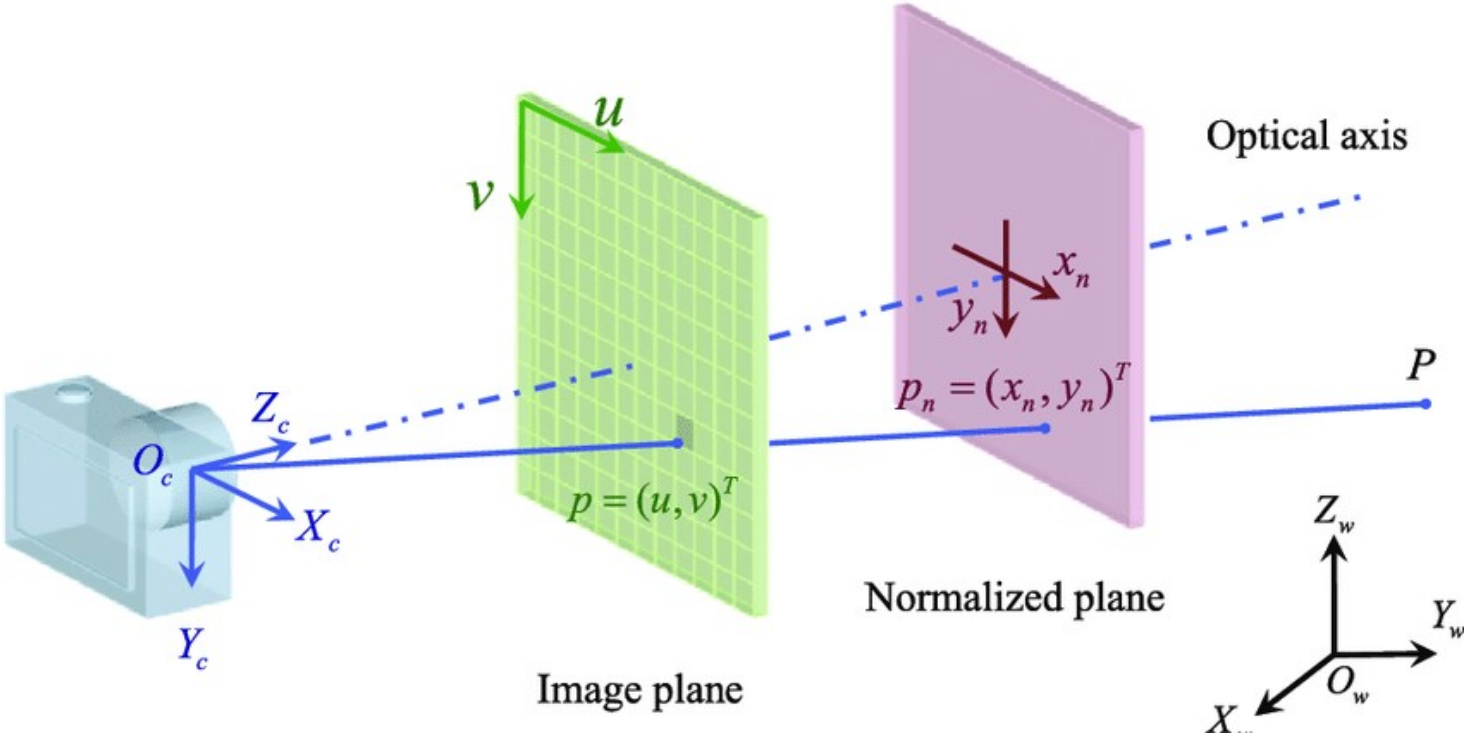
三维视觉中的相机模型

Camera Basics for 3D Vision

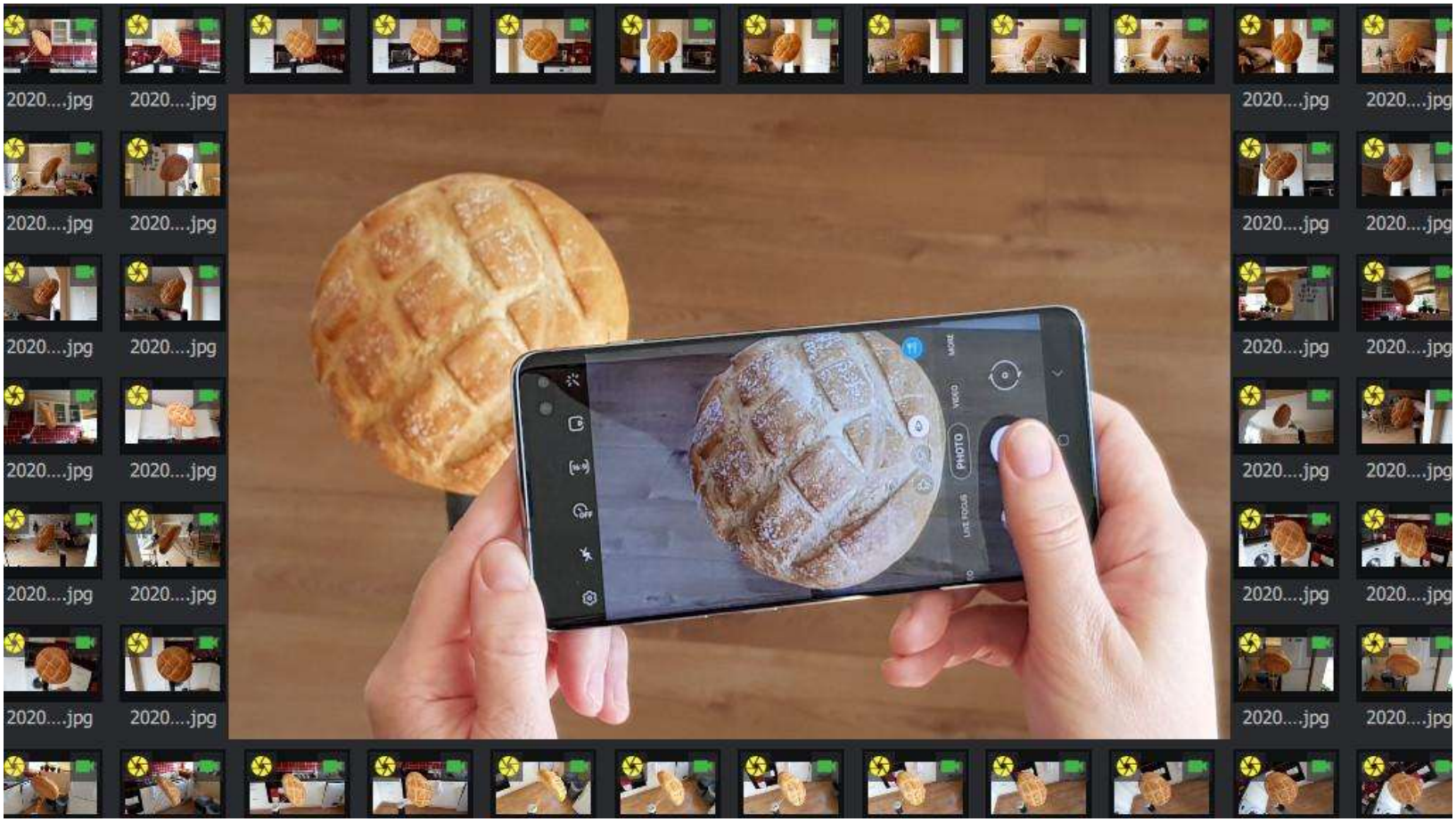


3D Vision: 3D & 2D Conversion

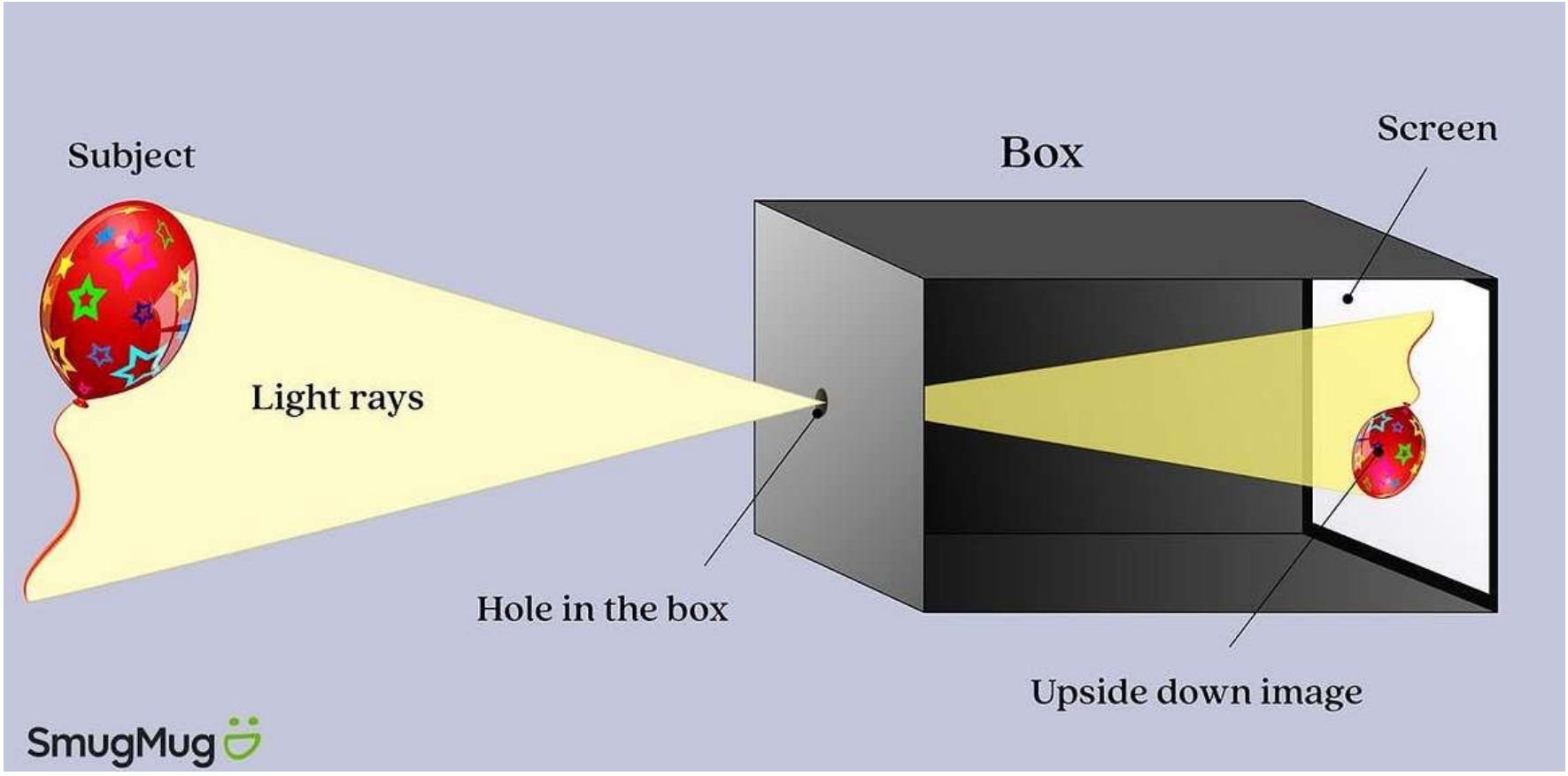
Rendering

Reconstruction

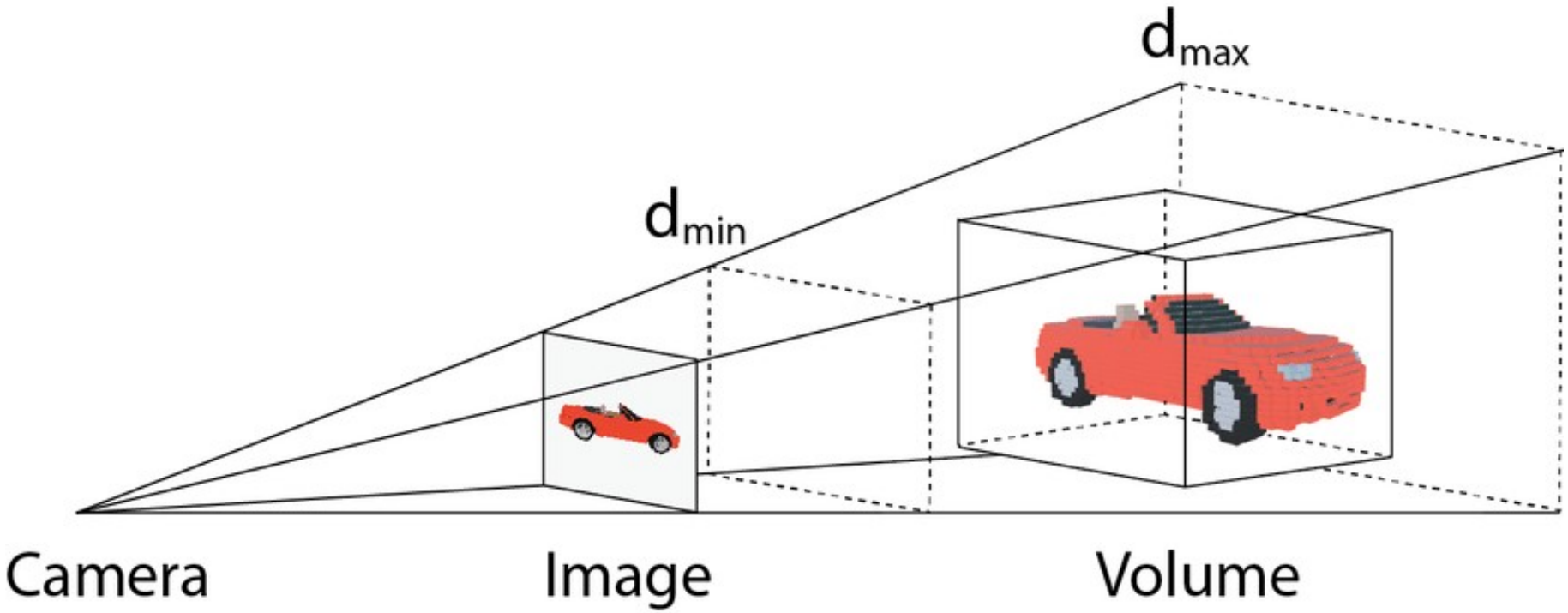
Camera: Bridge between 3D & 2D



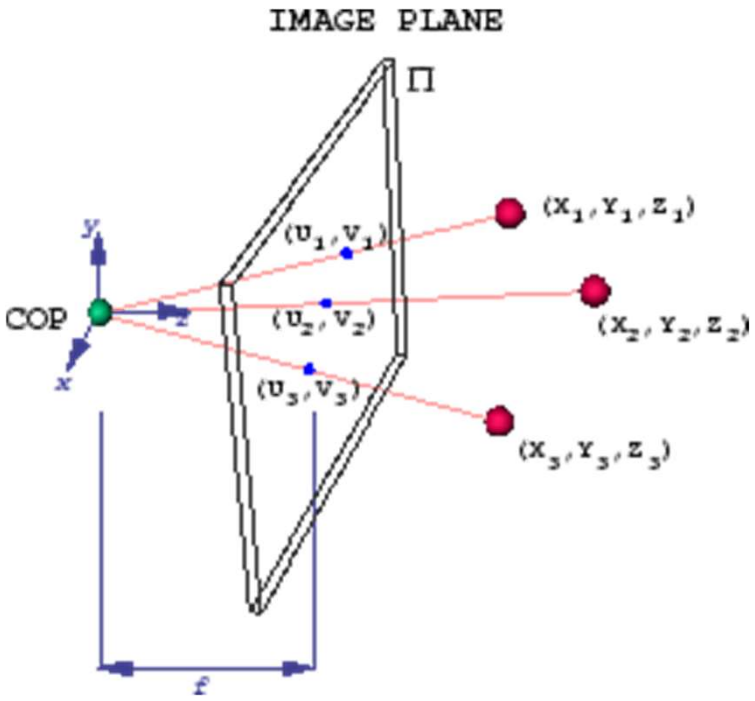
Pinhole Camera

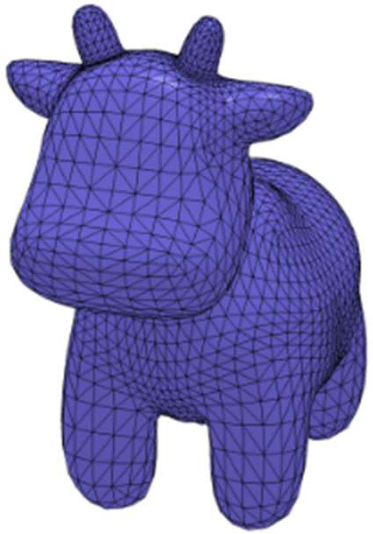


Digital Camera Now: Perspective Camera

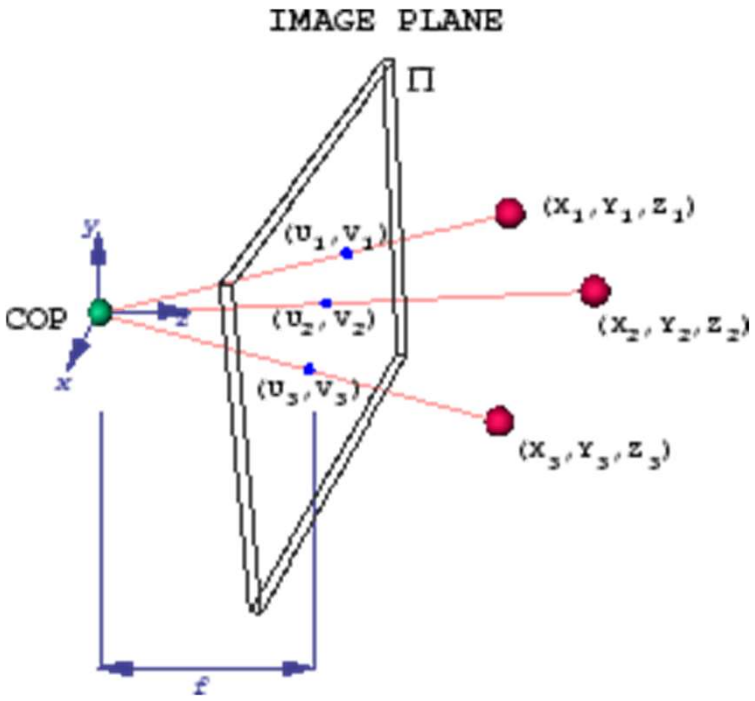


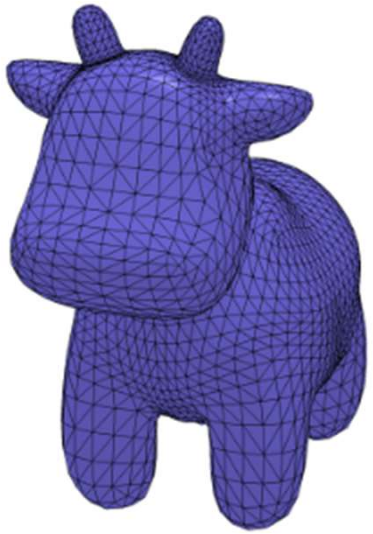
Bridge between 3D and 2D





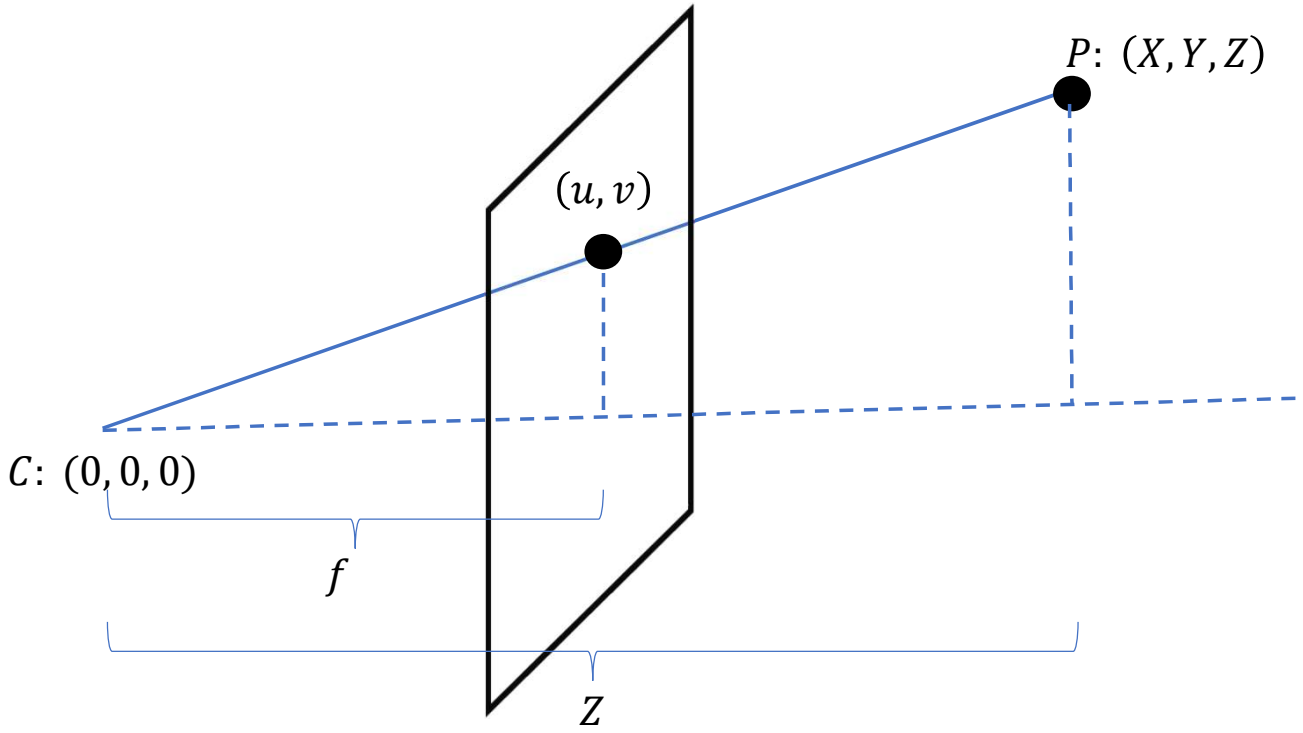
Bridge between 3D and 2D

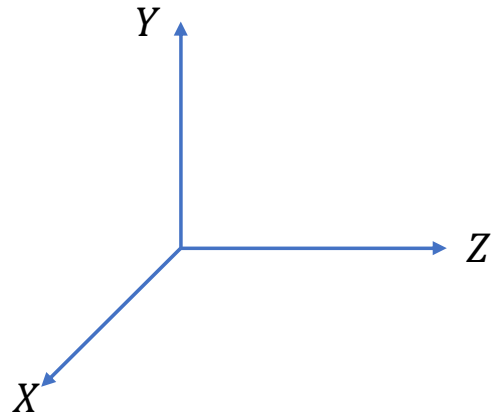




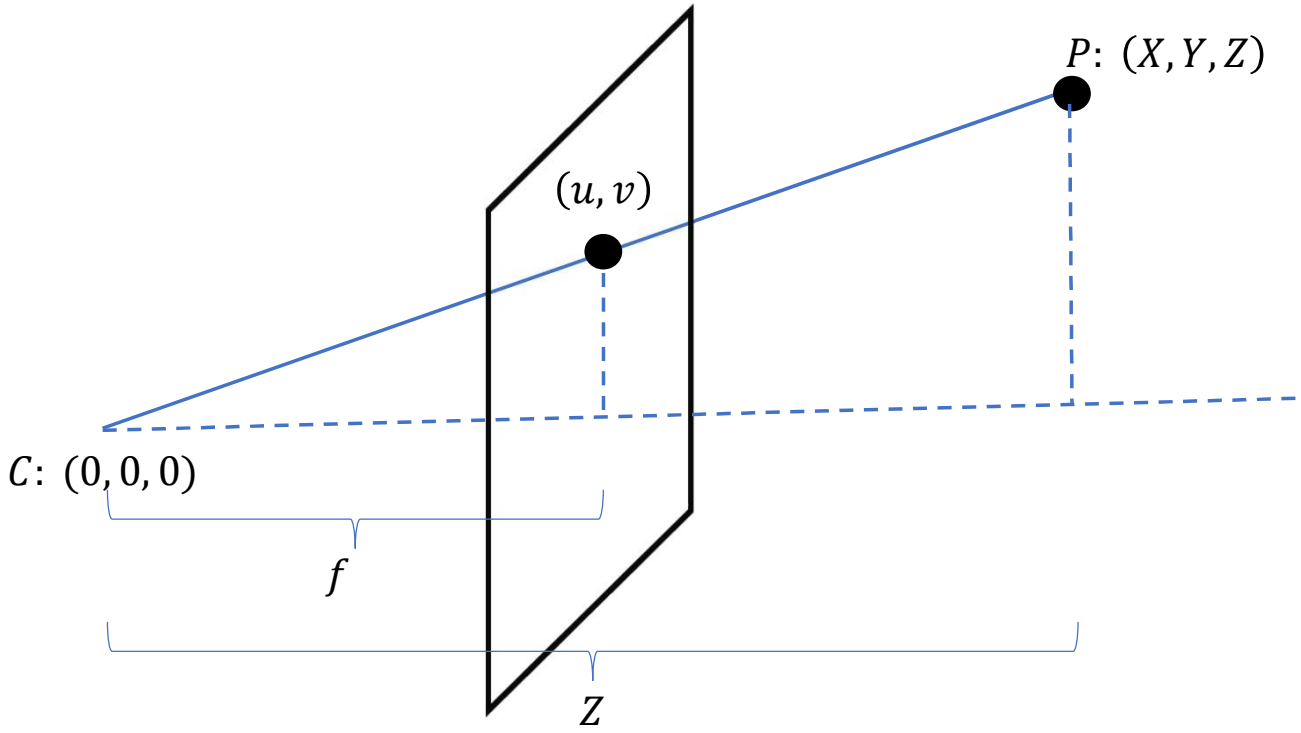
$$\Pi\colon R^3 \to R^2$$
$$(X, Y, Z) \to (u, v)$$

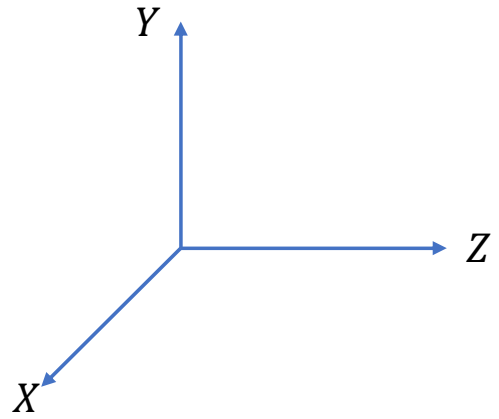
Bridge between 3D and 2D





Bridge between 3D and 2D



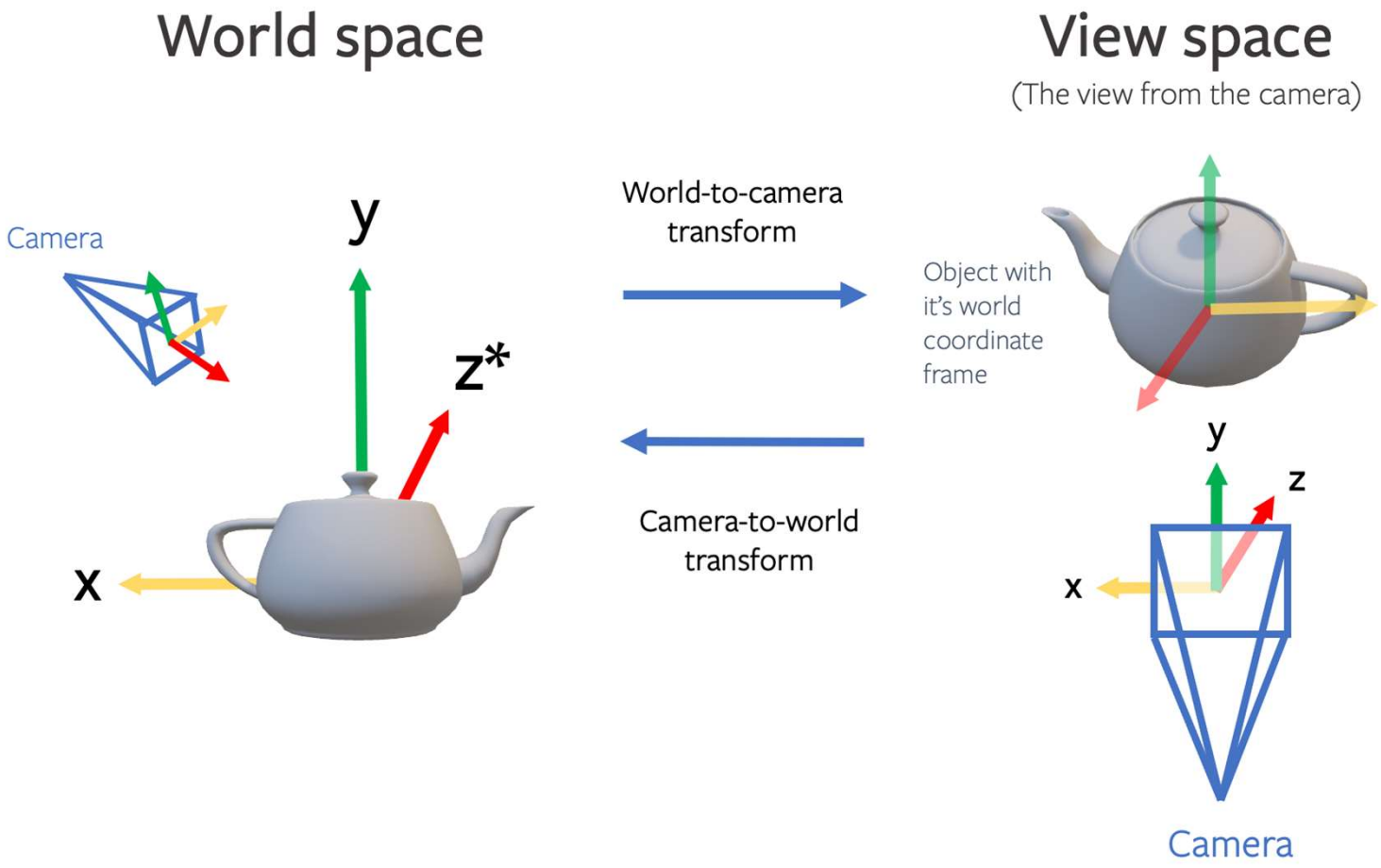


$$(X, Y, Z) \rightarrow (u, v)$$

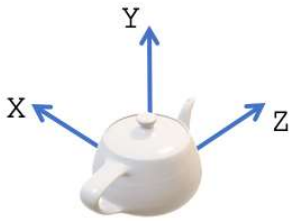
$$u = \frac{f}{Z} * X$$

$$v = \frac{f}{Z} * Y$$

Coordinate Systems (in 3D)

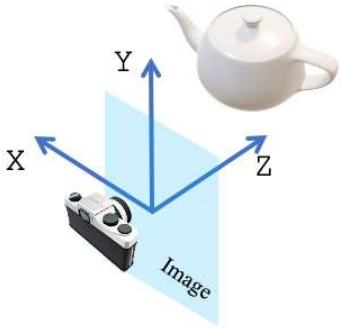


Coordinate Systems: Whole Pipeline

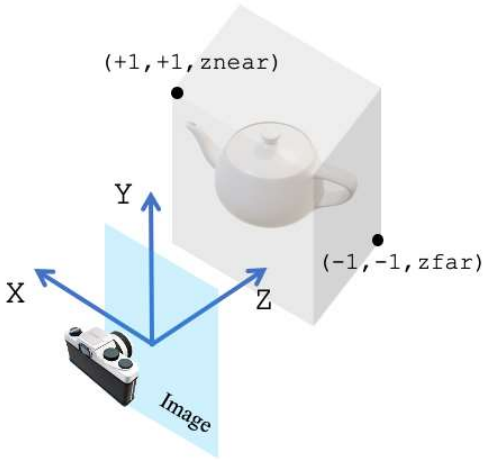


World
Coordinate System

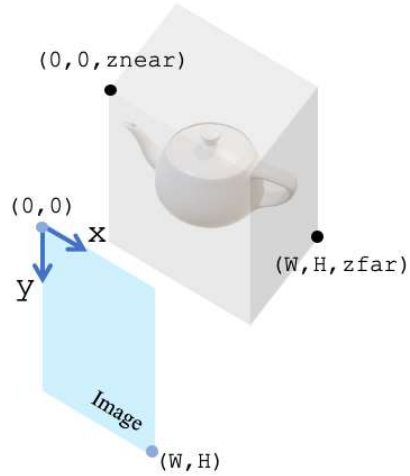
R, T



Camera View
Coordinate System

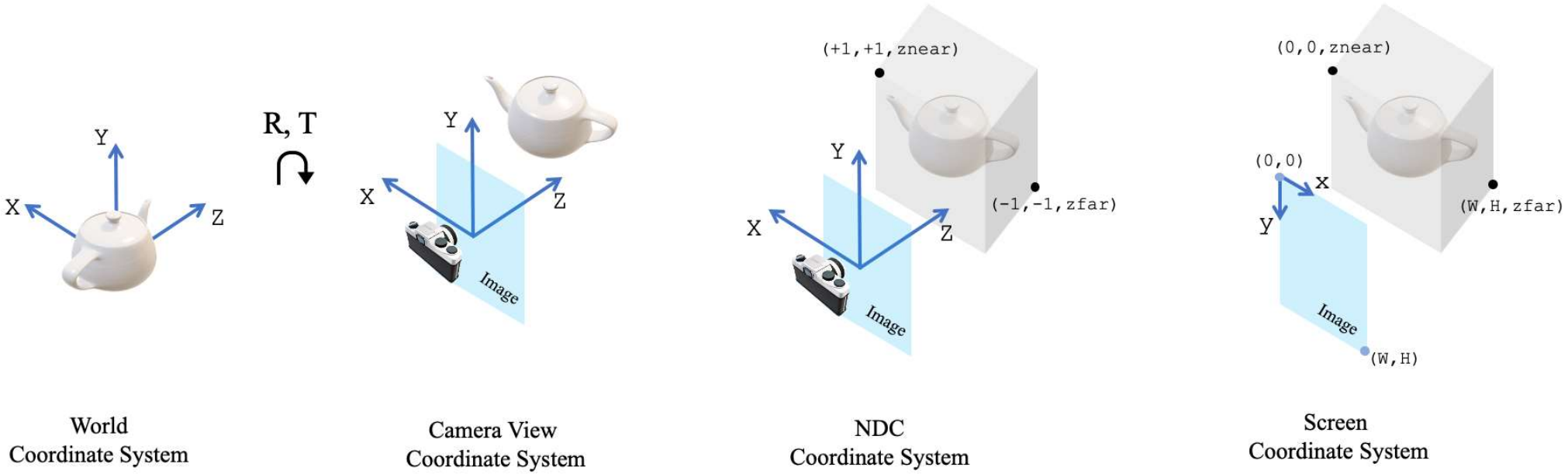


NDC
Coordinate System



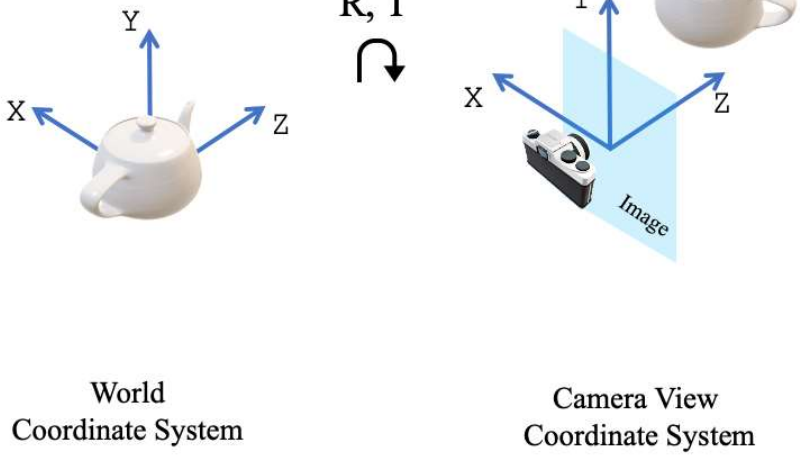
Screen
Coordinate System

Coordinate Systems: Whole Pipeline



How we transform each coordinate space to the next?

World Space to Camera (View) Space



World
Coordinate System

Camera View
Coordinate System

NDC
Coordinate System

Screen
Coordinate System

Transformation between two orthogonal system

if I have a Orthogonal coordinate system with normalized vector x,y,z and origin o1.
and another orthogonal coordinate system with normalized vector i,j,k and origin o2
please fomulate the transformation between the two space

- **Coordinate System 1:**
  - Basis vectors: $\mathbf{x}, \mathbf{y}, \mathbf{z}$
  - Origin: $\mathbf{o}_1$
  - Point in this system: $\mathbf{p}_1$

- **Coordinate System 2:**
  - Basis vectors: $\mathbf{i}, \mathbf{j}, \mathbf{k}$
  - Origin: $\mathbf{o}_2$
  - Point in this system: $\mathbf{p}_2$

- **Coordinate System S_1:**
  - Orthonormal basis vectors: $\mathbf{x}, \mathbf{y}, \mathbf{z}$.
  - Origin: $\mathbf{o}_1$.
  - Coordinates of point P: (α, β, γ).

- **Coordinate System S_2:**
  - Orthonormal basis vectors: $\mathbf{i}, \mathbf{j}, \mathbf{k}$.
  - Origin: $\mathbf{o}_2$.
  - Coordinates of point P: $(\alpha', \beta', \gamma')$.

- **In S_1:**

$$\mathbf{P} = \mathbf{o}_1 + \alpha\mathbf{x} + \beta\mathbf{y} + \gamma\mathbf{z}$$

- **In S_2:**

$$\mathbf{P} = \mathbf{o}_2 + \alpha'\mathbf{i} + \beta'\mathbf{j} + \gamma'\mathbf{k}$$

2. Equate the Two Expressions of $\mathbf{P}$

Since both expressions represent the same point in space, we can set them equal to each other:

$$\mathbf{o}_1 + \alpha\mathbf{x} + \beta\mathbf{y} + \gamma\mathbf{z} = \mathbf{o}_2 + \alpha'\mathbf{i} + \beta'\mathbf{j} + \gamma'\mathbf{k}$$

3. Rearrange the Equation

Bring all terms to one side:

$$\alpha\mathbf{x} + \beta\mathbf{y} + \gamma\mathbf{z} - \alpha'\mathbf{i} - \beta'\mathbf{j} - \gamma'\mathbf{k} = \mathbf{o}_2 - \mathbf{o}_1$$

4. Express $\mathbf{i}, \mathbf{j}, \mathbf{k}$ in Terms of $\mathbf{x}, \mathbf{y}, \mathbf{z}$

Since S_1 and S_2 are related by rotation, the basis vectors of S_2 can be expressed in terms of those of S_1:

$$\begin{aligned}\mathbf{i} &= r_{11}\mathbf{x} + r_{12}\mathbf{y} + r_{13}\mathbf{z} \\ \mathbf{j} &= r_{21}\mathbf{x} + r_{22}\mathbf{y} + r_{23}\mathbf{z} \\ \mathbf{k} &= r_{31}\mathbf{x} + r_{32}\mathbf{y} + r_{33}\mathbf{z}\end{aligned}$$

Here, $\mathbf{R} = [r_{ij}]$ is the rotation matrix from S_1 to S_2.

5. Substitute $\mathbf{i}, \mathbf{j}, \mathbf{k}$ into the Equation

Substitute the expressions of $\mathbf{i}, \mathbf{j}, \mathbf{k}$ into the equation:

$$\begin{aligned}&\alpha\mathbf{x} + \beta\mathbf{y} + \gamma\mathbf{z} - \alpha'(r_{11}\mathbf{x} + r_{12}\mathbf{y} + r_{13}\mathbf{z}) \\ &\quad - \beta'(r_{21}\mathbf{x} + r_{22}\mathbf{y} + r_{23}\mathbf{z}) - \gamma'(r_{31}\mathbf{x} + r_{32}\mathbf{y} + r_{33}\mathbf{z}) = \mathbf{o}_2 - \mathbf{o}_1\end{aligned}$$

6. Collect Like Terms

Group the terms with common basis vectors:

$$[\alpha - \alpha' r_{11} - \beta' r_{21} - \gamma' r_{31}]\,\mathbf{x} + [\beta - \alpha' r_{12} - \beta' r_{22} - \gamma' r_{32}]\,\mathbf{y} + [\gamma - \alpha' r_{13} - \beta' r_{23} - \gamma' r_{33}]\,\mathbf{z} = \mathbf{o}_2 - \mathbf{o}_1$$

7. Write Component Equations

Since $\mathbf{x}, \mathbf{y}, \mathbf{z}$ are linearly independent, their coefficients must equal the components of $\mathbf{o}_2 - \mathbf{o}_1$ in S_1:

$$\begin{cases} \alpha - \alpha' r_{11} - \beta' r_{21} - \gamma' r_{31} = (\mathbf{o}_2 - \mathbf{o}_1) \cdot \mathbf{x} \\ \beta - \alpha' r_{12} - \beta' r_{22} - \gamma' r_{32} = (\mathbf{o}_2 - \mathbf{o}_1) \cdot \mathbf{y} \\ \gamma - \alpha' r_{13} - \beta' r_{23} - \gamma' r_{33} = (\mathbf{o}_2 - \mathbf{o}_1) \cdot \mathbf{z} \end{cases}$$

8. Express α', β', γ' in Terms of α, β, γ

Rewriting the equations:

$$\begin{cases} \alpha' r_{11} + \beta' r_{21} + \gamma' r_{31} = \alpha - (\mathbf{o}_2 - \mathbf{o}_1) \cdot \mathbf{x} \\ \alpha' r_{12} + \beta' r_{22} + \gamma' r_{32} = \beta - (\mathbf{o}_2 - \mathbf{o}_1) \cdot \mathbf{y} \\ \alpha' r_{13} + \beta' r_{23} + \gamma' r_{33} = \gamma - (\mathbf{o}_2 - \mathbf{o}_1) \cdot \mathbf{z} \end{cases}$$

These can be written in matrix form:

$$\begin{pmatrix} r_{11} & r_{21} & r_{31} \\ r_{12} & r_{22} & r_{32} \\ r_{13} & r_{23} & r_{33} \end{pmatrix} \begin{pmatrix} \alpha' \\ \beta' \\ \gamma' \end{pmatrix} = \begin{pmatrix} \alpha - t_x \\ \beta - t_y \\ \gamma - t_z \end{pmatrix}$$

where $t_x = (\mathbf{o}_2 - \mathbf{o}_1) \cdot \mathbf{x}$, $t_y = (\mathbf{o}_2 - \mathbf{o}_1) \cdot \mathbf{y}$, $t_z = (\mathbf{o}_2 - \mathbf{o}_1) \cdot \mathbf{z}$.

Multiply both sides by $\mathbf{R}$ (since $\mathbf{R}\mathbf{R}^T = \mathbf{I}$ for an orthogonal matrix):

$$\begin{pmatrix} \alpha' \\ \beta' \\ \gamma' \end{pmatrix} = \mathbf{R} \begin{pmatrix} \alpha - t_x \\ \beta - t_y \\ \gamma - t_z \end{pmatrix} \qquad \mathbf{t} = \begin{pmatrix} t_\alpha \\ t_\beta \\ t_\gamma \end{pmatrix} = \mathbf{R} \begin{pmatrix} -t_x \\ -t_y \\ -t_z \end{pmatrix}$$

12. Final Transformation Formula

Thus, the transformation between the coordinates in S_1 and S_2 is:

$$\boxed{\begin{pmatrix} \alpha' \\ \beta' \\ \gamma' \end{pmatrix} = \mathbf{R} \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} + \mathbf{t}}$$

Implement (Represent) the 3D Transform

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

Translate

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{s}_x & 0 & 0 \\ 0 & \boldsymbol{s}_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ 1 \end{bmatrix}$$

Scale

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotate

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \boldsymbol{sh}_x & 0 \\ \boldsymbol{sh}_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ 1 \end{bmatrix}$$

Shear

Implement (Represent) the 3D Transform

$$\begin{matrix} X_c \\ Y_c \\ Z_c \\ 1 \end{matrix} = \begin{pmatrix} R_{33} & T_{13} \\ 0_{13} & 1 \end{pmatrix} \begin{matrix} X_w \\ Y_w \\ Z_w \\ 1 \end{matrix}$$

What if we want to optimize (backward)?

$$\begin{matrix} X_c \\ Y_c \\ Z_c \\ 1 \end{matrix} = \begin{pmatrix} R_{33} & T_{13} \\ 0_{13} & 1 \end{pmatrix} \begin{matrix} X_w \\ Y_w \\ Z_w \\ 1 \end{matrix}$$

$$R_{33} + \delta_{33}$$

What if we want to optimize (backward)?

$$\begin{matrix} X_c \\ Y_c \\ Z_c \\ 1 \end{matrix} = \begin{pmatrix} R_{33} & T_{13} \\ 0_{13} & 1 \end{pmatrix} \begin{matrix} X_w \\ Y_w \\ Z_w \\ 1 \end{matrix}$$

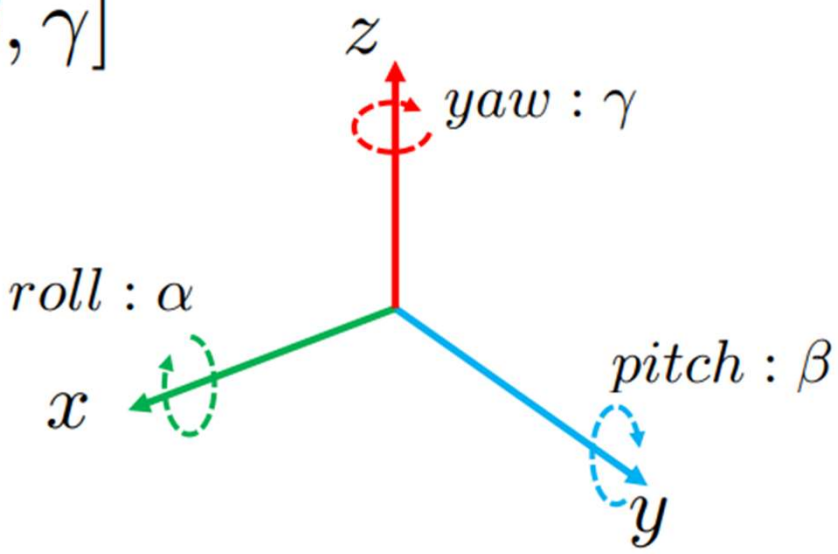
$$R_{33} + \delta_{33}$$

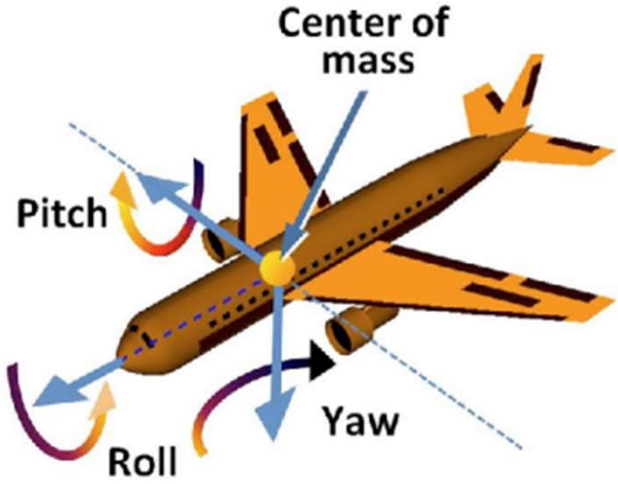
very likely not a rotation matrix

Euler Angles

Three elemental rotations sequentially applied on each axes.

$[\alpha, \beta, \gamma]$





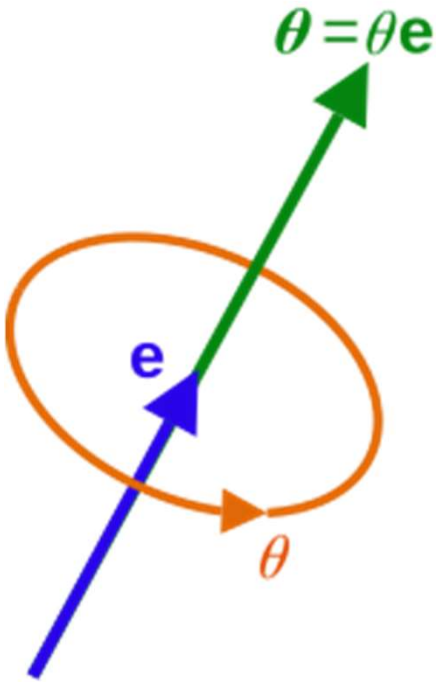
Euler Angles

$$R = R_z(\gamma)\, R_y(\beta)\, R_x(\alpha)$$

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

Order Matters

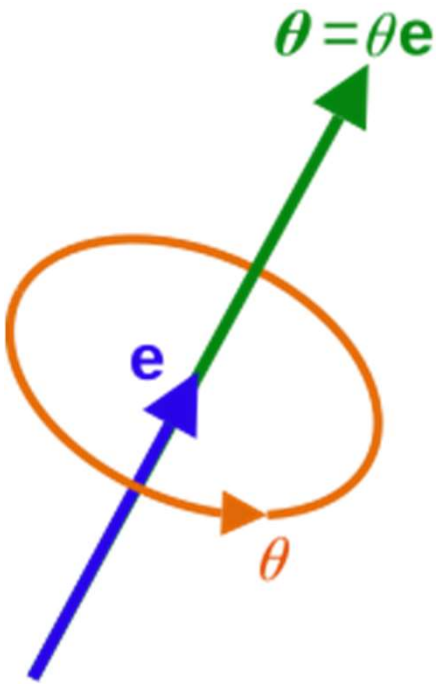
Axis-angle



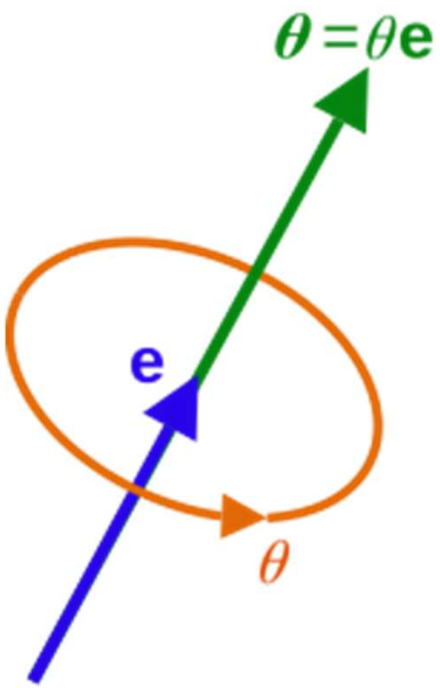
Axis-angle

Rotation Matrix:

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



Axis-angle



Rotation Matrix:

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

3. **Skew-Symmetric Matrix ($\mathbf{K}$)**:

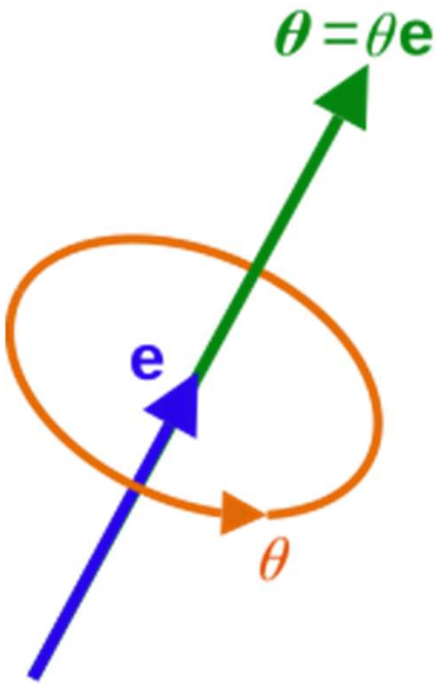
$$\mathbf{K} = \begin{bmatrix} 0 & -e_z & e_y \\ e_z & 0 & -e_x \\ -e_y & e_x & 0 \end{bmatrix}$$

4. **Rodrigues' Rotation Formula**:

$$\mathbf{R} = \mathbf{I}\cos\theta + (1-\cos\theta)(e \otimes e) + \mathbf{K}\sin\theta$$

This formula constructs the rotation matrix by combining the identity matrix, the outer product of the axis vector with itself, and the skew-symmetric matrix derived from the axis vector, all scaled appropriately by functions of the rotation angle **θ**.

Axis-angle



Rotation Matrix:

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

3. **Skew-Symmetric Matrix ($\mathbf{K}$):**

$$\mathbf{K} = \begin{bmatrix} 0 & -e_z & e_y \\ e_z & 0 & -e_x \\ -e_y & e_x & 0 \end{bmatrix}$$

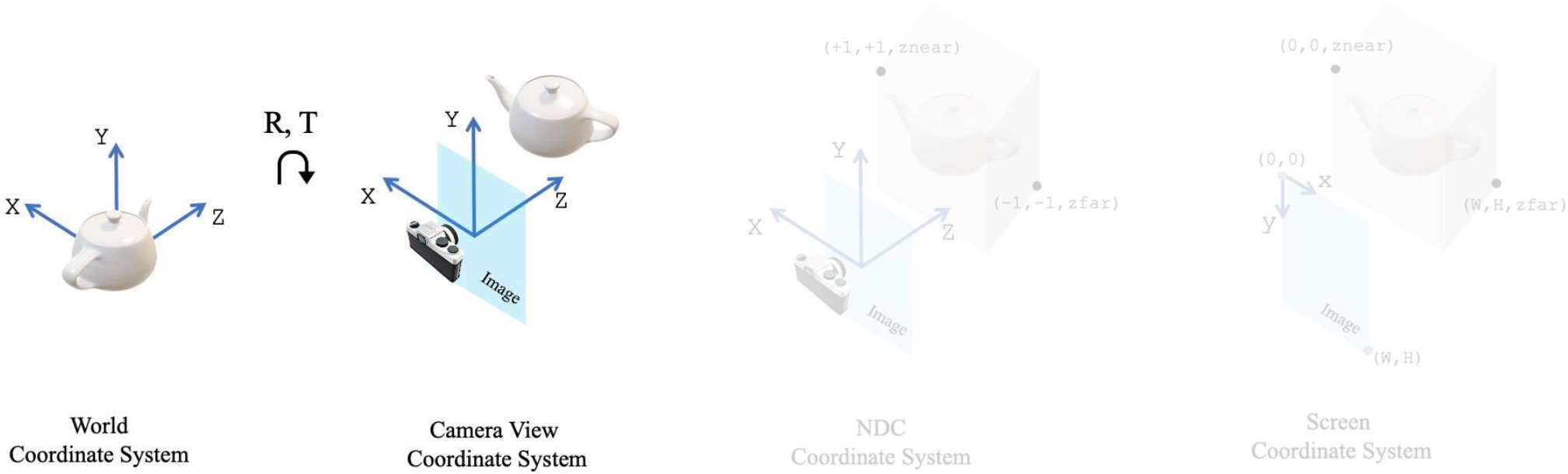
4. **Rodrigues' Rotation Formula:**

$$\mathbf{R} = \mathbf{I}\cos\theta + (1-\cos\theta)(e \otimes e) + \mathbf{K}\sin\theta$$

This formula constructs the rotation matrix by combining the identity matrix, the outer product of the axis vector with itself, and the skew-symmetric matrix derived from the axis vector, all scaled appropriately by functions of the rotation angle **θ**.

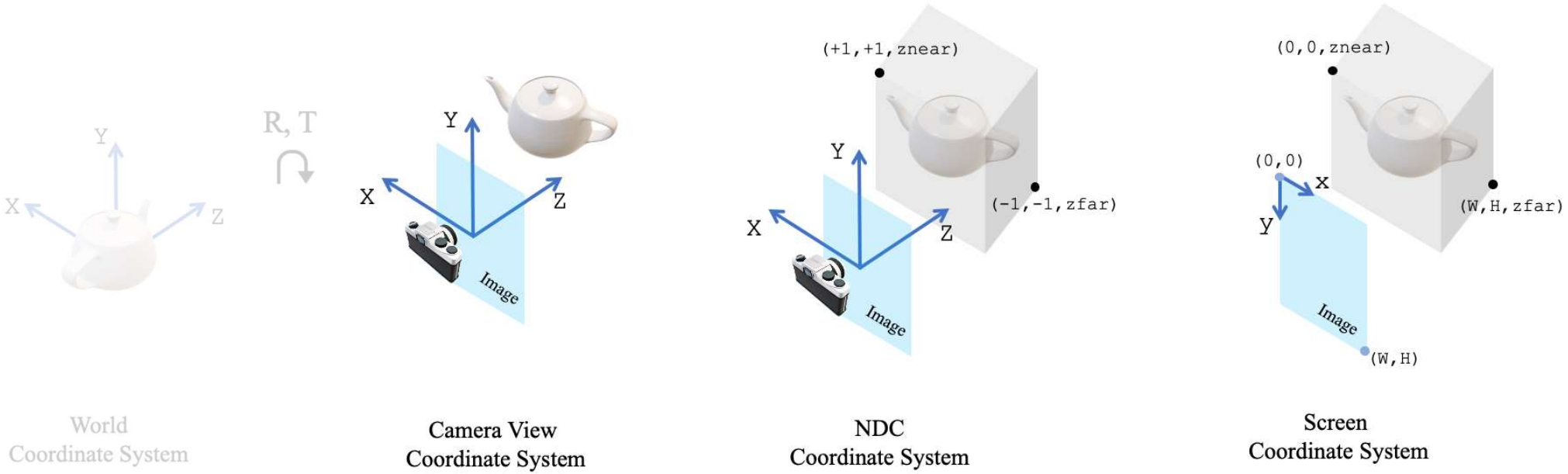
https://chatgpt.com/share/672231db-ed18-8005-8b31-8ef9655a1b60

World Space to Camera (View) Space



3d Rotation (euler angles/axis-angle, etc…) and Translation

Camera Space to Image Space



Projection Transformation

GAMES101: 现代计算机图形学入门

https://sites.cs.ucsb.edu/~lingqi/teaching/games101.html

光栅化成像 [实时软阴影 (NVIDIA)] 几何表示 [Adobe Photoshop 中的钢笔工具] 光线传播理论 [光子映射 (Jensen)] 动画与模拟 [流体模拟 (Muller)]

Credit to Lingqi Yan

Projection Transformation

- Projection in Computer Graphics
  - 3D to 2D
  - Orthographic projection
  - Perspective projection

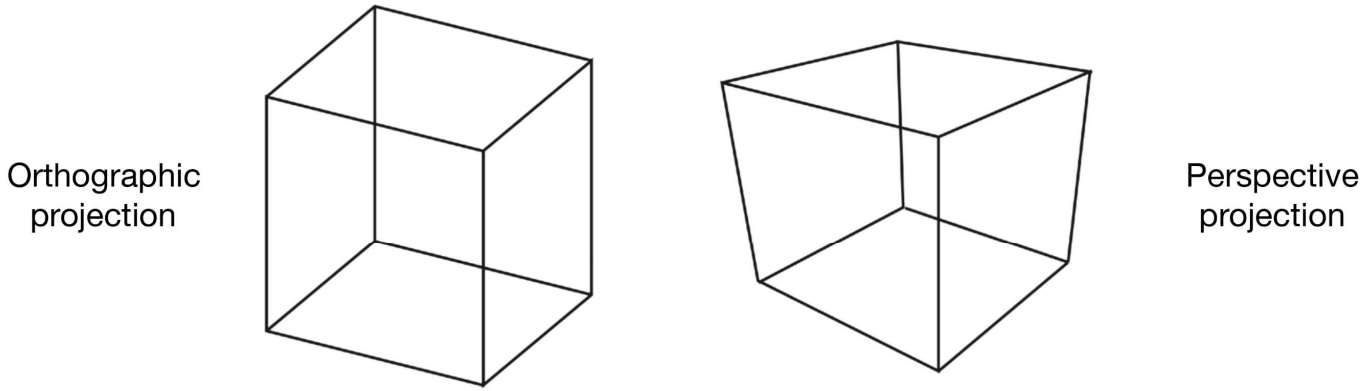
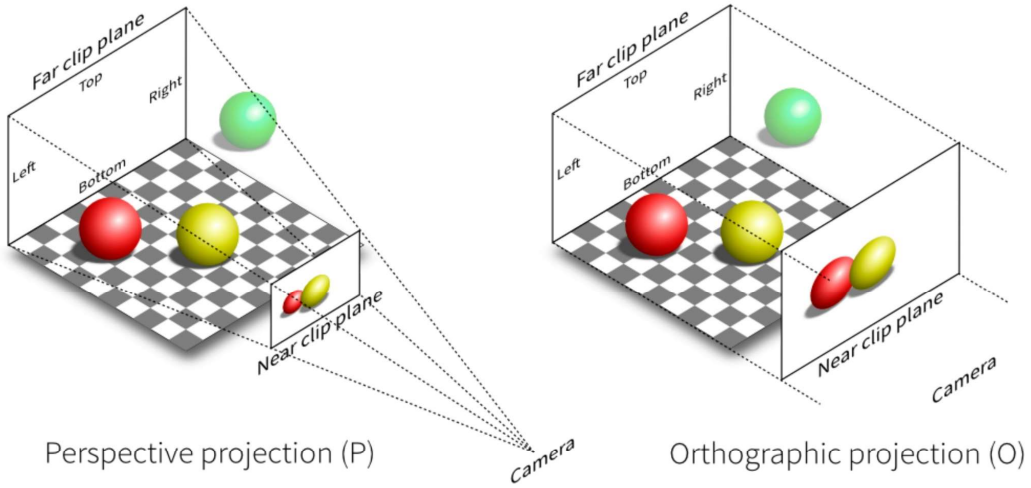


Fig. 7.1 from *Fundamentals of Computer Graphics, 4th Edition*

Projection Transformation

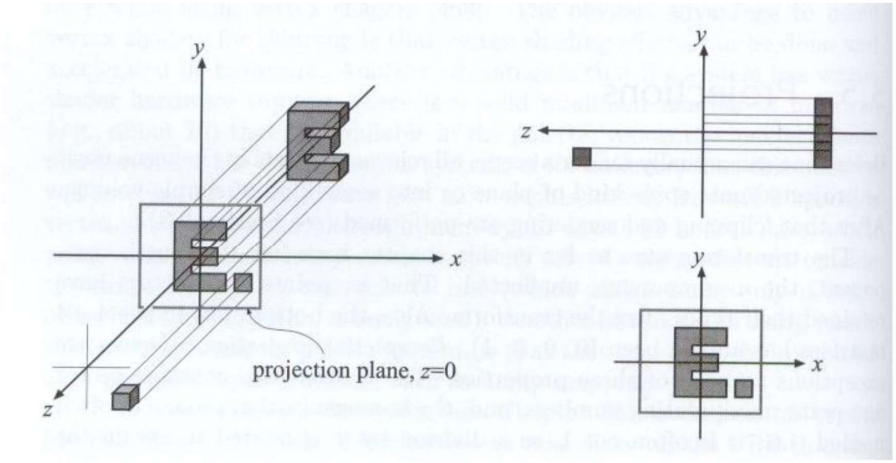
- Perspective projection vs. orthographic projection



https://stackoverflow.com/questions/36573283/from-perspective-picture-to-orthographic-picture

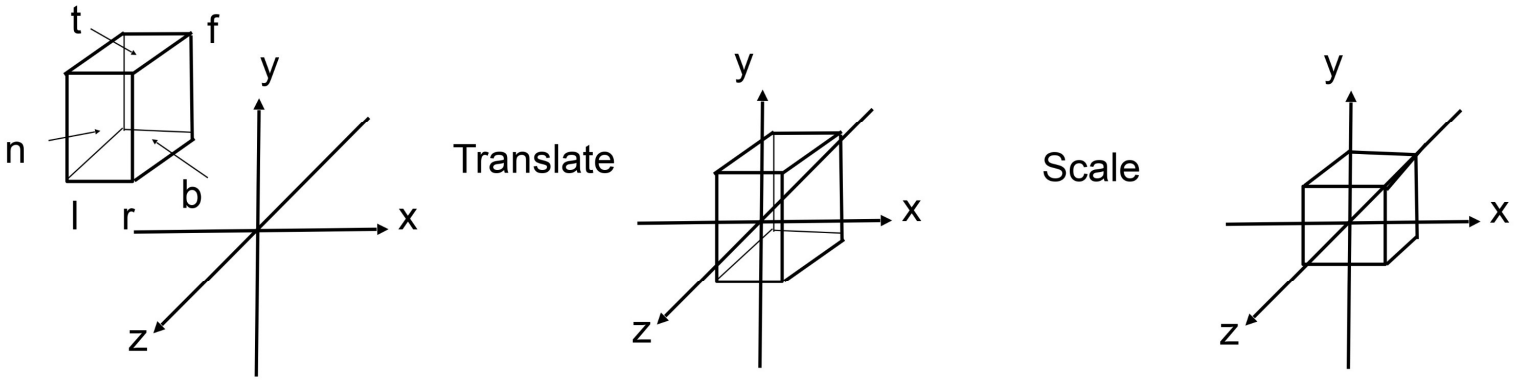
Orthographic Projection

- A simple way of understanding
  - Camera located at origin, looking at -Z, up at Y (looks familiar?)
  - Drop Z coordinate
  - Translate and scale the resulting rectangle to $[-1, 1]^2$



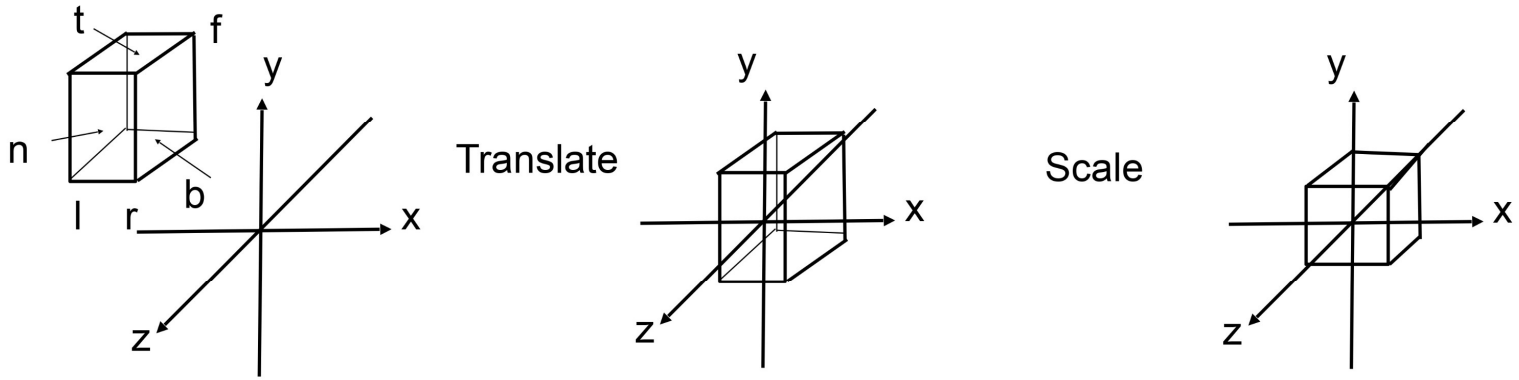
Orthographic Projection

- In general
  - We want to map a cuboid [l, r] x [b, t] x [**f, n**] to the “canonical (正则、规范、标准)” cube $[-1, 1]^3$



Orthographic Projection

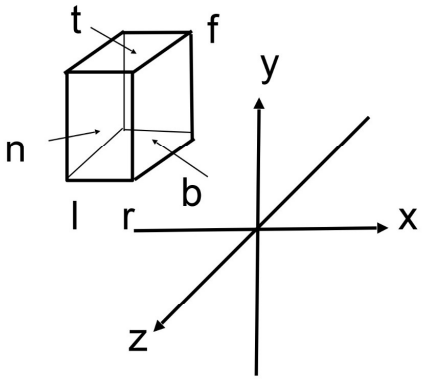
- Slightly different orders (to the “simple way”)
  - Center cuboid by translating
  - Scale into “canonical” cube



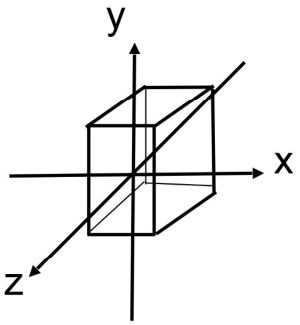
Orthographic Projection

- Transformation matrix?
  - Translate (**center** to origin) first, then scale (length/width/height to **2**)

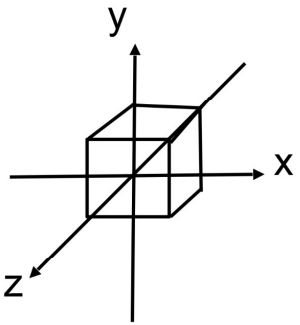
$$M_{ortho} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{n-f} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -\frac{r+l}{2} \\ 0 & 1 & 0 & -\frac{t+b}{2} \\ 0 & 0 & 1 & -\frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Translate

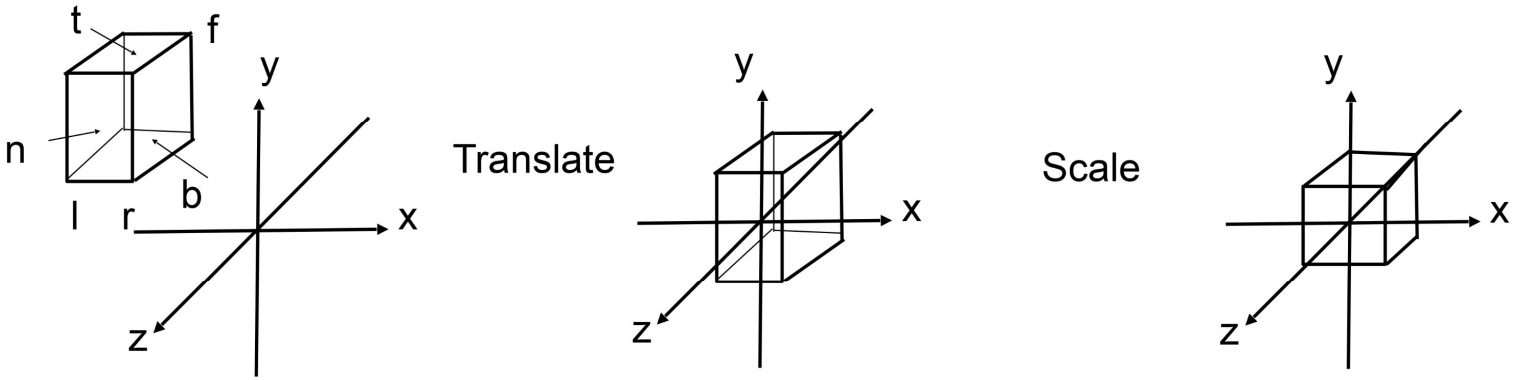


Scale



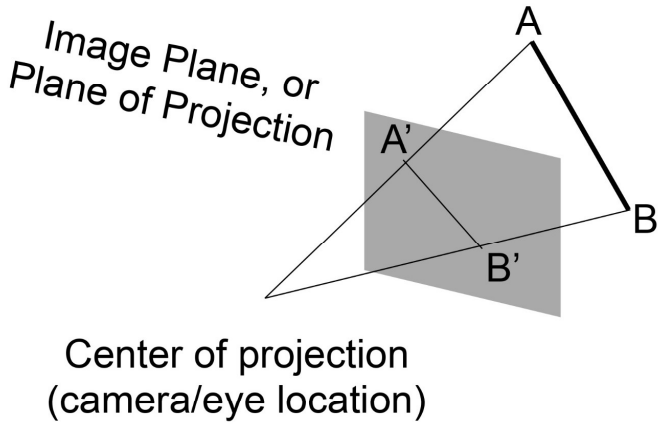
Orthographic Projection

- Caveat
  - Looking at / along -Z is making near and far not intuitive (n > f)
  - FYI: that's why OpenGL (a Graphics API) uses left hand coords.



Perspective Projection

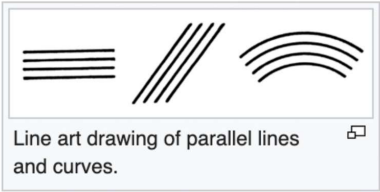
- Most common in Computer Graphics, art, visual system
- Further objects are smaller
- Parallel lines not parallel; converge to single point



Perspective Projection

- Euclid was wrong??!!

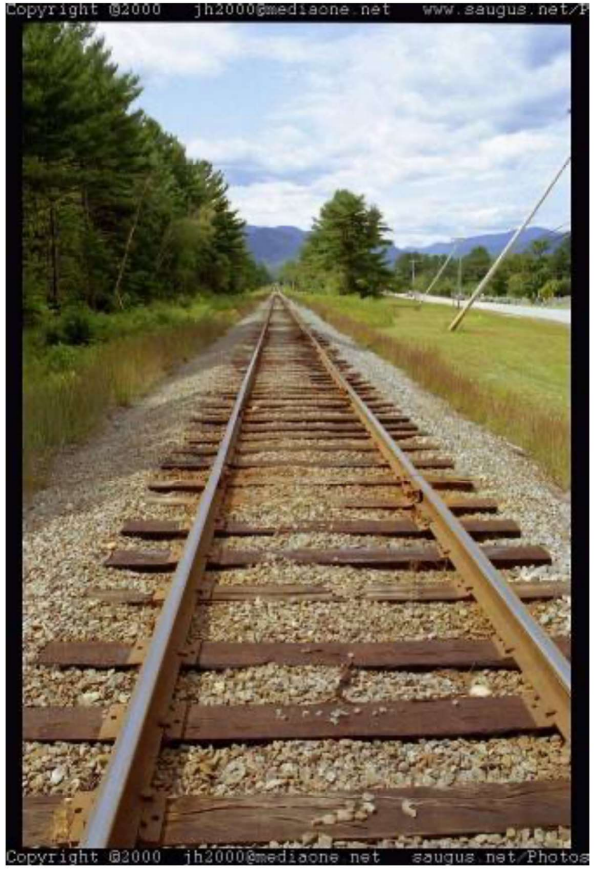
In geometry, **parallel** lines are lines in a plane which do not meet; that is, two lines in a plane that do not intersect or touch each other at any point are said to be parallel. By extension, a line and a plane, or two planes, in three-dimensional Euclidean space that do not share a point are said to be parallel. However, two lines in three-dimensional space which do not meet must be in a common plane to be considered parallel; otherwise they are called skew lines. Parallel planes are planes in the same three-dimensional space that never meet.



Line art drawing of parallel lines and curves.

Parallel lines are the subject of Euclid's parallel postulate.[1] Parallelism is primarily a property of affine geometries and Euclidean geometry is a special instance of this type of geometry. In some other geometries, such as hyperbolic geometry, lines can have analogous properties that are referred to as parallelism.

https://en.wikipedia.org/wiki/Parallel_(geometry)



Perspective Projection

- Before we move on
- Recall: property of homogeneous coordinates
  - (x, y, z, 1), (kx, ky, kz, k != 0), **(xz, yz, z^2, z != 0)** all represent the same point (x, y, z) in 3D
  - e.g. (1, 0, 0, 1) and (2, 0, 0, 2) both represent (1, 0, 0)
- Simple, but useful

Perspective Projection

- How to do perspective projection
  - First “squish” the frustum into a cuboid (n -> n, f -> f) ($M_{persp->ortho}$)
  - Do orthographic projection (M_{ortho}, already known!)

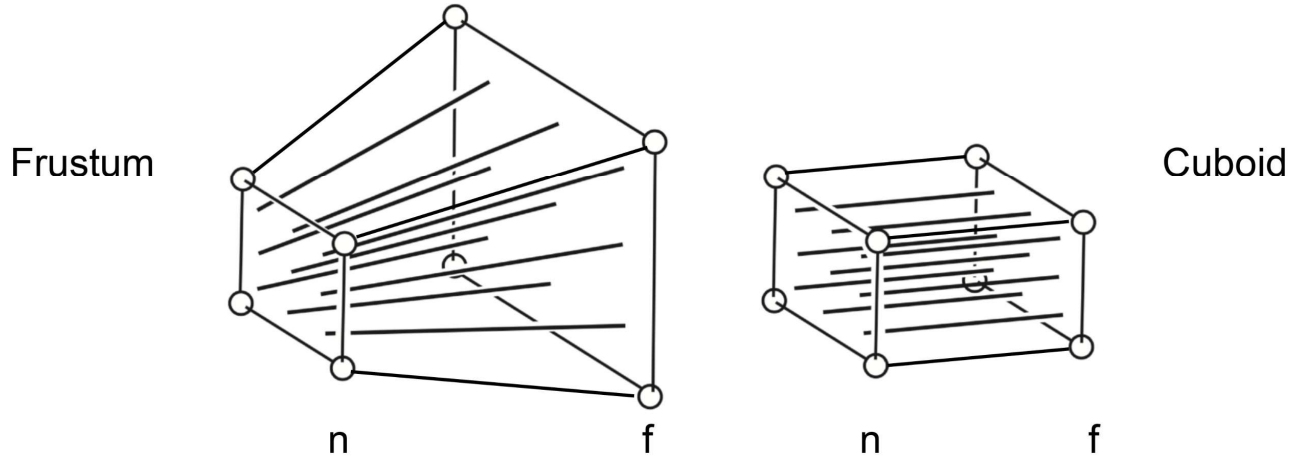
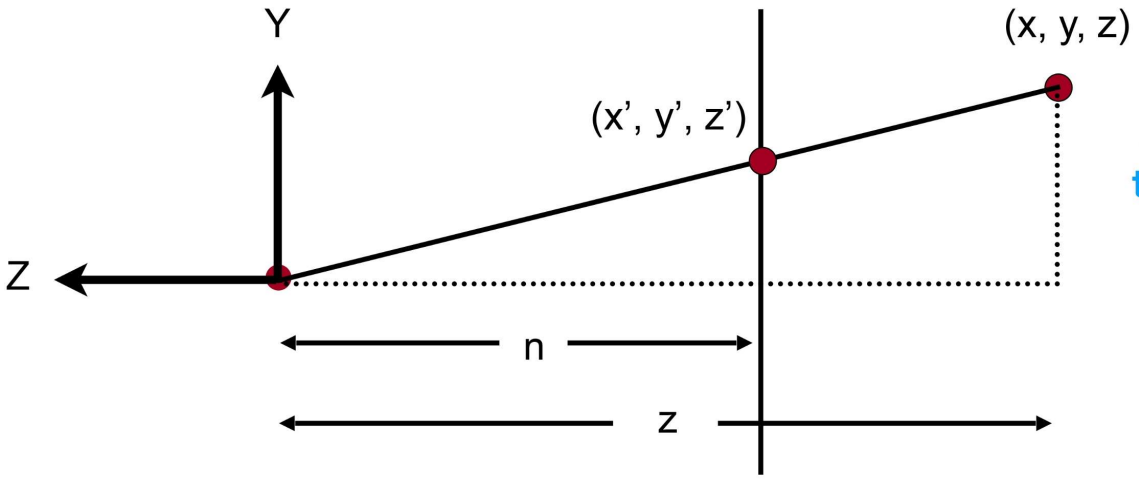


Fig. 7.13 from *Fundamentals of Computer Graphics, 4th Edition*

Perspective Projection

- In order to find a transformation
  - Recall the key idea: Find the relationship between transformed points (x', y', z') and the original points (x, y, z)



similar triangle

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

Perspective Projection

- In order to find a transformation
  - Find the relationship between transformed points (x', y', z') and the original points (x, y, z)

$$y' = \frac{n}{z}y \qquad x' = \frac{n}{z}x$$ (similar to y')

- In homogeneous coordinates,

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} nx/z \\ ny/z \\ \text{unknown} \\ 1 \end{pmatrix} \overset{\text{mult. by z}}{==} \begin{pmatrix} nx \\ ny \\ \text{still unknown} \\ z \end{pmatrix}$$

Perspective Projection

- So the “squish” (persp to ortho) projection does this

$$M_{persp\to ortho}^{(4\times 4)} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} nx \\ ny \\ \text{unknown} \\ z \end{pmatrix}$$

- Already good enough to figure out part of $M_{persp->ortho}$

$$M_{persp\to ortho} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ ? & ? & ? & ? \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

WHY?

Perspective Projection

- How to figure out the third row of $M_{persp->ortho}$
  - Any information that we can use?

$$M_{persp\to ortho} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ ? & ? & ? & ? \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

- Observation: the third row is responsible for z'
  - Any point on the near plane will not change
  - Any point's z on the far plane will not change

Perspective Projection

- Any point on the near plane will not change

$$M_{persp\to ortho}^{(4\times 4)} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} nx \\ ny \\ \text{unknown} \\ z \end{pmatrix}$$

replace z with n

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

- So the third row must be of the form (0 0 A B)

$$\begin{pmatrix} 0 & 0 & A & B \end{pmatrix} \begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix} = n^2$$

n^2 has nothing to do with x and y

Perspective Projection

- What do we have now?

$$\begin{pmatrix} 0 & 0 & A & B \end{pmatrix} \begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix} = n^2 \quad \Rightarrow \quad An + B = n^2$$

- Any point's z on the far plane will not change

$$\begin{pmatrix} 0 \\ 0 \\ f \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 \\ 0 \\ f \\ 1 \end{pmatrix} == \begin{pmatrix} 0 \\ 0 \\ f^2 \\ f \end{pmatrix} \quad \Rightarrow \quad Af + B = f^2$$

Perspective Projection

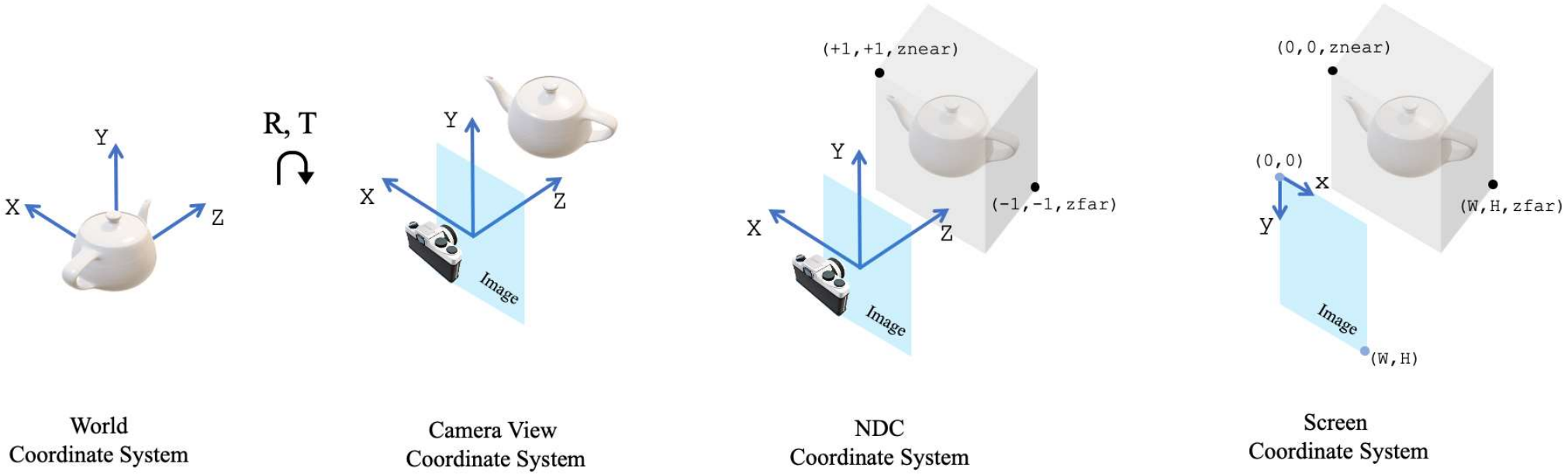
- Solve for A and B

$$An + B = n^2$$
$$Af + B = f^2$$

$$A = n + f$$
$$B = -nf$$

- Finally, every entry in $M_{persp->ortho}$ is known!
- What's next?
  - Do orthographic projection (M_{ortho}) to finish
  - $M_{persp} = M_{ortho} M_{persp \to ortho}$

Coordinate Systems: Whole Pipeline



World Coordinate System

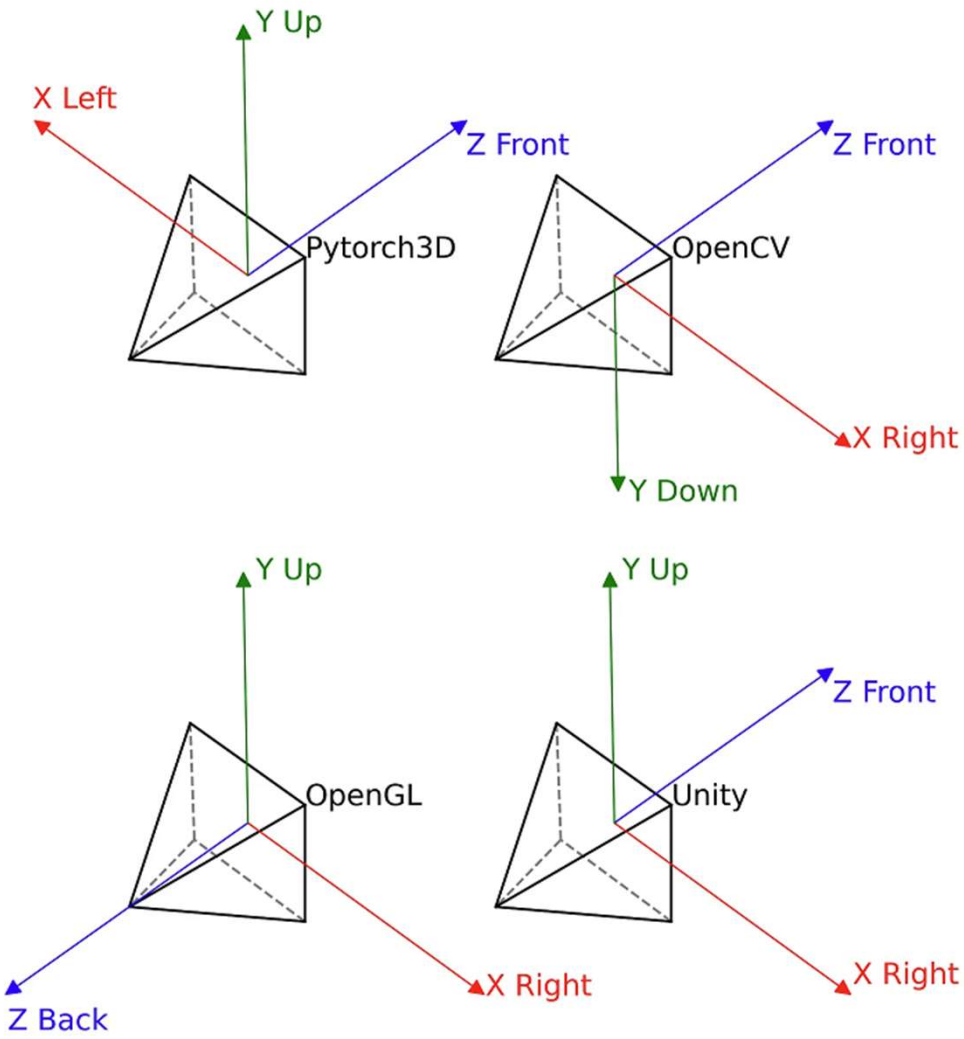
Camera View Coordinate System

NDC Coordinate System

Screen Coordinate System

All Homogeneous Matrix

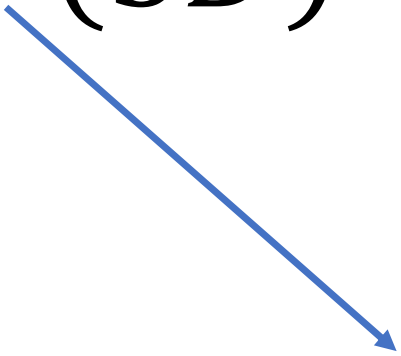
Different Coordinates with Different Libs



How to Convert between each other?

Revisit Rendering & Reconstruction (3D Vision)

$$F(3D) = 2D$$

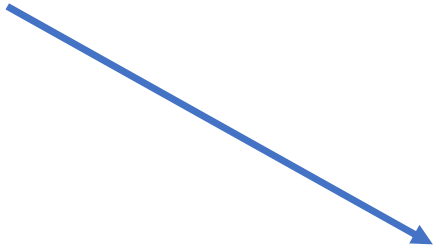


World space to camera space

Camera space projecting to (normalized space to) image space

Revisit Rendering & Reconstruction (3D Vision)

$$R(3D, light, material) = 2D$$

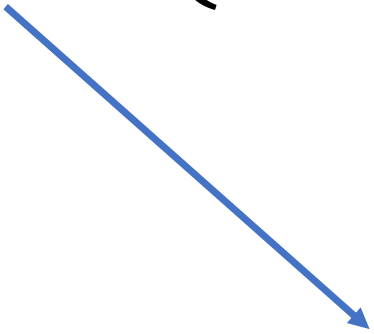


Rendering (next course)

- How to represent 3D
- How exactly the rendering function is

Revisit Rendering & Reconstruction (3D Vision)

$$F^{-1}(2D) = 3D$$



Reconstruction (Many remaining courses)

中国科学技术大学

University of Science and Technology of China

谢谢观看!