Tutorial 4

Camera

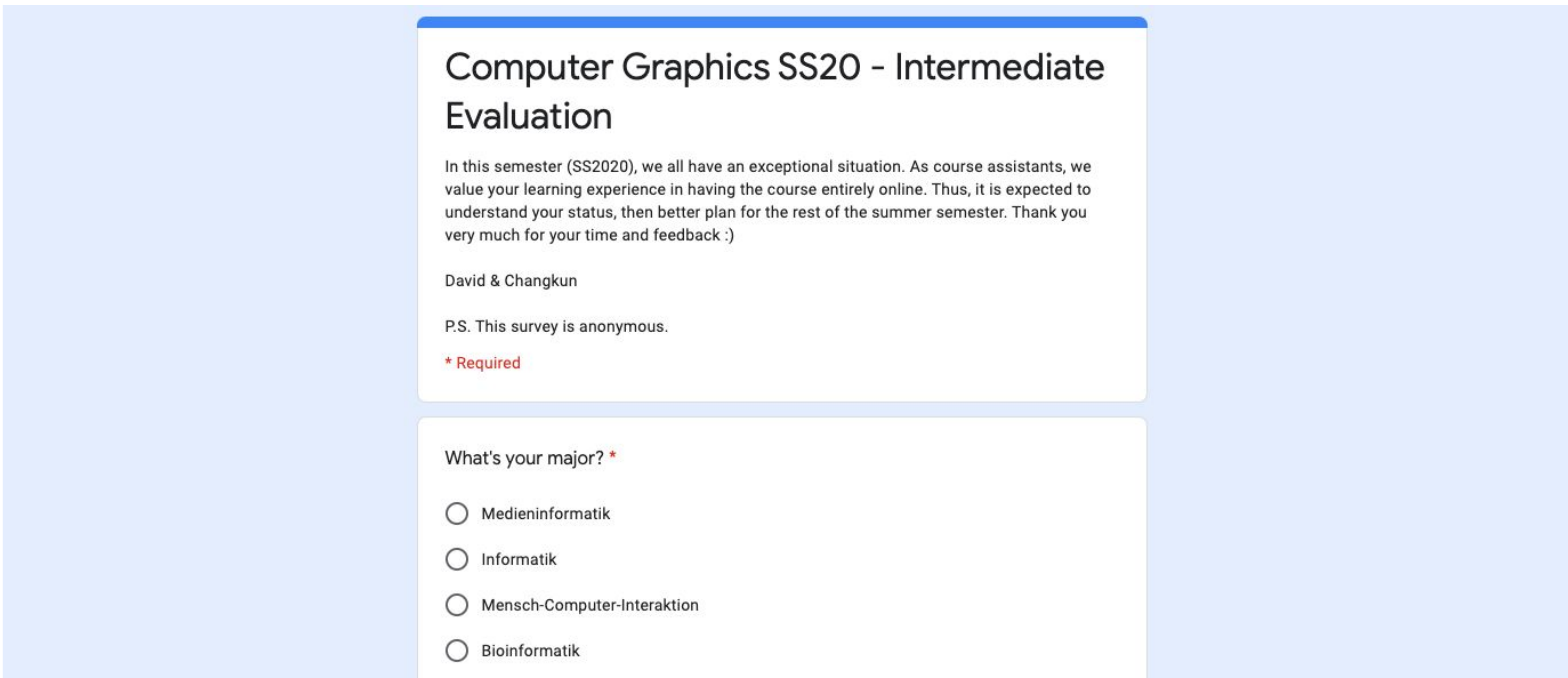
Computer Graphics

Summer Semester 2020

Ludwig-Maximilians-Universität München

Midterm Survey

Submit your feedback before 01.06.2020, the results will be available to you later when the evaluation is done.
Link: https://forms.gle/XqWC5cctM56GBvZV9



Computer Graphics SS20 - Intermediate Evaluation

In this semester (SS2020), we all have an exceptional situation. As course assistants, we value your learning experience in having the course entirely online. Thus, it is expected to understand your status, then better plan for the rest of the summer semester. Thank you very much for your time and feedback :)

David & Changkun

P.S. This survey is anonymous.

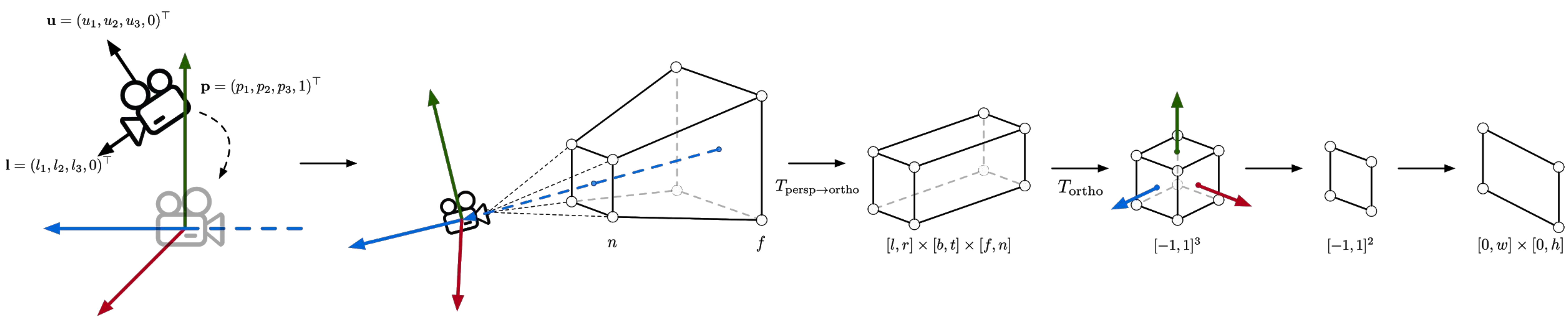
* Required

What's your major? *

- Medieninformatik
- Informatik
- Mensch-Computer-Interaktion
- Bioinformatik

Agenda

- View Transformation
- Projection Transformation
- Viewport Transformation
- Hitchcock Zoom



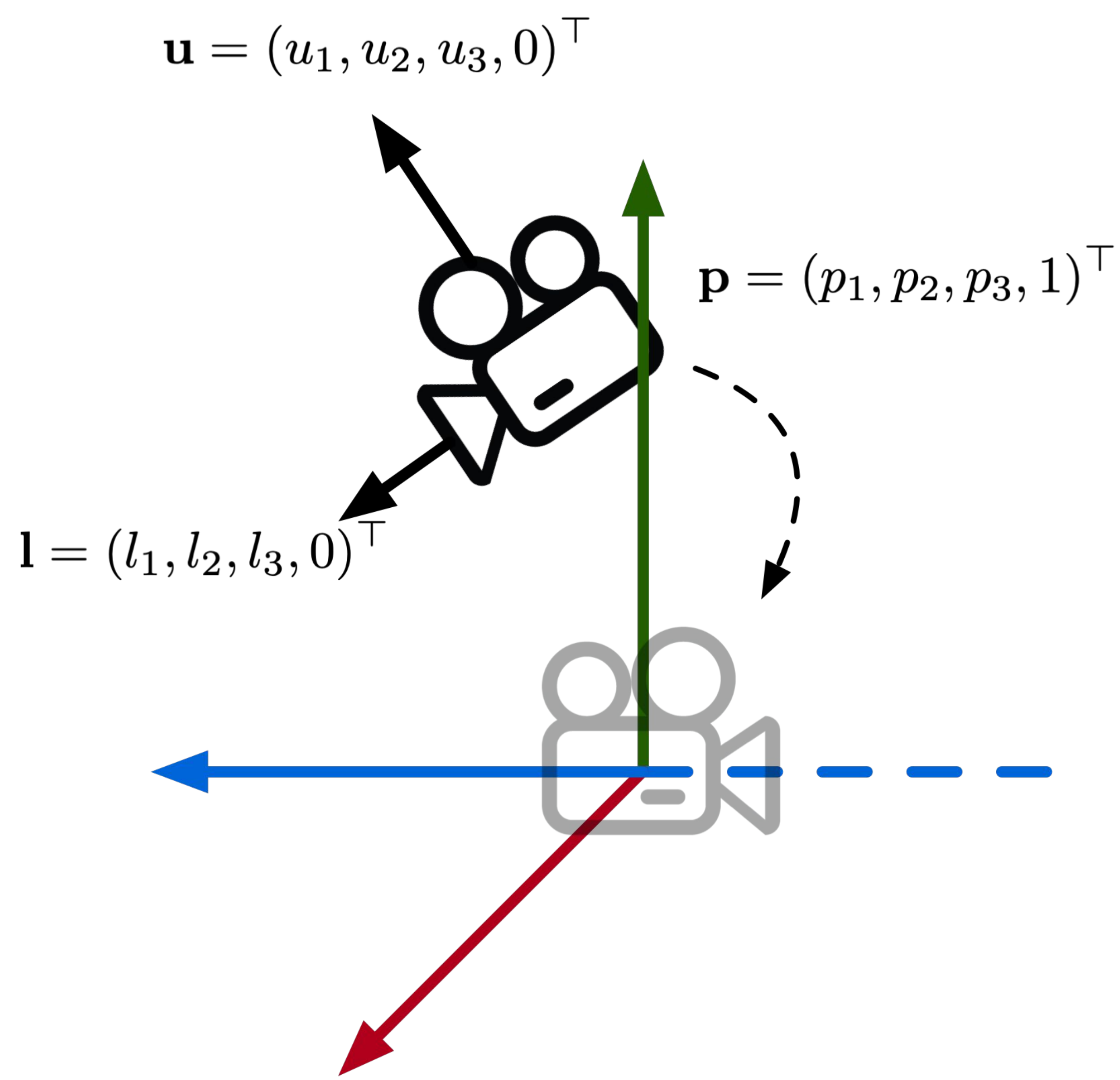
Tutorial 4: Camera

- View Transformation
- Projection Transformation
- Viewport Transformation
- Hitchcock Zoom

View Transformation

The view transformation transforms the camera to origin, it looks at -Z, up direction +Y.

Translation + Rotation: $M_{\text{view}} = R_{\text{view}} T_{\text{view}}$



Task 1 a) Camera Translation

Translate based on the camera position:

$$T_{\text{view}} = \begin{pmatrix} 1 & 0 & 0 & -p_1 \\ 0 & 1 & 0 & -p_2 \\ 0 & 0 & 1 & -p_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Task 1 b) Camera Rotation

Rotate camera coordinate frame from l to -Z, u to +Y and (lxu) to +X.

Inverse: Rotate +X to (lxu), +Y to u, and +Z to -l

$$R_{\text{view}}^{-1} = \begin{pmatrix} x_{\mathbf{l}\times\mathbf{u}} & x_{\mathbf{u}} & x_{-\mathbf{l}} & 0 \\ y_{\mathbf{l}\times\mathbf{u}} & y_{\mathbf{u}} & y_{-\mathbf{l}} & 0 \\ z_{\mathbf{l}\times\mathbf{u}} & z_{\mathbf{u}} & z_{-\mathbf{l}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Lemma: For rotation matrices (orthogonal matrices)

$$R^{-1} = R^T$$

$$\implies R_{\text{view}} = \begin{pmatrix} x_{\mathbf{l}\times\mathbf{u}} & y_{\mathbf{l}\times\mathbf{u}} & z_{\mathbf{l}\times\mathbf{u}} & 0 \\ x_{\mathbf{u}} & y_{\mathbf{u}} & z_{\mathbf{u}} & 0 \\ x_{-\mathbf{l}} & y_{-\mathbf{l}} & z_{-\mathbf{l}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Q: How to do it using quaternions?

Tutorial 4: Camera

- View Transformation
- Projection Transformation
- Viewport Transformation
- Hitchcock Zoom

Orthographic Projection

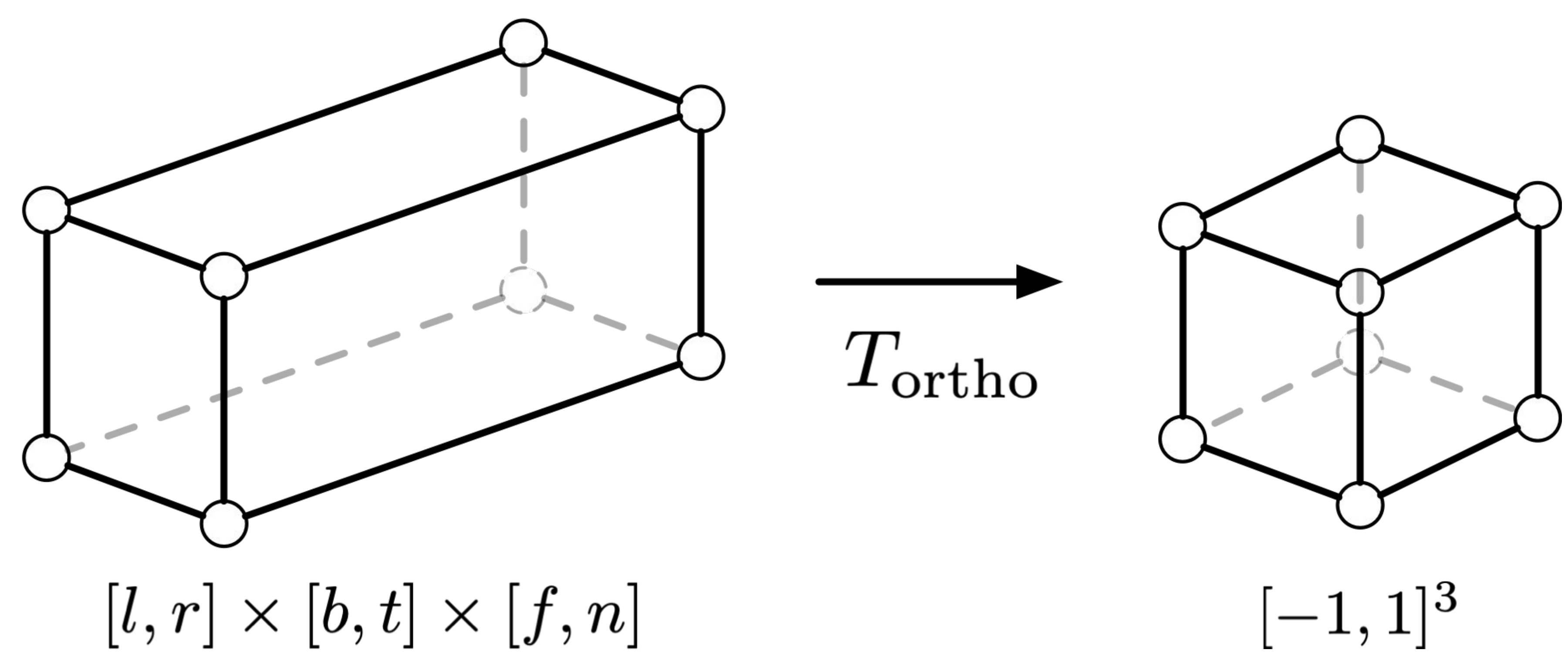
Task 2:

a) Projecting a point to the x-y plane is irrelevant to z, thus the projected coordinates are (x, y, 0)

b) How? translate the center of the cube to origin, then scale its length, width, and height

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

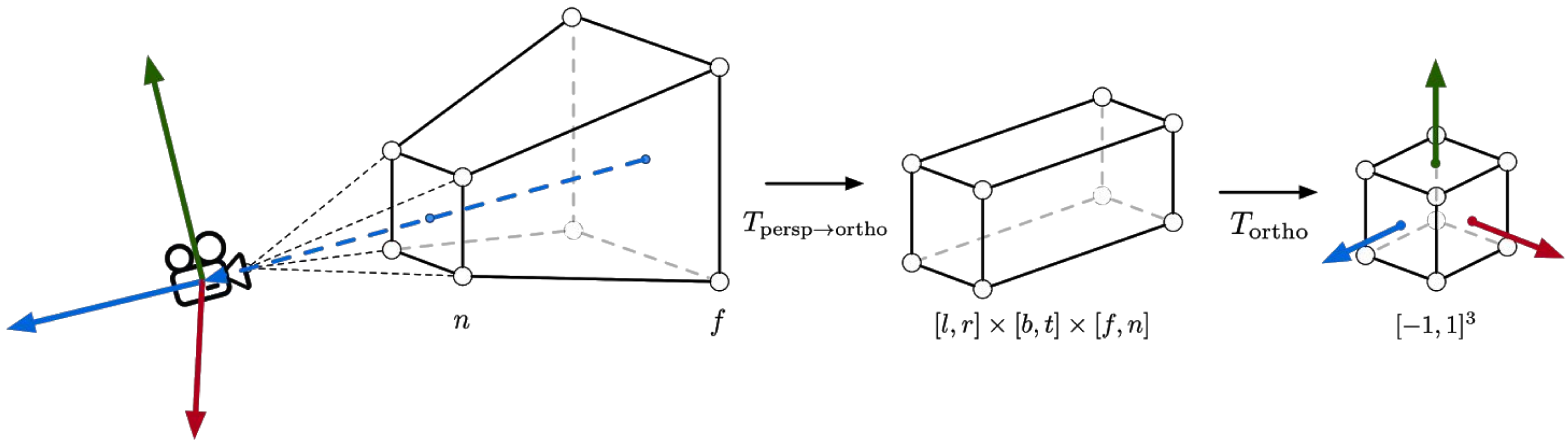
$$= \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & \frac{l+r}{l-r} \\ 0 & \frac{2}{t-b} & 0 & \frac{b+t}{b-t} \\ 0 & 0 & \frac{2}{n-f} & \frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Perspective Projection

A perspective projection can be decomposed into two transformations: $T_{\text{ortho}} T_{\text{persp}\to\text{ortho}}$

We already know T_{ortho}



Task 2 c)

If we consider (x, y, z) that projects to (x', y',?), we get similar triangles:

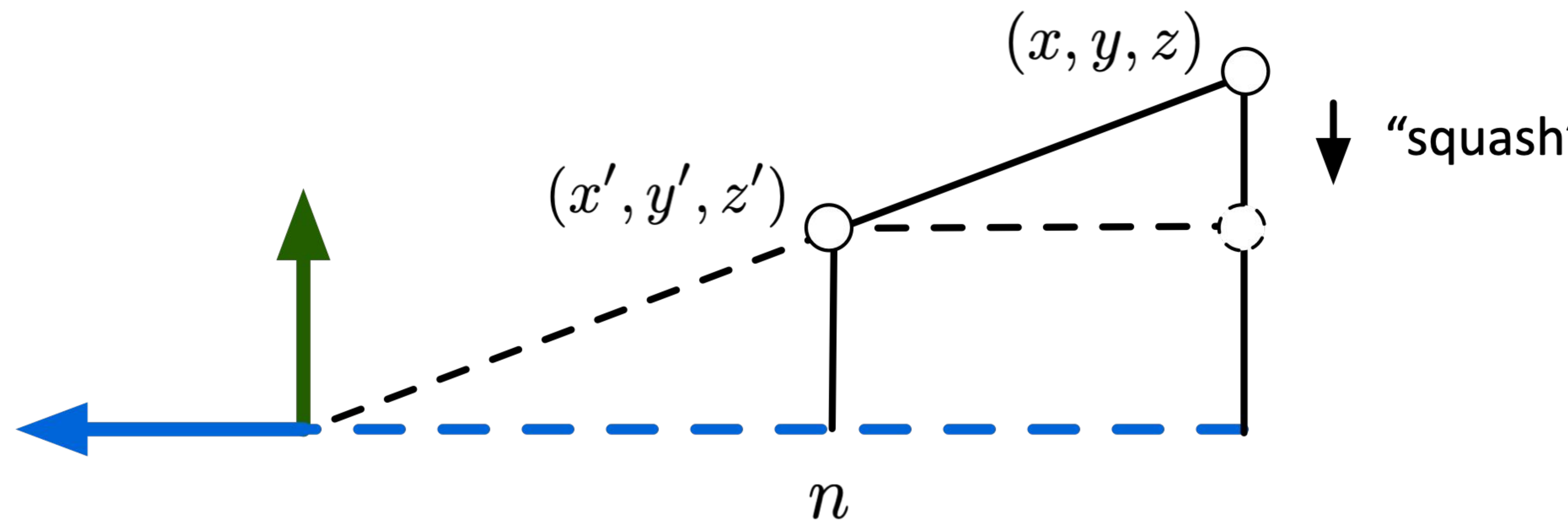
$$\frac{y'}{y} = \frac{n}{z}$$

Similarly:

$$\frac{x'}{x} = \frac{n}{z}$$

Thus:

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



Task 2 c) continuation.

One can observe:

$$\begin{pmatrix} nx \\ ny \\ ? \\ z \end{pmatrix} = T_{\text{persp}\rightarrow\text{ortho}} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ ? & ? & ? & ? \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Note that: 1. any point on the near plane will not change, thus when *z = n*: (x, y, n, 1) will not move and just transforms to itself:

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

Because **?** is irrelevant to x and y, the transformation matrix for the near plane should be:

$$\begin{pmatrix} nx \\ ny \\ n^2 \\ n \end{pmatrix} = T_{\text{persp}\rightarrow\text{ortho}} \begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & w_1 & w_2 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix}$$

Task 2 c) continuation. 2

Note that: 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, with

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

Task 2 c) continuation. 3

Near plane: $\begin{pmatrix} nx \\ ny \\ n^2 \\ z \end{pmatrix} = T_{\text{persp}\to\text{ortho}} \begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & w_1 & w_2 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix} \implies nw_1 + w_2 = n^2$

Far plane: $\begin{pmatrix} 0 \\ 0 \\ f^2 \\ f \end{pmatrix} = T_{\text{persp}\to\text{ortho}} \begin{pmatrix} 0 \\ 0 \\ f \\ 1 \end{pmatrix} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & w_1 & w_2 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ f \\ 1 \end{pmatrix} \implies fw_1 + w_2 = f^2$

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

$$\implies T_{\text{persp}\to\text{ortho}} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -nf \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Task 2 d) Perspective Projection Matrix

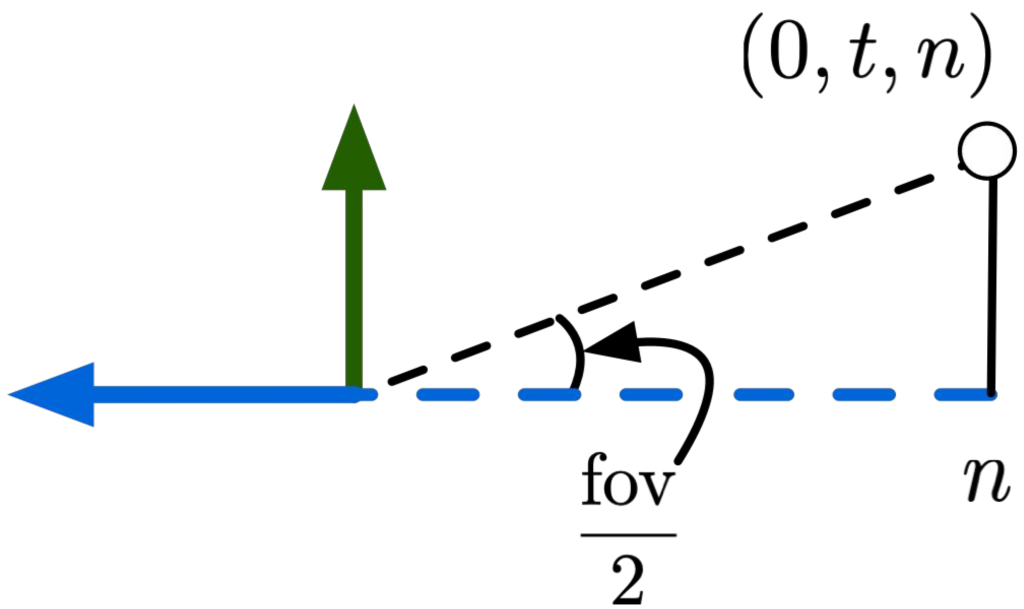
From b): $$T_{\text{ortho}} = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & \frac{l+r}{l-r} \\ 0 & \frac{2}{t-b} & 0 & \frac{b+t}{b-t} \\ 0 & 0 & \frac{2}{n-f} & \frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

From c): $$T_{\text{persp}\to\text{ortho}} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -nf \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

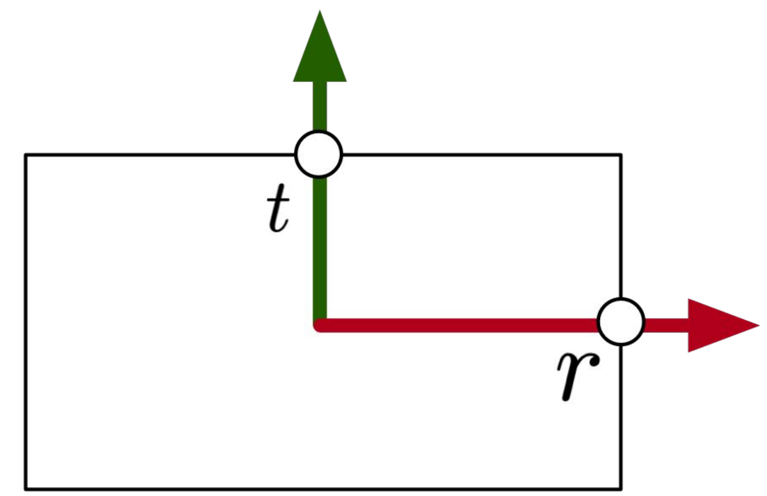
$$\implies T_{\text{ortho}}T_{\text{persp}\to\text{ortho}} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{l+r}{l-r} & 0 \\ 0 & \frac{2n}{t-b} & \frac{b+t}{b-t} & 0 \\ 0 & 0 & \frac{n+f}{n-f} & \frac{2nf}{f-n} \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Task 2 d) Are we finished? No, we don't know *l, r, b, t*!

But it is trivial...



$$\tan\frac{\text{fov}}{2} = \frac{t}{|n|}$$



$$\text{aspect} = \frac{r}{t}$$

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

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

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

Tutorial 4: Camera

- View Transformation
- Projection Transformation
- **Viewport Transformation**
- Hitchcock Zoom

Viewport Transformation

Task 3:

A projection to the x-y plane is irrelevant to z

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

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

Summary of Viewing Transformations

Model matrix	T_{model}
View matrix	M_{view}
Orthographic **p**rojection matrix	T_{ortho}
Perspective **p**rojection matrix	$T_{\text{persp}} = T_{\text{ortho}} T_{\text{persp}\to\text{ortho}}$
Viewport matrix	T_{viewport}

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

With all these transformations, we have prepared everything for creating the viewport, what's next?

⇒ Rasterization (see Assignment 5)

Tutorial 4: Camera

- View Transformation
- Projection Transformation
- Viewport Transformation
- Hitchcock Zoom

Hitchcock Zoom (aka Dolly Zoom)

In case you haven't seen it...

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

Math Behind the Hitchcock Zoom

If we want keep the size of an object to be fixed, then we need guarantee the projection of the object to be fixed. Based on the perspective projection matrix:

$$P' = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = T_{\text{persp}} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} -\frac{1}{\lambda \tan\frac{\theta}{2}} & 0 & 0 & 0 \\ 0 & -\frac{1}{\tan\frac{\theta}{2}} & 0 & 0 \\ 0 & 0 & \frac{n+f}{n-f} & \frac{2nf}{f-n} \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} -\frac{1}{\lambda \tan\frac{\theta}{2}} \\ -\frac{1}{\tan\frac{\theta}{2}} \\ \dots \\ z \end{pmatrix} = \begin{pmatrix} -\frac{1}{z\lambda \tan\frac{\theta}{2}} \\ -\frac{1}{z \tan\frac{\theta}{2}} \\ \dots \\ 1 \end{pmatrix}$$

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

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

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

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

More Math Behind the Hitchcock Zoom

$$P' = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = T_{\text{persp}} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} -\frac{1}{\lambda \tan \frac{\theta}{2}} & 0 & 0 & 0 \\ 0 & -\frac{1}{\tan \frac{\theta}{2}} & 0 & 0 \\ 0 & 0 & \frac{n+f}{n-f} & \frac{2nf}{f-n} \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} -\frac{1}{\lambda \tan \frac{\theta}{2}} \\ -\frac{1}{\tan \frac{\theta}{2}} \\ \dots \\ z \end{pmatrix} = \begin{pmatrix} -\frac{1}{z\lambda \tan \frac{\theta}{2}} \\ -\frac{1}{z \tan \frac{\theta}{2}} \\ \dots \\ 1 \end{pmatrix}$$

As the transformation tells us:

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

If we want keep the whole object to be fixed, then $\frac{h_{\text{obj}}}{w_{\text{obj}}} = \text{aspect}$

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

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

Task 4 - Step 1: Distance-FOV Conversion

```
 setup() {
  this.scene.add(this.models.sponza.scene)
  const bunny = new Mesh(...)

  ...
  bunny.scale.copy(new Vector3(1, 1, 1))
  this.models.bunny = bunny
  this.scene.add(bunny)
}
dollyZoomFOV(dist) {
  // TODO: calculate the corresponding fov of given distance
  return Math.atan(1/dist)*360/Math.PI
}
dollyZoomDist(fov) {
  // TODO: calculate the corresponding distance of given fov
  return 1 / Math.tan((fov*Math.PI)/360)
}
```

The bunny is not scaled. Therefore, these formulas are reasonable to use:

$$\Longrightarrow \quad \text{fov} = 2 \arctan \frac{1}{\text{distance}}$$

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

Task 4 - Step 2: Update Camera Settings

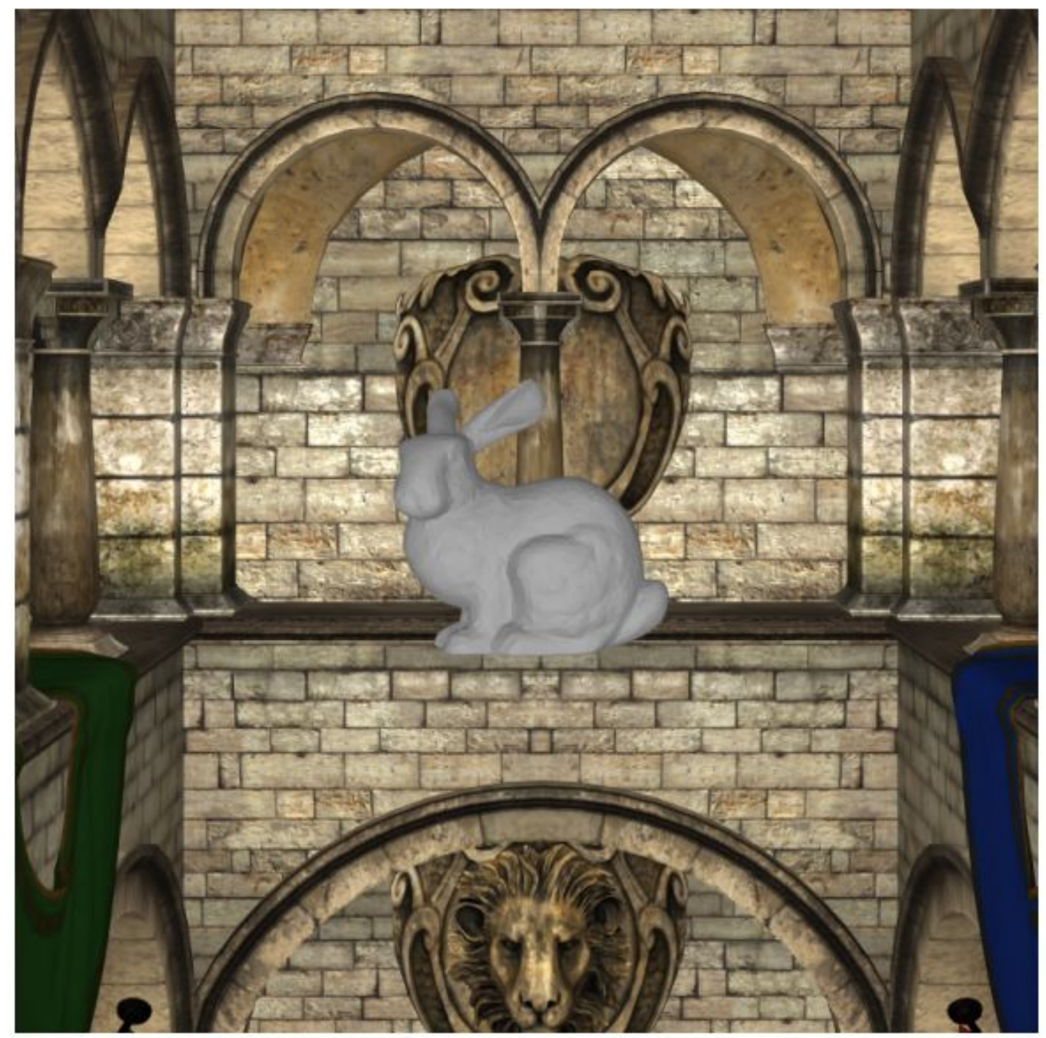
```
changeFOV() {
  // TODO: update the fov of the given camera.
  if (this.params.animate) {
    return
  }
  this.camera.fov = this.params.fov
  const dist = this.dollyZoomDist(this.params.fov)
  this.params.distance = dist
  this.camera.position.x = dist
}
changeDist() {
  // TODO: update the camera distance to the bunny.
  if (this.params.animate) {
    return
  }
  this.camera.position.x = this.params.distance
  const fov = this.dollyZoomFOV(this.params.distance)
  this.params.fov = fov
  this.camera.fov = fov
}
/**
 * update is executed in the render loop. One can use it to update
 * objects or camera for the next frame.
 */
update() {
  // TODO:
  this.cameraAnimate()
  this.camera.updateProjectionMatrix()
}
```

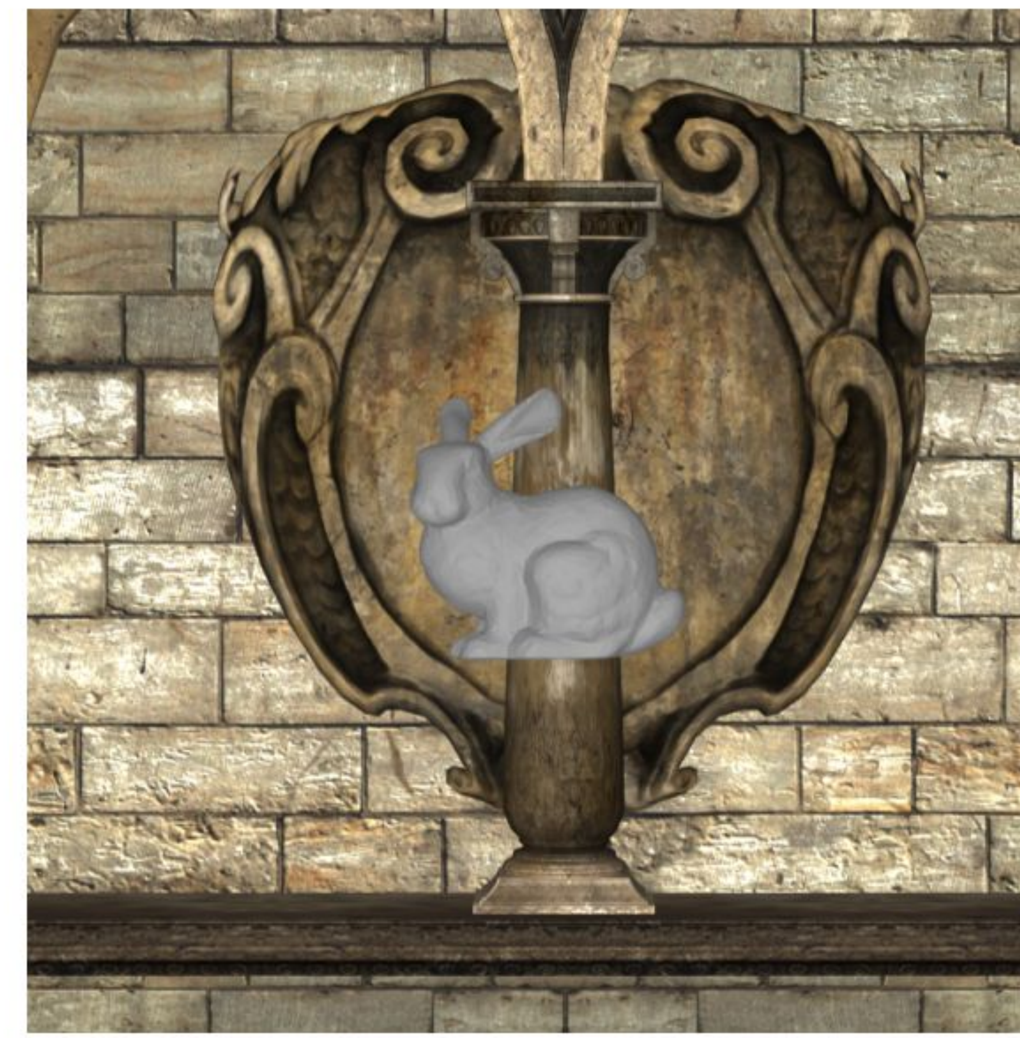
Task 4 - Step 3: Animating Camera

```
cameraAnimate() {
  // TODO: use this.path for updating camera position
  if (!this.params.animate) {
    return
  }
  if (this.frame === 1000) {
    this.frame--
    this.positive = false
  }
  if (this.frame === 0) {
    this.frame++
    this.positive = true
  }

  const p = this.path.getPoint(this.frame / 1000)
  this.camera.position.x = p.x
  this.params.distance = p.x
  const fov = this.dollyZoomFOV(this.params.distance)
  this.camera.fov = fov
  this.params.fov = fov

  if (this.positive) {
    this.frame++
  } else {
    this.frame--
  }
}
```

Final

Live demo (needs some time to load the models ~50MB). Before model is loaded it will stay black: https://www.medien.ifi.lmu.de/lehre/ss20/cg1/demo/4-camera/hitchcock/index.html





Task 4: Text Questions

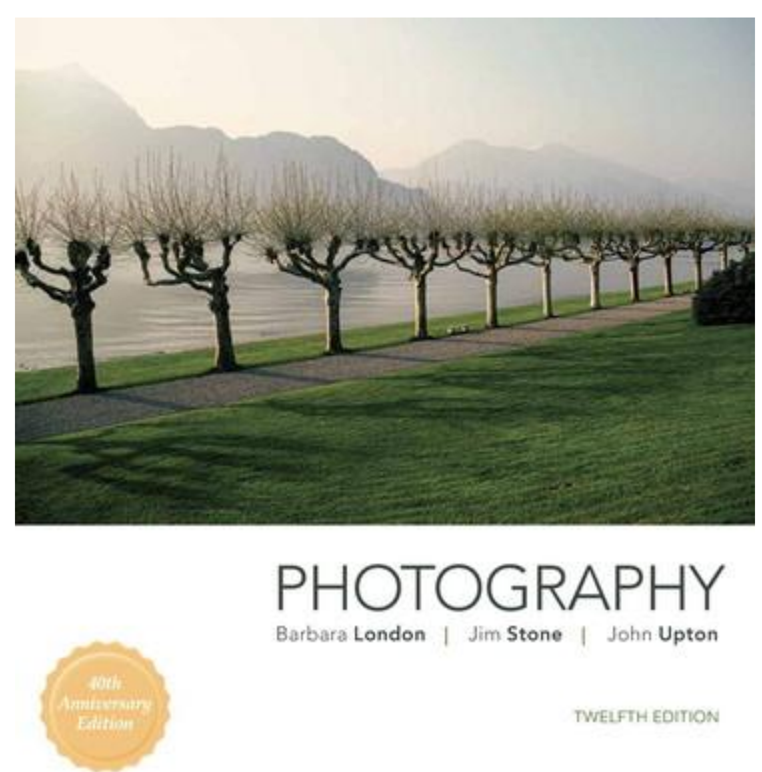
a) distance to the object and the fov of the given camera

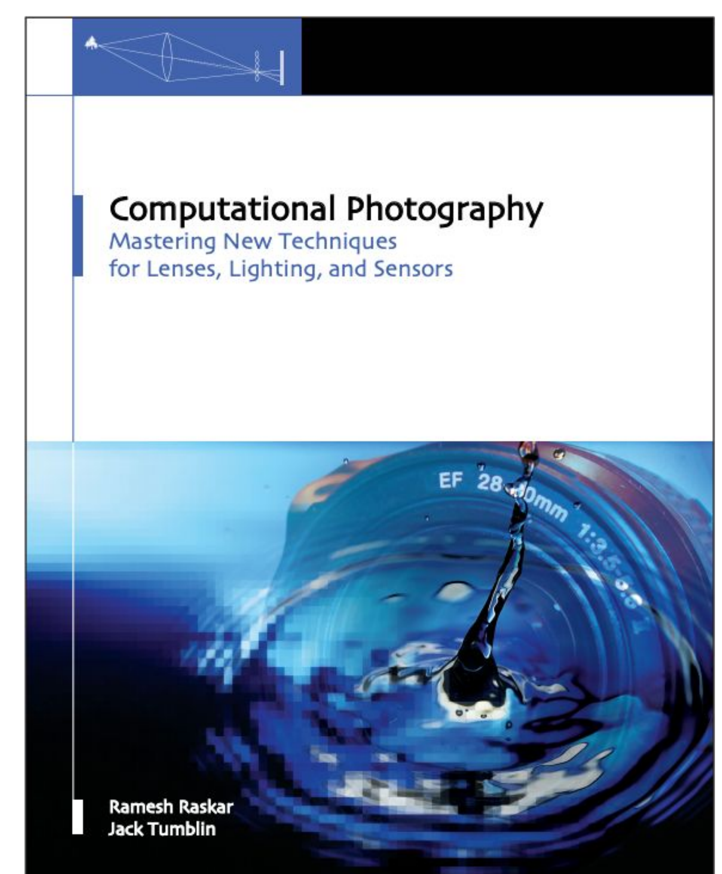
c) `.updateProjectionMatrix()`

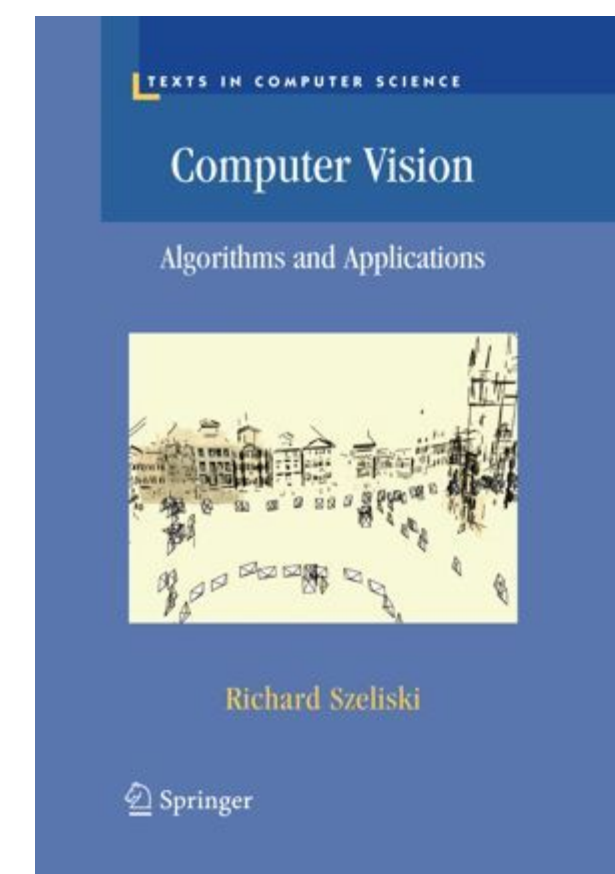
d) (0, 0, 0)

Take Away

- Understanding the whole transformation process (pipeline) is absolutely important
- The camera is a powerful tool to express visual effects not only for photography but also can be applied in computer-generated animations
- To realize camera effects in CG, you will need more knowledge in understanding practical photography and how it can be applied. We encourage you to check these books:







Thanks!
What are your questions?