Computer Graphics 1

4 Camera

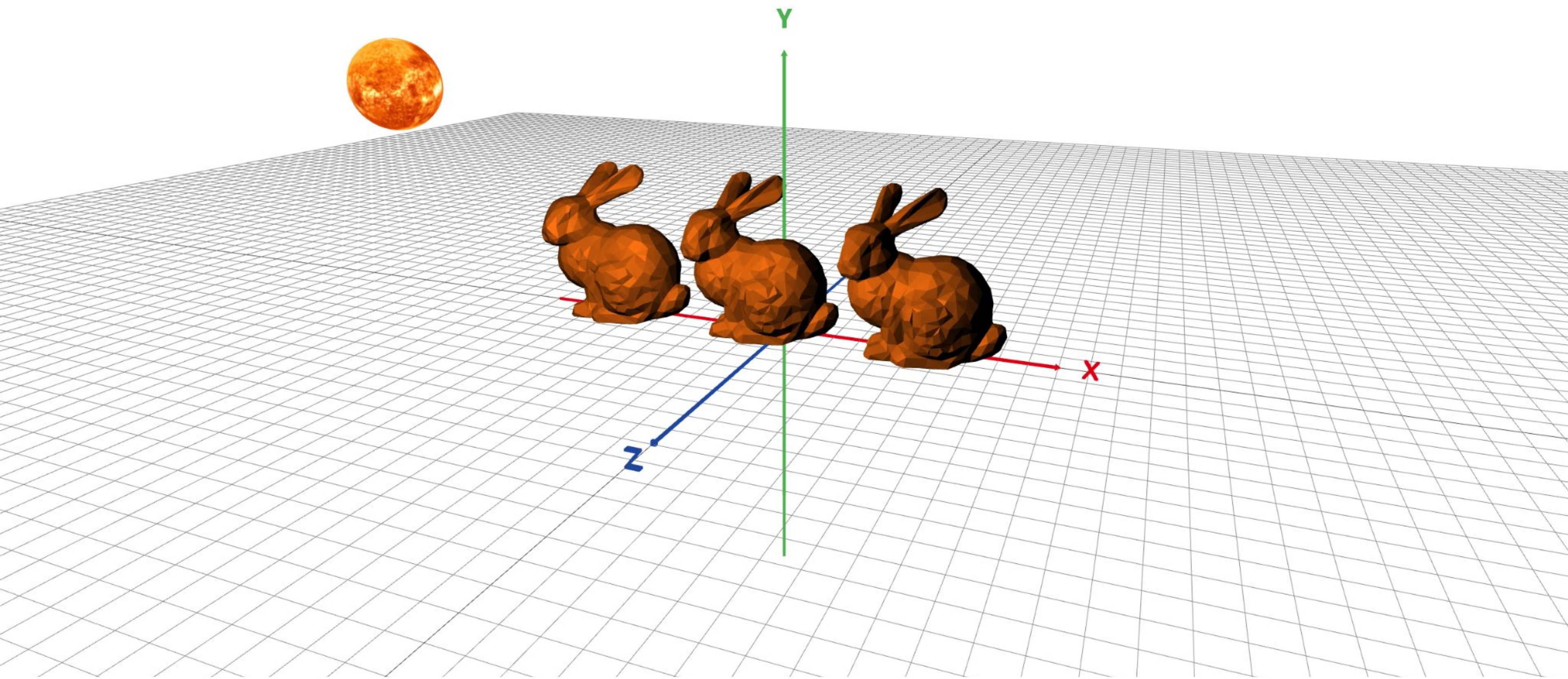
Summer Semester 2021

Ludwig-Maximilians-Universität München

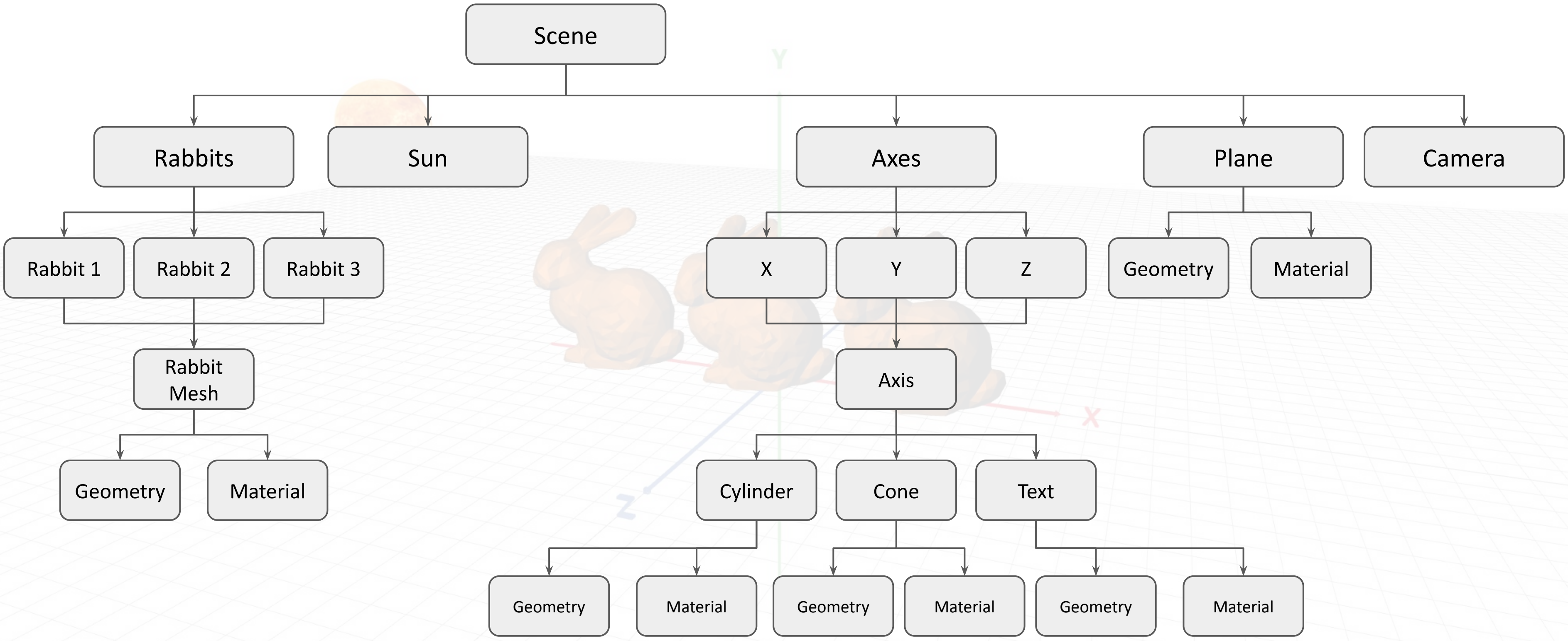
Tutorial 4: Camera

- Scene Graph and Model Transformation
- Viewing Transformations
  - View Transformation
  - Projection Transformations
  - Viewport Transformation
- Application: Hitchcock Zoom
- Summary

What's in the scene?

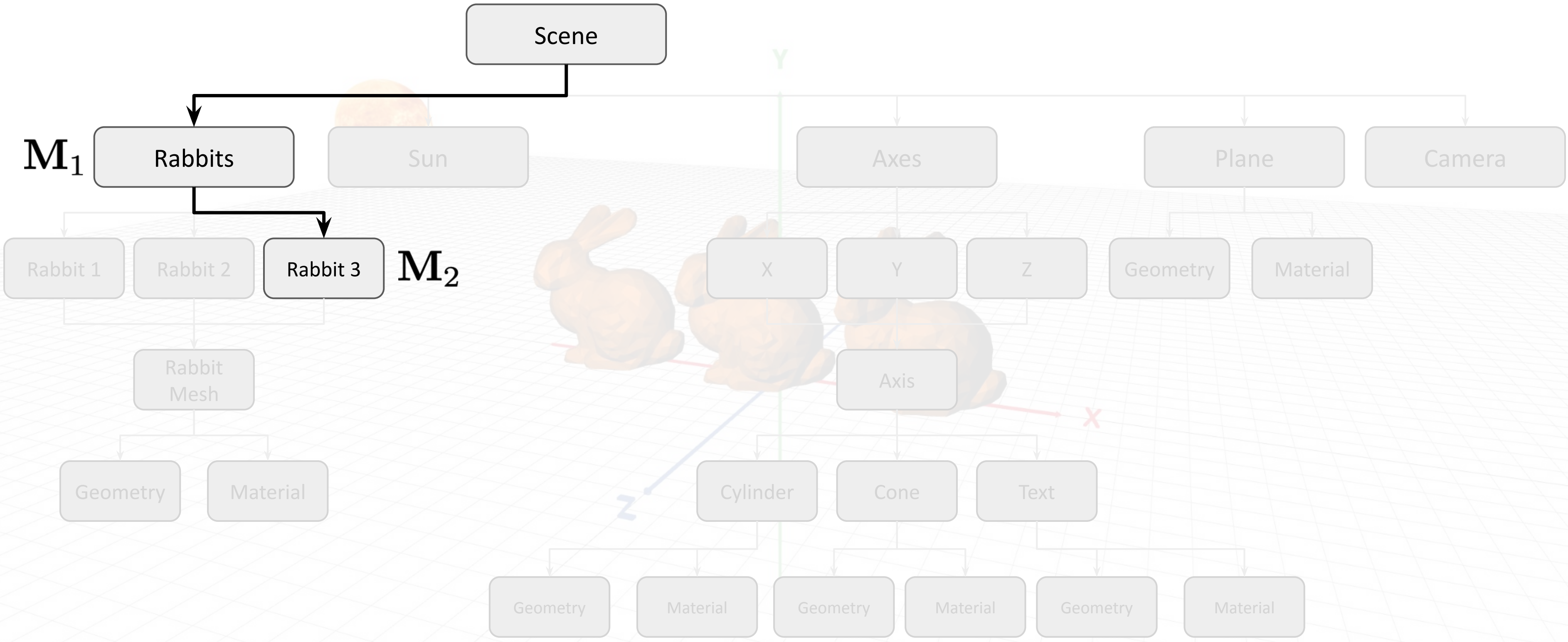


Scene Graph



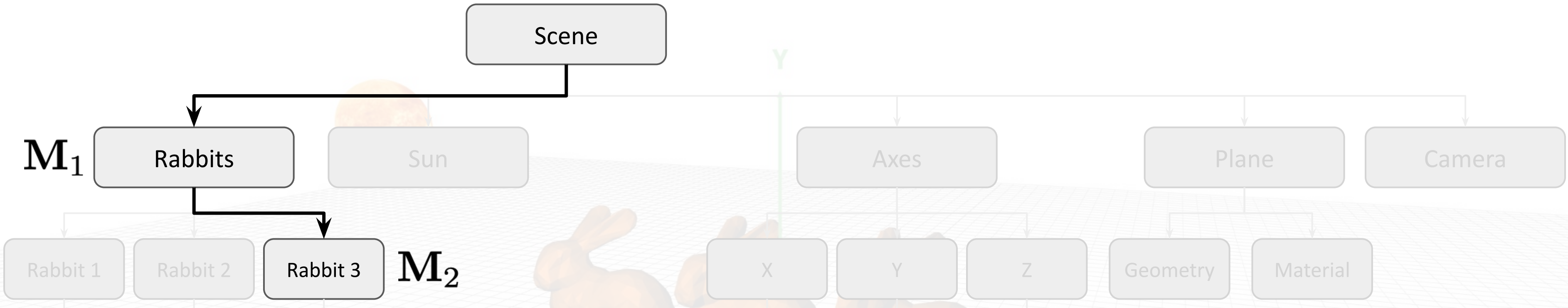
Scene
Rabbits
Sun
Axes
Plane
Camera
Rabbit 1
Rabbit 2
Rabbit 3
X
Y
Z
Geometry
Material
Rabbit Mesh
Axis
Geometry
Material
Cylinder
Cone
Text
Geometry
Material
Geometry
Material
Geometry
Material

Scene Graph



Scene
$\mathbf{M}_1$
Rabbits
Sun
Axes
Plane
Camera
Rabbit 1
Rabbit 2
Rabbit 3
$\mathbf{M}_2$
X
Y
Z
Geometry
Material
Rabbit Mesh
Axis
Geometry
Material
Cylinder
Cone
Text
Geometry
Material
Geometry
Material
Geometry
Material

Model Transformation



- The original bunny mesh first transformed by $\mathbf{M}_2$ then $\mathbf{M}_1$:

$$\mathbf{M}_1\mathbf{M}_2\mathrm{P}$$

  where P is a vertex of the bunny.

- The multiplied matrix of all transformations is the ***model transformation*** matrix of a given object.
  - e.g. $\mathbf{M}_1\mathbf{M}_2$ is the model transformation matrix of the bunny 3.

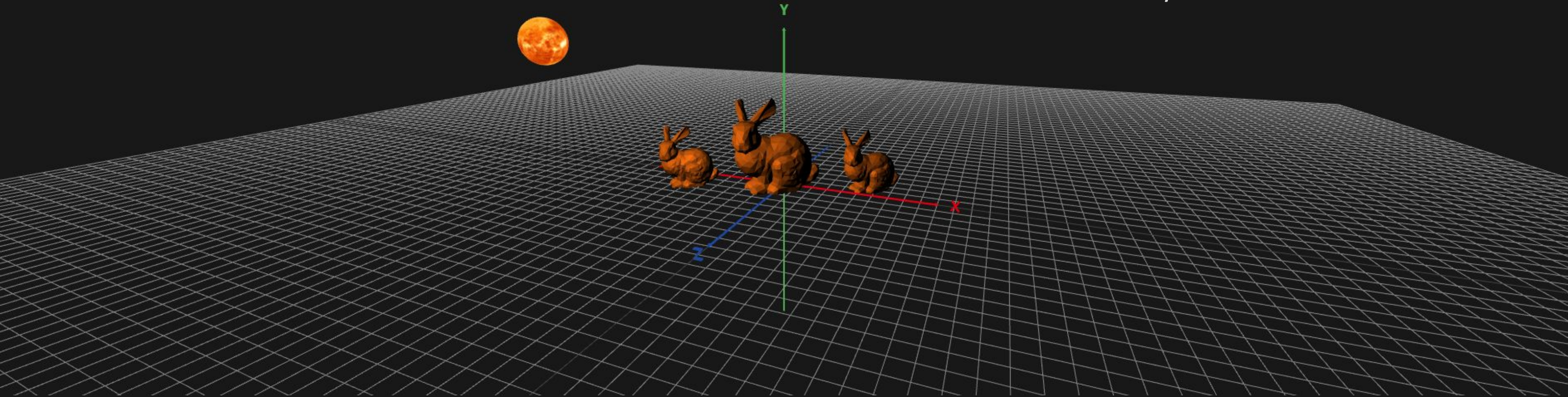
Breakout: Group Transformations

In the previously discussed scene graph the sun remains orbiting around +Y, but the sky is getting darker.

The middle bunny grows up and gets a little bit bigger than the others.

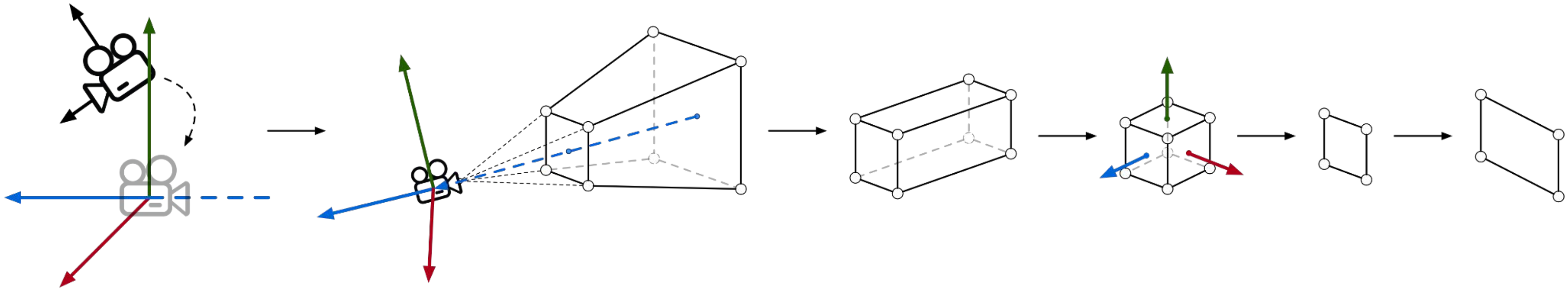
Find the TODO comment in the demos/04-camera/1-models (live demo).

1. Rotate the three bunnies around their intrinsic +Y axis individually
2. Rotate the three bunnies around the extrinsic +Y axis together
3. Rotate the three bunnies both around intrinsic +Y axis and extrinsic +Y axis simultaneously



Tutorial 4: Camera

- Scene Graph and Model Transformation
- **Viewing Transformations**
  - View Transformation
  - Projection Transformations
  - Viewport Transformation
- Application: Hitchcock Zoom
- Summary



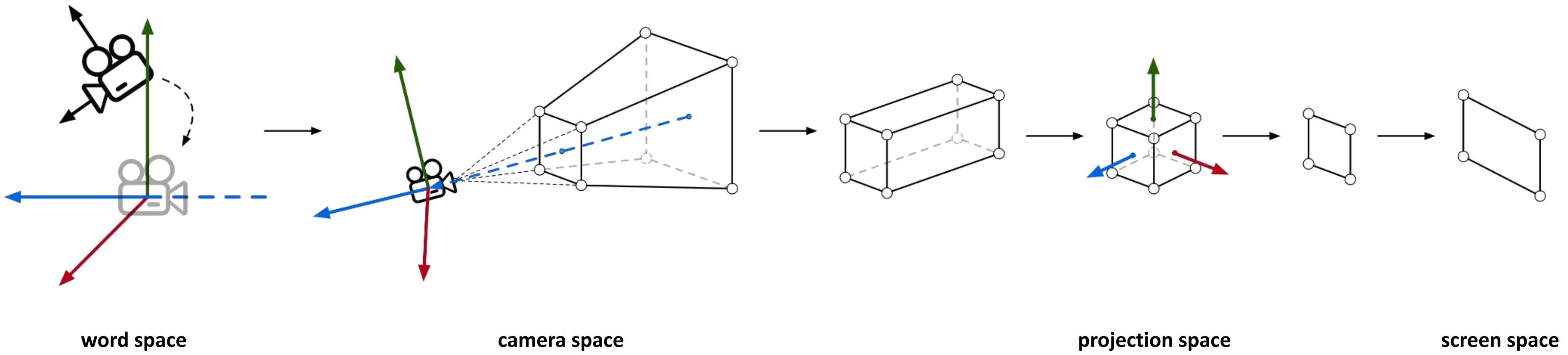
Viewing Transformation Pipeline

Viewing transformation is a 3D to 2D mapping that can be considered as 3 major transformation stages:

1. **View transformation:** World space to camera space

2. **Projection transformation:** Camera space to projective space

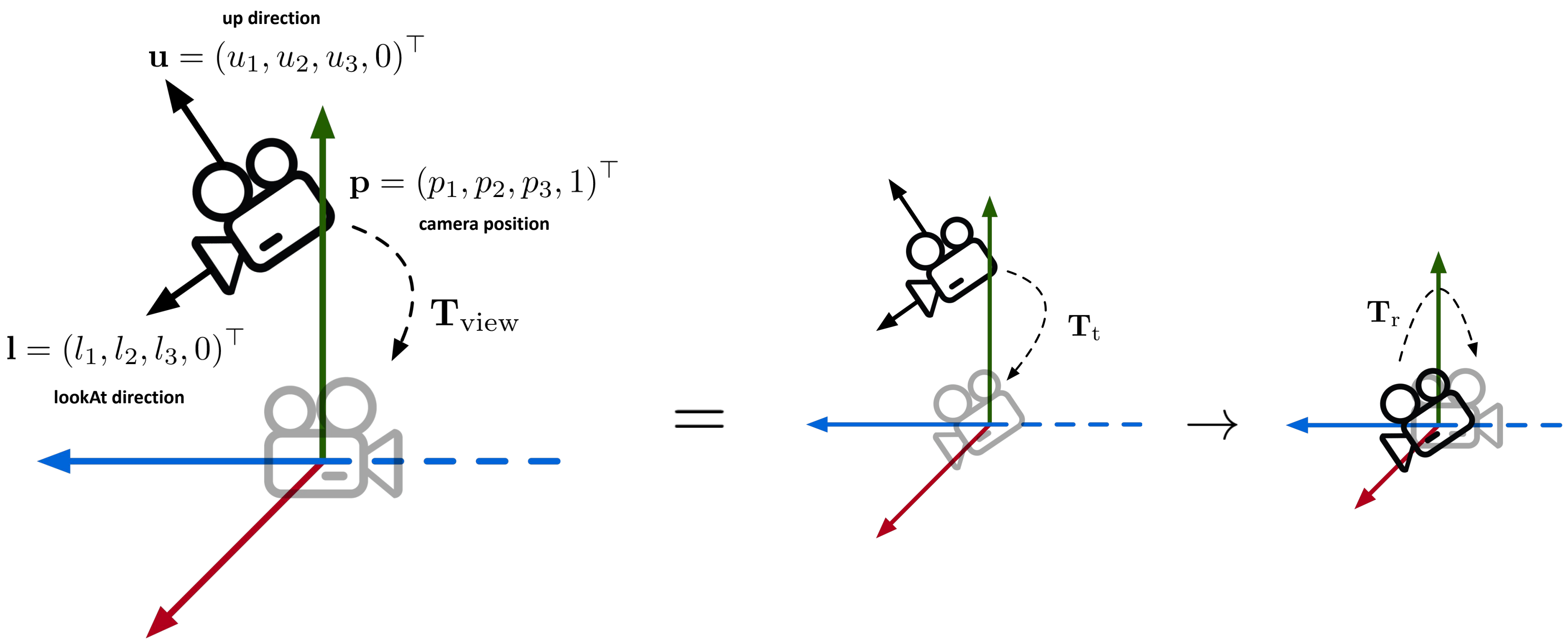
3. **Viewport transformation:** Projection space to screen space



View Transformation

The ***view transformation*** transforms the camera to the origin, it looks at $-Z$ and the upwards direction is $+Y$.

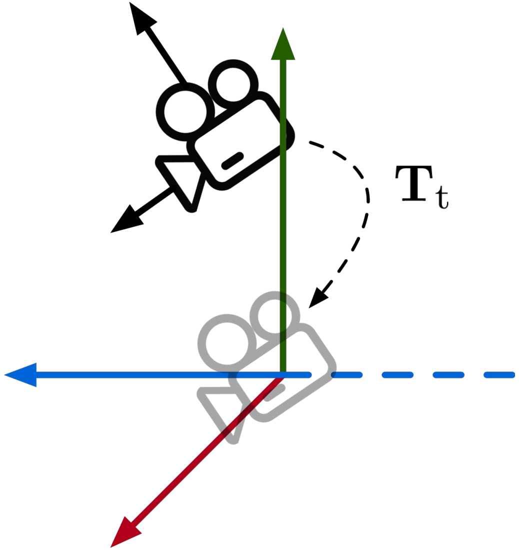
Translation first, then rotation: $\mathbf{T}_{\text{view}} = \mathbf{T}_r \mathbf{T}_t$



View Transformation: Translation

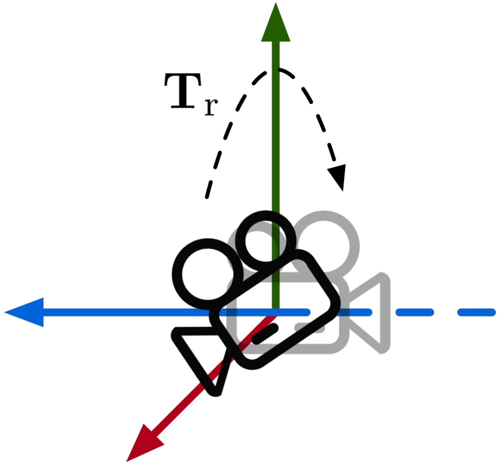
Translate based on the camera position:

$$\mathbf{T}_t = \begin{pmatrix} 1 & 0 & 0 & -p_1 \\ 0 & 1 & 0 & -p_2 \\ 0 & 0 & 1 & -p_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



View Transformation: Rotation

- **Goal**: Rotate camera coordinate frame from $\mathbf{l}$ to $-Z$, $\mathbf{u}$ to $+Y$ and $\mathbf{l} \times \mathbf{u}$ to $+X$.
- The inverse problem is easier: Rotate $+X$ to $\mathbf{l} \times \mathbf{u}$, $+Y$ to $\mathbf{u}$, and $+Z$ to $-\mathbf{l}$.
- For rotation matrices (orthonormal matrices), we have: $\mathbf{T}_r^{-1} = \mathbf{T}_r^\top$



Thus:

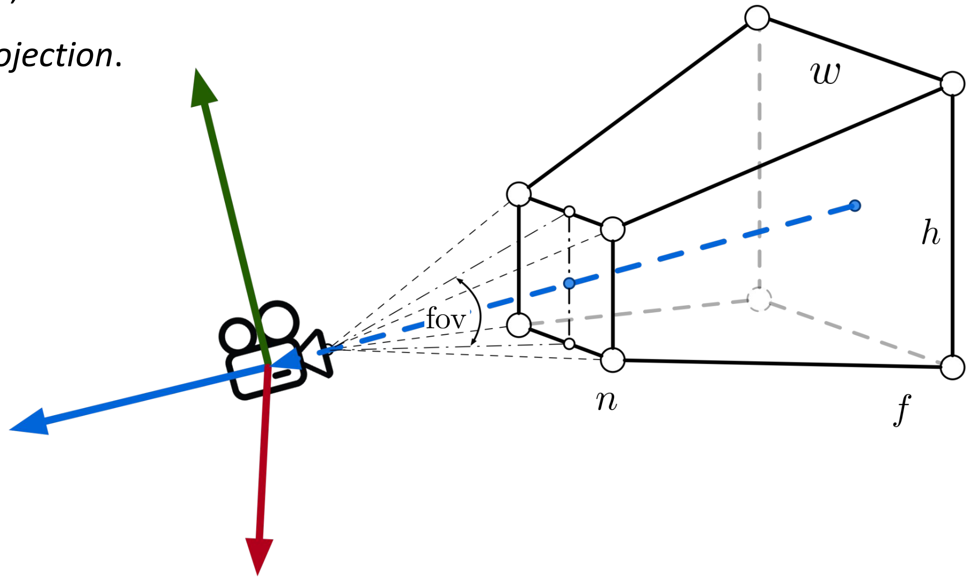
$$\mathbf{T}_r^{-1} = \begin{pmatrix} x_{\mathbf{l}\times\mathbf{u}} & x_{\mathbf{u}} & x_{-\mathbf{l}} & 0 \\ y_{\mathbf{l}\times\mathbf{u}} & y_{\mathbf{u}} & y_{-\mathbf{l}} & 0 \\ z_{\mathbf{l}\times\mathbf{u}} & z_{\mathbf{u}} & z_{-\mathbf{l}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \Rightarrow \mathbf{T}_r = (\mathbf{T}_r^{-1})^\top = \begin{pmatrix} x_{\mathbf{l}\times\mathbf{u}} & y_{\mathbf{l}\times\mathbf{u}} & z_{\mathbf{l}\times\mathbf{u}} & 0 \\ x_{\mathbf{u}} & y_{\mathbf{u}} & z_{\mathbf{u}} & 0 \\ x_{-\mathbf{l}} & y_{-\mathbf{l}} & z_{-\mathbf{l}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Q: How can we do this using quaternions?

Camera Projection

For the process of projecting our 3D scene onto a 2D image, we can use different methods.

One of the most common projections is the *perspective projection*.



It is similar to our visual process and similar to real cameras. Therefore, it is often perceived as natural. Objects far away appear smaller.

The captured area is defined by a near (*n*) and a far (*f*) plane, aspect ratio (*w/h*), as well as the field of view (*fov*)

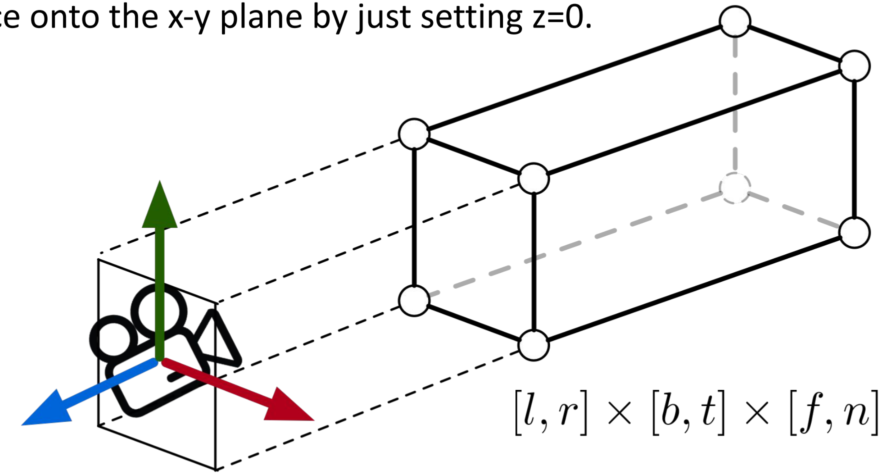
Camera Projection

Another commonly used form of projection is the *orthographic projection*.

It is a special form of parallel projection, which is independent of the distance to the camera.

As an example we can imagine to project the whole 3D space onto the x-y plane by just setting z=0.

Here, the area captured by the camera is given by a near (*n*) and a far (*f*) plane as well as left (*l*), bottom(*b*), top (*t*), and right (*r*) borders.



$$[l, r] \times [b, t] \times [f, n]$$

How can we orthographically project an object onto the camera plane (image)?

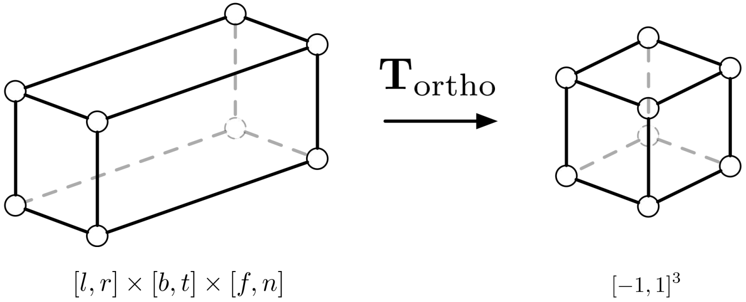
Orthographic Projection

We translate the center of the cube to the origin, then scale its length, width, and height to 2

This transforms the scene into the unit cube

$$\mathbf{T}_{\text{ortho}} = \underbrace{\begin{pmatrix} 2/(r-l) & 0 & 0 & 0 \\ 0 & 2/(t-b) & 0 & 0 \\ 0 & 0 & 2/(n-f) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\textbf{scale}} \underbrace{\begin{pmatrix} 1 & 0 & 0 & -(r+l)/2 \\ 0 & 1 & 0 & -(t+b)/2 \\ 0 & 0 & 1 & -(n+f)/2 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\textbf{translation}}$$

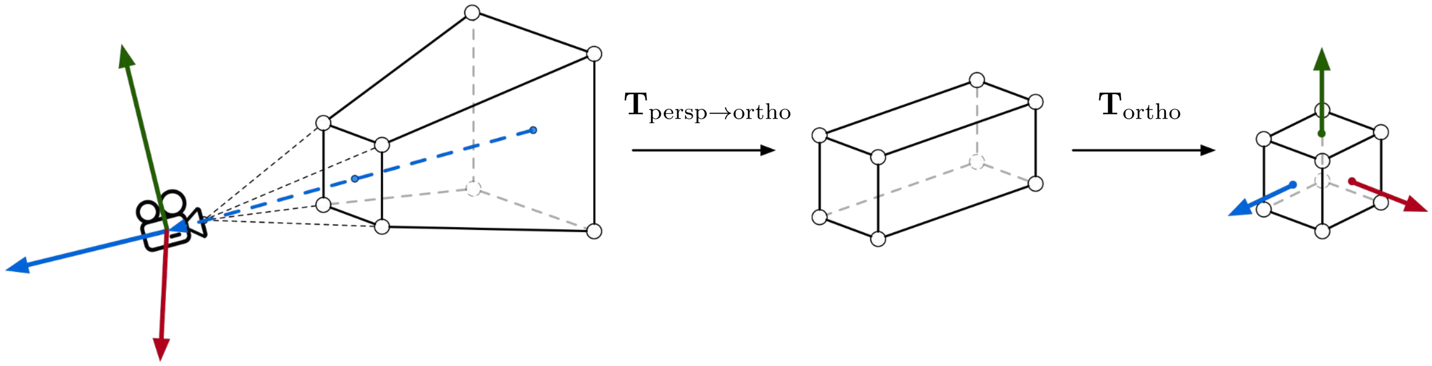
$$= \begin{pmatrix} 2/(r-l) & 0 & 0 & (l+r)/(l-r) \\ 0 & 2/(t-b) & 0 & (b+t)/(b-t) \\ 0 & 0 & 2/(n-f) & (f+n)/(f-n) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Perspective Projection

Now, a perspective projection can be considered as a composition two transformations: $\mathbf{T}_{\text{ortho}}\mathbf{T}_{\text{persp}\to\text{ortho}}$

We already know $\mathbf{T}_{\text{ortho}}$. So how can we calculate $\mathbf{T}_{\text{persp}\to\text{ortho}}$?



Perspective Projection (cont.)

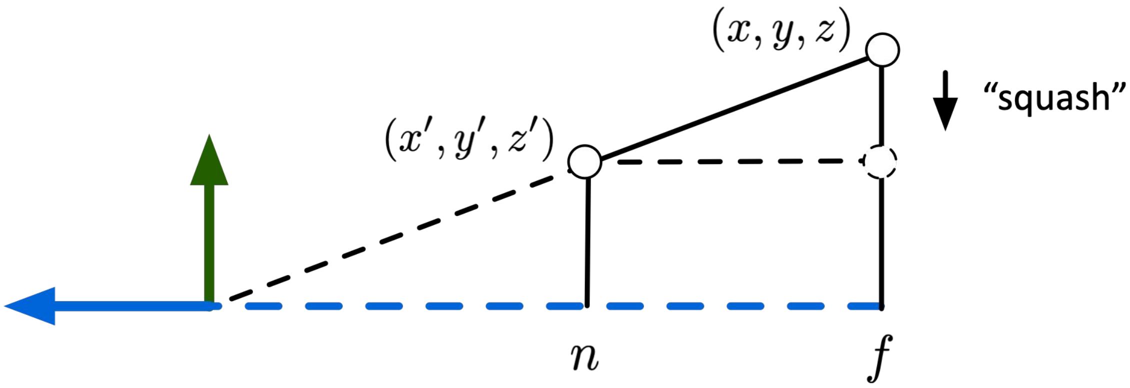
If we consider (x, y, z) with its projection being $(x', y', ?)$, we get similar triangles for *x* and *y*:

$$\frac{y'}{y} = \frac{n}{z}$$, similarly: $$\frac{x'}{x} = \frac{n}{z}$$

Thus, the transformation is:

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} nx/z \\ ny/z \\ ? \\ 1 \end{pmatrix} = \begin{pmatrix} nx \\ ny \\ ? \\ z \end{pmatrix}$$

homogeneous coordinates



Perspective Projection (cont.)

We can now define the transformation matrix, such that:

$$\begin{pmatrix} nx \\ ny \\ ? \\ z \end{pmatrix} = \mathbf{T}_{\text{persp}\to\text{ortho}} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ ? & ? & ? & ? \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Note that any point, which lies already on the near plane is not affected by this transformation. If *z* = *n* then (x, y, n, 1) will not move.

$$\begin{pmatrix} nx \\ ny \\ ? \\ n \end{pmatrix} \leftarrow \begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix} = \begin{pmatrix} nx \\ ny \\ n^2 \\ n \end{pmatrix}$$

Because **?** is irrelevant to x and y, the transformation matrix for points on the near plane looks like:

$$\begin{pmatrix} nx \\ ny \\ n^2 \\ n \end{pmatrix} = \mathbf{T}_{\text{persp}\to\text{ortho}} \begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & w_1 & w_2 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix}$$

Perspective Projection (cont.)

Further, the center of the far plane will not change, thus when *x = 0, y = 0, z = f*:

$$\begin{pmatrix} n \cdot 0 \\ n \cdot 0 \\ ? \\ f \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ f \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ f^2 \\ f \end{pmatrix}$$

Therefore, the transformation matrix looks like

$$\begin{pmatrix} 0 \\ 0 \\ f^2 \\ f \end{pmatrix} = \mathbf{T}_{\text{persp}\to\text{ortho}} \begin{pmatrix} 0 \\ 0 \\ f \\ 1 \end{pmatrix} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & w_1 & w_2 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ f \\ 1 \end{pmatrix}$$

Perspective Projection (cont.)

Near plane:

$$\begin{pmatrix} nx \\ ny \\ n^2 \\ n \end{pmatrix} = \mathbf{T}_{\text{persp}\to\text{ortho}} \begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & w_1 & w_2 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix} \implies nw_1 + w_2 = n^2$$

Center of the far plane:

$$\begin{pmatrix} 0 \\ 0 \\ f^2 \\ f \end{pmatrix} = \mathbf{T}_{\text{persp}\to\text{ortho}} \begin{pmatrix} 0 \\ 0 \\ f \\ 1 \end{pmatrix} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & w_1 & w_2 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ f \\ 1 \end{pmatrix} \implies fw_1 + w_2 = f^2$$

$$\Rightarrow w_1 = n + f, w_2 = -nf$$

$$\Rightarrow \mathbf{T}_{\text{persp}\to\text{ortho}} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -nf \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Perspective Projection (cont.)

We know that

$$\mathbf{T}_{\text{ortho}} = \begin{pmatrix} 2/(r-l) & 0 & 0 & (l+r)/(l-r) \\ 0 & 2/(t-b) & 0 & (b+t)/(b-t) \\ 0 & 0 & 2/(n-f) & (f+n)/(f-n) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

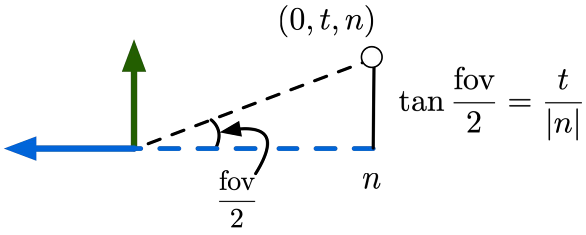
and

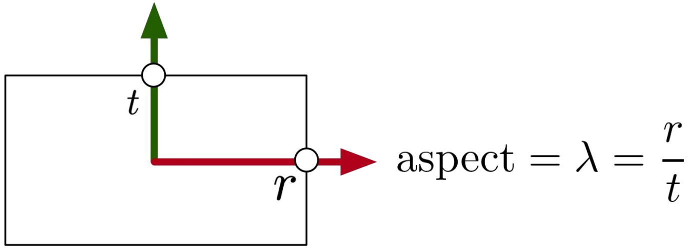
$$\mathbf{T}_{\text{persp}\rightarrow\text{ortho}} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -nf \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$\Rightarrow \quad \mathbf{T}_{\text{ortho}}\mathbf{T}_{\text{persp}\rightarrow\text{ortho}} = \begin{pmatrix} 2n/(r-l) & 0 & (l+r)/(l-r) & 0 \\ 0 & 2n/(t-b) & (b+t)/(b-t) & 0 \\ 0 & 0 & (n+f)/(n-f) & 2nf/(f-n) \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Perspective Projection Matrix

Are we finished? No, we don't know *l, r, b, t*! But we can easily get them:





$$l = -r = -\lambda t = -\lambda(-n)\tan\frac{\theta}{2} = \lambda n \tan\frac{\theta}{2}$$

$$b = -t = -(-n)\tan\frac{\theta}{2} = n\tan\frac{\theta}{2}$$

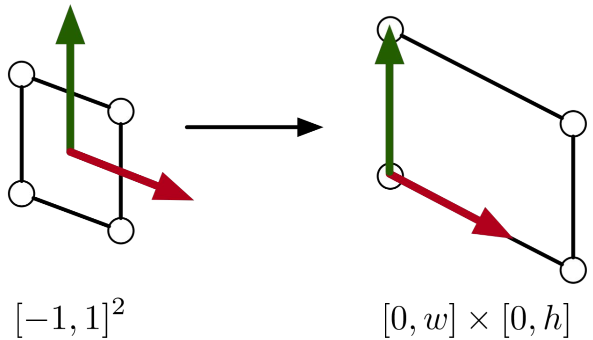
$$\Rightarrow \quad \mathbf{T}_{\text{ortho}}\mathbf{T}_{\text{persp}\to\text{ortho}} = \begin{pmatrix} -\frac{1}{\lambda\tan\frac{\theta}{2}} & 0 & 0 & 0 \\ 0 & -\frac{1}{\tan\frac{\theta}{2}} & 0 & 0 \\ 0 & 0 & \frac{n+f}{n-f} & \frac{2nf}{f-n} \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Viewport Transformation

A projection to the x-y plane is independent of z

Transform in x-y plane from $[-1,1]^2$ to $[0,w] \times [0,h]$

$$\mathbf{T}_{\text{viewport}} = \begin{pmatrix} w/2 & 0 & 0 & w/2 \\ 0 & h/2 & 0 & h/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Summary of Viewing Transformations

Model matrix $\mathbf{T}_{\text{model}} = \mathbf{M}_n \mathbf{M}_{n-1} \cdots \mathbf{M}_1$

View matrix $\mathbf{T}_{\text{view}} = \mathbf{T}_r \mathbf{T}_t$

Orthographic **p**rojection matrix $\mathbf{T}_{\text{ortho}}$

Perspective **p**rojection matrix $\mathbf{T}_{\text{persp}} = \mathbf{T}_{\text{ortho}} \mathbf{T}_{\text{persp}\to\text{ortho}}$

Viewport matrix $\mathbf{T}_{\text{viewport}}$

Model-View-Projection matrices are often called the *MVP-Matrices.*

We have prepared everything for creating the viewport, what's next? ⇒ Rasterization

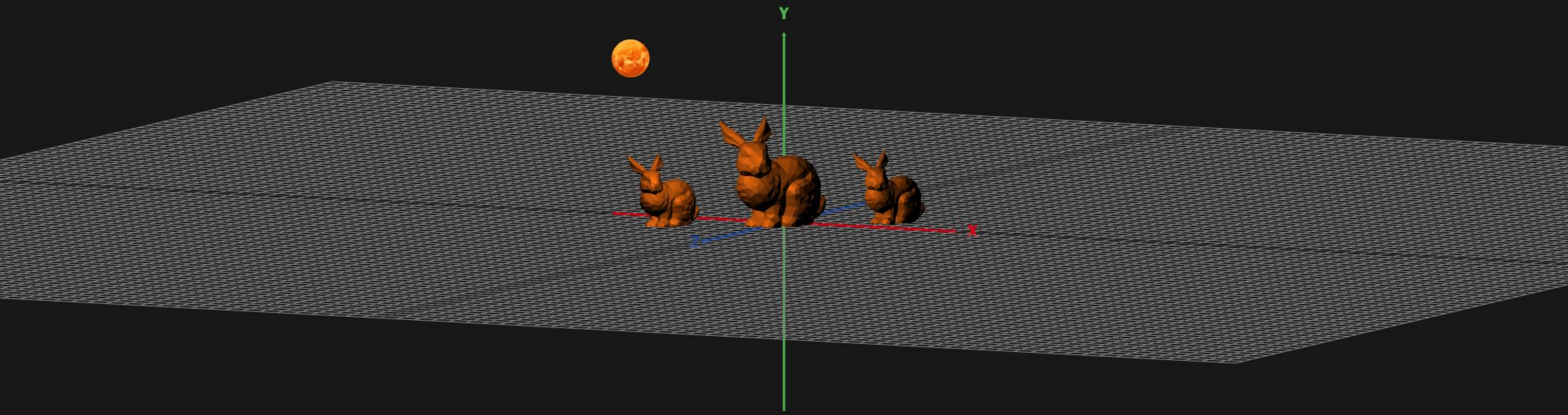
Breakout: Switch Between Cameras

Find TODO comment in the demos/04-camera/2-cameras/src/renderer.ts (live demo).

1. Render the view depending on different types of cameras (perspective and orthographic) in the render loop

2. Update projection matrix in the render loop then try tweaking parameters in the menu

Answer: Why was the menu not working before the projection matrix updates?



Tutorial 4: Camera

- Scene Graph and Model Transformation
- Viewing Transformations
  - View Transformation
  - Projection Transformations
  - Viewport Transformation
- Application: Hitchcock Zoom
- Summary

Hitchcock Zoom (aka Dolly Zoom)

https://www.youtube.com/watch?v=u5JBlwlnJX0

Math Behind the Hitchcock Zoom

If we want to keep the size of an object to be fixed, we need guarantee the projection of the object to be fixed.

Based on the perspective projection matrix:

$$P' = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \mathbf{T}_{\text{persp}} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} -\frac{1}{\lambda \tan\frac{\theta}{2}} & 0 & 0 & 0 \\ 0 & -\frac{1}{\tan\frac{\theta}{2}} & 0 & 0 \\ 0 & 0 & \frac{n+f}{n-f} & \frac{2nf}{f-n} \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} -\frac{1}{\lambda \tan\frac{\theta}{2}} \\ -\frac{1}{\tan\frac{\theta}{2}} \\ \cdots \\ z \end{pmatrix} = \begin{pmatrix} -\frac{1}{z\lambda \tan\frac{\theta}{2}} \\ -\frac{1}{z \tan\frac{\theta}{2}} \\ \cdots \\ 1 \end{pmatrix}$$

If we want keep an object's coordinates to stay where they were (say, inside a rectangle), for *y* coordinate:

$$-\frac{1}{z \tan\frac{\theta}{2}} = \frac{1}{\text{distance} \cdot \tan\frac{\text{fov}}{2}} = h_{\text{obj}}$$

In particular, if the height of the object is less than 1, we have:

$$\text{distance} = \frac{1}{\tan\frac{\text{fov}}{2}}$$

More Math Behind the Hitchcock Zoom

As the perspective projection transformation tells us:

$$P' = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \mathbf{T}_{\text{persp}} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} -\frac{1}{\lambda \tan\frac{\theta}{2}} & 0 & 0 & 0 \\ 0 & -\frac{1}{\tan\frac{\theta}{2}} & 0 & 0 \\ 0 & 0 & \frac{n+f}{n-f} & \frac{2nf}{f-n} \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} -\frac{1}{\lambda \tan\frac{\theta}{2}} \\ -\frac{1}{\tan\frac{\theta}{2}} \\ \cdots \\ z \end{pmatrix} = \begin{pmatrix} -\frac{1}{z\lambda \tan\frac{\theta}{2}} \\ -\frac{1}{z \tan\frac{\theta}{2}} \\ \cdots \\ 1 \end{pmatrix}$$

If we want keep the whole object to be fixed, then

$$x = \frac{1}{\text{distance} \cdot \text{aspect} \cdot \tan\frac{\text{fov}}{2}} = w_{\text{obj}} \qquad y = \frac{1}{\text{distance} \cdot \tan\frac{\text{fov}}{2}} = h_{\text{obj}}$$

In particular, it can be achieved if the object is a square and the viewport is also square

$$\frac{h_{\text{obj}}}{w_{\text{obj}}} = \text{aspect}$$

This is why the Hitchcock Zoom is not always perfectly achieved.

Breakout: Hitchcock Zoom

Enter folder demos/04-geometry/3-dolly (live demo)

Look for the TODO comment in the `main.ts` and implement:

Step 1: calculate the field of view and the distance if one is given.

Step 2: update the camera settings accordingly.

Hitchcock Zoom - Step 1

Calculate the field of view and the distance if one is given.

From the previous slides we know that for objects with a height below 1

$$\text{distance} = \frac{1}{\tan\frac{\text{fov}}{2}} \quad \text{which yields} \quad \text{fov} = 2\arctan\frac{1}{\text{distance}}$$

Implement these formulas in:

```
dollyZoomFOV(dist: number): number {
  // TODO: calculate the corresponding fov from the given distance
  return 2 * (Math.atan(1 / dist) * (180 / Math.PI));
}
dollyZoomDist(fov: number): number {
  // TODO: calculate the corresponding distance from the given fov
  return 2 / Math.tan(fov * (Math.PI / 180));
}
```

Hitchcock Zoom - Step 2

Use the previously implemented functions to adjust the camera's parameter if a value is changed. You can then experiment with the sliders or activate the animation.

```
changeFOV() {
  if (this.params.animate) {
    return;
  }

  // TODO: update the fov of the given camera.
  this.camera.fov = this.params.fov;
  const dist = this.dollyZoomDist(this.params.fov);
  this.params.distance = dist;
  this.camera.position.x = dist;
}
```

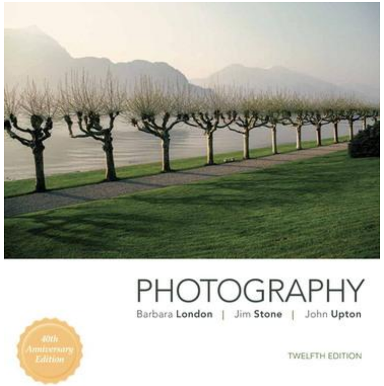
```
changeDist() {
  if (this.params.animate) {
    return;
  }

  // TODO: update the camera distance to the bunny.
  this.camera.position.x = this.params.distance;
  const fov = this.dollyZoomFOV(this.params.distance);
  this.params.fov = fov;
  this.camera.fov = fov;
}
```

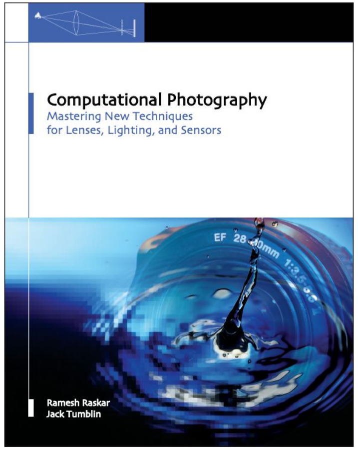
Tutorial 4: Camera

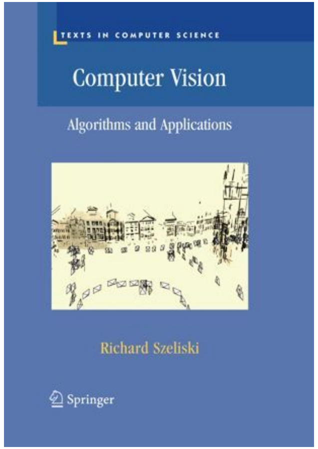
- Scene Graph and Model Transformation
- Viewing Transformations
  - View Transformation
  - Projection Transformations
  - Viewport Transformation
- Application: Hitchcock Zoom
- Summary

Summary

- We covered
  - The scene graph and model transformations
  - The viewing transformation pipeline from 3D model space to 2D screen space
  - Camera as a powerful tool to express visual effects not only for photography but also computer-generated animations
  - The Hitchcock zoom effects and how implement it
- To archive more camera effects, knowledge in practical photography and how it can be applied are needed. We encourage you to check these books:







Next

Rasterization I