Lecture 11 — Camera Geometry

Lecture 11

Kaveh Fathian
Computer Science Department, Colorado School of Mines

Learning Objectives

By the end of this lecture, you should be able to:

- Explain the pinhole camera model, its connection to a lens camera, and the effects of focal length and object distance.
- Use homogeneous coordinates to represent and compose transformations, and distinguish rigid, similarity, affine, and projective transformations.
- Explain how extrinsic and intrinsic parameters map a world point to an image pixel.
- Project 3D points using the camera matrix and OpenCV.

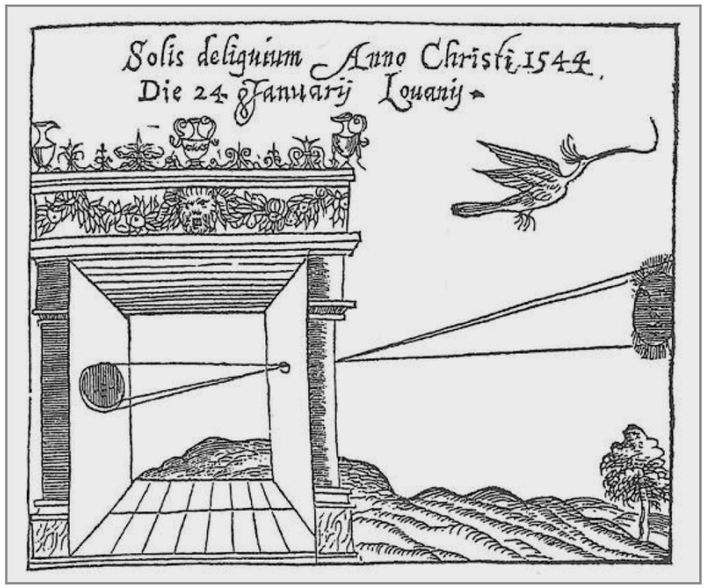
A Brief History: The Camera Obscura

- Light through a small hole in a dark room forms an **inverted image** of the scene on the opposite wall.
- This pinhole effect was described by Mozi (c. 470–391 BC); Aristotle (384–322 BC) used it to watch solar eclipses.

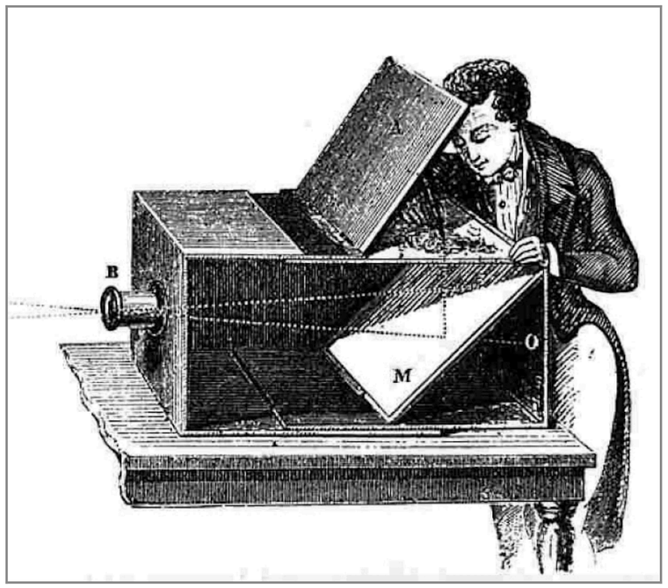
Mozi (c. 470–391 BC)



Gemma Frisius: solar eclipse of 1544 viewed through a camera obscura (published 1545)



A 19th-century illustration of a camera obscura box with mirror



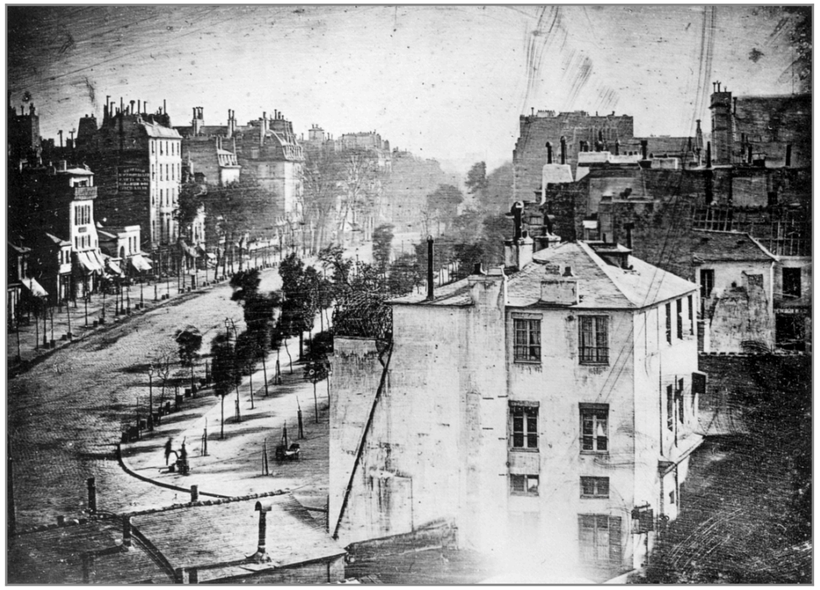
Frisius's woodcut is the oldest known published drawing of a camera obscura.

A Brief History: Recording the Image

Oldest surviving photograph
Niépce, c. 1827



First photograph with people
Daguerre, 1838

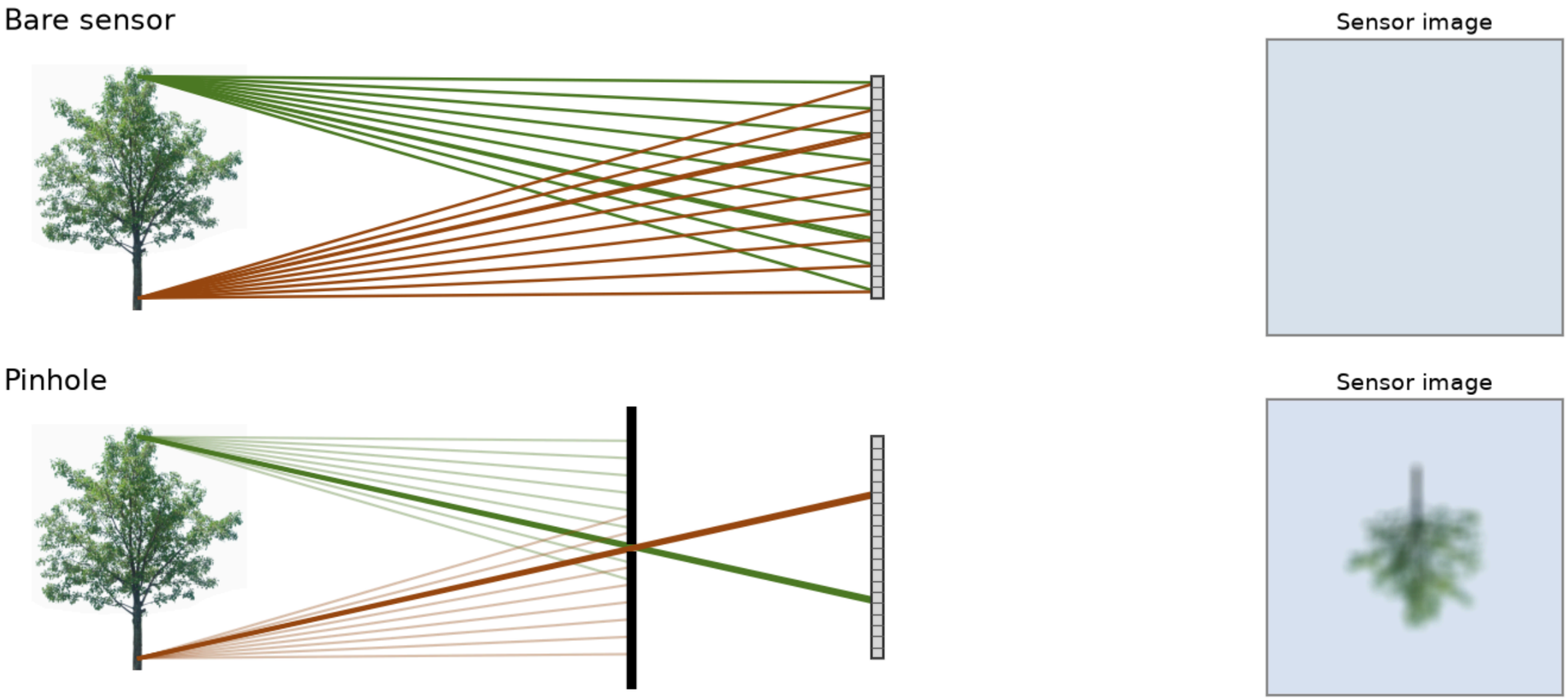


First digital camera
Sasson (Kodak), 1975



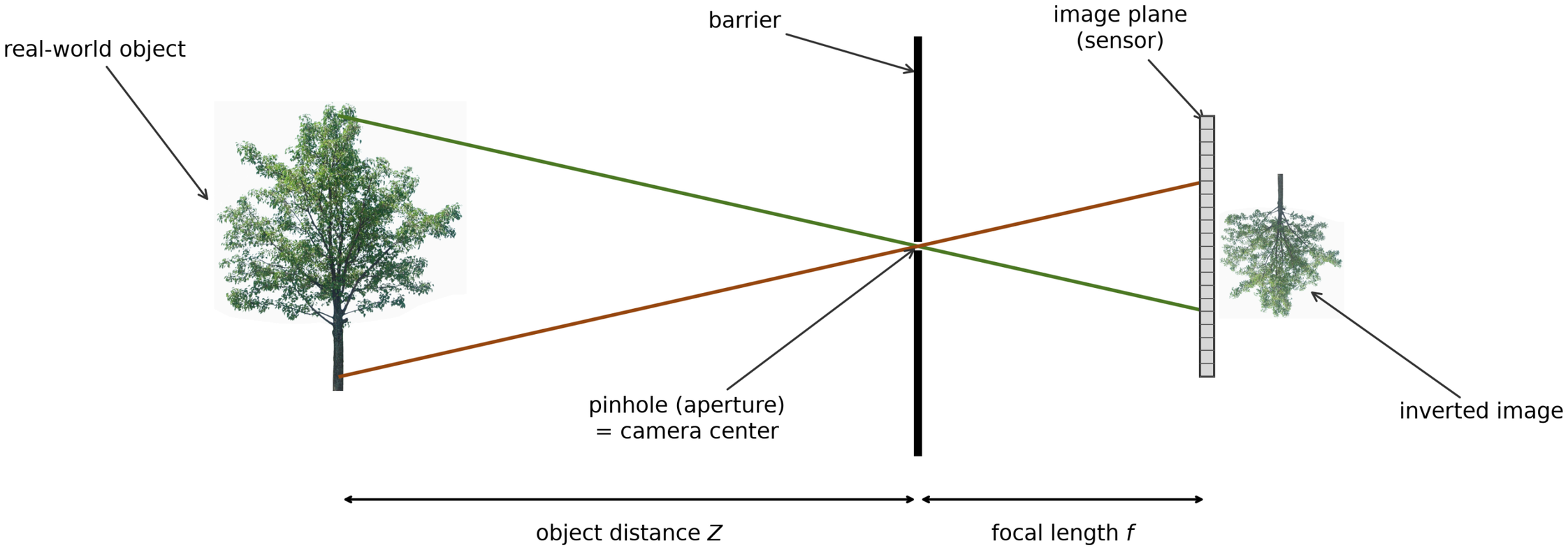
- **Niépce:** exposure of about 8 hours; held at the Harry Ransom Center, UT Austin.
- **Daguerre:** moving traffic left no trace; a man having his boots polished stood still long enough to appear.
- **Sasson:** 100×100 pixels (0.01 megapixels), black and white, stored on cassette tape; 23 seconds per image, about 8 pounds.

Bare Sensor vs. Pinhole



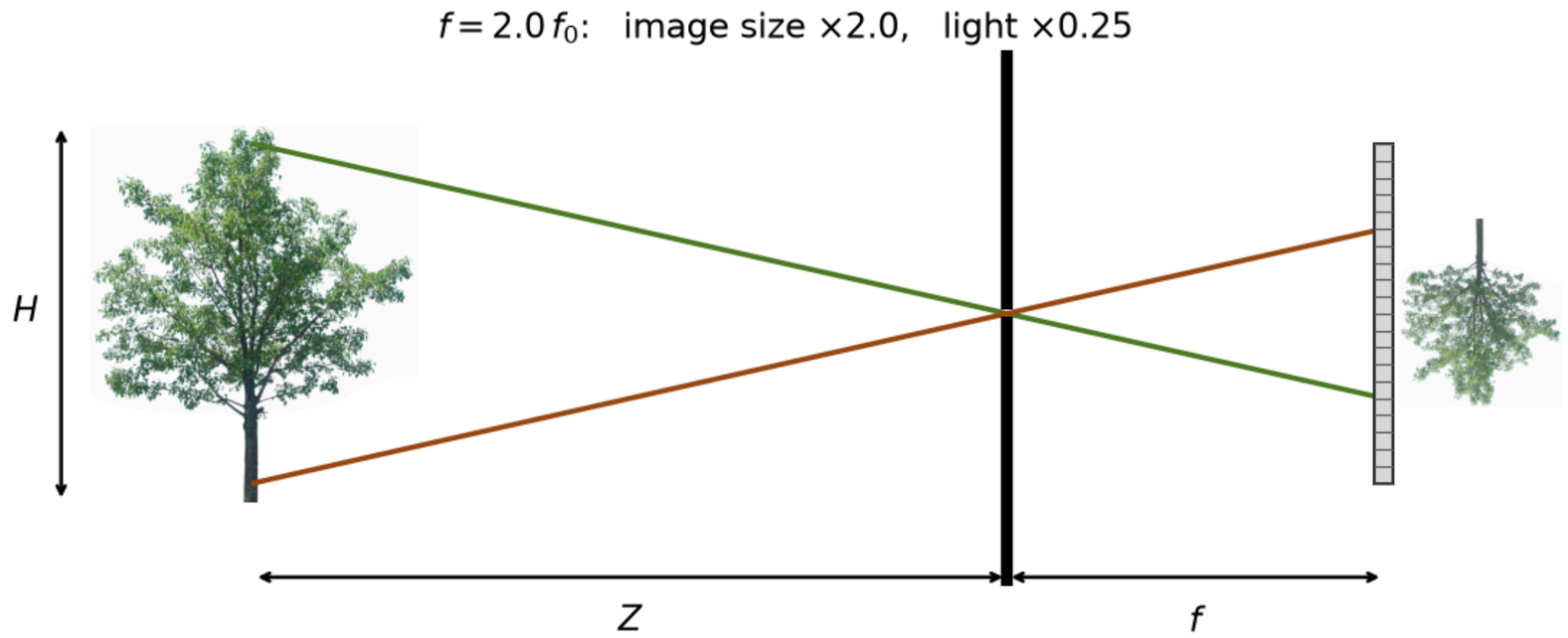
- **Bare sensor:** every scene point reaches every pixel, so every pixel records nearly the same average color.
- **Pinhole:** the barrier blocks all but one ray from each point, so each pixel sees one scene point: an **inverted** image.

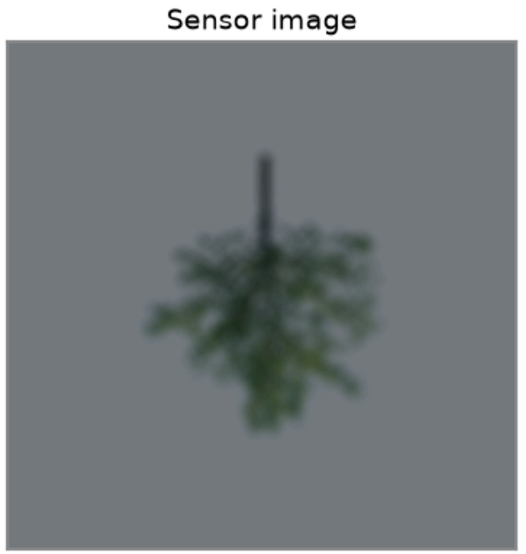
Pinhole Camera Terms



- The **pinhole** is the **camera center** (center of projection): every ray that forms the image passes through it.
- The sensor lies in the **image plane**, a distance f (the **focal length**) behind the pinhole.

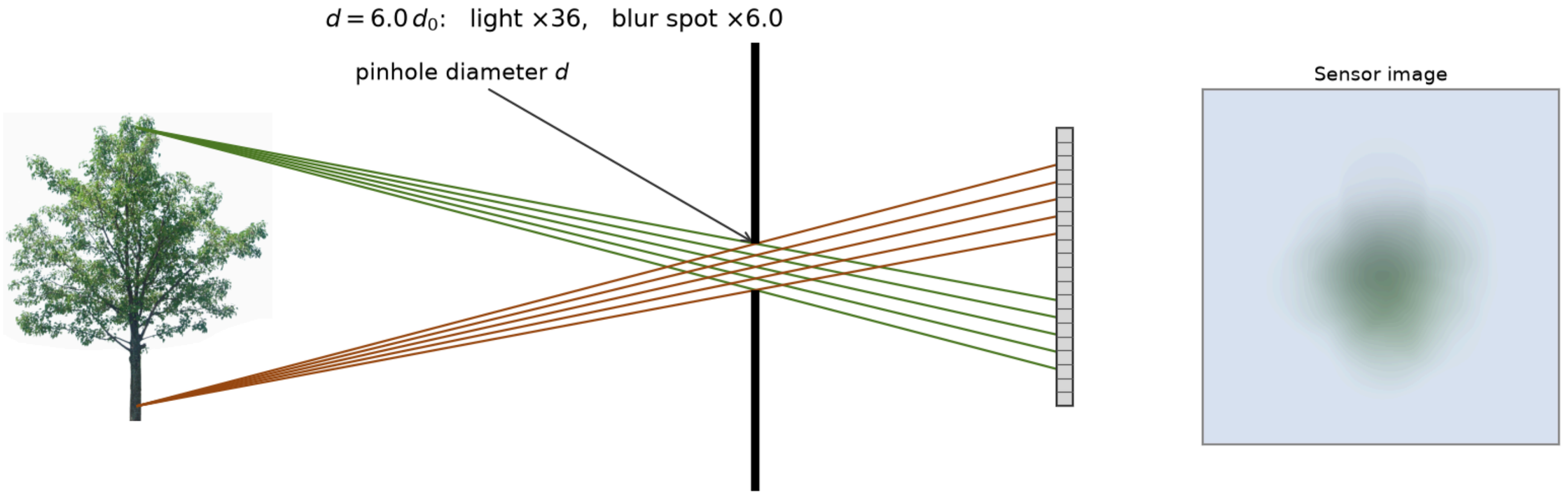
Focal Length





- Image size is $f\, H/Z$: doubling f doubles the image and narrows the field of view.
- With the same pinhole, the light reaching each pixel is proportional to $1/f^2$: doubling f gives ¼ of the light.

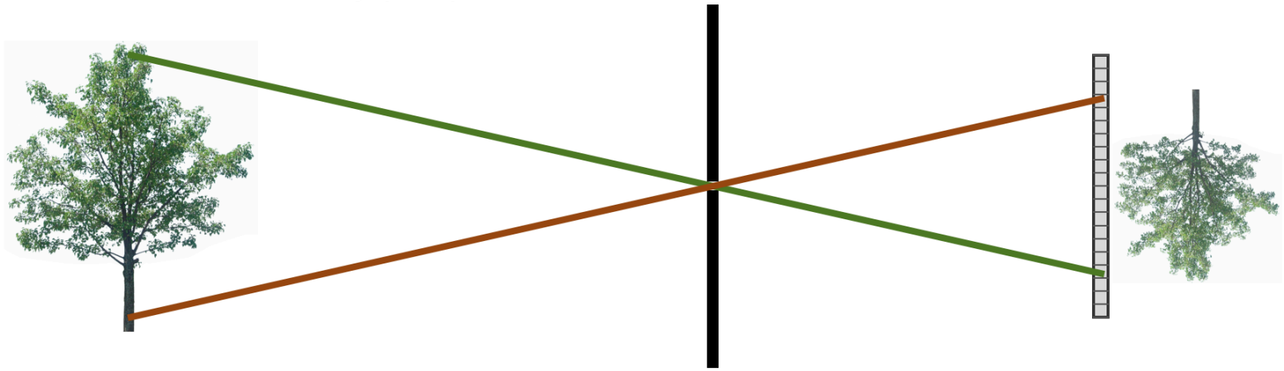
Pinhole Size: Sharpness vs. Light



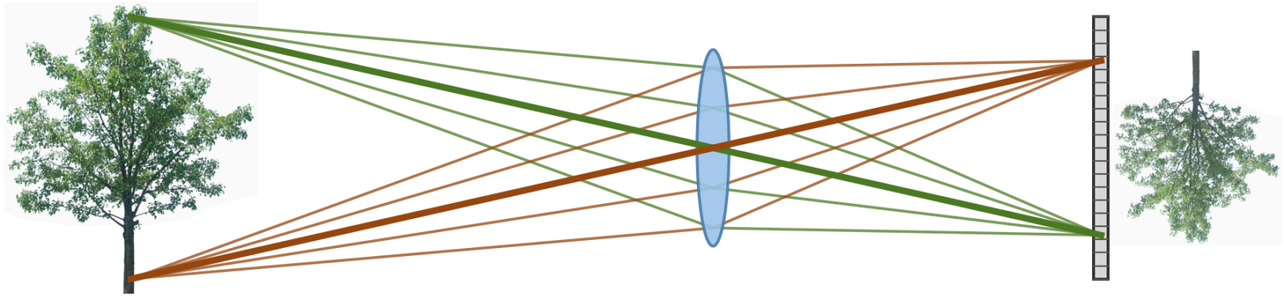
- Light collected grows with the pinhole's area ($\propto d^2$), but each scene point spreads over a spot of size $\propto d$.
- A very small pinhole is sharp but dark; below a certain size, **diffraction** blurs the image again, so the pinhole cannot be made arbitrarily small.

The Lens Camera

Pinhole camera: one ray per point



Lens camera: a bundle of rays per point

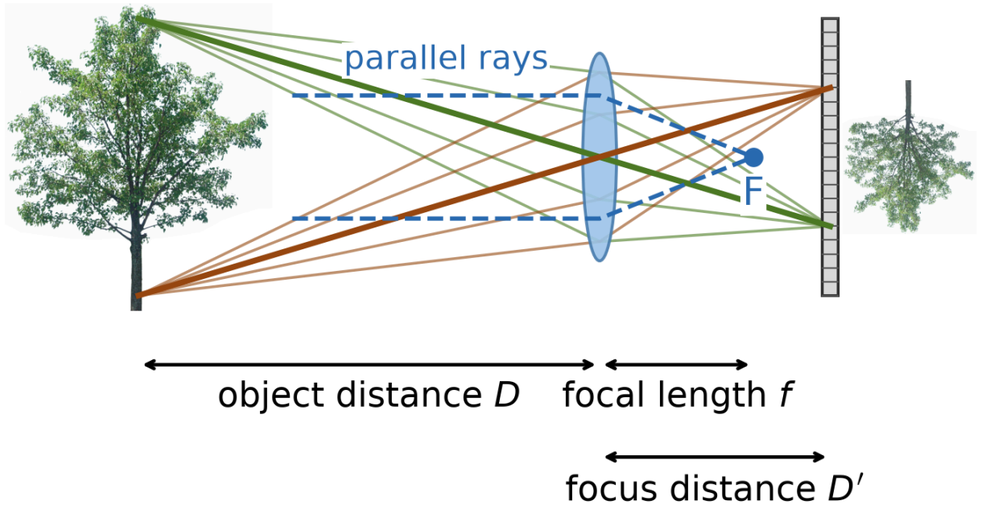


- A lens collects a **bundle of rays** from each scene point and focuses them back to one image point: far **more light** than a pinhole, **without the blur**.
- The **central ray** (thick) passes through the lens center undeflected, just as through a pinhole. When the lens is in focus, the pinhole model gives where each point lands.

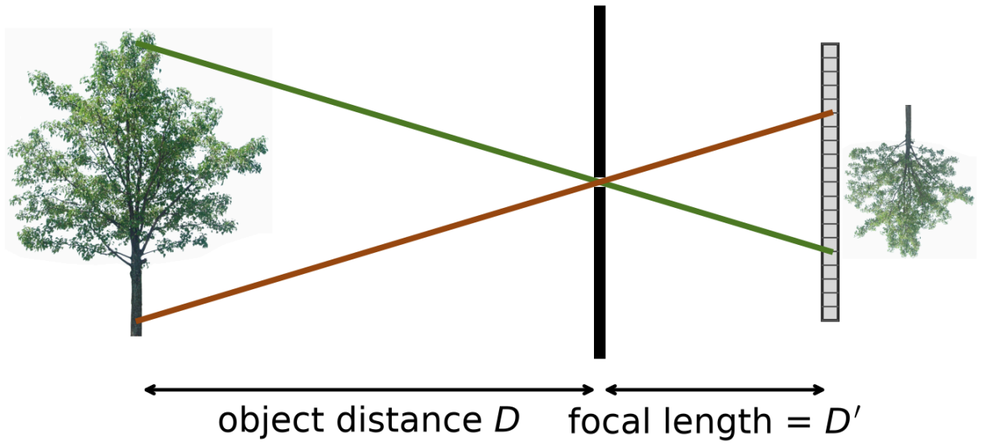
Focal Length of a Lens

- In a lens camera, the **focal length** f is the distance at which **parallel rays** intersect (the focal point F).
- The image of an object at distance D forms at the **focus distance** D': $\frac{1}{D} + \frac{1}{D'} = \frac{1}{f}$.
- The pinhole model describes a lens camera if:
  - We use only central rays
  - The lens is in focus
  - The pinhole's focal length equals the lens's focus distance D'; for a distant object, $D' = f$.
- In this course, **focal length** f means the **aperture–sensor distance**, as in the pinhole camera.

Lens camera (in focus)



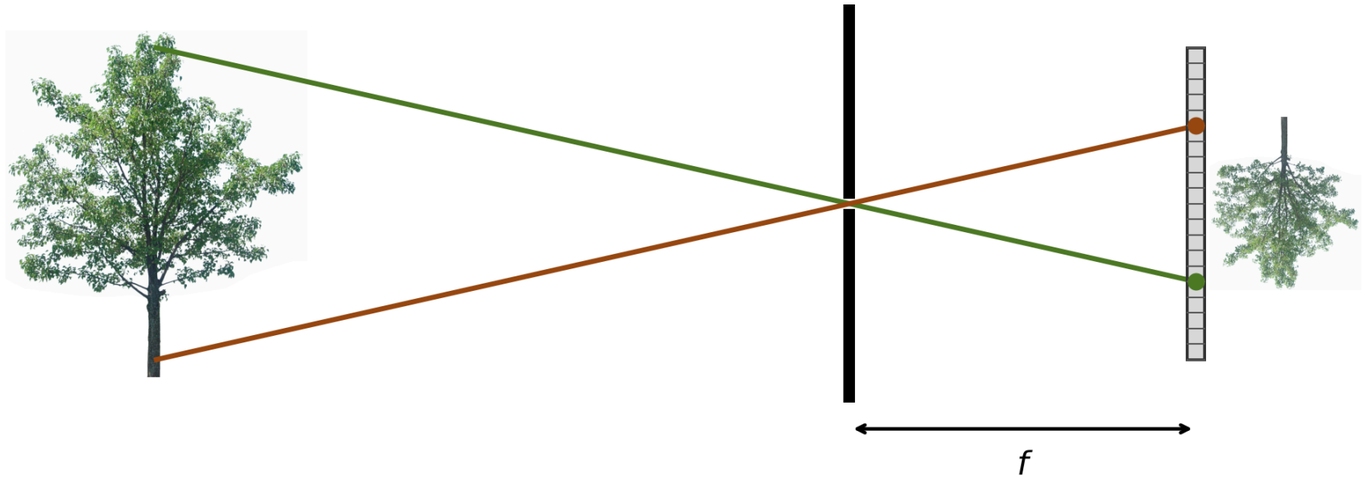
Pinhole model: central rays only



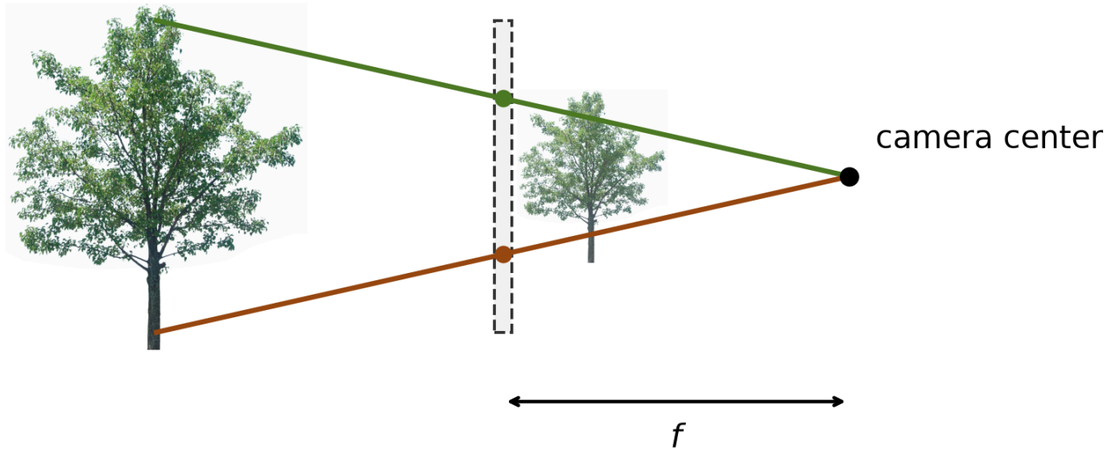
The Virtual Image Plane

- Placing a virtual image plane a distance f **in front of** the camera center gives the same image, the same size, but **upright**.
- The virtual plane is mathematically equivalent and easier to work with, so from now on we will use it instead.

Image plane behind the camera center: inverted image

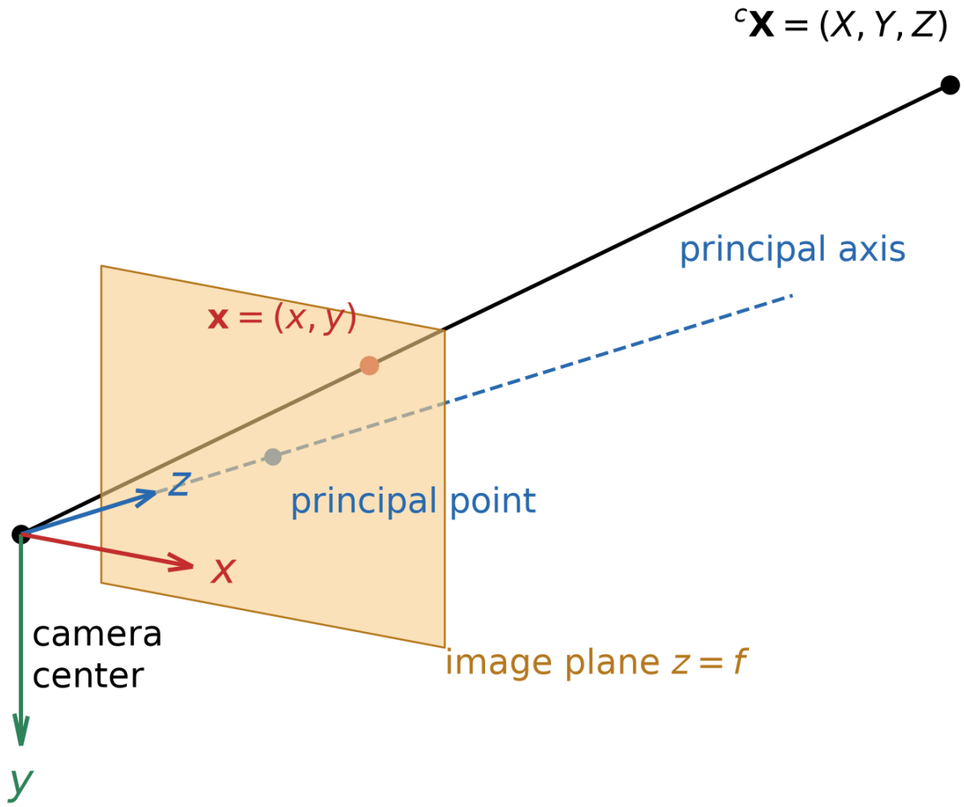


Virtual image plane in front of the camera center: upright image



The Camera Coordinate Frame

- Origin is at the **camera center**.
- Convention: x right, y down, z forward.
- The z-axis is the **principal (optical) axis**.
- The image plane is $z = f$.
- The principal axis meets the image plane at the **principal point**.
- A 3D point ${}^c\mathbf{X} = (X, Y, Z)$ in camera coordinates projects to the point $\mathbf{x} = (x, y)$ where the ray to the camera center crosses the image plane.



Perspective Projection

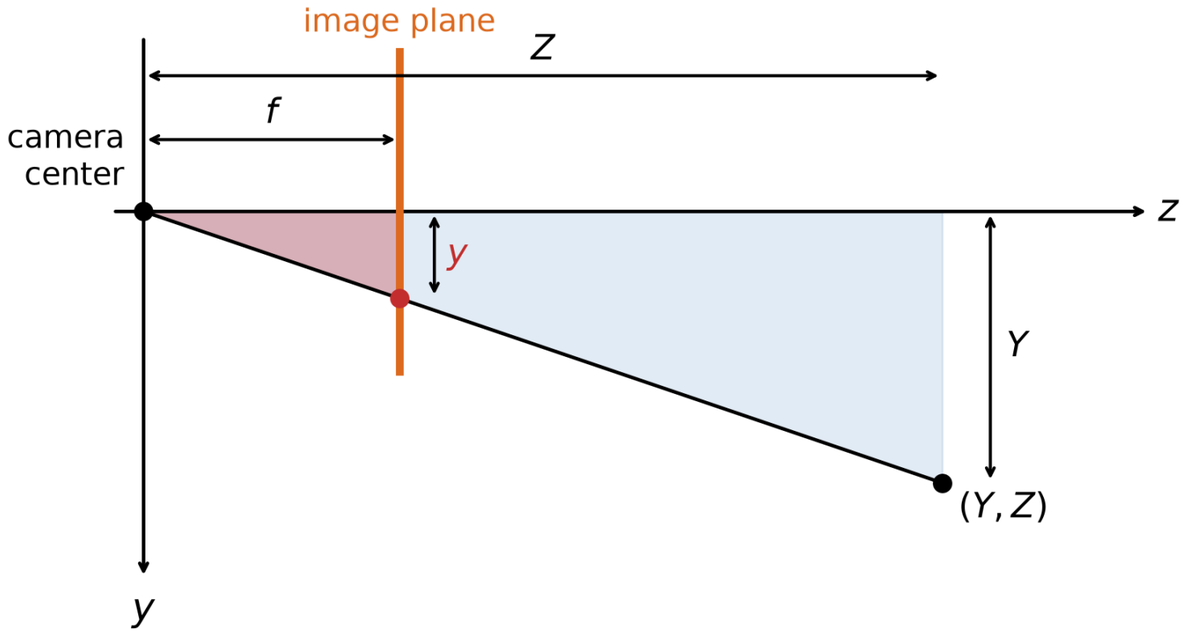
- Side view (y–z plane): the two shaded triangles are **similar**, so

$$\frac{y}{f} = \frac{Y}{Z}, \qquad \text{and likewise} \qquad \frac{x}{f} = \frac{X}{Z}.$$

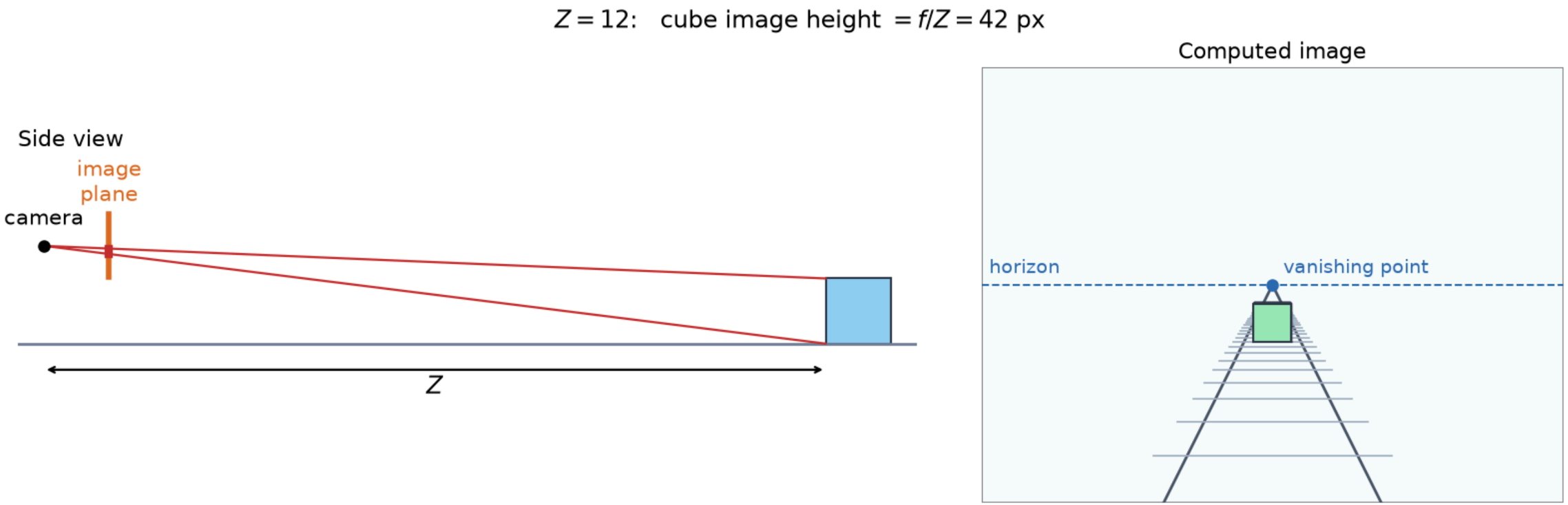
- **Perspective projection:**

$$(X,\ Y,\ Z) \ \mapsto\ \left(f\,\frac{X}{Z},\ f\,\frac{Y}{Z}\right)$$

- The division by Z makes the projection **nonlinear** in (X, Y, Z).
- Homogeneous coordinates (defined later) will make it a matrix product.



Consequences of Perspective Projection



- **Size** $\propto 1/Z$**:** doubling the distance halves the image of the cube.
- **Depth is lost:** every point on a ray through the camera center projects to the same image point.
- **Vanishing points:** parallel 3D lines (the rails) meet at one image point, on the **horizon**.

Consequences of Perspective Projection

Forced perspective: Leaning Tower of Pisa



StruckDuck, *Impossible Gravity Slide Illusion*



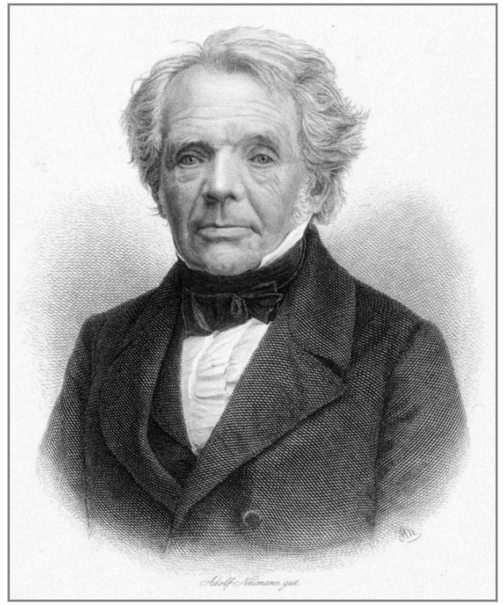
Fundamental Equations of Computer Vision

1. Image Filtering	$$y[m,n] = \sum_{k,l} h[k,l]\, I[m+k, n+l]$$
2. Optical Flow	$$I(x,y,t) = I(x+\Delta x,\ y+\Delta y,\ t+\Delta t)$$
3. Camera Geometry	$$\lambda\, \mathbf{x} = \mathbf{K} \begin{bmatrix} {}^{c}\mathbf{R}_{w} & {}^{c}\mathbf{t}_{w} \end{bmatrix} {}^{w}\mathbf{X} \qquad \mathbf{x}^{T} \mathbf{F} \mathbf{x}' = 0$$
4. Machine Learning	$$\arg\min_{\mathcal{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \left\| \mathbf{x} - \boldsymbol{\mu}_i \right\|^2 \qquad y = \varphi\left(\mathbf{w}^{T} \mathbf{x} + b\right)$$

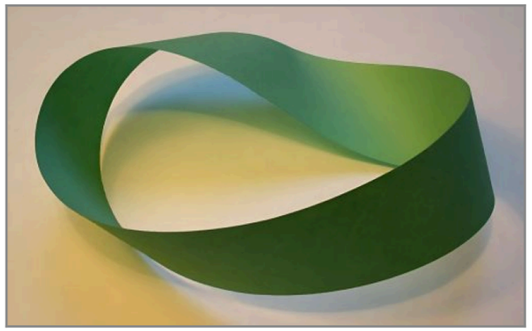
Homogeneous Coordinates

- Introduced by **August Ferdinand Möbius** (1827), as an alternative to Cartesian coordinates for projective geometry.
- Add **one extra coordinate**: a 2D point gets 3 numbers, a 3D point gets 4.
- Why we use them in computer vision:
  - **Translation** becomes a matrix product
  - **Perspective projection** becomes a matrix product
  - A chain of transformations becomes a **single matrix**

August Ferdinand Möbius (1790–1868)



Möbius strip



Converting To and From Homogeneous Coordinates

	To homogeneous: append a 1	From homogeneous: divide by the last coordinate
2D (image)	$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$	$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow \left(\frac{x}{w}, \frac{y}{w}\right)$
3D (scene)	$(x, y, z) \Rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$	$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \Rightarrow \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right)$

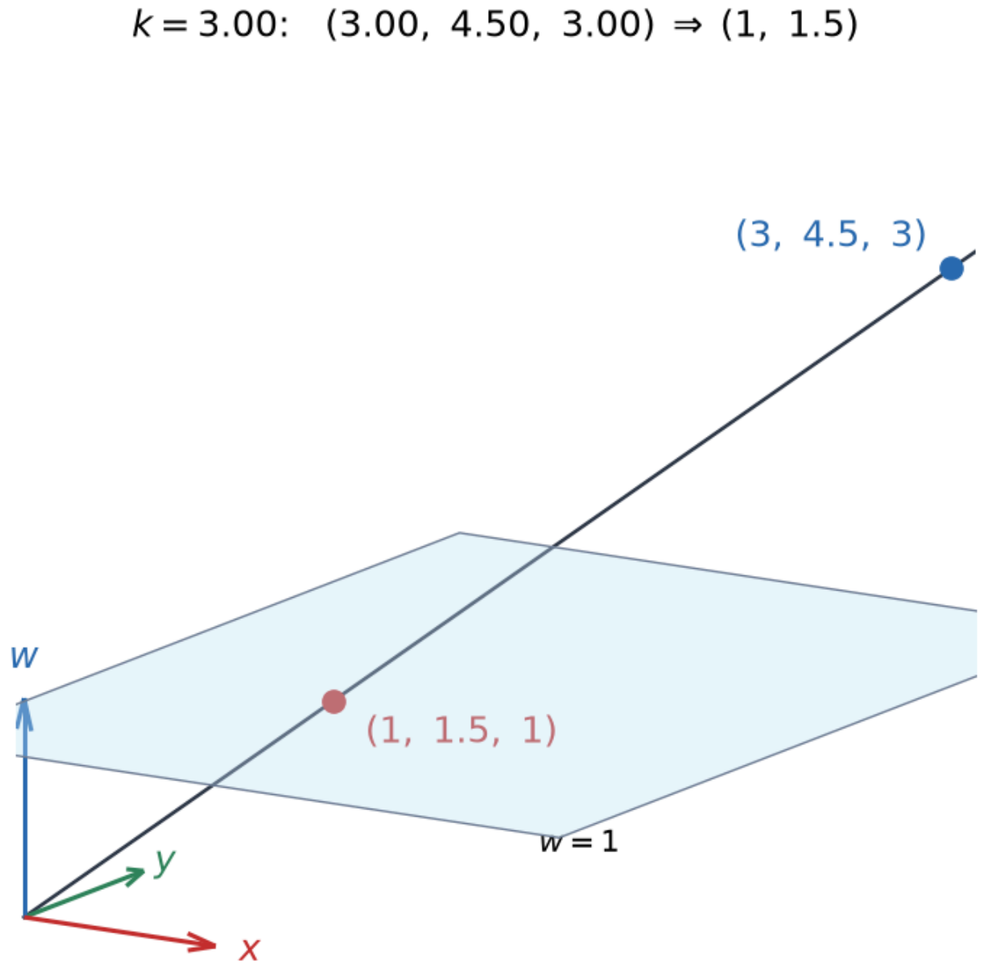
- Example: $(2, 3) \Rightarrow (2, 3, 1) \sim (4, 6, 2) \Rightarrow (2, 3)$: **scaling a homogeneous vector does not change the point** it represents.
- $w = 0$ has no Cartesian equivalent: it is a **point at infinity** (a direction), such as the vanishing point where parallel lines meet.

A Point Is a Ray

- All multiples of a homogeneous vector are the **same point**:

$$k \begin{bmatrix} x \\ y \\ w \end{bmatrix} \sim \begin{bmatrix} x \\ y \\ w \end{bmatrix}, \quad k \neq 0$$

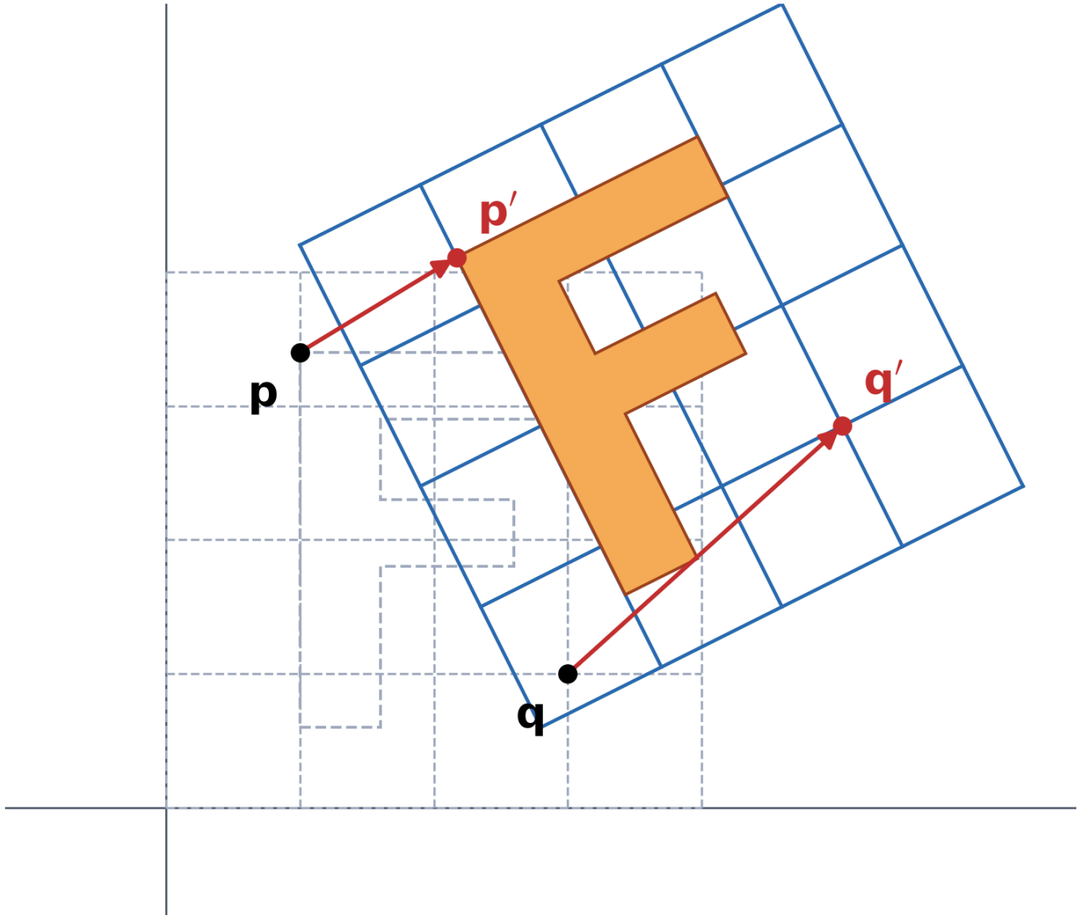
- So a 2D point is a **ray through the origin**; its Cartesian coordinates are where the ray crosses the plane $w = 1$.
- This is the pinhole camera: every 3D point on a ray through the camera center lands on the same image point (scale ambiguity).



Parametric (Global) Transformations

- A **transformation** T maps every point $\mathbf{p} = (x, y)$ to a new point $\mathbf{p}' = T(\mathbf{p})$.
- **Parametric:** T is described by a few numbers (its parameters).
- **Global:** the same T applies to every point of the image.
- If T is **linear**, it is a **matrix**: $\mathbf{p}' = \mathbf{M}\,\mathbf{p}$,

$$\begin{bmatrix} p'_x \\ p'_y \end{bmatrix} = \underbrace{\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix}}_{\mathbf{M}} \begin{bmatrix} p_x \\ p_y \end{bmatrix}.$$



Translation as a Matrix Product

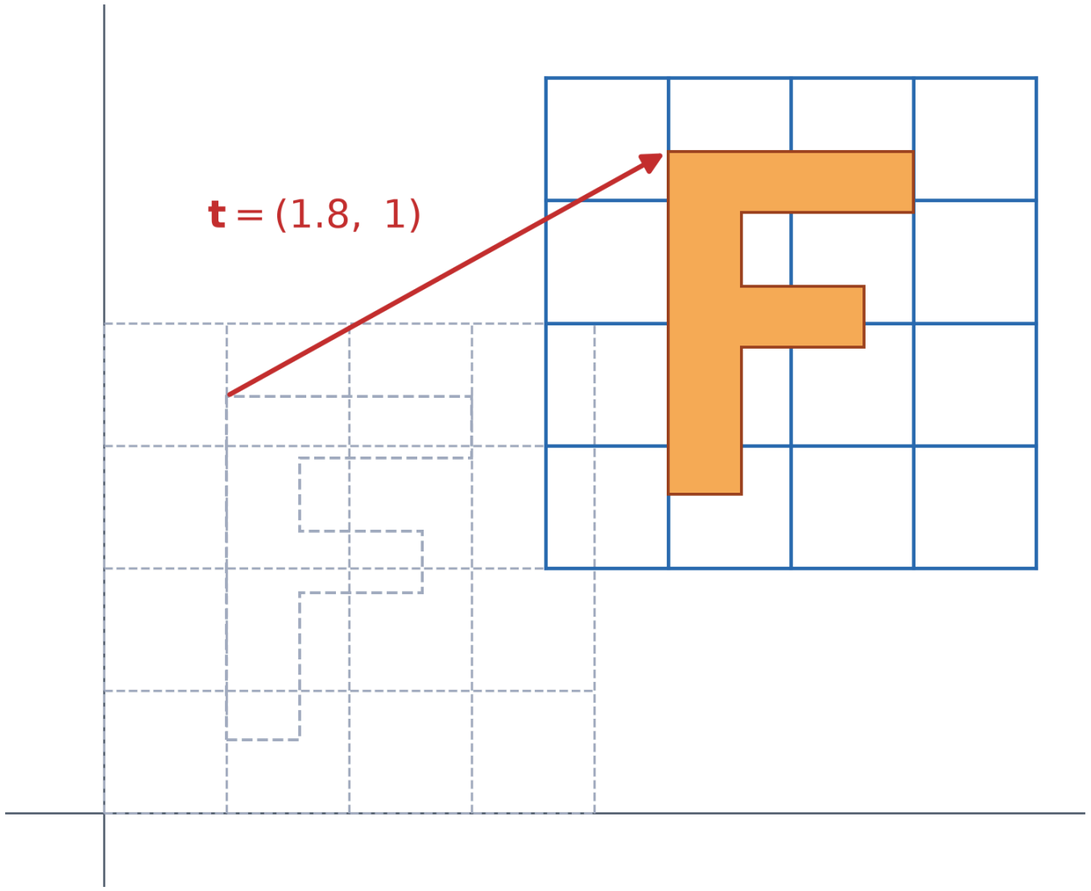
- In Cartesian coordinates, translation is **not** a matrix product:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

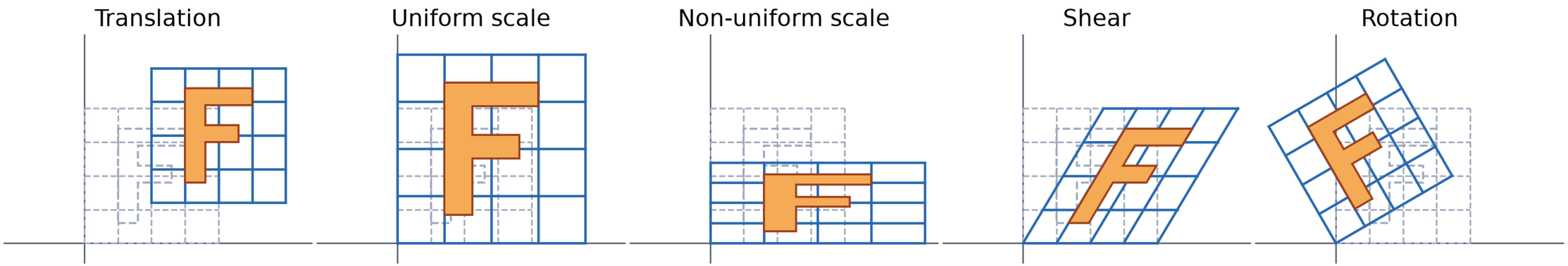
- In homogeneous coordinates, it is:

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

- Rotation, scaling, translation, and projection all become matrices, so a sequence of them is one matrix product.



2D Transformation Matrices



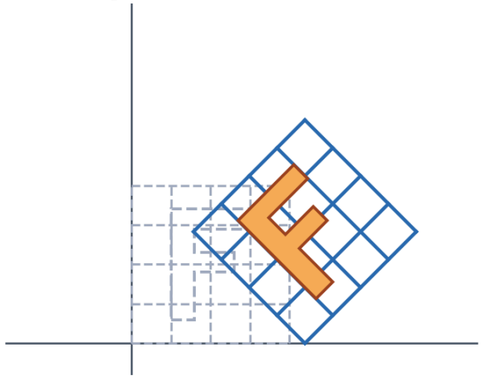
	Translation	Uniform scale	Non-uniform scale	Shear	Rotation
Cartesian	none: $\mathbf{p}' = \mathbf{p} + \mathbf{t}$	$\begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix}$	$\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$	$\begin{bmatrix} 1 & a_x \\ a_y & 1 \end{bmatrix}$	$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$
Homogeneous	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & a_x & 0 \\ a_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

- Scale, shear, and rotation are **linear**: they keep the origin fixed. Translation is not, so it has no 2×2 matrix.
- In homogeneous coordinates, **every** one of them is a 3×3 matrix.

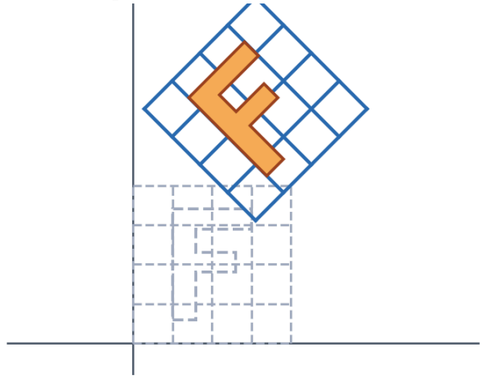
Composing Transformations

$$\mathbf{R} = \begin{bmatrix} \cos 45^\circ & -\sin 45^\circ & 0 \\ \sin 45^\circ & \cos 45^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{T} = \begin{bmatrix} 1 & 0 & 2.2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$\mathbf{p}' = \mathbf{T}\,\mathbf{R}\,\mathbf{p}$: rotate, then translate



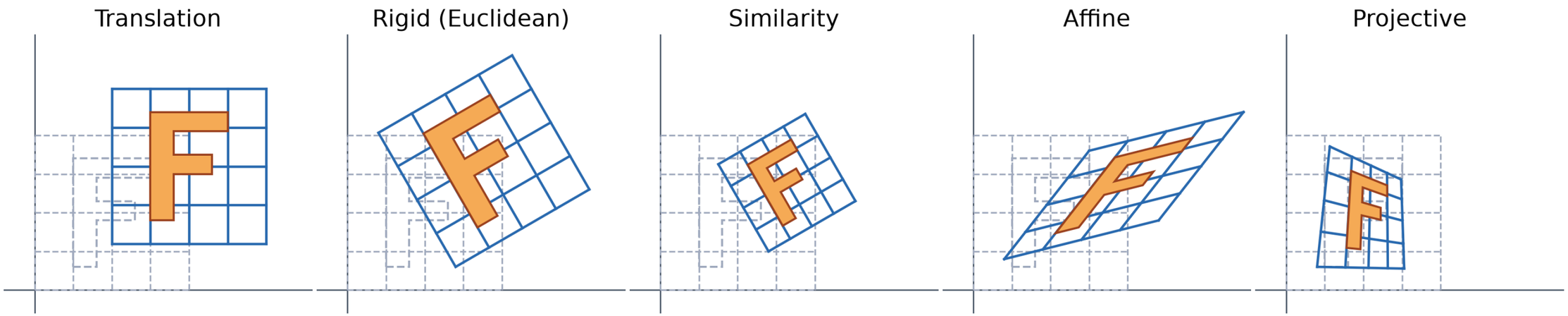
$\mathbf{p}' = \mathbf{R}\,\mathbf{T}\,\mathbf{p}$: translate, then rotate



$$\mathbf{T}\,\mathbf{R} = \begin{bmatrix} 0.71 & -0.71 & 2.2 \\ 0.71 & 0.71 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{R}\,\mathbf{T} = \begin{bmatrix} 0.71 & -0.71 & 1.56 \\ 0.71 & 0.71 & 1.56 \\ 0 & 0 & 1 \end{bmatrix}$$

- A chain of transformations is **one matrix**, and the rightmost is applied first. **Order matters:** $\mathbf{T}\,\mathbf{R} \neq \mathbf{R}\,\mathbf{T}$.

The 2D Transformation Hierarchy



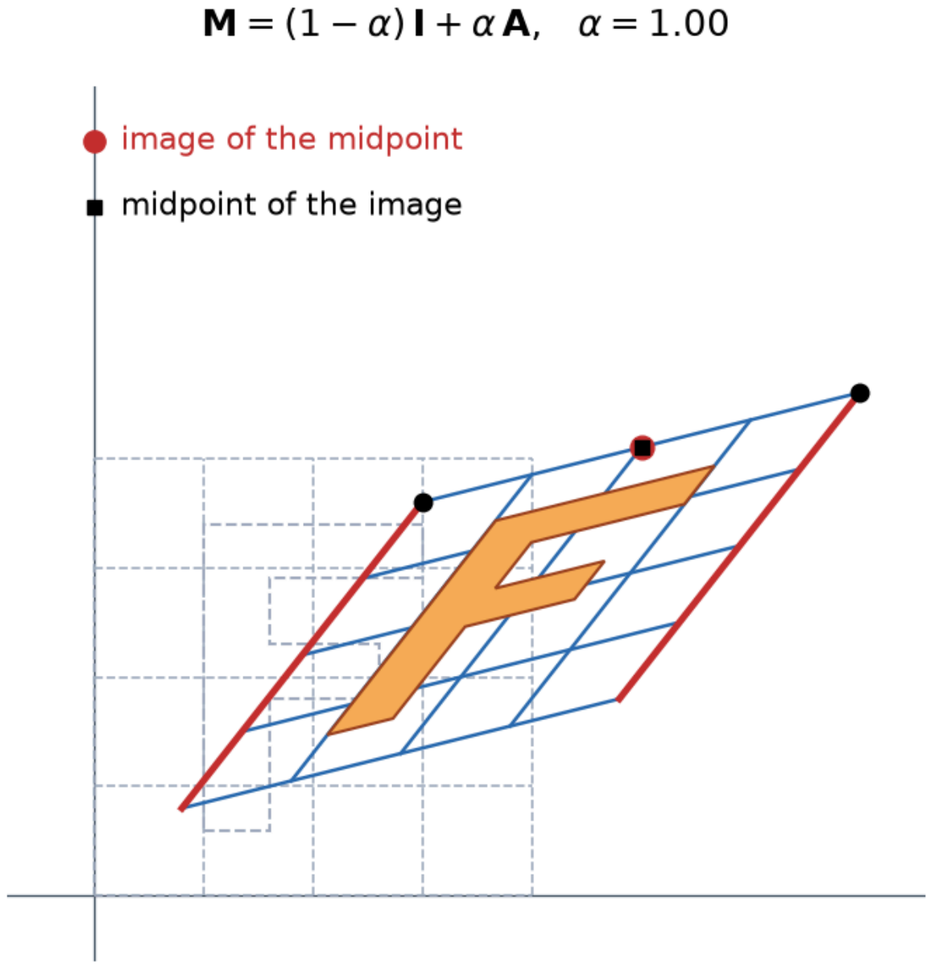
Transformation	Matrix	DOF	Preserves (and everything below)
Translation	$\begin{bmatrix} \mathbf{I} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$	2	orientation
Rigid (Euclidean)	$\begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$	3	lengths
Similarity	$\begin{bmatrix} s\mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$	4	angles
Affine	$\begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$	6	parallelism
Projective	$\mathbf{H}$ (any invertible 3×3)	8	straight lines

Affine Transformations

- Any combination of linear transformations and translation:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- 6 DOF; the last row is always $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$.
- Properties:
  - Lines map to lines
  - **Parallel lines stay parallel**
  - **Ratios** along a line are kept (a midpoint stays a midpoint)
  - Closed under composition

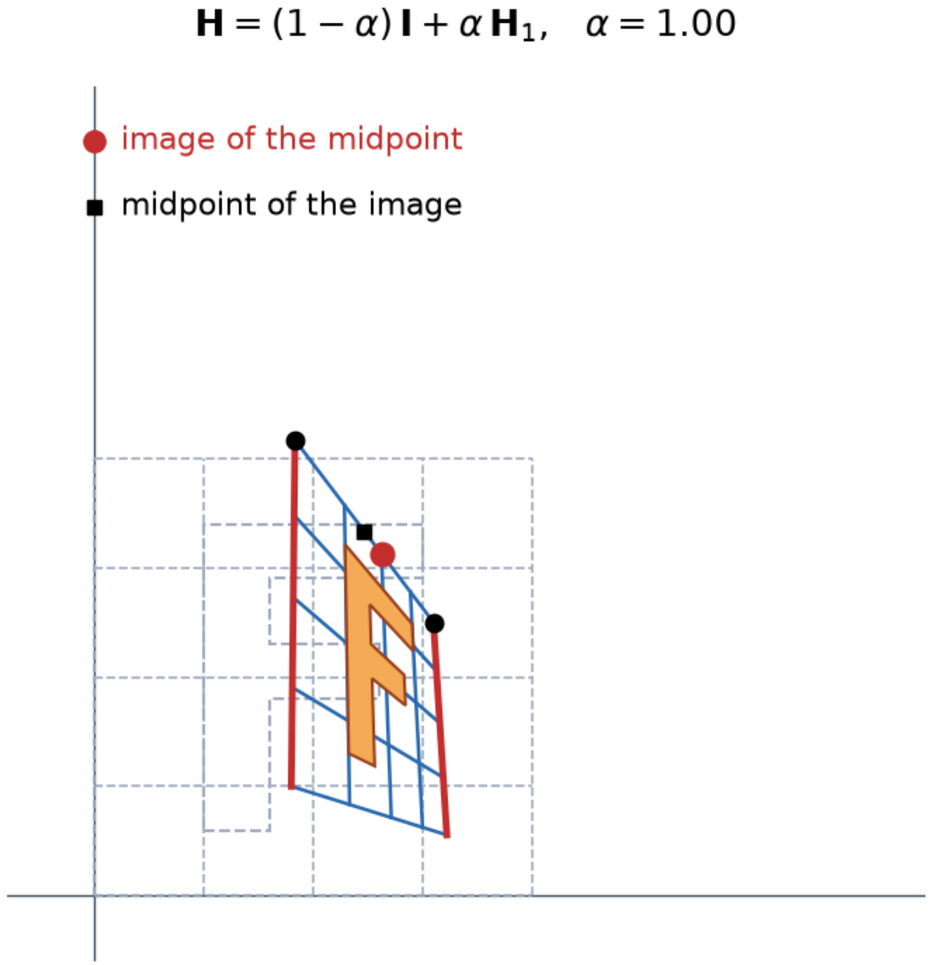


Projective Transformations (Homographies)

- A general 3×3 matrix in homogeneous coordinates:

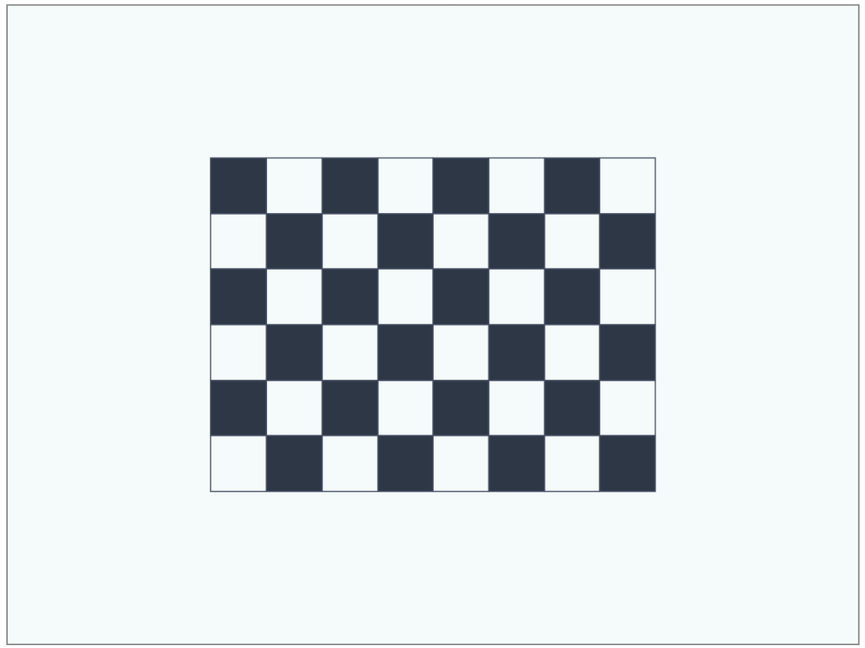
$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Divide by w' to get the point: a **projective warp**.
- 8 DOF: $\mathbf{H}$ is defined only up to scale.
- Properties:
  - Lines map to lines
  - Parallel lines **do not stay parallel**
  - Ratios are **not** kept
  - Closed under composition

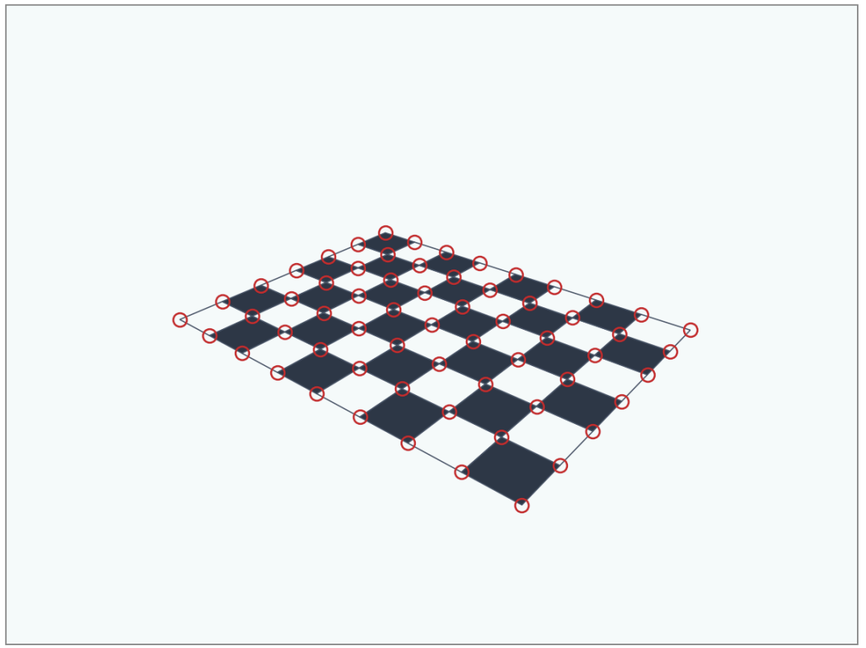


Cameras Are Projective Transformation Machines

View 1: facing the plane



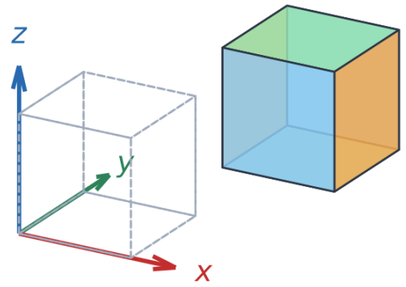
View 2: from the side



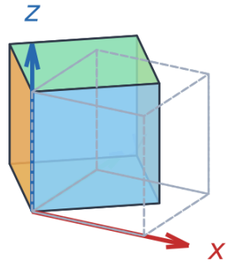
- A camera maps a **plane** in the world to the image by a homography, so two views of the same plane are related by a homography $\mathbf{H}$.
- This works only because the scene is **flat**: for a 3D scene, points at different depths shift by different amounts between views, so no single $\mathbf{H}$ relates them.

3D Rotation and Translation

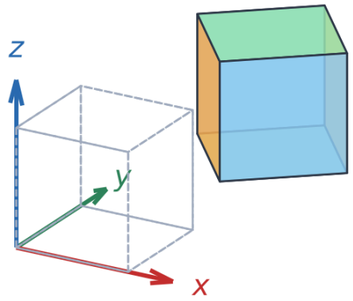
Translation



Rotation about z



Rotation, then translation



	Translation	Rotation	Rotation + translation (rigid)
Cartesian	none: $\mathbf{X}' = \mathbf{X} + \mathbf{t}$	$\mathbf{X}' = \mathbf{R}\,\mathbf{X}$	$\mathbf{X}' = \mathbf{R}\,\mathbf{X} + \mathbf{t}$
Homogeneous (4×4)	$\begin{bmatrix} \mathbf{I} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$	$\begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix}$	$\begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$

- $\mathbf{R}$ is a 3×3 rotation; e.g., about z it is the 2D rotation with a 1 added: $\mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$
- Every 3D rotation has $\mathbf{R}^T\mathbf{R} = \mathbf{I}$, $\det \mathbf{R} = 1$, and **3 DOF** (e.g., $\mathbf{R} = \mathbf{R}_z\mathbf{R}_y\mathbf{R}_x$). A rigid transformation has **6 DOF** $(3 + 3)$.

Changing Coordinate Frames

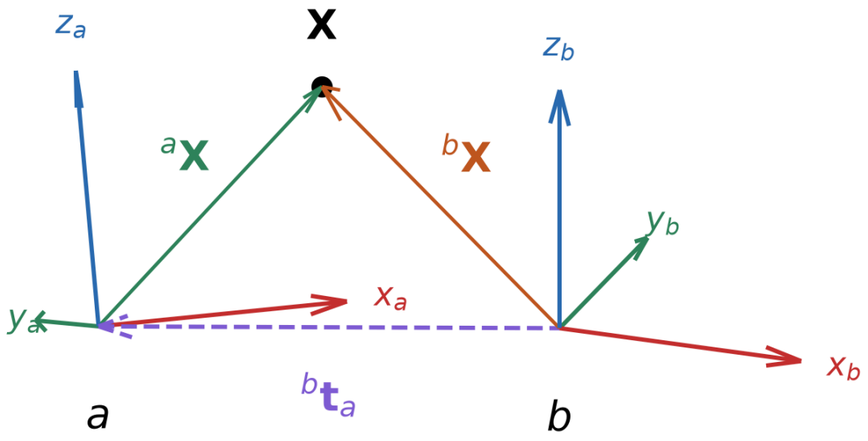
- The same $\left[\, \mathbf{R} \mid \mathbf{t} \,\right]$ can **move a point** within one frame, or **change frames** (i.e., give the coordinates of one fixed point in another frame):

$$ {}^{b}\mathbf{X} = {}^{b}\mathbf{R}_{a}\ {}^{a}\mathbf{X} + {}^{b}\mathbf{t}_{a} \qquad {}^{b}\mathbf{T}_{a} = \begin{bmatrix} {}^{b}\mathbf{R}_{a} & {}^{b}\mathbf{t}_{a} \\ \mathbf{0}^{T} & 1 \end{bmatrix} $$

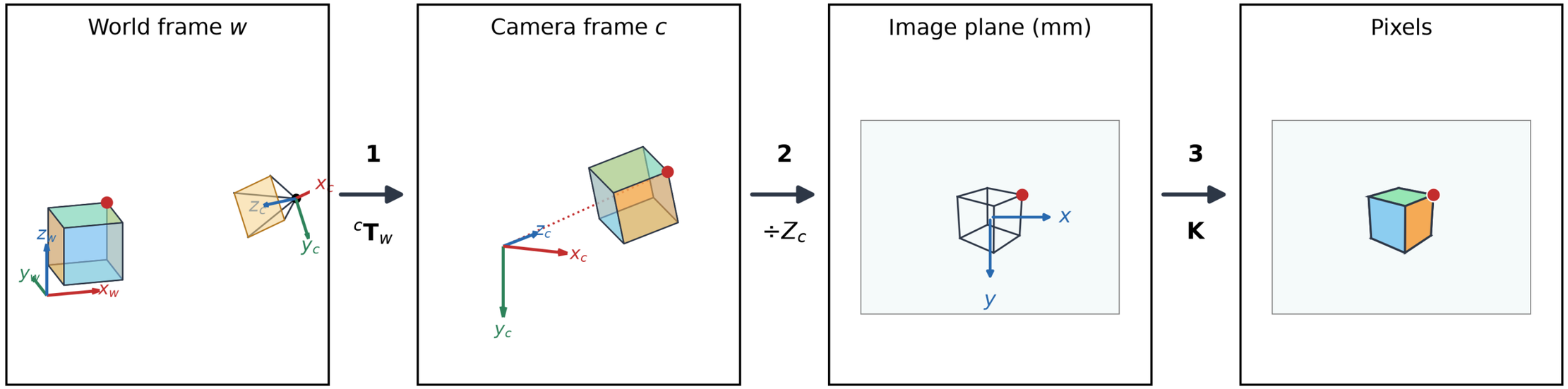
  - Columns of ${}^{b}\mathbf{R}_{a}$: the axes of a in frame b.
  - ${}^{b}\mathbf{t}_{a}$: the origin of a in frame b.
- **Inverse:**

$$ {}^{a}\mathbf{T}_{b} = \left({}^{b}\mathbf{T}_{a}\right)^{-1} = \begin{bmatrix} {}^{b}\mathbf{R}_{a}^{T} & -{}^{b}\mathbf{R}_{a}^{T}\,{}^{b}\mathbf{t}_{a} \\ \mathbf{0}^{T} & 1 \end{bmatrix} $$

- **Composition:** ${}^{c}\mathbf{T}_{a} = {}^{c}\mathbf{T}_{b}\ {}^{b}\mathbf{T}_{a}$ (matching inner frames cancel).



Camera Model: From a World Point to a Pixel



${}^{w}\mathbf{X} = (1.4,\ 1.4,\ 1.0)$ m ${}^{c}\mathbf{X} = (0.70,\ -0.49,\ 4.47)$ m $(x, y) = (0.393,\ -0.274)$ mm $(u, v) = (398.7,\ 185.1)$ px

1. Extrinsics	2. Perspective projection	3. Intrinsics
world frame $\rightarrow$ camera frame	camera frame $\rightarrow$ image plane	image plane $\rightarrow$ pixels

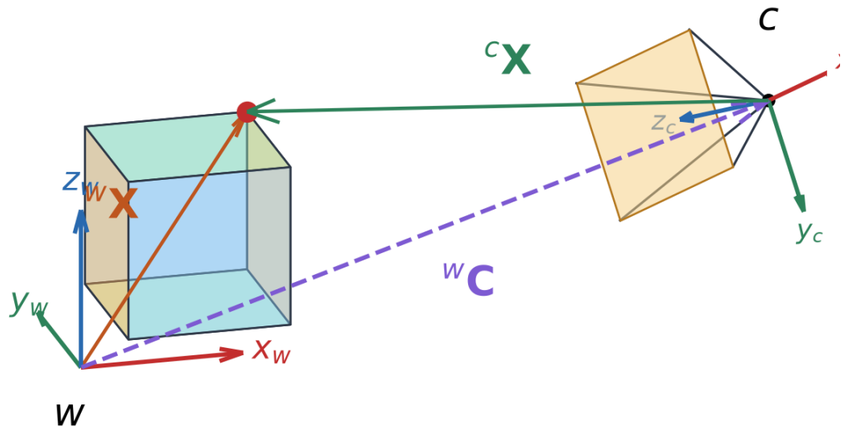
Each step is a matrix in homogeneous coordinates, so **the whole camera model is one matrix**.

Step 1 — World to Camera (Extrinsics)

- Points are given in a **world** frame w; projection needs them in the **camera** frame c:

$$ {}^{c}\mathbf{X} = {}^{c}\mathbf{R}_{w}\ {}^{w}\mathbf{X} + {}^{c}\mathbf{t}_{w} \qquad {}^{c}\mathbf{T}_{w} = \begin{bmatrix} {}^{c}\mathbf{R}_{w} & {}^{c}\mathbf{t}_{w} \\ \mathbf{0}^{T} & 1 \end{bmatrix} $$

- The **extrinsic matrix** is the top three rows of ${}^{c}\mathbf{T}_{w}$: $\left[\, {}^{c}\mathbf{R}_{w} \mid {}^{c}\mathbf{t}_{w} \,\right]$ (3×4).
- The camera center ${}^{w}\mathbf{C}$ maps to ${}^{c}\mathbf{X} = \mathbf{0}$, so ${}^{c}\mathbf{t}_{w} = -\,{}^{c}\mathbf{R}_{w}\ {}^{w}\mathbf{C}$.
- **6 DOF:** camera position is 3 and its orientation is 3.



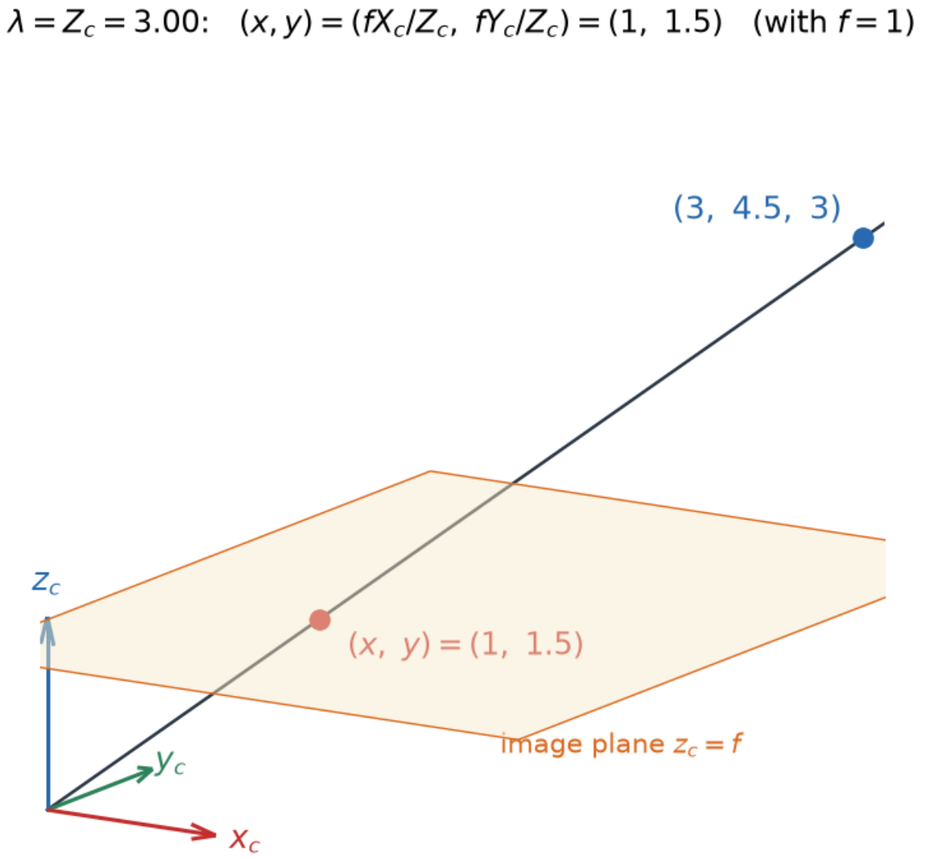
- Example: ${}^{w}\mathbf{C} = (3.6,\ -2.4,\ 2.2)$, and the red corner ${}^{w}\mathbf{X} = (1.4,\ 1.4,\ 1.0)$ becomes ${}^{c}\mathbf{X} = (0.70,\ -0.49,\ 4.47)$.

Step 2 — Camera to Image Plane (Projection)

- With ${}^c\mathbf{X} = (X_c, Y_c, Z_c)$, perspective projection $x = fX_c/Z_c,\ y = fY_c/Z_c$ is a matrix product in homogeneous coordinates:

$$Z_c \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} fX_c \\ fY_c \\ Z_c \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}$$

- The homogeneous scale is the **depth** $\lambda = Z_c$: we divide by it only at the end.
- All points on a ray through the camera center give the **same** (x, y): depth is lost.



- Example ($f = 2.5$ mm): ${}^c\mathbf{X} = (0.70,\ -0.49,\ 4.47)$ m $\Rightarrow (x, y) = (0.393,\ -0.274)$ mm, with $\lambda = 4.47$.

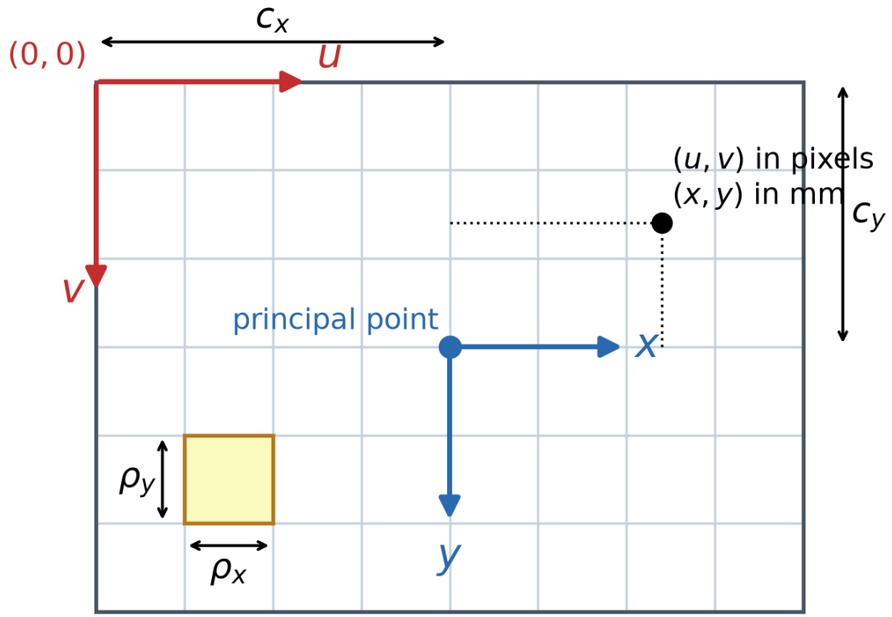
Step 3 — Image Plane to Pixels (Intrinsics)

- (x, y) are in mm, with the origin at the **principal point**.
- Pixel coordinates (u, v) start at the **top-left** pixel.
- With pixel size $\rho_x \times \rho_y$ (mm) and the principal point at pixel (c_x, c_y):

$$u = \frac{x}{\rho_x} + c_x, \qquad v = \frac{y}{\rho_y} + c_y$$

- As a matrix in homogeneous coordinates:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\rho_x} & 0 & c_x \\ 0 & \frac{1}{\rho_y} & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



The Intrinsic Matrix $\mathbf{K}$

- Steps 2 and 3 combine into one matrix: projection with f, then conversion to pixels:

$$\underbrace{\begin{bmatrix} \frac{1}{\rho_x} & 0 & c_x \\ 0 & \frac{1}{\rho_y} & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{\text{to pixels}} \underbrace{\begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{focal length}} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

- $f_x = f/\rho_x$, $f_y = f/\rho_y$: the **focal length in pixels** (e.g., 2.5 mm / $5\,\mu$m $= 500$ px).
- Adding a **skew** s gives the general **intrinsic matrix**:

$$\mathbf{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

- f_x, f_y: **focal length** in pixels, i.e., *zoom*; $f_x \neq f_y$ if pixels are not square.
- c_x, c_y: **principal point** in pixels, usually near the image center.
- s: **skew**; $s = 0$ for almost all modern cameras.

The Camera Matrix $\mathbf{P}$

Chaining the three steps, with $\mathbf{x} = (u, v, 1)$ in pixels and $\lambda = Z_c$:

$$\lambda\,\mathbf{x} = \underbrace{\mathbf{K}}_{3\times3}\,\underbrace{\begin{bmatrix}\mathbf{I} & \mathbf{0}\end{bmatrix}}_{3\times4}\,\underbrace{{}^{c}\mathbf{T}_{w}}_{4\times4}\,\underbrace{{}^{w}\mathbf{X}}_{4\times1}$$

$$= \underbrace{\mathbf{K}}_{\substack{\text{intrinsic}\\\text{matrix}}}\;\underbrace{\begin{bmatrix}{}^{c}\mathbf{R}_{w} \mid {}^{c}\mathbf{t}_{w}\end{bmatrix}}_{\substack{\text{extrinsic}\\\text{matrix}}}\;{}^{w}\mathbf{X} = \mathbf{P}\;{}^{w}\mathbf{X}$$

Part	Parameters	DOF
Intrinsic matrix $\mathbf{K}$	$f_x,\ f_y,\ c_x,\ c_y,\ s$	5
Extrinsic matrix $\begin{bmatrix}{}^{c}\mathbf{R}_{w} \mid {}^{c}\mathbf{t}_{w}\end{bmatrix}$	rotation (3) + translation (3)	6
Camera matrix $\mathbf{P}$		**11**

- $\mathbf{P}$ has 12 entries, but scaling it does not change (u, v): **11 DOF**.

The Camera Matrix: Cube Example

For the camera and the red corner in the cube example:

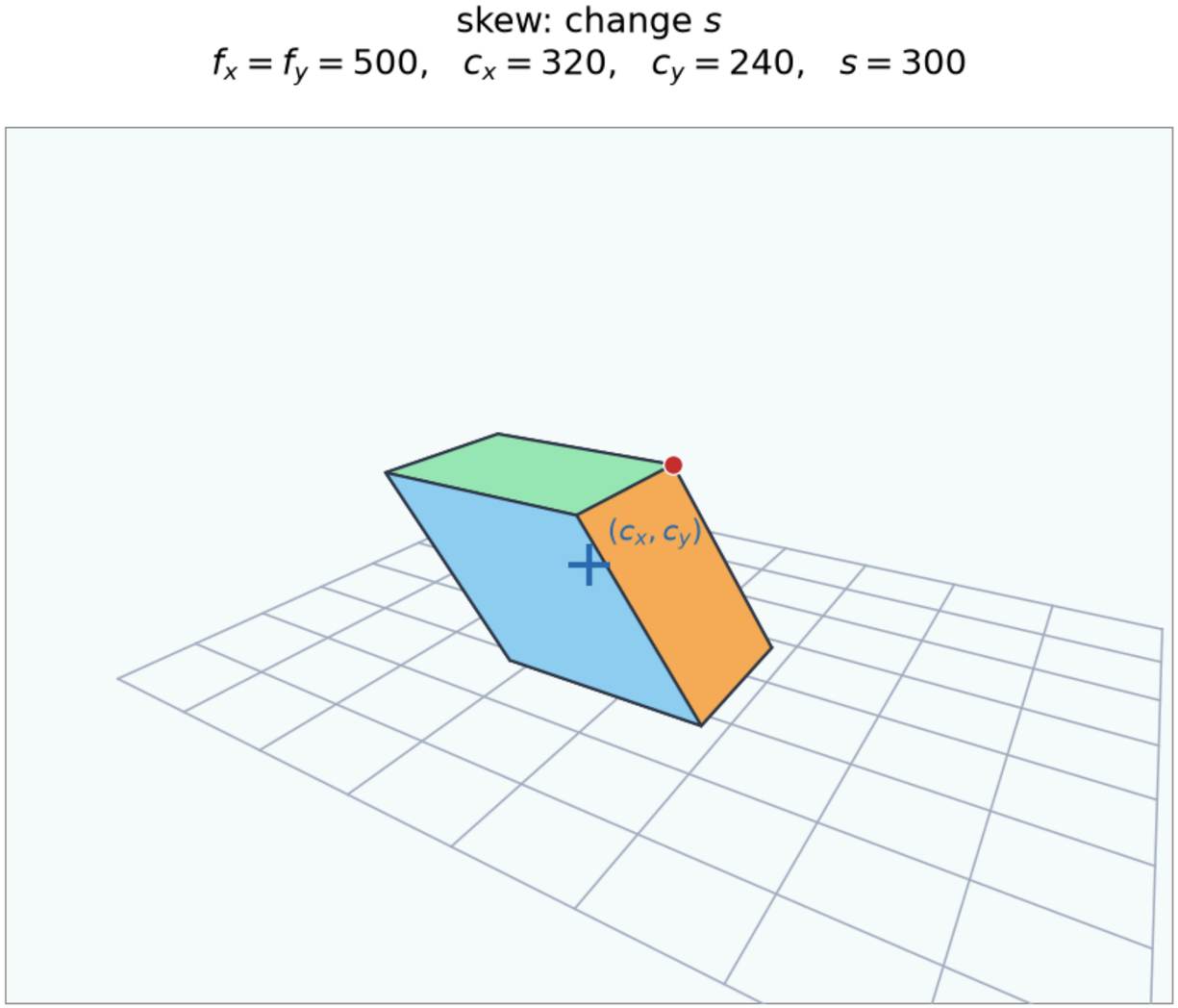
$$\underbrace{\begin{bmatrix} 500 & 0 & 320 \\ 0 & 500 & 240 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{K}\text{: intrinsic matrix}} \qquad \underbrace{\begin{bmatrix} 0.77 & 0.63 & 0 & -1.27 \\ 0.23 & -0.29 & -0.93 & 0.51 \\ -0.59 & 0.72 & -0.37 & 4.66 \end{bmatrix}}_{\left[\, {}^{c}\mathbf{R}_{w} | {}^{c}\mathbf{t}_{w} \,\right]\text{: extrinsic matrix}}$$

$$\mathbf{P} = \mathbf{K} \left[\, {}^{c}\mathbf{R}_{w} \mid {}^{c}\mathbf{t}_{w} \,\right] = \begin{bmatrix} 198.75 & 546.67 & -118.51 & 857.24 \\ -23.91 & 29.22 & -553.33 & 1373.53 \\ -0.59 & 0.72 & -0.37 & 4.66 \end{bmatrix}$$

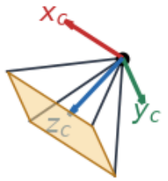
$$\lambda\, \mathbf{x} = \mathbf{P}\ {}^{w}\mathbf{X} = \mathbf{P}\,(1.4,\ 1.4,\ 1.0,\ 1.0)^{T} = (1782.3,\ 827.6,\ 4.5) \ \Rightarrow\ (u, v) = (398.7,\ 185.1)$$

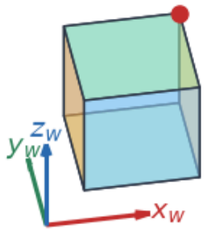
Intrinsics in Action

- f_x, f_y: **focal length** in pixels, i.e., *zoom*; $f_x \neq f_y$ if pixels are not square.
- c_x, c_y: **principal point** in pixels, usually near the image center.
- s: **skew**; $s = 0$ for almost all modern cameras.
- **5 DOF.** The animation changes one parameter at a time, with the camera fixed.

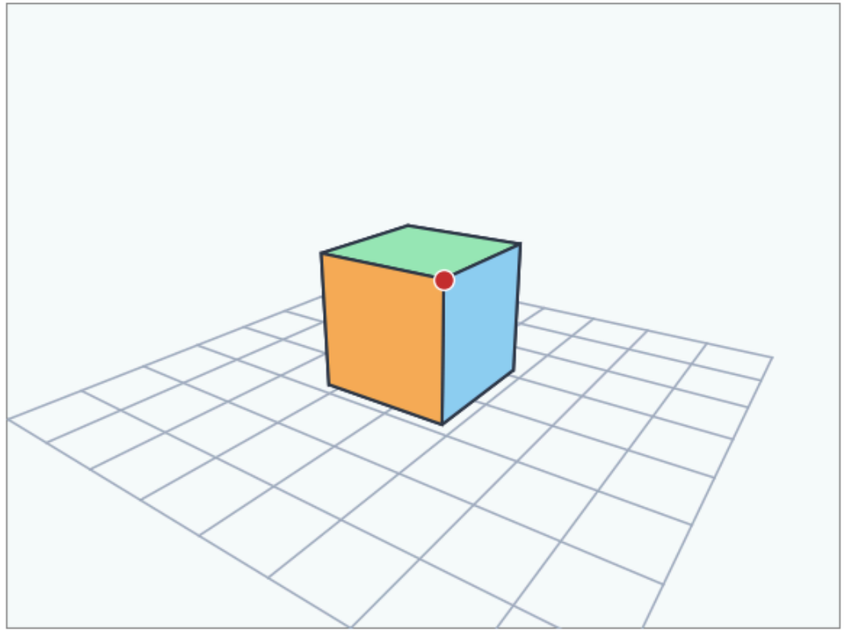


Extrinsics in Action





Camera orbits: ${}^c\mathbf{R}_w$ and ${}^c\mathbf{t}_w$ change



extrinsic matrix

${}^c\mathbf{R}_w$

```
-0.57   0.82   0.00
 0.29   0.21  -0.93
-0.76  -0.54  -0.36
```

${}^c\mathbf{t}_w$

```
-0.22   0.02   6.09
```

${}^c\mathbf{X}$ (red corner)

```
 0.12  -0.22   3.92
```

red corner:

```
(u, v) = (336, 212)
```

- $\mathbf{K}$ is fixed: every change in the image comes from the **extrinsic matrix** $\left[{}^c\mathbf{R}_w \mid {}^c\mathbf{t}_w \right]$, shown on the right.

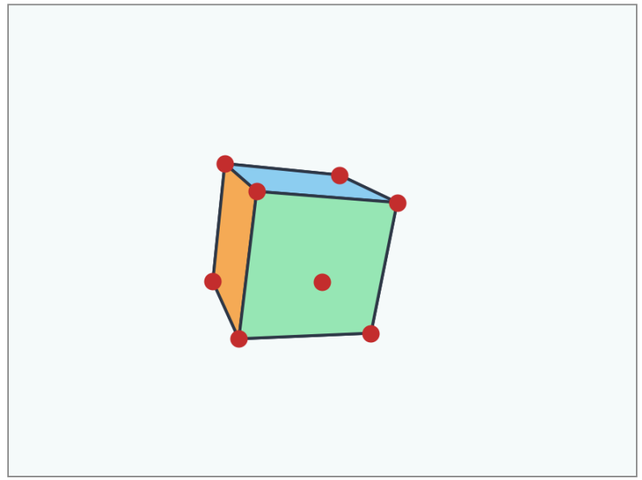
Projecting Points by Hand

```python
K = np.array([[500.0, 0.0, 320.0], [0.0, 500.0, 240.0], [0.0, 0.0, 1.0]]) # intrinsics
rvec = np.array([0.3, -0.4, 0.1])
R_c_w, _ = cv2.Rodrigues(rvec) # extrinsics rotation
t_c_w = np.array([[-0.4], [-0.3], [3.0]]) # extrinsics translation
P = K @ np.hstack([R_c_w, t_c_w]) # camera matrix

X_w = np.array([[x, y, z] for x in (0, 1) for y in (0, 1) for z in (0, 1)], dtype=float) # cub
X_w_h = np.hstack([X_w, np.ones((len(X_w), 1))]) # homogeneous coordinates
x = (P @ X_w_h.T).T # projection (lambda * u, lambda * v, lambda)
uv = x[:, :2] / x[:, 2:] # pixel coordinates (u, v)
```

- `rvec`: the rotation as a 3-vector (axis $\times$ angle); `cv2.Rodrigues` turns it into ${}^{c}\mathbf{R}_{w}$.
- `P` $= \mathbf{K}\left[{}^{c}\mathbf{R}_{w} \mid {}^{c}\mathbf{t}_{w}\right]$.
- `X_w_h` holds the corners as homogeneous rows.
- Each row of `x` is $\lambda\,(u, v, 1)$ with $\lambda = Z_c$; dividing by it gives `uv`.

Projected cube corners

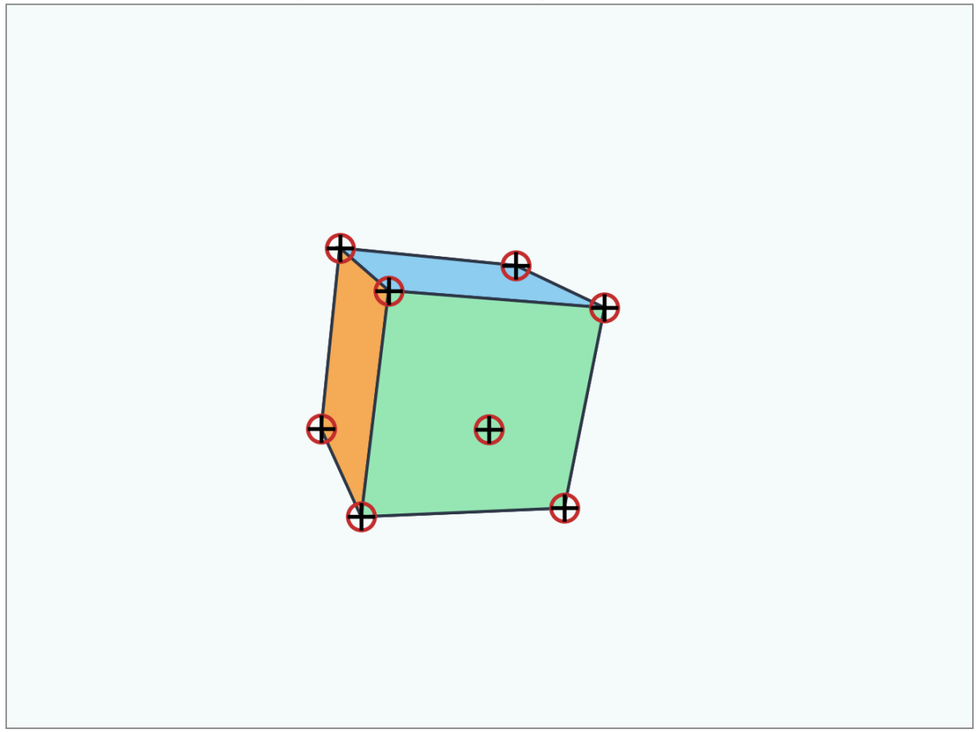


Projecting Points with OpenCV

```python
# Same rvec, t_c_w, K, and X_w as on the previous slide; None means no lens distortion.
uv_cv, _ = cv2.projectPoints(X_w, rvec, t_c_w, K, None)
uv_cv = uv_cv.reshape(-1, 2)
```

- `projectPoints` takes the rotation as `rvec` and applies $\mathbf{K}\left[{}^{c}\mathbf{R}_{w} \mid {}^{c}\mathbf{t}_{w}\right]$ internally.
- It returns an $(N, 1, 2)$ array; `reshape(-1, 2)` gives $N \times 2$ pixel coordinates.
- The last argument is the **lens distortion** model: `None` here.
  - We will cover it in the camera calibration lecture.

OpenCV (o) vs. by hand (+)



Key Takeaways

- A pinhole forms an image by letting one ray from each scene point reach the sensor. A lens gathers more light; when in focus, its central rays follow the same geometry as the pinhole model.
- Perspective projection divides by depth: $x = fX/Z, y = fY/Z$. Distant objects appear smaller, and a single image point does not determine depth.
- Homogeneous coordinates represent points up to a nonzero scale and let us compose transformations through matrix multiplication. The rightmost transformation acts first; order matters.
- Affine transformations preserve parallelism and ratios along a line. Projective transformations preserve straight lines, but generally change parallelism and ratios.
- Extrinsics transform world coordinates into camera coordinates. Intrinsics determine how viewing directions map to pixels:

$$\underbrace{\lambda}_{\text{depth } Z_c} \underbrace{\mathbf{x}}_{\substack{\text{pixel} \\ (u,\, v,\, 1)}} = \overbrace{\underbrace{\mathbf{K}}_{\substack{\text{intrinsics} \\ 3\times3}} \underbrace{\left[{}^{c}\mathbf{R}_{w} \mid {}^{c}\mathbf{t}_{w} \right]}_{\substack{\text{extrinsics} \\ 3\times4}}}^{\mathbf{P}\text{: camera matrix, } 3\times4} \underbrace{{}^{w}\mathbf{X}}_{\substack{\text{world point} \\ (X,\, Y,\, Z,\, 1)}}$$

Divide by the last coordinate to recover the pixel coordinates (u, v).