Tutorial 2

Transformations

Computer Graphics

Summer Semester 2020

Ludwig-Maximilians-Universität München

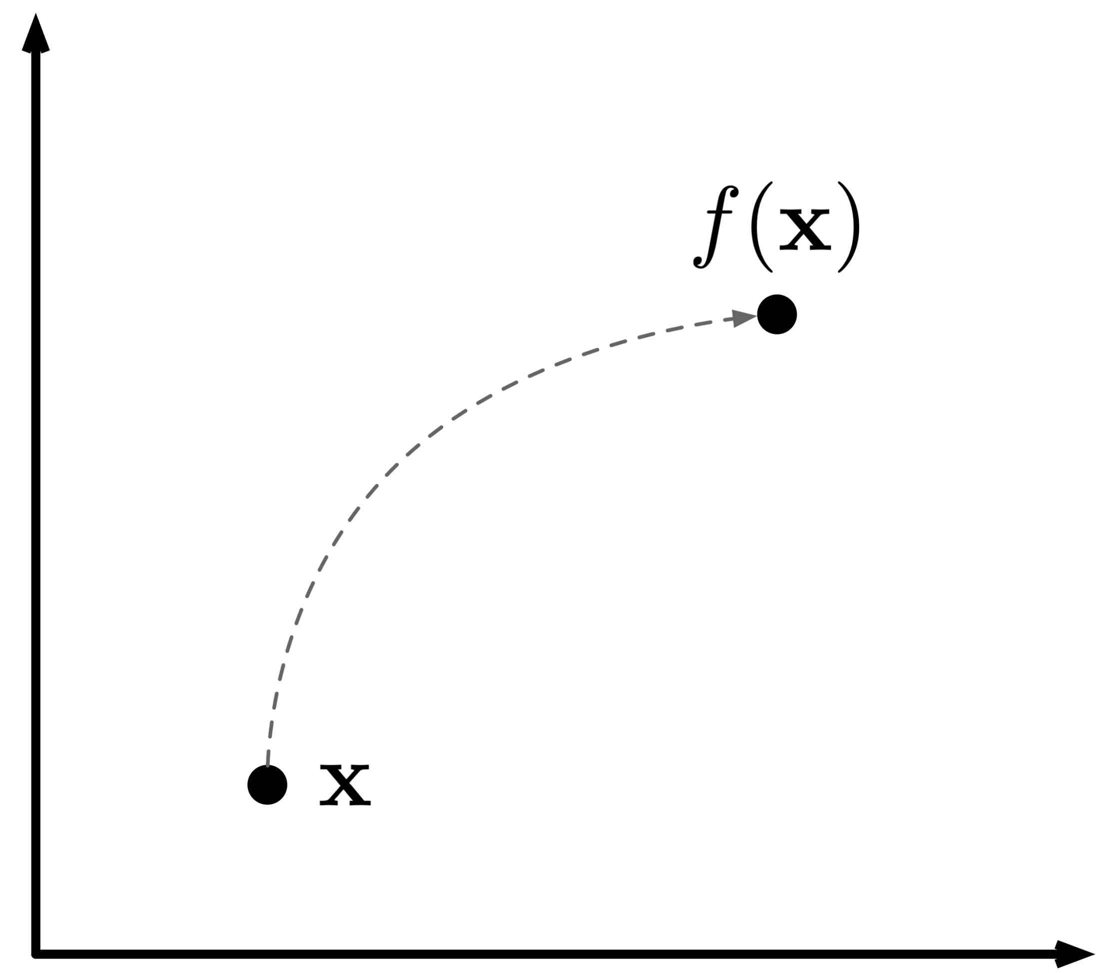
Agenda

- Homogeneous Coordinates
- Affine Transformations
- 3D Rotation: Euler Angles
- 3D Rotation: Quaternions
- Scene Graph
- three.js

Tutorial 2: Transformations

- Homogeneous Coordinates
- Affine Transformations
- 3D Rotation: Euler Angles
- 3D Rotation: Quaternions
- Scene Graph
- three.js

Transformation



Linear Transformations (Linear Map)

$$f(\mathbf{x}+\mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y})$$
$$f(a\mathbf{x}) = af(\mathbf{x})$$

Matrix multiplication $\mathbf{x}' = \mathbf{A}\mathbf{x}$ is a linear transformation because:

$$\mathbf{A}(\mathbf{x}+\mathbf{y}) = \mathbf{A}\mathbf{x} + \mathbf{A}\mathbf{y}$$
$$\mathbf{A}(a\mathbf{x}) = a\mathbf{A}\mathbf{x}$$

In 3D: $\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ E.g. scaling: $\begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{pmatrix}$

Translation??

Impossible to write in matrix multiplication:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}$$

So it is a special case. But we want a "*unified theory*".

Point or Vector??

Is this a point or a vector? We want to distinguish them.

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Solution: Homogeneous Coordinates

Go to a higher dimension:

- Point = $(x, y, z, \mathbf{1})^\top$
- Vector = $(x, y, z, \mathbf{0})^\top$
- Matrix = $\begin{pmatrix} a_{11} & a_{12} & a_{13} & t_x \\ a_{21} & a_{22} & a_{23} & t_y \\ a_{31} & a_{32} & a_{33} & t_z \\ w_1 & w_2 & w_3 & \mathbf{1} \end{pmatrix}$

Homogeneous Coordinates

- vector + vector = vector
- point - point = vector
- point + vector = point (see Assignment 1 Task 1 c)
- Homogeneous form of translation (**Assignment 2 Task 1 a**):

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Point remains a point
(location dependent)

$$\begin{pmatrix} x' \\ y' \\ z' \\ 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 0 \end{pmatrix}$$

Vector remains a vector
(location independent)

- Homogeneous form of scaling (**Assignment 2 Task 1 a**):

$$\mathbf{x}' = \begin{pmatrix} s_x x \\ s_y y \\ s_z z \\ 1 \end{pmatrix} = \mathbf{T}_s \mathbf{x} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Task 1 b) Why Homogeneous Coordinates?

- Homogeneous coordinates enable us to combine translation (and more) with linear transformations
- Homogeneous coordinates enable us to apply non-linear transformations as matrix multiplication, e.g. translation
- Homogeneous transformation can be inverted

… Invert???

Inverse Transformation

Inverse transformation == "*undo*" transformation $\mathbf{T}^{-1}$

$$\mathbf{x} = \mathbf{T}^{-1}\mathbf{T}\mathbf{x} = \mathbf{I}\mathbf{x} = \mathbf{x}$$

Task 1 c)

$$\mathbf{T}_s^{-1}\mathbf{T}_s = \begin{pmatrix} s'_x & 0 & 0 & 0 \\ 0 & s'_y & 0 & 0 \\ 0 & 0 & s'_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \mathbf{I}$$

$$\implies s'_x = \frac{1}{s_x}, s'_y = \frac{1}{s_y}, s'_z = \frac{1}{s_z} \implies \mathbf{T}_s^{-1} = \begin{pmatrix} 1/s_x & 0 & 0 & 0 \\ 0 & 1/s_y & 0 & 0 \\ 0 & 0 & 1/s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Task 1 c)

$$\mathbf{T}_t^{-1}\mathbf{T}_t = \begin{pmatrix} 1 & 0 & 0 & t'_x \\ 0 & 1 & 0 & t'_y \\ 0 & 0 & 1 & t'_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \mathbf{I}$$

$$\implies t'_x = -t_x, t'_y = -t_y, t'_z = -t_z$$

$$\implies \mathbf{T}_t^{-1} = \begin{pmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Task 1 d)

$$P = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ 1 \end{pmatrix} = \mathbf{T}_s^{-1}\mathbf{T}_t^{-1}P' = \begin{pmatrix} 1/s_x & 0 & 0 & 0 \\ 0 & 1/s_y & 0 & 0 \\ 0 & 0 & 1/s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p'_1 \\ p'_2 \\ p'_3 \\ 1 \end{pmatrix}$$

$$\implies P = \begin{pmatrix} 1/s_x & 0 & 0 & -t_x/s_x \\ 0 & 1/s_y & 0 & -t_y/s_y \\ 0 & 0 & 1/s_z & -t_z/s_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p'_1 \\ p'_2 \\ p'_3 \\ 1 \end{pmatrix} = \begin{pmatrix} p'_1/s_x - t_x/s_x \\ p'_2/s_y - t_y/s_y \\ p'_3/s_z - t_z/s_z \\ 1 \end{pmatrix}$$

Tutorial 2: Transformations

- Homogeneous Coordinates
- Affine Transformations
- 3D Rotation: Euler Angles
- 3D Rotation: Quaternions
- Scene Graph
- three.js

Task 1 e) Affine Transformation

Affine = Linear transformation + Translation. Is it linear map first or translation first?

$$\mathbf{T}_t\mathbf{T}_s = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 & t_x \\ 0 & s_y & 0 & t_y \\ 0 & 0 & s_z & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{T}_s\mathbf{T}_t = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 & s_x t_x \\ 0 & s_y & 0 & s_y t_y \\ 0 & 0 & s_z & s_z t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

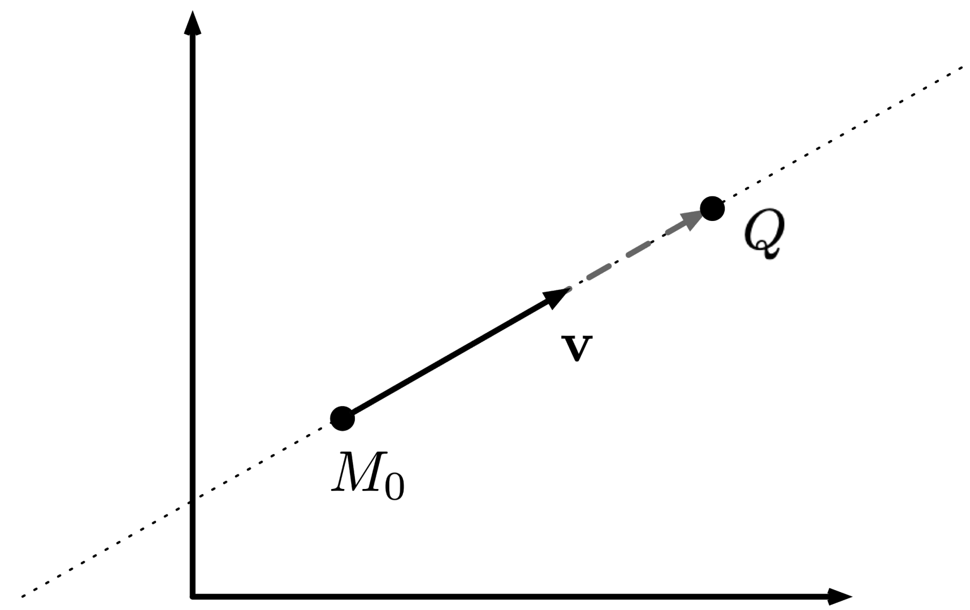
⇒ equal if and only if T*s* is an identity matrix ⇒ **Order matters!**

(Trivial: matrix multiplication has no commutation)

Affine transformations always apply linear map first then translation

Task 1 f) Parametric Equation

The given **Q** is on a line ⇒ The given coordinates represent a parametric equation of a line

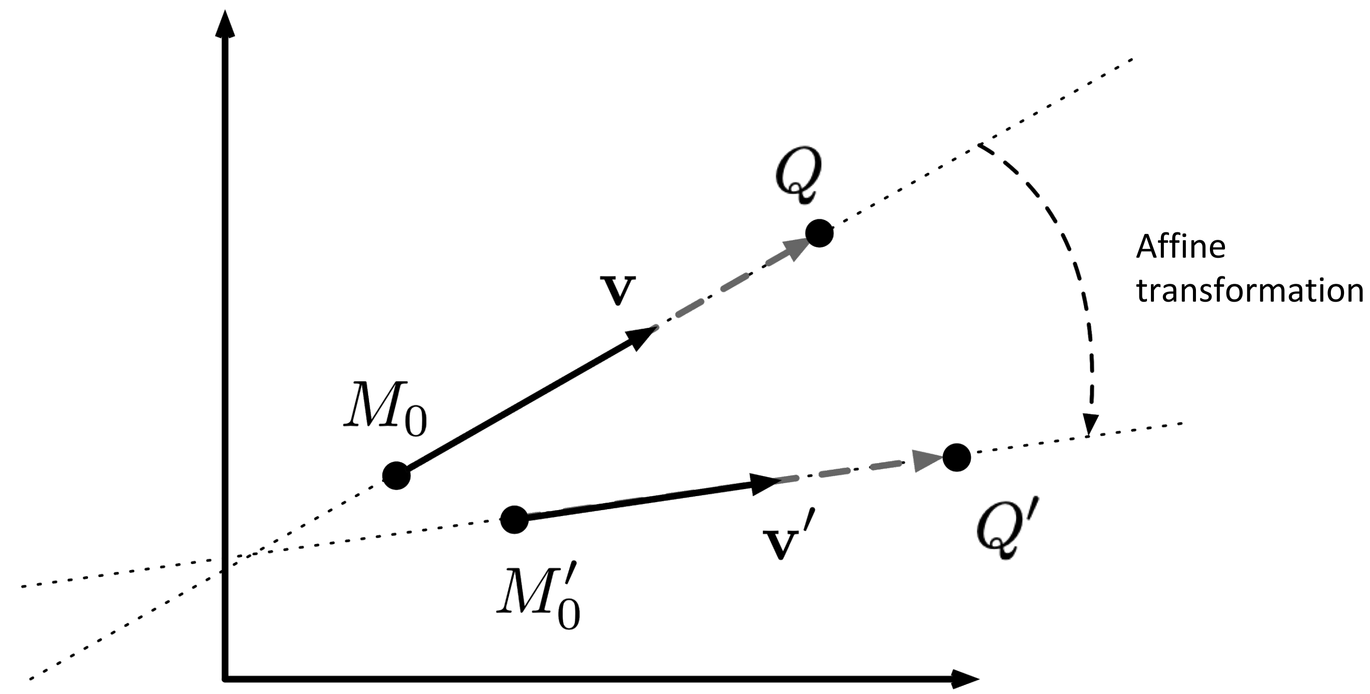


$$\mathbf{T}_t\mathbf{T}_sQ = \begin{pmatrix} s_x & 0 & 0 & t_x \\ 0 & s_y & 0 & t_y \\ 0 & 0 & s_z & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 + mt \\ y_0 + nt \\ z_0 + pt \\ 1 \end{pmatrix} = \begin{pmatrix} s_x(x_0 + mt) + t_x \\ s_y(y_0 + nt) + t_y \\ s_z(z_0 + pt) + t_z \\ 1 \end{pmatrix} = \begin{pmatrix} (s_x x_0 + t_x) + (s_x m)t \\ (s_y y_0 + t_y) + (s_y n)t \\ (s_z z_0 + t_z) + (s_z p)t \\ 1 \end{pmatrix}$$

A transformed line remains a line on the direction of $\mathbf{v} = (s_x m, s_y n, s_z p)^\top$

concerning $M_0' = (s_x x_0 + t_x, s_y y_0 + t_y, s_z z_0 + t_z)^\top$

Task 1 g) Property of Affine Transformation



$$\mathbf{T}_t\mathbf{T}_s Q = \begin{pmatrix} s_x & 0 & 0 & t_x \\ 0 & s_y & 0 & t_y \\ 0 & 0 & s_z & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 + mt \\ y_0 + nt \\ z_0 + pt \\ 1 \end{pmatrix} = \begin{pmatrix} s_x(x_0 + mt) + t_x \\ s_y(y_0 + nt) + t_y \\ s_z(z_0 + pt) + t_z \\ 1 \end{pmatrix} = \begin{pmatrix} (s_x x_0 + t_x) + (s_x m)t \\ (s_y y_0 + t_y) + (s_y n)t \\ (s_z z_0 + t_z) + (s_z p)t \\ 1 \end{pmatrix}$$

Collinearity: An affine transformed line remains a line in the direction of $\mathbf{v} = (s_x m, s_y n, s_z p)^\top$

concerning $M_0' = (s_x x_0, s_y y_0, s_z z_0)^\top$

More Properties of Affine Transformation

Collinearity: Lines remain a line

Parallelism: Parallels remain parallel

Convexity: Convex curves remain a convex curve

$\Rightarrow$ (line) proportion preserving

Types of Transformations

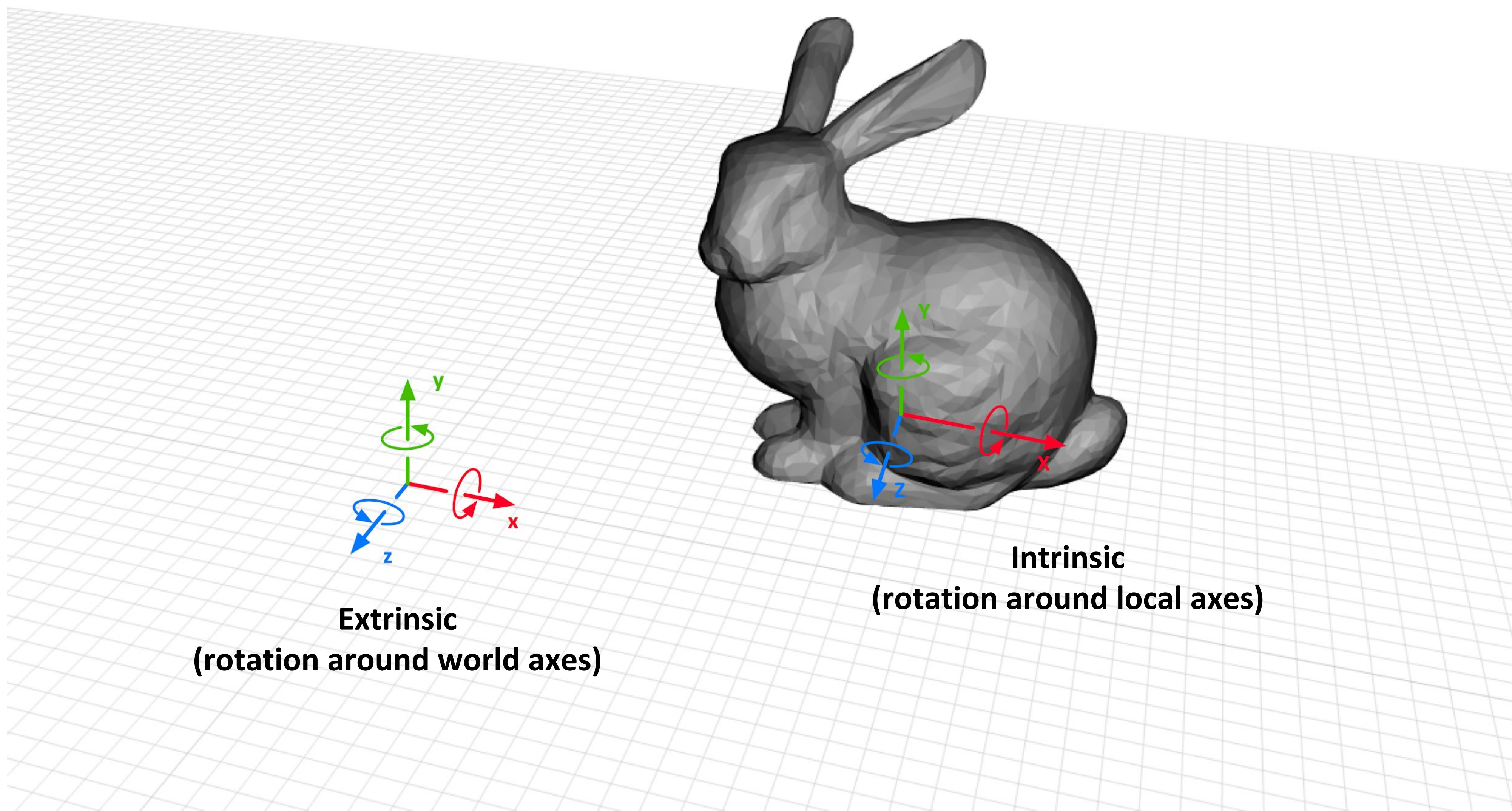
- Linear: Scale, rotation, reflection, shear, ...
- Non-linear: translation, ...
- **Affine: linear transformation + translation**
    - line proportion preserving
- *Isometric*: Translation, rotation, reflection
    - distance preserving
- Non-affine? Not now.

Tutorial 2: Transformations

- Homogeneous Coordinates
- Affine Transformations
- 3D Rotation: Euler Angles
- 3D Rotation: Quaternions
- Scene Graph
- three.js

Euler Angles

- A rotation around axes can be expressed using the so called ***Euler Angles***
- Depending on the reference, there are ***Extrinsic*** and ***Intrinsic*** Euler Angles
  - Extrinsic rotation has a fixed reference frame
  - Intrinsic rotation has a dynamic reference frame (why?)



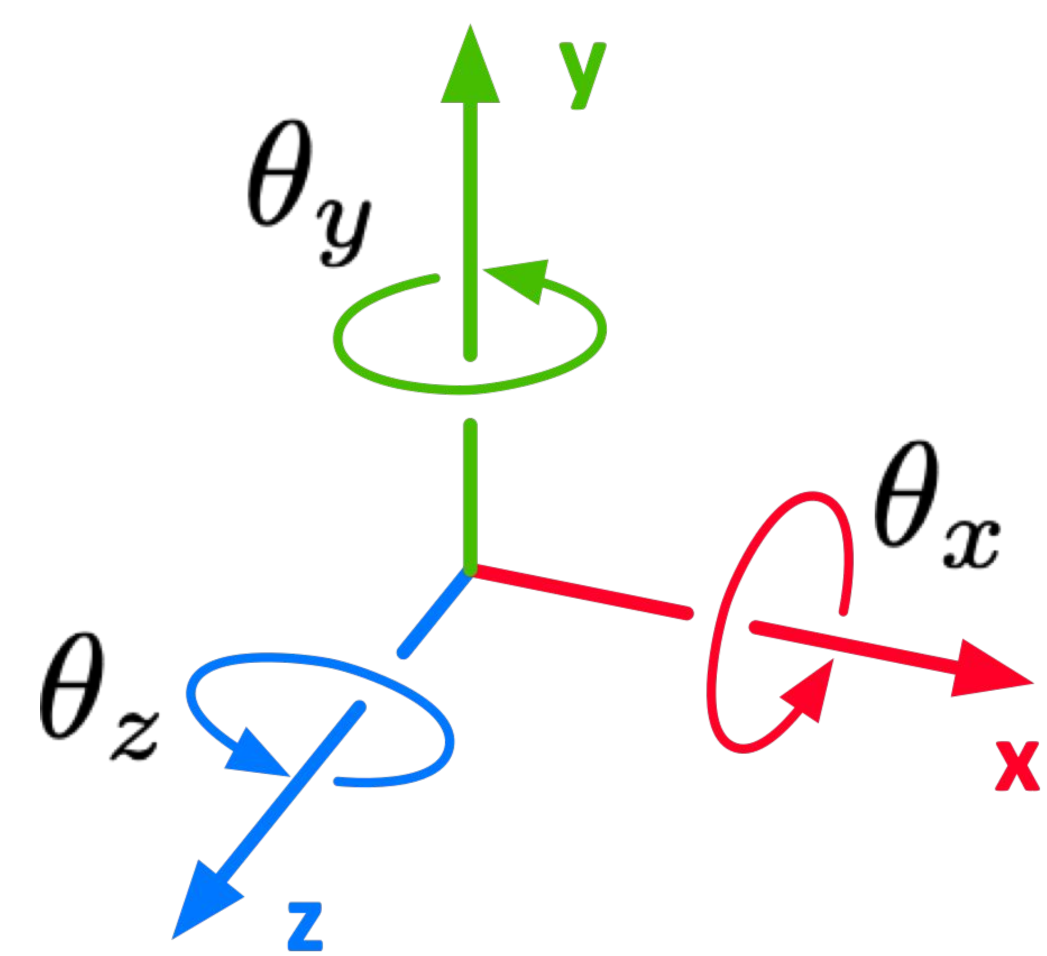
Task 2 a) Extrinsic Rotation

Formulas taken from lecture slides:

$$\mathbf{R}_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{pmatrix}$$

$$\mathbf{R}_y = \begin{pmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{pmatrix}$$

$$\mathbf{R}_z = \begin{pmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{pmatrix}$$



Task 2 b) and c)

$$\mathbf{R}_x\mathbf{R}_y\mathbf{R}_z = \begin{pmatrix} \cos\theta_y\cos\theta_z & -\cos\theta_y\sin\theta_z & \sin\theta_y \\ \cos\theta_z\sin\theta_x\sin\theta_y + \cos\theta_x\sin\theta_z & \cos\theta_x\cos\theta_z - \sin\theta_x\sin\theta_y\sin\theta_z & -\cos\theta_y\sin\theta_x \\ -\cos\theta_x\cos\theta_z\sin\theta_y + \sin\theta_x\sin\theta_z & \cos\theta_z\sin\theta_x + \cos\theta_x\sin\theta_y\sin\theta_z & \cos\theta_x\cos\theta_y \end{pmatrix}$$

You learned order matters in Task 1 e), so don't do this:

$$\xcancel{\mathbf{R}_z\mathbf{R}_y\mathbf{R}_x = \begin{pmatrix} \cos\theta_x\cos\theta_y & \cos\theta_x\sin\theta_y\sin\theta_z - \cos\theta_z\sin\theta_x & \cos\theta_x\cos\theta_z\sin\theta_y + \sin\theta_x\sin\theta_z \\ \cos\theta_y\sin\theta_x & \sin\theta_x\sin\theta_y\sin\theta_z + \cos\theta_x\cos\theta_z & \cos\theta_z\sin\theta_x\sin\theta_x - \cos\theta_x\sin\theta_z \\ -\sin\theta_y & \cos\theta_y\sin\theta_z & \cos\theta_y\cos\theta_z \end{pmatrix}}$$

You just need a counterexample, **do this:**

The element on the first row and first column:

$$\cos\theta_y\cos\theta_z = \cos\theta_x\cos\theta_y \text{ if and only if } \theta_z = \theta_x + 2n\pi, n \in \mathbb{N}$$

This is generally not true. Thus: No, we can't switch the order.

Task 2 f) and g) Euler Angle Sequences

- Depends on the order of rotation, there are ***Tait-Bryan angles*** or ***Proper Euler angles***
- **Tait-Bryan angles**
  - 6 possible sequences: xyz, xzy, yxz, yzx, zxy, zyx.
- **Proper Euler angles**
  - 6 possible sequences: xyx, xzx, yxy, yzy, zxz, zyz.

What about xxy, xxz, yyx, yyz, zzx, zzy, xyy, xzz, yxx, yzz, zxx, zyy?

xx, yy, zz can be merged to a single X, Y, or Z rotation.

These orders can collapse to xy, xz, yx, yz, zx, zy (which are not *three* composed rotations)

Task 2 d)

$$\mathbf{R}_x\mathbf{R}_y\mathbf{R}_z = \begin{pmatrix} \cos\theta_y\cos\theta_z & -\cos\theta_y\sin\theta_z & \sin\theta_y \\ \cos\theta_z\sin\theta_x\sin\theta_y + \cos\theta_x\sin\theta_z & \cos\theta_x\cos\theta_z - \sin\theta_x\sin\theta_y\sin\theta_z & -\cos\theta_y\sin\theta_x \\ -\cos\theta_x\cos\theta_z\sin\theta_y + \sin\theta_x\sin\theta_z & \cos\theta_z\sin\theta_x + \cos\theta_x\sin\theta_y\sin\theta_z & \cos\theta_x\cos\theta_y \end{pmatrix}$$

If $\theta_y = \frac{\pi}{2}$ then $\cos\theta_y = 0, \sin\theta_y = 1$

$$\mathbf{R}_x\mathbf{R}_y\mathbf{R}_z = \begin{pmatrix} 0 & 0 & 1 \\ \cos\theta_z\sin\theta_x + \cos\theta_x\sin\theta_z & \cos\theta_x\cos\theta_z - \sin\theta_x\sin\theta_z & 0 \\ -\cos\theta_x\cos\theta_z + \sin\theta_x\sin\theta_z & \cos\theta_z\sin\theta_x + \cos\theta_x\sin\theta_z & 0 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 0 & 1 \\ \sin(\theta_x+\theta_z) & \cos(\theta_x+\theta_z) & 0 \\ -\cos(\theta_x+\theta_z) & \sin(\theta_x+\theta_z) & 0 \end{pmatrix}$$

What's the geometric meaning behind it?

Task 2 d) *Gimbal Lock*

No matter what you do with θ_x, θ_z , you will always land on a the same plane, *the x*-axis is fixed to *z*, meaning a *single axis rotation*.

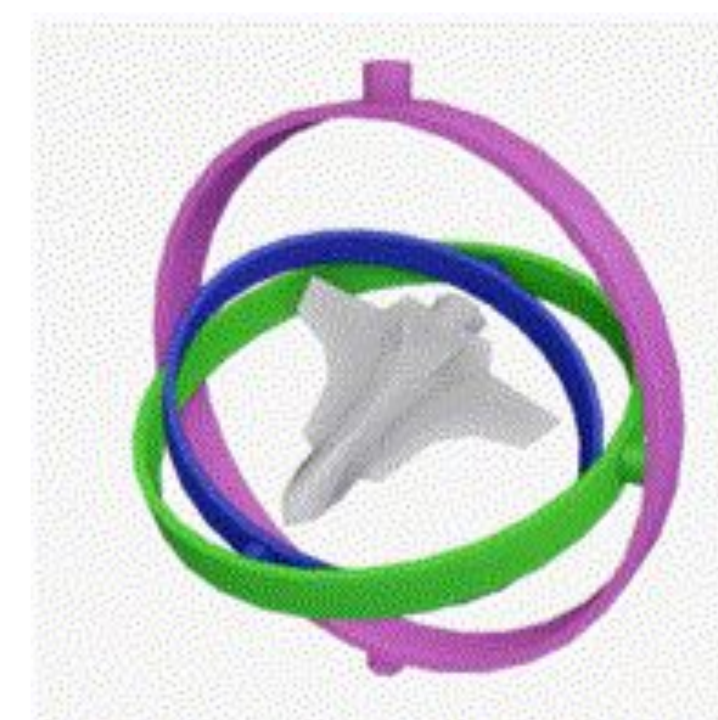
$$\mathbf{R}_x\mathbf{R}_y\mathbf{R}_z P = \begin{pmatrix} 0 & 0 & 1 \\ \sin(\theta_x + \theta_z) & \cos(\theta_x + \theta_z) & 0 \\ -\cos(\theta_x + \theta_z) & \sin(\theta_x + \theta_z) & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} z \\ \dots \\ \dots \end{pmatrix}$$

This is called the **"Gimbal Lock".**

It is one of the limitations of Euler angles.

Q: Is a reference frame influencing our result?

A: No, the calculation process is irrelevant to the reference frame.



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

Tutorial 2: Transformations

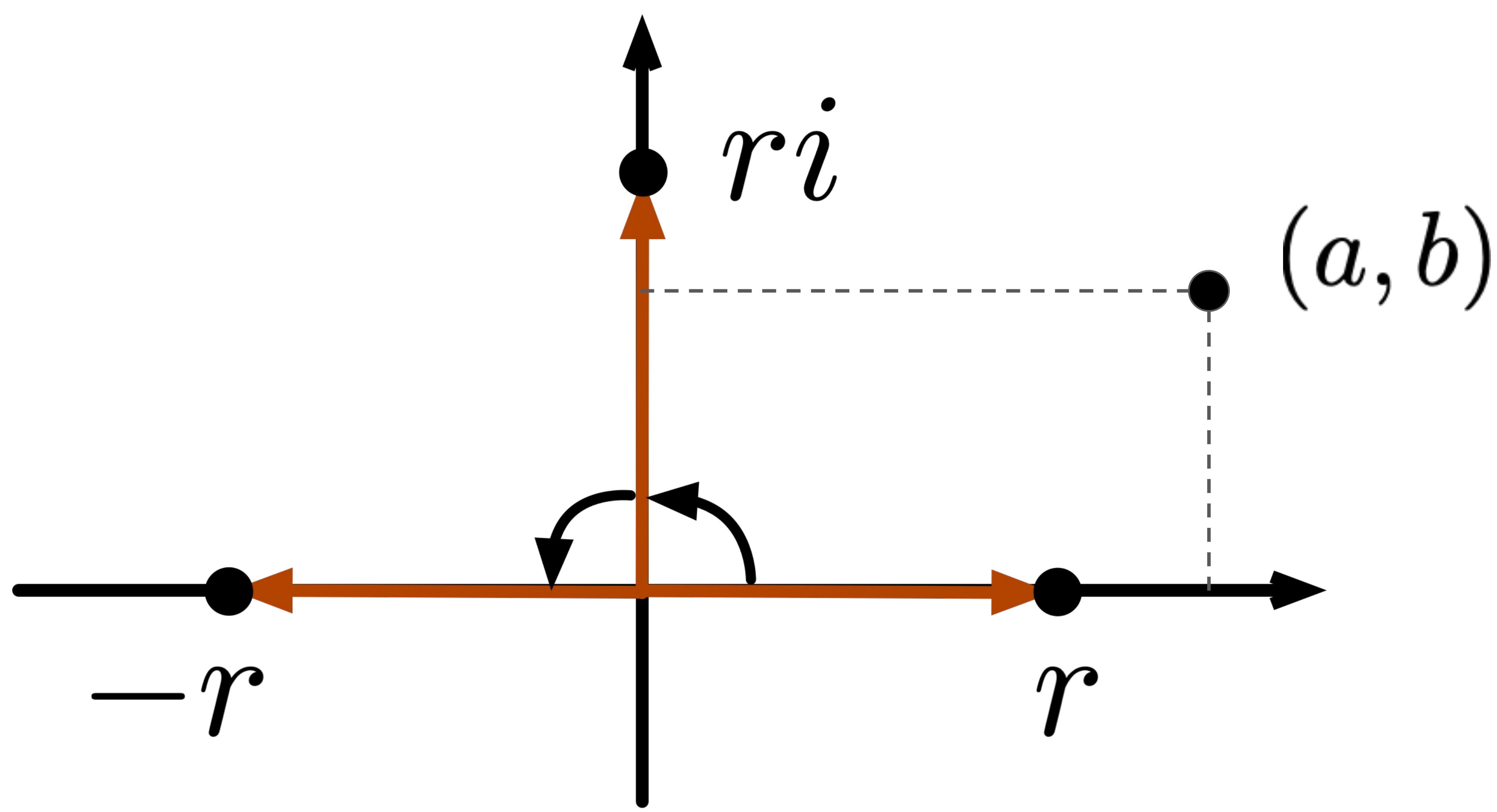
- Homogeneous Coordinates
- Affine Transformations
- 3D Rotation: Euler Angles
- 3D Rotation: Quaternions
- Scene Graph
- three.js

Complex Numbers

- Consist of a real part, and an imaginary part:

$$a + bi \in \mathbb{C}, a, b \in \mathbb{R} \qquad i^2 = -1$$

- Complex number multiplication represents a 2D rotation, simple case:



- Complex numbers looks like points on 2D plane. What's the equivalent in 3D?

Quaternions

A ***quaternion*** has one real, three imaginary parts:

$$\mathbf{q} = a + bi + cj + dk, i^2 = j^2 = k^2 = ijk = -1$$

Hard to imagine something in 4D!

Let's do the math that can help us to understand more what's going on here.

Task 3 a)

$$\mathbf{pq} = (e + fi + gj + hk)(a + bi + cj + dk)$$
$$= ea + ebi + ecj + edk$$

Task 3 a)

$$\mathbf{pq} = (e + fi + gj + hk)(a + bi + cj + dk)$$
$$= ea + ebi + ecj + edk$$
$$+afi + bfi^2 + cfij + dfik$$

Task 3 a)

$$
\begin{aligned}
\mathbf{pq} &= (e + fi + gj + hk)(a + bi + cj + dk) \\
&= ea + ebi + ecj + edk \\
&\quad + afi + bfi^2 + cfij + dfik \\
&\quad + agj + bgji + cgj^2 + dgjk
\end{aligned}
$$

Task 3 a)

$$
\begin{aligned}
\mathbf{pq} &= (e + fi + gj + hk)(a + bi + cj + dk) \\
&= ea + ebi + ecj + edk \\
&\quad + afi + bfi^2 + cfij + dfik \\
&\quad + agj + bgji + cgj^2 + dgjk \\
&\quad + ahk + bhki + chkj + dhk^2
\end{aligned}
$$

Task 3 a)

$$\mathbf{pq} = (e + fi + gj + hk)(a + bi + cj + dk)$$
$$= ea + ebi + ecj + edk$$
$$+afi + bfi^2 + cfij + dfik$$
$$+agj + bgji + cgj^2 + dgjk$$
$$+ahk + bhki + chkj + dhk^2$$

$$i^2 = j^2 = k^2 = ijk = -1$$
$$ij = ?$$
$$ji = ?$$
$$ik = ?$$
$$ki = ?$$
$$jk = ?$$
$$kj = ?$$

Task 3 a)

$$i^2 = j^2 = k^2 = ijk = -1$$

$$i^2 = ijk \implies jk = i$$

$$k^2 = ijk \implies ij = k$$

$$-ijk = 1 = k(-k) = kjjk \implies kj = -i$$

$$-ijk = 1 = j(-j) = jiij \implies ji = -k$$

$$ki = ki(jkkj) = k(ijk)kj = -kkj = j$$

$$ik = i(jkkj)k = (ijk)kjk = -kjk = -ki = -j$$

Task 3 a)

$$\begin{aligned}\mathbf{pq} &= (e + fi + gj + hk)(a + bi + cj + dk) \\ &= ea + ebi + ecj + edk \\ &\quad +afi + bfi^2 + cfij + dfik \\ &\quad +agj + bgji + cgj^2 + dgjk \\ &\quad +ahk + bhki + chkj + dhk^2\end{aligned}$$

$$\begin{aligned} i^2 &= j^2 = k^2 = ijk = -1 \\ ij &= k \\ ji &= -k \\ ik &= -j \\ ki &= j \\ jk &= i \\ kj &= -i \end{aligned}$$

Task 3 a)

$$\mathbf{pq} = (e + fi + gj + hk)(a + bi + cj + dk)$$
$$= \boxed{ea} + ebi + ecj + edk$$
$$+afi + \boxed{bfi^2} + cfij + dfik$$
$$+agj + bgji + \boxed{cgj^2} + dgjk$$
$$+ahk + bhki + chkj + \boxed{dhk^2}$$
$$= (ea - bf - cg - dh)$$

$$i^2 = j^2 = k^2 = ijk = -1$$
$$ij = k$$
$$ji = -k$$
$$ik = -j$$
$$ki = j$$
$$jk = i$$
$$kj = -i$$

Task 3 a)

$$\begin{aligned}\mathbf{pq} &= (e + fi + gj + hk)(a + bi + cj + dk) \\ &= ea + \boxed{ebi} + ecj + edk \\ &\quad + \boxed{afi} + bfi^2 + cfij + dfik \\ &\quad + agj + bgji + cgj^2 + \boxed{dgjk} \\ &\quad + ahk + bhki + \boxed{chkj} + dhk^2 \\ &= (ea - bf - cg - dh) \\ &\quad + (eb + af + dg - ch)i\end{aligned}$$

$$i^2 = j^2 = k^2 = ijk = -1$$
$$ij = k$$
$$ji = -k$$
$$ik = -j$$
$$ki = j$$
$$\boxed{\begin{aligned} jk &= i \\ kj &= -i \end{aligned}}$$

Task 3 a)

$$\mathbf{pq} = (e + fi + gj + hk)(a + bi + cj + dk)$$
$$= ea + ebi + \boxed{ecj} + edk$$
$$+afi + bfi^2 + cfij + \boxed{dfik}$$
$$+\boxed{agj} + bgji + cgj^2 + dgjk$$
$$+ahk + \boxed{bhki} + chkj + dhk^2$$
$$= (ea - bf - cg - dh)$$
$$+(eb + af + dg - ch)i$$
$$+(ec - df + ag + bh)j$$

$$i^2 = j^2 = k^2 = ijk = -1$$
$$ij = k$$
$$ji = -k$$
$$\boxed{\begin{aligned} ik &= -j \\ ki &= j \end{aligned}}$$
$$jk = i$$
$$kj = -i$$

Task 3 a)

$$
\begin{aligned}
\mathbf{pq} &= (e + fi + gj + hk)(a + bi + cj + dk) \\
&= ea + ebi + ecj + \boxed{edk} \\
&\quad + afi + bfi^2 + \boxed{cfij} + dfik \\
&\quad + agj + \boxed{bgji} + cgj^2 + dgjk \\
&\quad + \boxed{ahk} + bhki + chkj + dhk^2 \\
&= (ea - bf - cg - dh) \\
&\quad + (eb + af + dg - ch)i \\
&\quad + (ec - df + ag + bh)j \\
&\quad + (ed + cf - bg + ah)k
\end{aligned}
$$

$$
i^2 = j^2 = k^2 = ijk = -1
$$

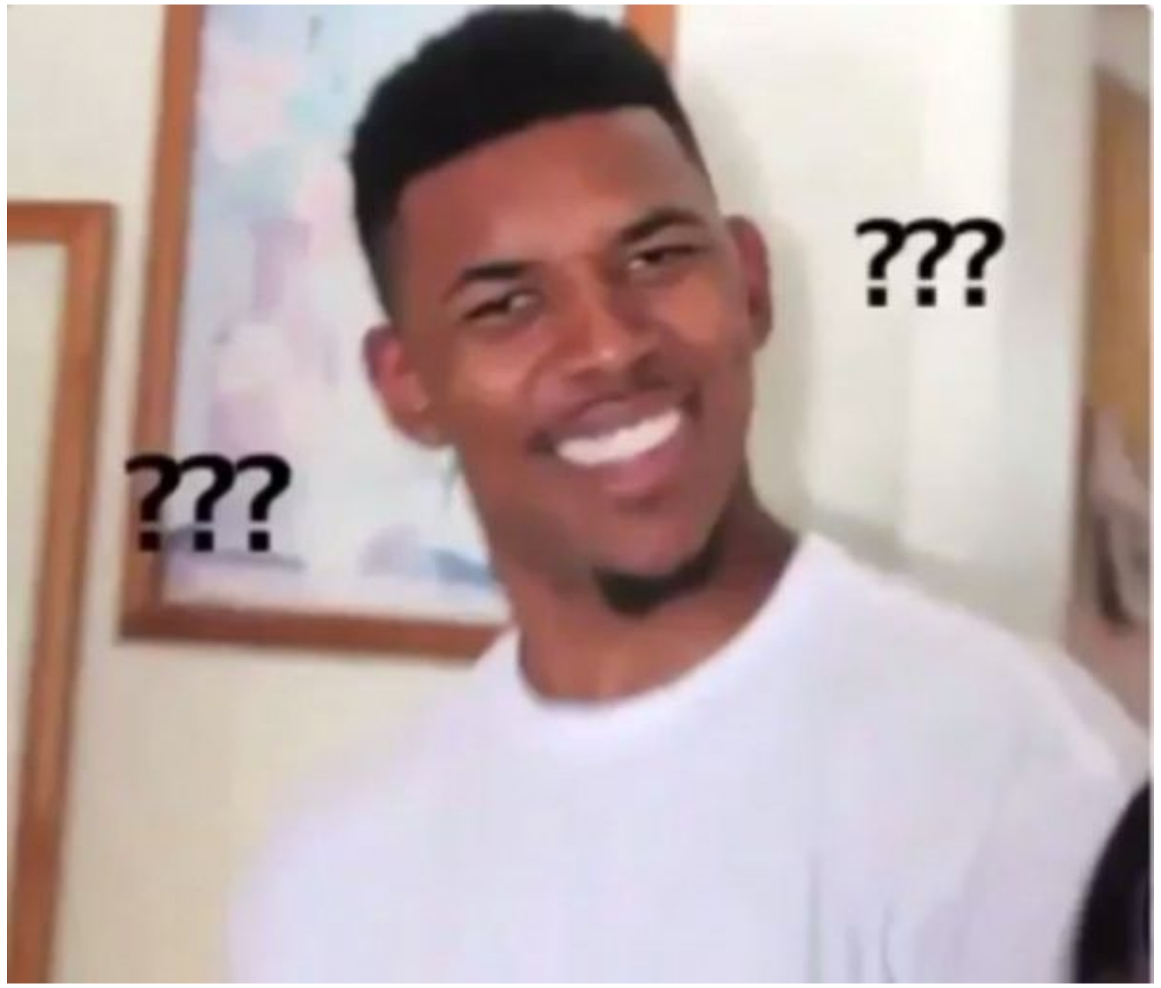
$$
\boxed{\begin{aligned} ij &= k \\ ji &= -k \end{aligned}}
$$

$$
\begin{aligned}
ik &= -j \\
ki &= j \\
jk &= i \\
kj &= -i
\end{aligned}
$$

Task 3 a)

$$\mathbf{pq} = (e + fi + gj + hk)(a + bi + cj + dk)$$

$$\begin{aligned} &= (ea - bf - cg - dh) \\ &+(eb + af + dg - ch)i \\ &+(ec - df + ag + bh)j \\ &+(ed + cf - bg + ah)k \end{aligned}$$



Q: Why am I doing this?

A: I know, it's complicated. Be patient.

Quaternion Multiplication

Let's simplify this:

$$
\begin{aligned}
\mathbf{pq} &= (e + fi + gj + hk)(a + bi + cj + dk) \\
&= (ea - bf - cg - dh) \\
&\quad + (eb + af + dg - ch)i \\
&\quad + (ec - df + ag + bh)j \\
&\quad + (ed + cf - bg + ah)k \\
&= ea - (bf + cg + dh)
\end{aligned}
$$

$$\mathbf{p} = (e, \mathbf{w}), \mathbf{w} = (f, g, h)^\top$$

$$\mathbf{q} = (a, \mathbf{v}), \mathbf{v} = (b, c, d)^\top$$

$$ea - \mathbf{w}^\top \cdot \mathbf{v}$$

Quaternion Multiplication

Let's simplify this:

$$
\begin{aligned}
\mathbf{pq} &= (e + fi + gj + hk)(a + bi + cj + dk) \\
&= (ea - bf - cg - dh) \\
&\quad +(eb + af + dg - ch)i \\
&\quad +(ec - df + ag + bh)j \\
&\quad +(ed + cf - bg + ah)k \\
&= ea - (bf + cg + dh) \\
&\quad +e(bi + cj + dk)
\end{aligned}
$$

$$\mathbf{p} = (e, \mathbf{w}), \mathbf{w} = (f, g, h)^\top$$

$$\mathbf{q} = (a, \mathbf{v}), \mathbf{v} = (b, c, d)^\top$$

$$ea - \mathbf{w}^\top \cdot \mathbf{v}$$

$$e\mathbf{v}$$

Quaternion Multiplication

Let's simplify this:

$$\mathbf{pq} = (e + fi + gj + hk)(a + bi + cj + dk)$$

$$\begin{aligned} &= (ea - bf - cg - dh) \\ &\quad +(eb + af + dg - ch)i \\ &\quad +(ec - df + ag + bh)j \\ &\quad +(ed + cf - bg + ah)k \end{aligned}$$

$$\mathbf{p} = (e, \mathbf{w}), \mathbf{w} = (f, g, h)^\top$$

$$\mathbf{q} = (a, \mathbf{v}), \mathbf{v} = (b, c, d)^\top$$

$$\begin{aligned} &= ea - (bf + cg + dh) && ea - \mathbf{w}^\top \cdot \mathbf{v} \\ &\quad +e(bi + cj + dk) && e\mathbf{v} \\ &\quad +a(fi + gj + hk) && a\mathbf{w} \\ &\quad +(dg - ch)i + (bh - df)j + (cf - bg)k && \mathbf{w} \times \mathbf{v} \end{aligned}$$

Quaternion Multiplication

$$\mathbf{p} = (e, \mathbf{w}), \mathbf{w} = (f, g, h)^\top$$

$$\mathbf{q} = (a, \mathbf{v}), \mathbf{v} = (b, c, d)^\top$$

$$\mathbf{pq} = (ea - \mathbf{w}^T \cdot \mathbf{v}, e\mathbf{v} + a\mathbf{w} + \mathbf{w} \times \mathbf{v})$$

Task 3 b)

$$\mathbf{q} = (a, \mathbf{v}), \mathbf{v} = (b, c, d)^\top \ \mathbf{pq} = (ea - \mathbf{w}^T \cdot \mathbf{v}, e\mathbf{v} + a\mathbf{w} + \mathbf{w} \times \mathbf{v})$$

$$\mathbf{q} = (\cos\theta, \mathbf{u}\sin\theta), \bar{\mathbf{q}} = (\cos\theta, -\mathbf{u}\sin\theta)$$

The imaginary part: $-\cos\theta\mathbf{u} + \cos\theta\mathbf{u} + (-\mathbf{u} \times \mathbf{u}) = 0$

The real part: $\cos^2\theta + \sin^2\theta\mathbf{u}^\top \cdot \mathbf{u}$

The multiplication result in a real number:

$$\bar{\mathbf{q}}\mathbf{q} = \cos^2\theta + \sin^2\theta\mathbf{u}^\top \cdot \mathbf{u}$$

Task 3 b) *Unit Quaternion*

$$\mathbf{q} = (\cos\theta, \mathbf{u}\sin\theta), \bar{\mathbf{q}} = (\cos\theta, -\mathbf{u}\sin\theta)$$

$$\cos^2\phi + \sin^2\phi = 1$$

$$\implies ||\mathbf{q}||^2 = \bar{\mathbf{q}}\mathbf{q} = \cos^2\theta + \sin^2\theta ||\mathbf{u}||^2 = 1 \iff ||\mathbf{u}|| = 1$$

A quaternion with unit norm is the so called ***unit quaternion***.

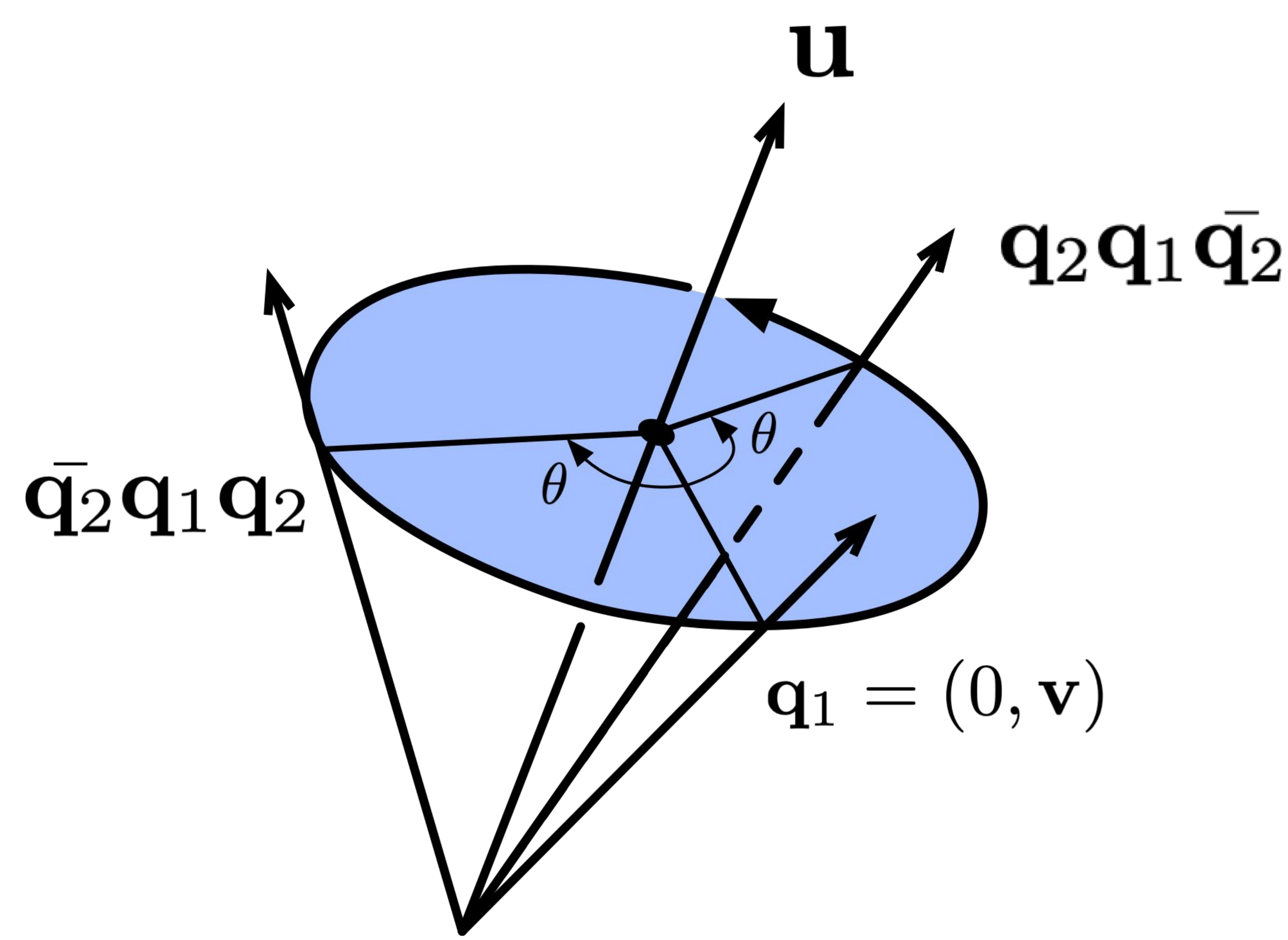
Rotation with Quaternions

Rotation in 3D can be expressed using ***unit quaternion***.

Given unit axis **u**, angle **θ**, unit quaternion **q** can represent a rotation.

$$\mathbf{q} = (\cos\frac{\theta}{2}, \mathbf{u}\sin\frac{\theta}{2})$$

angle **axis**



Be careful about the angle: the rotation is expressed by multiplication of two unit quaternions,
then for one quaternion, θ is divided by 2 and results in θ/2

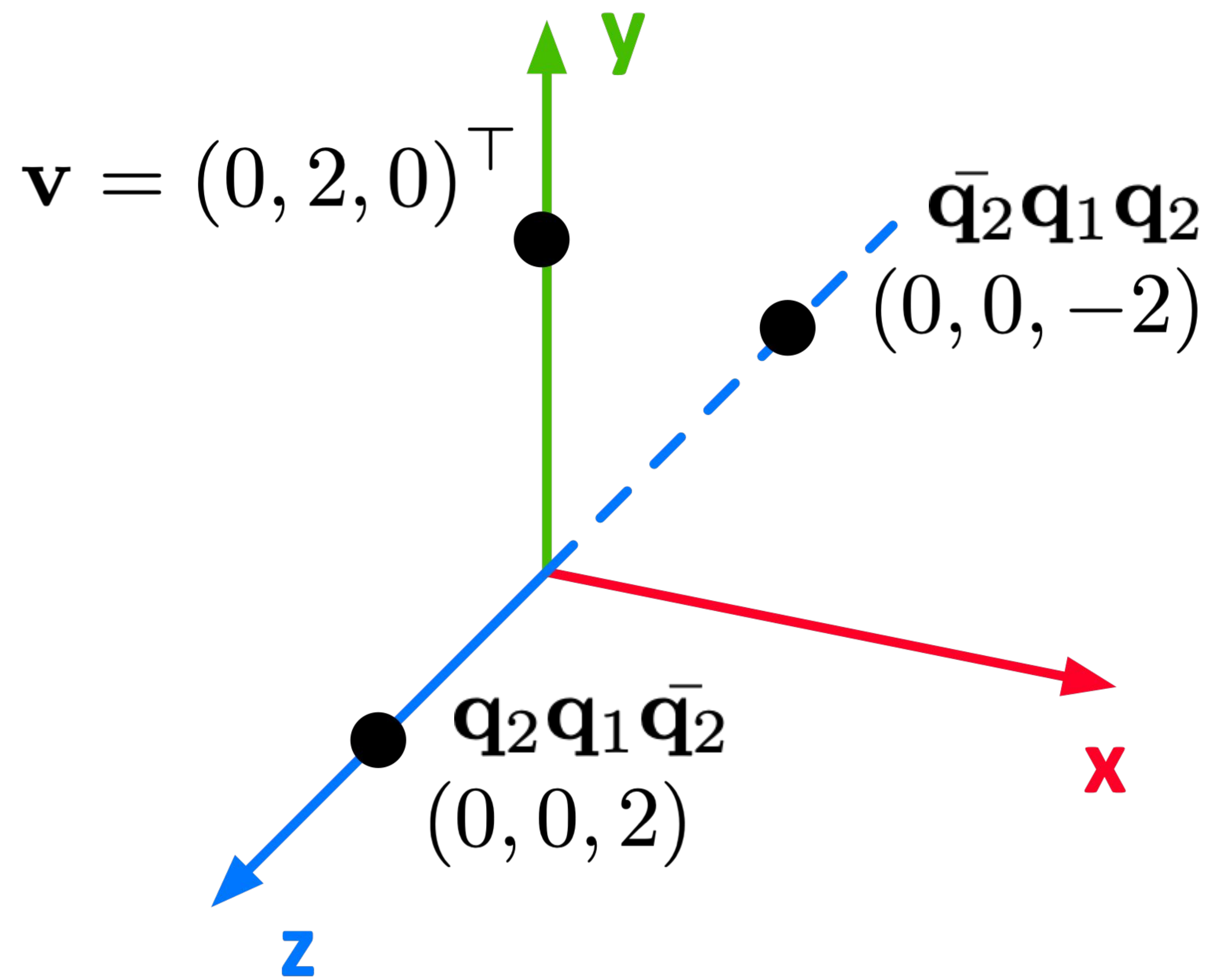
Task 3 c)

The meaning of those quaternions is quite clear.

$\mathbf{q}_2 = (\cos\frac{\pi}{4}, \mathbf{u}\sin\frac{\pi}{4})$ means rotate 90 degrees on the x-axis $\mathbf{u} = (1,0,0)^\top$

The clockwise rotation $\bar{\mathbf{q}}_2\mathbf{q}_1\mathbf{q}_2$

The counterclockwise rotation $\mathbf{q}_2\mathbf{q}_1\bar{\mathbf{q}}_2$



Task 3 d)

Task 3 c) already tells you the quaternion for the rotation around the x-axis:

$$\mathbf{q}_x = (\cos\frac{\theta}{2}, \mathbf{u}\sin\frac{\theta}{2}), \mathbf{u} = (1,0,0)^\top$$

Rotation around y-axis:

$$\mathbf{q}_y = (\cos\frac{\theta}{2}, \mathbf{u}\sin\frac{\theta}{2}), \mathbf{u} = (0,1,0)^\top$$

Rotation around z-axis:

$$\mathbf{q}_z = (\cos\frac{\theta}{2}, \mathbf{u}\sin\frac{\theta}{2}), \mathbf{u} = (0,0,1)^\top$$

Euler Angles v.s. Quaternions

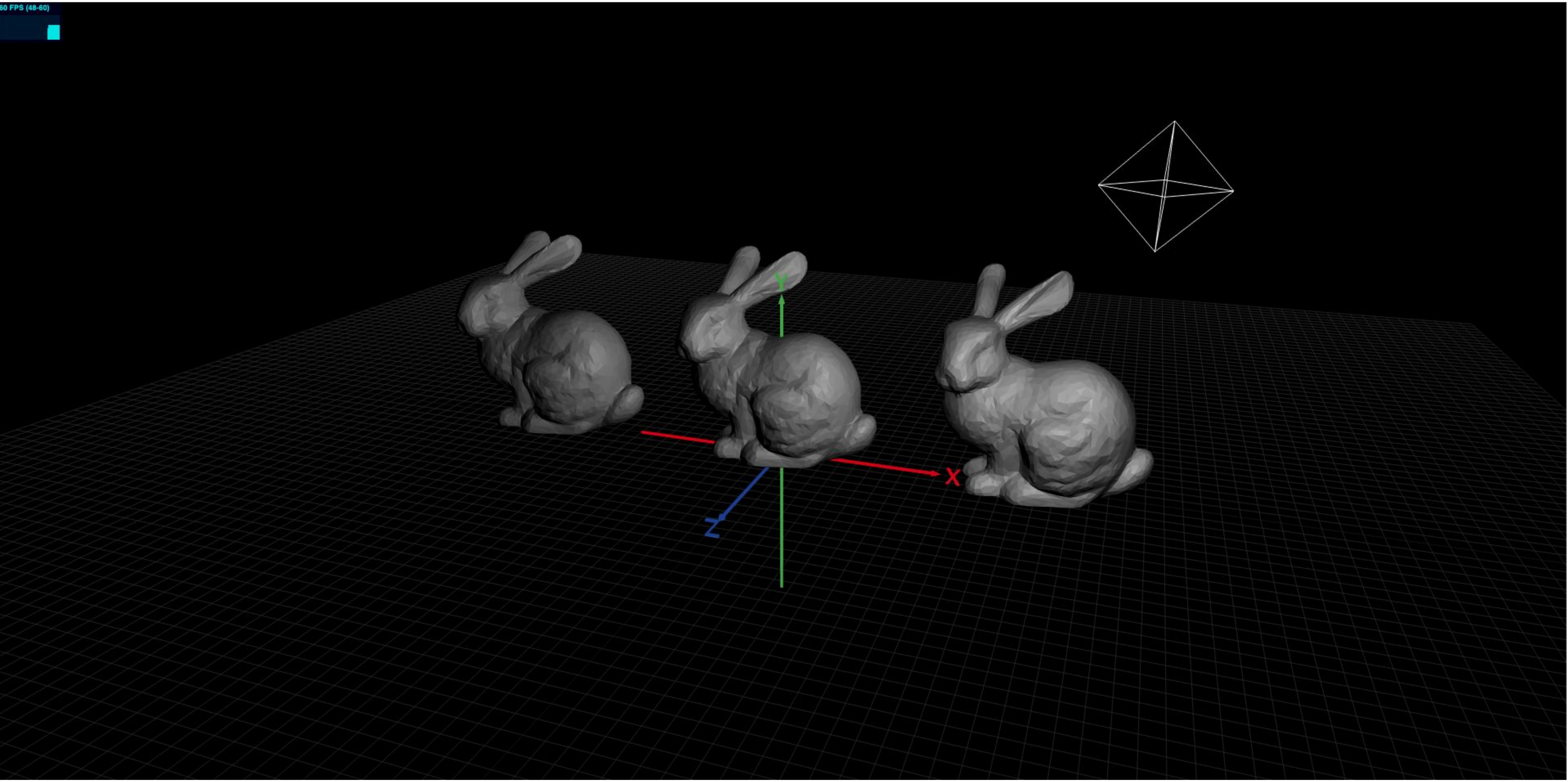
- Quaternions are much less intuitive than Euler angles.
- However, the rotation around an arbitrary direction is much easier to express using quaternions than using Euler angles.

  (Think about how to determine Euler angles in this case)

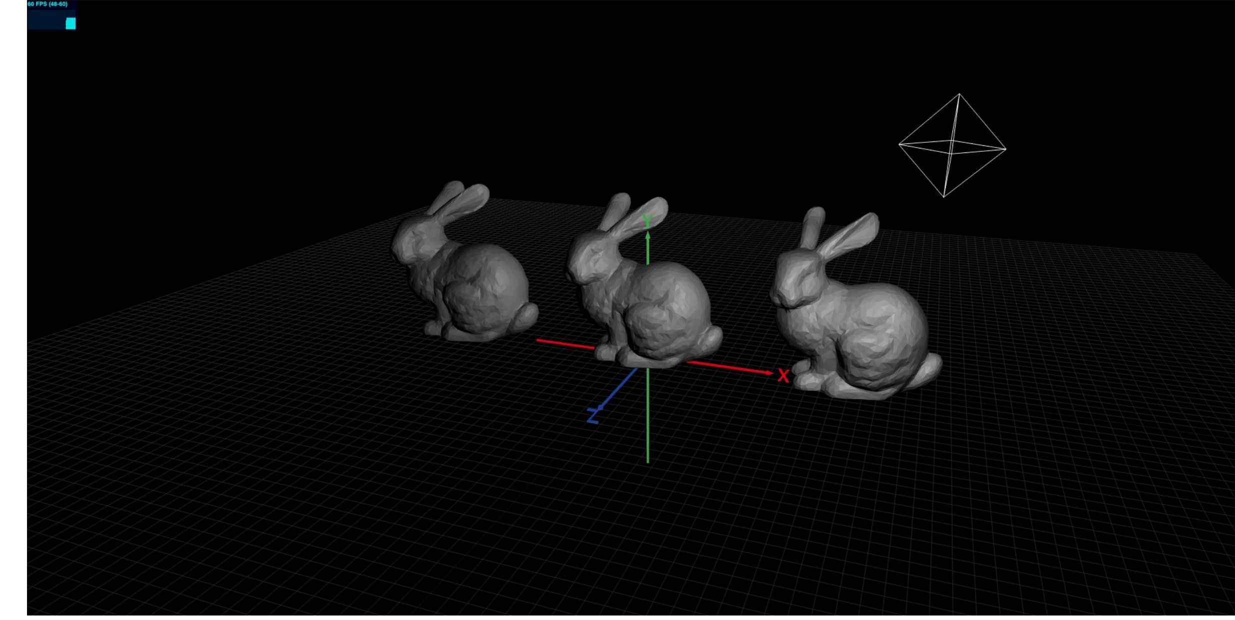
Tutorial 2: Transformations

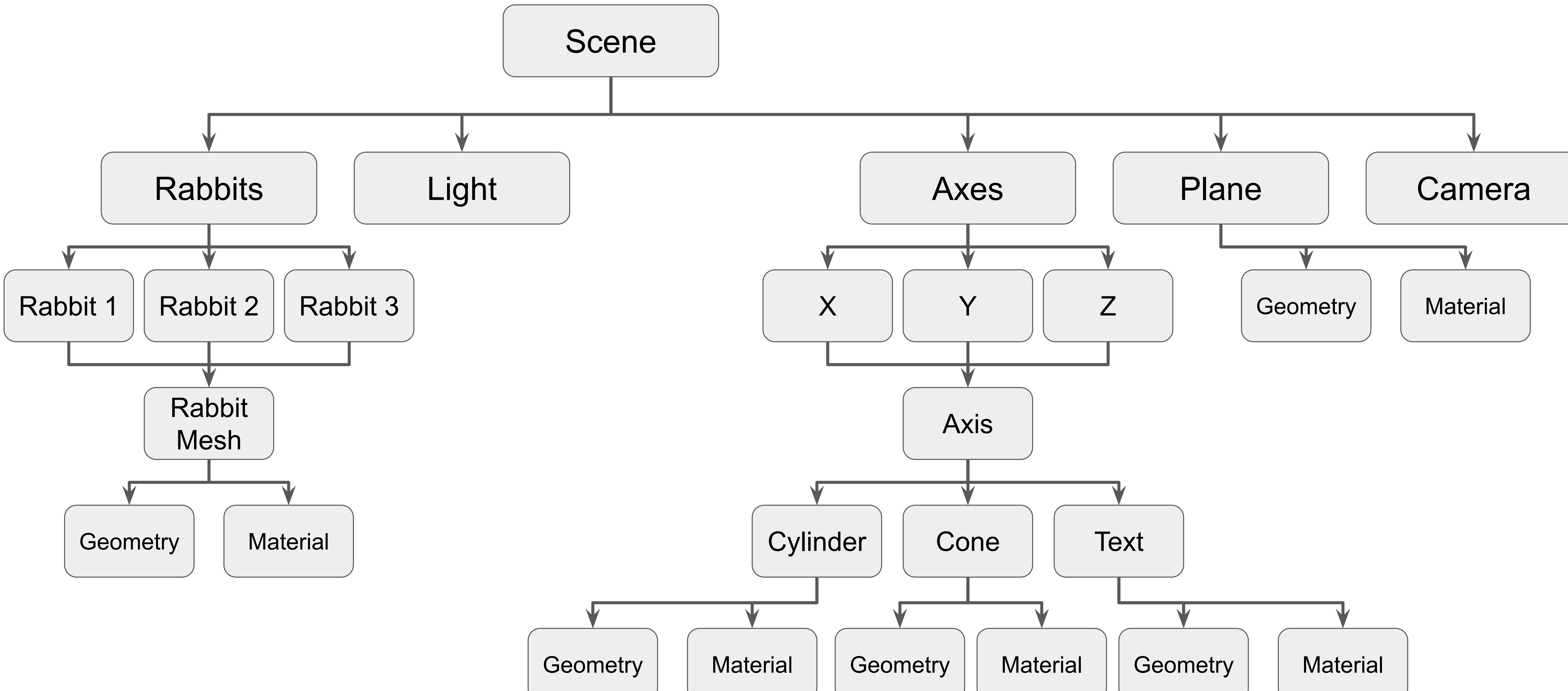
- Homogeneous Coordinates
- Affine Transformations
- 3D Rotation: Euler Angles
- 3D Rotation: Quaternions
- Scene Graph
- three.js

What's in the scene?



Task 4 a)

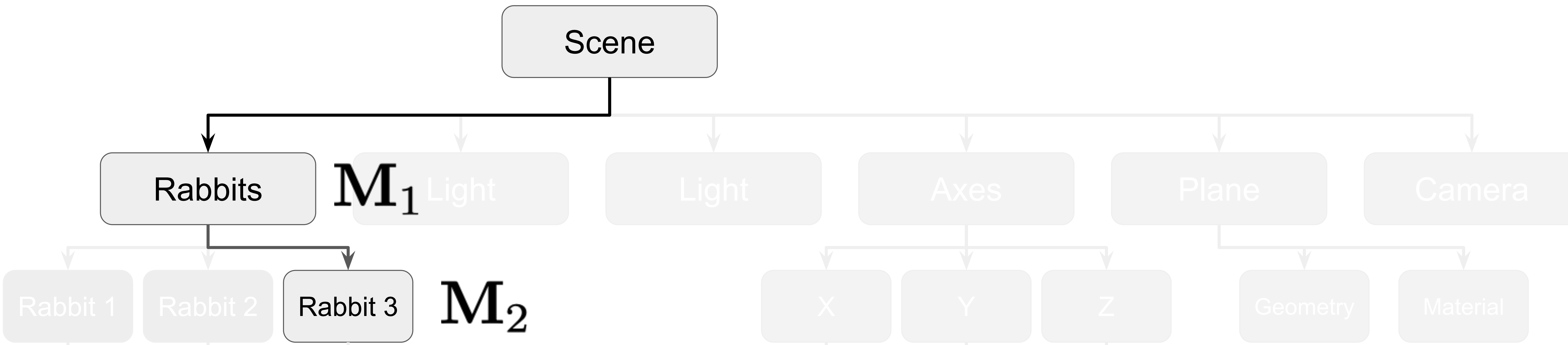



Scene
Rabbits
Light
Axes
Plane
Camera
Rabbit 1
Rabbit 2
Rabbit 3
X
Y
Z
Geometry
Material
Rabbit Mesh
Axis
Geometry
Material
Cylinder
Cone
Text
Geometry
Material
Geometry
Material
Geometry
Material

Task 4 b)

- Each node has its own transformation matrix
- If a node is transformed, all child nodes are also transformed

Task 4 c)



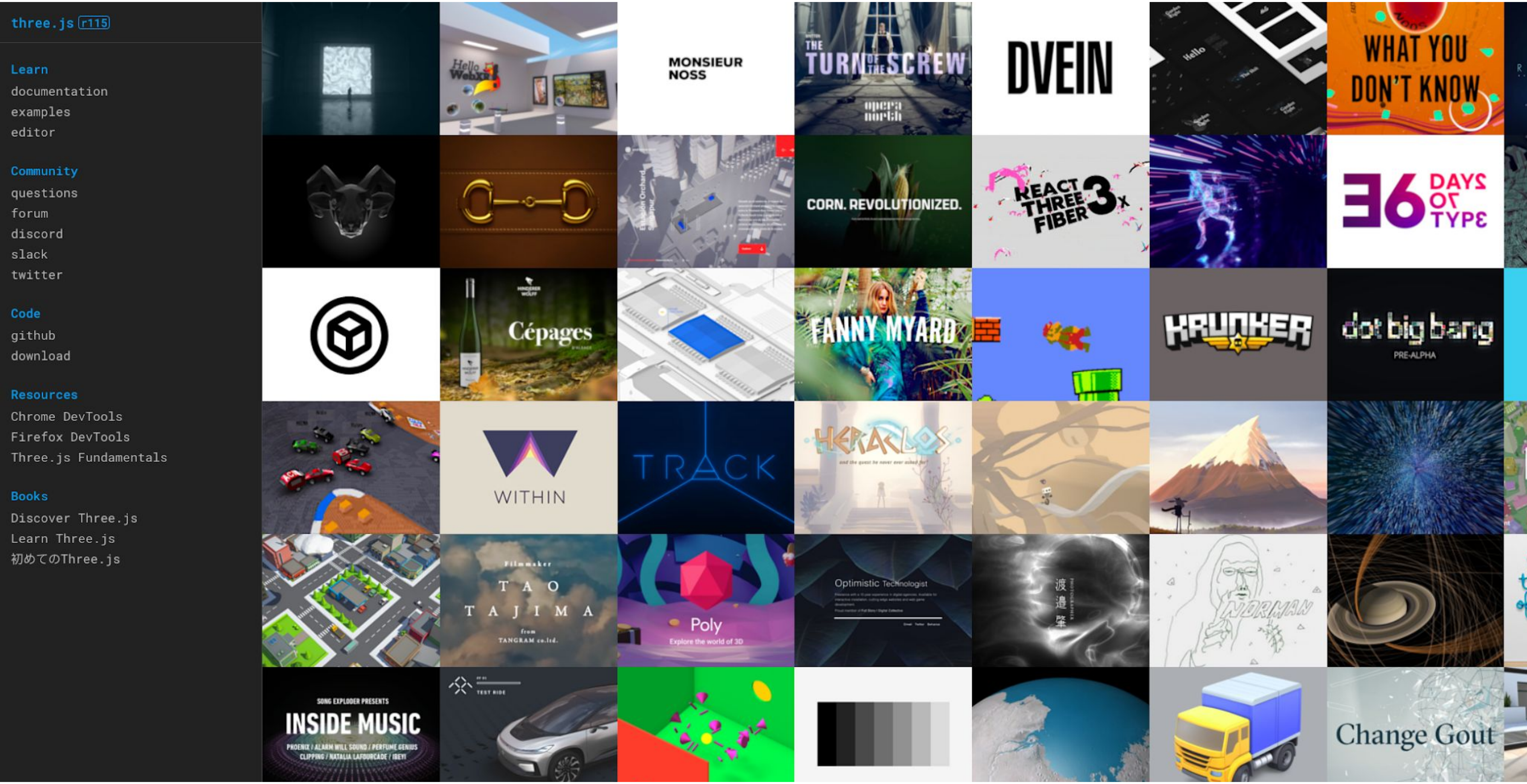
- The position of an object is calculated by multiplying it with all transformations along the path from the root: $\mathbf{M}_1\mathbf{M}_2\mathrm{P}$
- Assume the *rabbit 3* is translated from *rabbit 2* (origin), then the translation is from origin along with x-axis, then the transformation matrix is

$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Tutorial 2: Transformations

- Homogeneous Coordinates
- Affine Transformations
- 3D Rotation: Euler Angles
- 3D Rotation: Quaternions
- Scene Graph
- three.js

three.js: A JavaScript 3D library



Why didn't we choose XYZ?

- *"JavaScript is a joke!"*
- *"Unity is more popular!"*
- *"I don't want write a single program!"*
- …

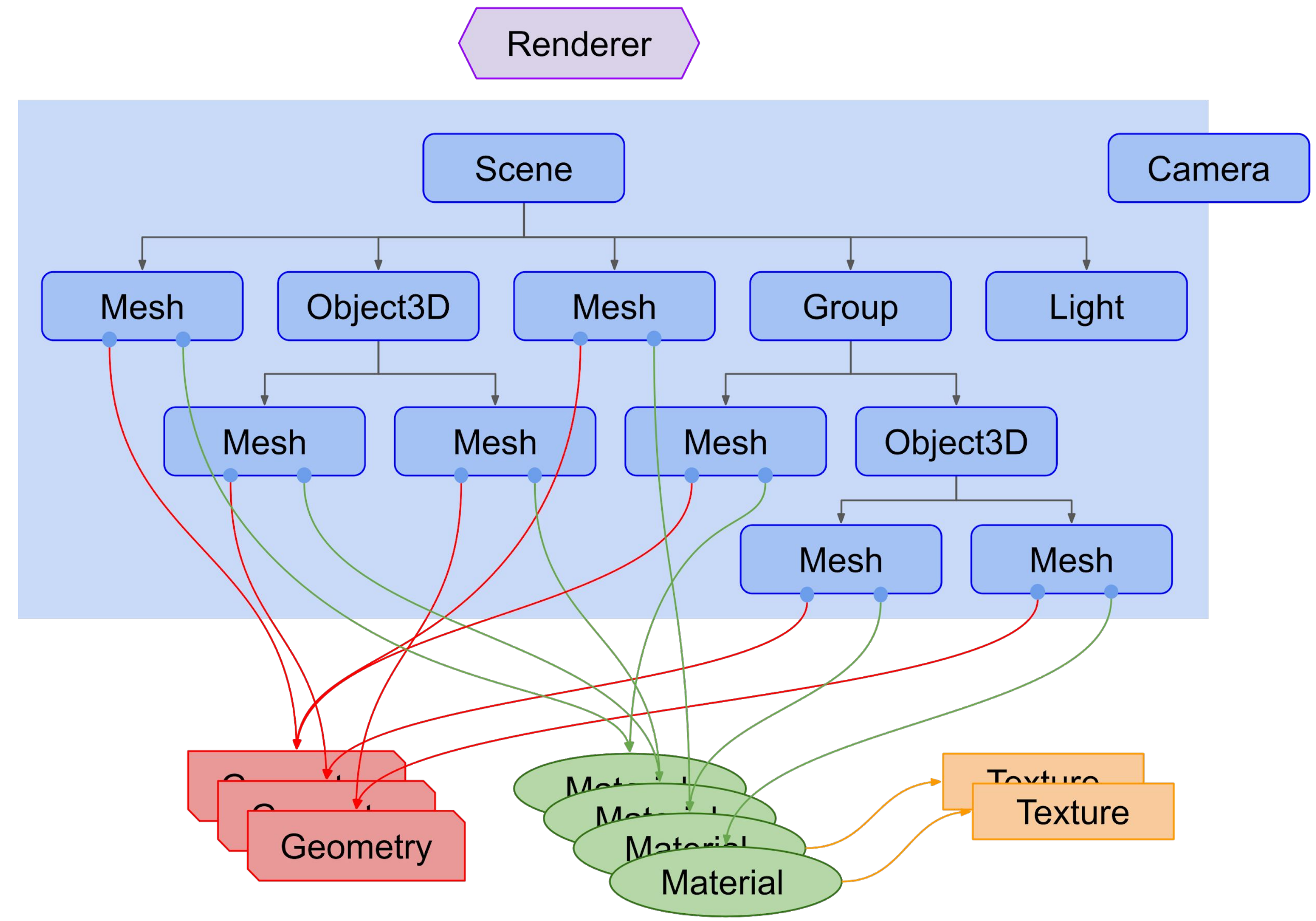
- OpenGL/WebGL/Vulkan are cross platform
- DirectX for Microsoft, and Metal for Apple
- Unity, Unreal, … are engines build on top of them
- …

- **You are in this course and we use *three.js*. That's it.**

three.js

The core concepts in three.js are:

- Renderer
- Scene
- Camera
- Mesh
- Geometry
- Material
- Light
- ...



Renderer and Scene

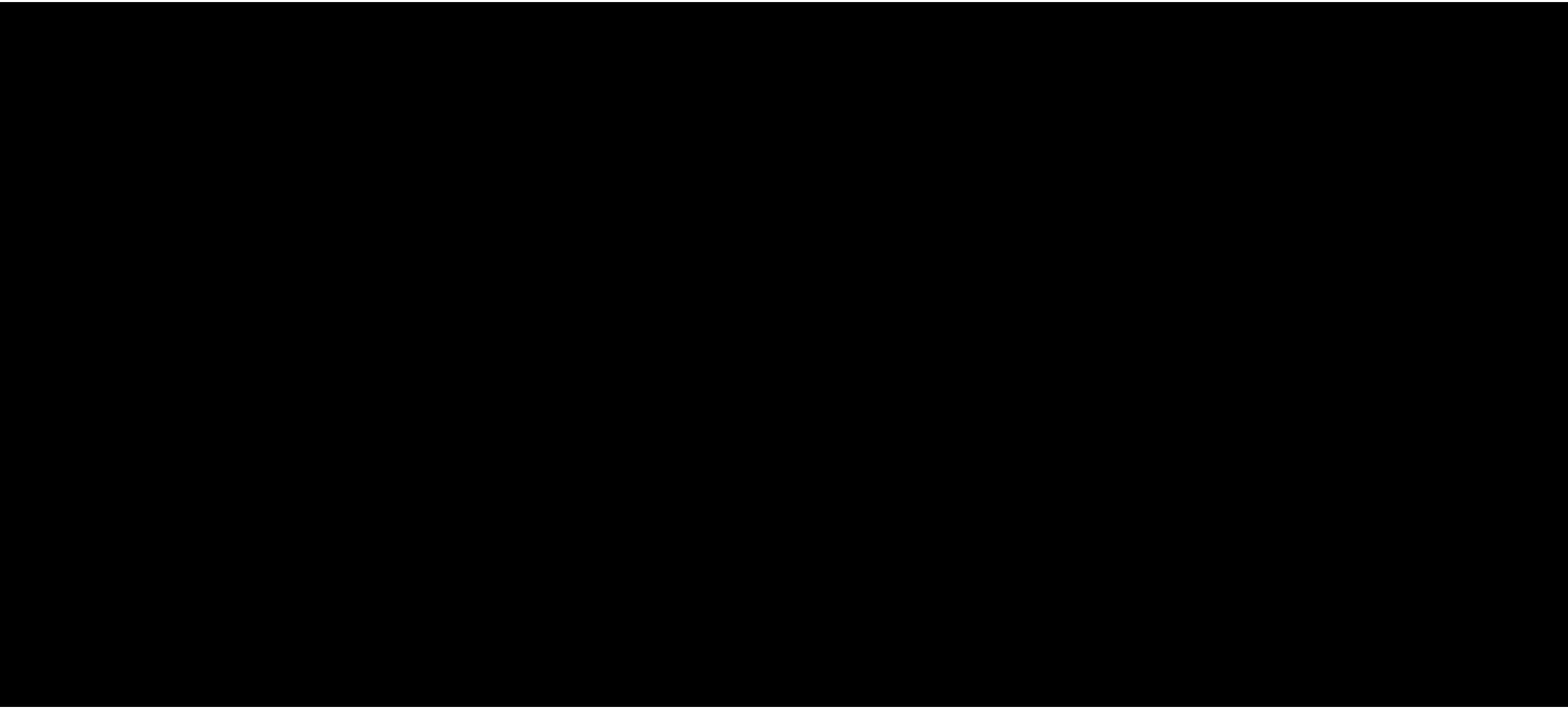
- A ***Renderer*** is created by `WebGLRenderer`, and renders your scene in the browser
- HTML manages everything using *DOM tree*, it's necessary to add renderer DOM element to the `document.body`. *Don't pay too much attention to how DOM works.*
- A ***Scene*** is created by `Scene`, and represents the scene graph
- The following code snippet is provided in the code skeleton:

Q: What happens if you don't pass `{antialias: true}` to the renderer?

```
constructor() {
  const container = document.body
  ...
  // 1. create renderer and add to the container
  this.renderer = new WebGLRenderer({ antialias: true })
  container.appendChild(this.renderer.domElement)
  // 2. create scene
  this.scene = new Scene()
  ...
}
```

Task 4 d)

We have nothing except we create it.



What should be implemented first?

Our TODOs:

Camera	"eyes", ability to see
OrbitControl	Move around
Performance monitor	Am I slow?
Render loop	Your "brain": processes what you got
Grid plane	An object, tells where you stand on
Axes	Reference, tells where are you
Light	Lights up everything
Rabbits	Living beings

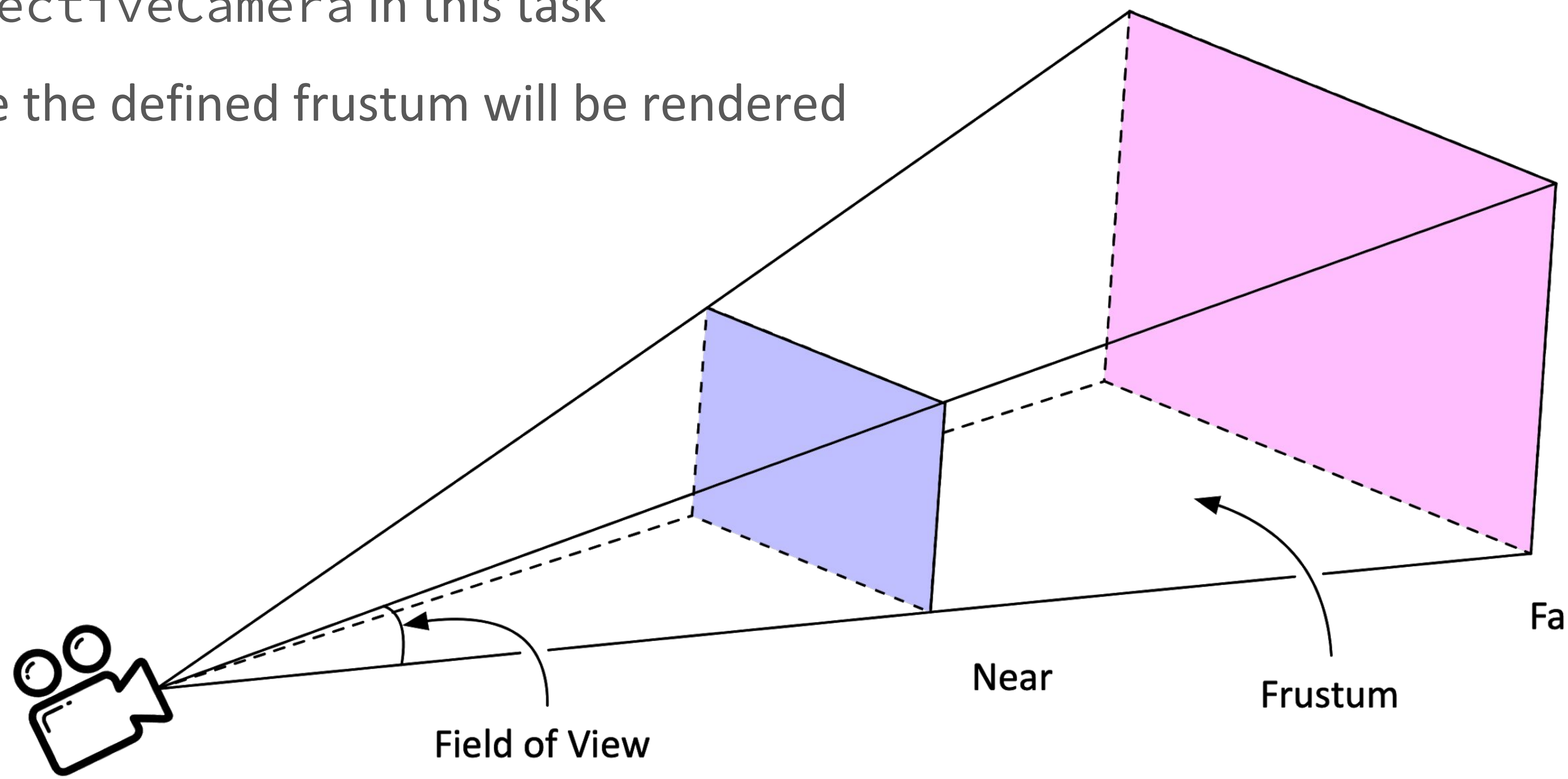
What should be implemented first?

Our TODOs:

- Camera ← **1. "eyes", ability to see**
- OrbitControl ← Move around
- Performance monitor ← Am I slow?
- Render loop ← **2. Your "brain": processes what you got**
- Grid plane ← **3. An object, tells where you stand on**
- Axes ← Reference, tells where are you
- Light ← Lights up everything
- Rabbits ← Living beings

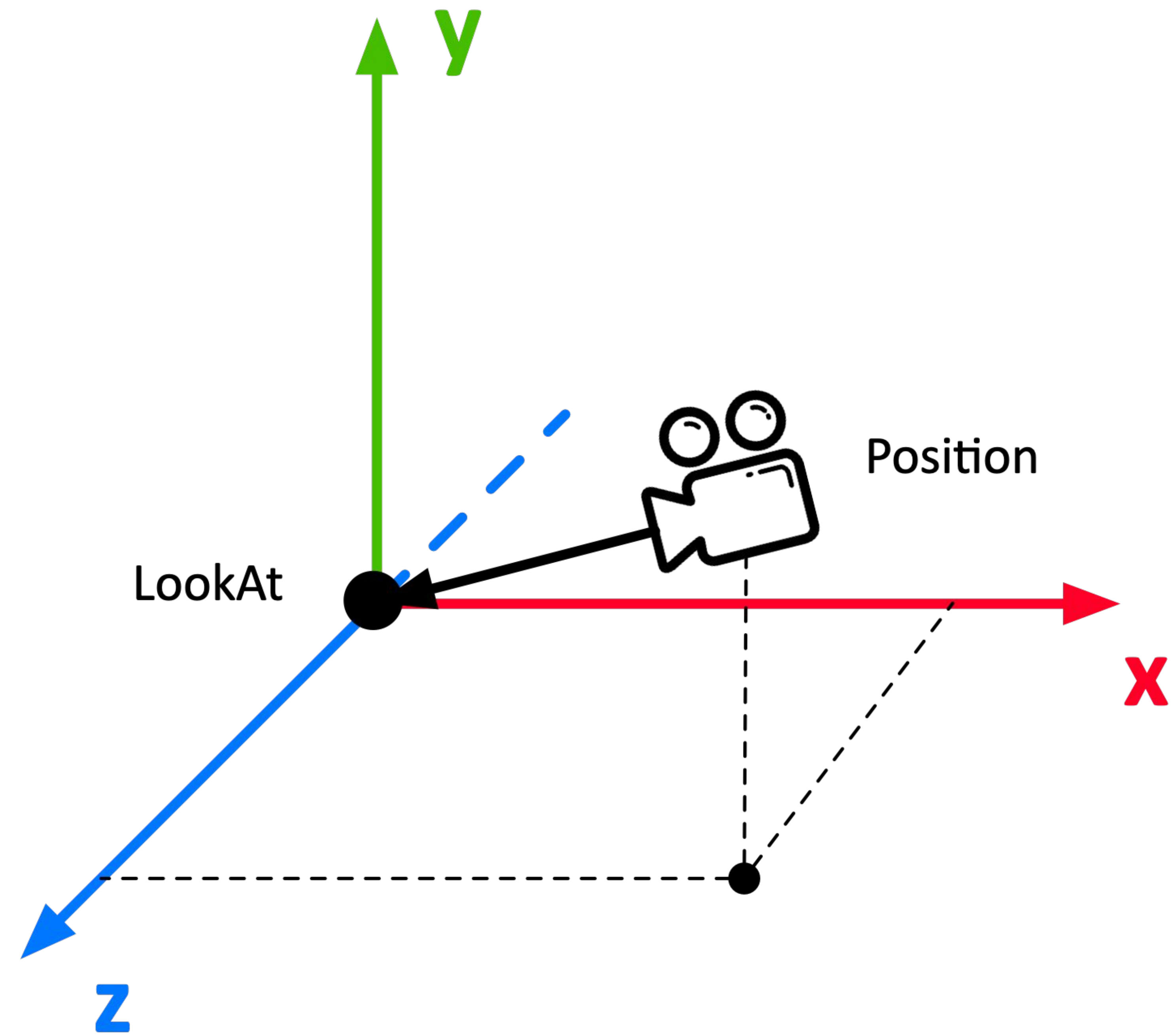
Camera

- There are many ***different types of camera (later)***
- We use `PerspectiveCamera` in this task
- Anything inside the defined frustum will be rendered



1. Create A Camera

```
constructor() {
  ...
  const cameraParam = {
    fov: 50,
    aspect: window.innerWidth / window.innerHeight,
    near: 0.1,
    far: 2000,
    position: new Vector3(10, 15, 25),
    lookAt: new Vector3(0, 0, 0),
  }
  // TODO: create a camera
  this.camera = new PerspectiveCamera(
    cameraParam.fov, cameraParam.aspect, cameraParam.near, cameraParam.far)
  this.camera.position.copy(cameraParam.position)
  this.camera.lookAt(cameraParam.lookAt)
  ...
}
```



Animation Frames

- An animation is a series of rendered images, `requestAnimationFrame` is a request to the browser that you want animate something.
- The `animate` callback will be executed by the browser if anything is updated (loop occurs)
- The `Renderer.render()` draws the scene according to the camera's definition

2. Implement Render Loop

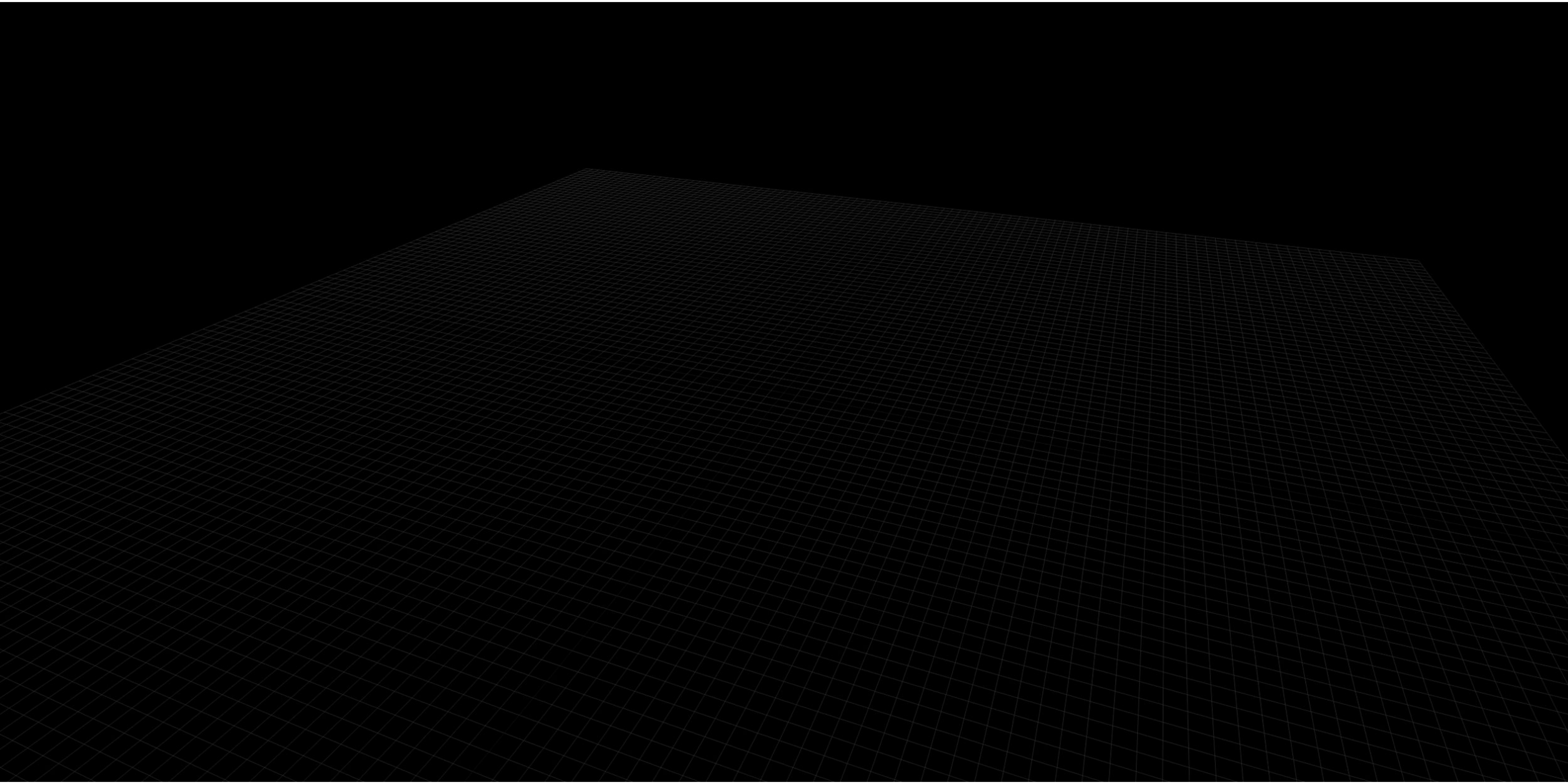
```
render() {
  this.renderer.setSize(window.innerWidth, window.innerHeight)
  const animate = () => {
    window.requestAnimationFrame(animate)
    // TODO: complete render loop for renderer
    this.renderer.render(this.scene, this.camera)
  }
  animate()
}
```

3. GridPlane

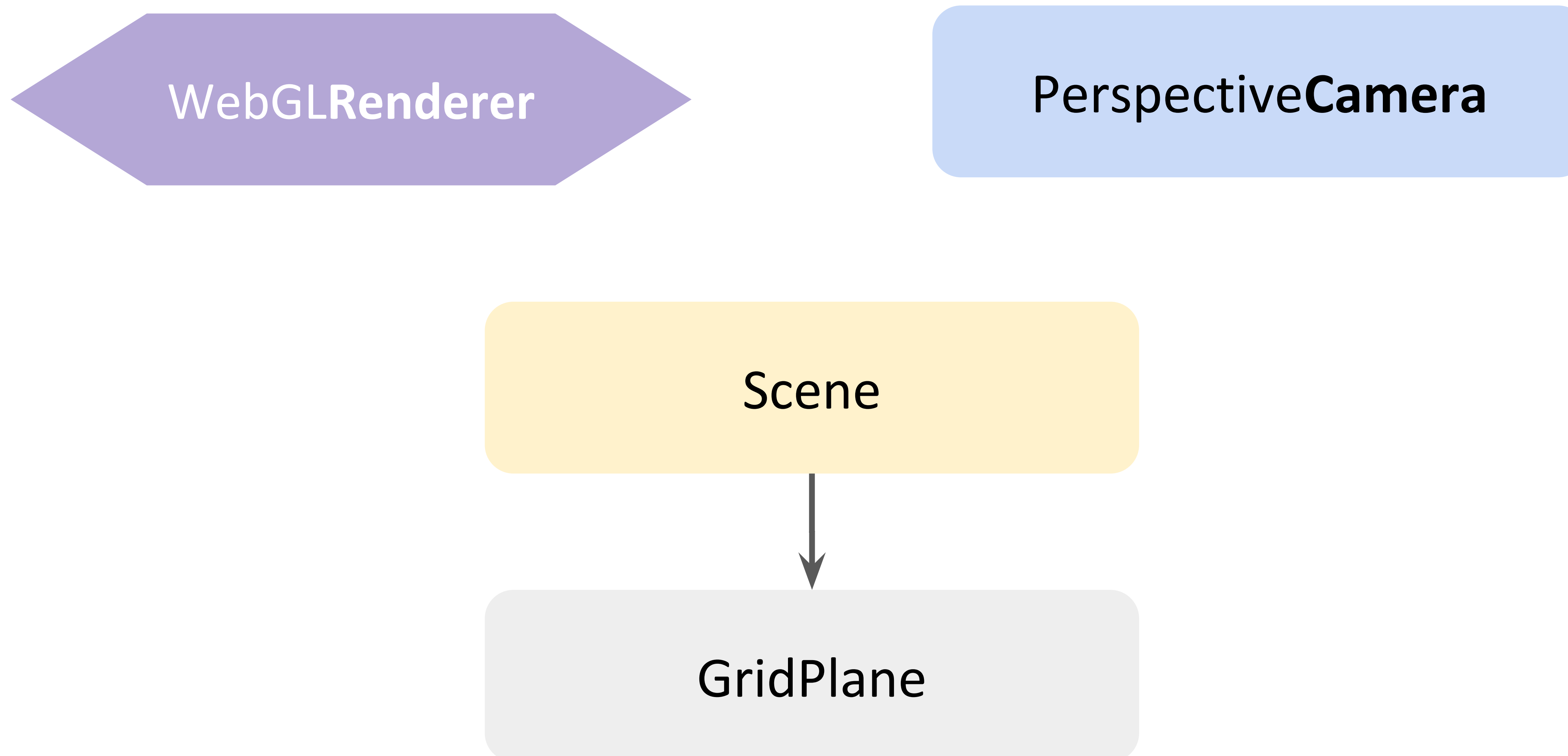
- three.js offers us several helpers to debug our rendering
- `GridPlane` creates a plane with grids

```
setupGridPlane() {
  const gridParam = {
    size: 50,
    divisions: 100,
    materialOpacity: 0.25,
    materialTransparency: true,
  }
  // TODO: create a GridHelper then add it to the scene.
  const helper = new GridHelper(gridParam.size, gridParam.divisions)
  helper.material.opacity = gridParam.materialOpacity
  helper.material.transparent = gridParam.materialTransparency
  this.scene.add(helper)
}
```

Can you see the ground? Barely...



What we have so far...



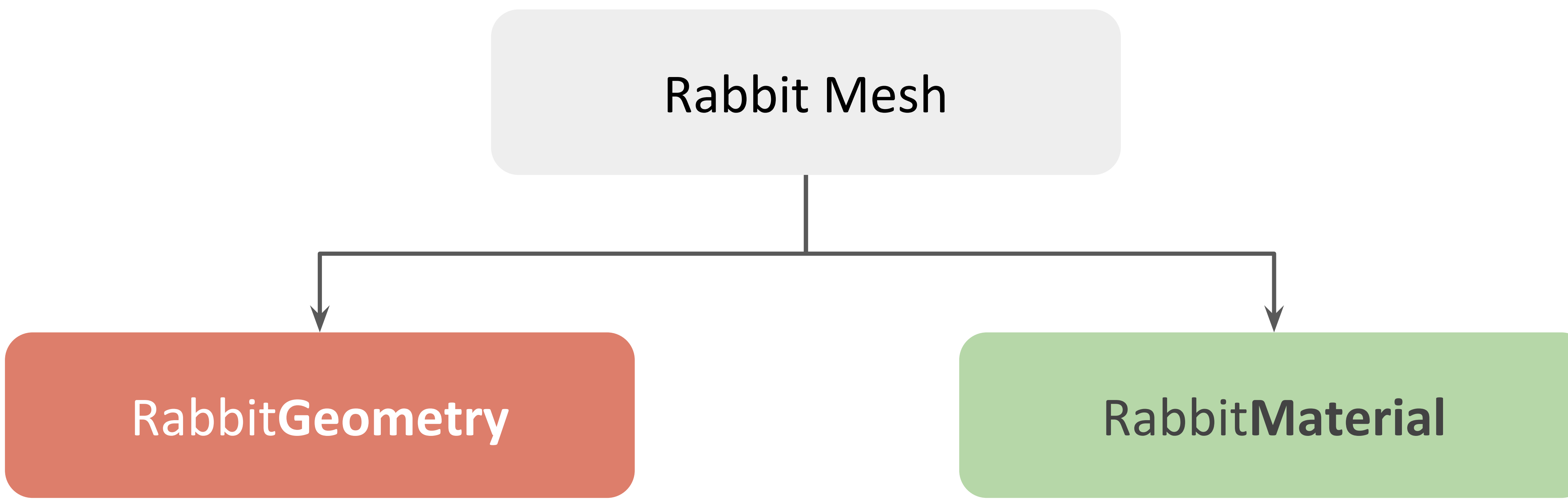
What should be implemented then?

Our TODOs:

- Camera ← 1. "eyes", ability to see
- OrbitControl ← Move around
- Performance monitor ← Am I slow?
- Render loop ← 2. Your "brain": processes what you got
- Grid plane ← 3. An object, tells where you stand on
- Axes ← Reference, tells where are you
- Light ← **2. Lights up everything**
- Rabbits ← **1. Living beings**

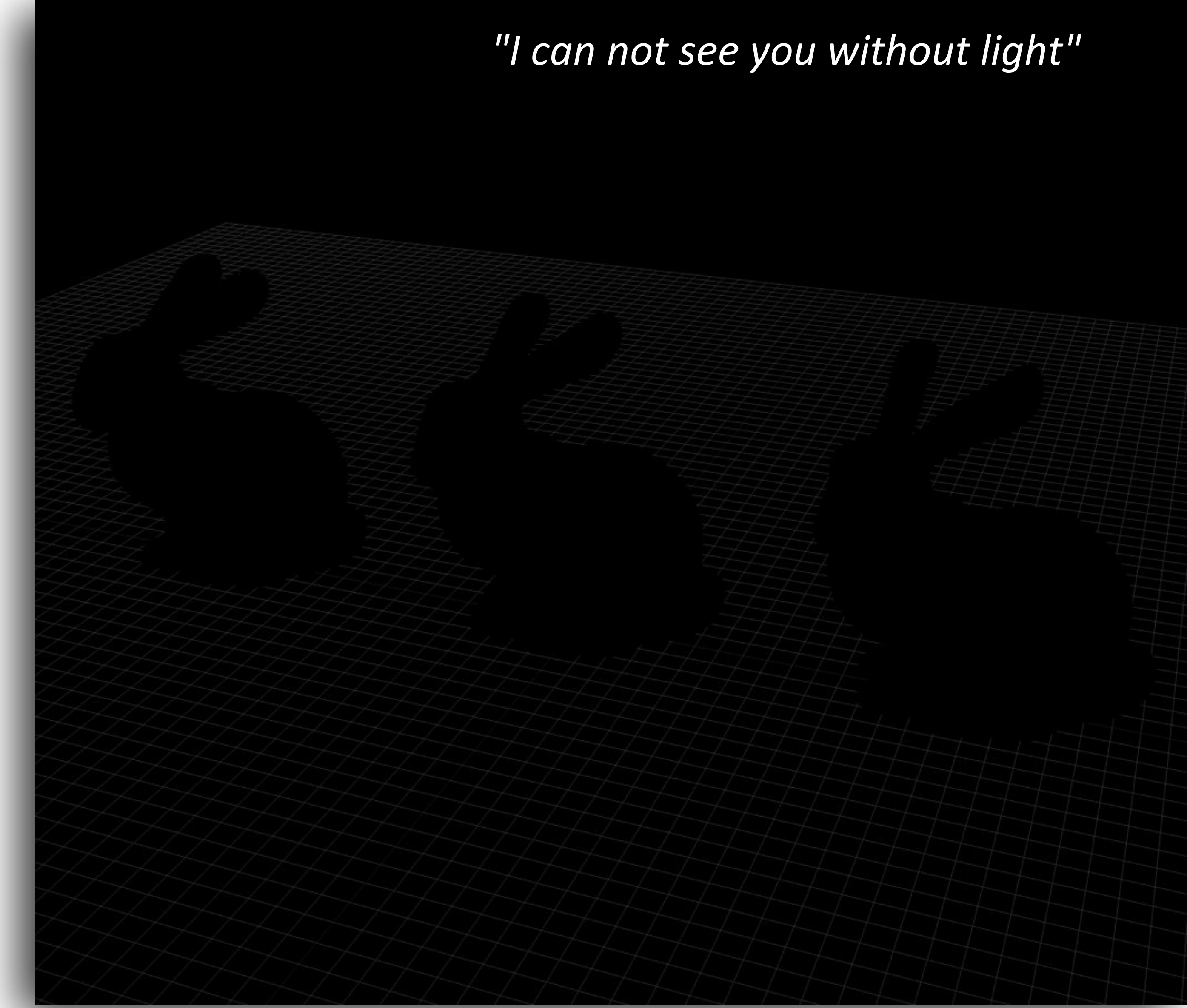
Mesh, Geometry and Material

- A Mesh represents a drawing Geometry with a specific Material
- Geometry includes vertices, edges, and etc
- A Material represents the properties of its corresponding geometry, e.g. rule of how the object is colored



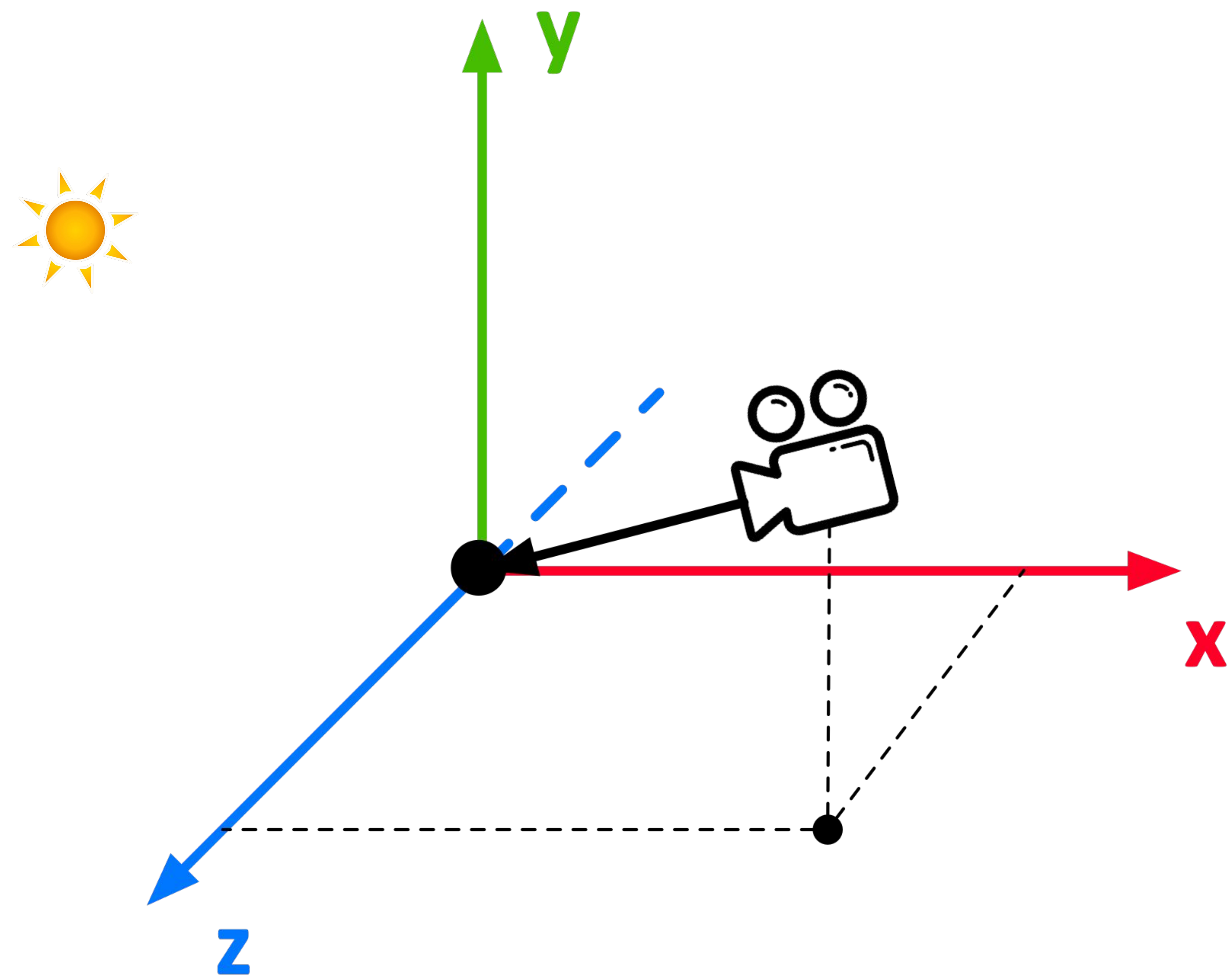
1. Create Rabbits

```
setupBunnies() {
  const rabbits = new Group()
  const loader = new GLTFLoader()
  loader.load('assets/bunny.glb', (model => {
    ...
    // TODO: duplicate, scale and translate the bunny model,
    //       then add it to the rabbits group.
    const mesh = model.scene.children[0]
    mesh.scale.copy(scale)
    rabbits.add(mesh,
      mesh.clone().translateOnAxis(translate.axis, translate.distance),
      mesh.clone().translateOnAxis(translate.axis, -translate.distance)
    )
  }).bind(this))
  this.scene.add(rabbits)
}
```



Light

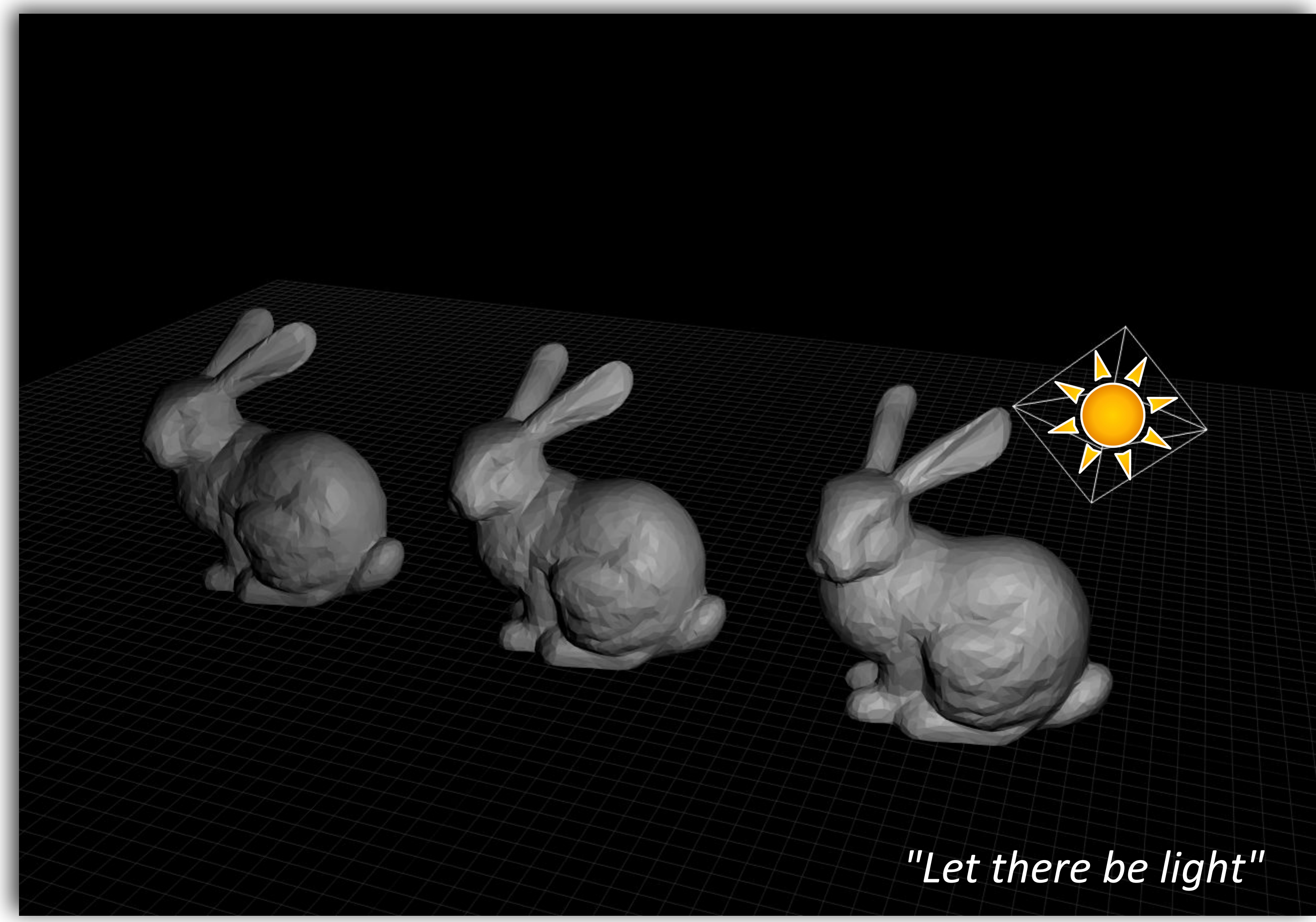
- Light represents ***different kinds of lights (later)***
- `PointLight` basic properties: color, strength/intensity, maximum range (distance)



2. Lights up

```
setupLight() {
  ...
  // TODO: 1. create a PointLight and add to the lightGroup
  const light = new PointLight(lightParams.color, lightParams.intensity, lightParams.distance);
  light.position.copy(lightParams.position)
  lightGroup.add(light)
  // TODO: 2. create a PointLightHelper and add to the lightGroup
  const helper = new PointLightHelper(light)
  lightGroup.add(helper)

  this.scene.add(lightGroup)
}
```



What should be implemented then?

Our TODOs:

- Camera ← 1. "eyes", ability to see
- OrbitControl ← **Move around**
- Performance monitor ← **Am I slow?**
- Render loop ← 2. Your "brain": processes what you got
- Grid plane ← 3. An object, tells where you stand on
- Axes ← Reference, tells where are you
- Light ← 2. Lights up everything
- Rabbits ← 1. Living beings

OrbitControl & Performance Monitor

```
constructor() { ...
  // TODO: 4. setup orbit controls
  this.controls = new OrbitControls(this.camera, this.renderer.domElement)
  // TODO: 5. setup performance monitor
  this.stats = new Stats()
  this.stats.showPanel(0) // fps
  container.appendChild(this.stats.domElement)
}
render() { ...
  const animate = () => {
    window.requestAnimationFrame(animate)
    this.controls.update()
    this.stats.update()
    this.renderer.render(this.scene, this.camera)
  }
  animate()
}
```

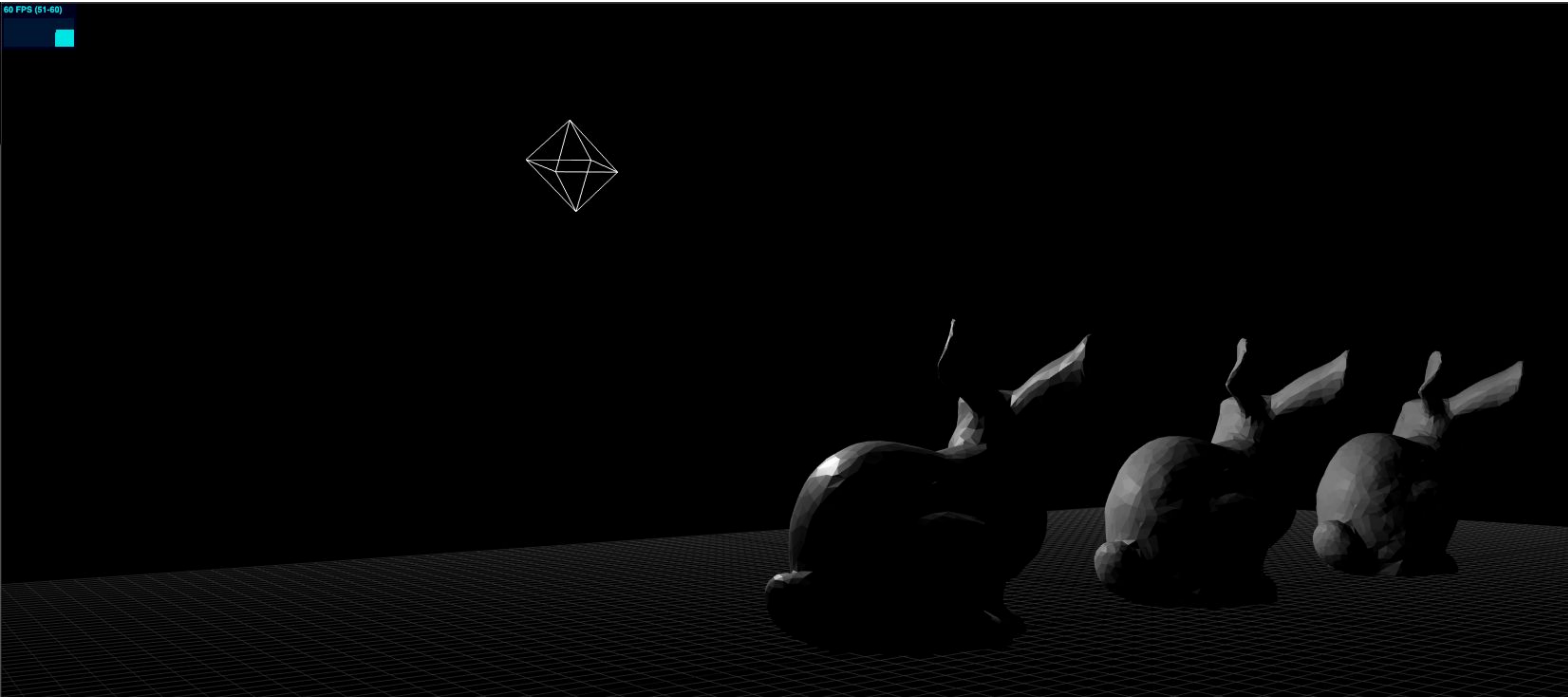
OrbitControls

- Orbits the camera around a target
- Update camera position in render loop

Performance Monitor

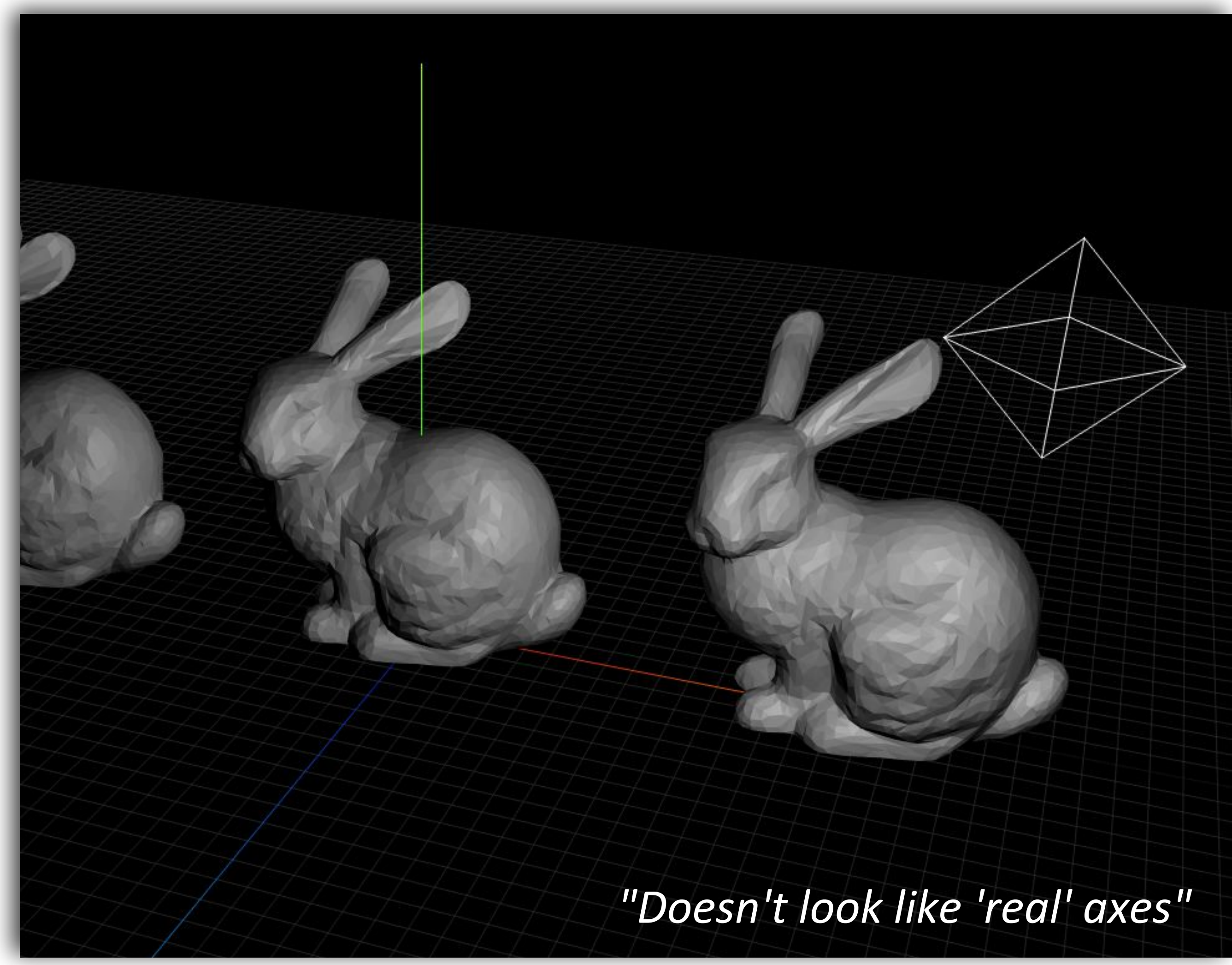
- Things we care about
  - FPS: Frame per second
  - MS: Milliseconds needed to render a frame
  - ...

OrbitControl & Performance Monitor



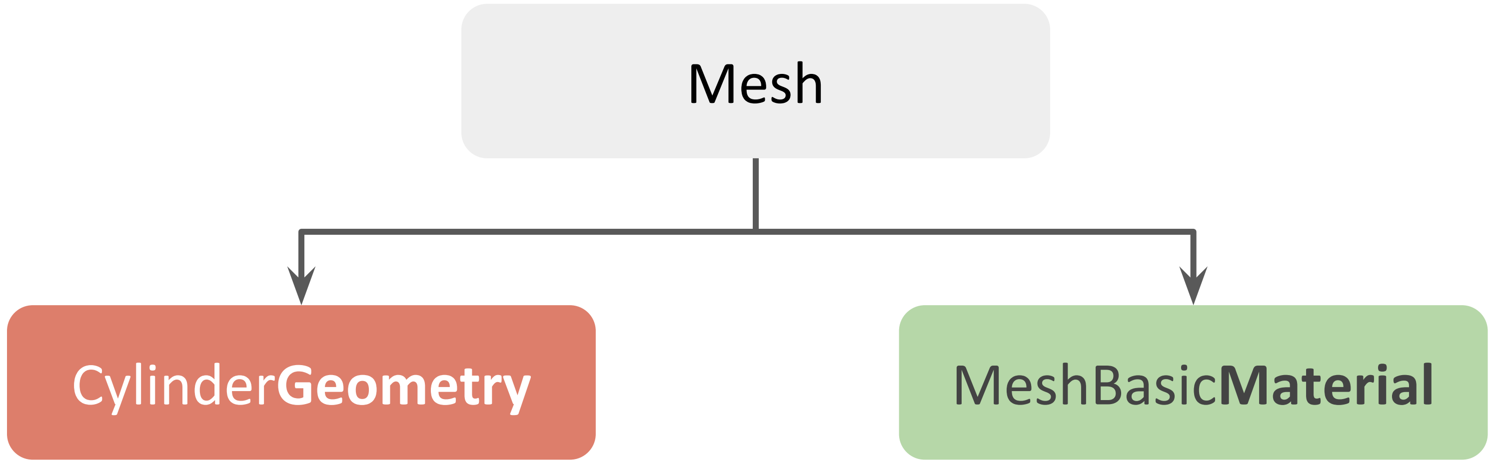
Axes

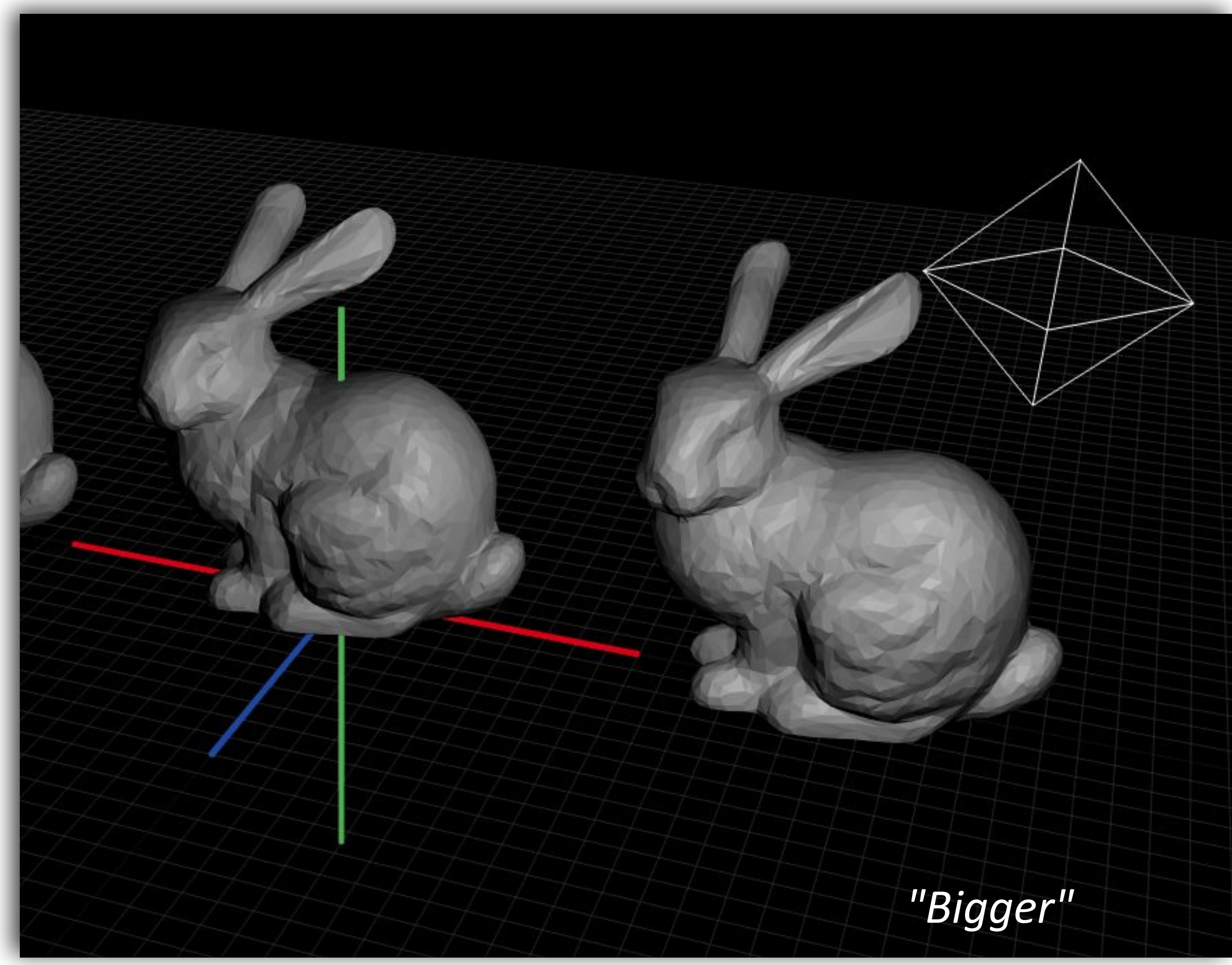
```
setupAxes() {
  // You can comment out the following two lines to get an ugly
  // three.js built-in axes.
  const axesHelper = new AxesHelper(10)
  this.scene.add(axesHelper)
  ...
}
```



(Visually) Better Axes

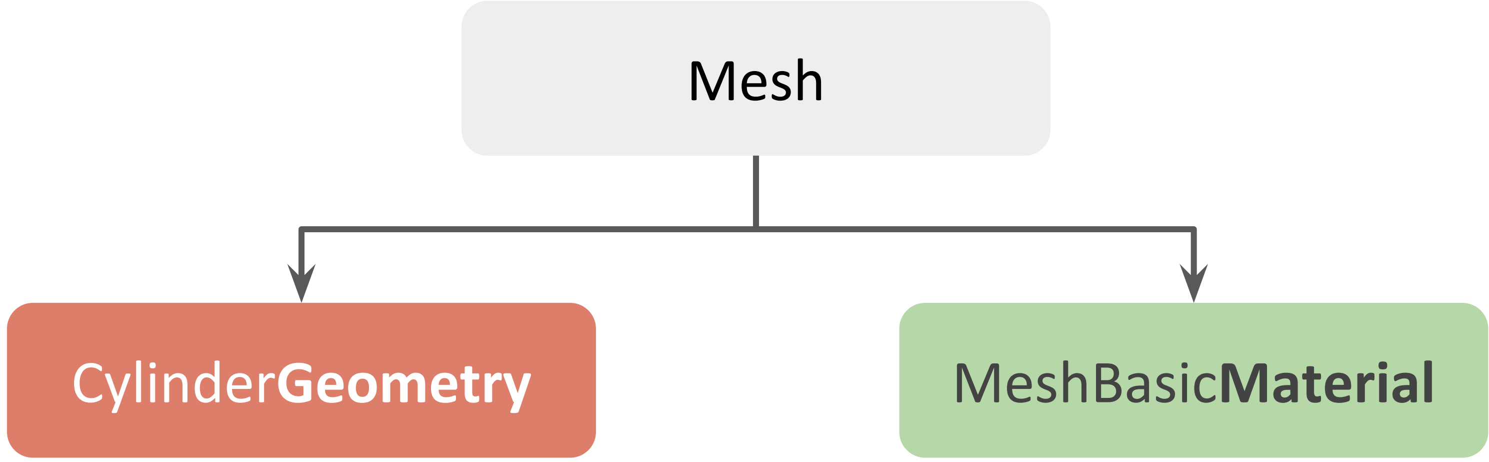
```
createAxis(label, euler, direction, color) {
  ...
  // TODO: 1. create the axis cylinder, use CylinderGeometry and MeshBasicMaterial
  const line = new Mesh(
    new CylinderGeometry(radius, radius, length, segments),
    new MeshBasicMaterial({ color: color })
  )
  line.setRotationFromEuler(euler)
  axis.add(line)
  ...
  return axis
}
```

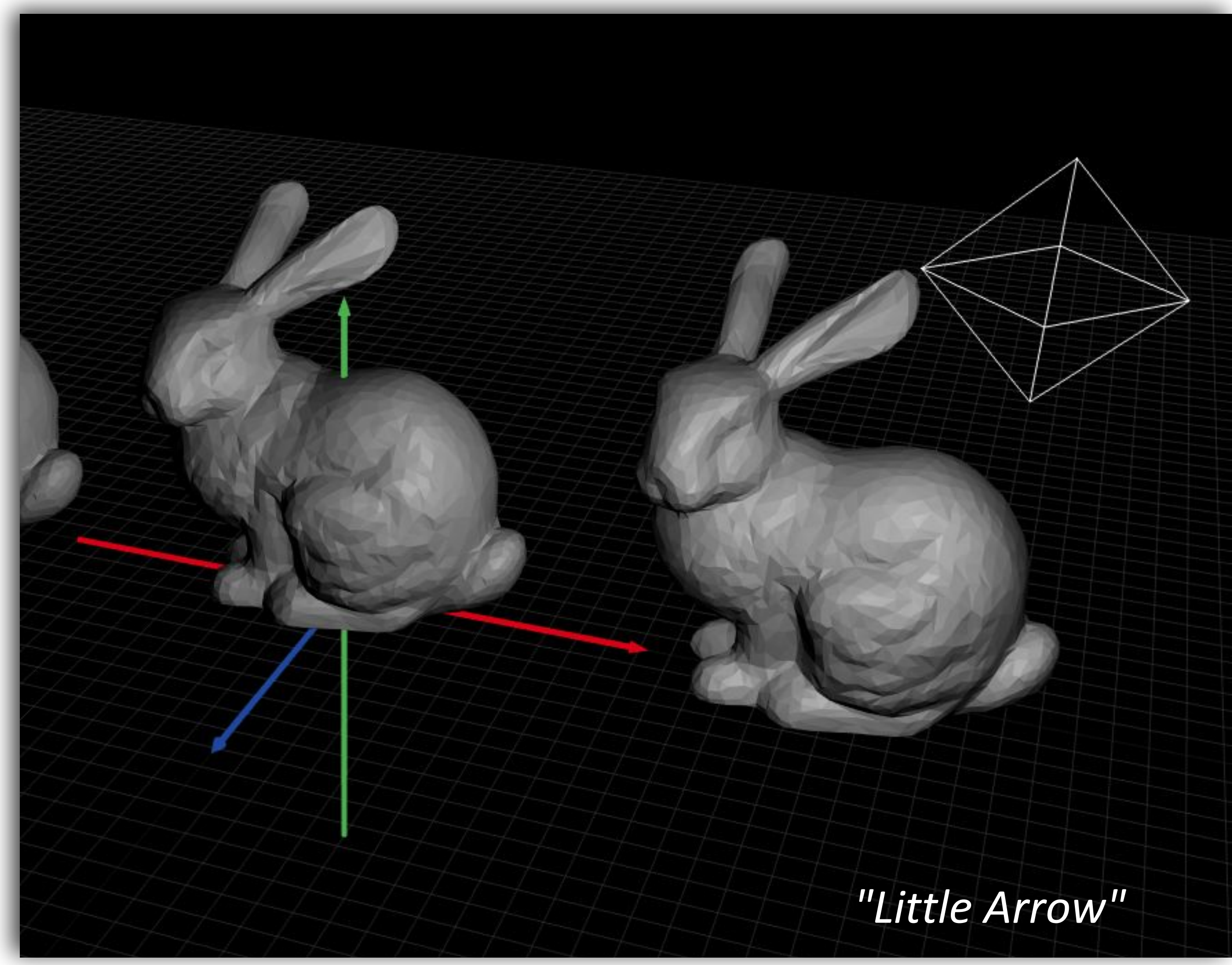




(Visually) Better Axes

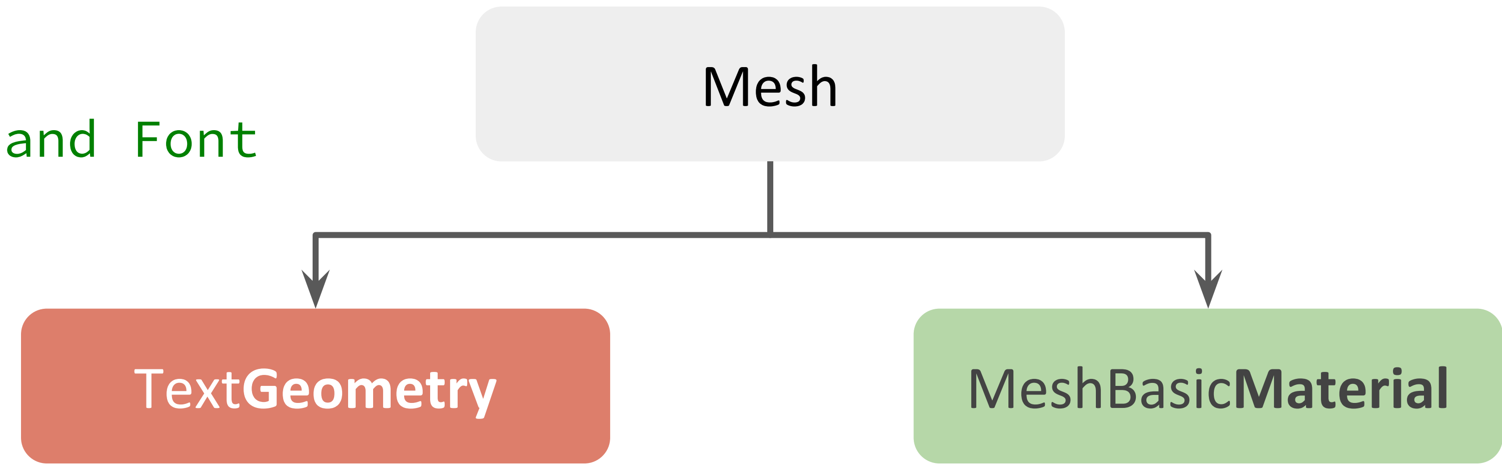
```
createAxis(label, euler, direction, color) {
  ...
  // TODO: 2. create the axis arrow (i.e. the cone), use CylinderGeometry and MeshBasicMaterial
  const arrow = new Mesh(
    new CylinderGeometry(0, radius*2, height, segments),
    new MeshBasicMaterial({ color: color })
  )
  arrow.translateOnAxis(direction, length/2)
  arrow.setRotationFromEuler(euler)
  axis.add(arrow)
  ...
  return axis
}
```

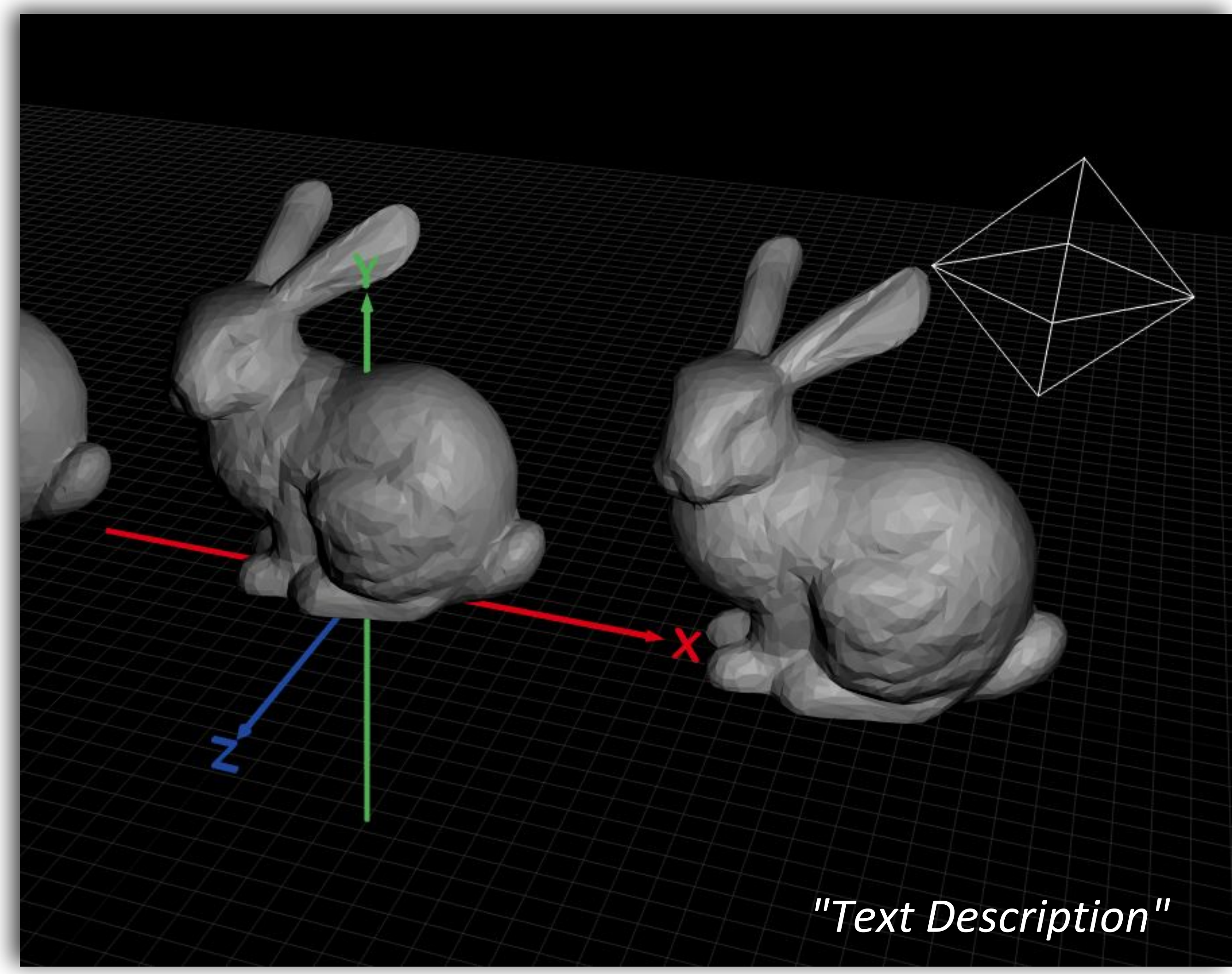




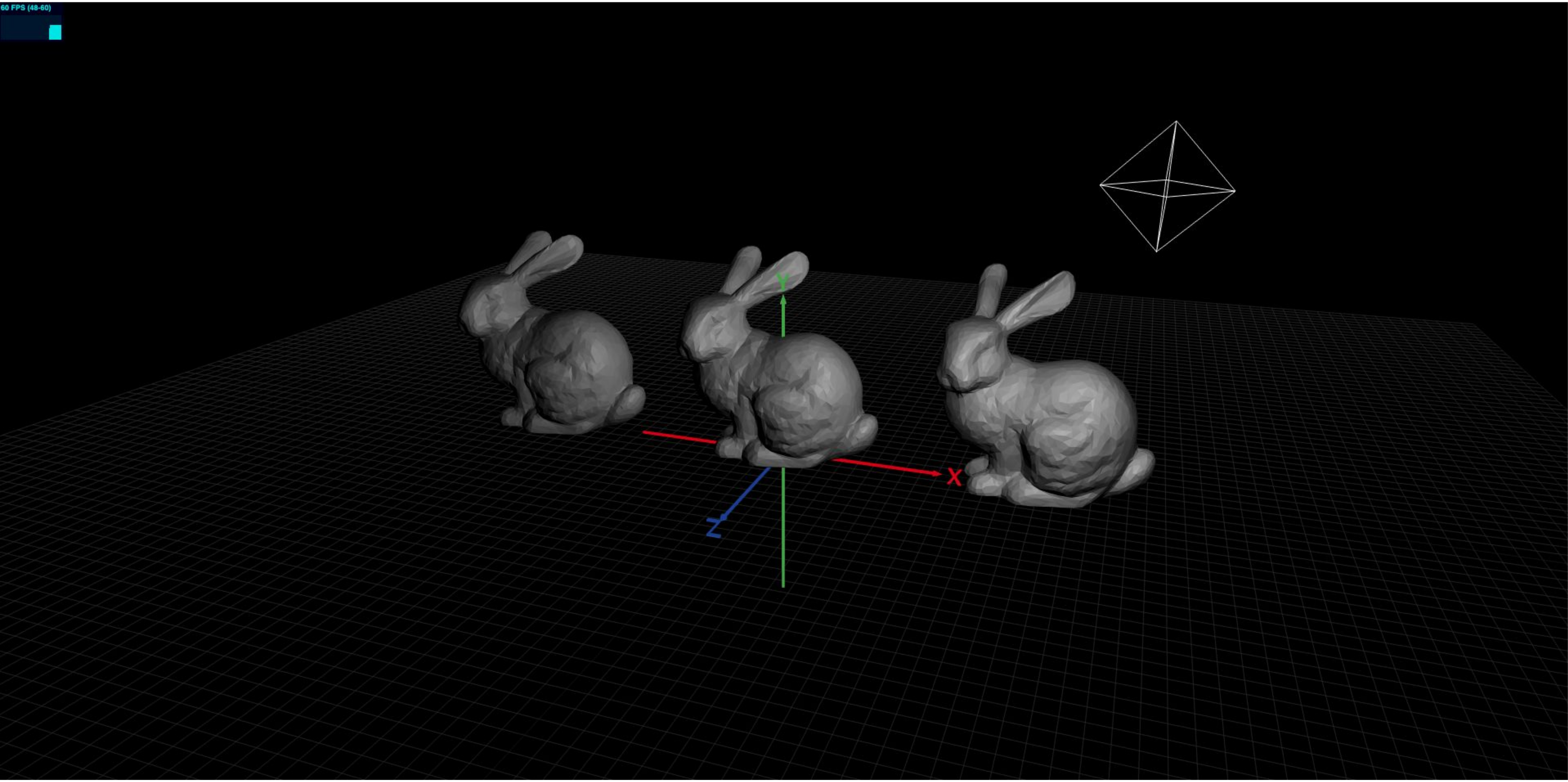
(Visually) Better Axes

```
createAxis(label, euler, direction, color) {
  ...
  // TODO: 3. create the text label, use TextGeometry and Font
  const text = new Mesh(
    new TextGeometry(label, {
      font: new Font(fontParam.object),
      size: fontParam.size,
      height: fontParam.height,
    }),
    new MeshBasicMaterial({ color: color }),
  )
  text.translateOnAxis(direction, length/2 + 0.5)
  text.translateOnAxis(new Vector3(-1, 0, 0), 0.2)
  text.translateOnAxis(new Vector3(0, -1, 0), 0.25)
  axis.add(text)
  return axis
}
```





Final



Task 4

If you implemented the scene, these questions can be easily answered:

e) Camera and Scene

f) Geometry and Material

g) Up

h) intrinsic Tait-Bryan angles, default XYZ.

Why is 3D programming supposed to be (ultra-)difficult?

- Math is absolutely important and really hard to most of people
- Geometric imagination is sometimes non-trivial, i.e. viewing 3D through 2D
- Details tuning is (super) time consuming, e.g. create an eyeball that is not just a sphere
- Runtime performance is critical if you desire real-time rendering, each frame should be rendered in 1s/60fps ≈ 16.67 ms
- Testing graphics can be painful, and most of the time need a person to "see" what went wrong
- No common industry standard or agreement, i.e. different platform with different APIs (DirectX v.s. Metal), and massive API changes over time (OpenGL: 1.1 != 2.1 != 3.3 != 4.6)
- Interdisciplinary. You might also need knowledge in physics, biology, and … more!

Don't panic! 🙂

Take Away

- Order of transformation matters
- 3D Rotation with Euler angles is intuitive but with limitations
- 3D Rotation with quaternion is non-trivial but simple/powerful to express
- In 3D programming, always think about how to test your implementation quickly
- Learn to read development documents

Thanks!
What are your questions?