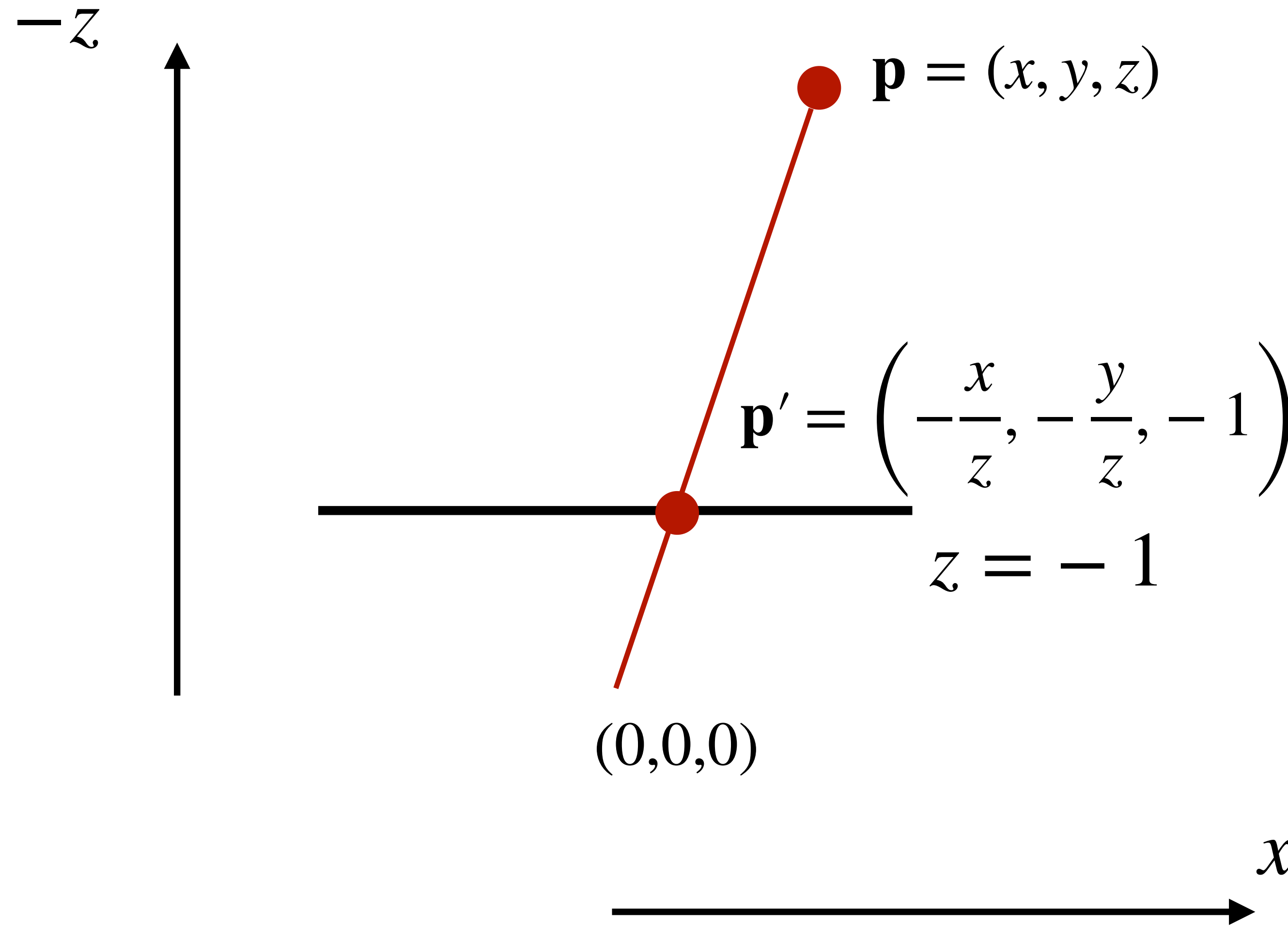


3D transformations

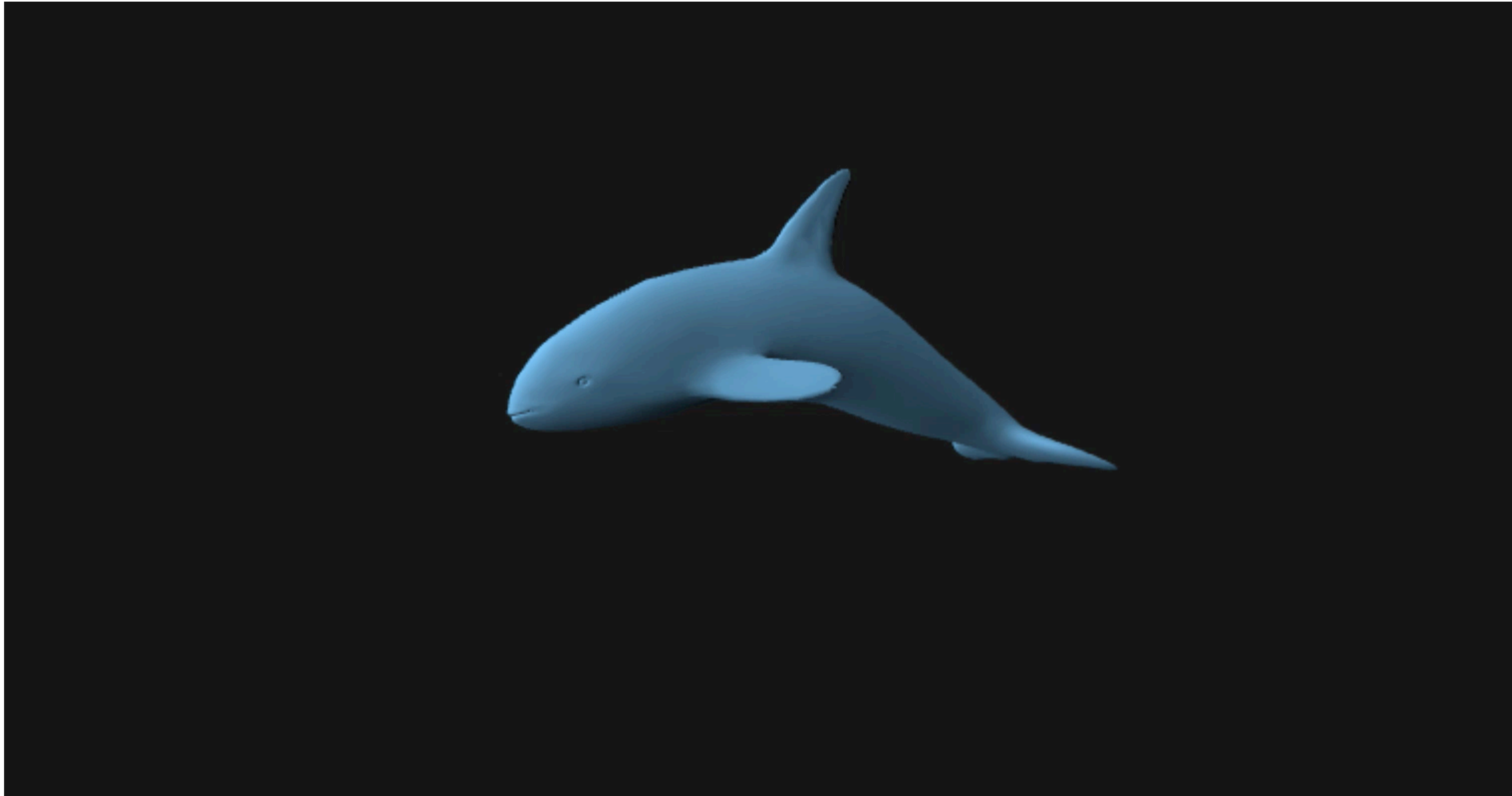
UCSD CSE 167

Tzu-Mao Li

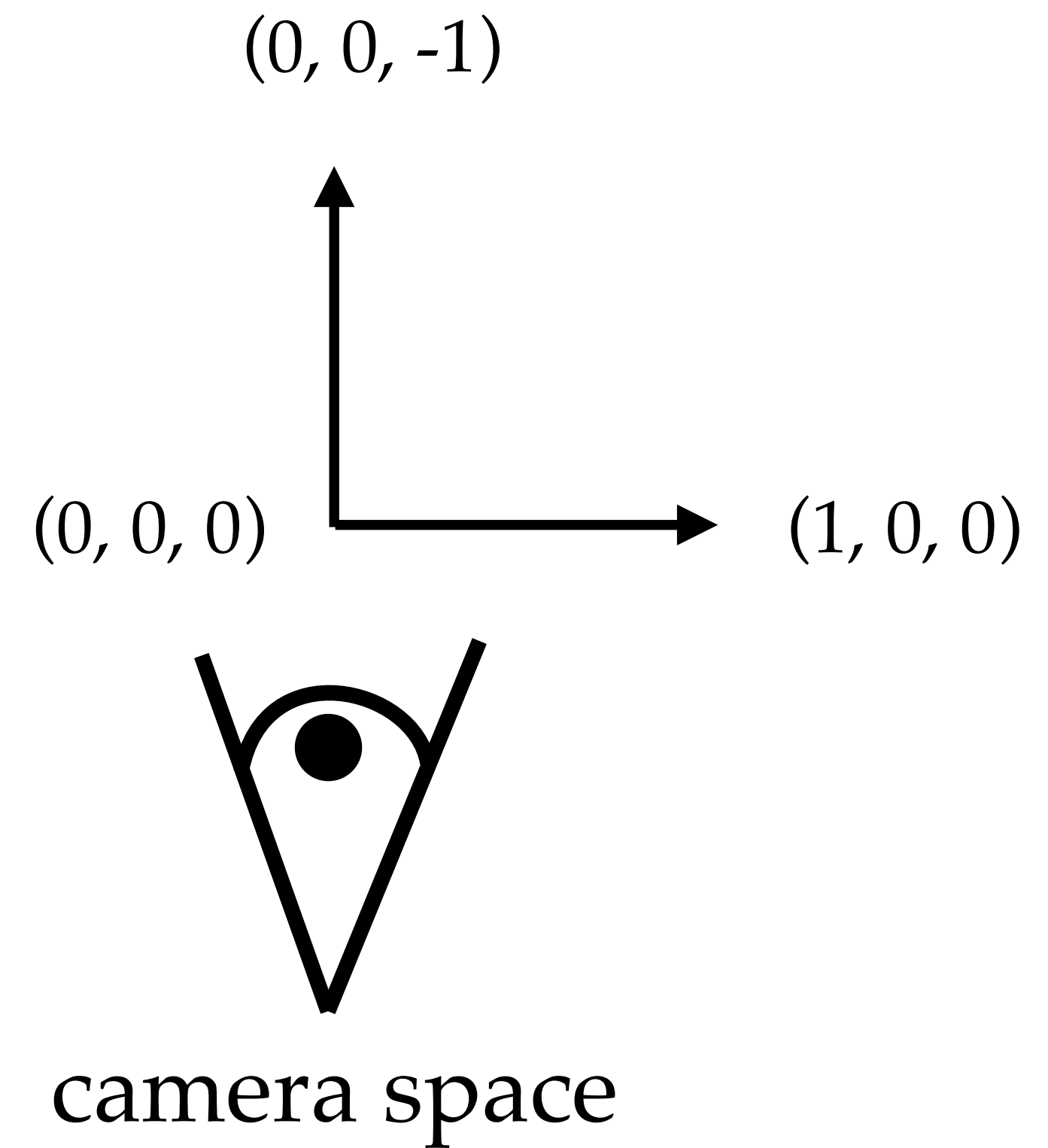
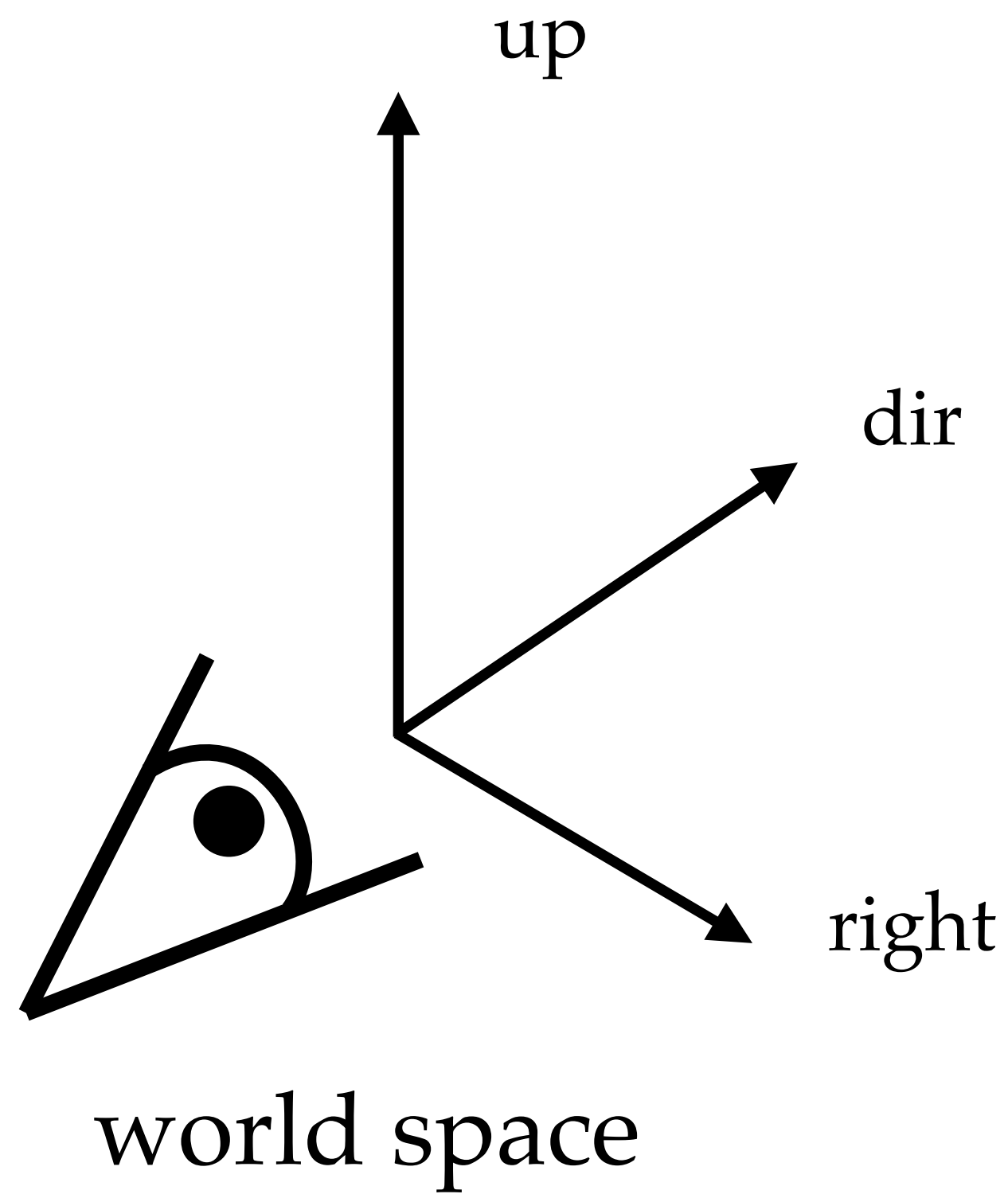
Motivation: generalize from fixed camera pose



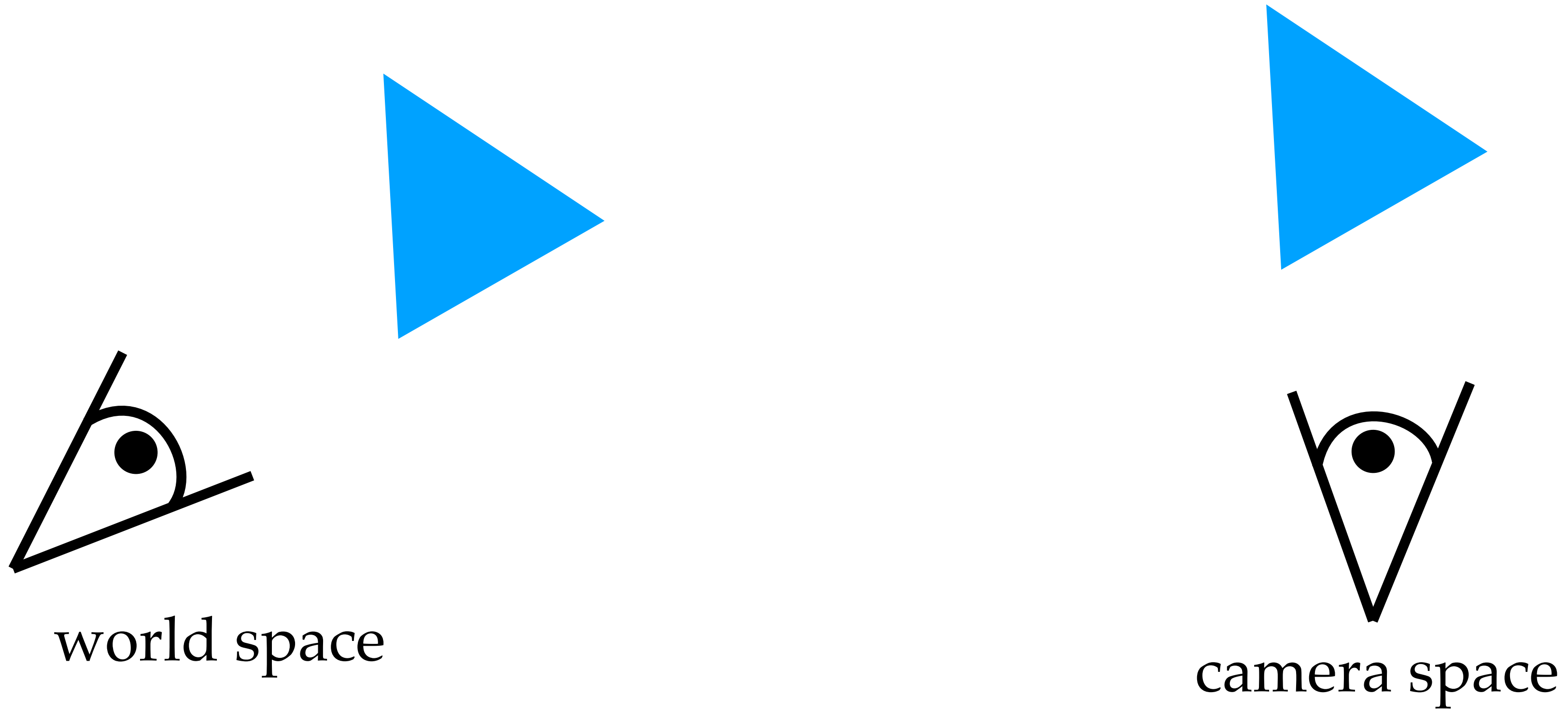
Motivation: 3D animation



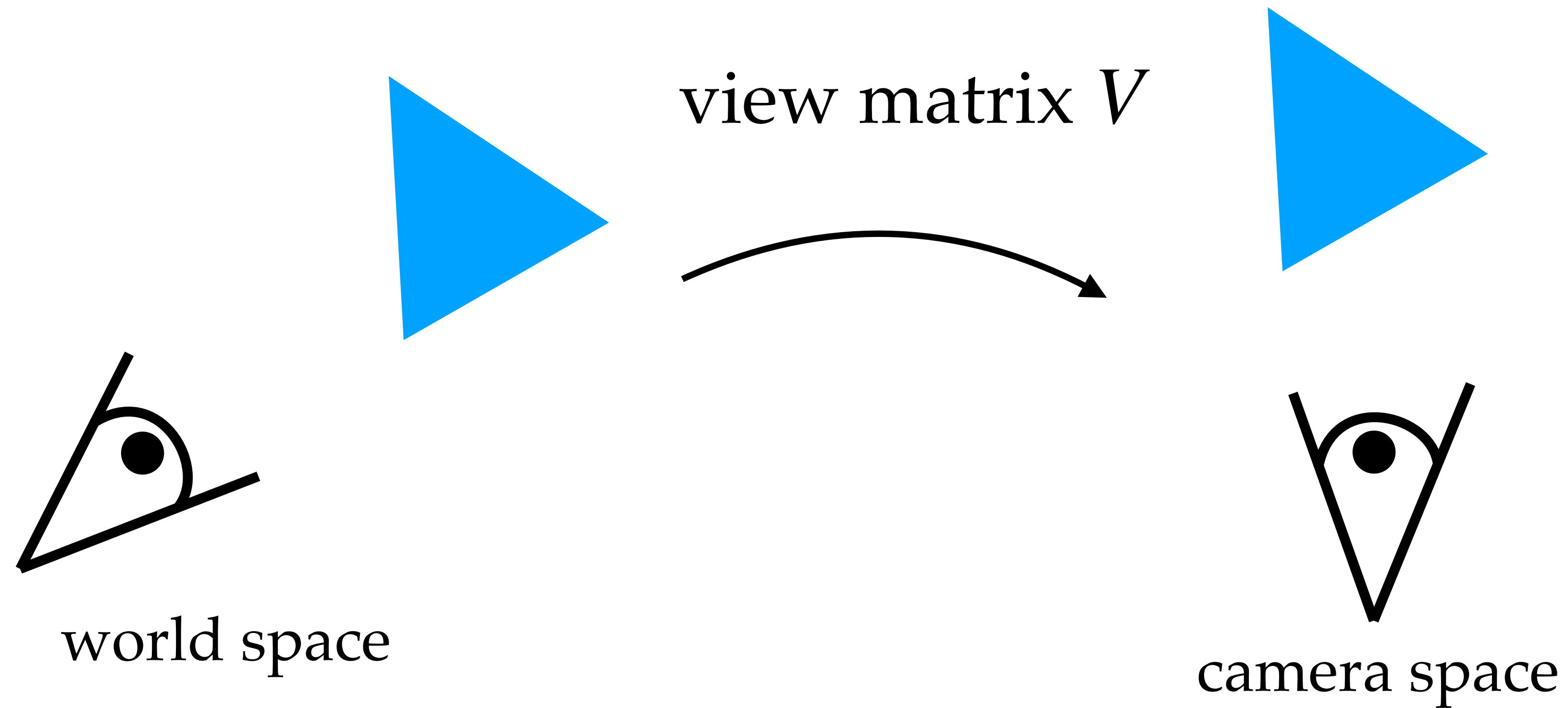
World space vs camera space



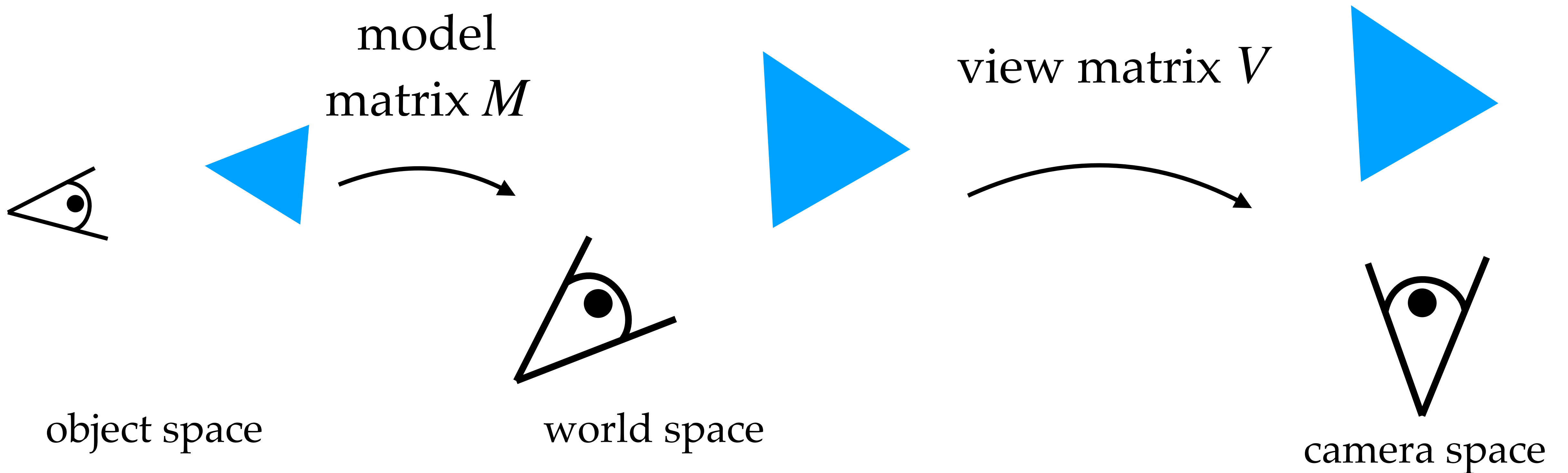
Given a triangle in world space,
we can transform it to camera space to rasterize it



The transform matrix from world to camera
is called the view matrix

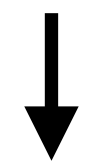


For animation, it's convenient to define a transformation between object and world



Recall: 2D affine transform

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$



$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

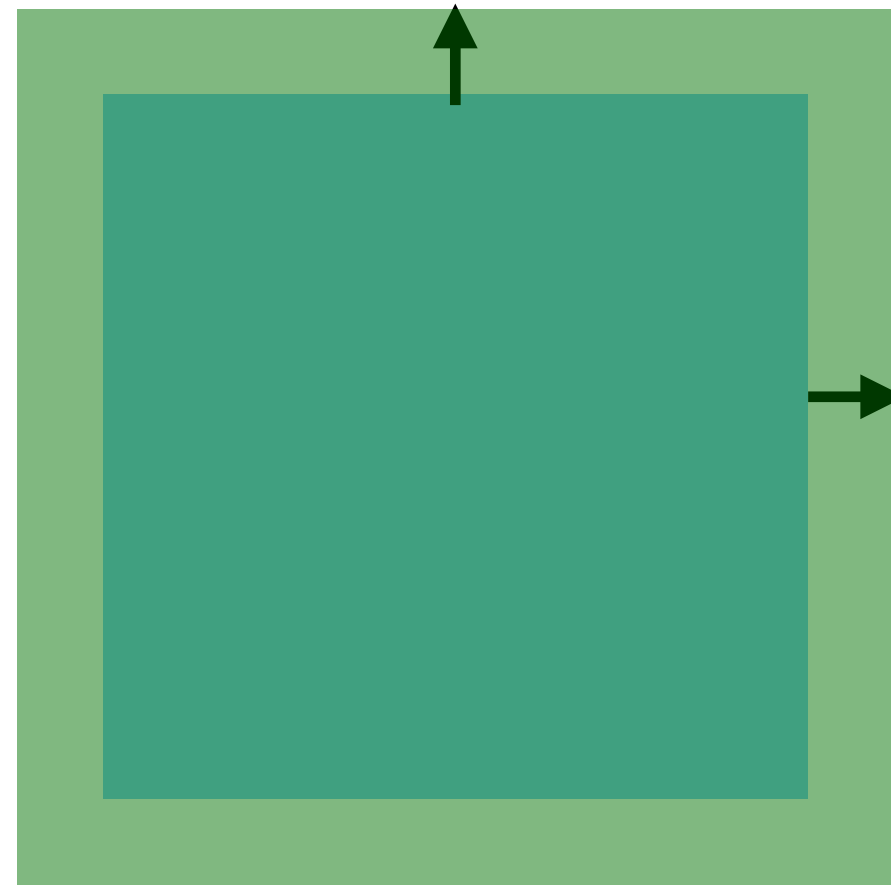
3D affine transform

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$



$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c & t_x \\ d & e & f & t_y \\ g & h & i & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Common 3D affine transformations



scaling

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Common 3D affine transformations

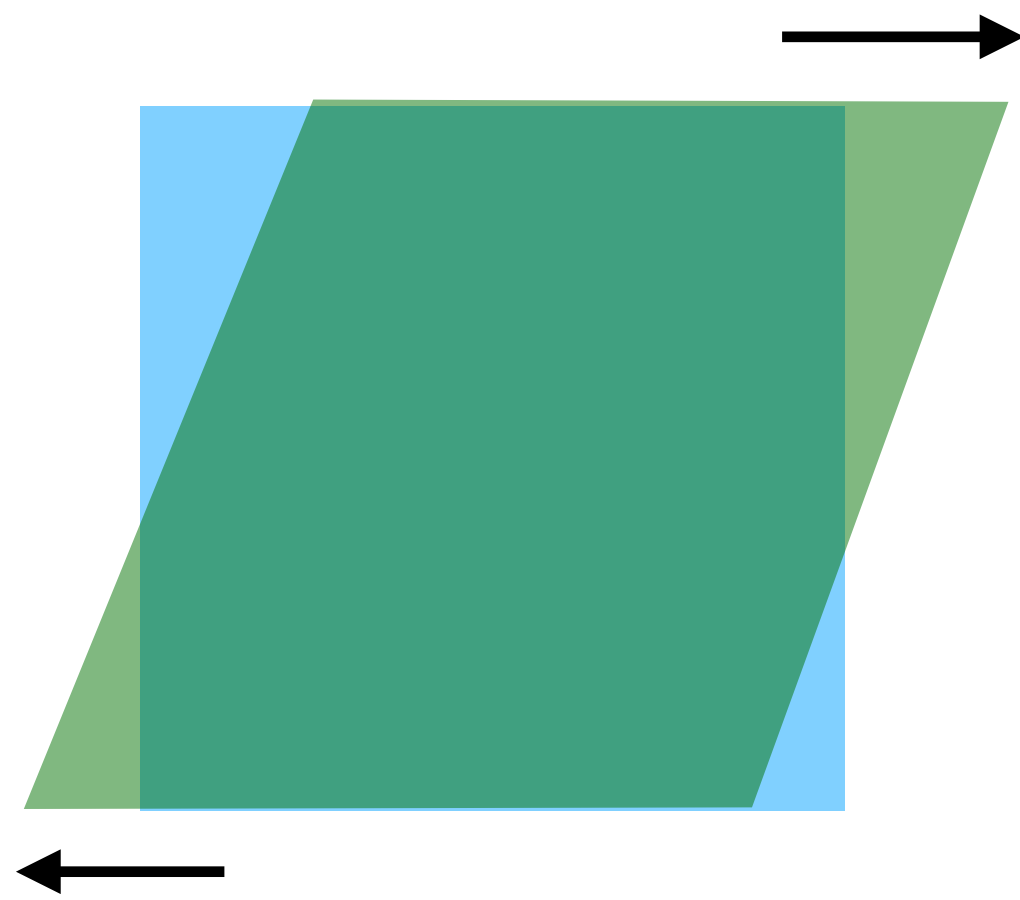
translation



$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Common 3D affine transformations

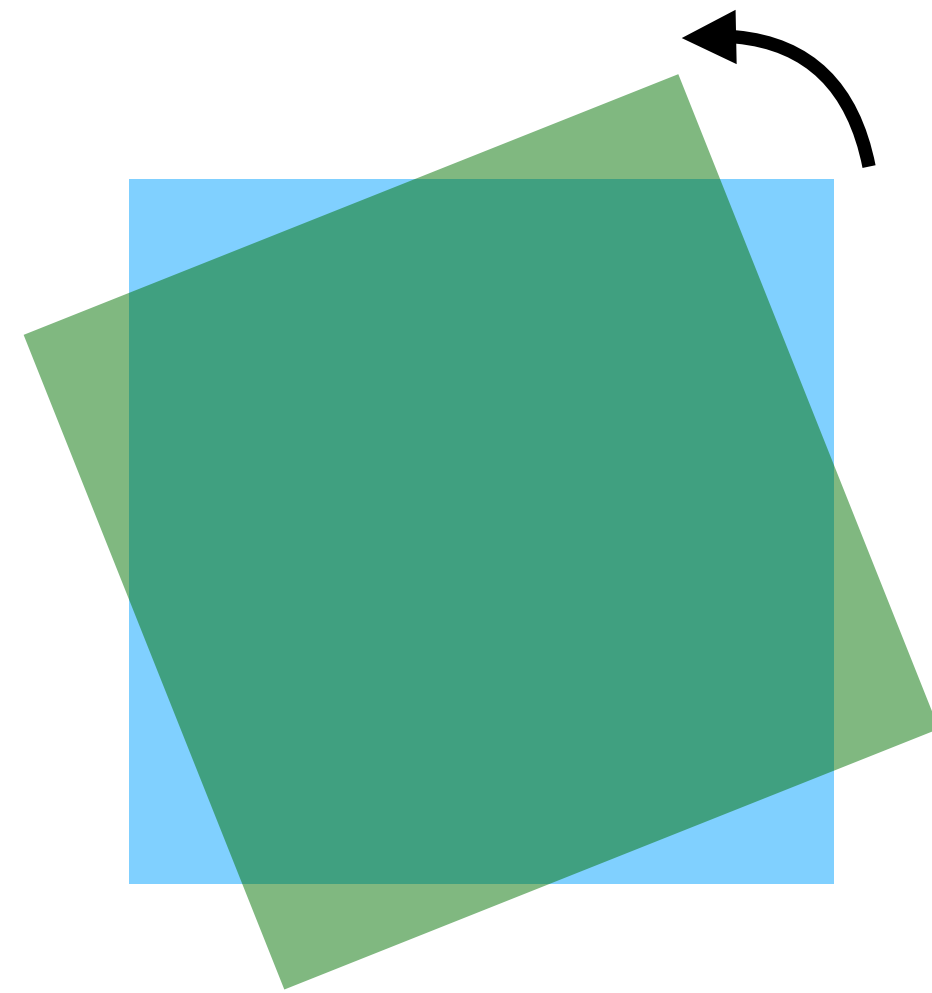
shearing



$$\begin{bmatrix} 1 & \lambda_x^y & \lambda_x^z & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Common 3D affine transformations

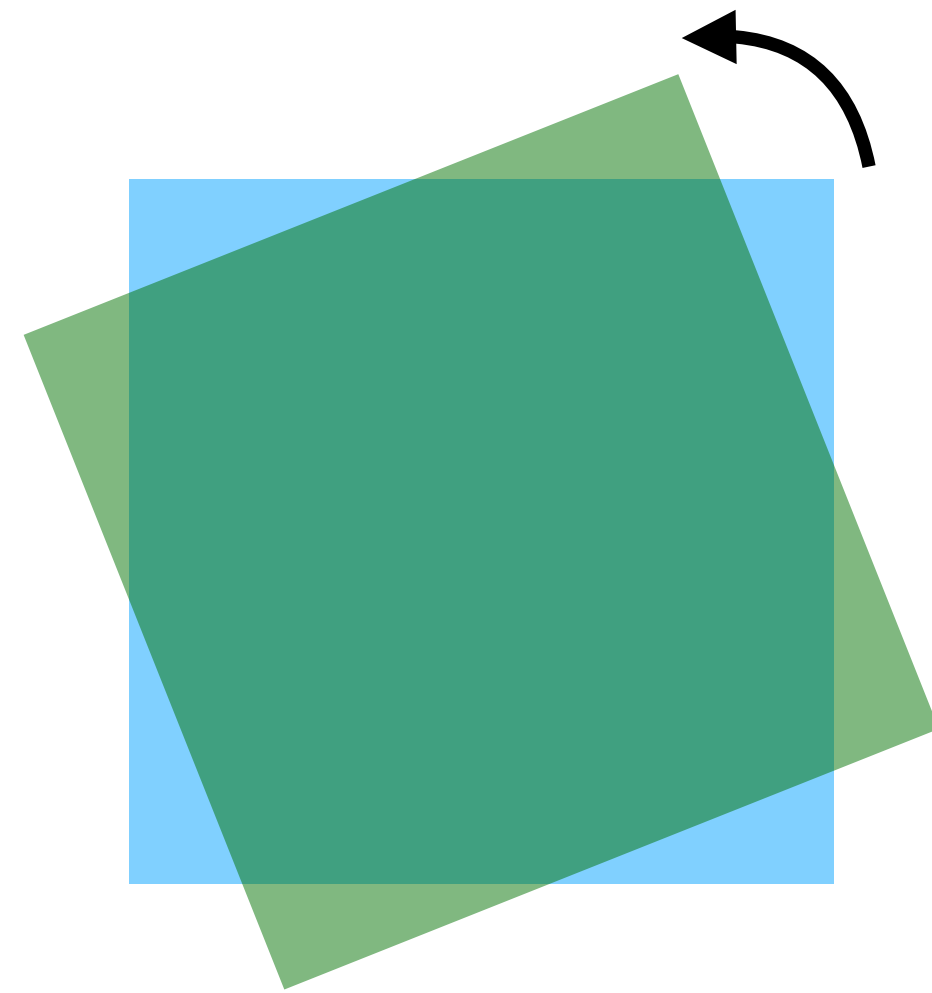
rotate around z



$$\begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

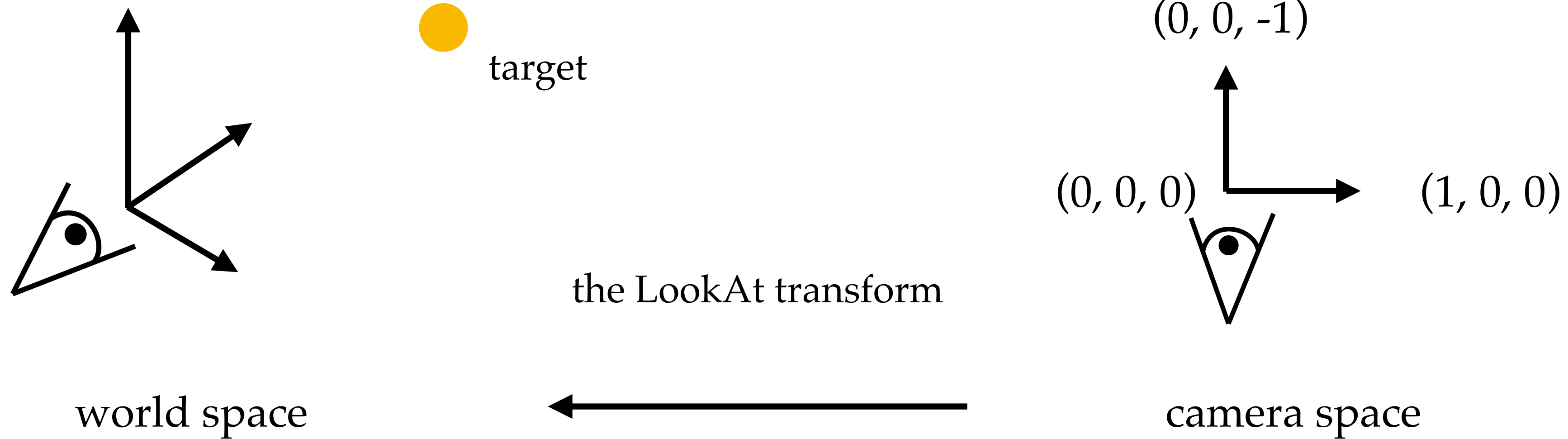
Common 3D affine transformations

rotate around an arbitrary axis \mathbf{a}
(we will get to this next Monday)



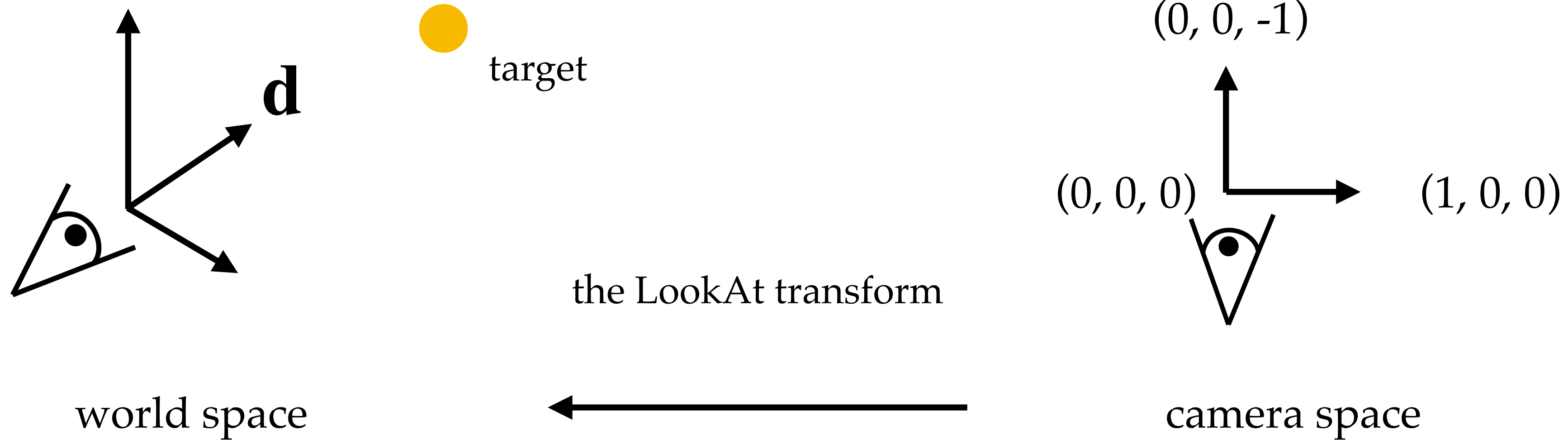
$$\begin{bmatrix} \mathbf{a} \cdot \mathbf{x}^2 + (1 - \mathbf{a} \cdot \mathbf{x}^2) \cos \theta & \mathbf{a} \cdot \mathbf{y} \mathbf{a} \cdot \mathbf{x} (1 - \cos \theta) - \mathbf{a} \cdot \mathbf{z} \sin \theta & \mathbf{a} \cdot \mathbf{z} \mathbf{a} \cdot \mathbf{x} (1 - \cos \theta) + \mathbf{a} \cdot \mathbf{y} \sin \theta & 0 \\ \mathbf{a} \cdot \mathbf{x} \mathbf{a} \cdot \mathbf{y} (1 - \cos \theta) + \mathbf{a} \cdot \mathbf{z} \sin \theta & \mathbf{a} \cdot \mathbf{y}^2 + (1 - \mathbf{a} \cdot \mathbf{y}^2) \cos \theta & \mathbf{a} \cdot \mathbf{z} \mathbf{a} \cdot \mathbf{y} (1 - \cos \theta) - \mathbf{a} \cdot \mathbf{x} \sin \theta & 0 \\ \mathbf{a} \cdot \mathbf{x} \mathbf{a} \cdot \mathbf{z} (1 - \cos \theta) - \mathbf{a} \cdot \mathbf{y} \sin \theta & \mathbf{a} \cdot \mathbf{y} \mathbf{a} \cdot \mathbf{z} (1 - \cos \theta) - \mathbf{a} \cdot \mathbf{x} \sin \theta & \mathbf{a} \cdot \mathbf{z}^2 + (1 - \mathbf{a} \cdot \mathbf{z}^2) \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Common 3D affine transformations



input: position (\mathbf{p}), target (\mathbf{t}), and up (\mathbf{u})

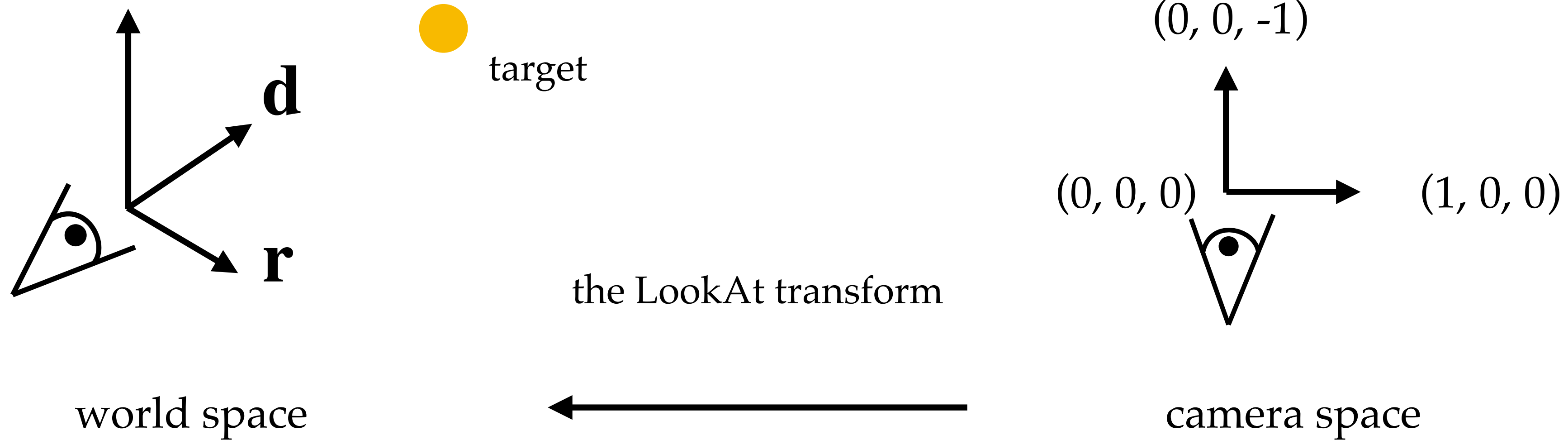
Common 3D affine transformations



input: position (\mathbf{p}), target (\mathbf{t}), and up (\mathbf{u})

$$\mathbf{d} = \text{normalize}(\mathbf{t} - \mathbf{p})$$

Common 3D affine transformations

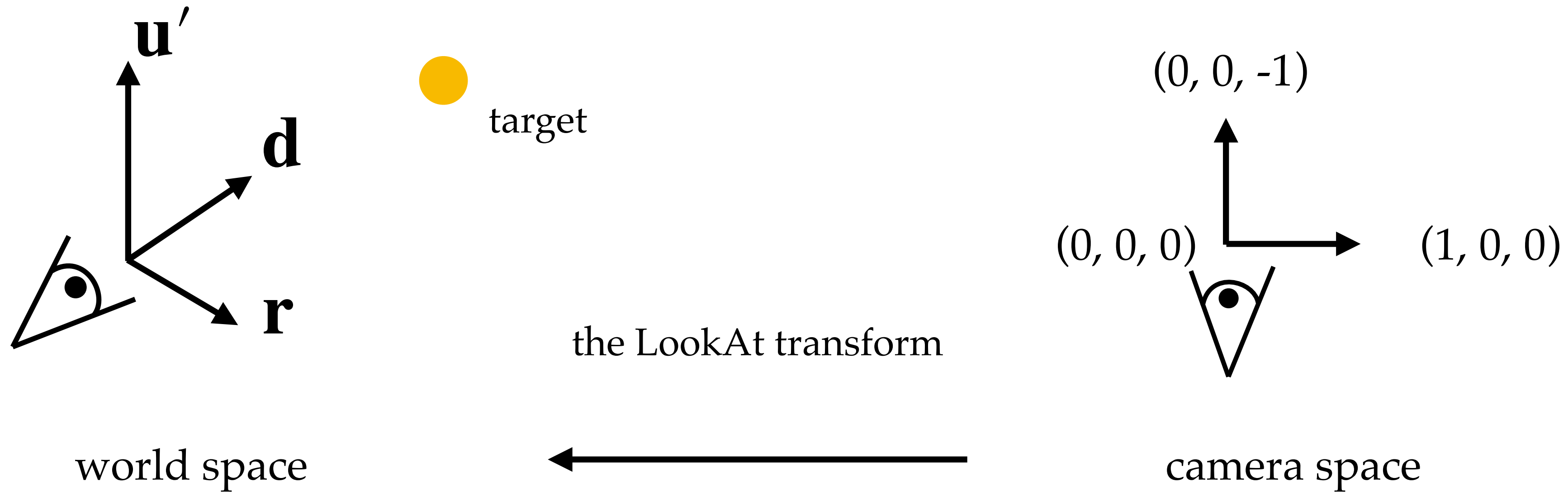


input: position (\mathbf{p}), target (\mathbf{t}), and up (\mathbf{u})

$$\mathbf{d} = \text{normalize}(\mathbf{t} - \mathbf{p})$$

$$\mathbf{r} = \text{normalize}(\mathbf{d} \times \mathbf{u})$$

Common 3D affine transformations



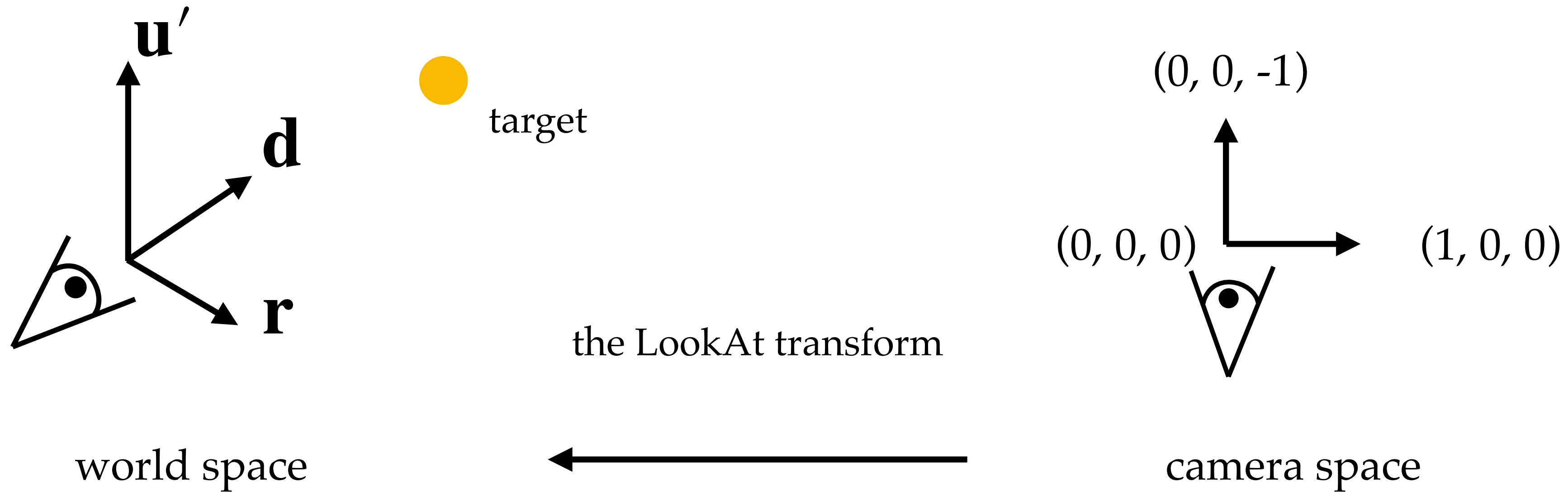
input: position (\mathbf{p}), target (\mathbf{t}), and up (\mathbf{u})

$$\mathbf{d} = \text{normalize}(\mathbf{t} - \mathbf{p})$$

$$\mathbf{r} = \text{normalize}(\mathbf{d} \times \mathbf{u})$$

$$\mathbf{u}' = \mathbf{r} \times \mathbf{d}$$

Common 3D affine transformations



input: position (\mathbf{p}), target (\mathbf{t}), and up (\mathbf{u})

$$\mathbf{d} = \text{normalize}(\mathbf{t} - \mathbf{p})$$

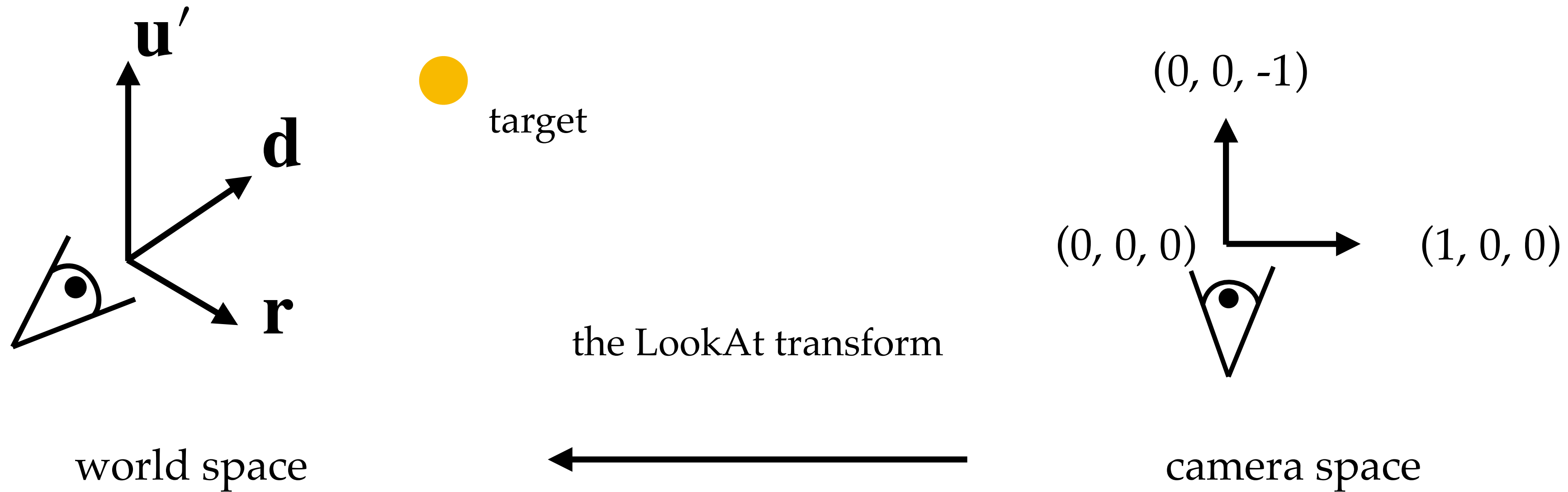
$$\mathbf{r} = \text{normalize}(\mathbf{d} \times \mathbf{u})$$

$$\mathbf{u}' = \mathbf{r} \times \mathbf{d}$$

$$\begin{bmatrix} \mathbf{r} \cdot x \\ \mathbf{r} \cdot y \\ \mathbf{r} \cdot z \\ 0 \end{bmatrix}$$

where x axis goes to

Common 3D affine transformations



input: position (\mathbf{p}), target (\mathbf{t}), and up (\mathbf{u})

$$\mathbf{d} = \text{normalize}(\mathbf{t} - \mathbf{p})$$

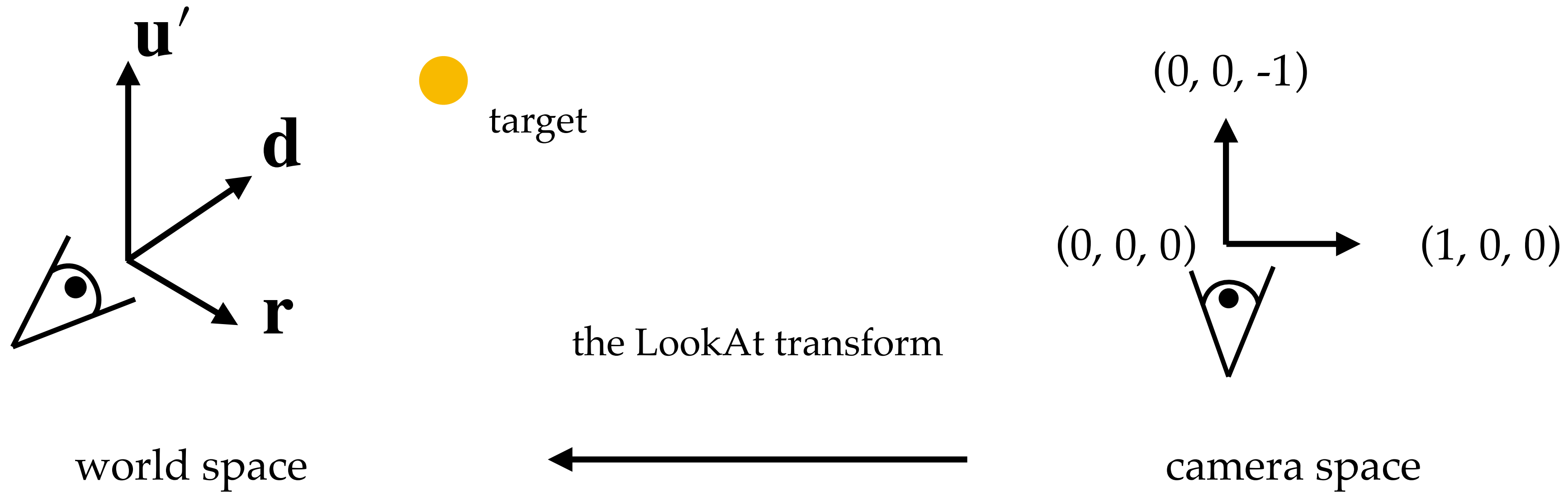
$$\mathbf{r} = \text{normalize}(\mathbf{d} \times \mathbf{u})$$

$$\mathbf{u}' = \mathbf{r} \times \mathbf{d}$$

$$\begin{bmatrix} \mathbf{r} \cdot x & \mathbf{u}' \cdot x \\ \mathbf{r} \cdot y & \mathbf{u}' \cdot y \\ \mathbf{r} \cdot z & \mathbf{u}' \cdot z \\ 0 & 0 \end{bmatrix}$$

where y axis goes to

Common 3D affine transformations



input: position (\mathbf{p}), target (\mathbf{t}), and up (\mathbf{u})

$$\mathbf{d} = \text{normalize}(\mathbf{t} - \mathbf{p})$$

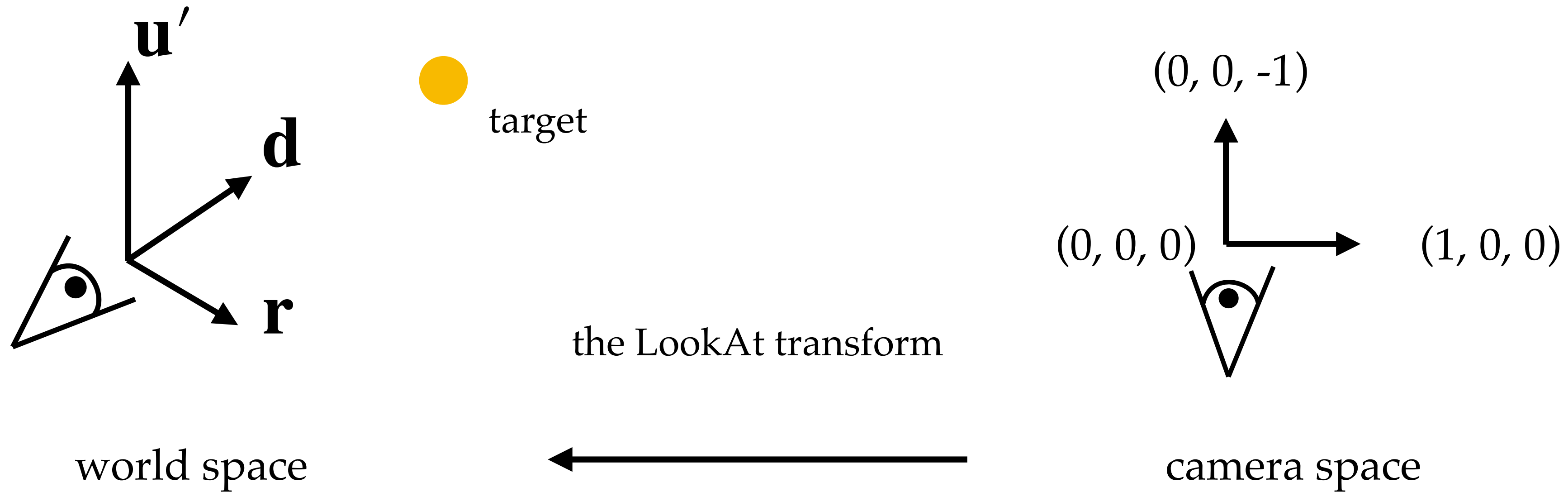
$$\mathbf{r} = \text{normalize}(\mathbf{d} \times \mathbf{u})$$

$$\mathbf{u}' = \mathbf{r} \times \mathbf{d}$$

$$\begin{bmatrix} \mathbf{r} \cdot x & \mathbf{u}' \cdot x & -\mathbf{d} \cdot x \\ \mathbf{r} \cdot y & \mathbf{u}' \cdot y & -\mathbf{d} \cdot y \\ \mathbf{r} \cdot z & \mathbf{u}' \cdot z & -\mathbf{d} \cdot z \\ 0 & 0 & 0 \end{bmatrix}$$

where z axis goes to

Common 3D affine transformations



input: position (\mathbf{p}), target (\mathbf{t}), and up (\mathbf{u})

$$\mathbf{d} = \text{normalize}(\mathbf{t} - \mathbf{p})$$

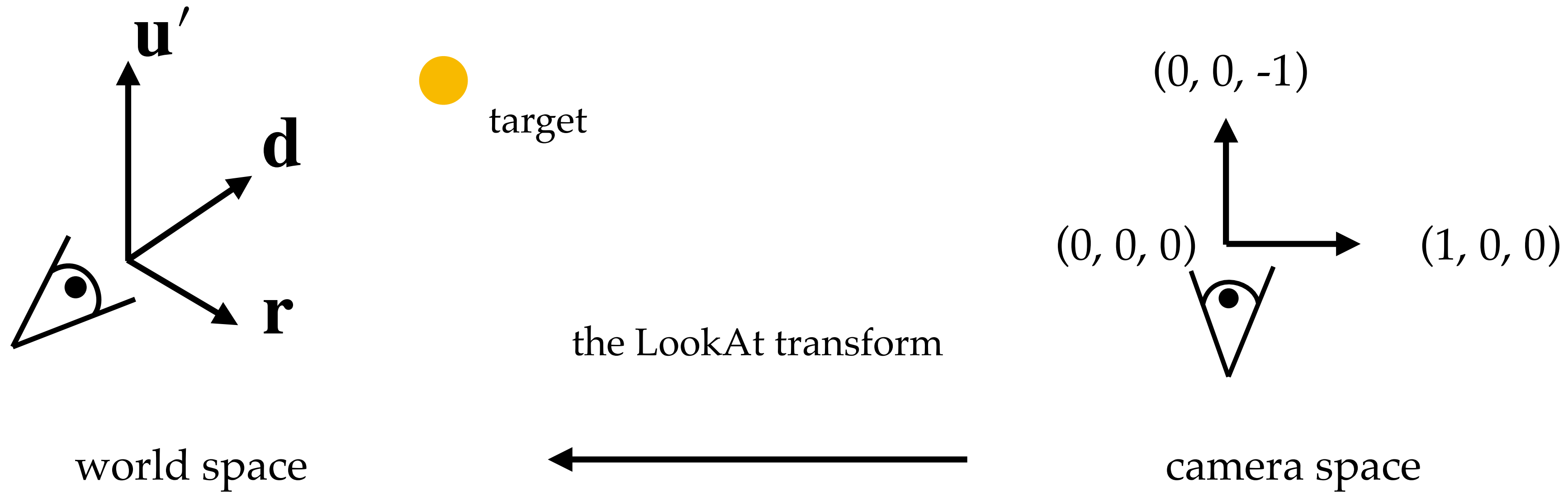
$$\mathbf{r} = \text{normalize}(\mathbf{d} \times \mathbf{u})$$

$$\mathbf{u}' = \mathbf{r} \times \mathbf{d}$$

$$\begin{bmatrix} \mathbf{r} \cdot x & \mathbf{u}' \cdot x & -\mathbf{d} \cdot x & \mathbf{p} \cdot x \\ \mathbf{r} \cdot y & \mathbf{u}' \cdot y & -\mathbf{d} \cdot y & \mathbf{p} \cdot y \\ \mathbf{r} \cdot z & \mathbf{u}' \cdot z & -\mathbf{d} \cdot z & \mathbf{p} \cdot z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

translation

Common 3D affine transformations



input: position (\mathbf{p}), target (\mathbf{t}), and up (\mathbf{u})

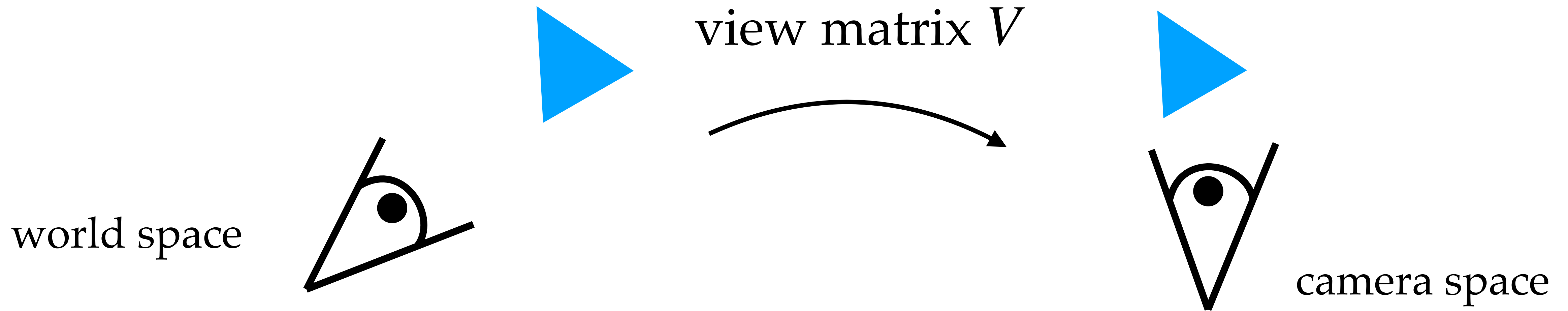
$$\mathbf{d} = \text{normalize}(\mathbf{t} - \mathbf{p})$$

$$\mathbf{r} = \text{normalize}(\mathbf{d} \times \mathbf{u})$$

$$\mathbf{u}' = \mathbf{r} \times \mathbf{d}$$

$$\begin{bmatrix} \mathbf{r} \cdot x & \mathbf{u}' \cdot x & -\mathbf{d} \cdot x & \mathbf{p} \cdot x \\ \mathbf{r} \cdot y & \mathbf{u}' \cdot y & -\mathbf{d} \cdot y & \mathbf{p} \cdot y \\ \mathbf{r} \cdot z & \mathbf{u}' \cdot z & -\mathbf{d} \cdot z & \mathbf{p} \cdot z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

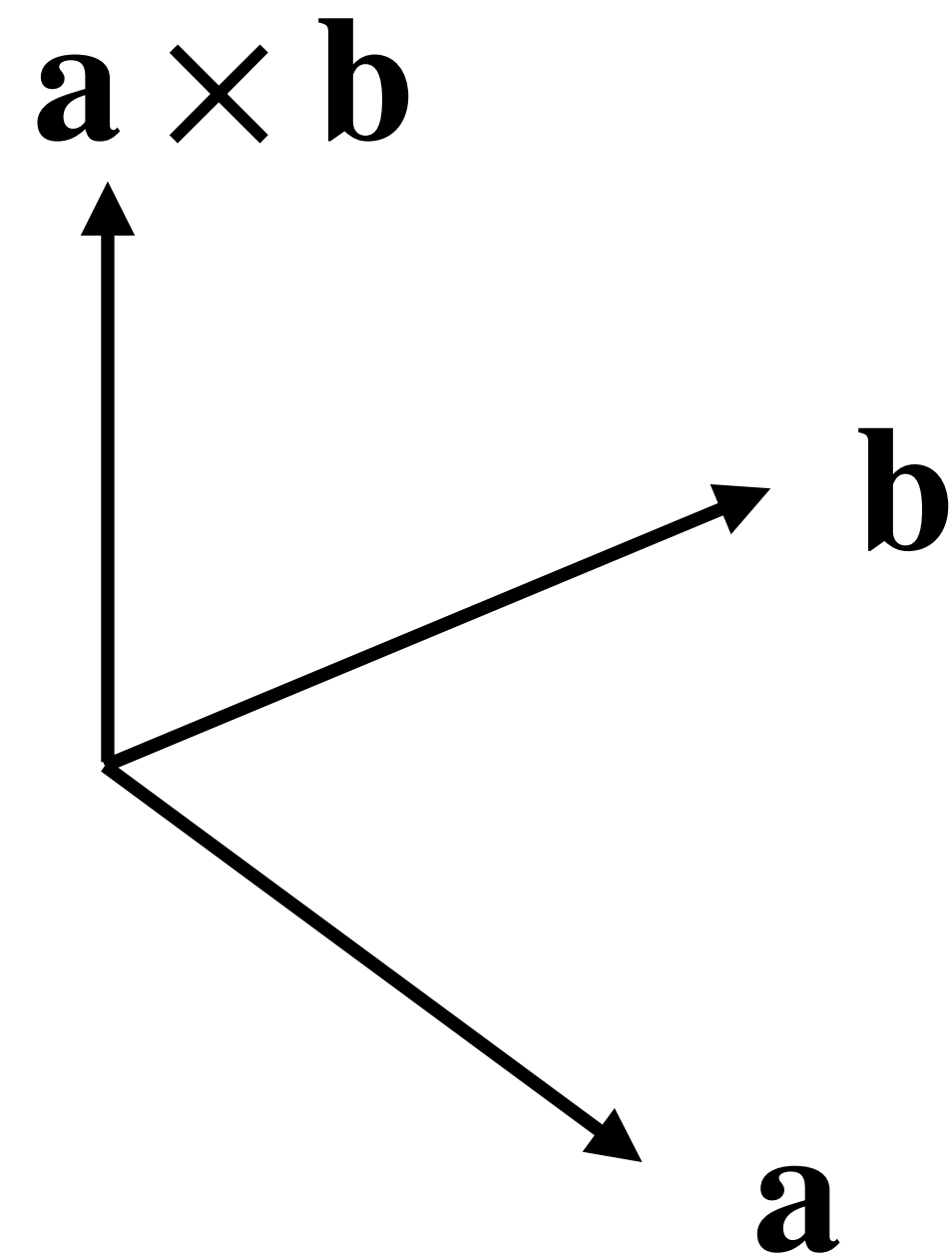
Common 3D affine transformations



we use the convention that the view matrix is the inverse of the LookAt matrix

$$V = \begin{bmatrix} \mathbf{r} \cdot x & \mathbf{u}' \cdot x & -\mathbf{d} \cdot x & \mathbf{p} \cdot x \\ \mathbf{r} \cdot y & \mathbf{u}' \cdot y & -\mathbf{d} \cdot y & \mathbf{p} \cdot y \\ \mathbf{r} \cdot z & \mathbf{u}' \cdot z & -\mathbf{d} \cdot z & \mathbf{p} \cdot z \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1}$$

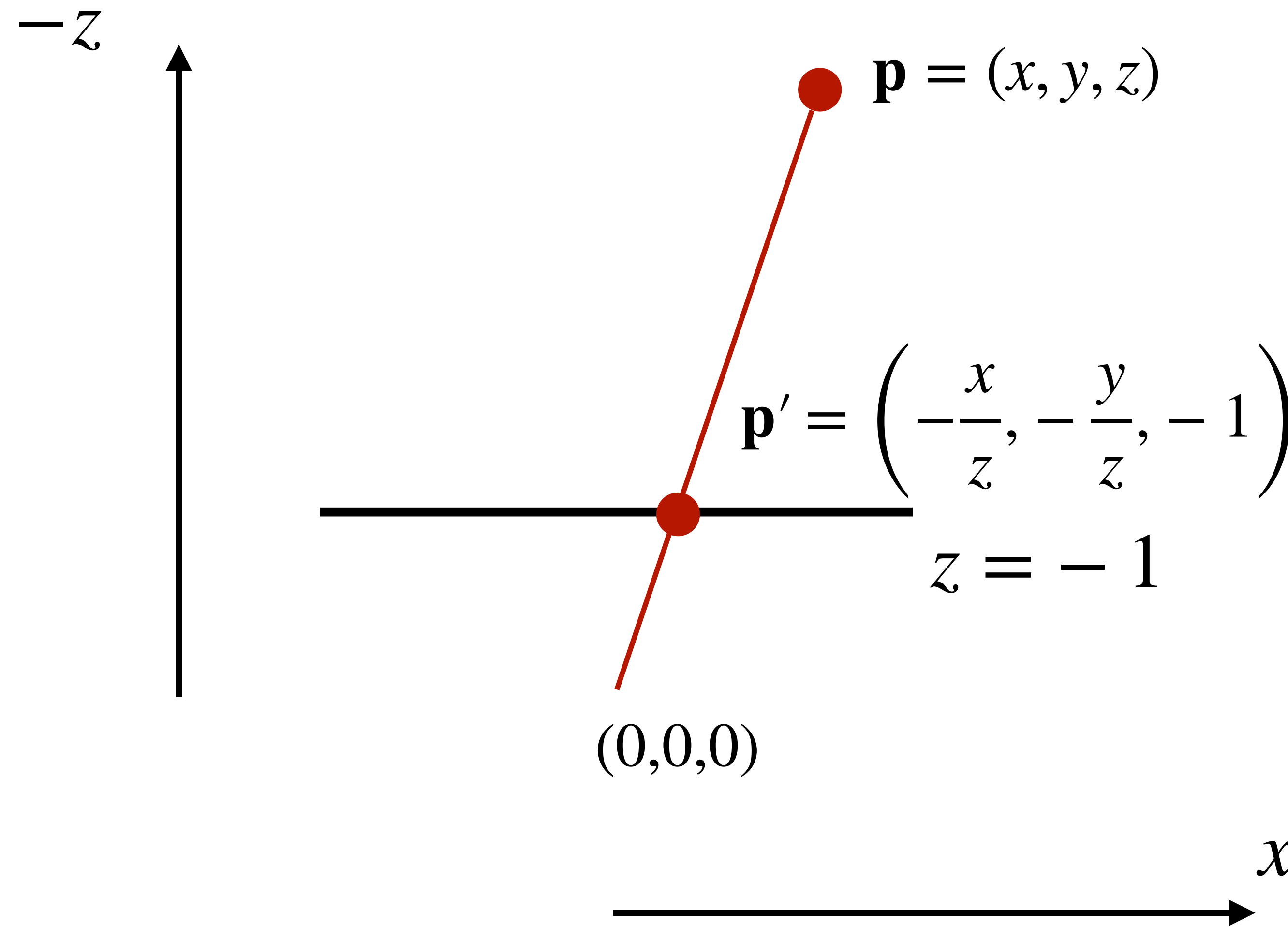
Common 3D affine transformations



cross product

$$\mathbf{a} \times \mathbf{b} = \begin{bmatrix} 0 & -\mathbf{a} \cdot \mathbf{z} & \mathbf{a} \cdot \mathbf{y} \\ \mathbf{a} \cdot \mathbf{z} & 0 & -\mathbf{a} \cdot \mathbf{x} \\ -\mathbf{a} \cdot \mathbf{y} & \mathbf{a} \cdot \mathbf{x} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{b} \cdot \mathbf{x} \\ \mathbf{b} \cdot \mathbf{y} \\ \mathbf{b} \cdot \mathbf{z} \end{bmatrix}$$

Is it possible to use a matrix to represent perspective projection?



Yes, if we introduce homogeneous coordinates

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

points

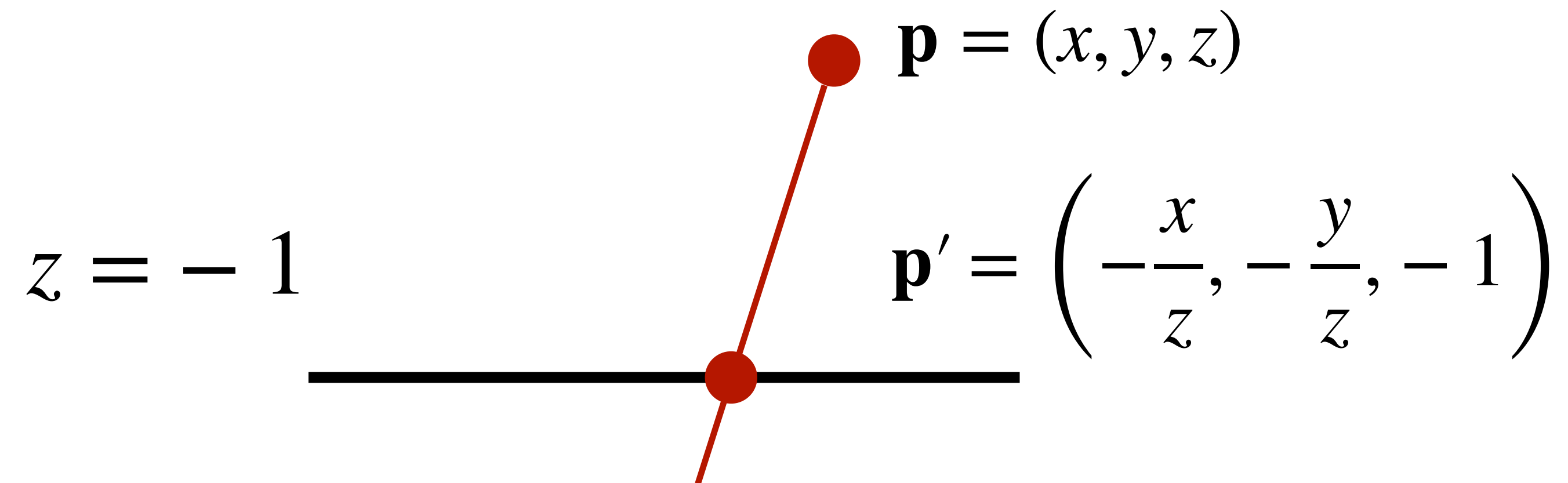
$$\begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix}$$

vectors

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} \frac{x}{w} \\ \frac{y}{w} \\ \frac{z}{w} \\ 1 \end{bmatrix} = \begin{bmatrix} 100x \\ 100y \\ 100z \\ 100w \end{bmatrix} \quad w \neq 0$$

points with
homogeneous coordinates

Yes, if we introduce homogeneous coordinates



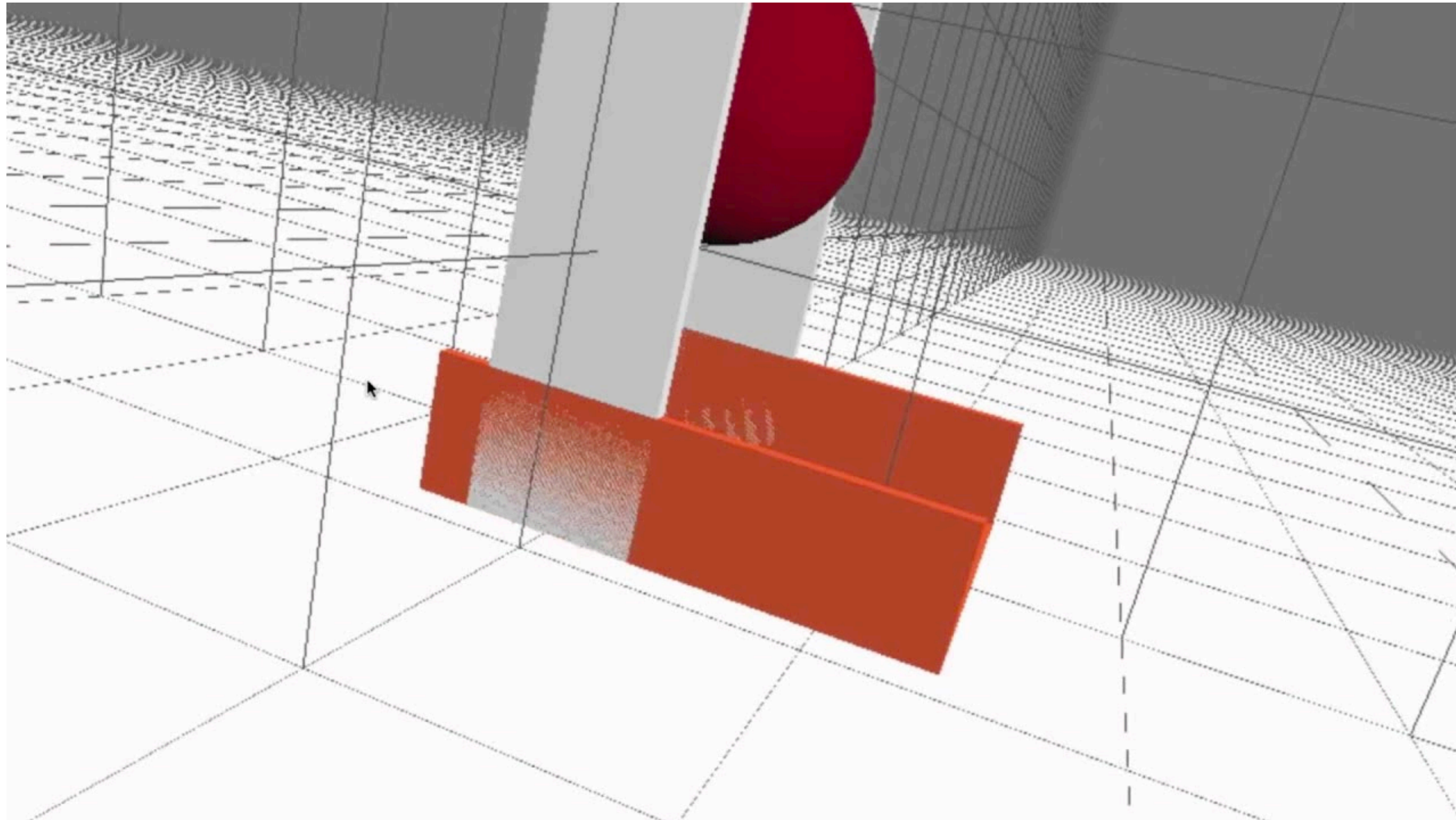
$$\begin{bmatrix} \frac{\mathbf{p}' \cdot x}{\mathbf{p}' \cdot w} \\ \frac{\mathbf{p}' \cdot y}{\mathbf{p}' \cdot w} \\ \frac{\mathbf{p}' \cdot z}{\mathbf{p}' \cdot w} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{p}' \cdot x \\ \mathbf{p}' \cdot y \\ \mathbf{p}' \cdot z \\ \mathbf{p}' \cdot w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p} \cdot x \\ \mathbf{p} \cdot y \\ \mathbf{p} \cdot z \\ 1 \end{bmatrix}$$

General perspective transformation

$$\begin{bmatrix} \frac{\mathbf{p}' \cdot x}{\mathbf{p}' \cdot w} \\ \frac{\mathbf{p}' \cdot y}{\mathbf{p}' \cdot w} \\ \frac{\mathbf{p}' \cdot z}{\mathbf{p}' \cdot w} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{p}' \cdot x \\ \mathbf{p}' \cdot y \\ \mathbf{p}' \cdot z \\ \mathbf{p}' \cdot w \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} \mathbf{p} \cdot x \\ \mathbf{p} \cdot y \\ \mathbf{p} \cdot z \\ 1 \end{bmatrix}$$

Z-fighting

setting up a far clipping plane and mapping Z to Z_{near}/Z_{far} helps resolving Z-fighting



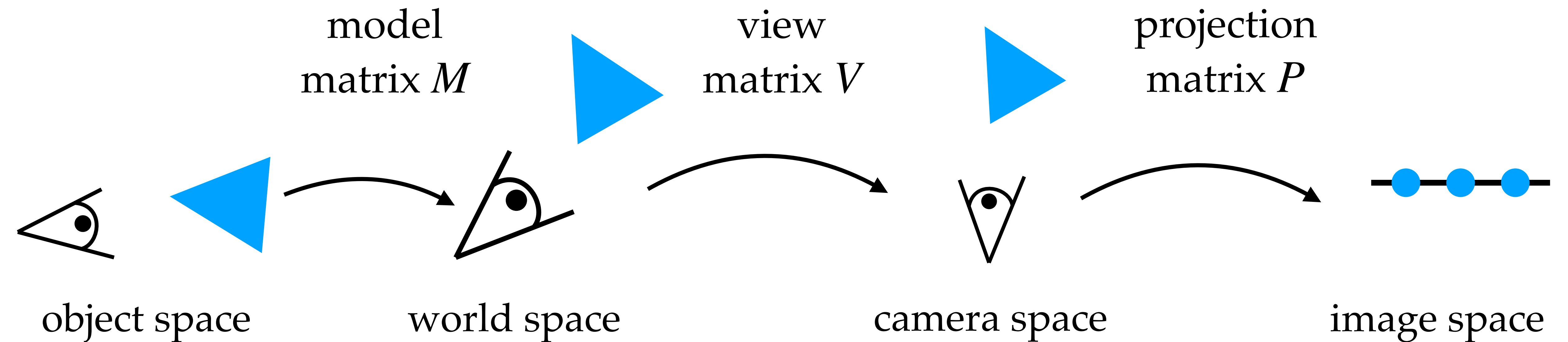
The projection matrix often used in rasterizers

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{f}{f-n} & -\frac{fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

n: near clipping Z
f: far clipping Z

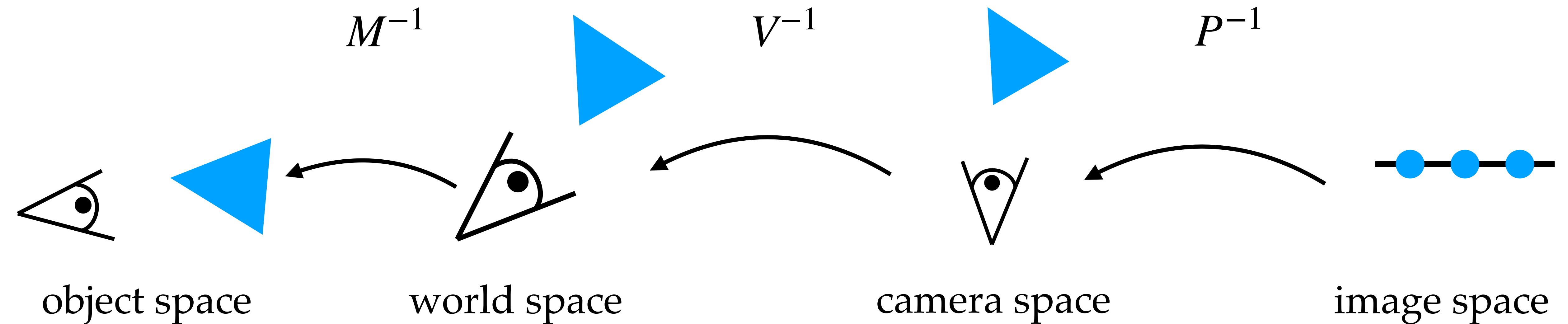
this row maps Z from [n, f] to [0, 1]

The model-view-projection (MVP) matrices for rasterization



(in OpenGL, the projection transforms points to the "clip space" waiting for clipping)

In ray tracing, we usually use the inverses of these matrices



Do homogeneous coordinates
preserve linear combination of points?

$$0.5 \begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix} + 0.5 \begin{bmatrix} 6 \\ 4 \\ 2 \\ 2 \end{bmatrix} \quad ? \quad 0.5 \begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix} + 0.5 \begin{bmatrix} 3 \\ 2 \\ 1 \\ 1 \end{bmatrix}$$

Homogeneous coordinates **do not** preserve linear combination of points

normalize the last coordinate to "1" or "0" before you do something with your homogeneous coordinates vectors!

$$0.5 \begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix} + 0.5 \begin{bmatrix} 6 \\ 4 \\ 2 \\ 2 \end{bmatrix} \neq 0.5 \begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix} + 0.5 \begin{bmatrix} 3 \\ 2 \\ 1 \\ 1 \end{bmatrix}$$
$$= \begin{bmatrix} \frac{7}{2} \\ 3 \\ \frac{5}{2} \\ \frac{3}{2} \end{bmatrix} \qquad = \begin{bmatrix} 2 \\ 2 \\ 2 \\ 1 \end{bmatrix}$$

An alternative to homogeneous coordinates: Grassmann coordinates

assign each point with a “mass” m

$$\begin{bmatrix} mx \\ my \\ mz \\ m \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

see this paper for a really nice motivation and explanation

On the Algebraic and Geometric Foundations of Computer Graphics

RON GOLDMAN
Rice University

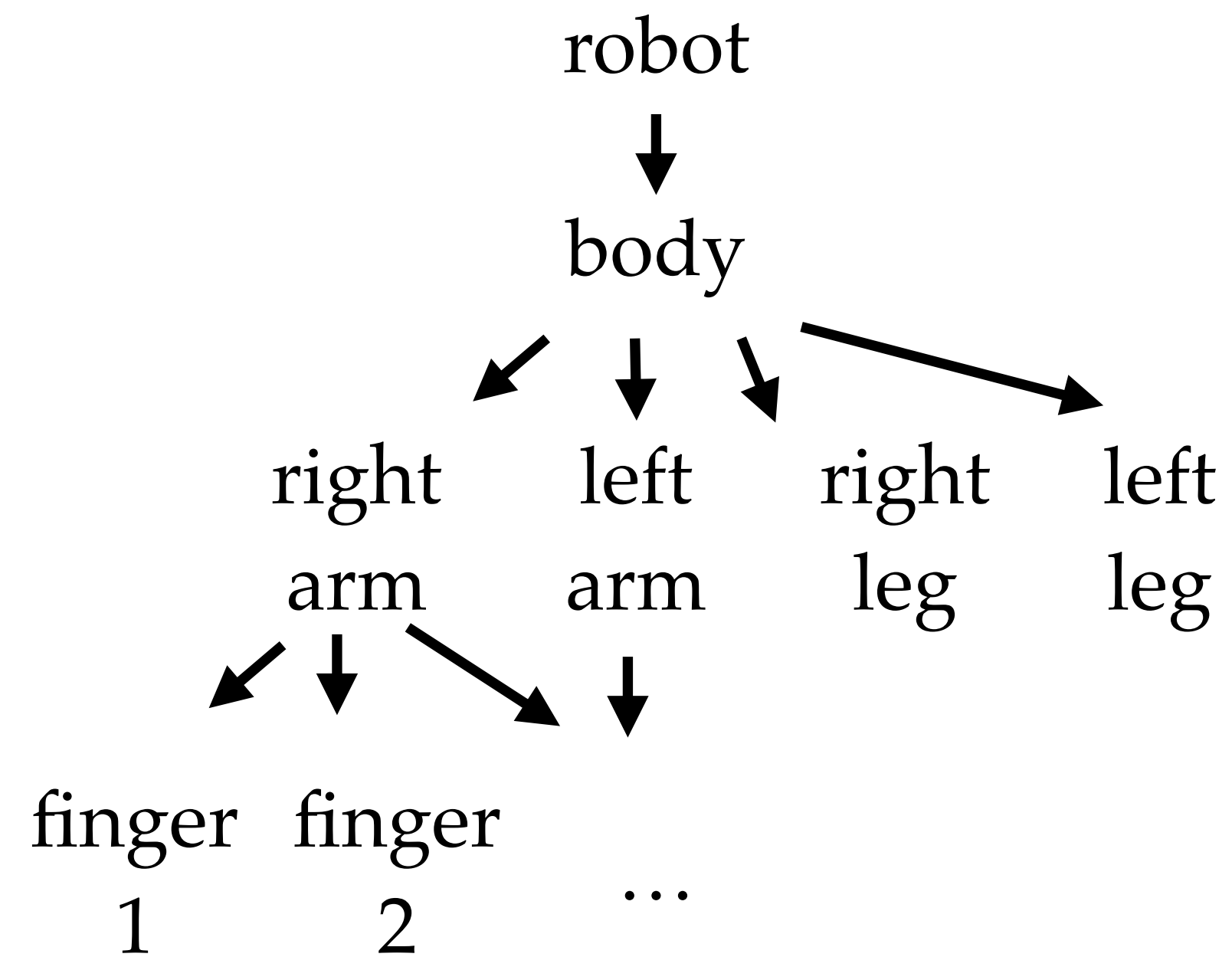
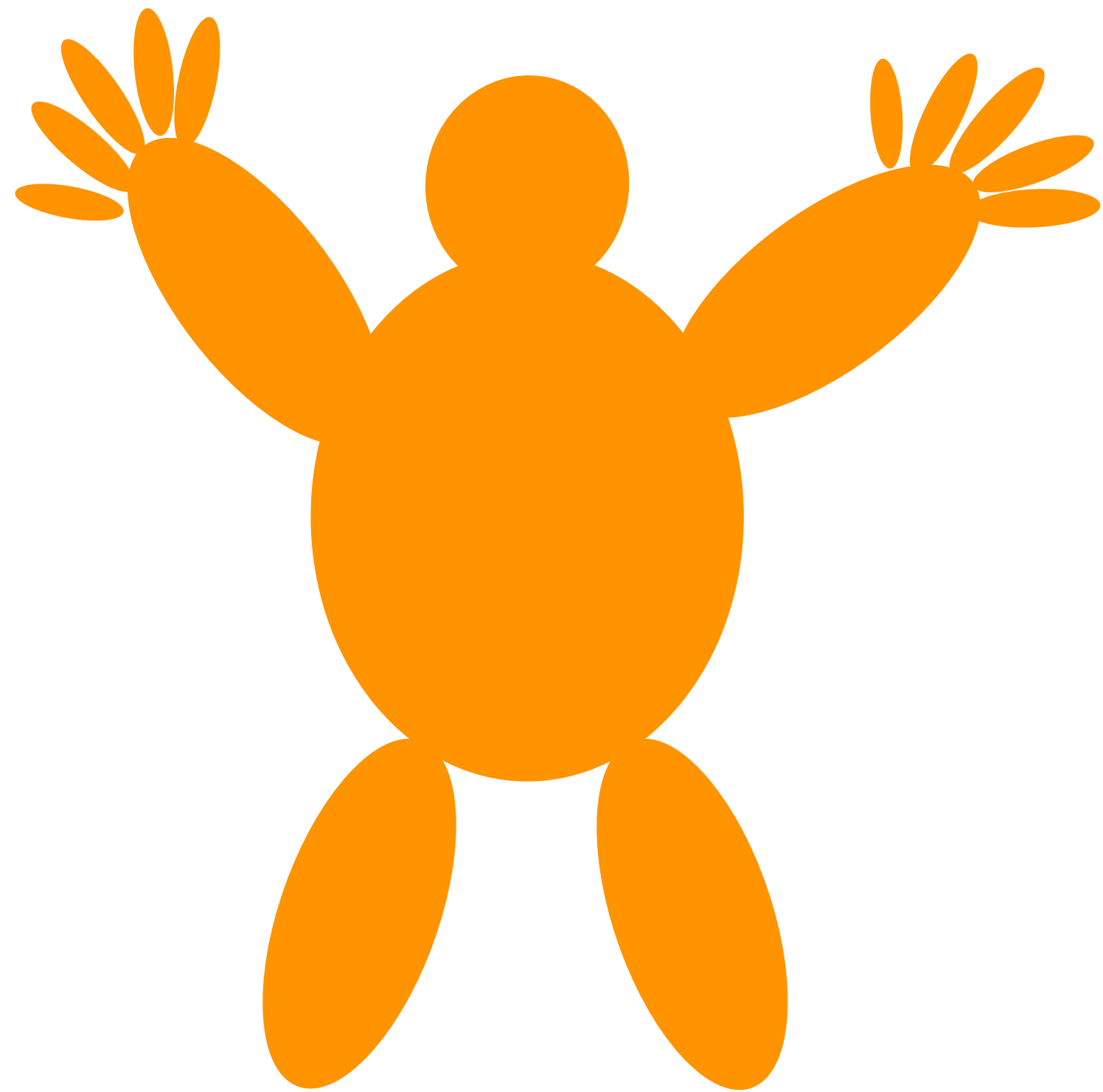
Today’s computer graphics is ostensibly based upon insights from projective geometry and computations on homogeneous coordinates. Paradoxically, however, projective spaces and homogeneous coordinates are incompatible with much of the algebra and a good deal of the geometry currently in actual use in computer graphics. To bridge this gulf between theory and practice, Grassmann spaces are proposed here as an alternative to projective spaces. We establish that unlike projective spaces, Grassmann spaces do support all the algebra and geometry needed for contemporary computer graphics. We then go on to explain how to exploit this algebra and geometry for a variety of applications, both old and new, including the graphics pipeline, shading algorithms, texture maps, and overcrown surfaces.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*hierarchy and geometric transformations*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Grassmann space, homogeneous coordinates, mass-points, projective space

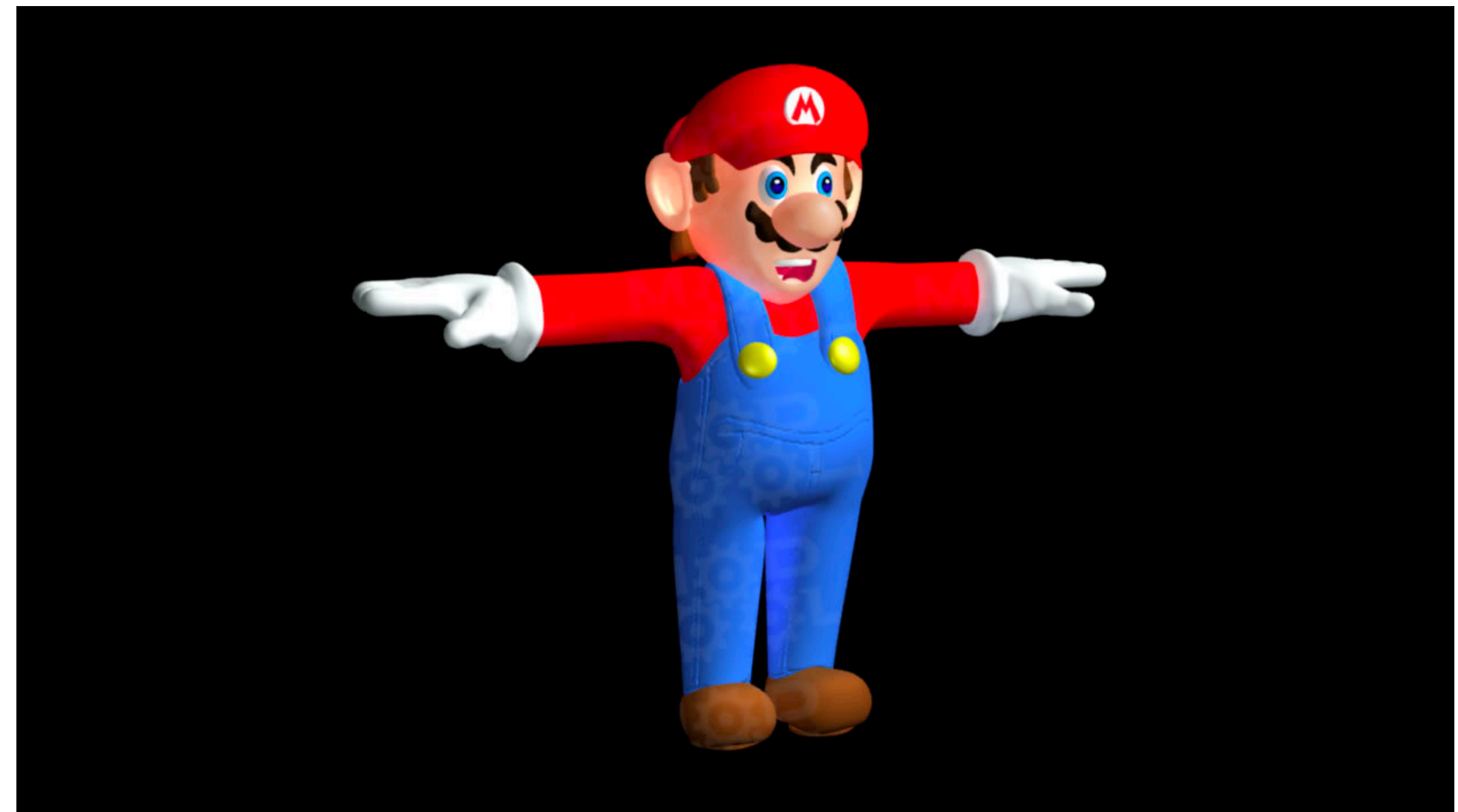
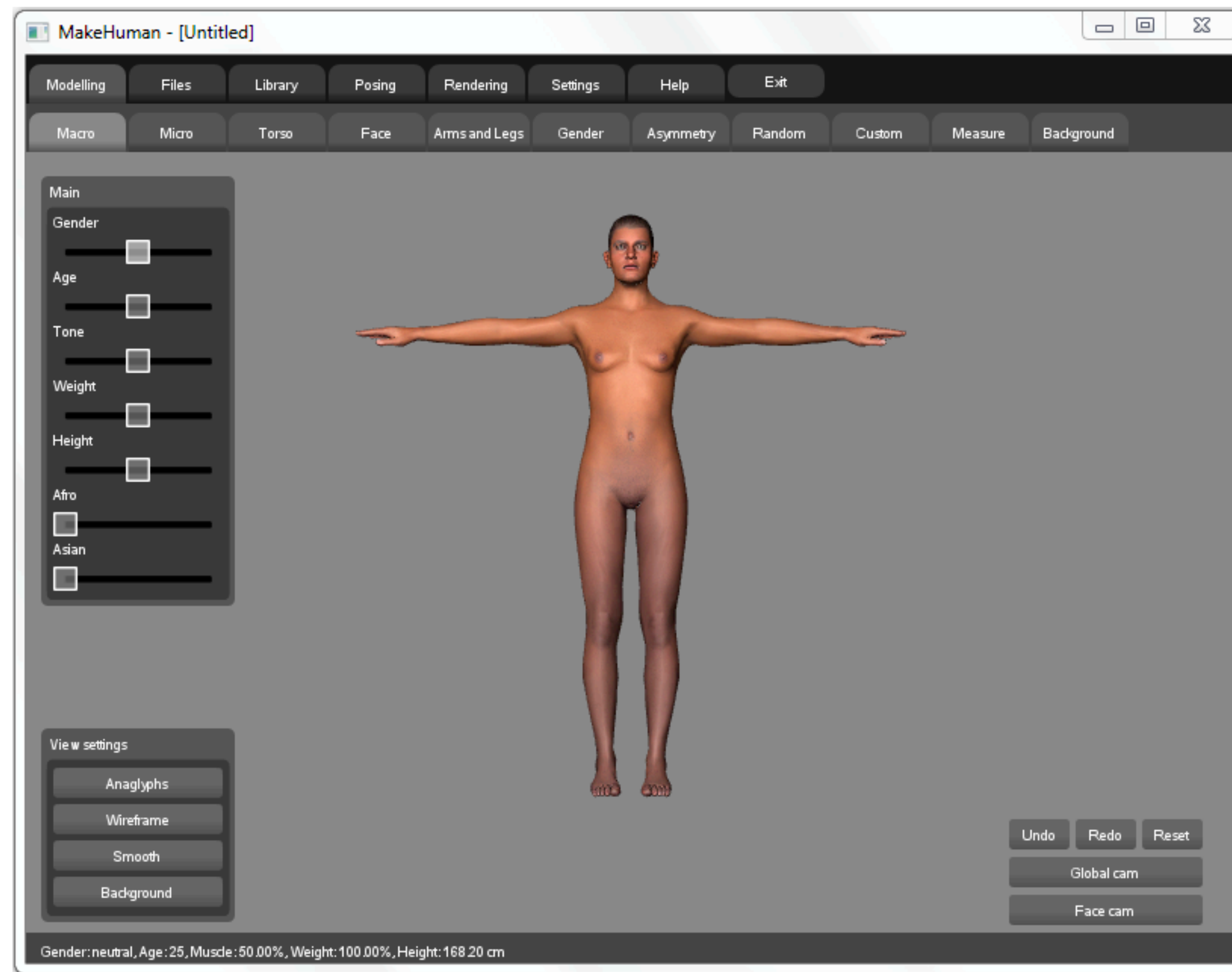
Application of 3D transformation: scene graph



also see "Entity Component System"

https://en.wikipedia.org/wiki/Entity_component_system

T-pose in 3D models



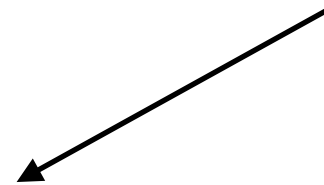
<https://en.wikipedia.org/wiki/T-pose>

Application of 3D transformation: instancing



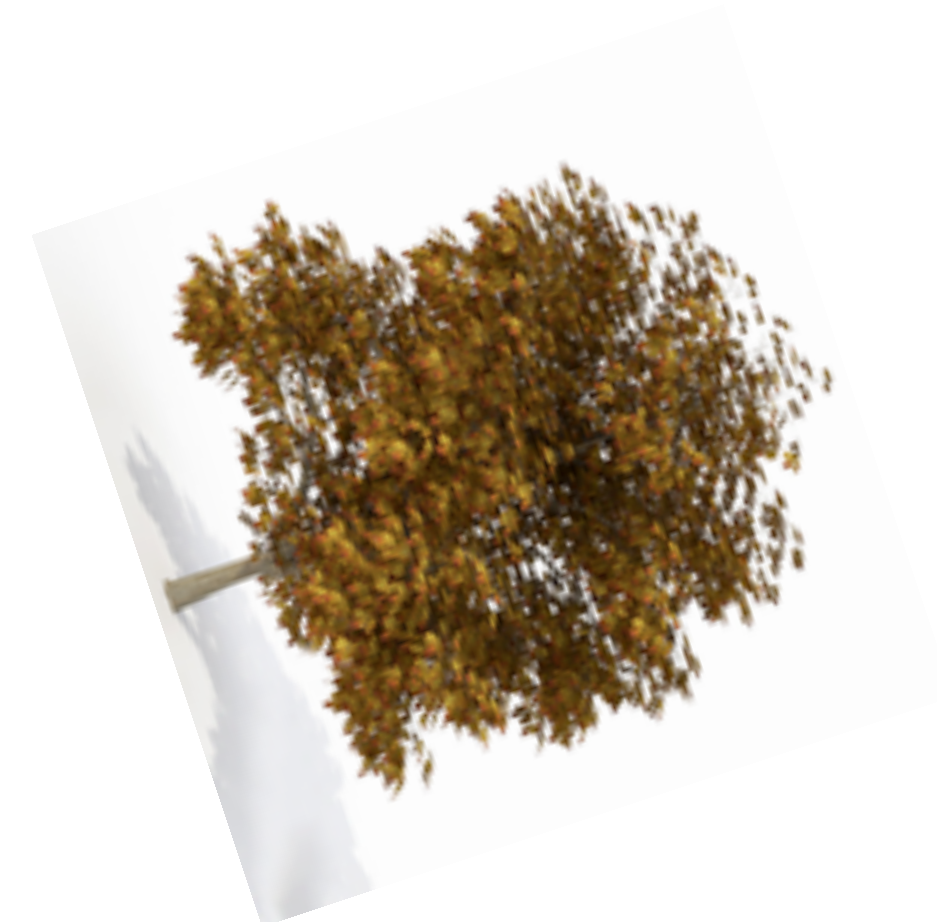
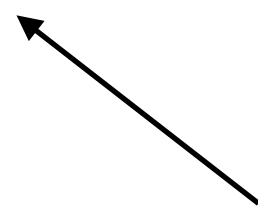
object 1

transformation matrix
pointer to the triangle mesh



object 2

transformation matrix
pointer to the triangle mesh



Application of 3D transformation: instancing

from Solid Angle Arnold



Data	Usage
Polygon meshes	4,015 MB
Indices	2,031 MB
Vertices	1,266 MB
Normals	441 MB
UV coordinates	180 MB
BVHs	2,528 MB
Node overhead	800 MB
Instance overhead	456 MB
Total	8,784 MB

Fig. 5. The production version of the *Elysium* station (left, middle) comprised 5 trillion triangles thanks to the use of multi-level instancing of .ass files and Arnold's space-efficient geometry structures. The incomplete version of the scene we have, with no textures and only 4 trillion (144 million unique) triangles, requires about 9GB of RAM to render with the current Arnold version (5.0.2). A partial memory-usage breakdown is shown on the right. We suspect that with textures (studio used 4GB of texture cache) and the additional missing data (max. 2GB), it would require at most 15GB. ©2013 CTMG, courtesy of Whiskytree.

Next: 3D rotation



https://en.wikipedia.org/wiki/File:Spinning_Dancer.gif