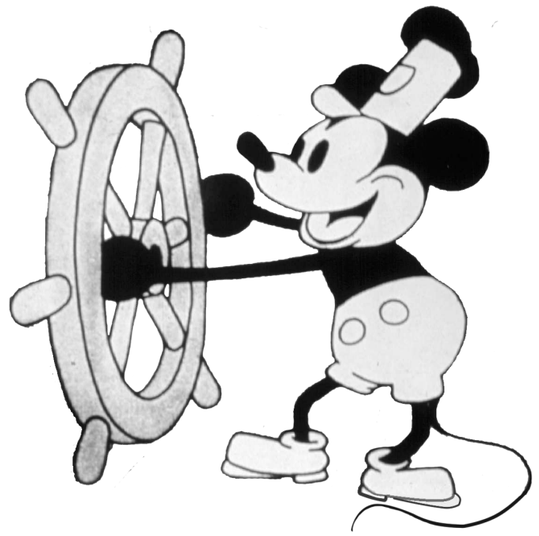
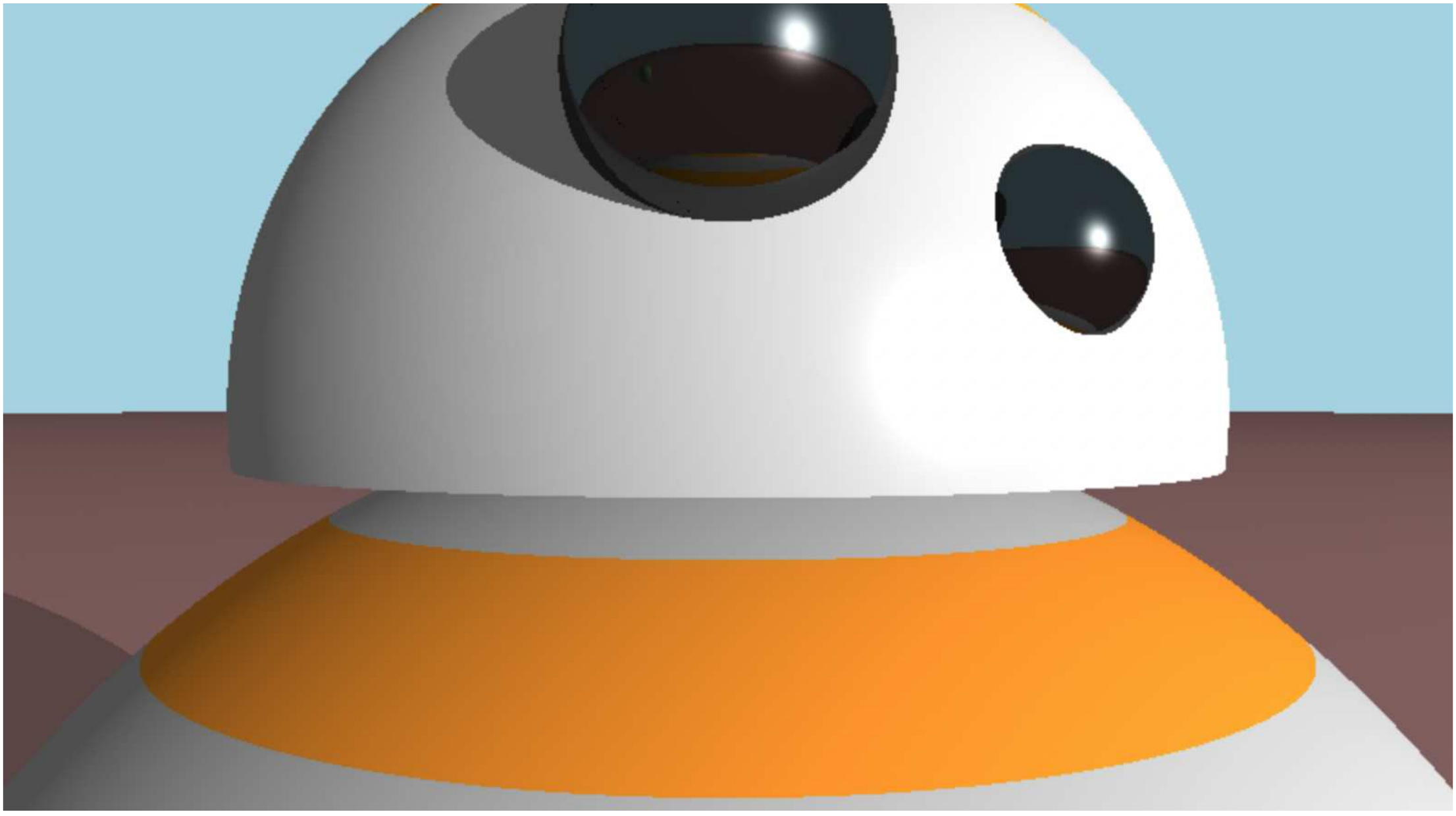


CSCI 461: Computer Graphics

Middlebury College, Fall 2025

Lecture 05: Scenes

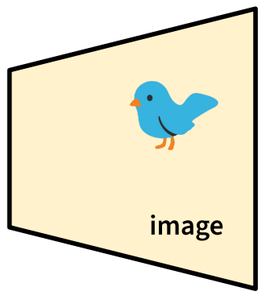
We have always been looking down the $-z$ axis!

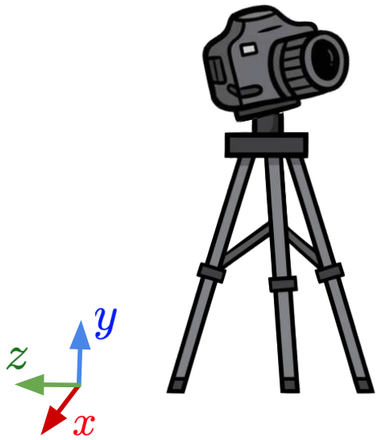


And the eye has always been at the origin.

By the end of today's lecture, you will be able to:

- set up a camera to look in more **general directions**,
- use **homogeneous coordinates** to represent all our 3d transformations with a 4x4 matrix,
- compound rotations, scalings and translations of objects in your scene into a **model matrix**,
- transform the normal vector of objects in your scene using a **normal matrix**.





First: a note about the "transpose."

$$\mathbf{M} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

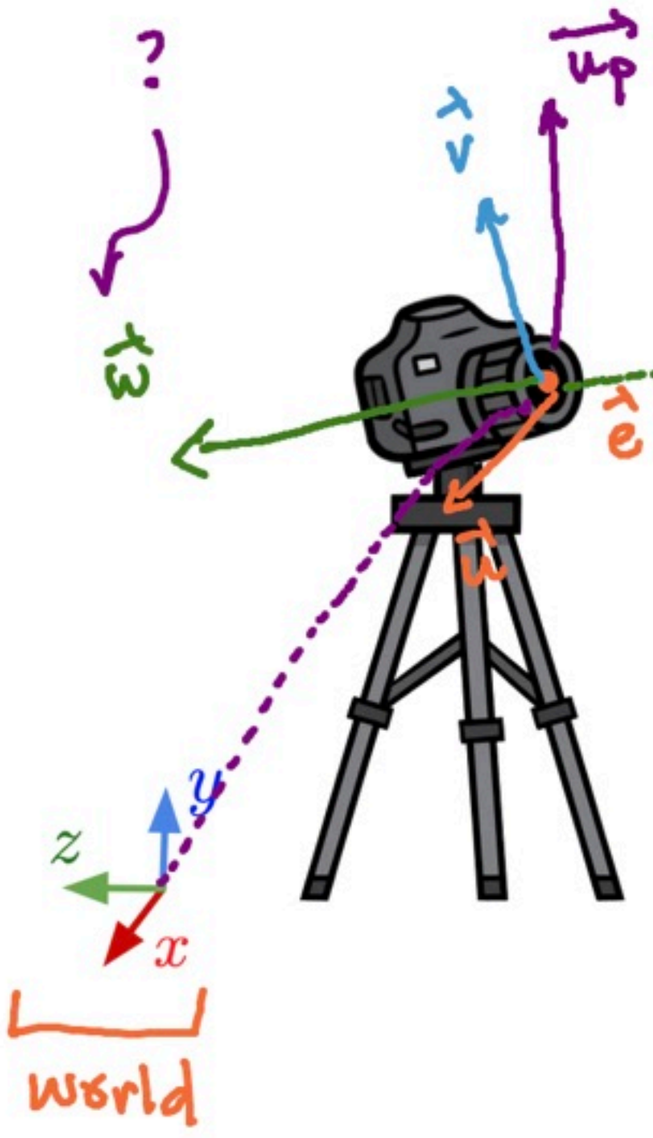
M^T

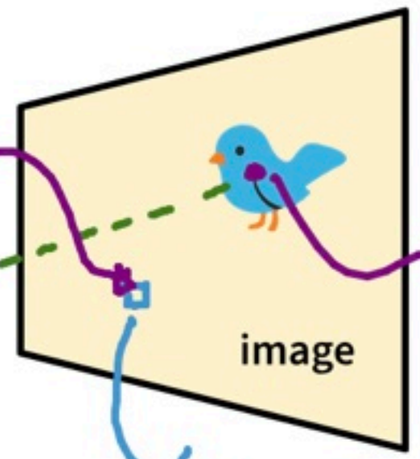
$$\mathbf{M}^T = \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix}$$

We can also interpret the dot product $\vec{u} \cdot v$ of column vectors $\vec{u}$ and $\vec{v}$ as the multiplication $\vec{u}^T\vec{v}$:

$$\vec{u} \cdot \vec{v} = \vec{u}^T\vec{v} = \begin{bmatrix} u_x & u_y & u_z \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = u_x v_x + u_y v_y + u_z v_z.$$

Pointing the camera at some point to look at.





gaze $\vec{g} = \vec{a} - \vec{e}$

point to "look at"

(x_c, y_c, z_c)

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

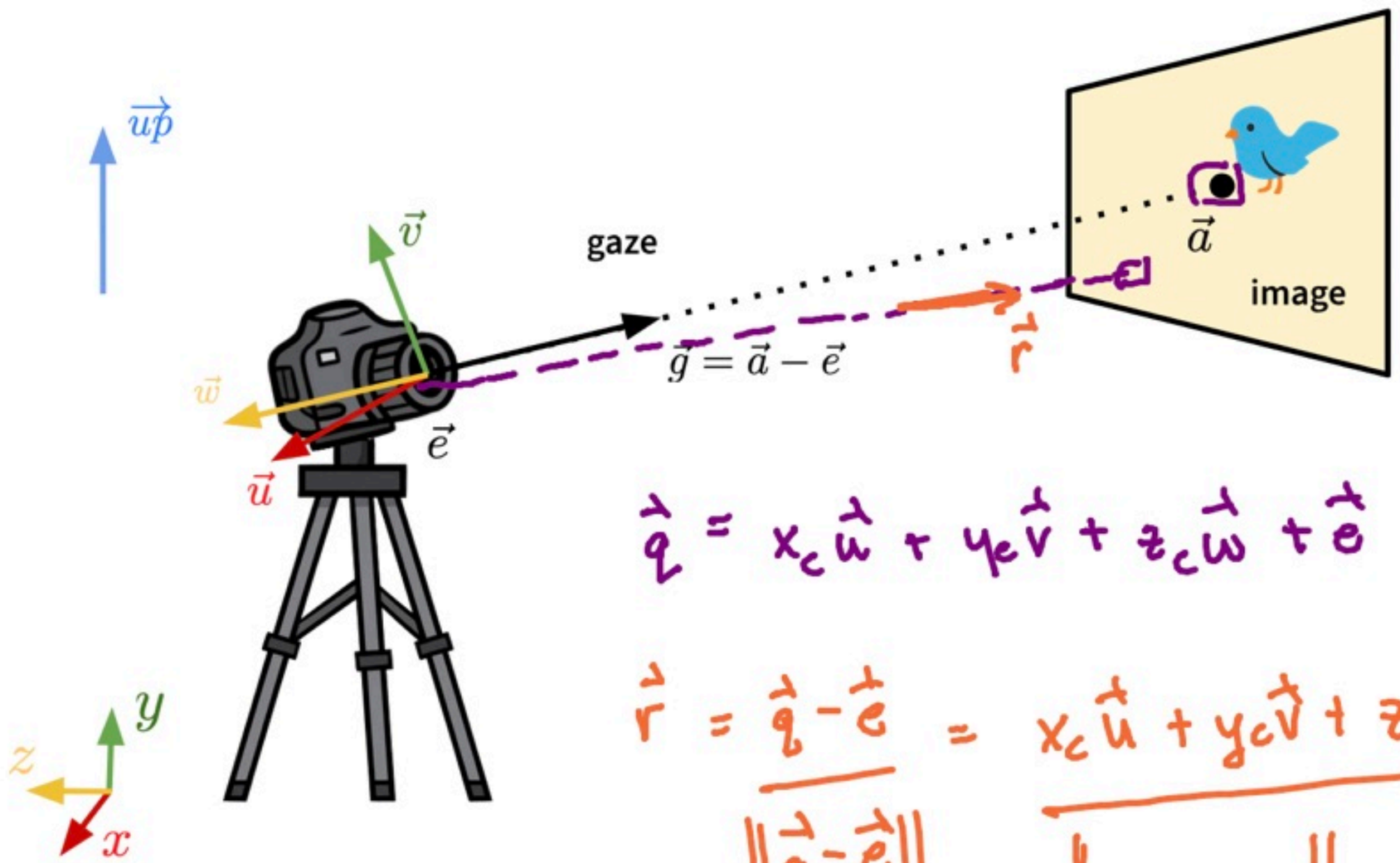
$\vec{v} = \vec{w} \times \vec{u}$

↑ adding 'c' subscript to imply it's measured wrt the camera

$\vec{u} = \frac{\vec{up} \times \vec{w}}{\|\vec{up} \times \vec{w}\|}$

$\vec{q} = x_c \vec{u} + y_c \vec{v} + z_c \vec{w} + \vec{e}$

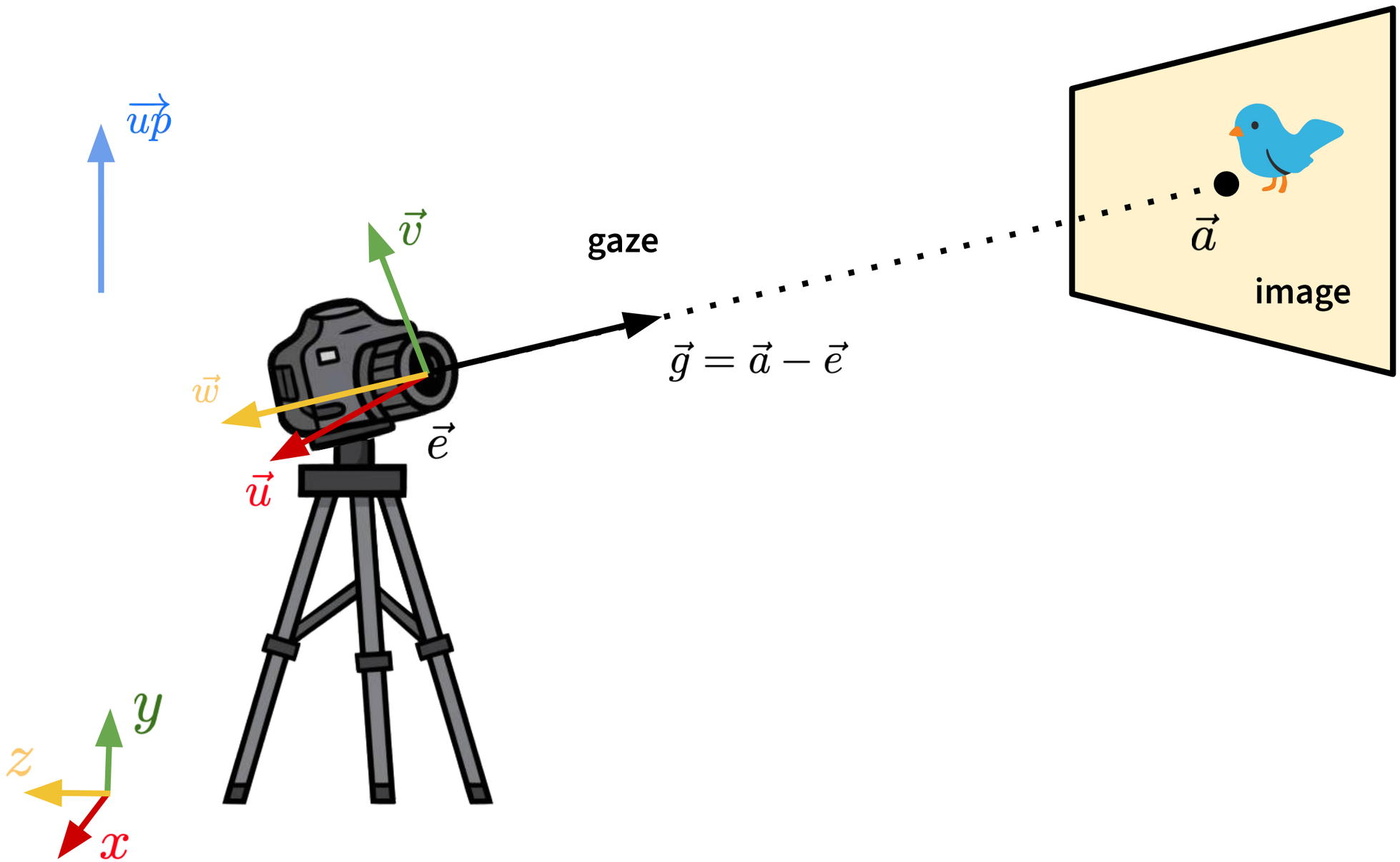
Pointing the camera at some point to look at.



$$\vec{q} = x_c\vec{u} + y_c\vec{v} + z_c\vec{w} + \vec{e}$$

$$\vec{r} = \frac{\vec{q}-\vec{e}}{\|\vec{q}-\vec{e}\|} = \frac{x_c\vec{u} + y_c\vec{v} + z_c\vec{w}}{\| \dots \|}$$

What is the ray direction then?

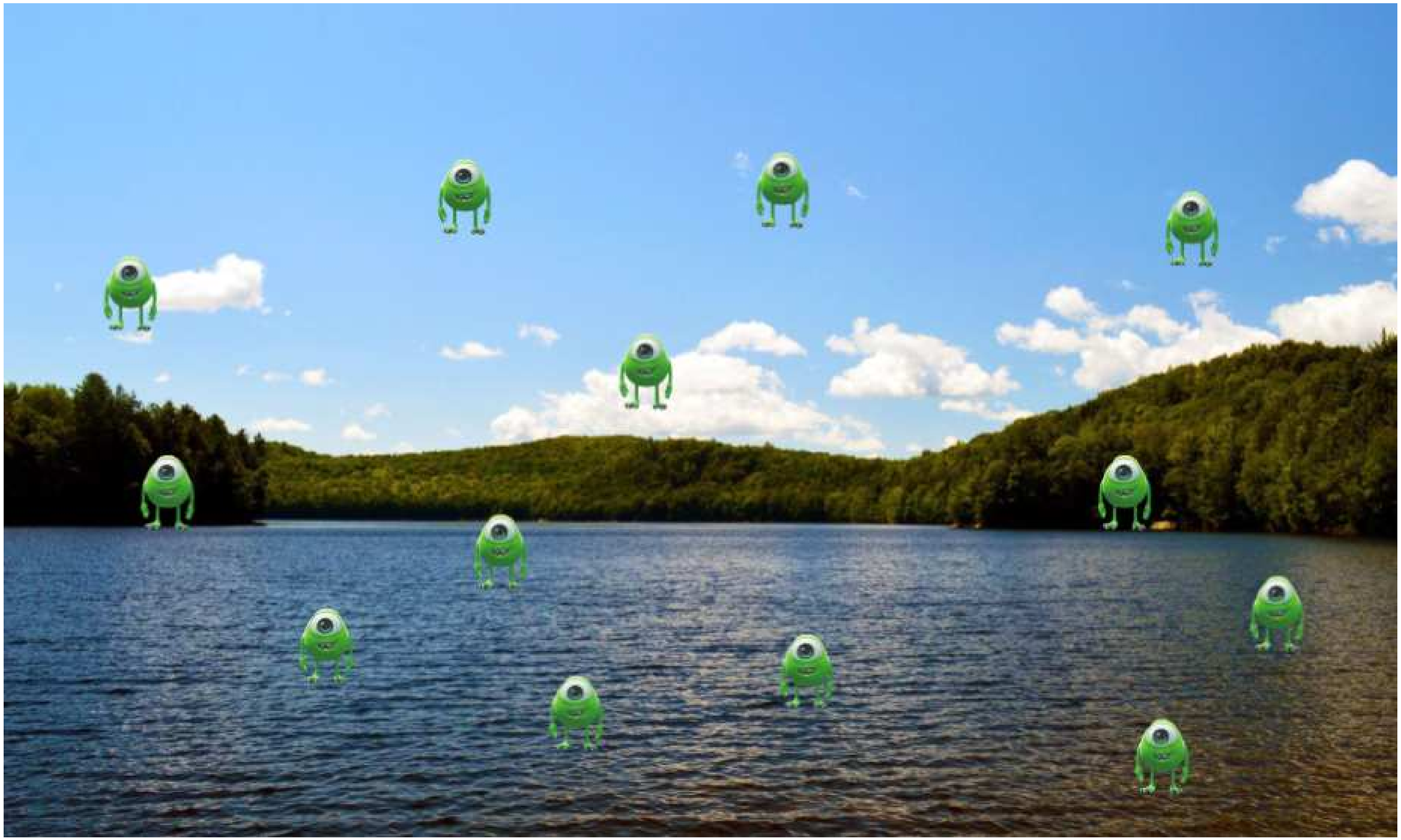


How should we place a model in the scene?

How should we place a model in the scene?

How should we place a model in the scene?

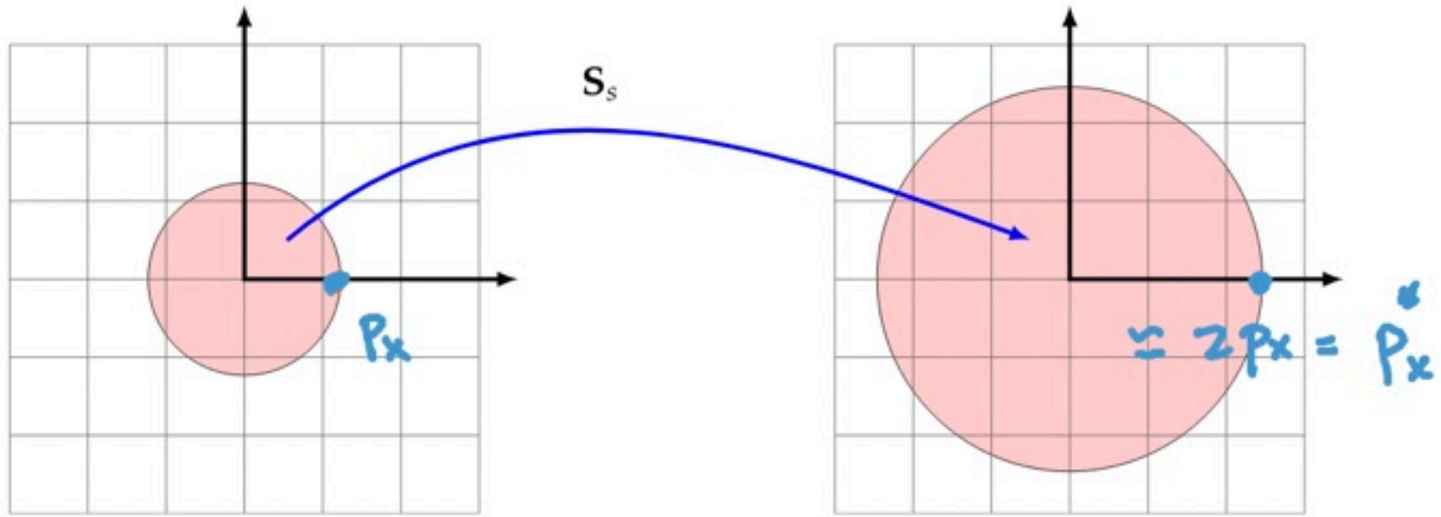
How should we place a model in the scene?



How do we animate and/or interact with our model?

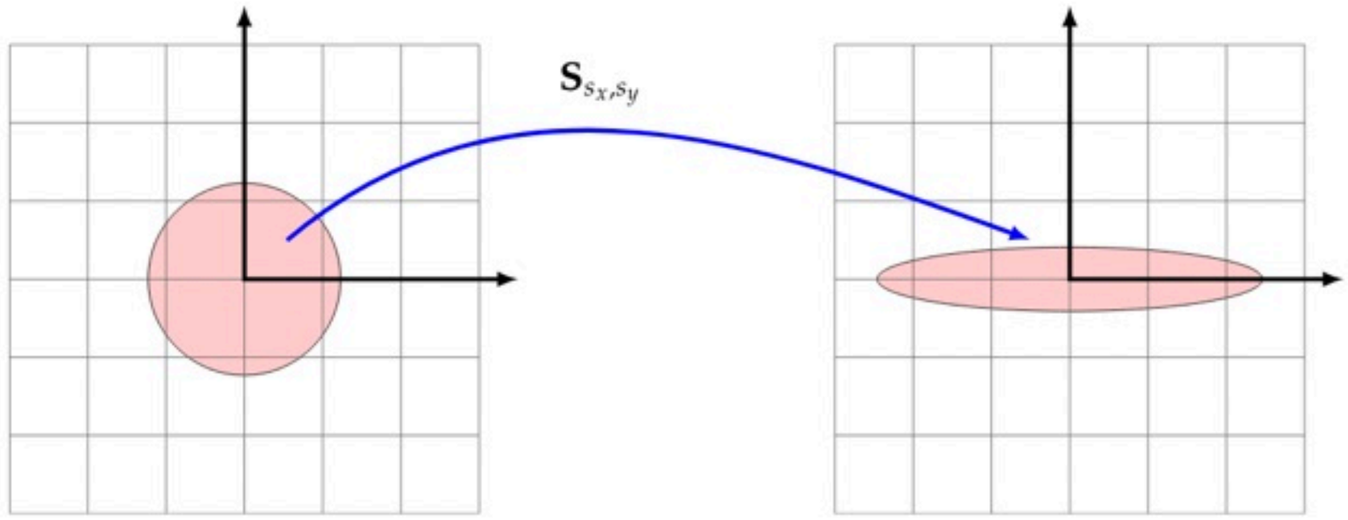
Transformations! We'll look at scaling, rotations and translations.

Scaling transformation stretches points in each dimension.



$$\vec{p}^* = \begin{bmatrix} x^* \\ y^* \\ z^* \end{bmatrix} = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & s \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \mathbf{S}_s \vec{p} = \begin{bmatrix} s\,p_x \\ s\,p_y \\ s\,p_z \end{bmatrix}$$

Scaling transformation stretches points in each dimension (can be non-uniform).



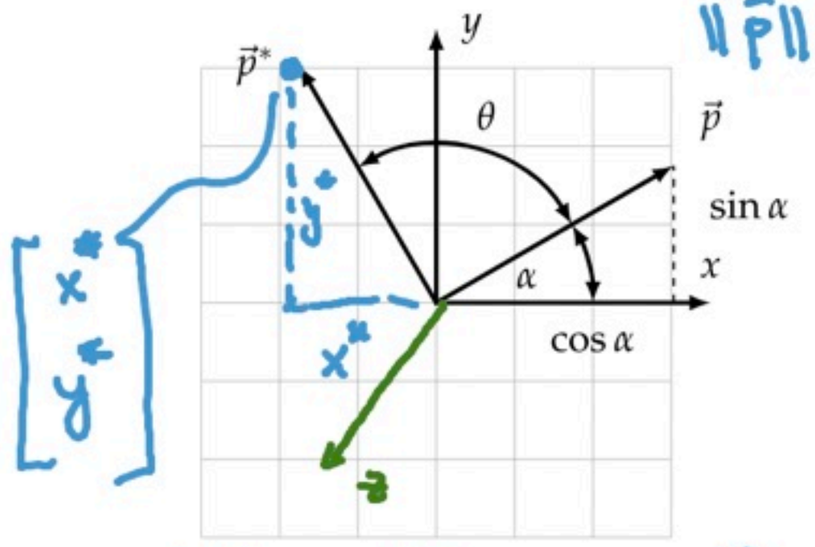
$$\vec{p}^* = \begin{bmatrix} x^* \\ y^* \\ z^* \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \mathbf{S}_s \vec{p} = \begin{bmatrix} s_x p_x \\ s_y p_y \\ s_z p_z \end{bmatrix}$$

Rotation matrix rotates vectors by an angle about some axis.

$x^* = \ell\cos(\theta + \alpha)$

$y^* = \ell\sin(\theta + \alpha)$

$\|\vec{p}\| = \ell$



$$x^* = \ell\cos(\alpha+\theta) = \ell\cos\alpha\cos\theta - \ell\sin\alpha\sin\theta = x\cos\theta - y\sin\theta, \quad \simeq x^*$$
$$y^* = \ell\sin(\alpha+\theta) = \ell\sin\alpha\cos\theta + \ell\cos\alpha\sin\theta = x\sin\theta + y\cos\theta. \quad \simeq y^*$$

$$\vec{p}^* = \begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \mathbf{R}^{2d}_{\theta,z}\vec{p}.$$

Rotation matrices are *orthogonal*.

$$\mathbf{R}_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$\cos(-\theta) = \cos\theta$

$\sin(-\theta) = -\sin\theta$

$$R_{-\theta} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} = R_\theta^T$$

$$R^{-1} = \frac{1}{\cos^2\theta - (-\sin^2\theta)} \begin{bmatrix} \cos\theta & +\sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

$\cos^2\theta - (-\sin^2\theta) = 1$

When you substitute $-\theta$ into $\mathbf{R}$, what do you obtain?

A. $\mathbf{R}^T$
B. $\mathbf{R}^{-1}$
C. Both (A) and (B).
D. None of the above.

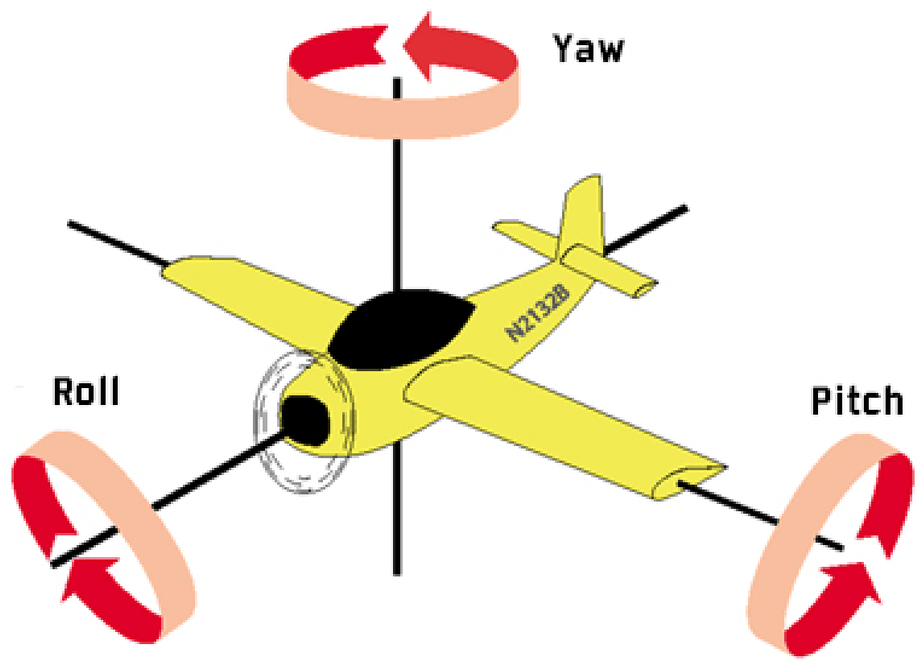
Vote using **Reactions** tab on course webpage.

Rotations about the x, y and z-axes.

$$\mathbf{R}_{\theta,x} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$\mathbf{R}_{\theta,y} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$\mathbf{R}_{\theta,z} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Model not centered at the origin?
First translate model to origin, then rotate, then translate back.

How to represent a translation (shift) with a matrix?

$$\vec{p}' = \vec{p} + \vec{t} = \begin{bmatrix} p_x + t_x \\ p_y + t_y \end{bmatrix} \qquad \vec{t} = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$ 2d

$$\vec{p}' = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix} \vec{p}$$ how? we can't

$$\vec{p}^{\,h} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$$ 3d

use homogeneous coordinates

$$\begin{bmatrix} x' \\ y' \\ \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix} = \begin{bmatrix} p_x + t_x \\ p_y + t_y \\ 1 \end{bmatrix}$$

Homogeneous coordinates is a trick to combine everything into a single matrix.

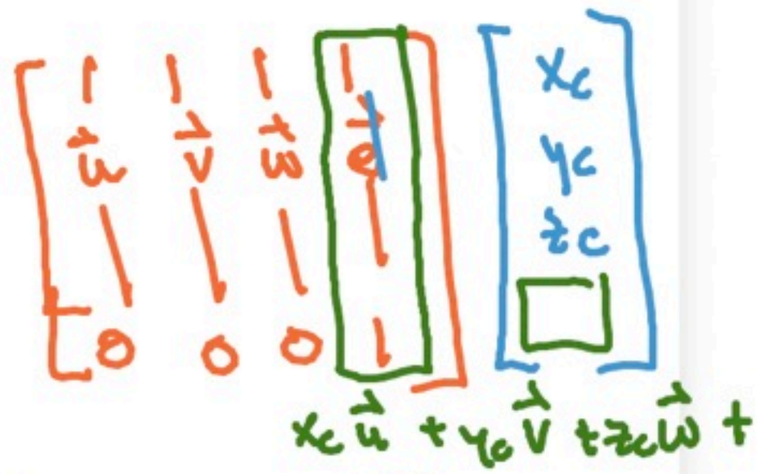
Main idea: introduce new coordinate equal to 1 for points.

$$\vec{p}^{\,h} = \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} \qquad \vec{q}^{\,h} = \begin{bmatrix} x_q \\ y_q \\ z_q \\ 1 \end{bmatrix}$$

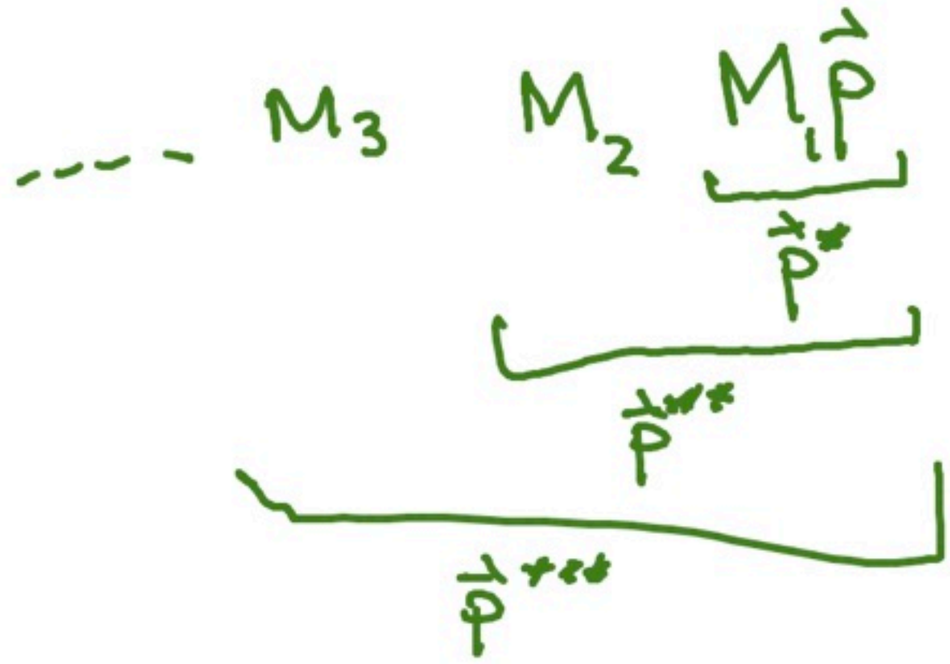
$$\vec{v}^{\,h} = \vec{q}^{\,h} - \vec{p}^{\,h} = \begin{bmatrix} x_q - x_p \\ y_q - y_p \\ z_q - z_p \\ 0 \end{bmatrix}$$

Another way to get the 3d pixel coordinates (using `glMatrix`).

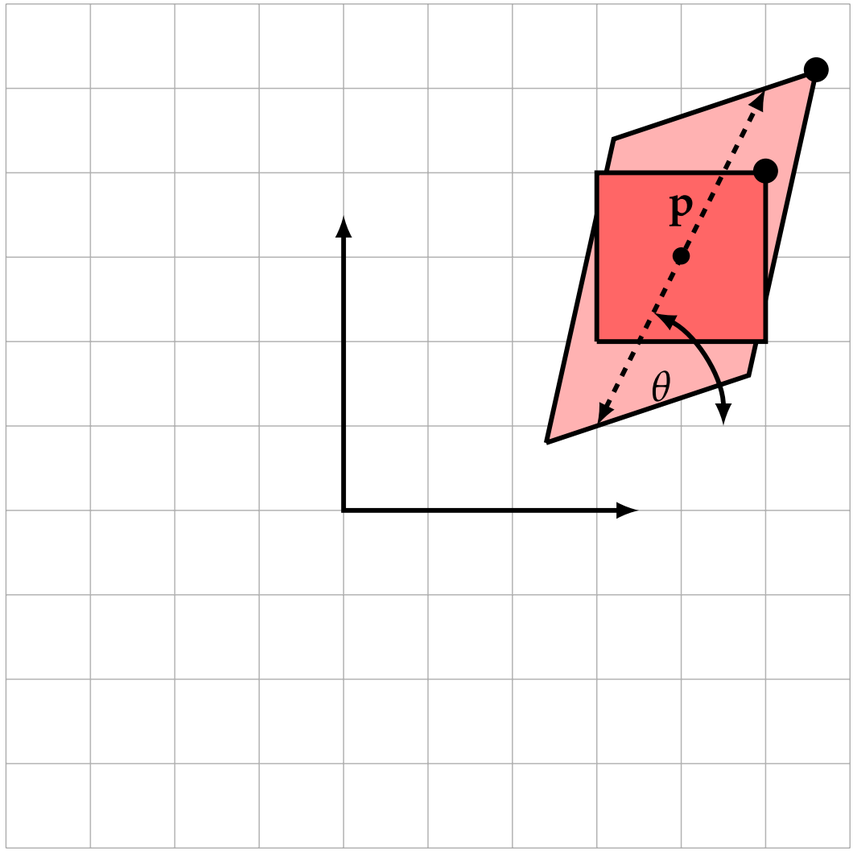
```
let eye = ... // some vec3 representing the eye
let center = ... // some vec3 representing a point to "look at"
let up = ... // some vec3 representing the "up" direction
let C = mat4.targetTo(mat4.create(), eye, center, up);

// for each pixel:
for (let j = 0; j < ny; ++j) {
  for (let i = 0; i < nx; ++i) {
    // 3d pixel coordinates relative to CAMERA
    const xc = -0.5 * w + w * (i + 0.5) / nx;
    const yc = -0.5 * h + h * (ny - 0.5 - j) / ny;
    const zc = -1;

    // 3d pixel coordinates relative to WORLD
    const q = vec4.transformMat4(vec4.create(), vec4.fromValues(xc, yc, zc, 1), C);

    // ray direction is then (qx, qy, qz) - eye
  }
}
```



Combining transformations: read from right-to-left!

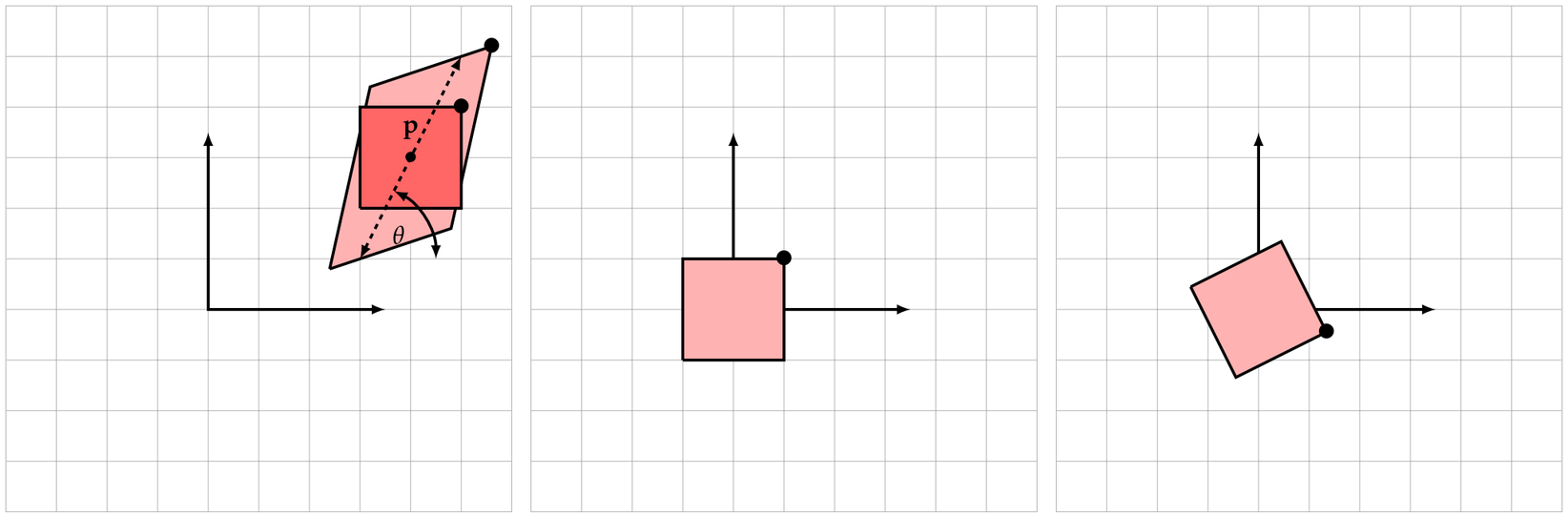


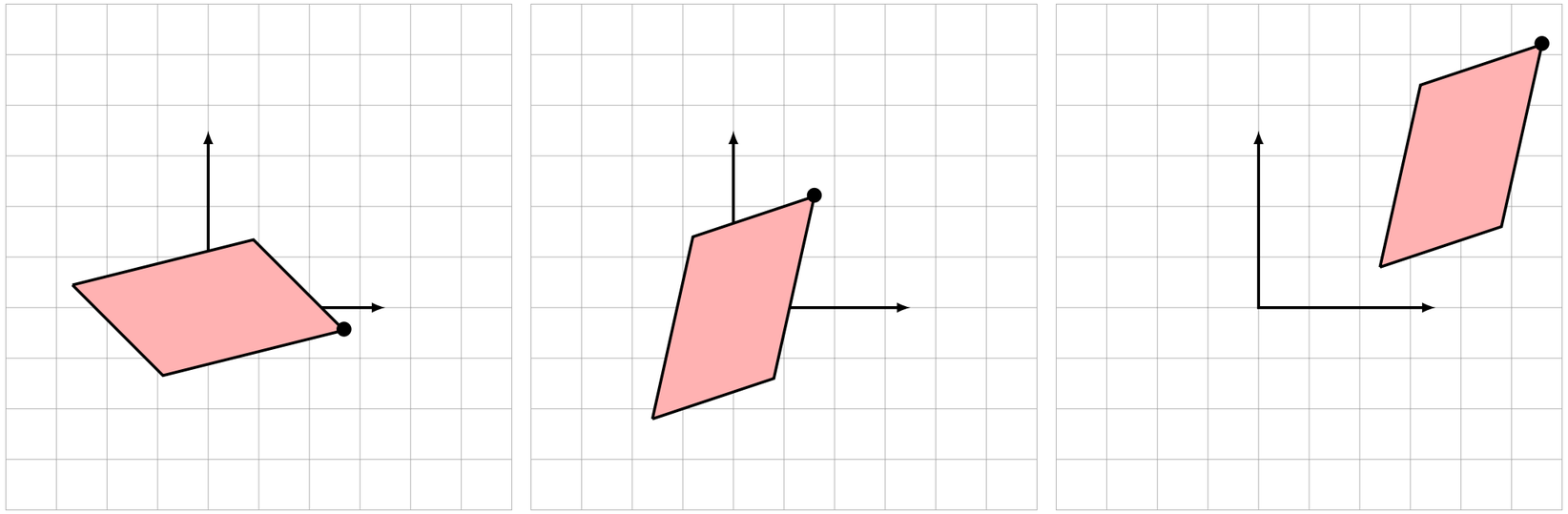
Example: Transforming the top-right corner (dot) of the square.



$$\text{Using}: \quad \mathbf{S} = \begin{bmatrix} s & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{R} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{T} = \begin{bmatrix} 1 & 0 & -p_x \\ 0 & 1 & -p_y \\ 0 & 0 & 1 \end{bmatrix}.$$

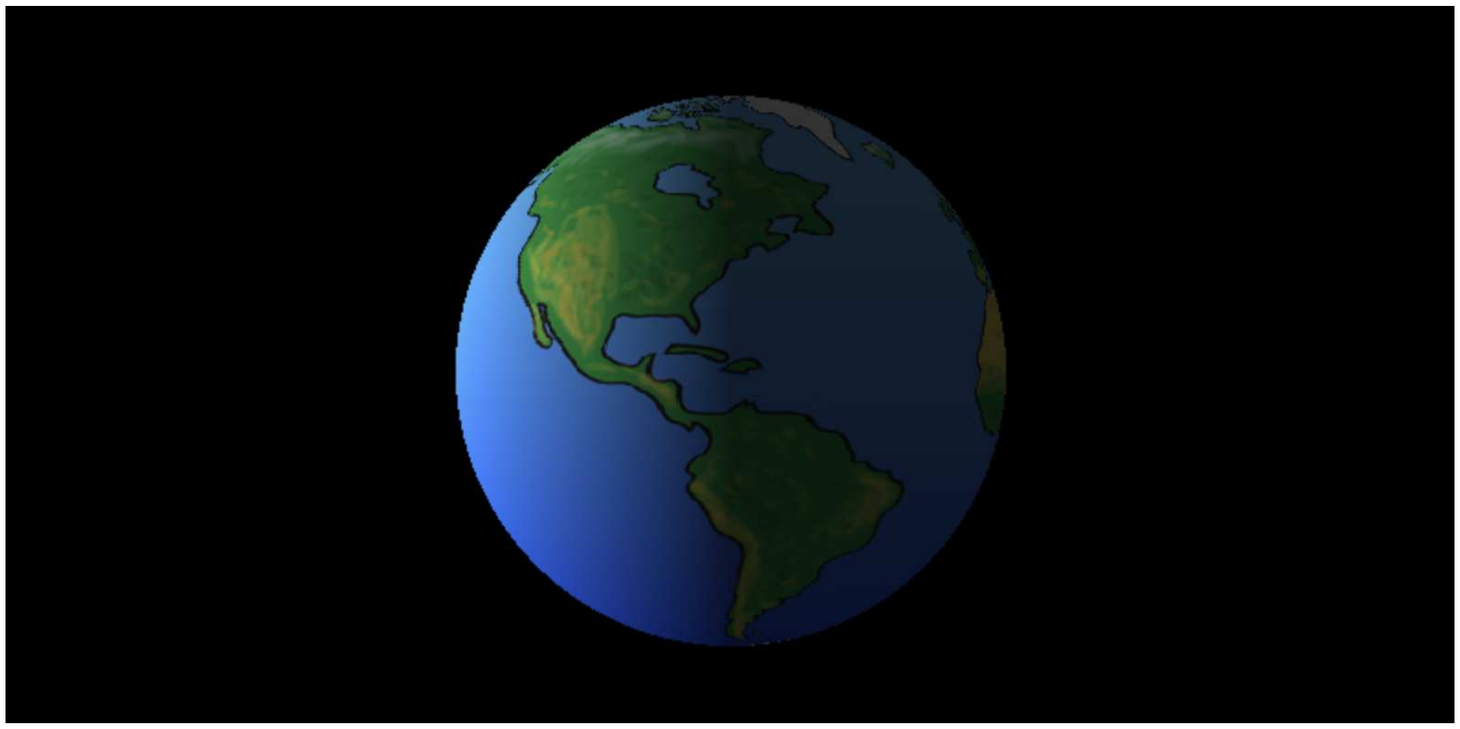
Solution: $\mathbf{M} = \mathbf{T}^{-1}\mathbf{R}^{-1}\mathbf{S}\mathbf{R}\mathbf{T}$.





Exercise: Rotate the Earth about its center.

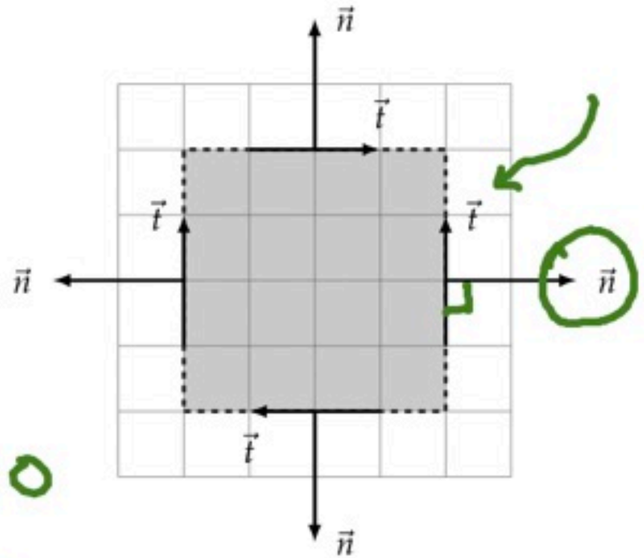
Click to open the editor.

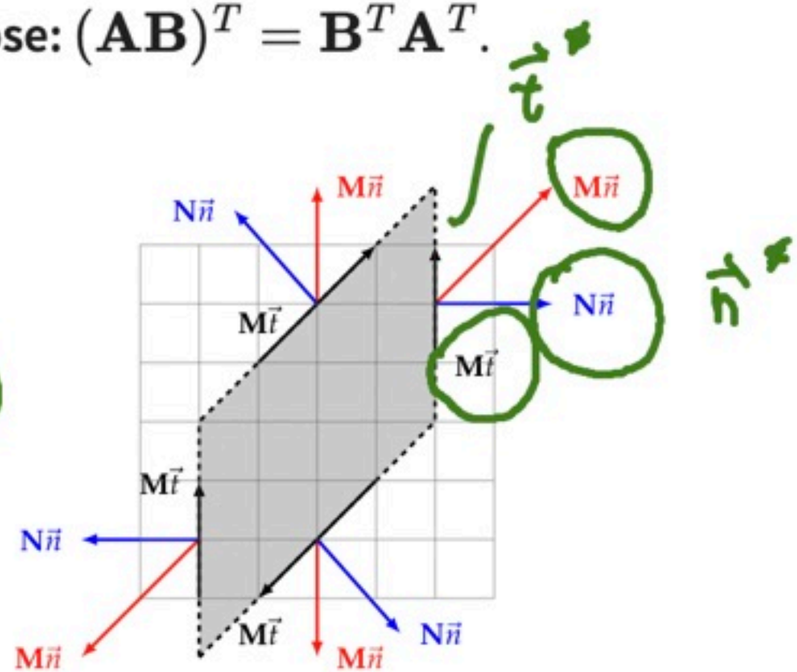


- Rotation about y-axis is called `theta`,
- Center of the Earth is called `center`,
- Look up `glMatrix` documentation (`vec4` and `mat4`)!

Transform normals using the inverse-transpose of your transformation.

Property of the transpose: $(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T$.





initially $\vec{n} \cdot \vec{t} = 0$

$\vec{n}_I^T \vec{t} = 0$

want: $\vec{n}^* \cdot \vec{t}^* = 0$

$(N\vec{n})^T (M\vec{t})$

$\vec{n}^T N^T M \vec{t} = 0$

$I = N^T M \rightarrow M^{-1} = N^T \rightarrow N = (M^{-1})^T$

inverse-transpose

Summary

- Transform ray direction using change-of-basis $\mathbf{B}$,
- Read transformations from right-to-left.
- Transform normals using inverse-transpose of transformation.