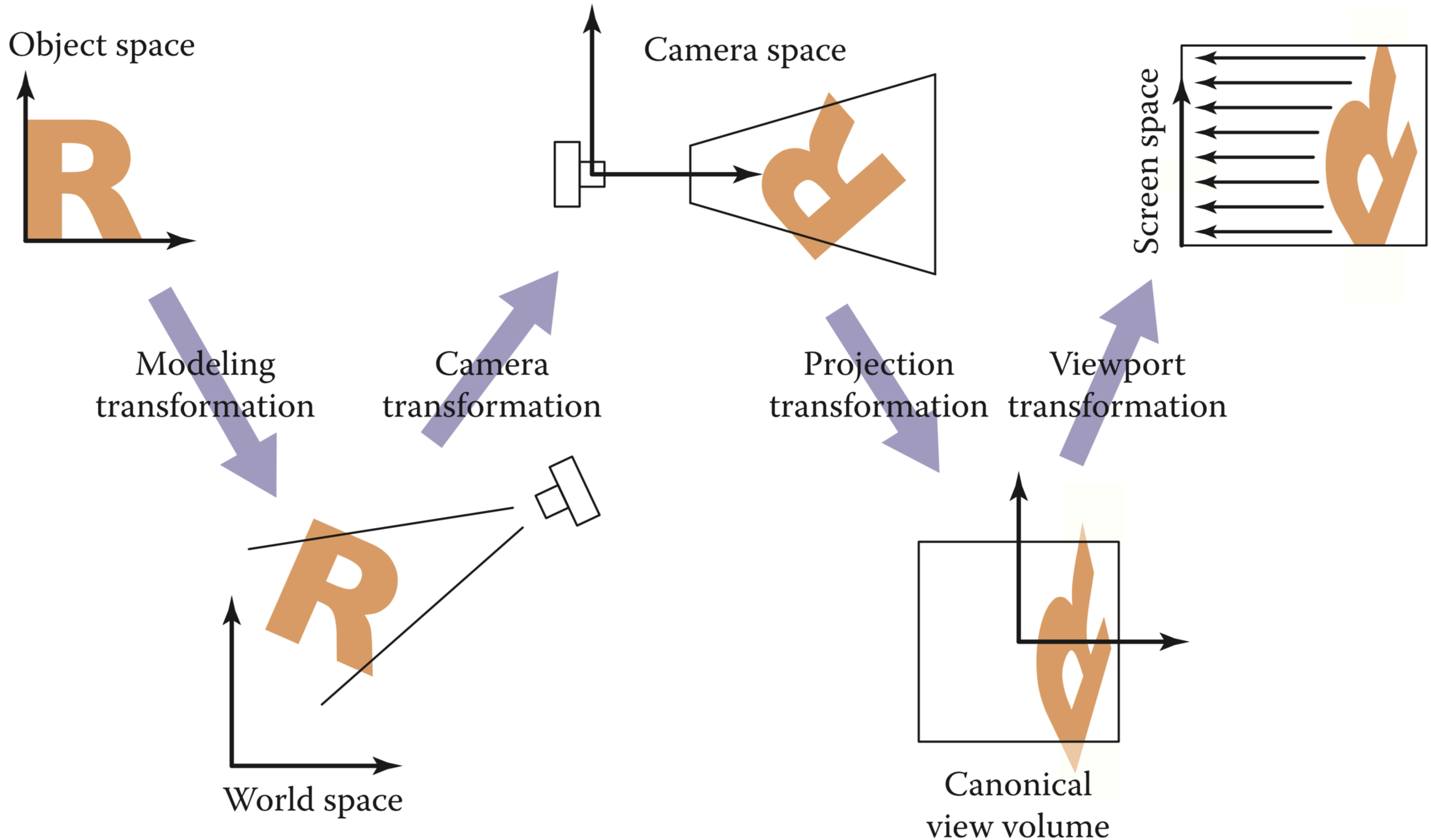
- For now, we will consider drawing wireframe objects (collections of 3D line segments)



Orthographic **Perspective** **Perspective + Hidden Line Removal**

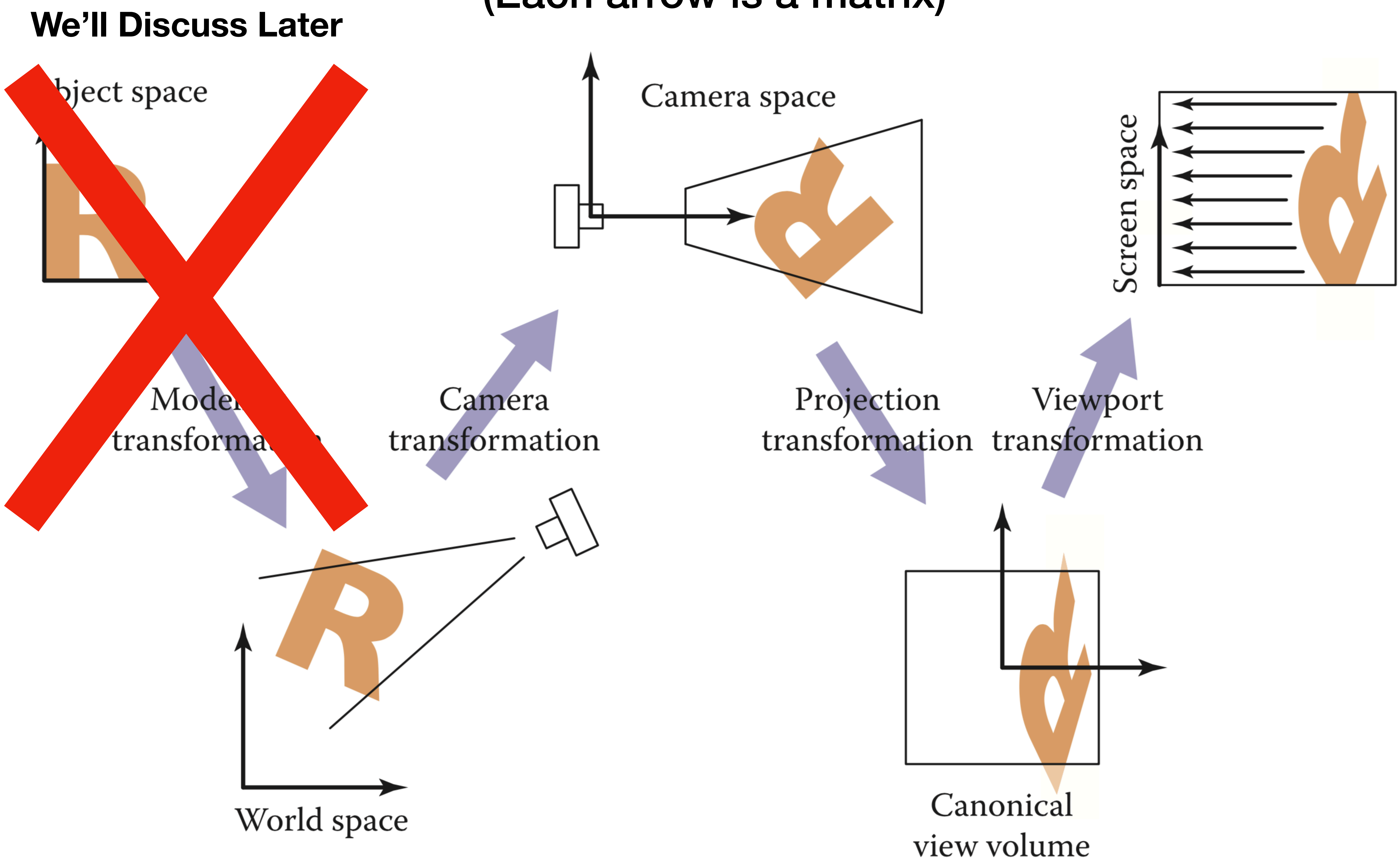
Step-by-Step Viewing Transformations

(Each arrow is a matrix)



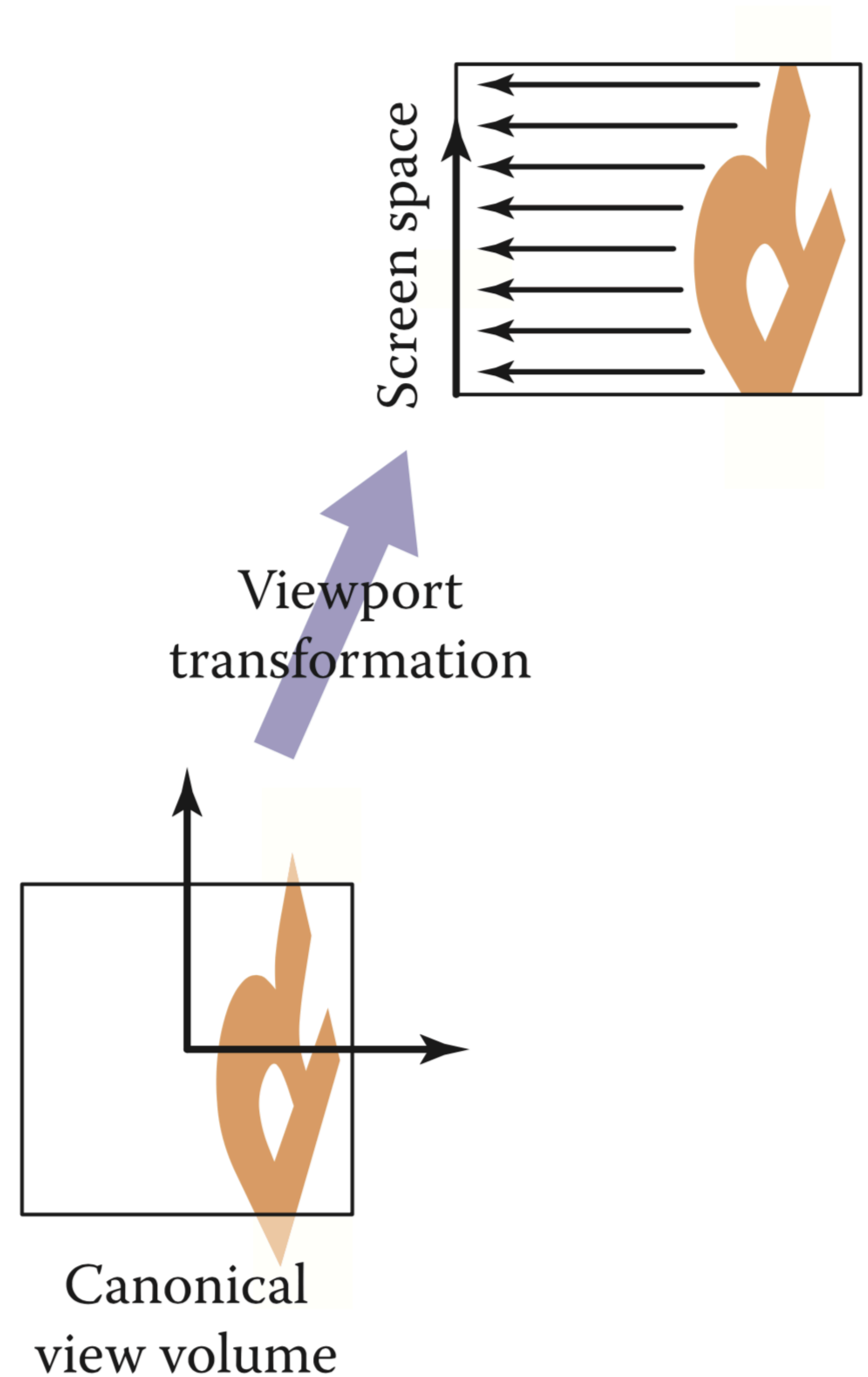
Step-by-Step Viewing Transformations

(Each arrow is a matrix)



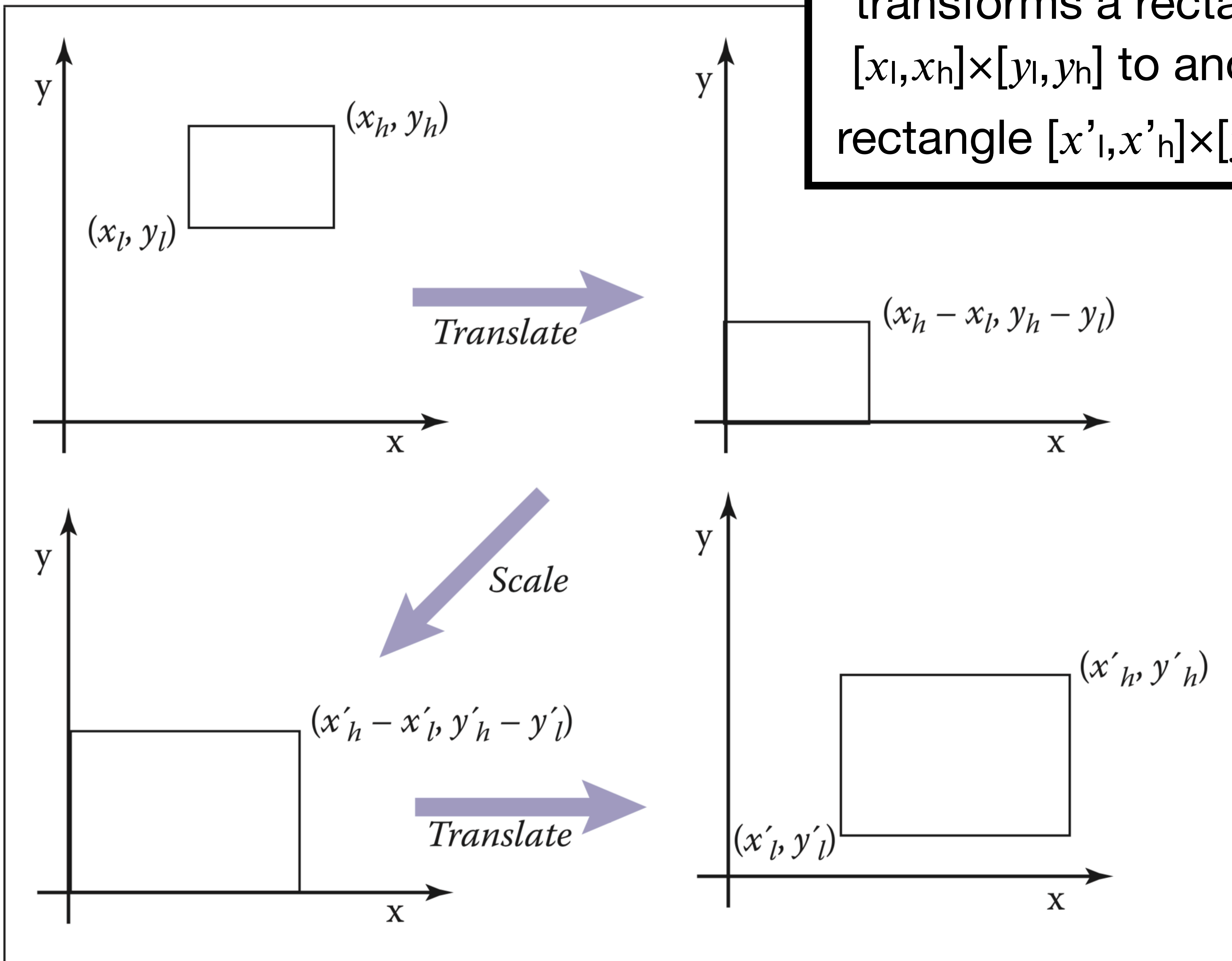
Viewport Transformation

- Goal: Transform from a canonical 2D space to pixel coordinates
  - Canonical space: $(x_{canonical}, y_{canonical}) \in [-1,1] \times [-1,1]$
  - Pixel space: $(x_{screen}, y_{screen}) \in$ $[0.5, n_x-0.5] \times [0.5, n_y-0.5]$
- Initially, we will think of this as transformation of a 2D to 2D space



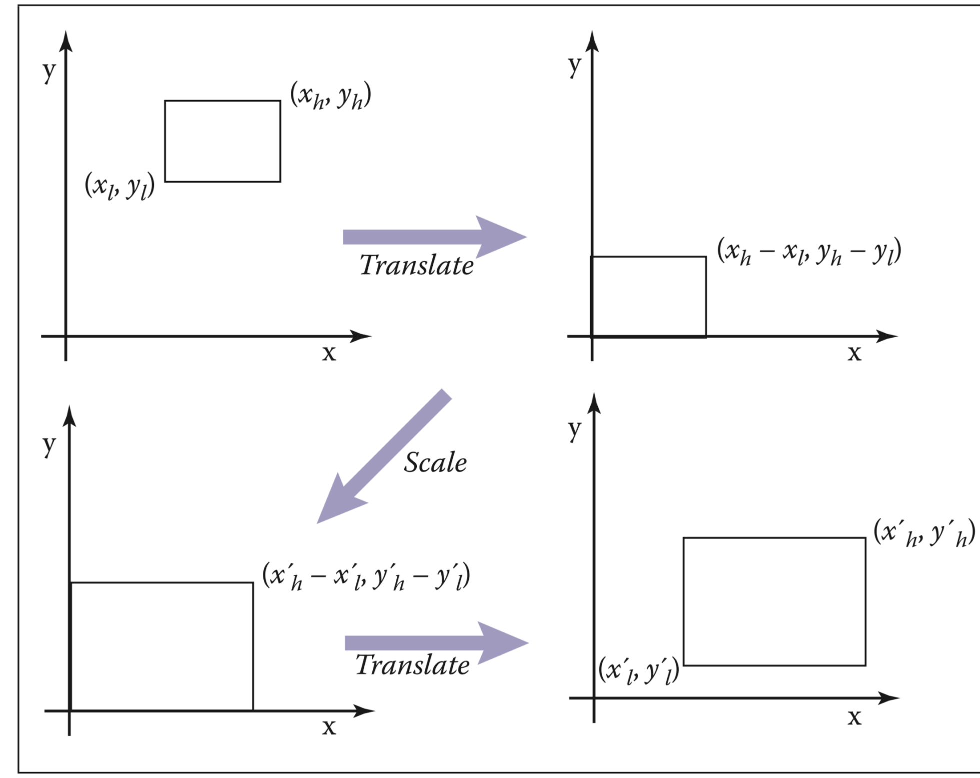
Viewports as Windowing

A **windowing operation** transforms a rectangle $[x_l, x_h] \times [y_l, y_h]$ to another rectangle $[x'_l, x'_h] \times [y'_l, y'_h]$



Viewports as Windowing

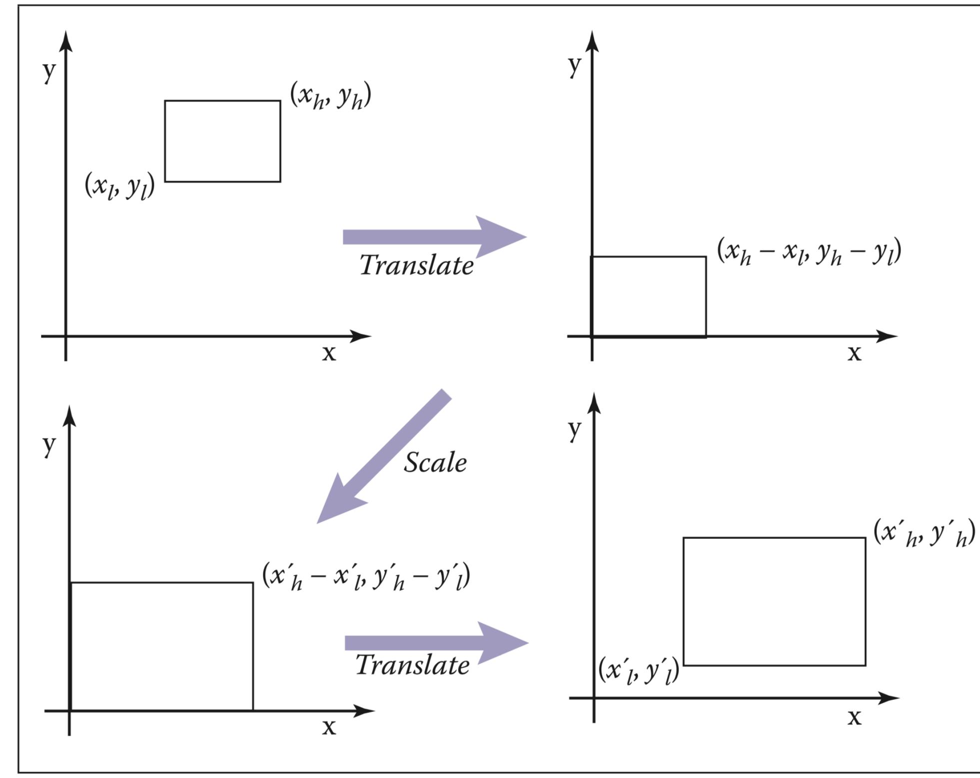
- Decompose windowing into three steps



$$\mathrm{translate}(x'_l,\ y'_l)\ \mathrm{scale}\left(\frac{x'_h - x'_l}{x_h - x_l},\ \frac{y'_h - y'_l}{y_h - y_l}\right)\ \mathrm{translate}(-x_l, -y_l)$$

Viewports as Windowing

- Decompose windowing into three steps

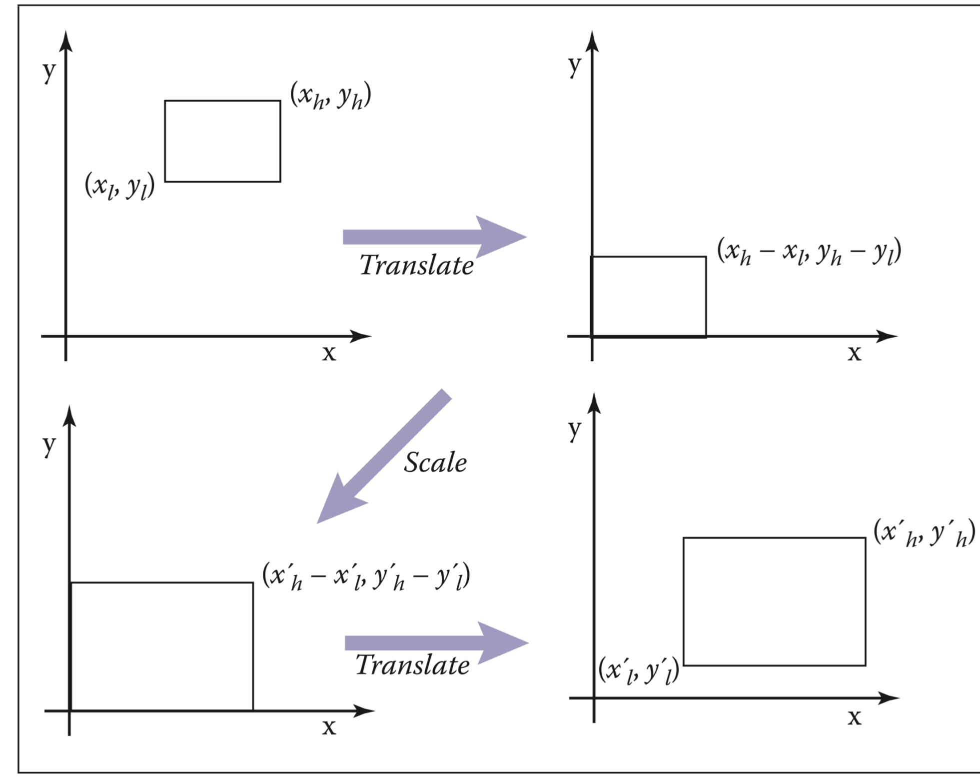


$$\text{translate}\left(x'_l,\ y'_l\right)\ \text{scale}\left(\frac{x'_h - x'_l}{x_h - x_l},\ \frac{y'_h - y'_l}{y_h - y_l}\right)\ \text{translate}(-x_l, -y_l)$$

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

Viewports as Windowing

- Decompose windowing into three steps

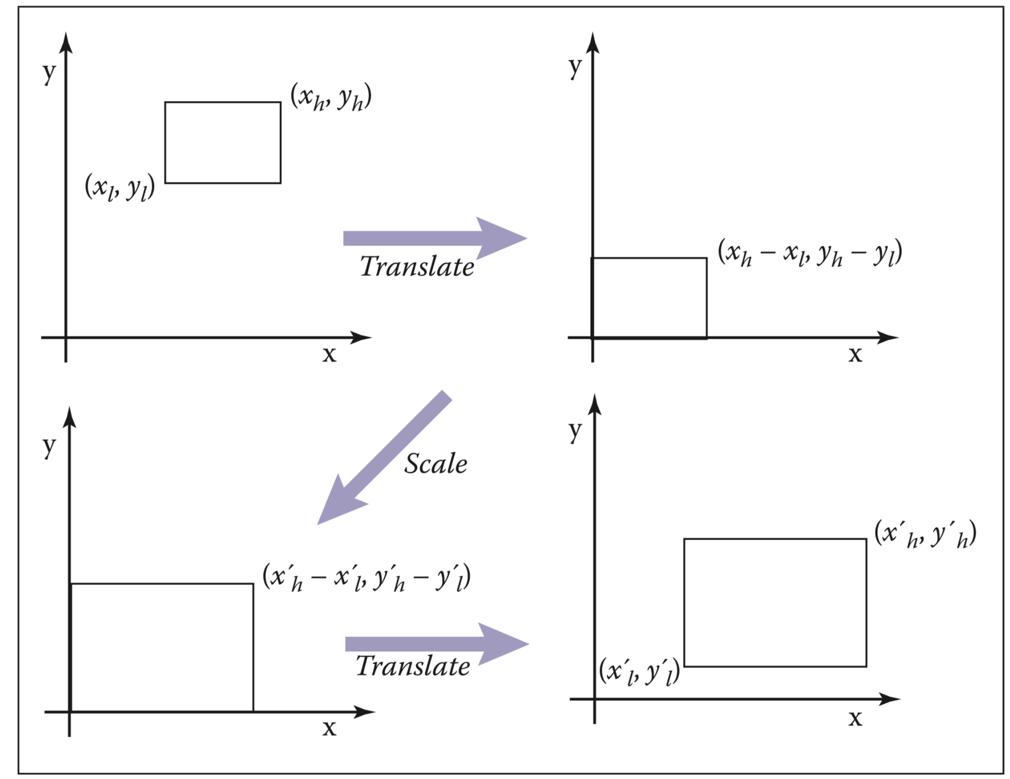


$$\text{translate}(x'_l,\ y'_l)\ \text{scale}\left(\frac{x'_h - x'_l}{x_h - x_l},\ \frac{y'_h - y'_l}{y_h - y_l}\right)\ \text{translate}(-x_l, -y_l)$$

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

Viewports as Windowing

- Decompose windowing into three steps



$$\text{translate}(x'_l,\ y'_l)\ \text{scale}\left(\frac{x'_h - x'_l}{x_h - x_l},\ \frac{y'_h - y'_l}{y_h - y_l}\right)\ \text{translate}(-x_l, -y_l)$$

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

Viewports as Windowing

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

- Multiplying together:

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

Sidebar: Combining a 3x3 Linear Matrix Followed by a Translation

- Translation *after* the linear transformation can always be read off separately.
- Often useful for debugging.

$$\begin{bmatrix} 1 & 0 & 0 & x_t \\ 0 & 1 & 0 & y_t \\ 0 & 0 & 1 & z_t \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & x_t \\ a_{21} & a_{22} & a_{23} & y_t \\ a_{31} & a_{32} & a_{33} & z_t \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Using Windowing to Define the Viewport Transformation

- Plugging in with our known constants:

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

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

Using Windowing to Define the Viewport Transformation

- Plugging in with our known constants:

$$\begin{bmatrix} \frac{x'_h - x'_l}{x_h - x_l} & 0 & \frac{x'_l x_h - x'_h x_l}{x_h - x_l} \\ 0 & \frac{y'_h - y'_l}{y_h - y_l} & \frac{y'_l y_h - y'_h y_l}{y_h - y_l} \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} x_{\text{screen}} \\ y_{\text{screen}} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{n_x}{2} & 0 & \frac{n_x - 1}{2} \\ 0 & \frac{n_y}{2} & \frac{n_y - 1}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\text{canonical}} \\ y_{\text{canonical}} \\ 1 \end{bmatrix}$$

- Right now, we do not need z-values, but eventually we will need to carry them through with no changes:

Using Windowing to Define the Viewport Transformation

- Plugging in with our known constants:

$$\begin{bmatrix} \frac{x'_h - x'_l}{x_h - x_l} & 0 & \frac{x'_l x_h - x'_h x_l}{x_h - x_l} \\ 0 & \frac{y'_h - y'_l}{y_h - y_l} & \frac{y'_l y_h - y'_h y_l}{y_h - y_l} \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} x_{\text{screen}} \\ y_{\text{screen}} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{n_x}{2} & 0 & \frac{n_x - 1}{2} \\ 0 & \frac{n_y}{2} & \frac{n_y - 1}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\text{canonical}} \\ y_{\text{canonical}} \\ 1 \end{bmatrix}$$

- Right now, we do not need z-values, but eventually we will need to carry them through with no changes:

$$M_{\text{vp}} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x - 1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y - 1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Using Windowing to Define the Viewport Transformation

- Plugging in with our known constants:

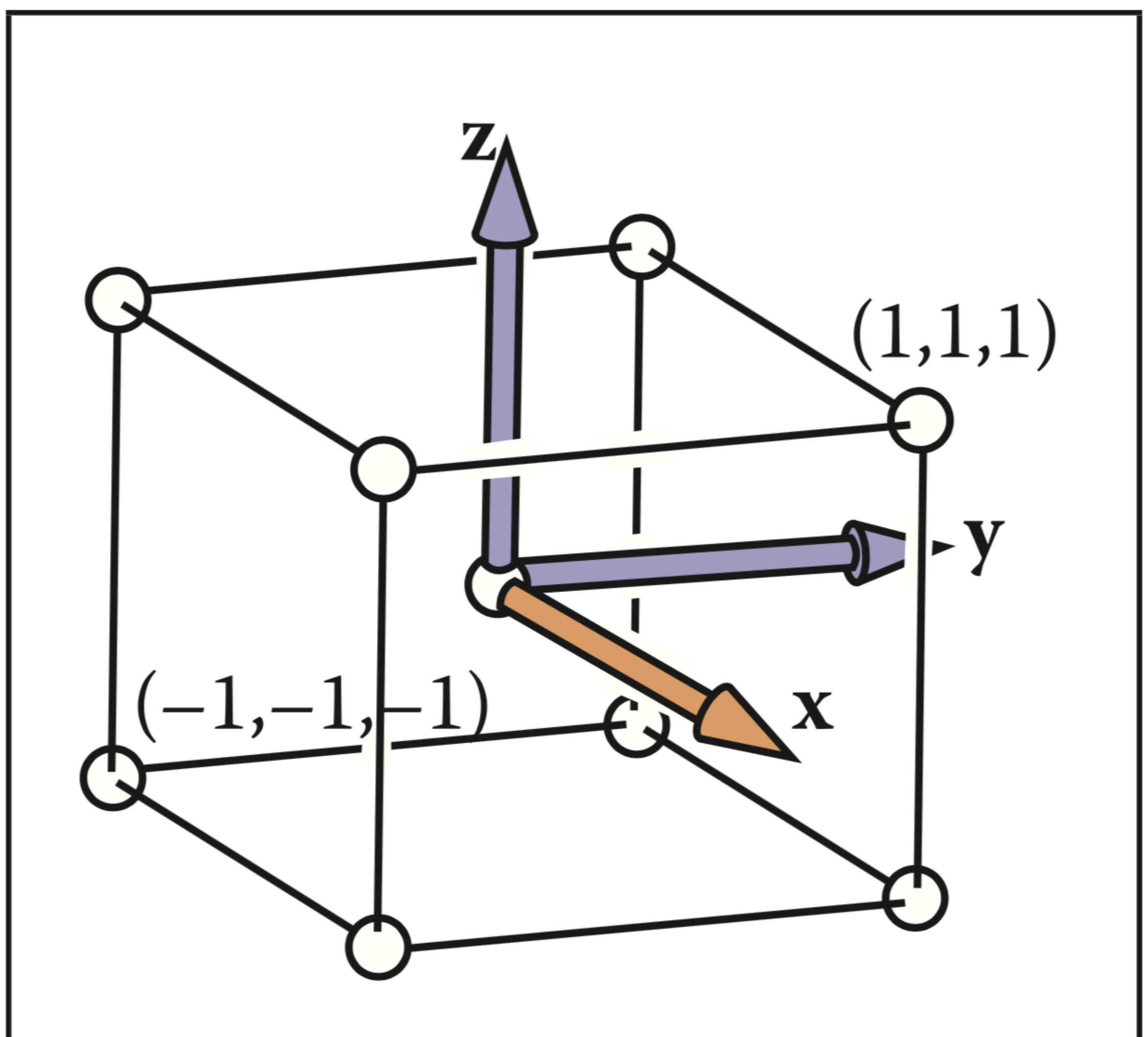
$$\begin{bmatrix} \frac{x'_h - x'_l}{x_h - x_l} & 0 & \frac{x'_l x_h - x'_h x_l}{x_h - x_l} \\ 0 & \frac{y'_h - y'_l}{y_h - y_l} & \frac{y'_l y_h - y'_h y_l}{y_h - y_l} \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} x_{\text{screen}} \\ y_{\text{screen}} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{n_x}{2} & 0 & \frac{n_x - 1}{2} \\ 0 & \frac{n_y}{2} & \frac{n_y - 1}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\text{canonical}} \\ y_{\text{canonical}} \\ 1 \end{bmatrix}$$

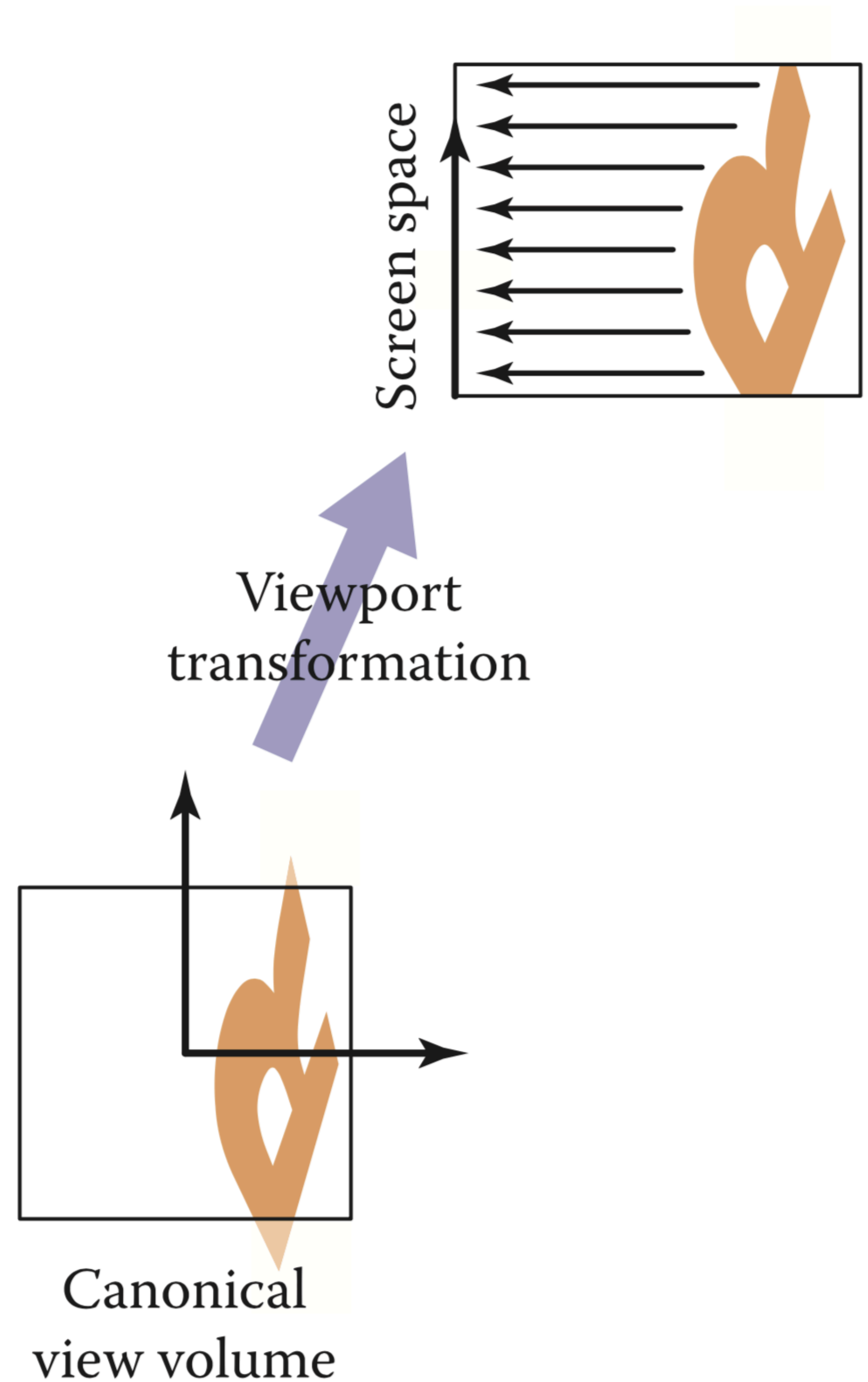
- Right now, we do not need z-values, but eventually we will need to carry them through with no changes:

$$M_{\text{vp}} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x - 1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y - 1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Canonical View Volume

- In actuality, our viewport transformation will work with the **canonical view volume**





Orthographic Projection

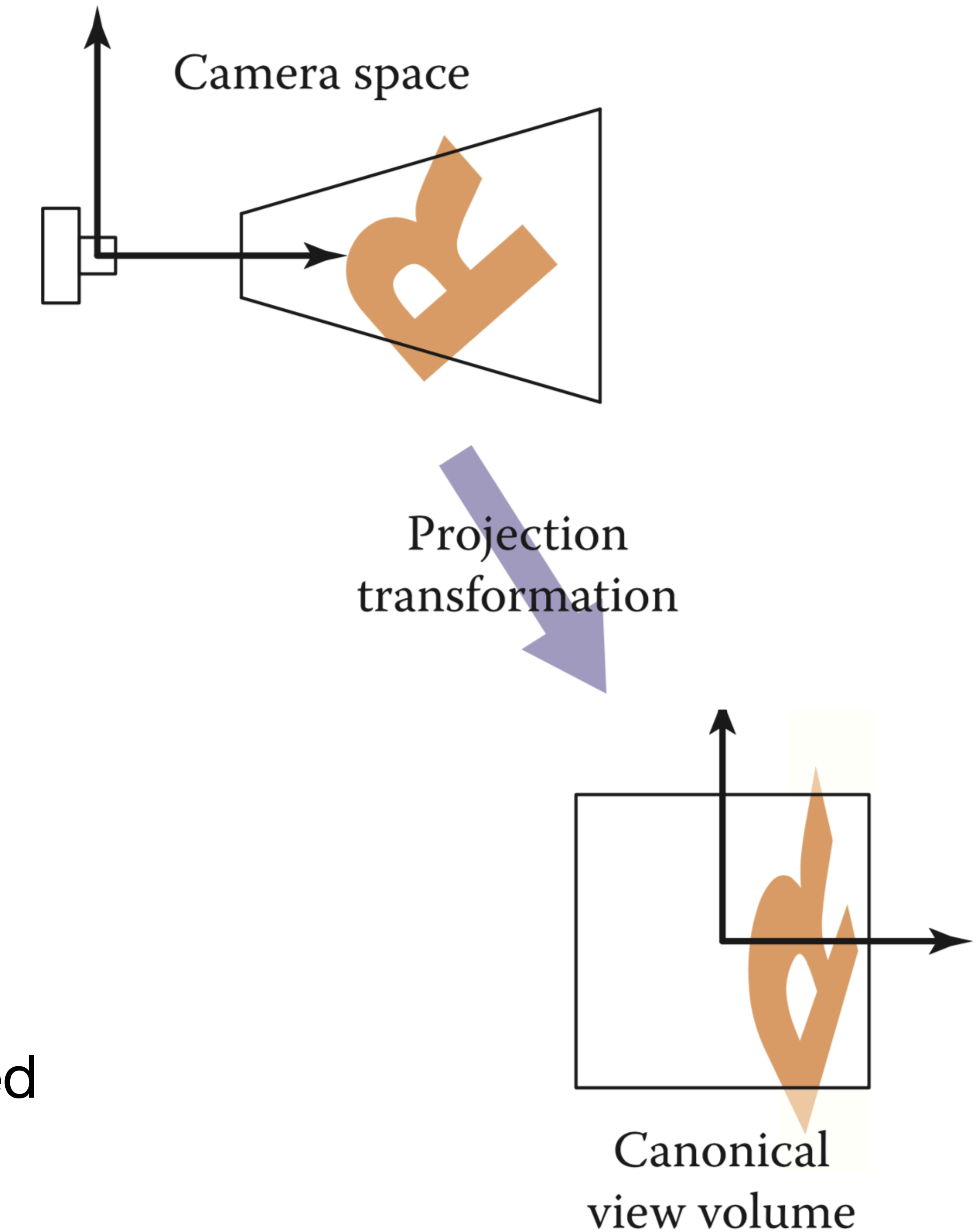
- Goal: Convert objects from 3D representation to canonical view volume

- We will start by modeling this 3D space as an axis-aligned boxe

  - View volume: $[l,r]\times[b,t]\times[f,n]$

  - Canonical view volume: [-1,1]×[-1,1]×[-1,1]

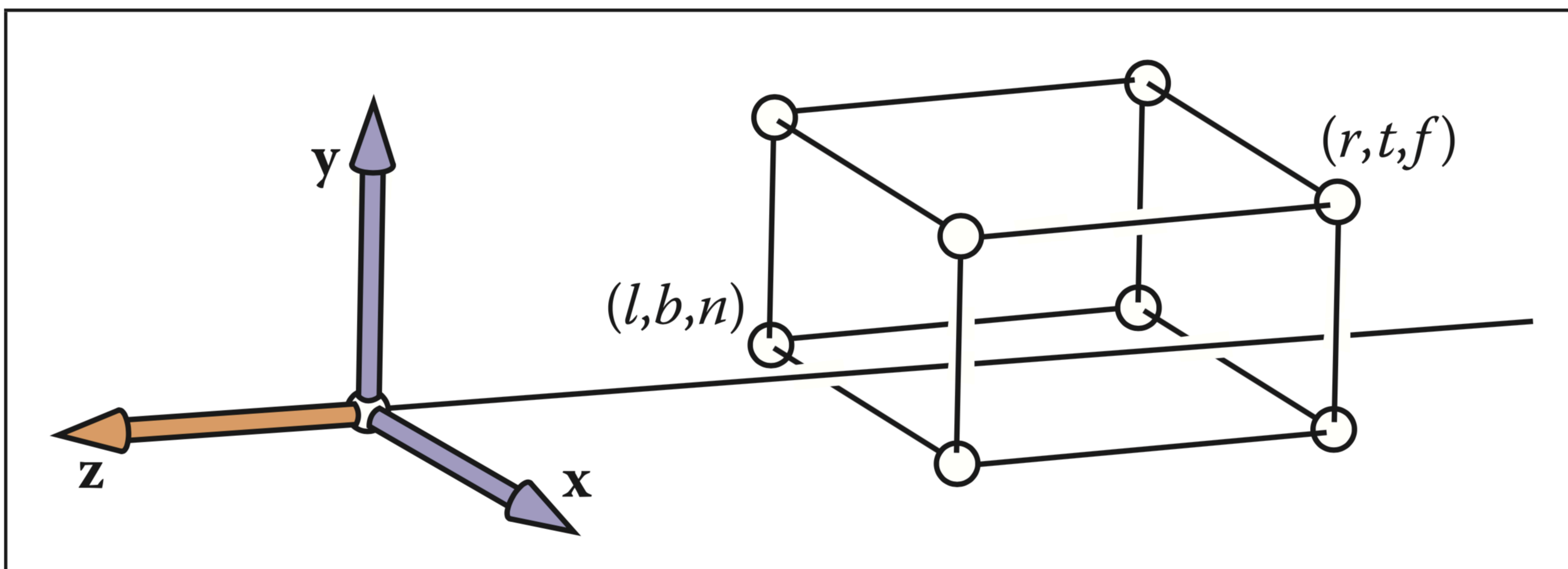
- Reshapes the view volume as defined by the camera



Orthographic Projection

- **Orthographic view volume** defined by six scalars:
- Convention: $n > f$, but note that both are *negative*

$$
\begin{aligned}
x = l &\equiv \text{left plane},\\
x = r &\equiv \text{right plane},\\
y = b &\equiv \text{bottom plane},\\
y = t &\equiv \text{top plane},\\
z = n &\equiv \text{near plane},\\
z = f &\equiv \text{far plane}.
\end{aligned}
$$



Orthographic Projection

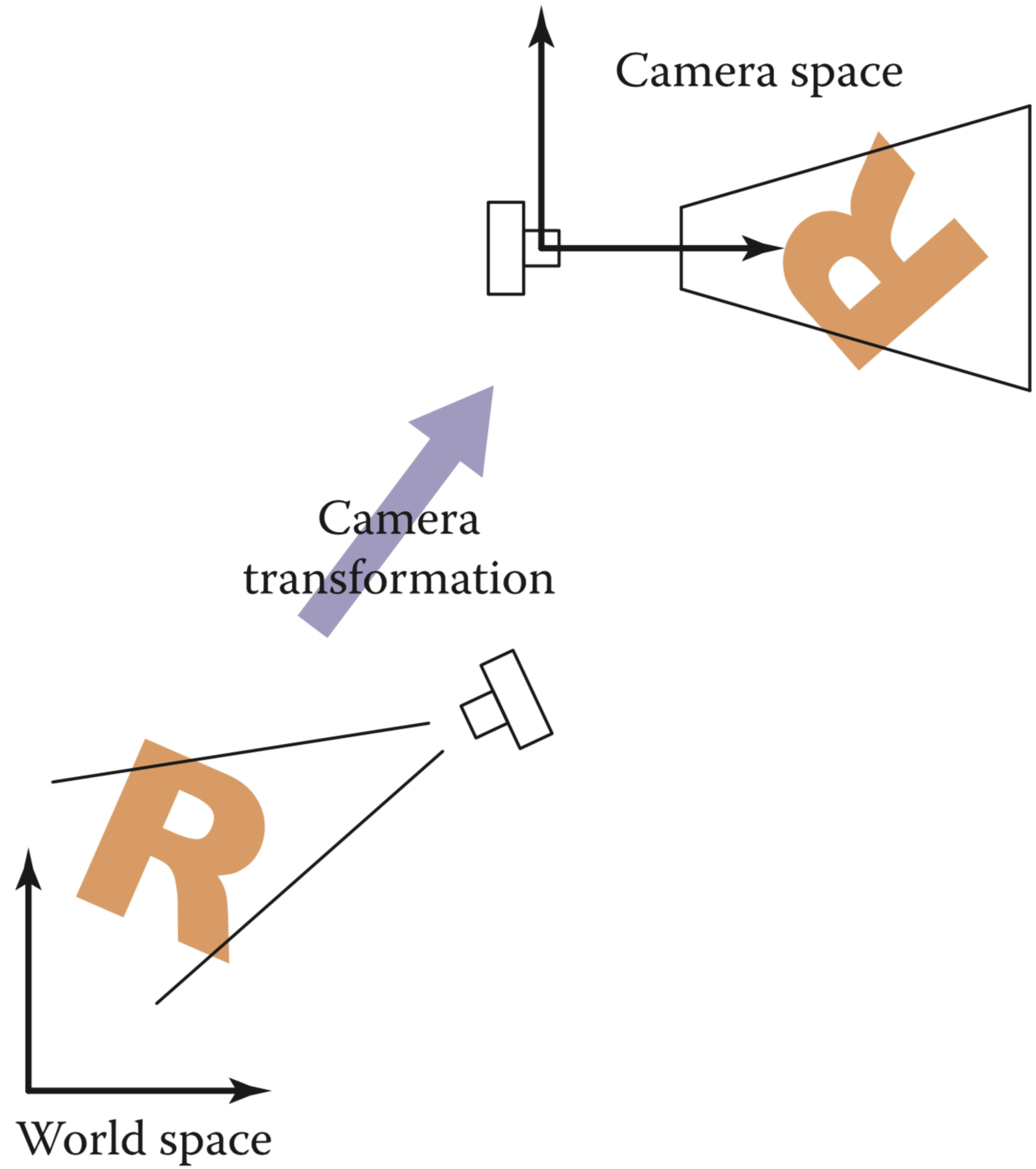
- Just a 3D windowing transformation!

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

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

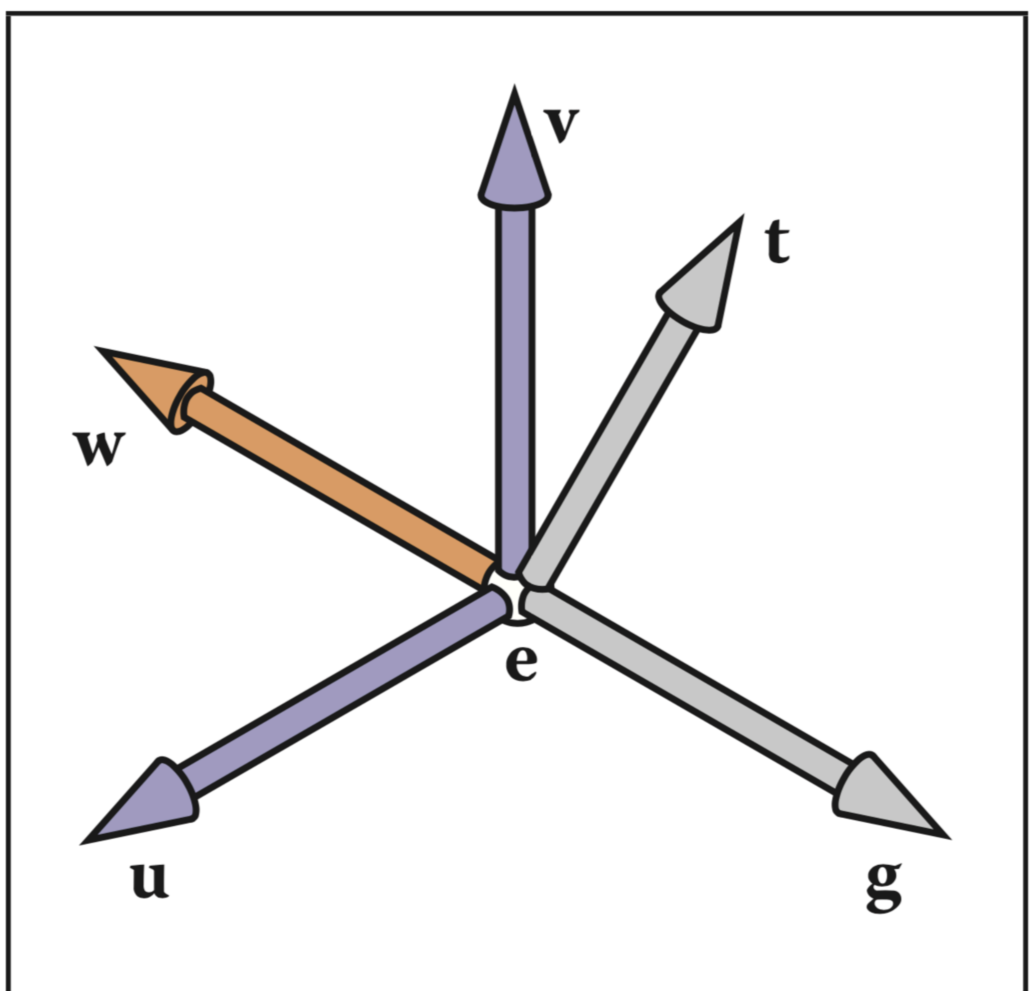
Camera Transformations

- Goal: Transform 3D space to arbitrary camera parameters
- Camera modeled with three vectors:
  - **e**, the eye position
  - **g**, the gaze direction
  - **t**, the view up direction



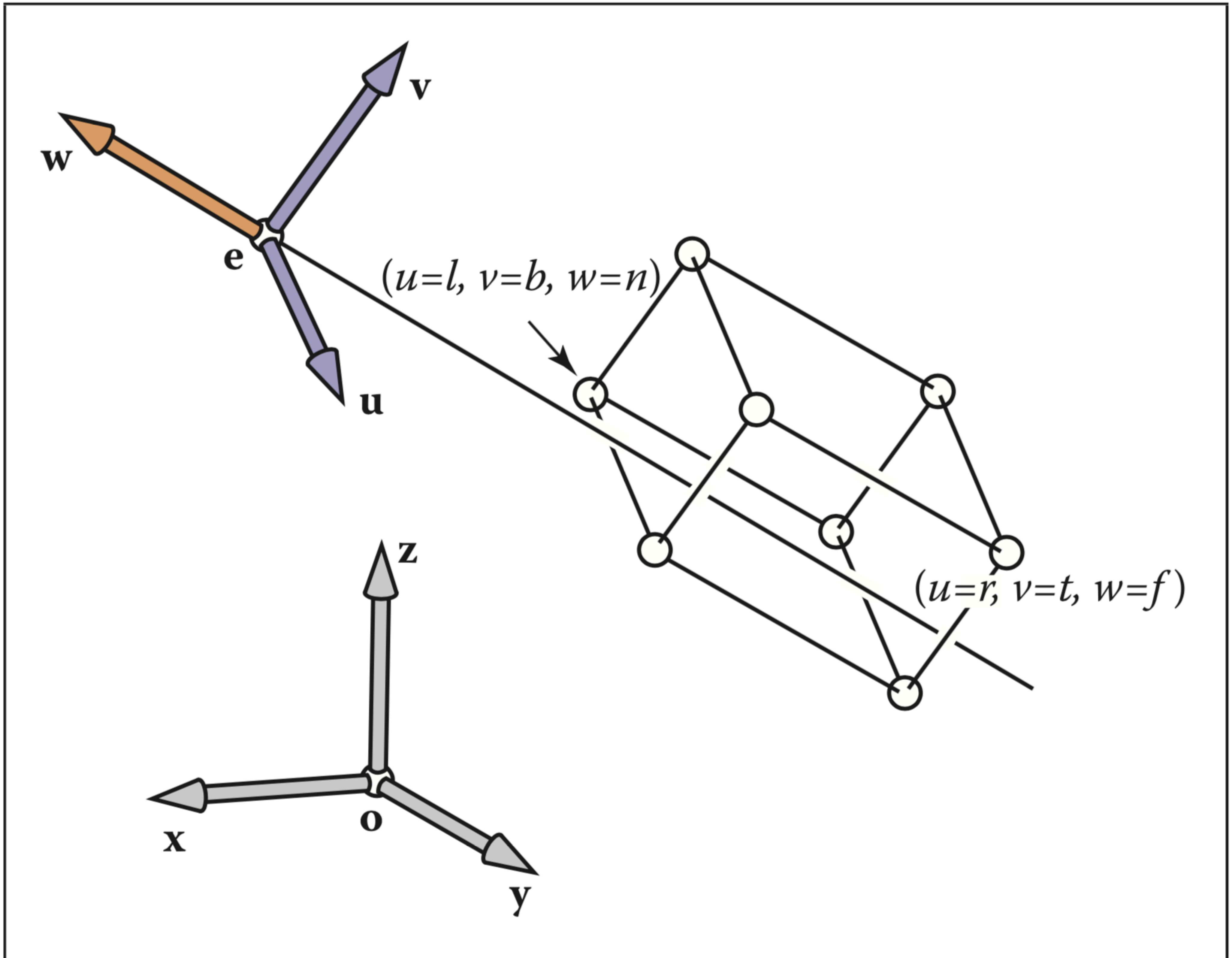
Camera Coordinates

- We will convert to a camera coordinate system with origin, $\mathbf{e}$, and orthogonal basis vectors $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$



$$\mathbf{w} = -\frac{\mathbf{g}}{\|\mathbf{g}\|},$$
$$\mathbf{u} = -\frac{\mathbf{t}\times\mathbf{w}}{\|\mathbf{t}\times\mathbf{w}\|},$$
$$\mathbf{v} = \mathbf{w}\times\mathbf{u}.$$

Camera Coordinates



Changing Coordinates

- We need to both translate the origin and change coordinate systems

$$\mathbf{M}_{\text{cam}} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Viewing Algorithm

```
construct M_vp
construct M_orth
construct M_cam
M = M_vp * M_orth * M_cam
for each 3D object O {
  O_screen = M * O
  draw(O_screen)
}
```

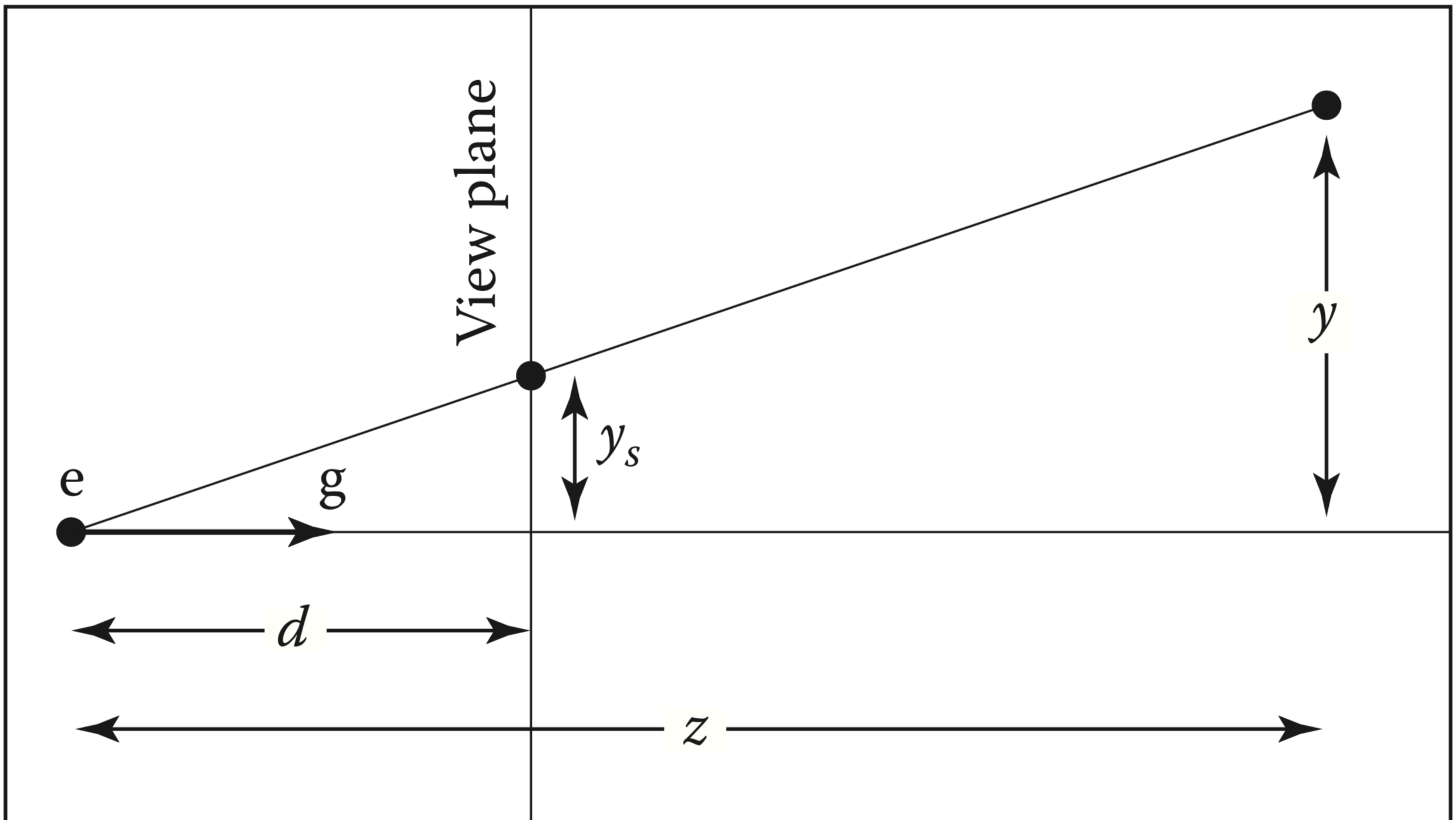
For example, if O is a triangle, with vertices $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$, transform all 3 vertices $\mathbf{Ma}$, $\mathbf{Mb}$, and $\mathbf{Mc}$

Projective Transformations

Relative Size Based on Distance

- Key idea of perspective: the size of an object on the screen is proportional to 1/z

$$y_s = \frac{d}{z}\,y$$



Problem: How to divide by z?

- Linear transformations: $$x' = ax + by + cz$$

Problem: How to divide by z?

- Linear transformations: $$x' = ax + by + cz$$
- Affine transformations: $$x' = ax + by + cz + d$$

Problem: How to divide by z?

- Linear transformations: $$x' = ax + by + cz$$
- Affine transformations: $$x' = ax + by + cz + d$$
- Our trick: using w in homogeneous coordinates as a denominator: $$x' = \frac{a_1 x + b_1 y + c_1 z + d_1}{ex + fy + gz + h}$$

Problem: How to divide by z?

- Linear transformations:

$$x' = ax + by + cz$$

- Affine transformations:

$$x' = ax + by + cz + d$$

- Our trick: using w in homogeneous coordinates as a denominator:

- Same denominator for all coordinates.

$$\begin{aligned} x' &= \frac{a_1 x + b_1 y + c_1 z + d_1}{ex + fy + gz + h} \\ y' &= \frac{a_2 x + b_2 y + c_2 z + d_2}{ex + fy + gz + h} \\ z' &= \frac{a_3 x + b_3 y + c_3 z + d_3}{ex + fy + gz + h} \end{aligned}$$

Projective Transformations, or Homographies

$$\begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ \widetilde{w} \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \\ e & f & g & h \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Where we reinterpret coordinates by diving by w:

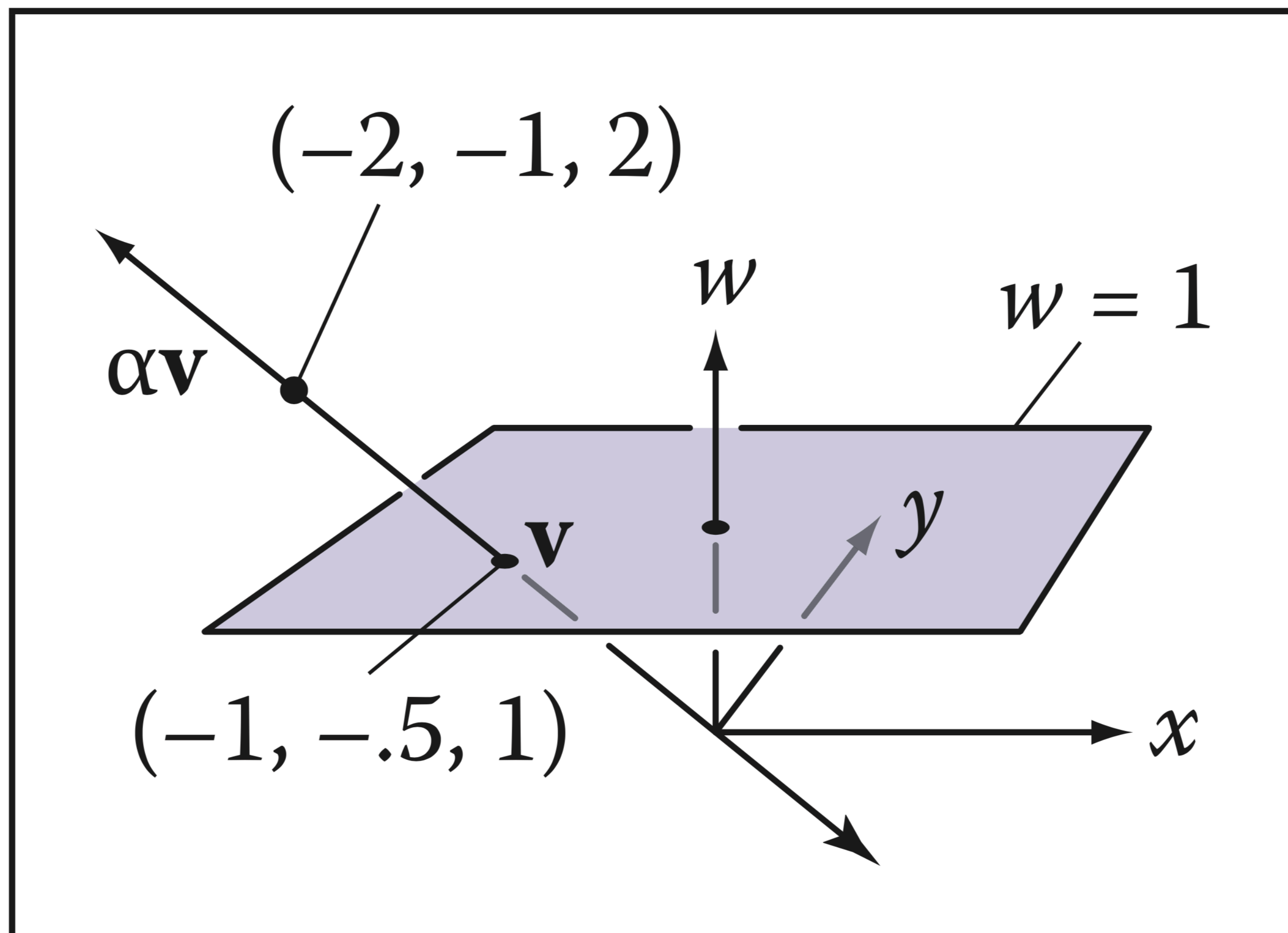
$$\left(x', y', z'\right) = (\tilde{x}/\widetilde{w}, \tilde{y}/\widetilde{w}, \tilde{z}/\widetilde{w})$$

Equivalence of Points

- Key idea: all scalar multiples of a vector are the same!
- Equivalently: we're treating points as lines in one dimension higher

$$\mathbf{x} \sim \alpha \mathbf{x}$$
$$\text{for all } \alpha \neq 0$$

We will only divide by w when we want the Cartesian coordinates

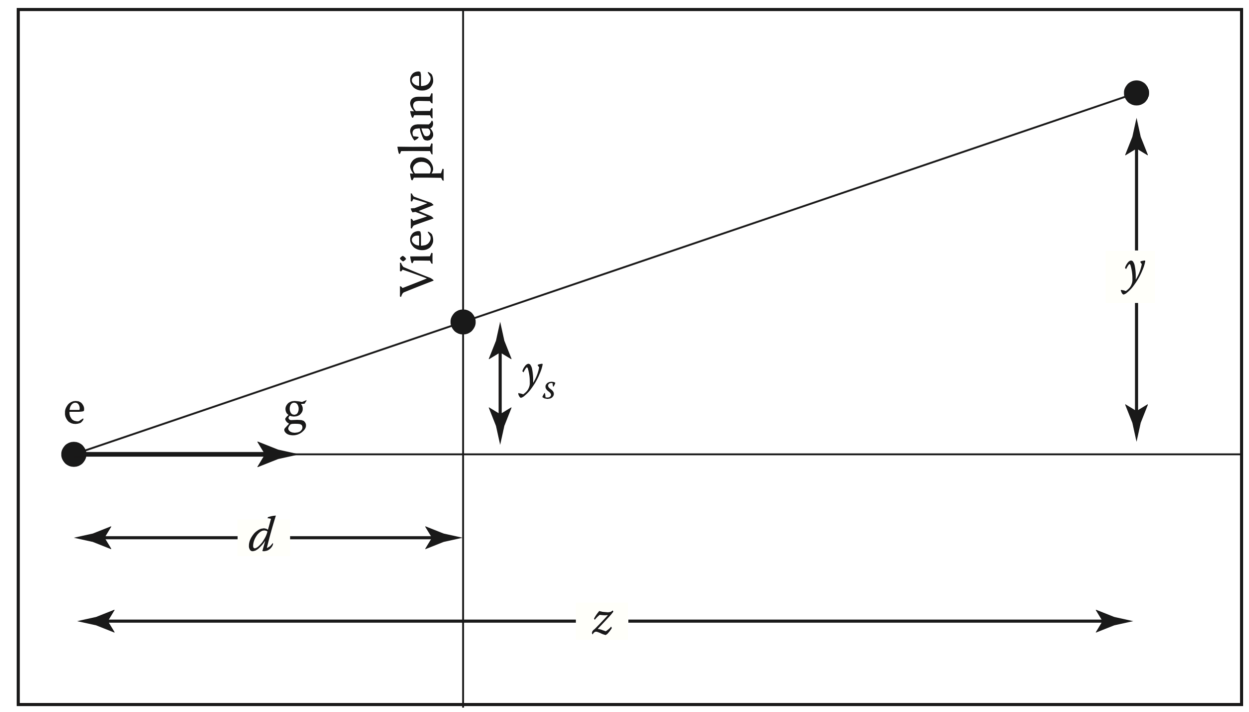


Perspective Projection

Using Homographies for Perspective

- We can now replace:

$$y_s = \frac{d}{z}\,y$$



- With:

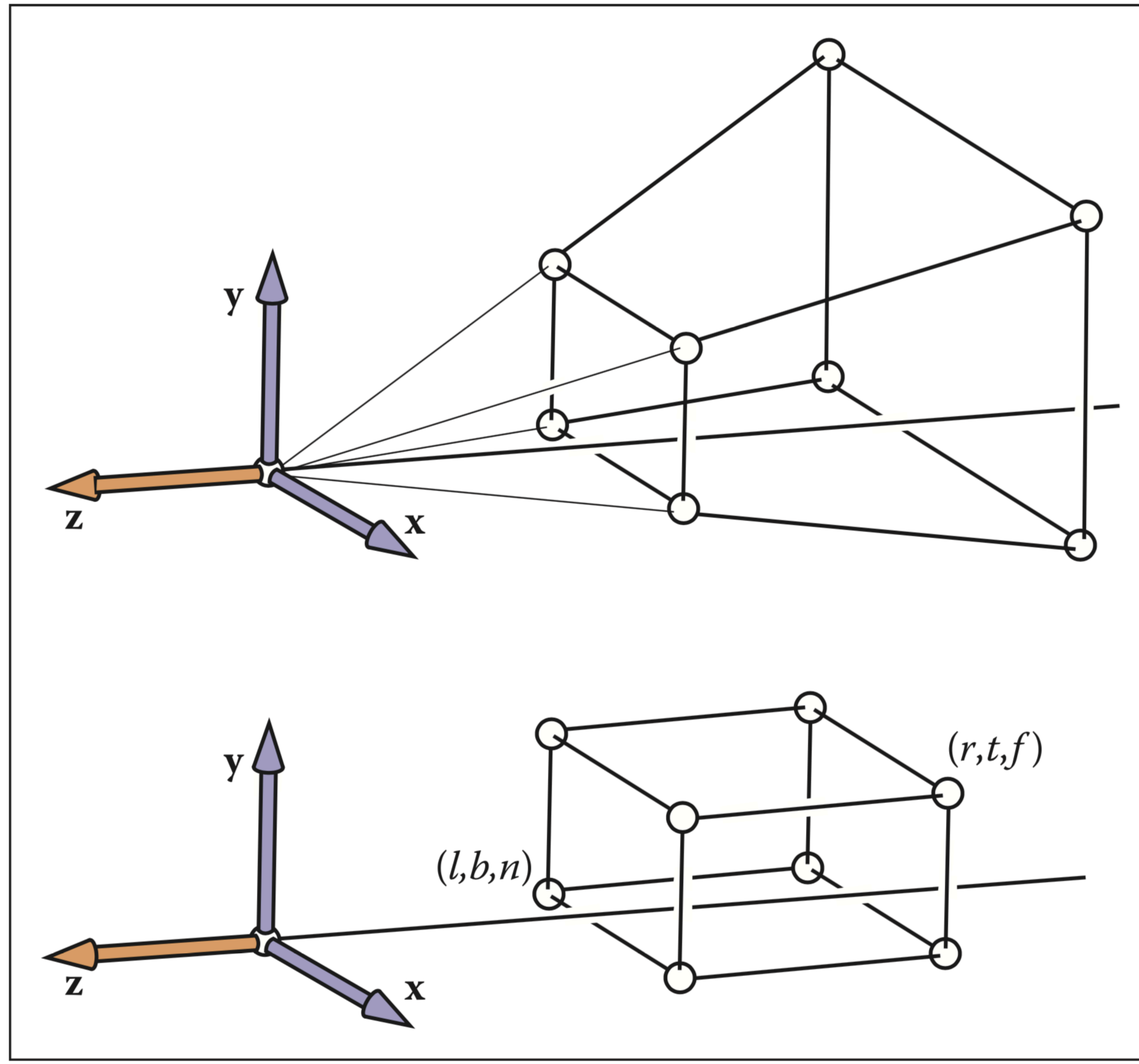
$$\begin{bmatrix} y_s \\ 1 \end{bmatrix} \sim \begin{bmatrix} d & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} y \\ z \\ 1 \end{bmatrix}$$

Perspective Matrix

- Our matrix:

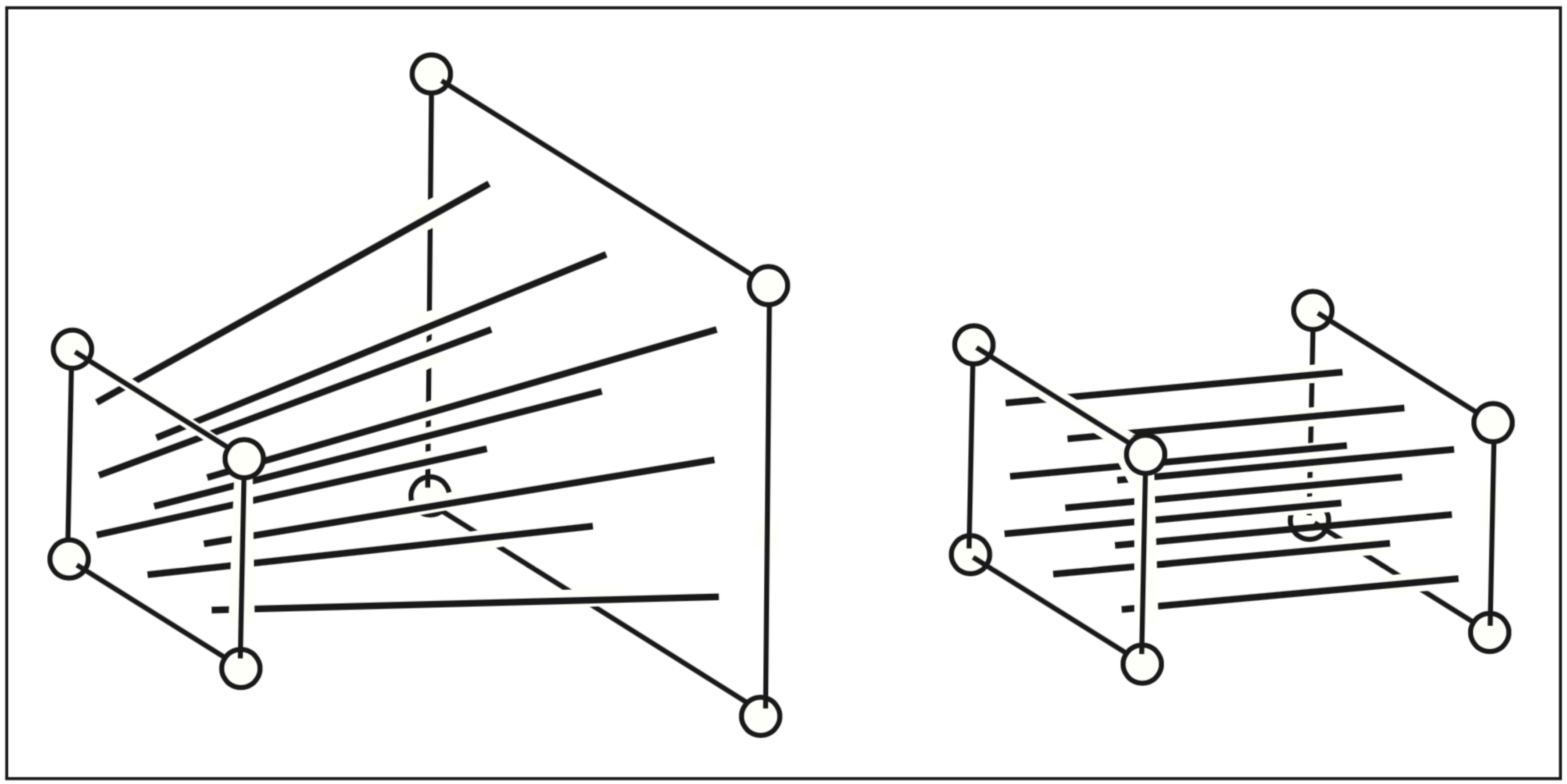
$$\mathbf{P} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

- Keeps near plane fixed, maps far plane to back of the box



Perspective Distortion

- Effect on view rays / lines:



- Note that affine transformation cannot do this because it keeps parallel lines parallel

Perspective Distortion

- Perspective matrix effect on coordinates is nonlinear distortion in z:

$$\mathbf{P}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Perspective Distortion

- Perspective matrix effect on coordinates is nonlinear distortion in z:

$$\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Perspective Distortion

- Perspective matrix effect on coordinates is nonlinear distortion in z:

$$\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} nx \\ ny \\ (n+f)z - fn \\ z \end{bmatrix} \sim \begin{bmatrix} \frac{nx}{z} \\ \frac{ny}{z} \\ n + f - \frac{fn}{z} \\ 1 \end{bmatrix}$$

Perspective Distortion

- Perspective matrix effect on coordinates is nonlinear distortion in z:

$$\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} nx \\ ny \\ (n+f)z - fn \\ z \end{bmatrix} \sim \begin{bmatrix} \frac{nx}{z} \\ \frac{ny}{z} \\ n + f - \frac{fn}{z} \\ 1 \end{bmatrix}$$

- But it does, however, preserve order in the z-coordinate (which will become useful very soon)

Perspective Projection Matrix

- Concatenating the perspective matrix with the orthographic projection provides the perspective projection matrix:

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

- We can define l, r, b, and t relative to the near plane, since we keep it fixed

Putting it all together

```
construct M_vp
construct M_per
construct M_cam
M = M_vp * M_per * M_cam
for each 3D object O {
  O_screen = M * O
  draw(O_screen)
}
```

Equivalently:

$$\mathbf{M} = \mathbf{M}_{\mathrm{vp}} \mathbf{M}_{\mathrm{orth}} \mathbf{P} \mathbf{M}_{\mathrm{cam}}$$

For a given vertex $\mathbf{a} = (x, y, z)$, $\mathbf{p} = \mathbf{M}\mathbf{a}$ should result in drawing $(x_{\mathbf{p}}/w_{\mathbf{p}}, y_{\mathbf{p}}/w_{\mathbf{p}}, z_{\mathbf{p}}/w_{\mathbf{p}})$ on the screen

Lec20 Required Reading

- FOCG, Ch. 8

Reminder: Assignment 05

Assigned: Wednesday, Oct. 30
Written Due: Monday, Nov. 11, 4:59:59 pm
Program Due: Wednesday, Nov. 13, 4:59:59 pm