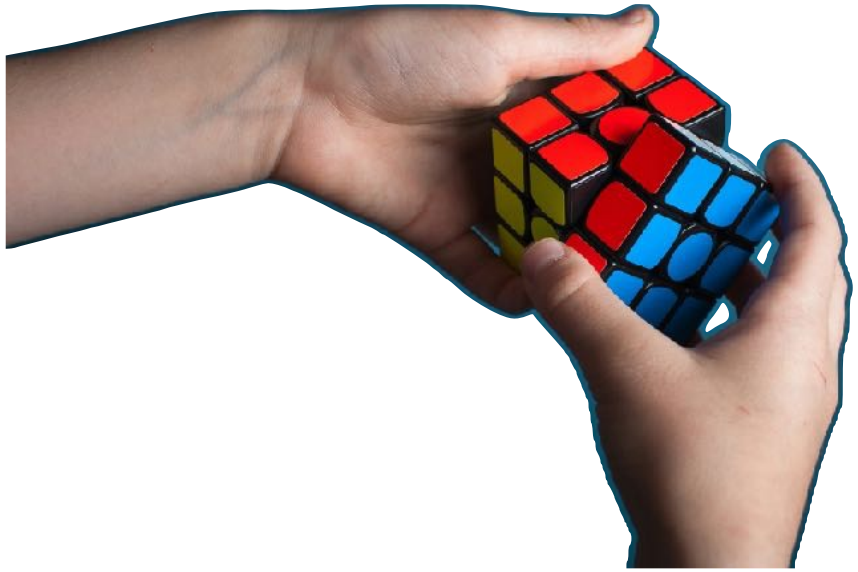
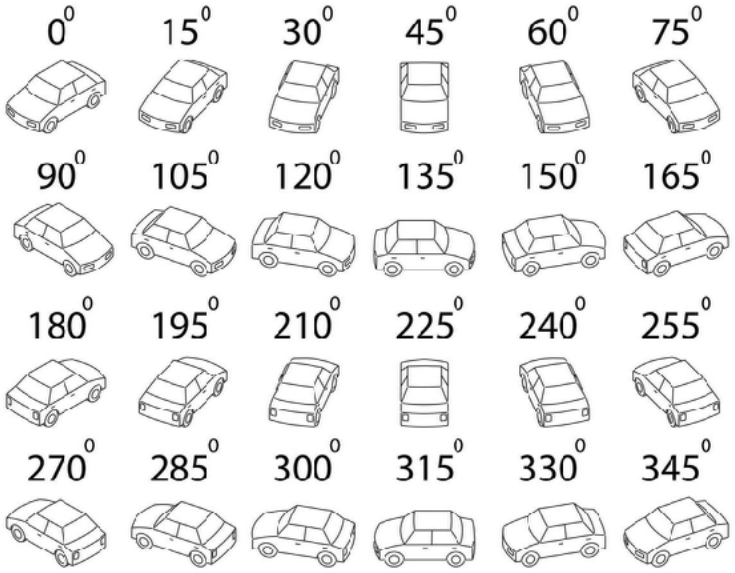
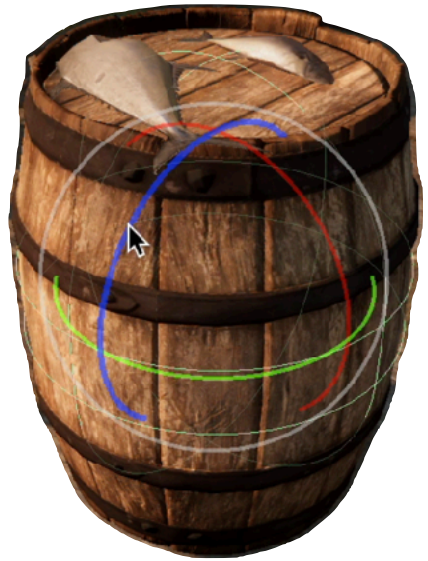
Visual Computing I:

Interactive Computer Graphics and Vision







3D Geometry and Cameras

Stefanie Zollmann and Tobias Langlotz

Last time..

Homographies and Image Stitching

- To stitch an image:
  - Need to warp images to align
  - This warping is a homography

- How to find the homography?
  - If a feature at (u,v) in one image matches to (u',v') in the other, then

$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} \equiv \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Do we still have a problem?

- For each point we get constraints of the form:

$$\begin{bmatrix} 0 & 0 & 0 & -u_i & -v_i & -1 & u_i v_i' & v_i v_i' & v_i' \\ u_i & v_i & 1 & 0 & 0 & 0 & -u_i u_i' & -v_i u_i' & -u_i' \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_9 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

  - us and vs are pixel values – on the order of 100 or 1000
  - So errors in h_3 and h_6 get multiplied by 1
  - Errors in h_7 and h_8 get multiplied by 10,000 to 1,000,000

- This makes the solution very unstable

Algorithm: Normalized DLT

Input: $n \geq 4$ correspondences $\mathbf{u}_i \leftrightarrow \mathbf{u}_i'$

Output: Homography, H, such that $\mathbf{u}_i' = \mathrm{H}\mathbf{u}_i$

1. **Normalisation:** Find T and T' to centre $\tilde{\mathbf{u}}_i = \mathrm{T}\mathbf{u}_i$ and $\tilde{\mathbf{u}}_i' = \mathrm{T}'\mathbf{u}_i'$ on the origin with average length $\sqrt{2}$

2. **Direct Linear Transform:**
   1. Form the matrix A from $\tilde{\mathbf{u}}_i$ and $\tilde{\mathbf{u}}_i'$
   2. Compute the SVD of A, and find the smallest eigenvector, $\tilde{\mathbf{h}}$
   3. Reshape $\tilde{\mathbf{h}}$ to give the 3×3 homography matrix $\tilde{\mathrm{H}}$

3. **Denormalisation:** Final solution is $\mathrm{H} = \mathrm{T}'^{-1}\tilde{\mathrm{H}}\mathrm{T}$

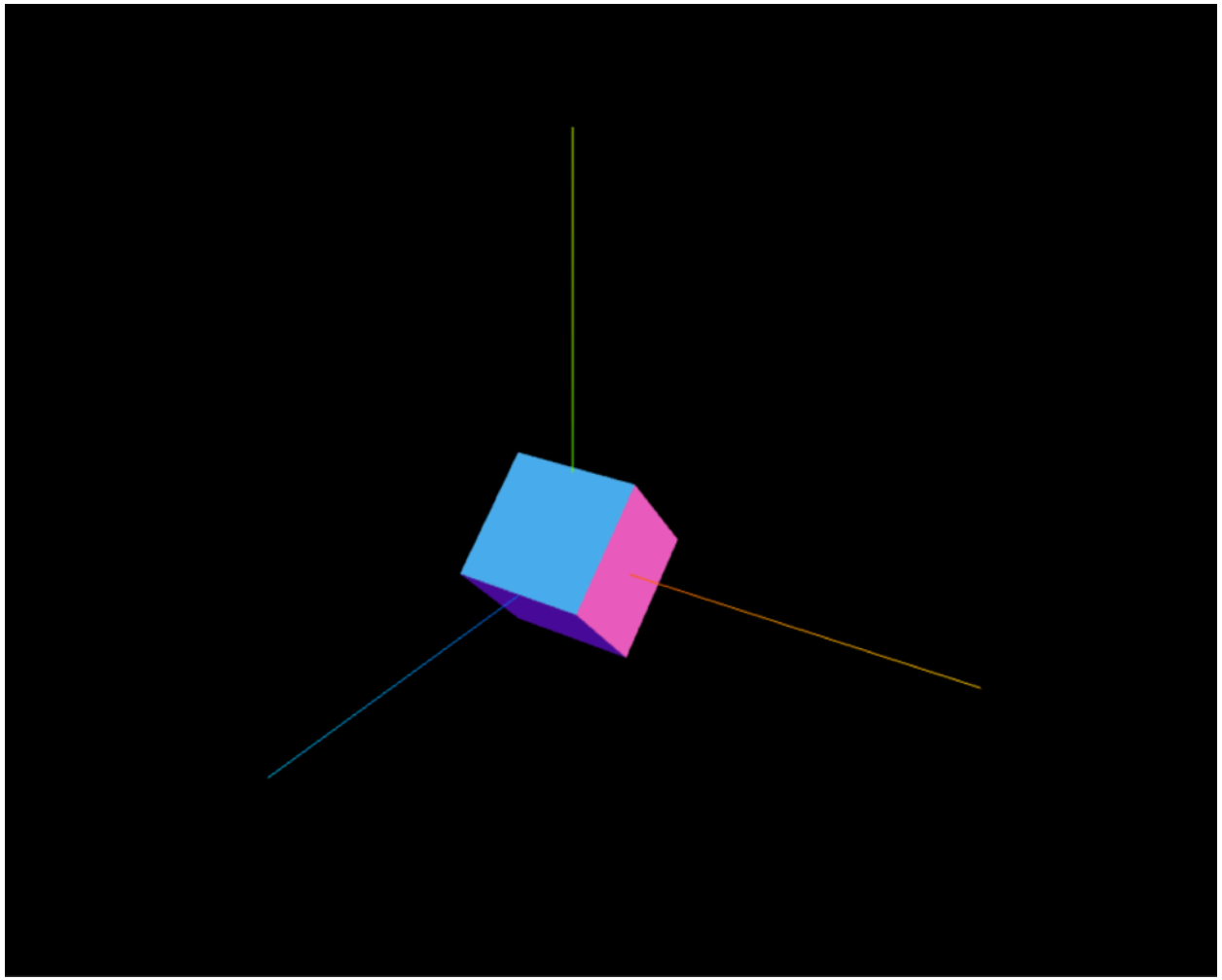
RANSAC for General Model Fitting

- We need:
  - A set of n points or items, p_i
  - To be able to fit a model, M, to $k \ll n$ points
  - To find the distance, $d(M, p_i)$
- RANSAC has parameters:
  - A threshold for acceptance
  - A number of trials (see later)
- RANSAC then proceeds as for the line fitting, except:
  - We select k items at random
  - We fit our model, M, to them

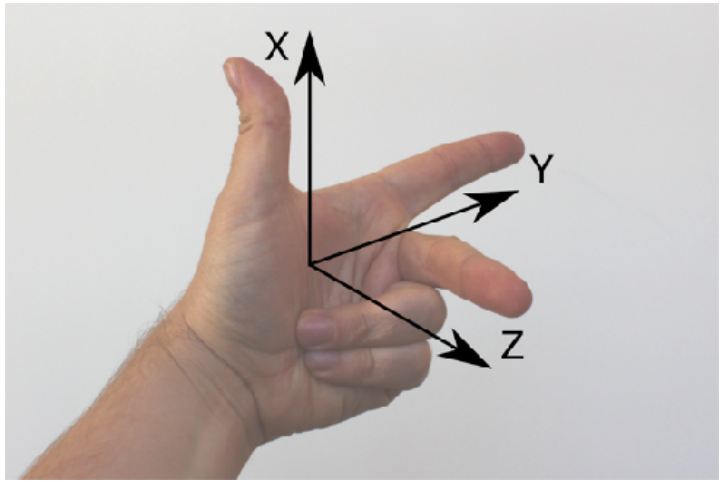
This time: 3D Geometry/Cameras

Three Dimensions

- We move from 2D to 3D
- Many ideas the same:
  - Homogeneous co-ordinates
  - Scaling and translation
- Some things get tricky
  - Choice of left- or right- handed co-ordinates
  - Rotations get complicated
- Projection from 3D to 2D



Left & Right-Handed Coordinates

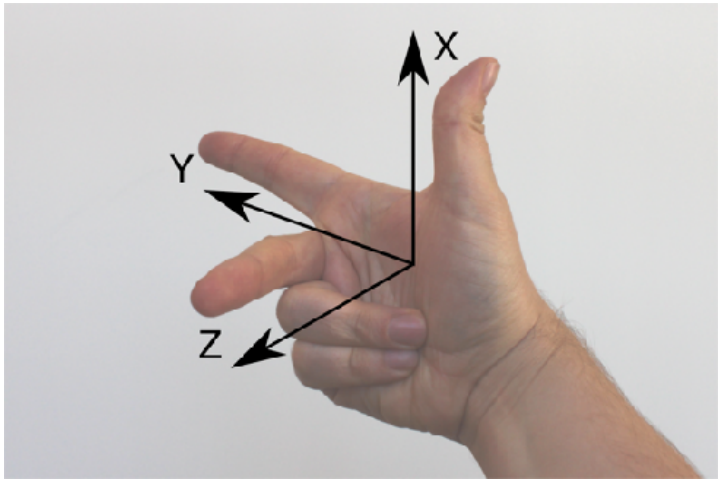


Left handed

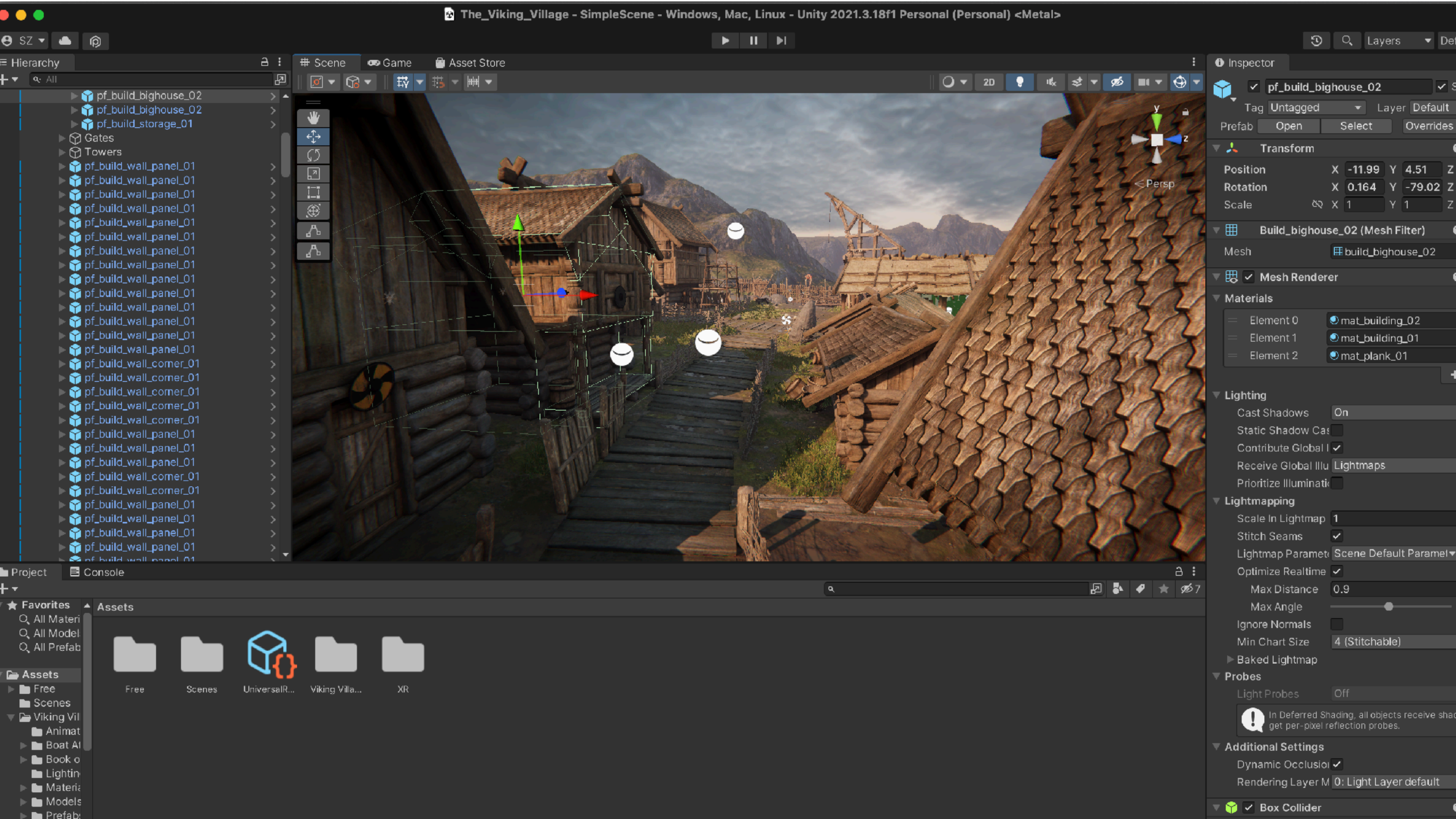
Thumb is axis

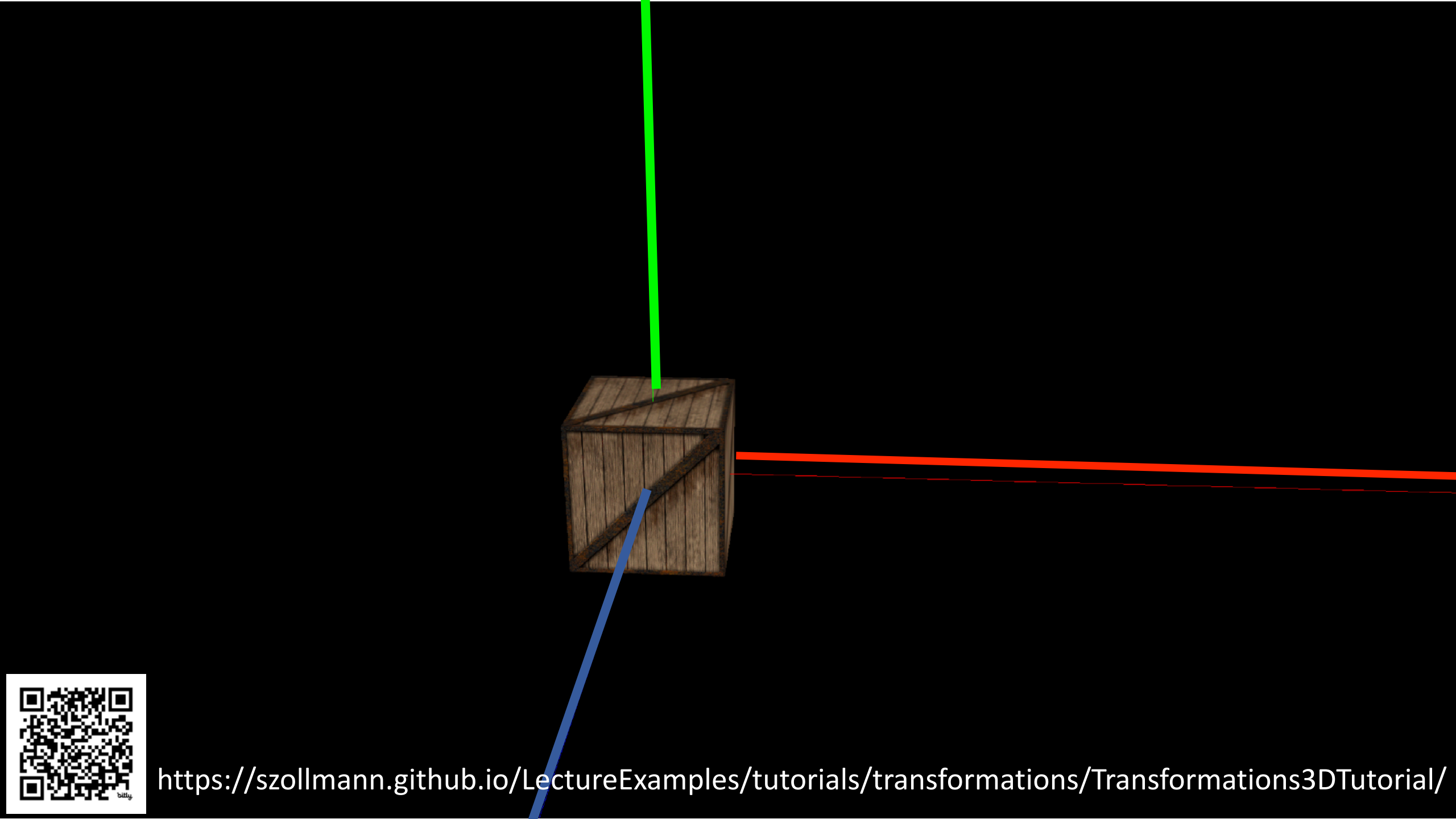
Forefinger is axis

Middle finger is axis



Right-Handed


The_Viking_Village - SimpleScene - Windows, Mac, Linux - Unity 2021.3.18f1 Personal (Personal) <Metal>
Hierarchy
pf_build_bighouse_02
pf_build_bighouse_02
pf_build_storage_01
Gates
Towers
pf_build_wall_panel_01
pf_build_wall_corner_01
Scene
Game
Asset Store
Persp
Inspector
pf_build_bighouse_02
Tag Untagged
Layer Default
Prefab Open Select Overrides
Transform
Position X -11.99 Y 4.51 Z
Rotation X 0.164 Y -79.02 Z
Scale X 1 Y 1 Z
Build_bighouse_02 (Mesh Filter)
Mesh build_bighouse_02
Mesh Renderer
Materials
Element 0 mat_building_02
Element 1 mat_building_01
Element 2 mat_plank_01
Lighting
Cast Shadows On
Receive Global Illu Lightmaps
Lightmapping
Scale In Lightmap 1
Stitch Seams
Optimize Realtime
Max Distance 0.9
Max Angle
Ignore Normals
Min Chart Size 4 (Stitchable)
Baked Lightmap
Probes
Light Probes Off
In Deferred Shading, all objects receive shad
get per-pixel reflection probes.
Additional Settings
Dynamic Occlusio
Rendering Layer M 0: Light Layer default
Box Collider
Project
Console
Favorites
All Materi
All Model
All Prefab
Assets
Free
Scenes
Viking Vil
Animat
Boat Al
Book o
Lightin
Materia
Models
Prefab
Assets
Free
Scenes
UniversalR...
Viking Villa...
XR



https://szollmann.github.io/LectureExamples/tutorials/transformations/Transformations3DTutorial/

Homogeneous Coordinates in 3D

- In 3D we represent the point (x, y, z) as a set of 4-vectors

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow k \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \; k \neq 0$$

- The vector $\begin{bmatrix} a & b & c & d \end{bmatrix}^T$ corresponds to the point $\left(\frac{a}{d}, \frac{b}{d}, \frac{c}{d} \right)$
- Directions are of the form $\begin{bmatrix} x & y & z & 0 \end{bmatrix}^T$
- The basic transformations are now 4×4 matrices

Translation and Scaling in 3D

- Translation by $(\Delta x,\ \Delta y,\ \Delta z)$

$$\mathbf{x}' = \mathrm{T}\mathbf{x}$$

$$\mathrm{T} = \begin{bmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

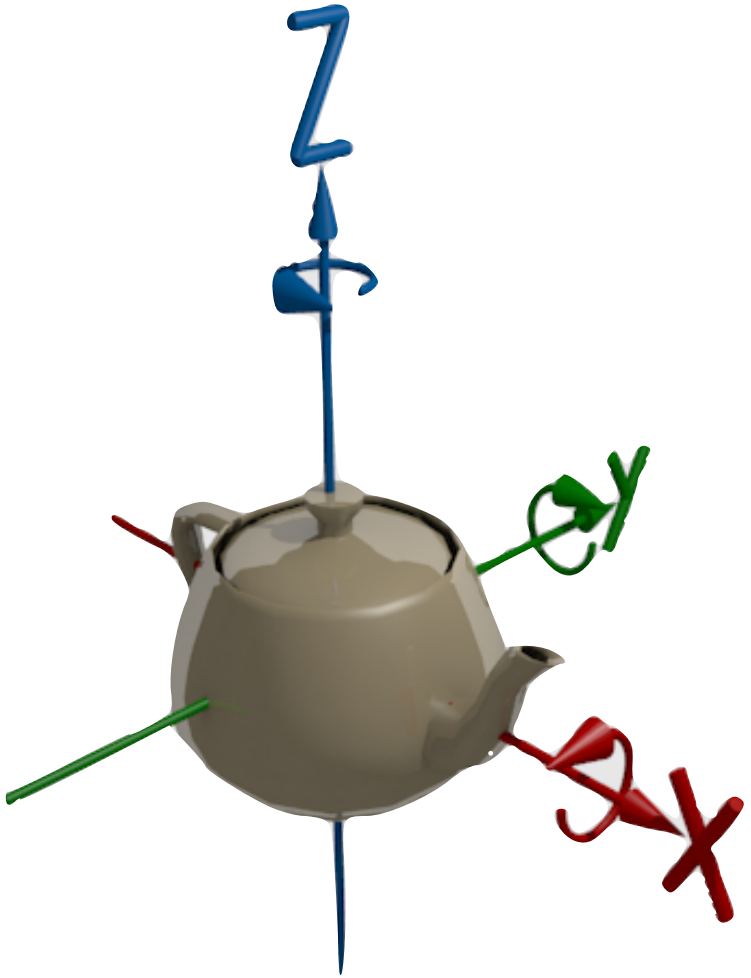
- Scaling by Scaling by a factor s

$$\mathbf{x}' = \mathrm{S}\mathbf{x}$$

$$\mathrm{S} = \begin{bmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation in 3D

- Rotation in 3D
  - More complex than in 2D
  - Three ‘degrees of freedom’
- There are many ways to represent 3D rotations
  - ‘Euler angles’ – rotations around X, Y, and Z axes
  - Yaw, pitch, roll
  - Some angle about an axis
  - Rotation Matrix
  - Quaternions

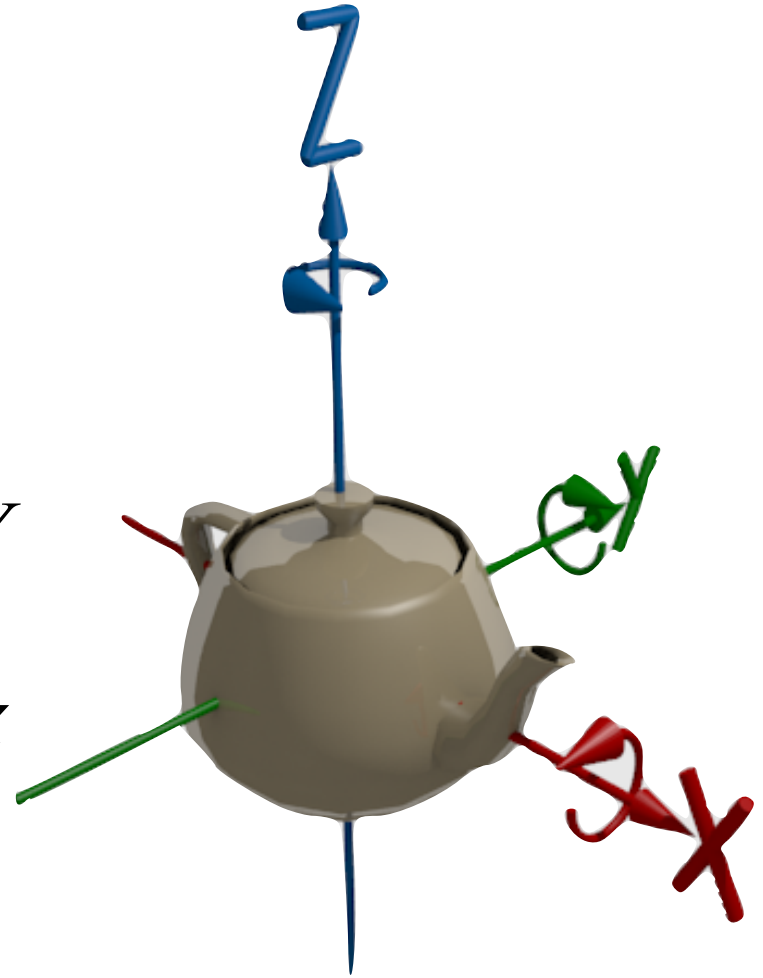


Rotation about X, Y, and Z Axes

- We know that a 2D rotation looks like

$$\mathrm{R}_\theta = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

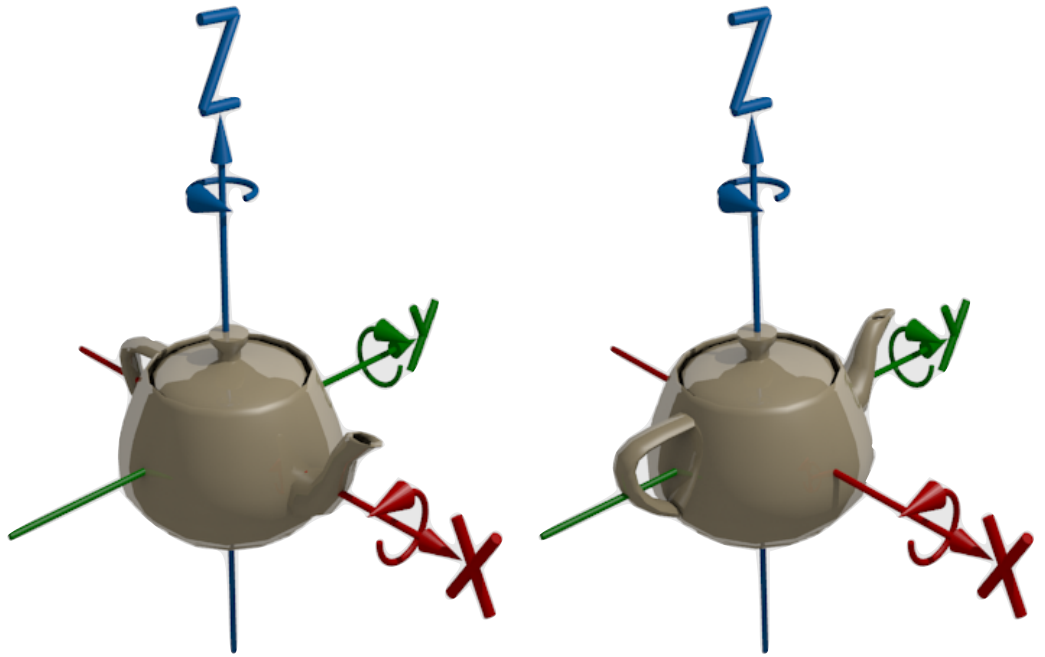
- This is a rotation **from** the U axis **to** the V axis
  - Replace ‘rotation about X’ with ‘rotation **from** the Y **to** Z axis’
  - Replace ‘rotation about Y’ with ‘rotation **from** the Z **to** X axis’
  - Replace ‘rotation about Z’ with ‘rotation **from** the X **to** Y axis’



Rotation about the Z Axis

- From X to Y
  - Top left corner is 2D rotation
  - Z co-ordinate is unchanged

$$R_Z = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

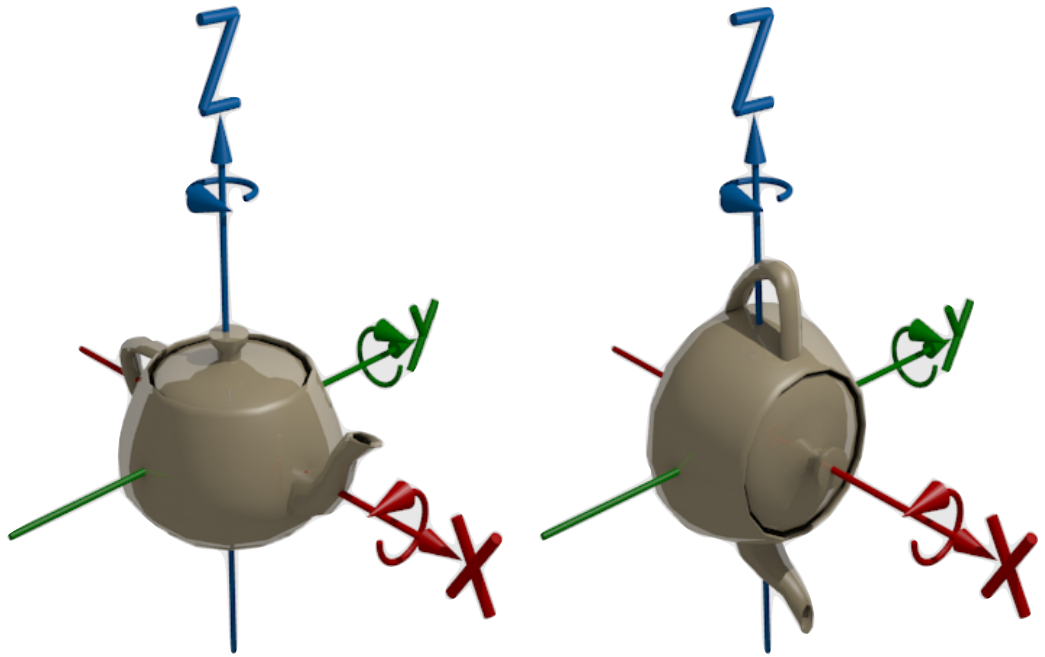


Rotation by 90° about the Z axis – from X to Y

Rotation about the Y Axis

- From Z to X
  - Y co-ordinate is unchanged

$$R_Y = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

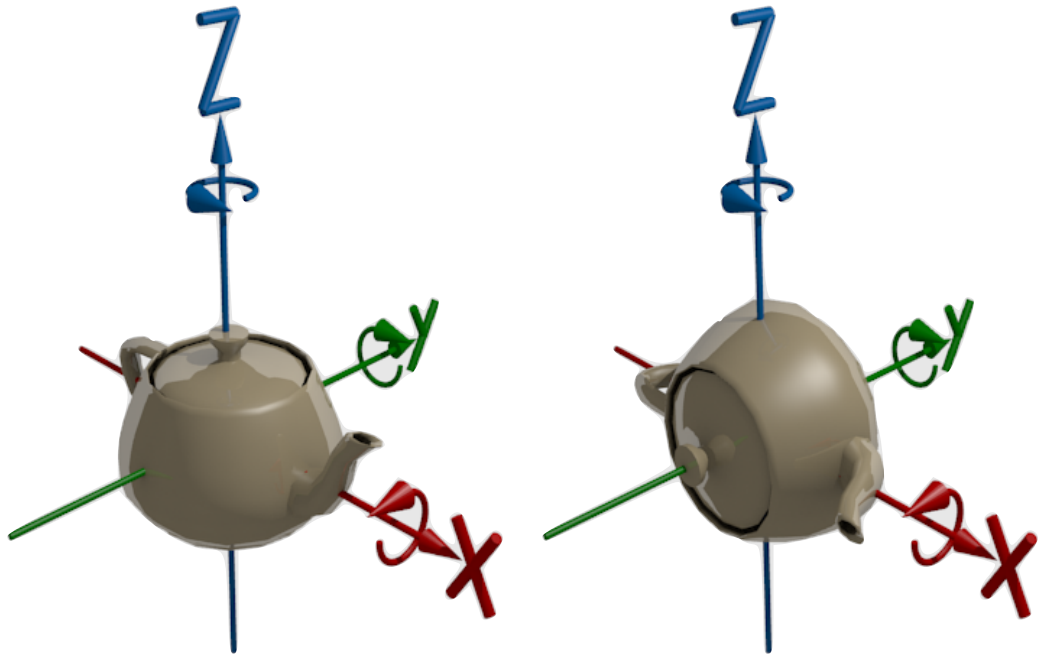


Rotation by $90°$ about the Z axis – from X to Y

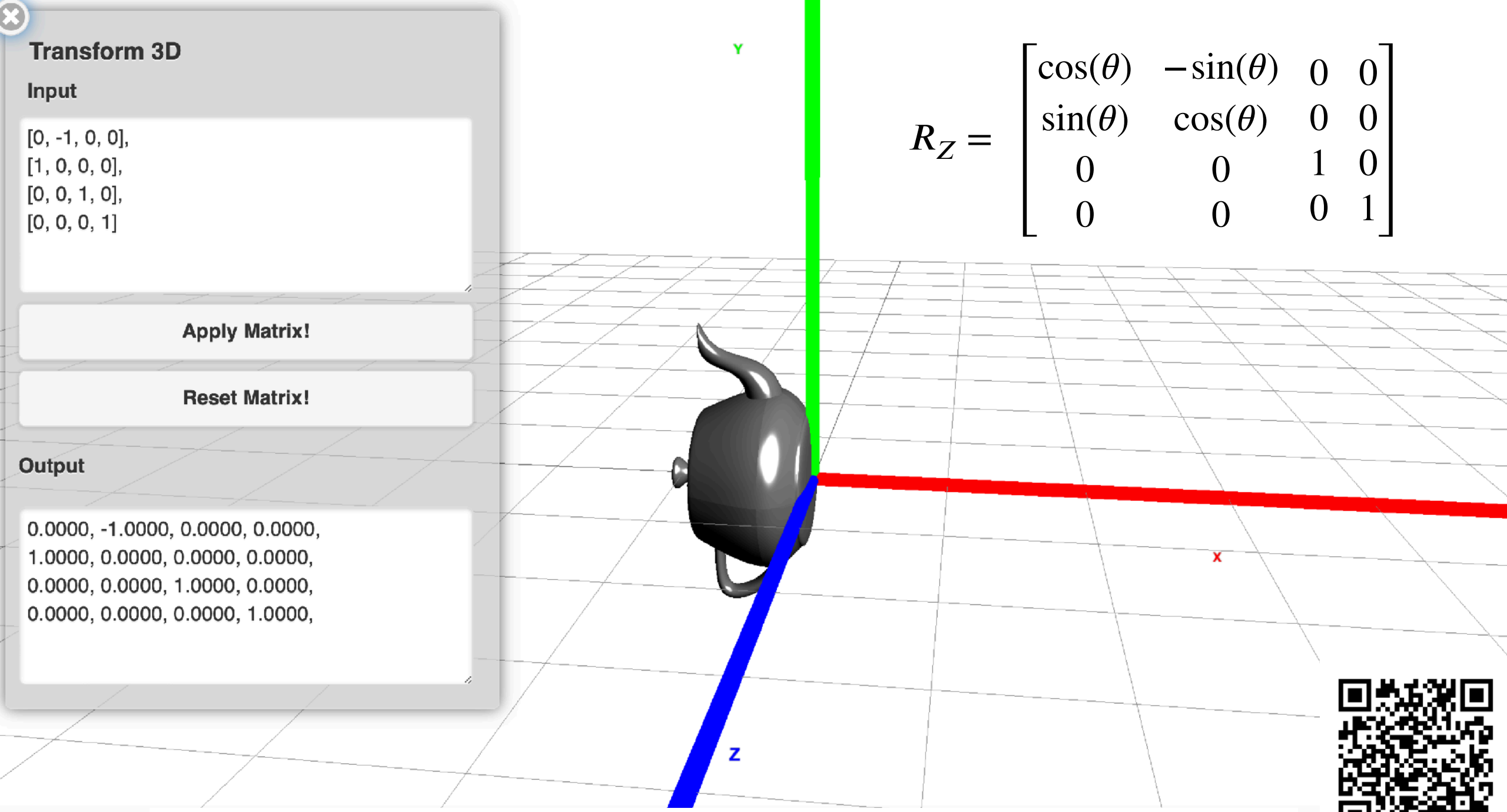
Rotation about the X Axis

- From Y to Z
  - X co-ordinate is unchanged

$$R_X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Rotation by $90°$ about the Z axis – from X to Y



$$R_Z = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

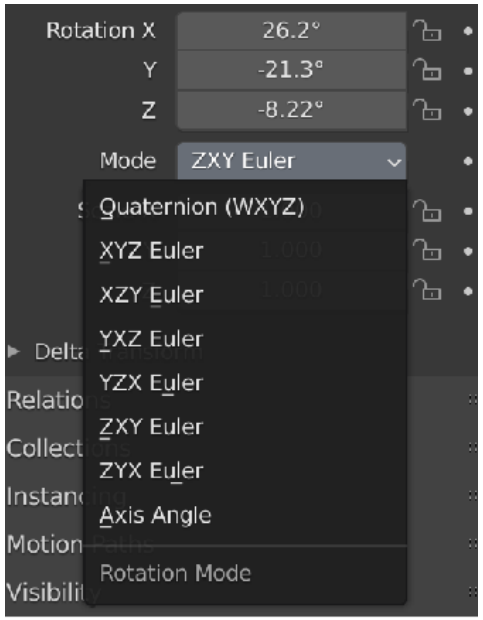
https://szollmann.github.io/LectureExamples/tutorials/transformations/Transformations3DTeapot/

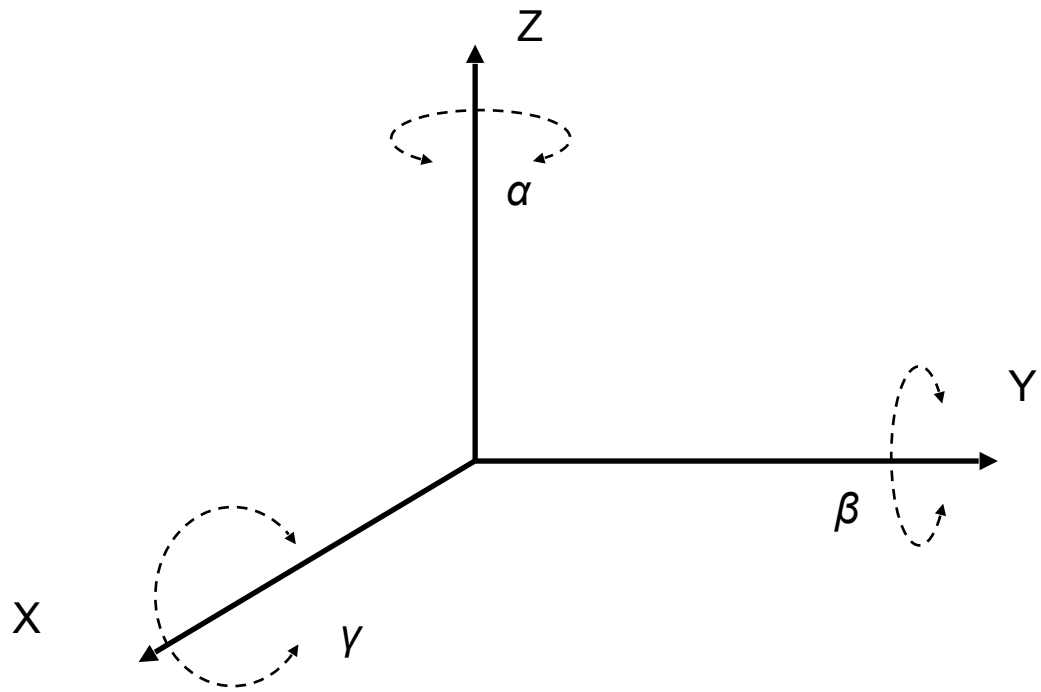
Summary of 3D Transformations

- Translation $T(d)=\begin{pmatrix}1&0&0&d_x\\0&1&0&d_y\\0&0&1&d_z\\0&0&0&1\end{pmatrix}$ Inverse Translation $T^{-1}(d)=T(-d)=\begin{pmatrix}1&0&0&-d_x\\0&1&0&-d_y\\0&0&1&-d_z\\0&0&0&1\end{pmatrix}$
- Scale $S(s)=\begin{pmatrix}s_x&0&0&0\\0&s_y&0&0\\0&0&s_z&0\\0&0&0&1\end{pmatrix}$ Inverse Scale $S^{-1}(s)=S\left(\frac{1}{s}\right)=\begin{pmatrix}1/s_x&0&0&0\\0&1/s_y&0&0\\0&0&1/s_z&0\\0&0&0&1\end{pmatrix}$
- Rotation $R_z(\theta)=\begin{pmatrix}\cos\theta&-\sin\theta&0&0\\\sin\theta&\cos\theta&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}$ Inverse Rotation $R_z^{-1}(\theta)=R_z(-\theta)=\begin{pmatrix}\cos-\theta&-\sin-\theta&0&0\\\sin-\theta&\cos-\theta&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}$

Euler Angles

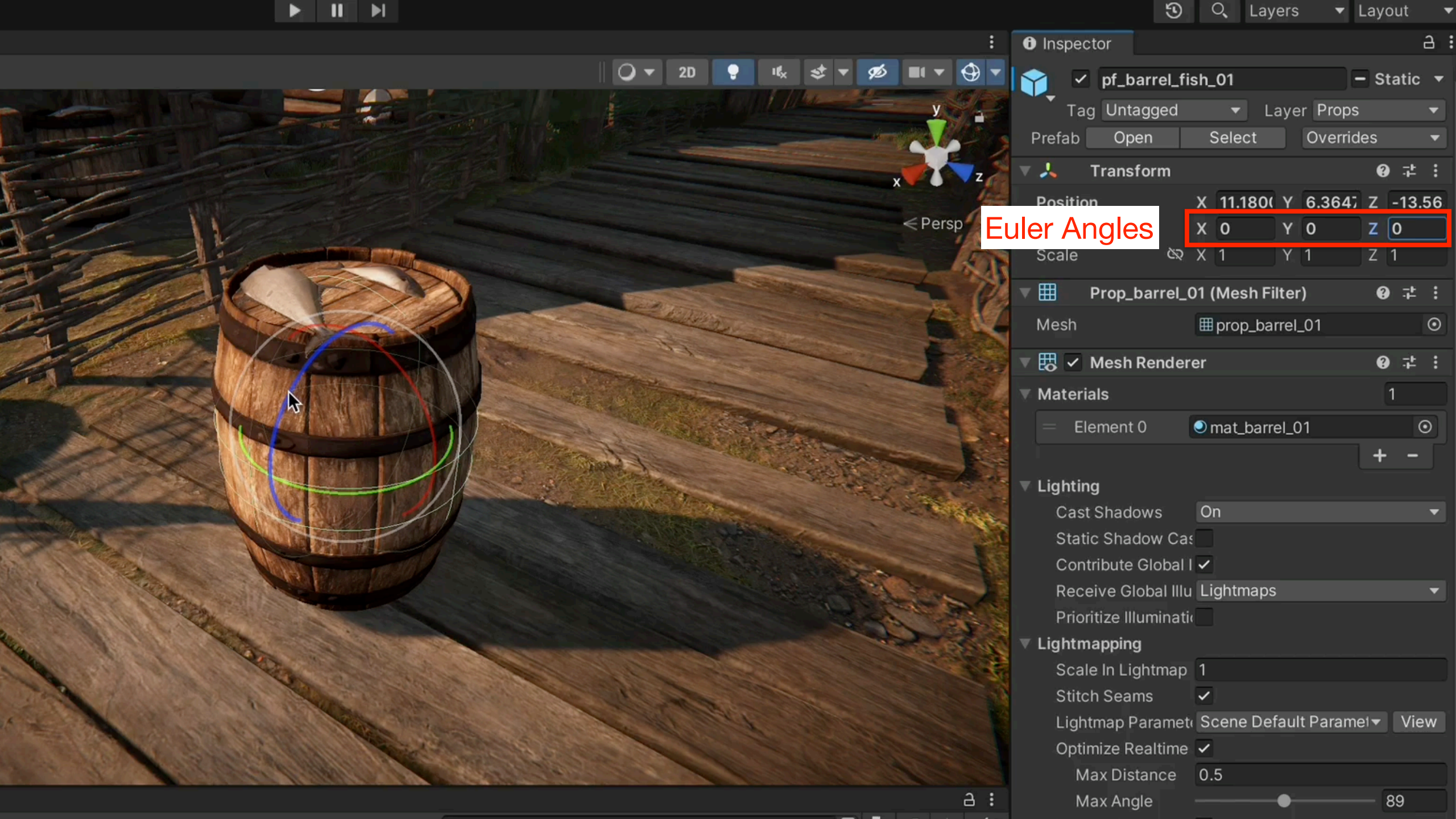
- Rotation defined by angles of rotation around the X-, Y-, and Z- axes
- Different conventions
  - Example Blender Rotation Mode:





$$\mathbf{R} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix}$$

$$\mathbf{R} = \mathbf{R_Z R_Y R_X}$$


Layers
Layout
Inspector
pf_barrel_fish_01
Static
Tag Untagged
Layer Props
Prefab Open Select Overrides
Transform
Position
X 11.180(Y 6.364; Z -13.56
Euler Angles
X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Prop_barrel_01 (Mesh Filter)
Mesh prop_barrel_01
Mesh Renderer
Materials 1
Element 0 mat_barrel_01
Lighting
Cast Shadows On
Static Shadow Cas
Contribute Global I
Receive Global Illu Lightmaps
Prioritize Illuminati
Lightmapping
Scale In Lightmap 1
Stitch Seams
Lightmap Paramet Scene Default Parame View
Optimize Realtime
Max Distance 0.5
Max Angle 89
Persp
y
x
z
2D

Challenges in 3D Rotations

Euler Angles and the Gimbal Lock

- Gimbal system can be used to visualize Euler Angles
- Specific configuration caused by using Euler Angles
  - Aligning two axes (here red and blue)
  - Rotation around red and blue is around the same axis
  - Lost a degree of freedom / cannot rotate freely

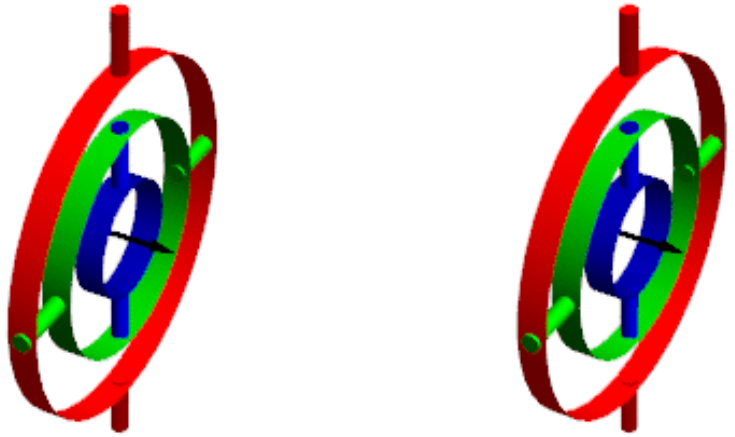
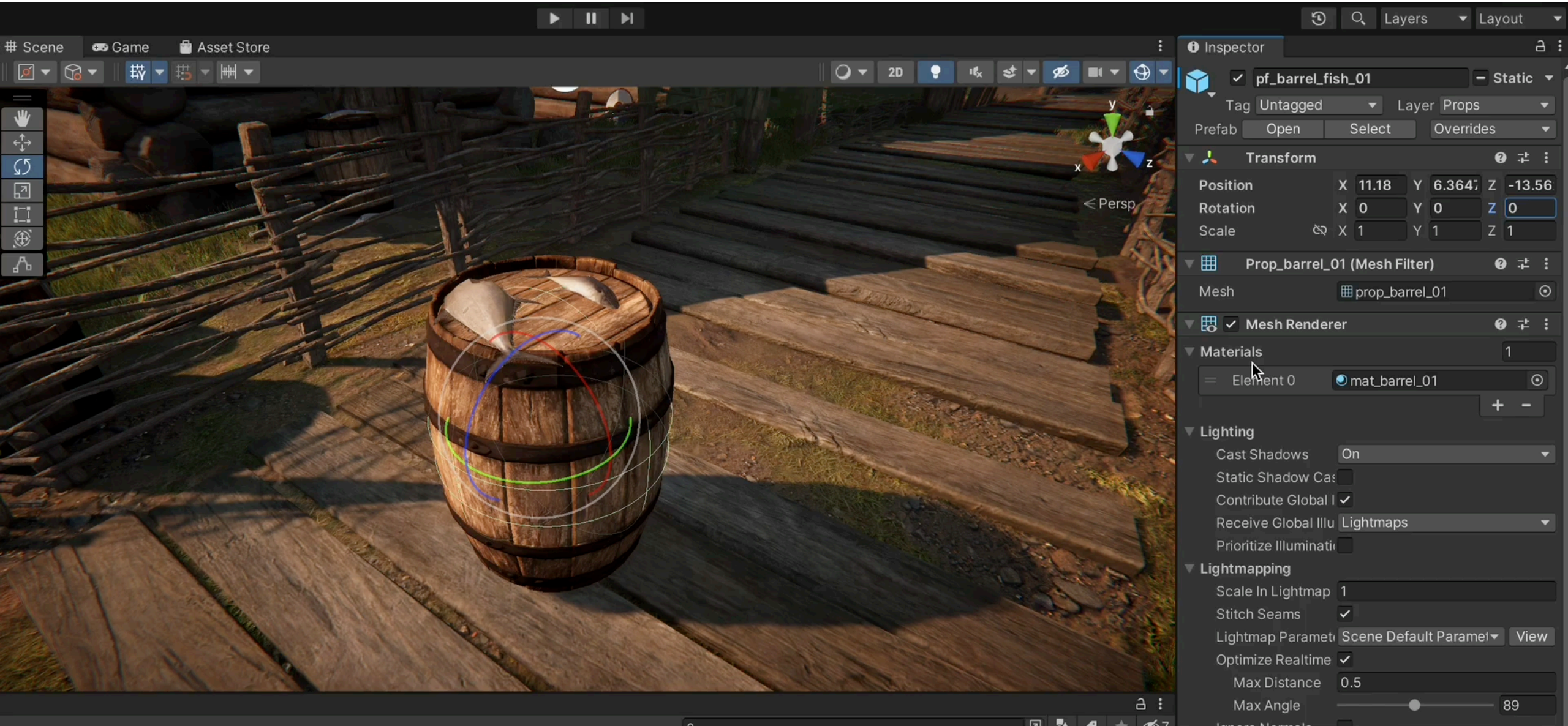


Image credit: Mark Hughes.

Unity Demo

Euler Angles and the Gimbal Lock



Euler Angles and the Gimbal Lock

- When $\beta = 90°$,

$$\mathbf{R} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix}$$

$$= \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix}$$

$$= \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & \sin\gamma & \cos\gamma \\ 0 & \cos\gamma & -\sin\gamma \\ 1 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & \cos\alpha\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\cos\gamma + \sin\alpha\sin\gamma \\ 0 & \sin\alpha\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\cos\gamma - \cos\alpha\sin\gamma \\ -1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & \sin(\gamma-\alpha) & \cos(\gamma-\alpha) \\ 0 & \cos(\gamma-\alpha) & -\sin(\gamma-\alpha) \\ -1 & 0 & 0 \end{bmatrix}$$

Euler Angles and the Gimbal Lock

- When $\beta = 90°$,

$$\mathbf{R} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix}$$

$$= \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix}$$

$$= \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & \sin\gamma & \cos\gamma \\ 0 & \cos\gamma & -\sin\gamma \\ 1 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & \cos\alpha\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\cos\gamma + \sin\alpha\sin\gamma \\ 0 & \sin\alpha\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\cos\gamma - \cos\alpha\sin\gamma \\ -1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & \sin(\gamma-\alpha) & \cos(\gamma-\alpha) \\ 0 & \cos(\gamma-\alpha) & -\sin(\gamma-\alpha) \\ -1 & 0 & 0 \end{bmatrix}$$

Rotation matrix loses one degree of freedom:
yaw and roll have merged when $\beta = 90°$

Can you gimbal lock?

https://szollmann.github.io/LectureExamples/tutorials/transformations/gimbal.html

bit.ly/46xexuy

Quaternions

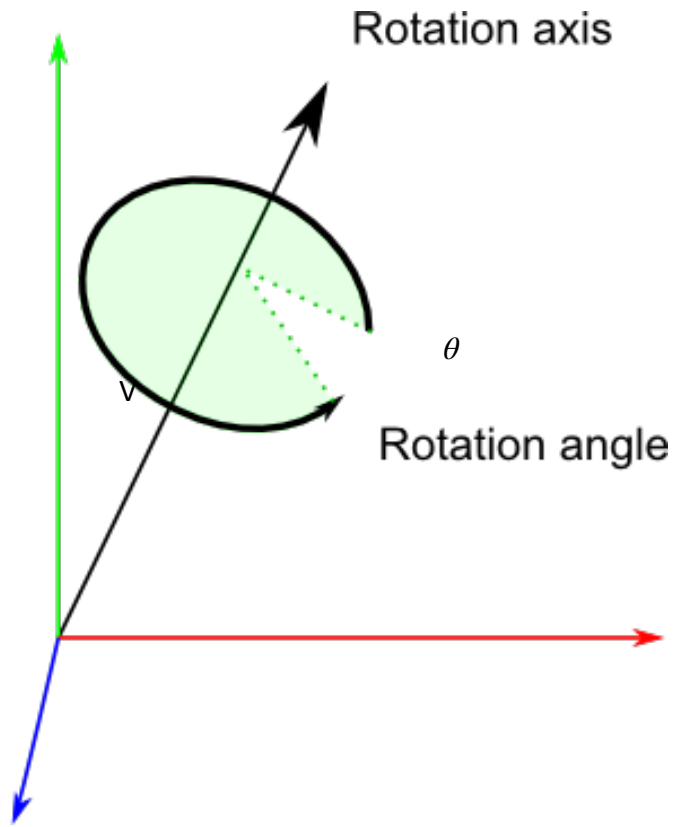
Quaternion

- Another way of presenting rotations (avoiding Gimbal Lock)
  - Can be expressed as a scalar and a 3- vector: (a, v)
  - A rotation about the unit vector v by an angle θ can be represented by the unit quaternion

angle in radians, unit-length axis

$$q(\theta, v) = \left(\cos\frac{\theta}{2}, v_x \sin\frac{\theta}{2}, v_y \sin\frac{\theta}{2}, v_z \sin\frac{\theta}{2} \right)$$

scalar = angle; vector = axis



http://www.opengl-tutorial.org

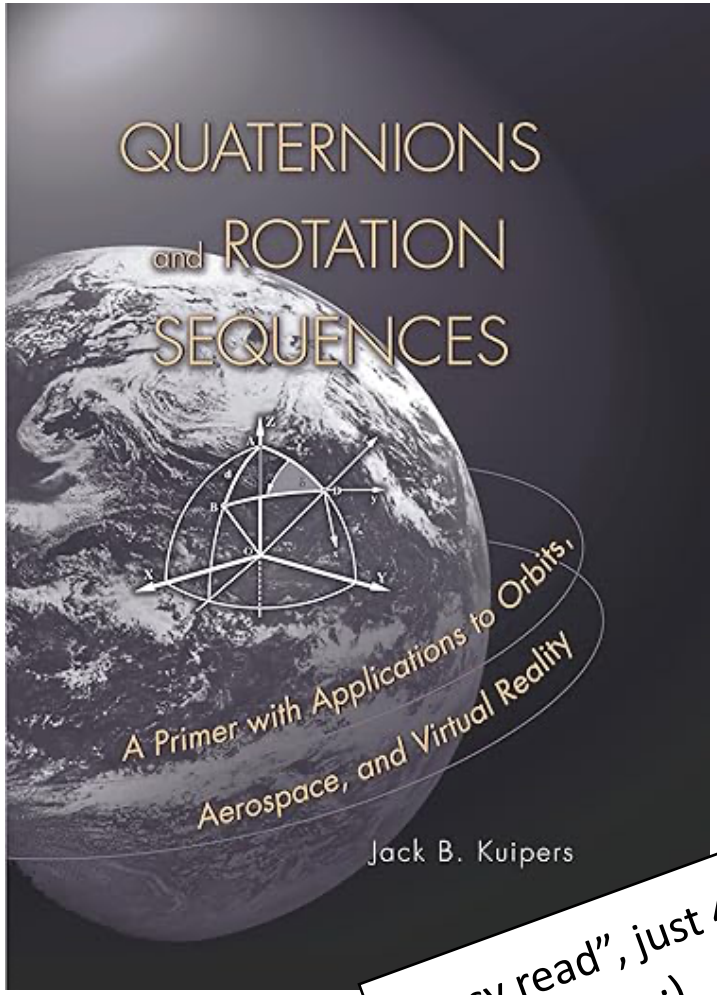
Quaternion

- Based on an extension of complex numbers
  - 4-dimensional
  - Definitions

$$(x, y, z, w) = \underbrace{x\mathbf{i} + y\mathbf{j} + z\mathbf{k}}_{imaginary} + \underbrace{w}_{real}$$

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

  - Mathematical notation for representing orientations and rotations of objects in three dimensions



"Easy read", just 400 pages ;)

Can you quaternion?

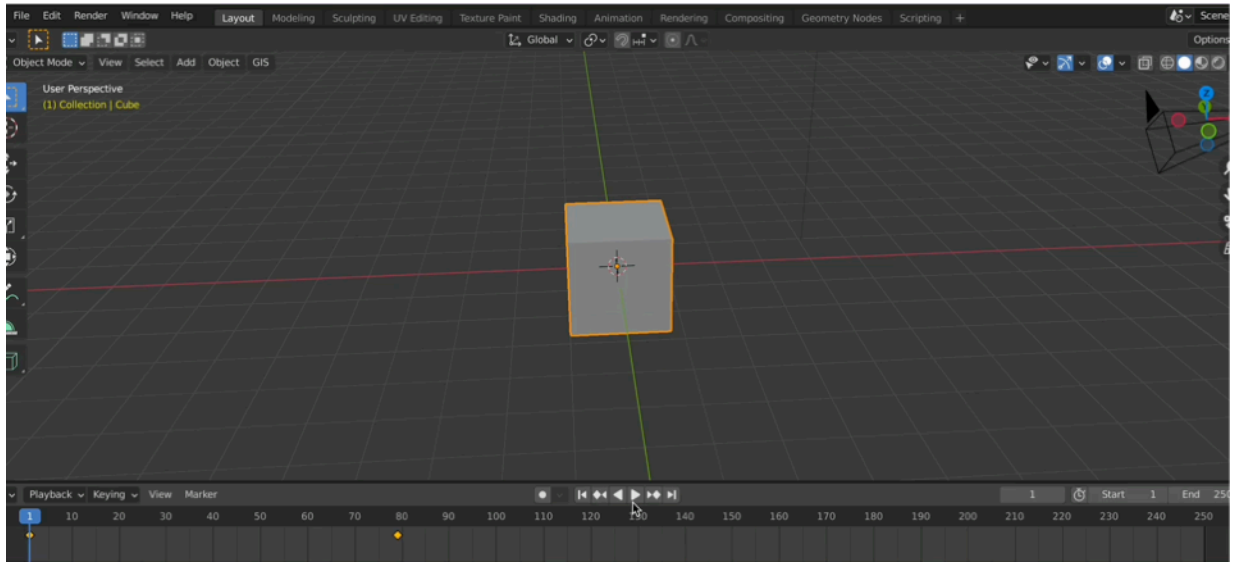
https://szollmann.github.io/LectureExamples/tutorials/transformations/quaternions.html

Outlook Animations

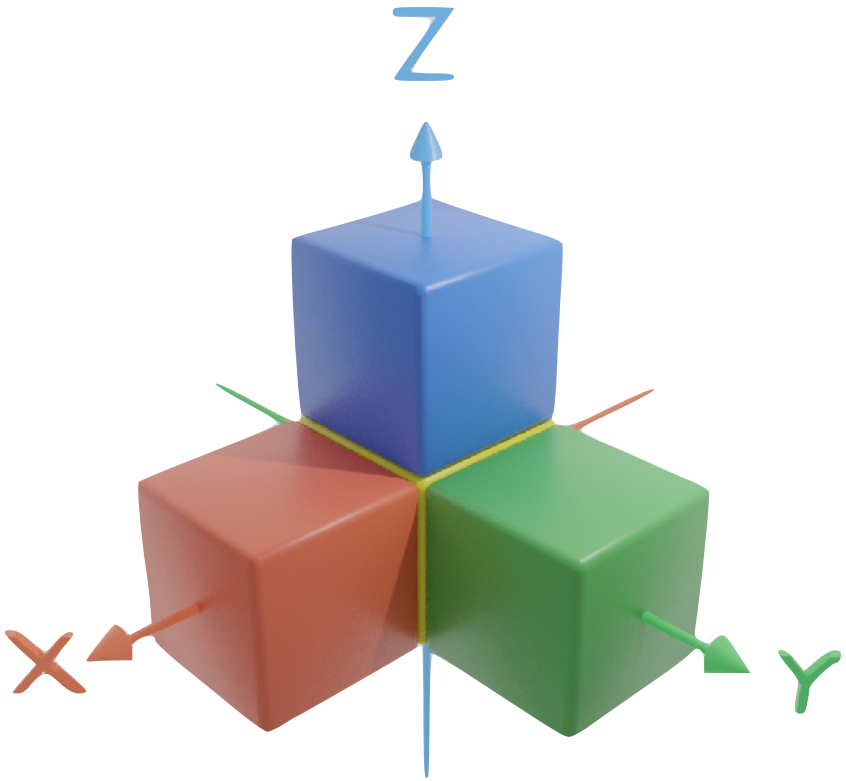
- Define first and last position
- Use linear interpolation to determine other position

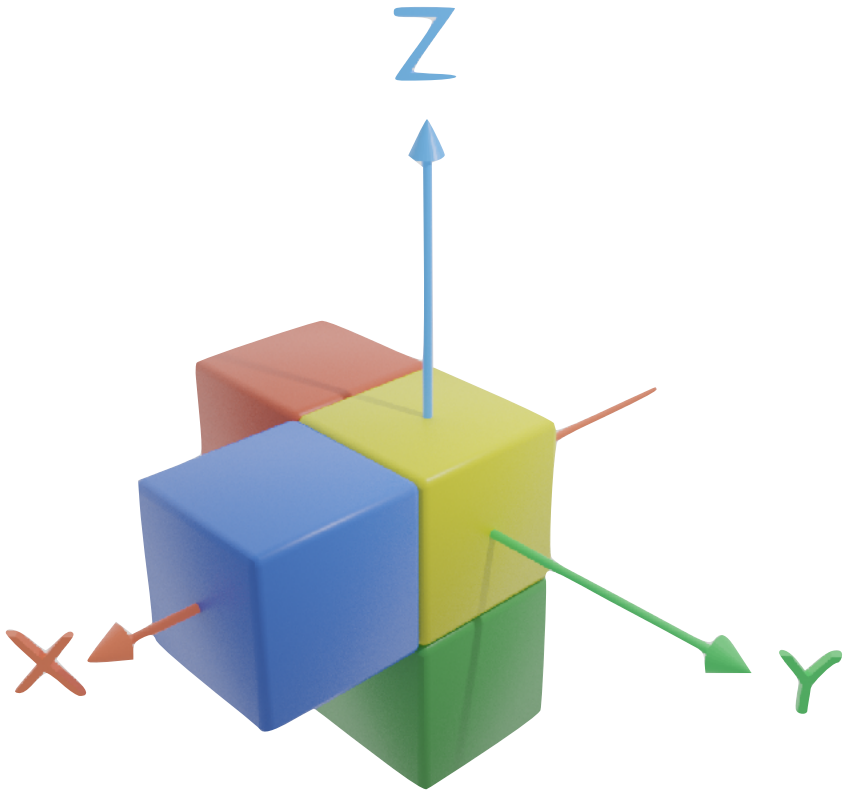
$$lerp\left(p_0, p_1; t\right) = \left(1 - t\right) p_0 + t p_1$$

- Linear interpolation of rotation?

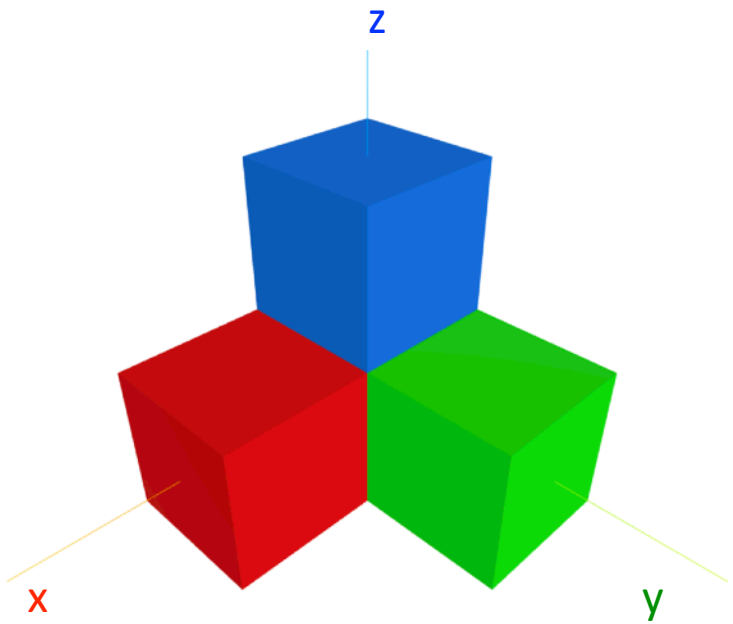


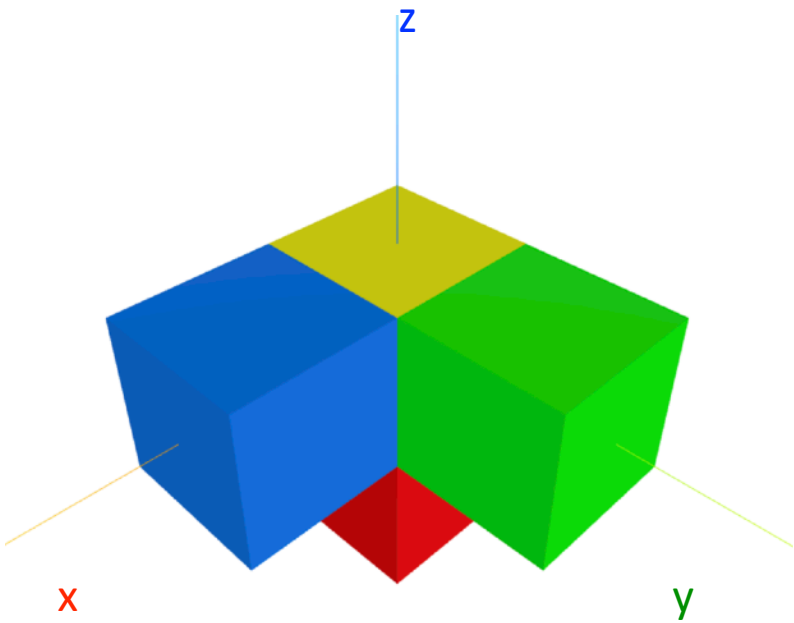
Quiz: Rotation in 3D



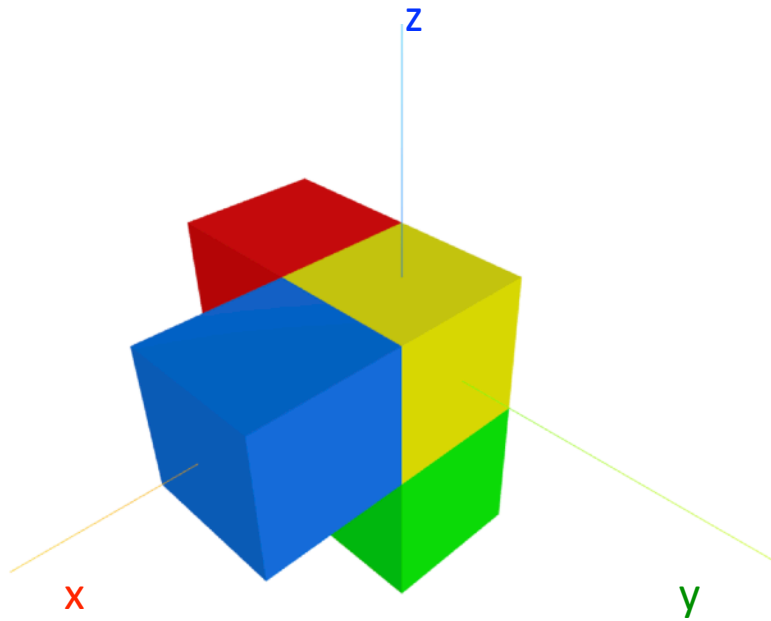


Answer



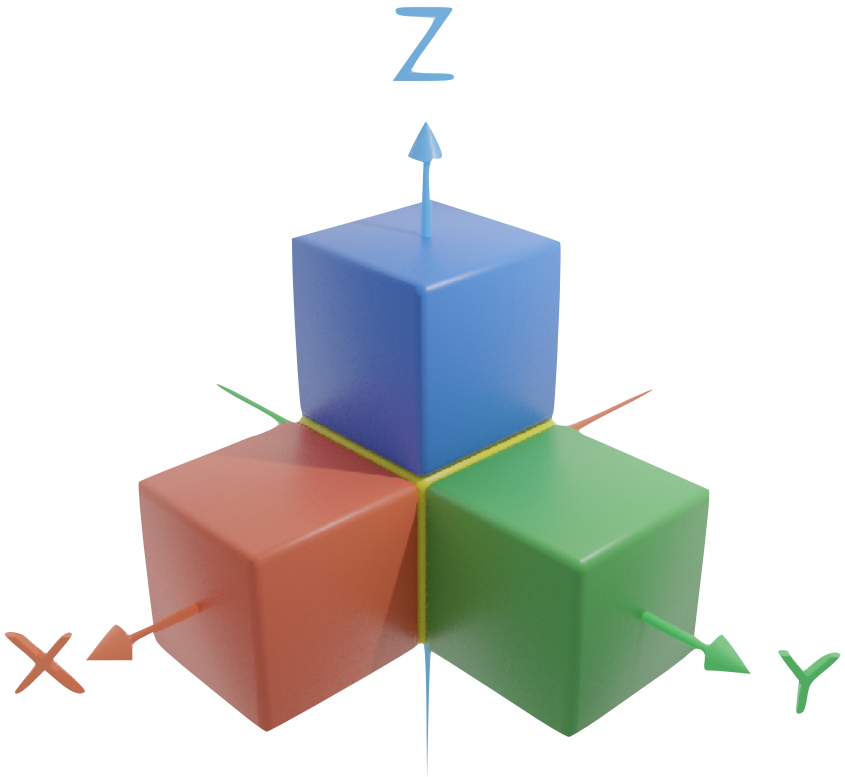


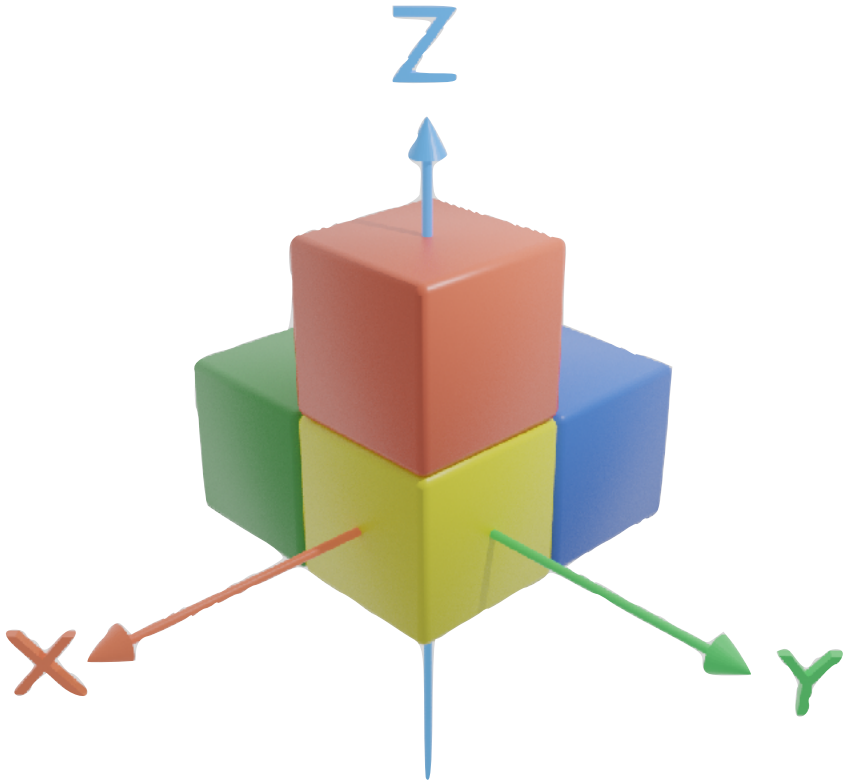
1. 90 Degree around y- Axis



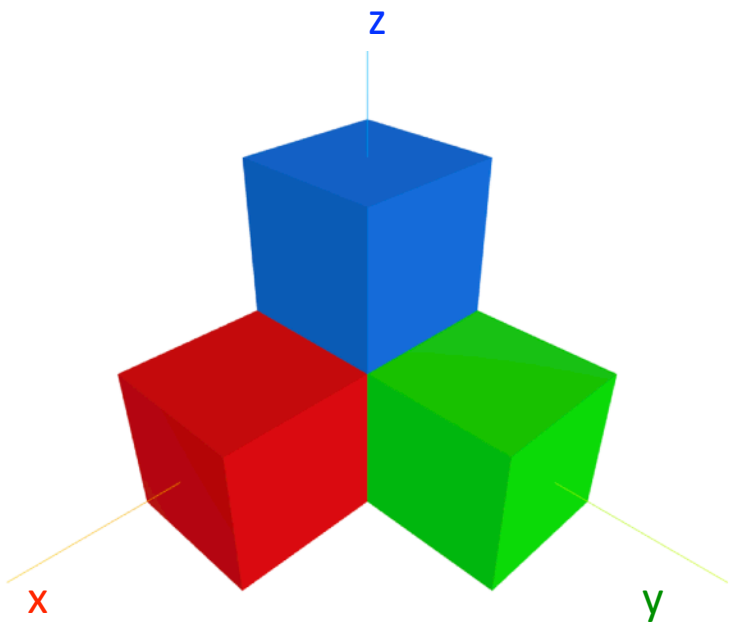
2. 270 Degree around y-Axis

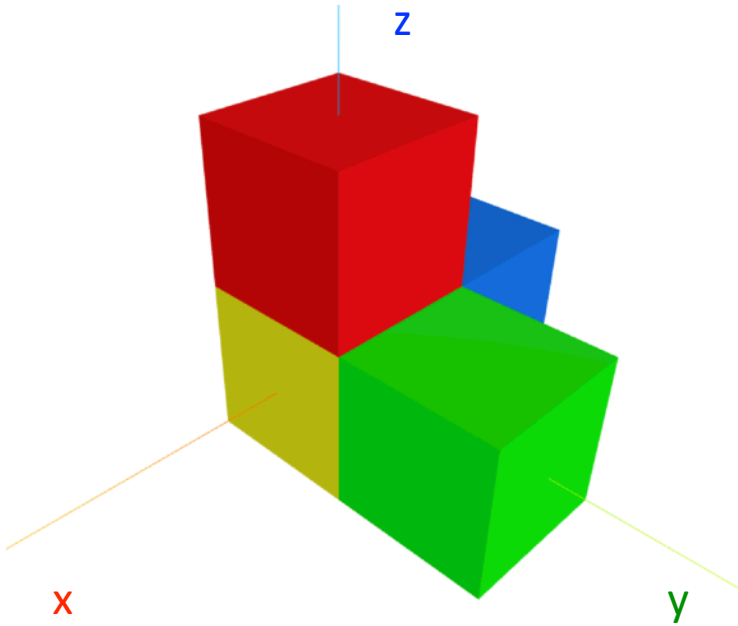
Quiz: Rotation in 3D



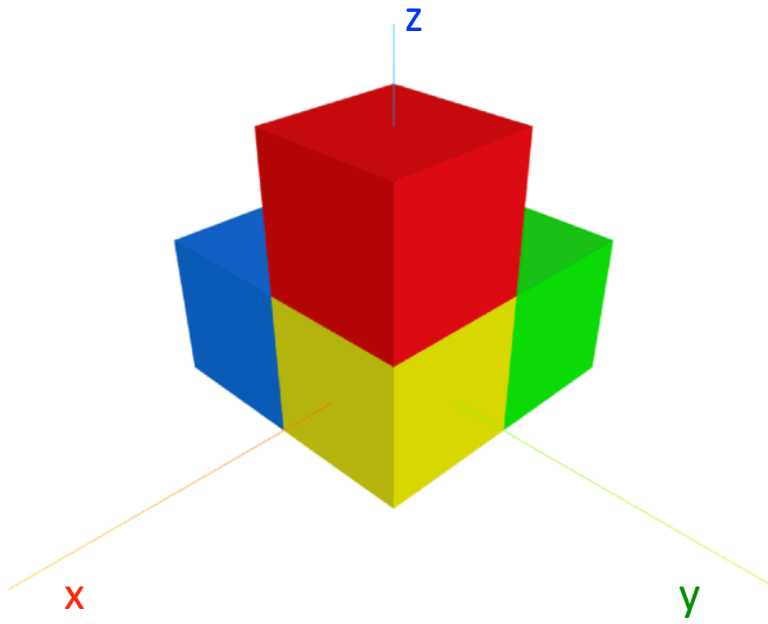


Answer: Not Possible





1. 270 Degree around y- Axis



2. 90 Degree around z-Axis

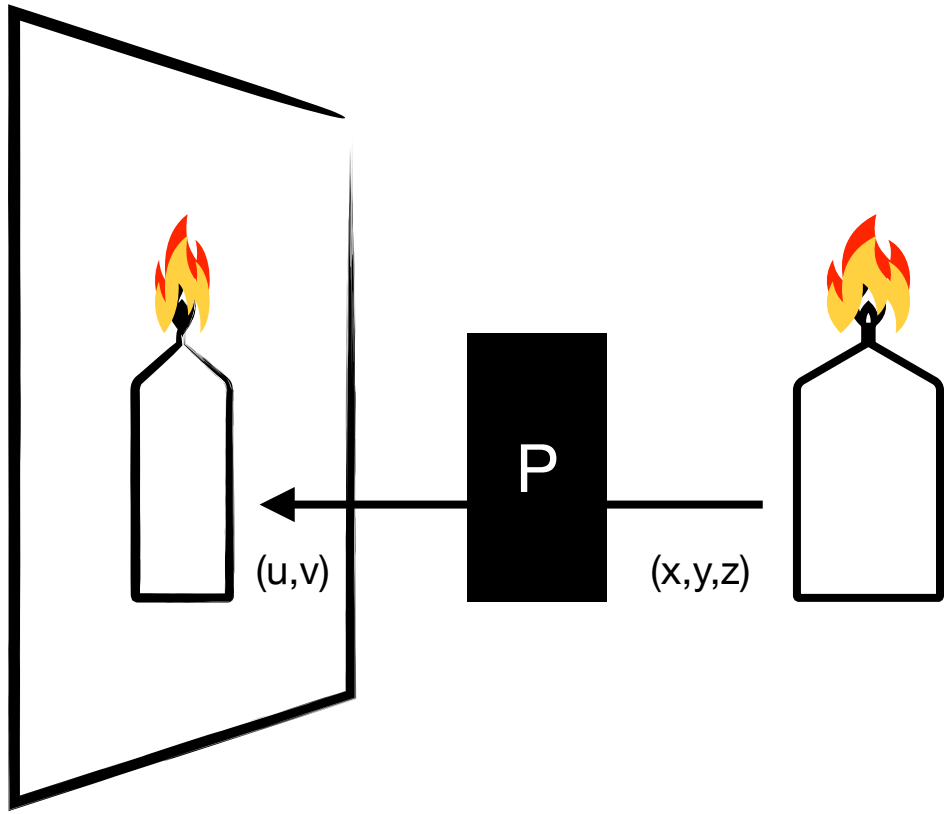
Flip needed

Cameras and Projections

Cameras and Projections

- Cameras project the 3D world onto a 2D image
  - Input is 3D points: (x, y, z)
  - Output is 2D points: (u, v)

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \mathrm{P} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

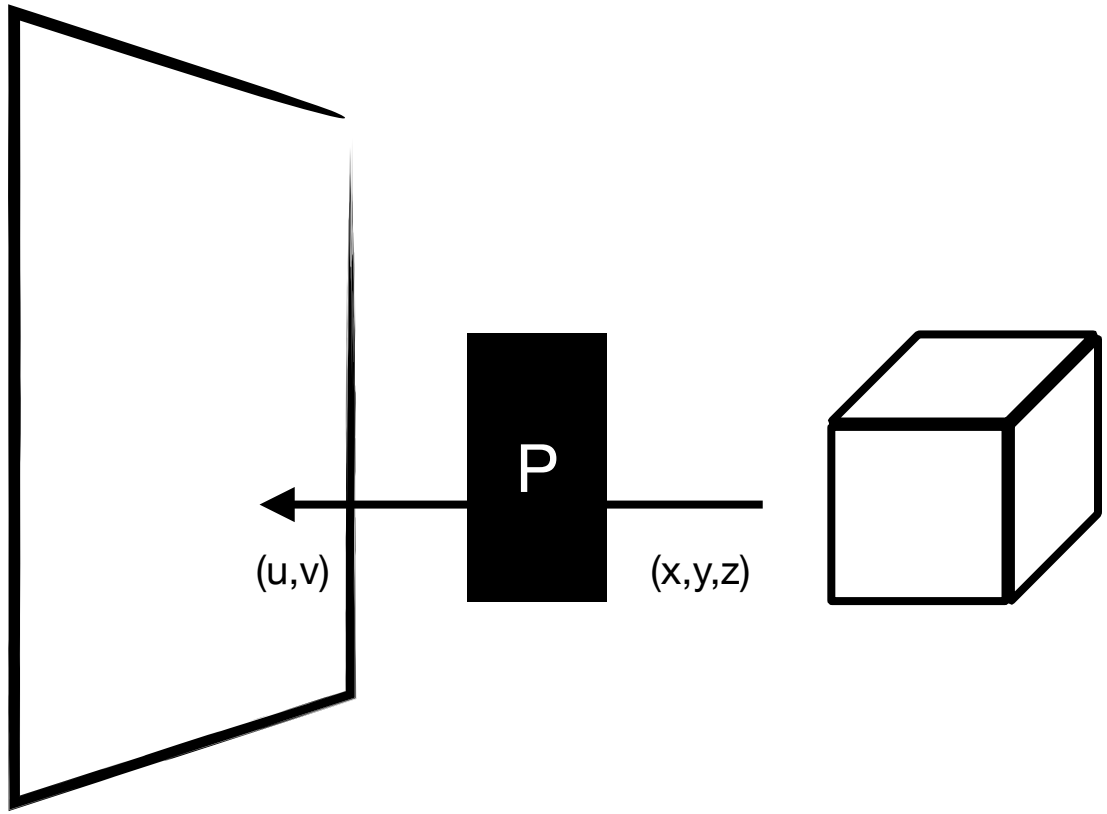


- What form should P have?

Cameras and Projections

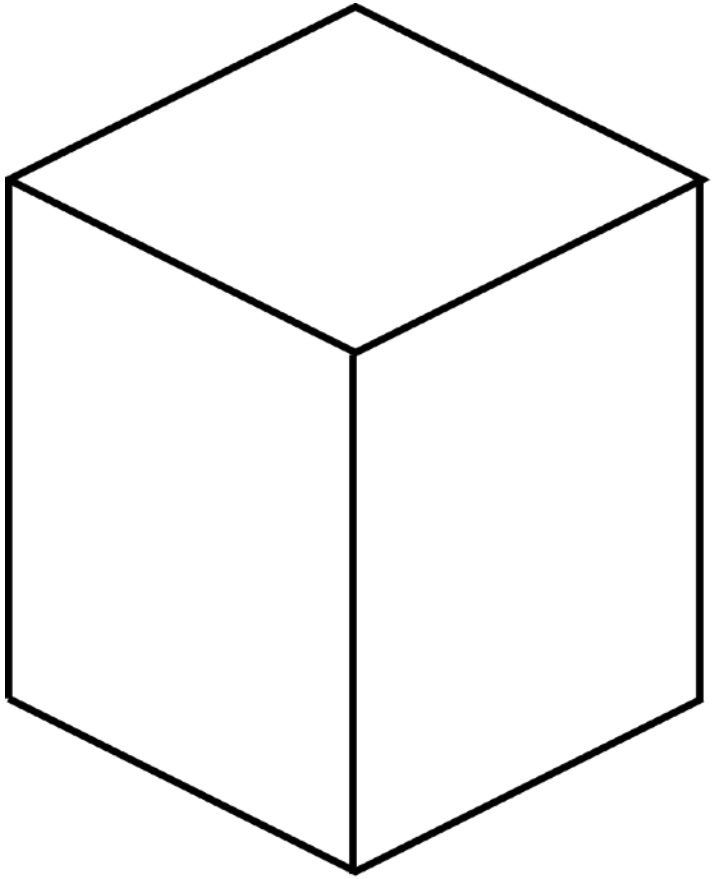
- Cameras project the 3D world onto a 2D image
  - Input is 3D points: (x, y, z)
  - Output is 2D points: (u, v)

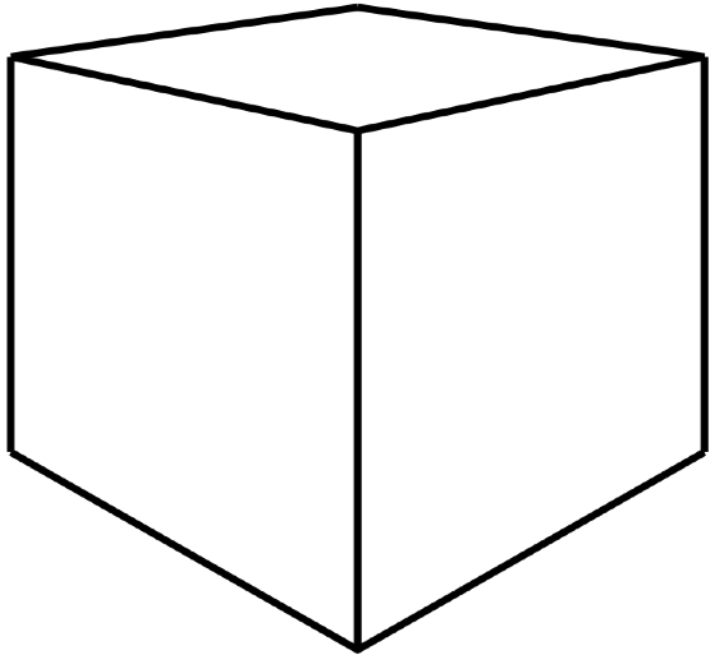
$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \mathrm{P} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

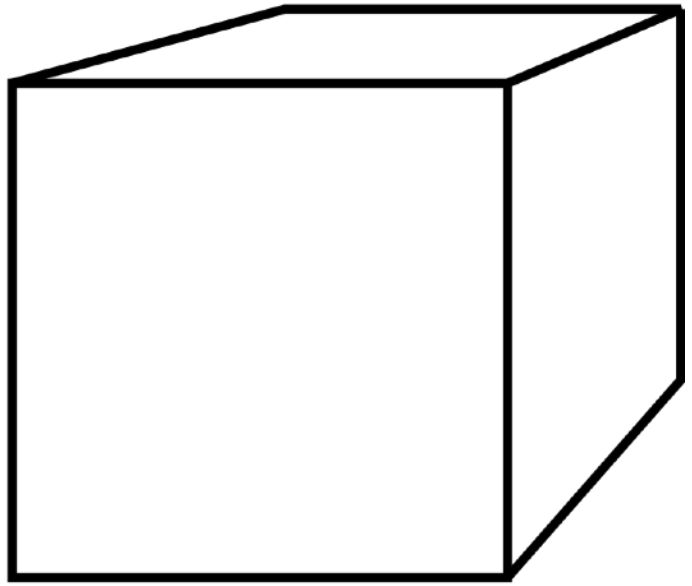


- What form should P have?

Quiz: Which Cube Looks Right?





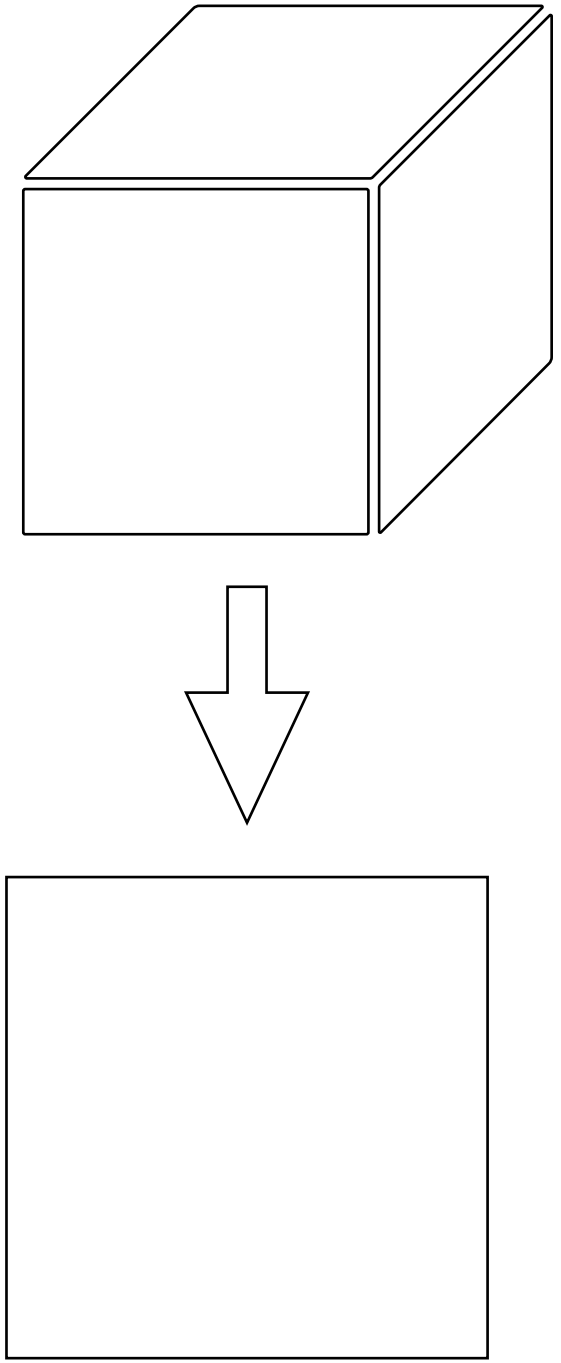


Orthographic Projection

- Simple way to go from 3D to 2D
  - Delete one dimension!
  - Deleting Z projects to the X-Y plane

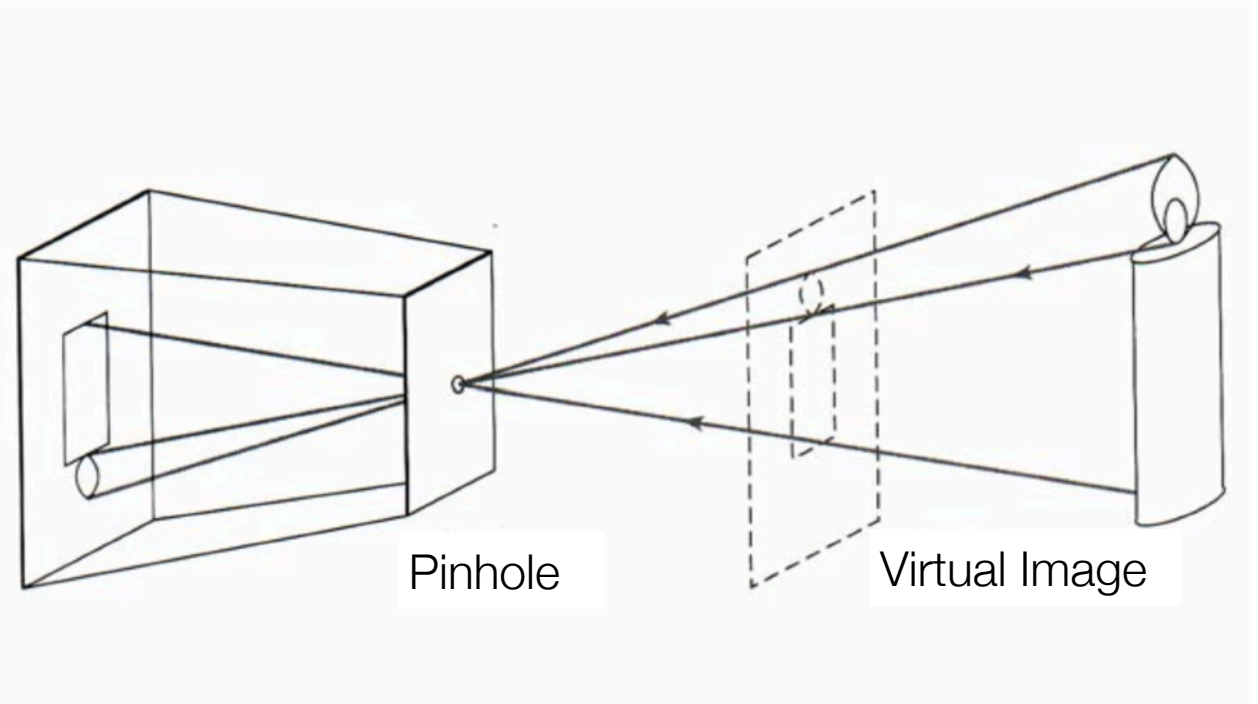
$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

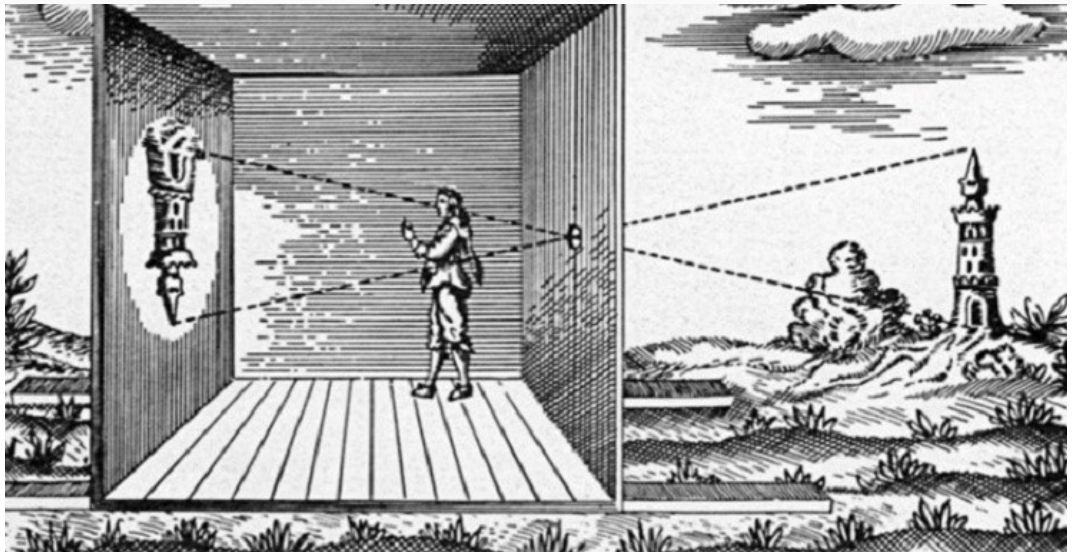
- Parallel lines do not converge (stay parallel)
- Maintains true size (scale)
- This is not how our eyes work



Perspective Projection

- Camera Obscura (dark chamber, 16th century)
  - No lens but a pinhole to focus light onto a wall
  - If opening is reduced to a point, exactly one light ray passes through each point on the image plane
  - In reality, the pinhole has a finite size and points on the image plane collect light from multiple rays

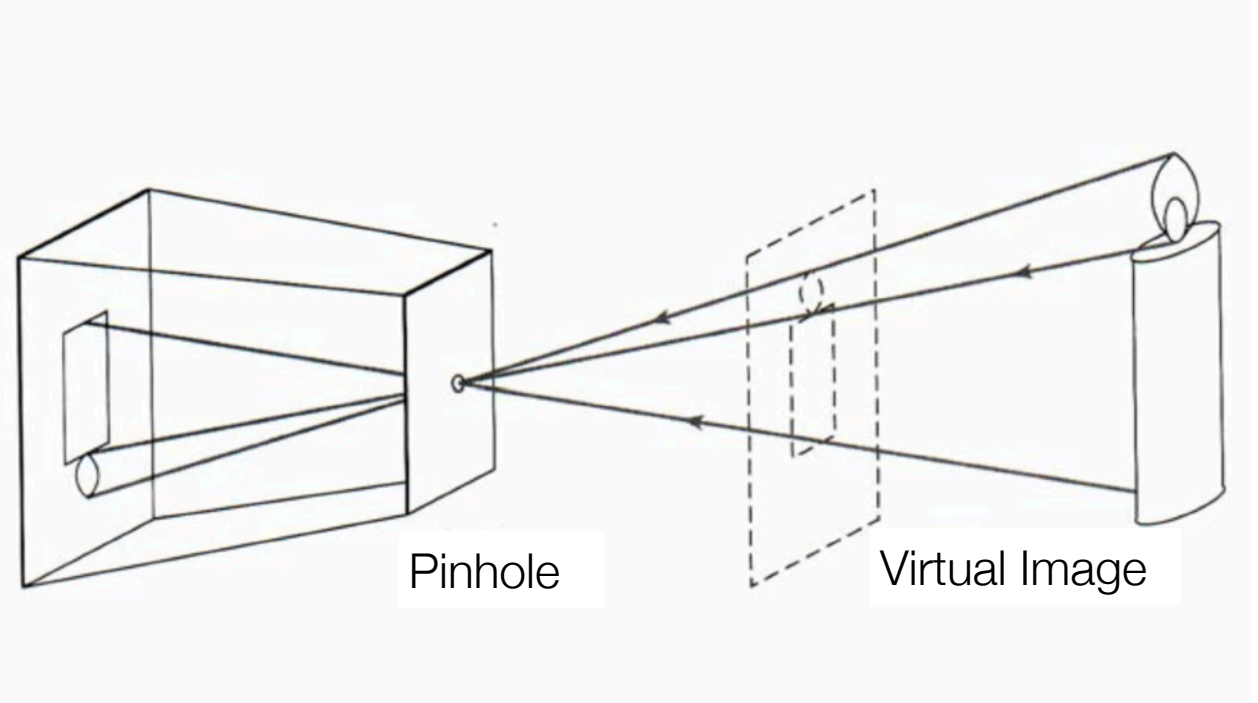


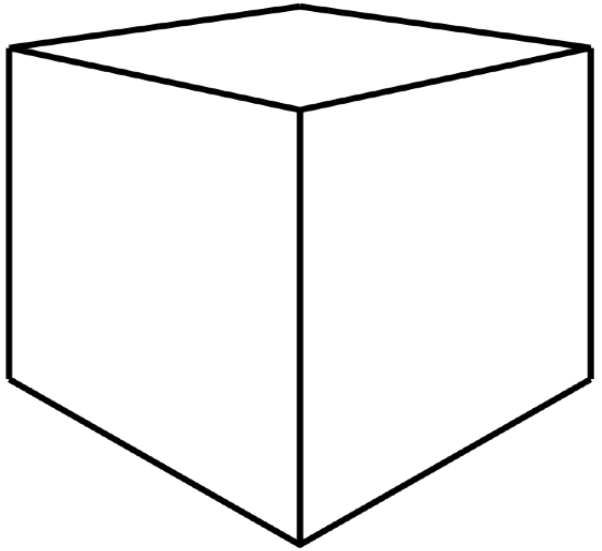


Example: Camera Obscura

Perspective Projection

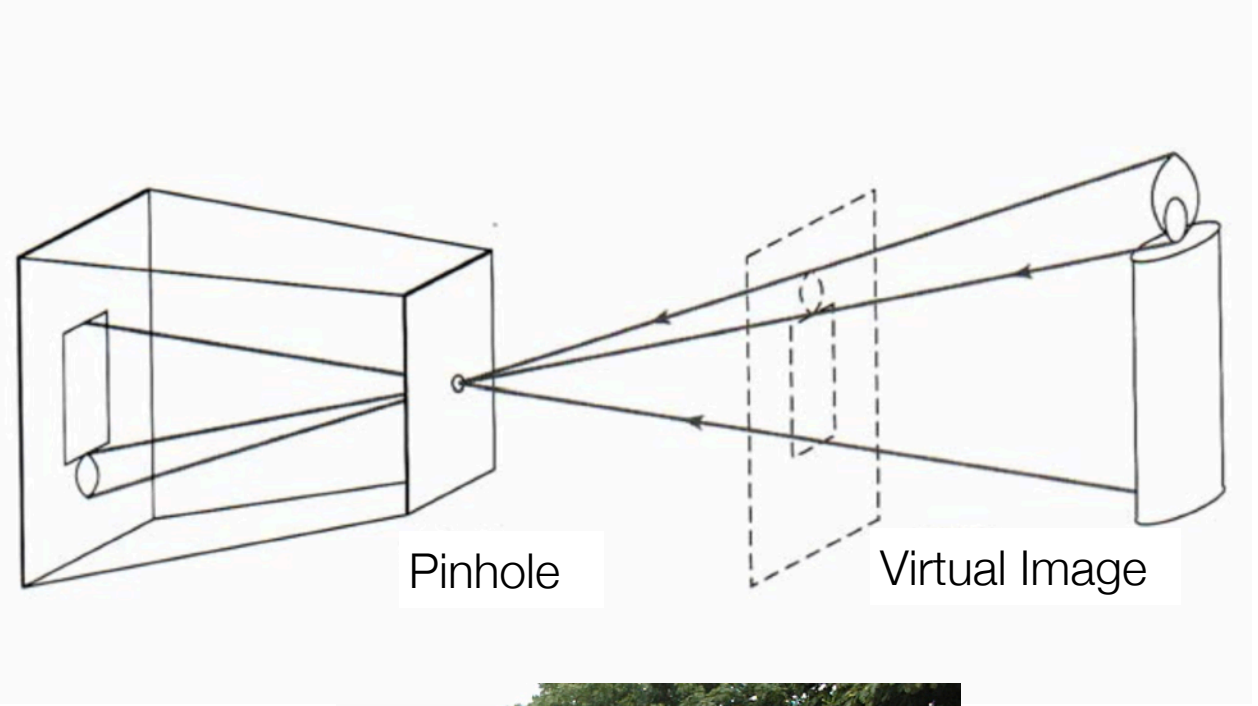
- Perspective projection mimics the human eye
  - Showing depth, making objects appear smaller as they recede
  - Parallel lines converge at vanishing points
- Pinhole camera
  - A simple, but useful, model for perspective projection
  - Central point of projection (the pinhole, often a lens in reality)
  - Light travels from the world, through the pinhole, to the image plane

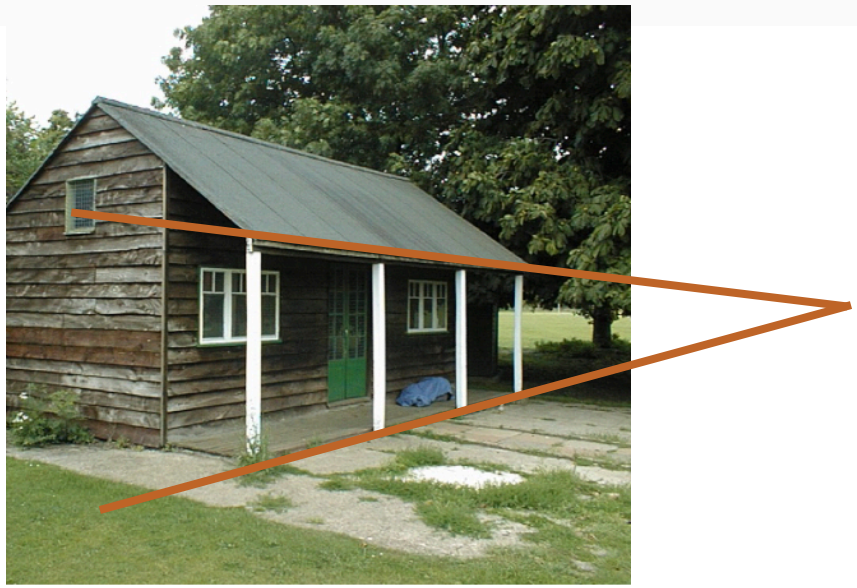




Perspective Projection

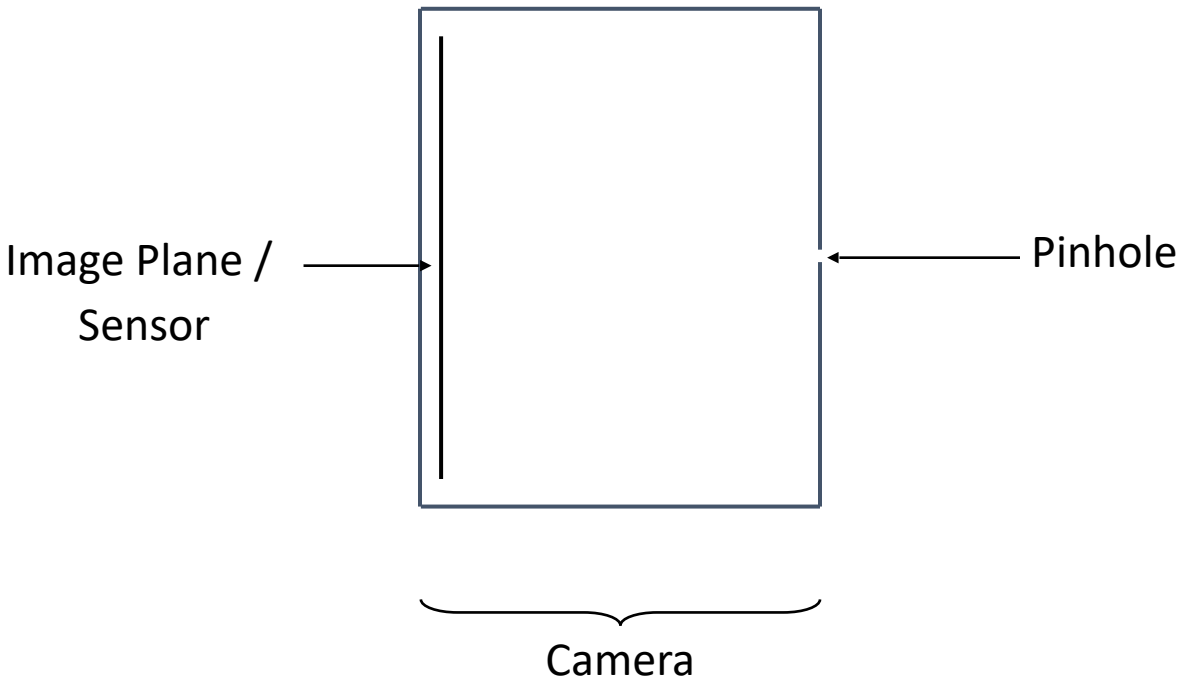
- Perspective projection mimics the human eye
  - Showing depth, making objects appear smaller as they recede
  - Parallel lines converge at vanishing points
- Pinhole camera
  - A simple, but useful, model for perspective projection
  - Central point of projection (the pinhole, often a lens in reality)
  - Light travels from the world, through the pinhole, to the image plane

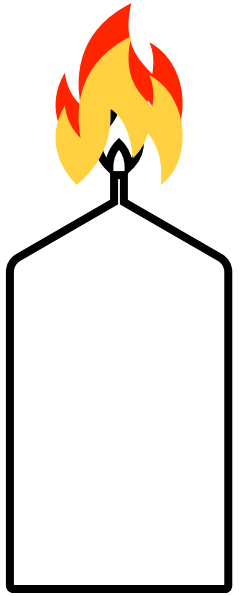




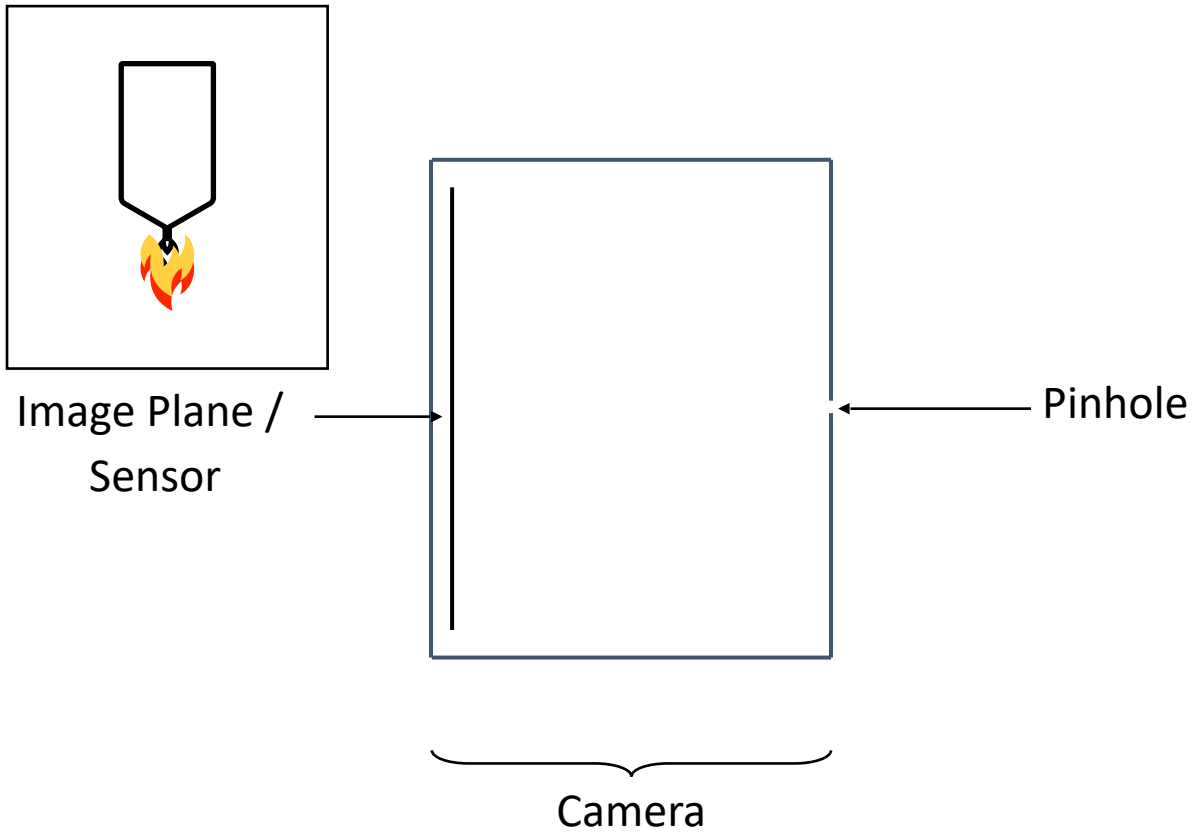
A. Criminisi et al. Single View Metrology. IJCV 2000

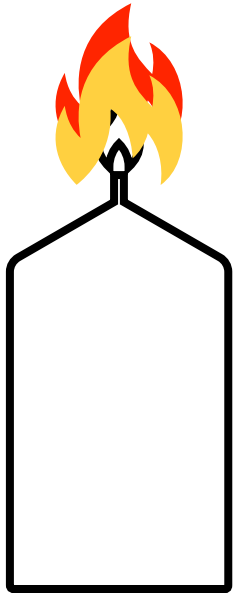
The Pinhole Camera Model



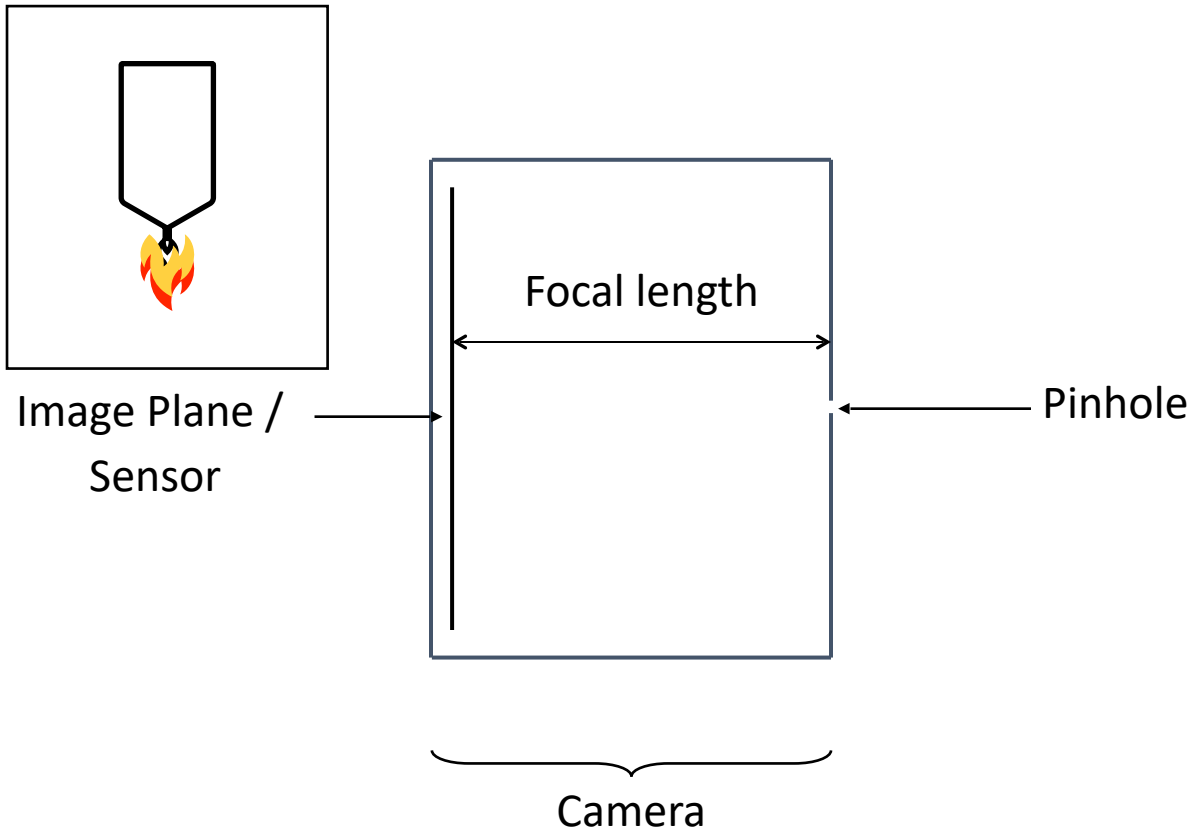


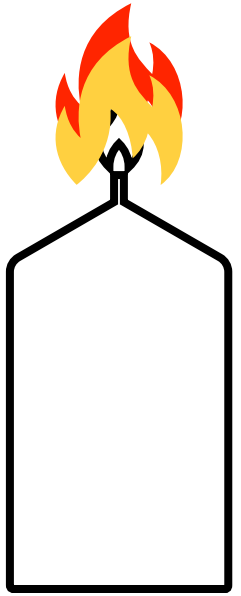
The Pinhole Camera Model



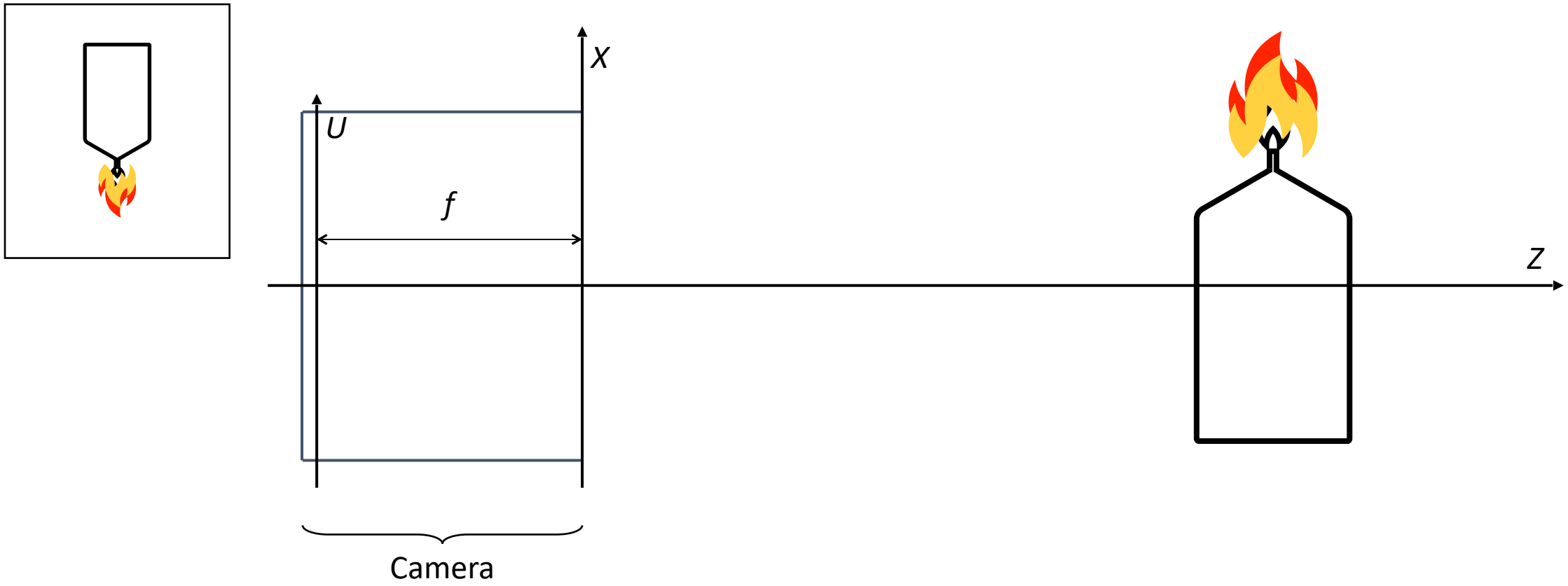


The Pinhole Camera Model

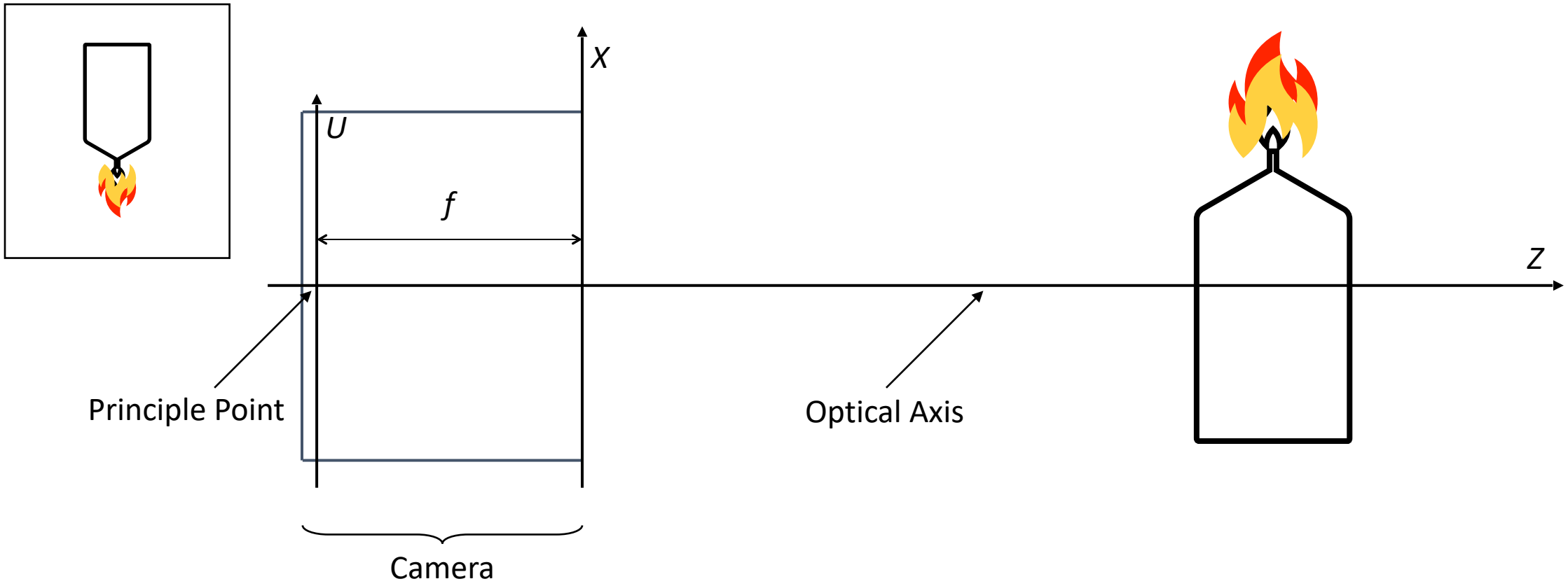




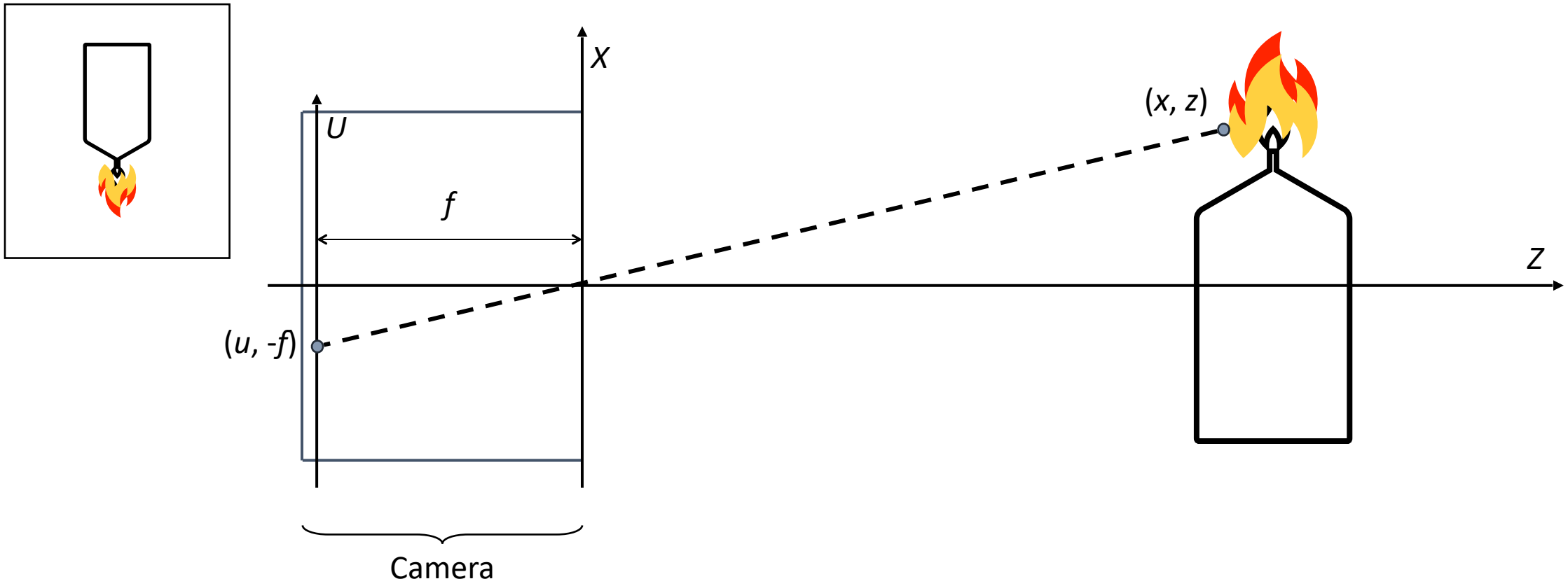
The Pinhole Camera Model



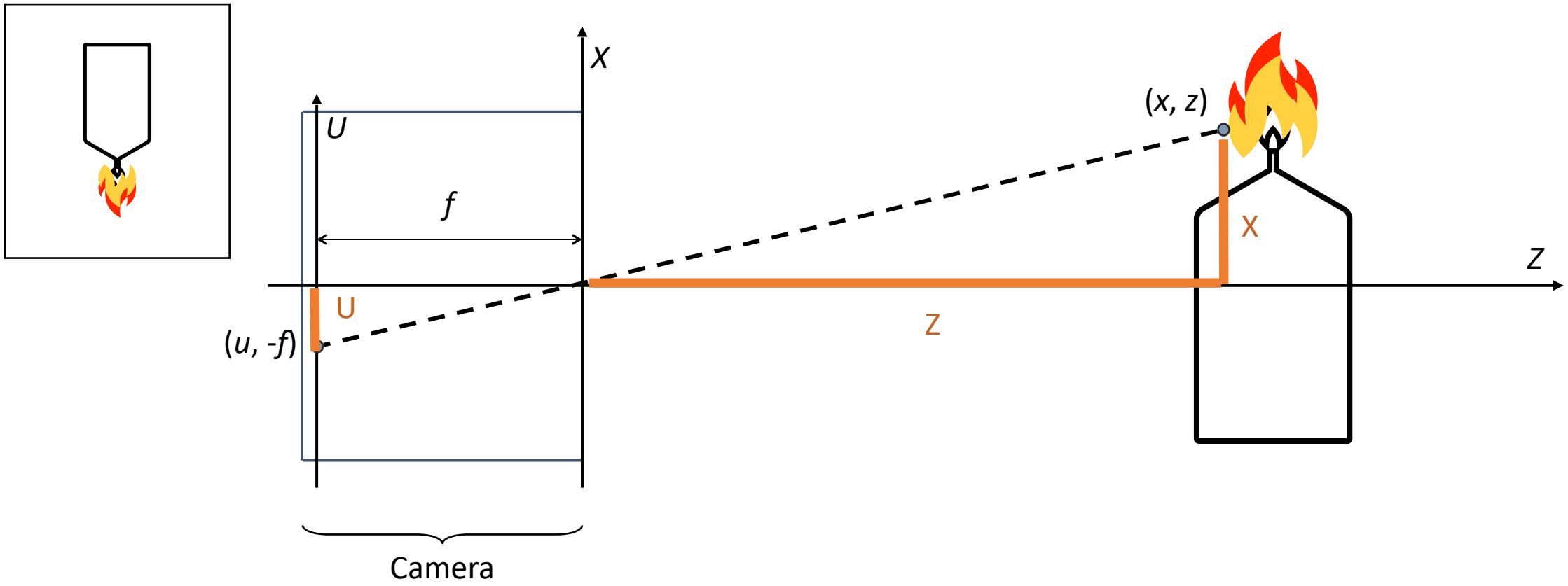
The Pinhole Camera Model



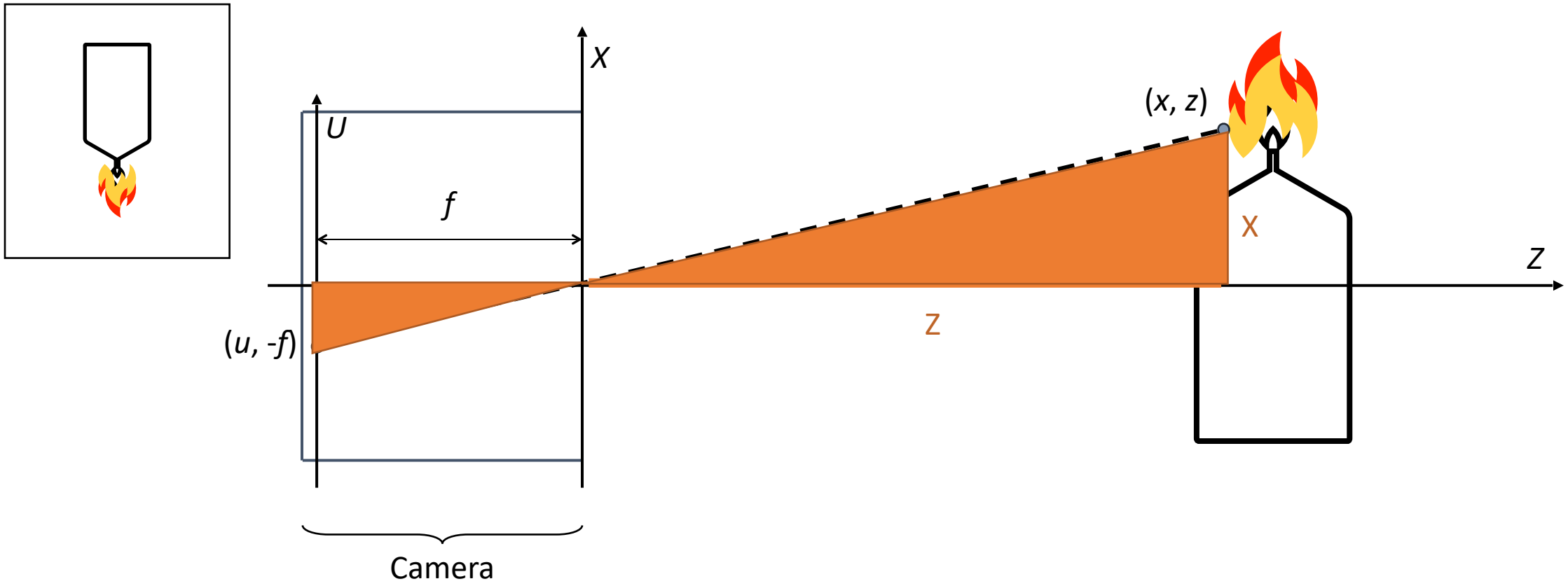
The Pinhole Camera Model



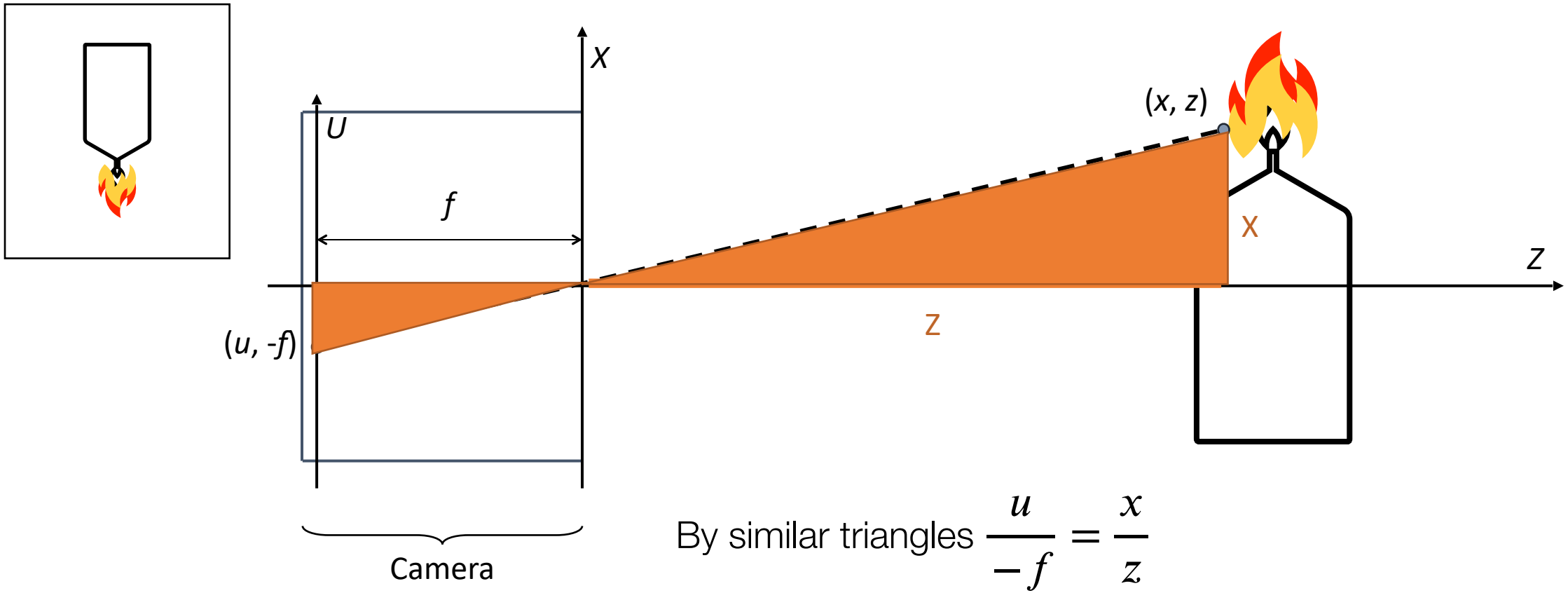
The Pinhole Camera Model



The Pinhole Camera Model



The Pinhole Camera Model

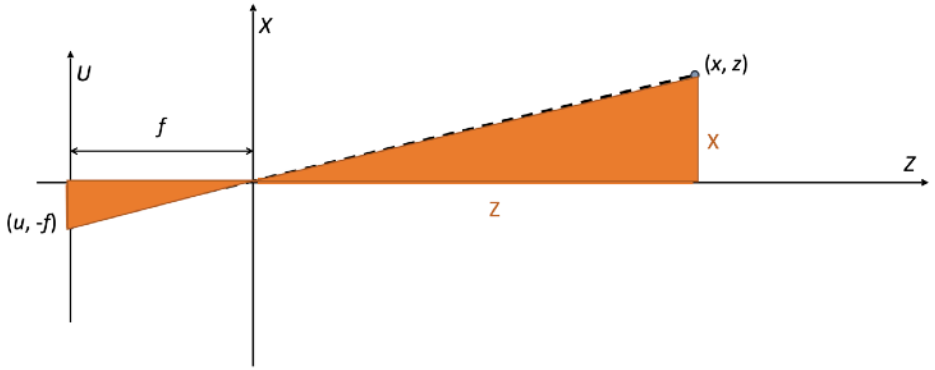


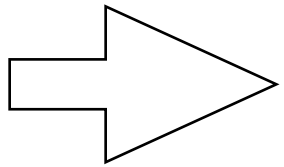
By similar triangles $\frac{u}{-f} = \frac{x}{z}$

The Pinhole Camera Model

- By similar triangles $\frac{u}{-f} = \frac{x}{z}$
  - Similarly for y and v, so we have

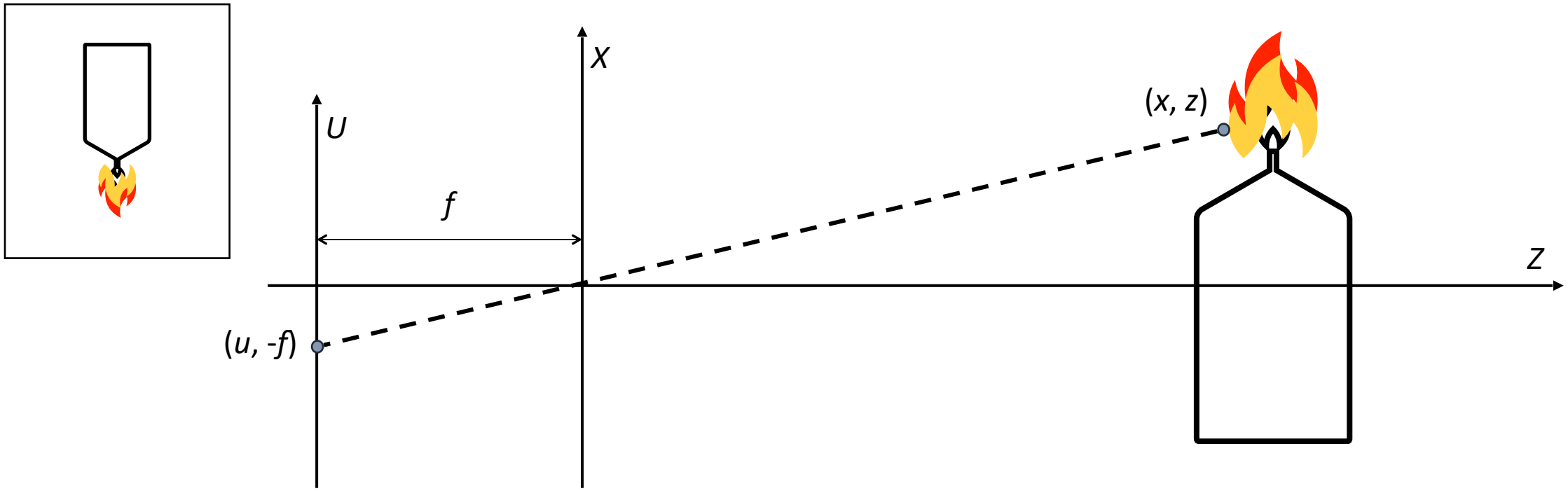
$$u = \frac{-fx}{z} \qquad v = \frac{-fy}{z}$$





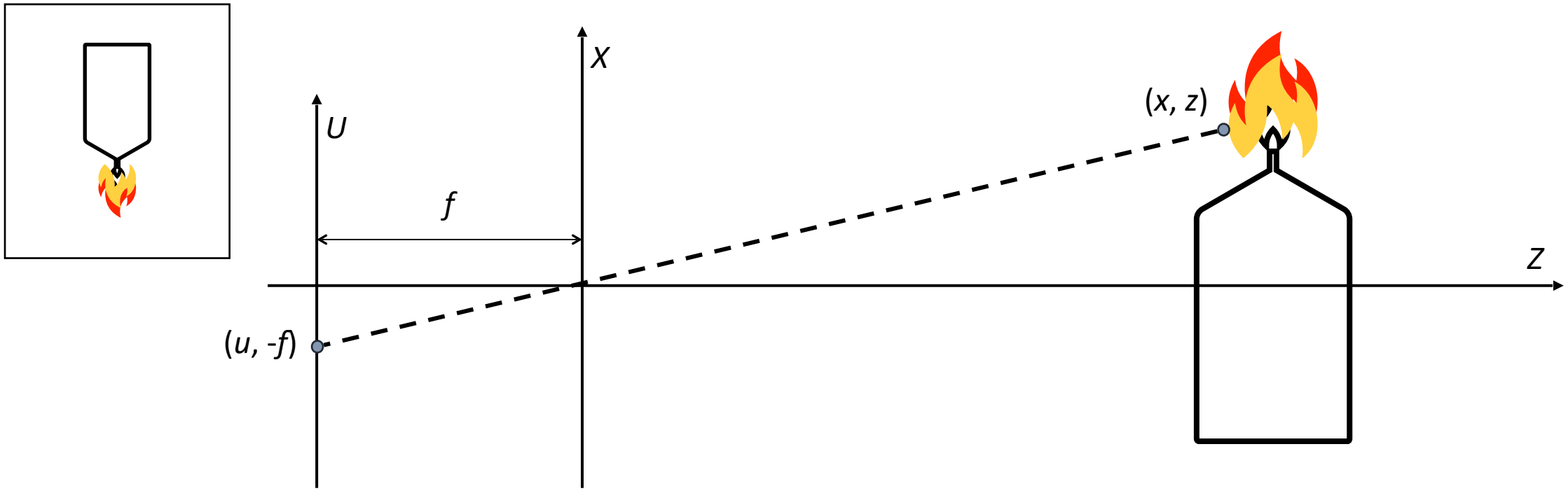
$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \begin{bmatrix} -f & 0 & 0 & 0 \\ 0 & -f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

The Pinhole Camera Model and Projective Geometry



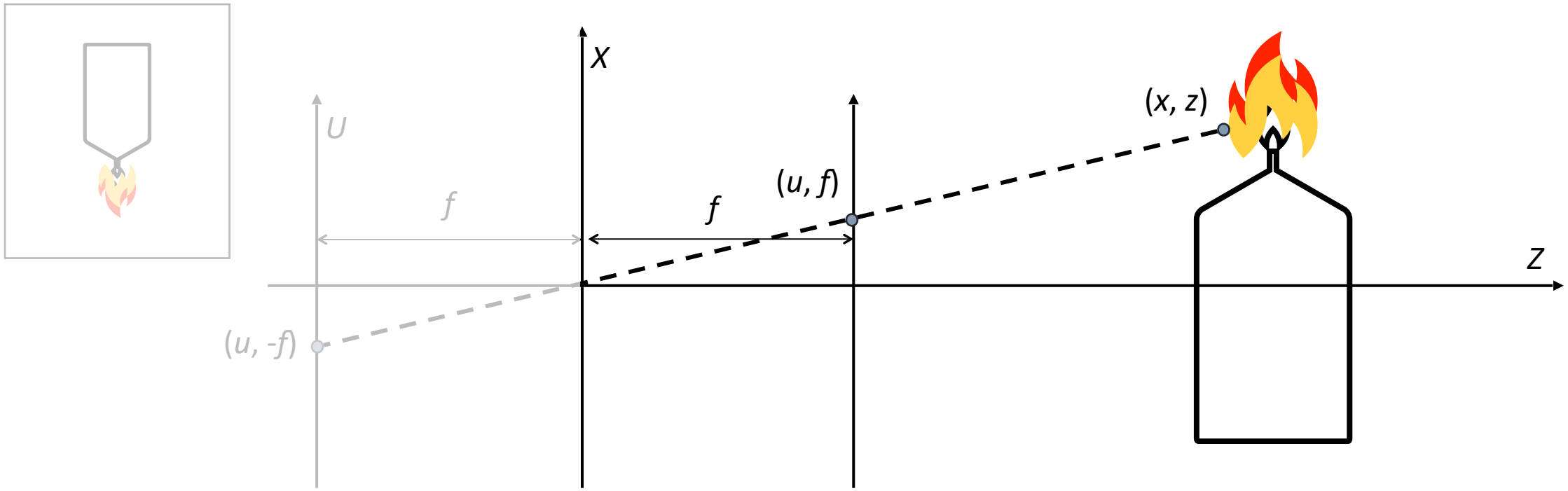
The Pinhole Camera Model and Projective Geometry

- Removing the sign change



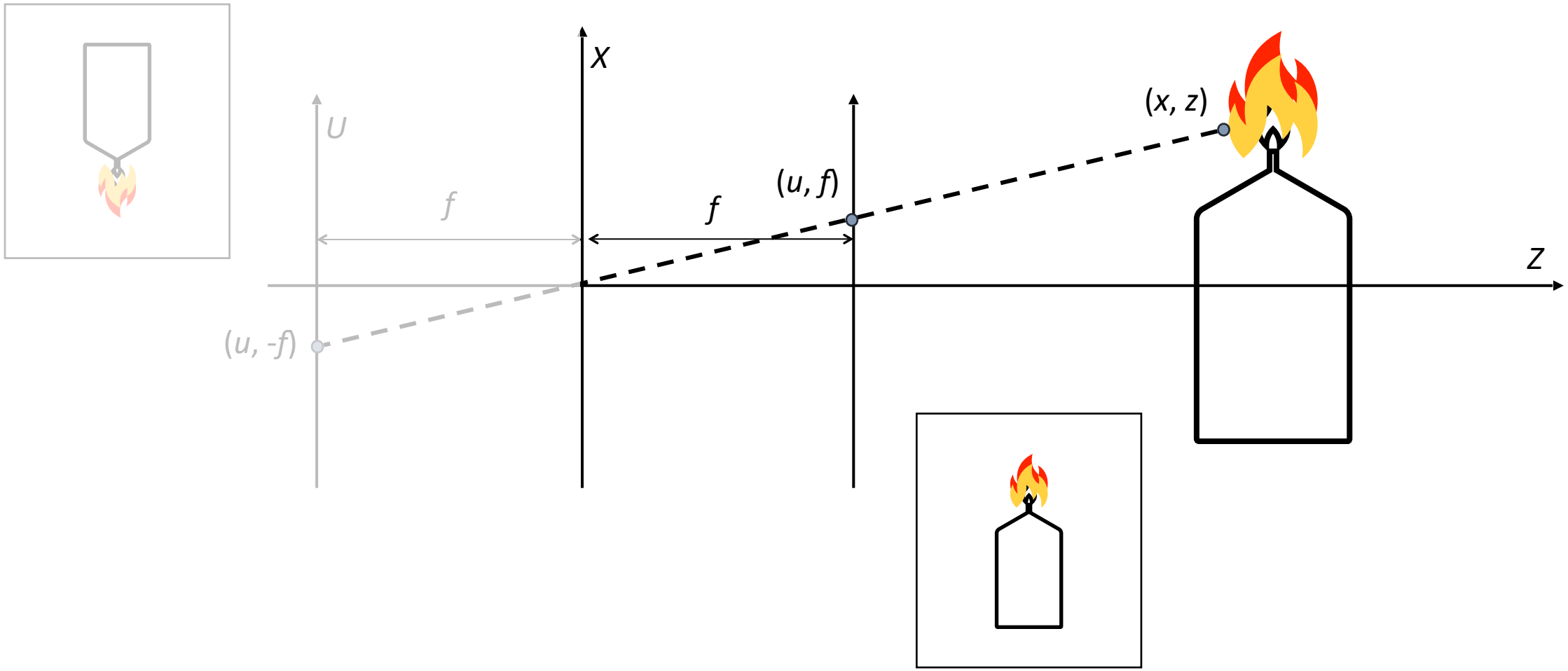
The Pinhole Camera Model and Projective Geometry

- Removing the sign change



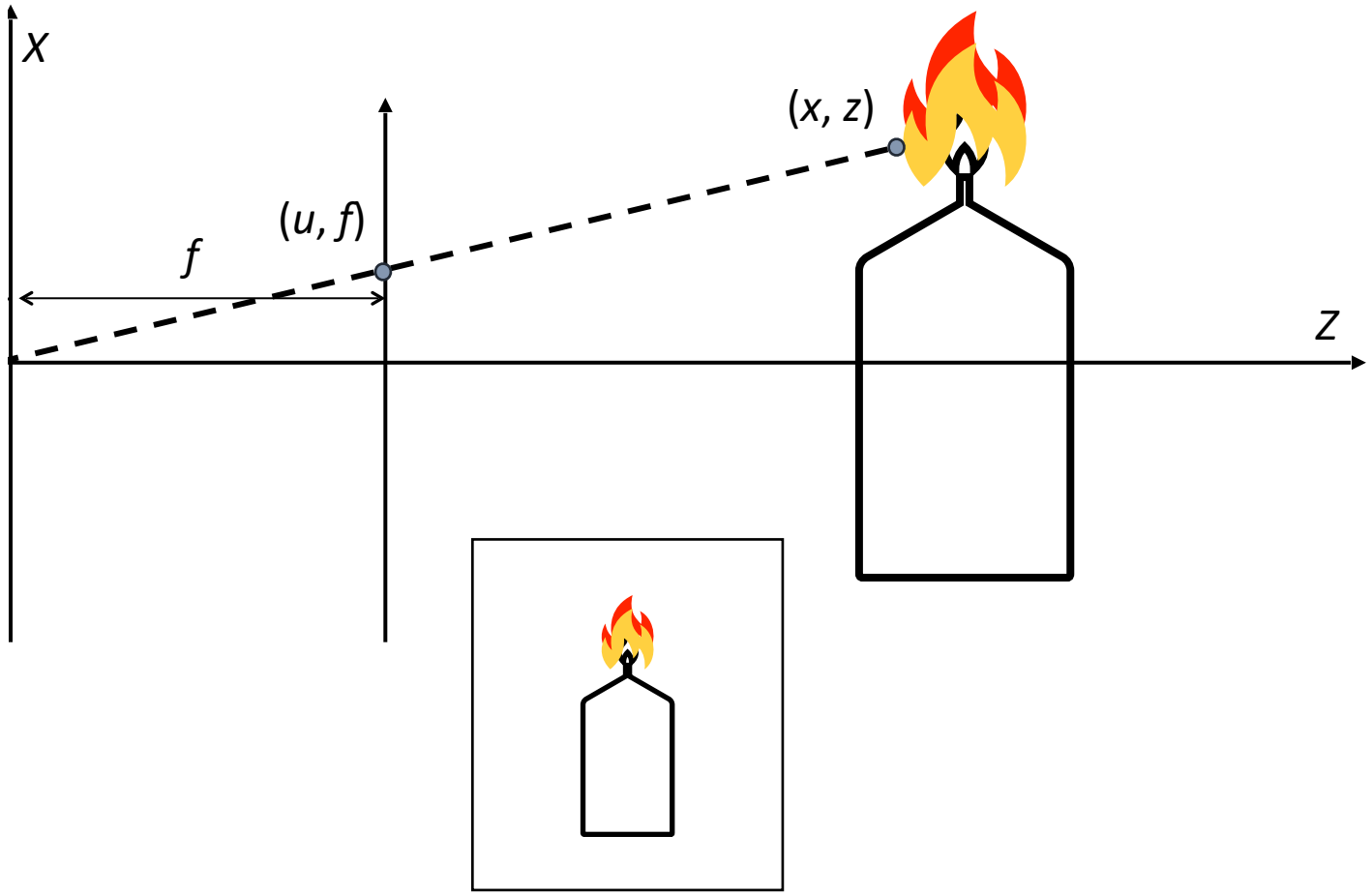
The Pinhole Camera Model and Projective Geometry

- Removing the sign change



The Pinhole Camera Model and Projective Geometry

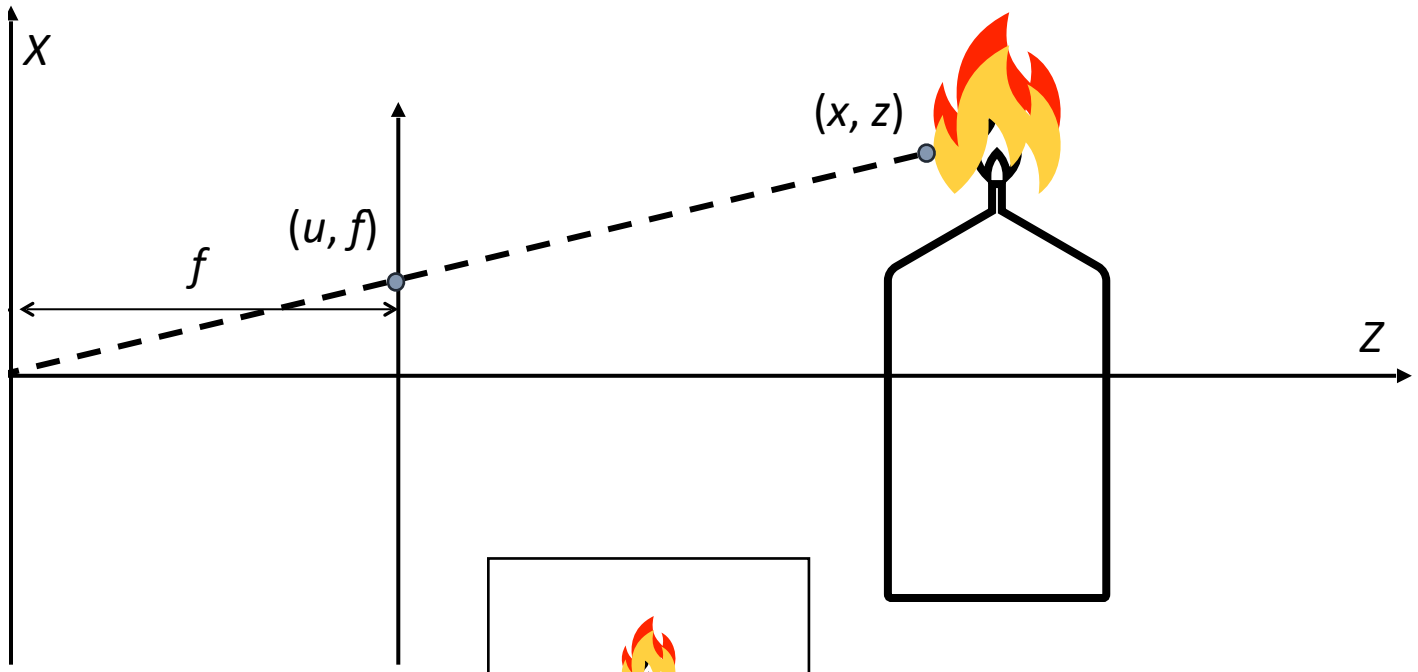
- Removing the sign change

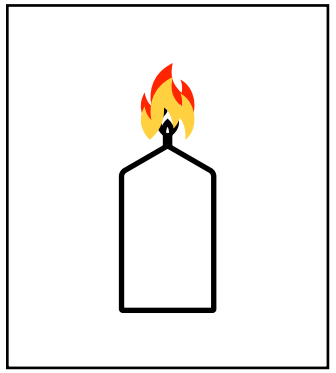


The Pinhole Camera Model and Projective Geometry

- Removing the sign change

- We can put the image plane in front of the pinhole
  - Removes the sign change
  - Not practical for real cameras
  - The maths works out just fine





$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Tour into the picture

https://stefanie.cspages.otago.ac.nz/tip/

Next time:
Camera Calibration and Stereo

The end!