Camera Model and Perspective Geometry

Homogeneous Coordinates, Intrinsics and Extrinsic Parameters



Antonio Carta

2026-10-07

University of Pisa

antonio.carta@unipi.it

Motivation and Roadmap

Recap: the Pinhole Camera Model

- **Pinhole camera**: light from a 3D point passes through a single small aperture and forms an inverted image on the sensor plane
- **Projection equations** (simplified setting): $x = f\frac{X}{Z}, y = f\frac{Y}{Z}$

Today: We expand this model to support a general coordinate system. Coordinate transformation will be modeled as matrix multiplications.

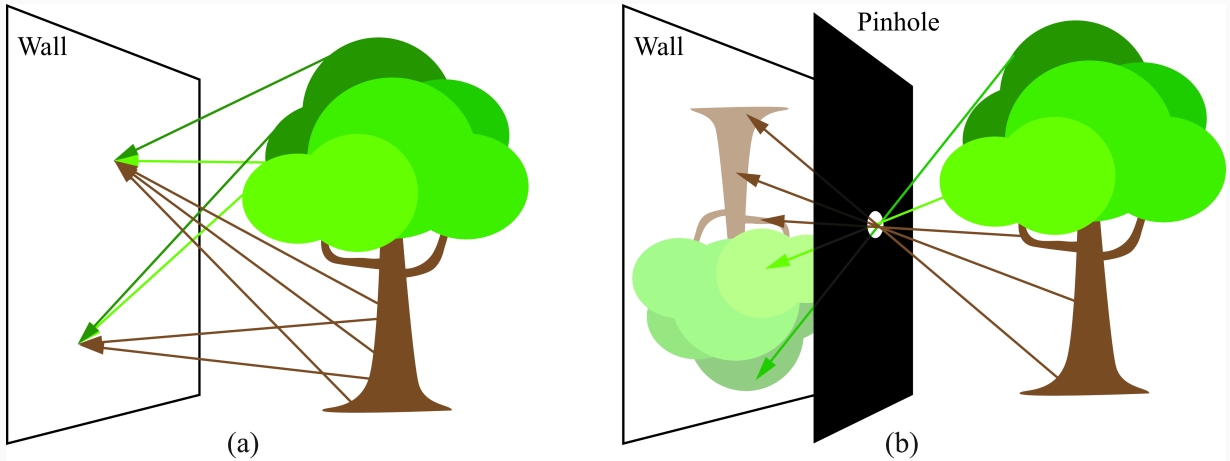


Figure 1: source: visionbook

Why We Need a Projective Model

- **From pixels to point cloud**: Any 3D task like depth prediction or tracking requires **inverting** the projection and going from 2D pixels back to 3D structure
- The natural way to express 2D coordinates (Cartesian plane) is a bit clunky (e.g. projection requires **division** by Z)
  - Nonlinear and awkward to compose and invert
- **Homogeneous coordinates** provide an alternative representation where projection, rotation, translation, and scaling are all **matrix multiplications**
  - A complex camera model can be represented with a single matrix.

From Pixels to Point Cloud

image as pixel grids:

$$\ell = \begin{bmatrix} \ell[1,1] & \dots & \\ \vdots & \ell[n,m] & \vdots \\ & \dots & \ell[N,M] \end{bmatrix}$$

image as point cloud: $\{(\ell_i, x_i, y_i) : i\}$

- l_i intensity at position (x_i, y_i)
- easier to manipulate positions and geometry with this representation
- issue: when we convert from point cloud to pixel grid, points will not necessarily align perfectly with a pixel grid

implicit image representation: $l(x, y) = f_\theta(x, y)$

- we can treat images as functions (i.e. continuous 2D signals)
- f_θ could be a DNN or a function that interpolates to fill the holes in a pixel representation

Roadmap

- **Homogeneous coordinates**: projective geometry and how to represent rigid transformations and points

Camera Model

- **Intrinsics** K: camera-to-pixel mapping (focal length, principal point, skew, aspect ratio)
- **Extrinsics** R, T: world-to-camera rigid transform
- **Camera matrix** $P = K[R \mid -RT]$: full projection from world to pixels

Homogeneous Coordinates

Why Euclidean Coordinates Are Awkward for Projection

Heterogeneous (Cartesian) coordinates:

- note (visionbook notation): uppercase for world coordinates, lowercase for image coordinates. We will write coordinates both as row or column vectors
- (x, y) 2D Cartesian coordinates
- (X, Y, Z) 3D world coordinates

With heterogeneous coordinates, transformations have different "forms"

- Projection $x = f\frac{X}{Z}$ is a ratio. Not expressible as a linear map on (X, Y, Z)
- Translation is an addition $x = X + T_x$ (affine transformation)
- Rotation and scaling are matrix multiplications

Heterogeneous vs Homogeneous in Code

- heterogeneous: `point2d = scale(translate(rotate(point3d, 30), 10), 2)`
  - we need to keep track and be aware of every transformation separately
  - we need to keep track of the order of transformations
  - It's harder to define composition and inversion of arbitrary transformations
- homogeneous: `point2d = map_homogeneous(point3d, Amat)`
  - we can ignore exactly what transformations we have done and their order, because we only need to know the matrix that represents their composition
  - composition = matrix multiplication
  - inversion = matrix inversion

Homogeneous coordinates add one extra dimension

- $[x, y, 1]$ for a 2D point, $[X, Y, Z, 1]$ for a 3D point
- visionbook notation: we use square brackets to distinguish from the heterogeneous coordinates in round brackets
- Note: in heterogeneous coordinates we can sum points, in homogeneous coordinates we cannot

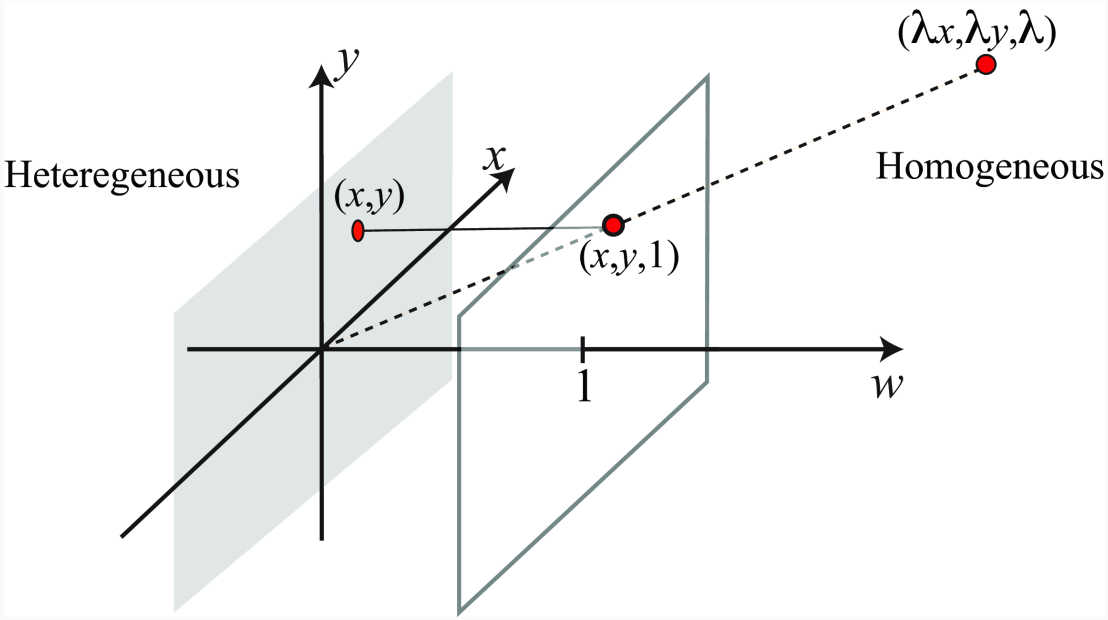


Figure 2: source: visionbook

Scale invariance:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \equiv \begin{bmatrix} \lambda x \\ \lambda y \\ \lambda w \end{bmatrix}$$

for any $\lambda \neq 0$

- a point in heterogeneous coordinates becomes an entire ray through the origin in homogeneous coordinates
- conversion to heterogeneous: $[x, y, w] \rightarrow \left(\frac{x}{w}, \frac{y}{w}\right)$
- This extra degree of freedom is what makes projection **linear**

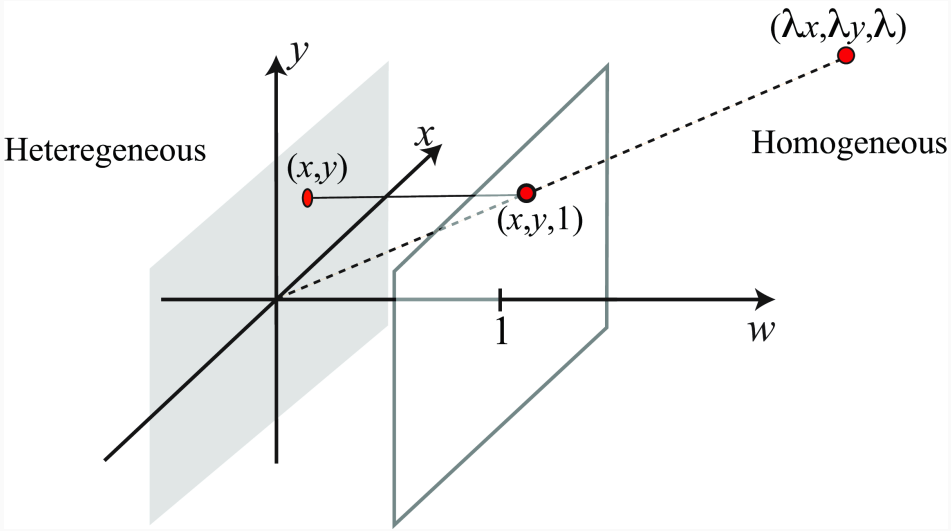


Figure 3: source: visionbook

2D Image Transformations

- we want to model rigid image transformations as matrix multiplication in homogeneous coordinates
- we can obtain more complex transformations by compositions and they are still just a matrix multiplication
- not every transformation is matrix multiplication (e.g. warping, fisheye lens), but they will be sufficient for our purposes

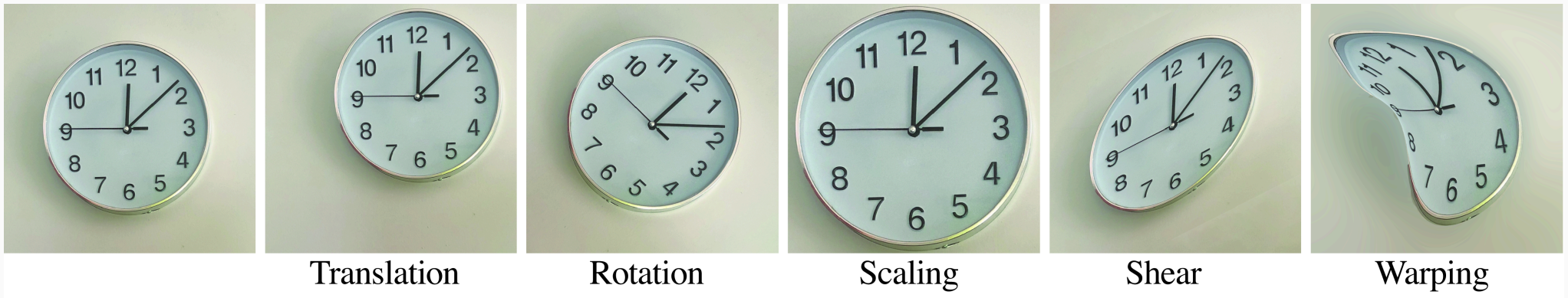


Figure 4: source: visionbook

Translation: Matrix in Euclidean and Homogeneous Coordinates

Euclidean:

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \end{pmatrix}$$

additive, not a linear map

Homogeneous translation matrix by (t_x, t_y):

$$T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Now translation is applied by matrix multiplication: $\boldsymbol{p}' = T\boldsymbol{p}$, consistent with all other transforms

Composition of Translations

- Composing translation by $\left(t_{x_1}, t_{y_1}\right)$ then $\left(t_{x_2}, t_{y_2}\right)$: $T_2 T_1$
- Matrix product of two translation matrices is again a translation matrix, by $\left(t_{x_1} + t_{x_2}, t_{y_1} + t_{y_2}\right)$
- General principle: composing 2D transformations = multiplying their matrices
- in heterogeneous coordinates we add the two translation vectors

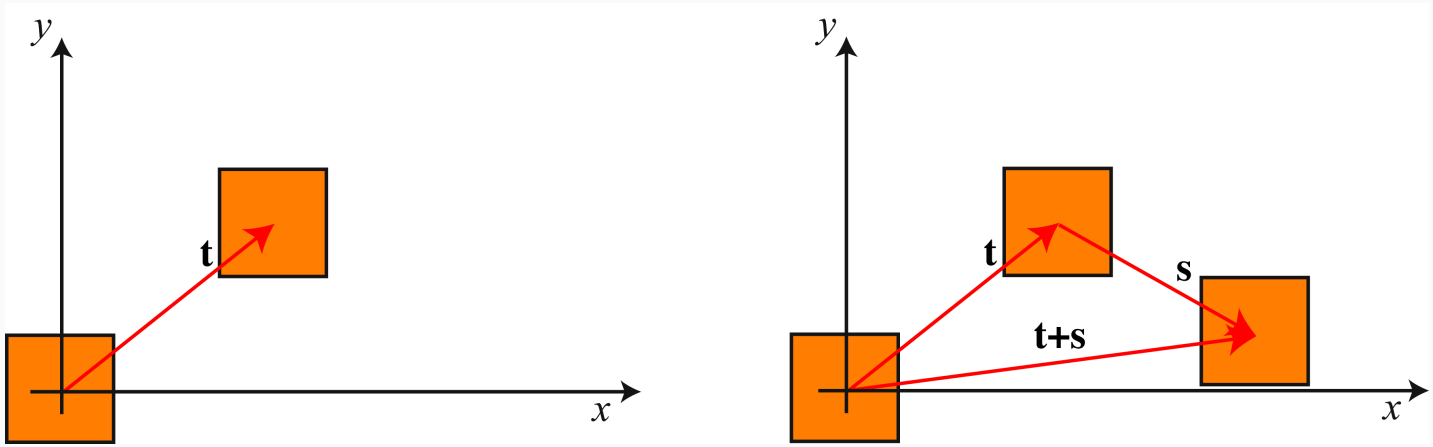


Figure 5: source: visionbook

Scaling

- **Scaling** by (s_x, s_y) about the origin:

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Stretches/compresses coordinates relative to the origin
- note: the top-left 2x2 matrix is the scaling matrix in heterogeneous coordinates

Scaling by (s_x, s_y) about the origin:

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Uniform scaling ($s_x = s_y$) preserves angles; non-uniform (anisotropic) scaling does not

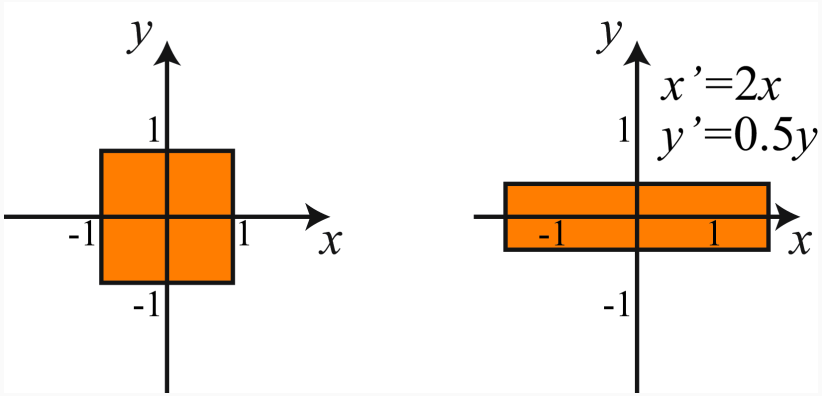


Figure 6: Uniform vs non-uniform scaling. Source: visionbook

Rotation

- Counterclockwise rotation by angle θ about the origin:

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

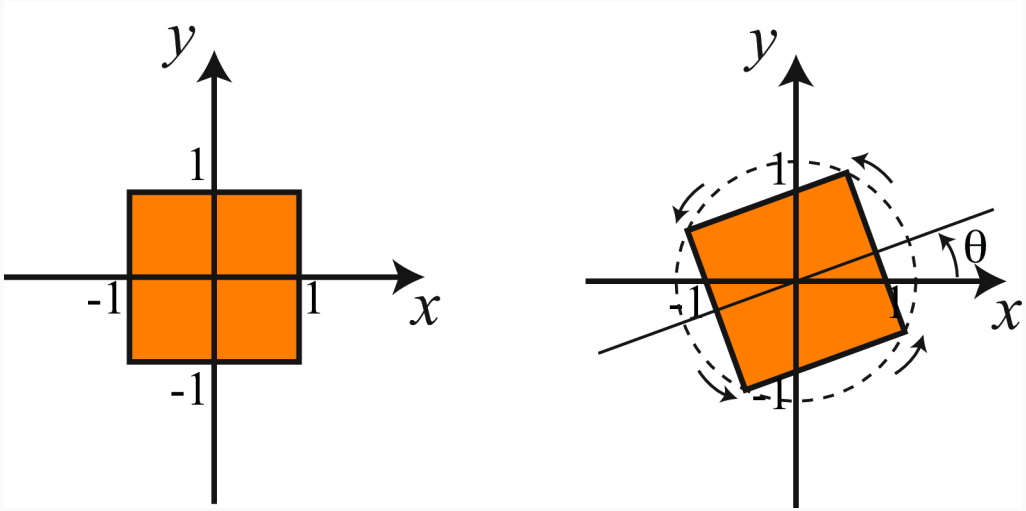
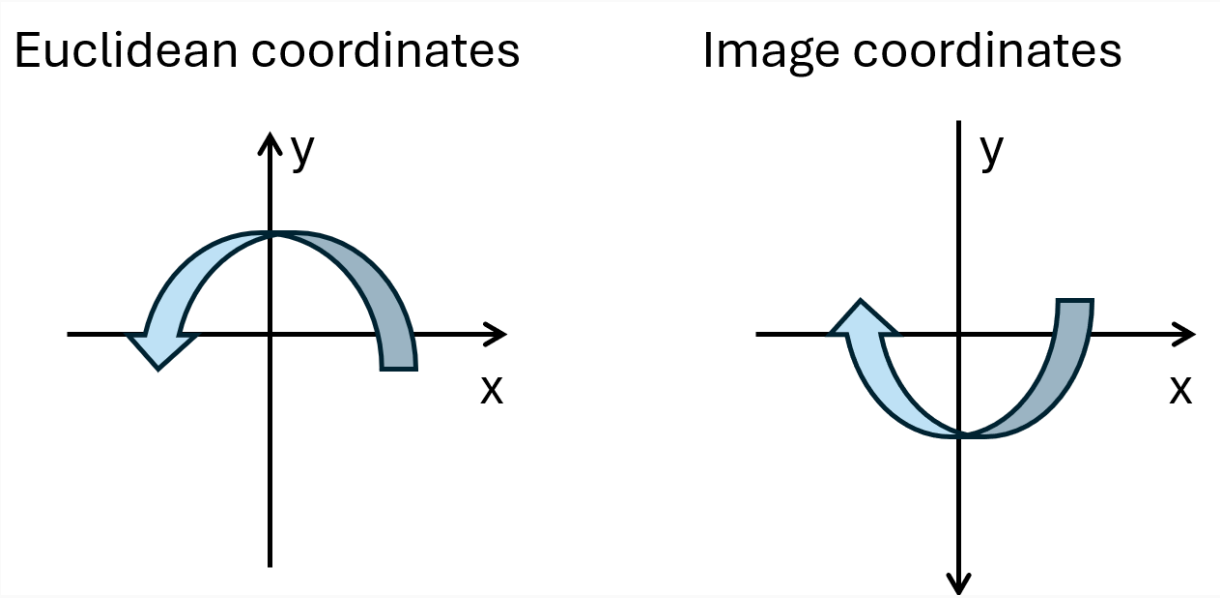


Figure 7: source: visionbook

Rotation: Euclidean vs Image Coordinates

note: the book is a bit confusing here.

- the rotation matrix shown in the book (Chap. 38, except Fig. 38.7) is actually a **clockwise** rotation in the euclidean coordinates
  - we show the actual counter-clockwise rotation in these slides
- Note that pixel coordinates usually have the y axis pointing down
  - counter-clockwise rotation in euclidean looks clockwise in pixel coordinates



Rotation

- Counterclockwise rotation by angle θ about the origin:

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

- note: the top-left 2x2 matrix is the rotation matrix in heterogeneous coordinates
- R is an orthogonal matrix both in heterogeneous and homogeneous coordinates
  - easy to invert: $R^T = R^{-1}$
  - preserves distances from the origin

Counterclockwise rotation by angle θ about the origin:

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

Think about how the basis vectors are transformed by the rotation:

- $\begin{pmatrix} 1 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix}$
- $\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\frac{\pi}{2} \\ \sin\frac{\pi}{2} \end{pmatrix} \rightarrow \begin{pmatrix} \cos(\frac{\pi}{2}+\theta) \\ \sin(\frac{\pi}{2}+\theta) \end{pmatrix} = \begin{pmatrix} -\sin\theta \\ \cos\theta \end{pmatrix}$

Placing these transformed basis vectors as the columns gives the rotation matrix R.

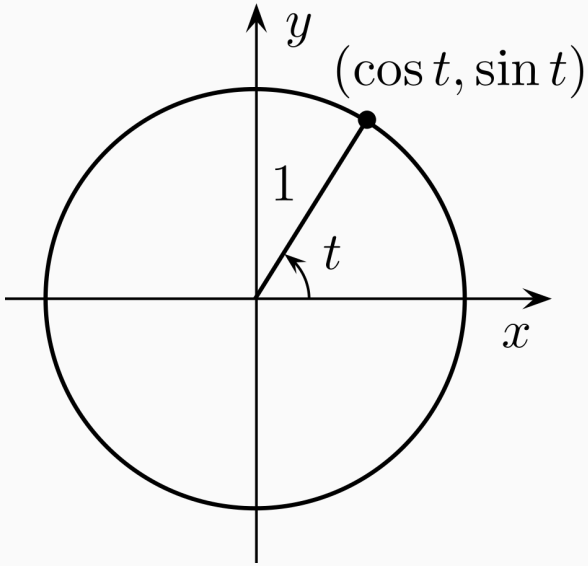


Figure 9: Gustavb, Public domain, da Wikimedia Commons

Shearing

- Horizontal shear by q_x: $x' = x + q_x y$
- Homogeneous shear matrix:

$$Q = \begin{pmatrix} 1 & q_x & 0 \\ q_y & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- Combines horizontal (q_x) and vertical (q_y) shear; preserves area but not angles

Horizontal ($q_x = 1, q_y = 0$), vertical ($q_x = 0, q_y = 1$), and arbitrary shear.

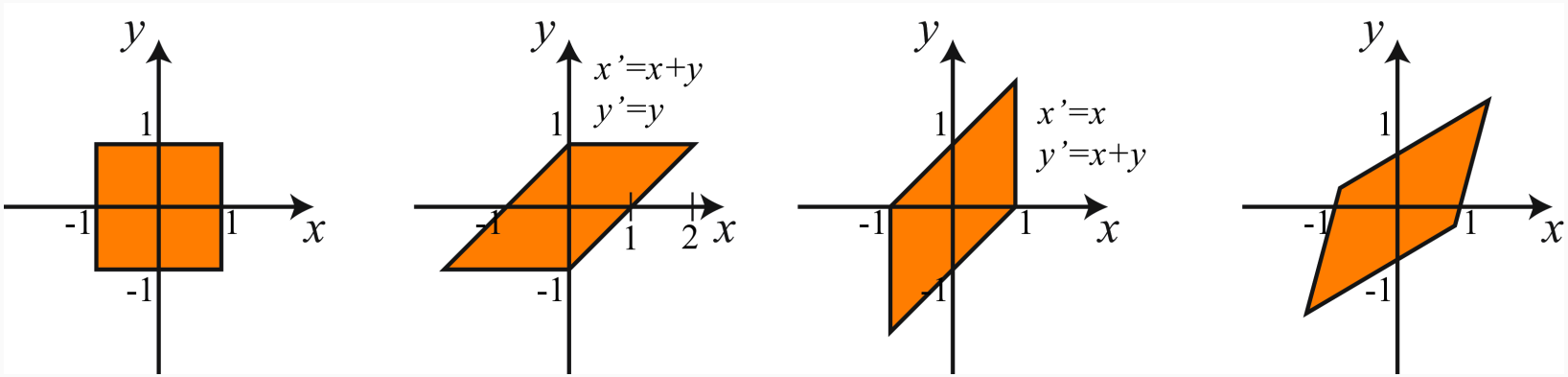


Figure 10: source: visionbook

Summary

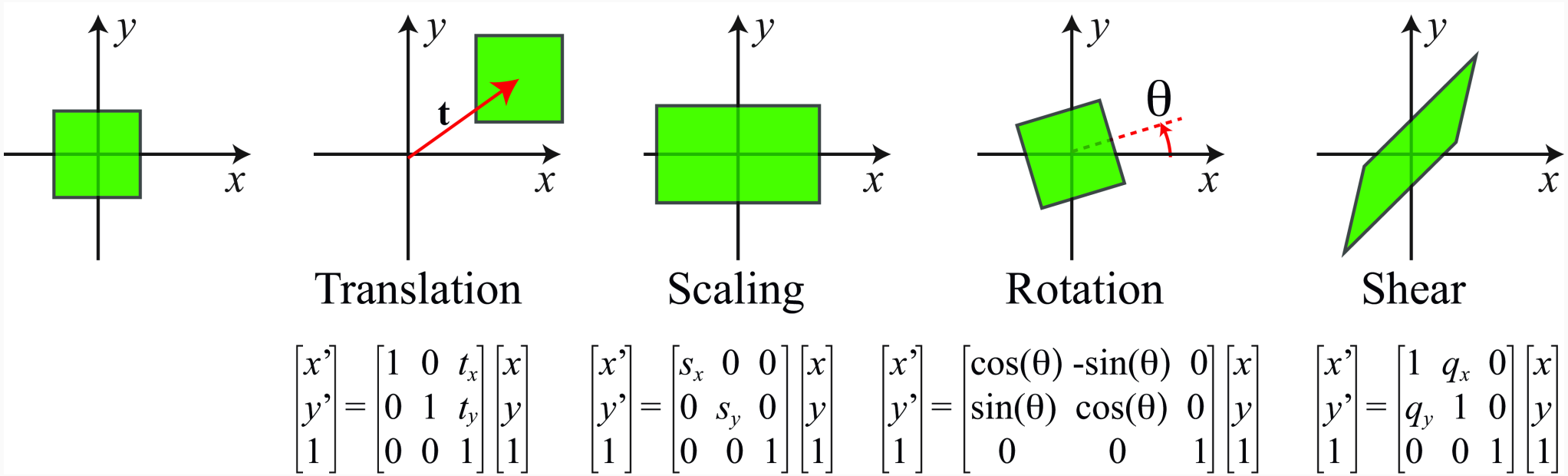


Figure 11: source: visionbook

Chaining Transformations

- Any sequence of translations, scalings, rotations, shears can be combined into a **single** 3×3 matrix: $M = T_n ... T_2 T_1$
- Order matters: matrix multiplication is not commutative
- One matrix multiplication per point, regardless of how many transforms were chained
- each set of transformation is a group (closed under matrix multiplication)
  - example: $R_{\theta_1} R_{\theta_2} = R_{\theta_1 + \theta_2}$, same for every transform

$$\boldsymbol{p}' = \begin{bmatrix} 1 & q_x & 0 \\ q_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \boldsymbol{p}$$

Chaining: Example

Rotation around a non-origin point t.

- We did not define a matrix for this operation because R rotates only around the origin
- We can create it by composition

steps:

- first, translate by $-t$ to move the rotation center from t to $(0, 0)$
- then, rotate around the origin using the standard rotation matrix
- finally, translate by t to move the center back

in matrix form:

$$p' = T_t R_\theta T_{-t} p$$

Note that transformations apply right-to-left

Generic 2D Transformations

- all the transformations we defined make an **affine** transformation (6 DoF): linear part + translation, last row fixed to $[0, 0, 1]$

$$\begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix}$$

- Preserves parallelism (parallel lines remain parallel)
- **Projective** transformation (8 DoF, scale-invariant): general invertible 3×3 matrix

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

- Preserves only collinearity (straight lines remain straight)

2D Transformations as Convolutions

- For images represented as point clouds (point sets $\{[\ell_i, x_i, y_i]\}$), geometric transformation can be viewed as a 1D convolution over the explicit coordinate list
- For example, with rotation the convolution kernels are:
  - $w_x = [\cos(\theta), \sin(\theta)]$
  - $w_y = [-\sin(\theta), \cos(\theta)]$.
  - weights are the same for all inputs (as in every 1D conv).

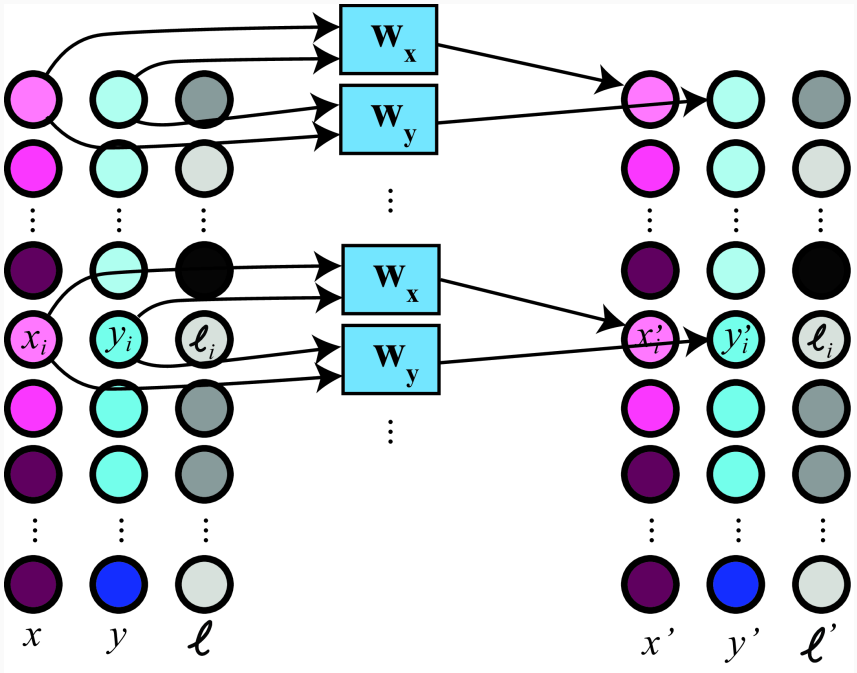


Figure 12: source: visionbook

Image Warping

Image Warping and Geometric Transforms

- **Warping**: resampling an image under a geometric transformation M (one of the 3×3 matrices above, or a general nonlinear map)
  - to get the usual rectangular grid of pixels we need to compute pixel intensity in the "holes"
- Two strategies: **forward mapping** (source pixels to target) and **backward mapping** (pull target pixels from source)

Forward Mapping

- For each source pixel (x, y), compute $(x', y') = M(x, y)$ and write the source value into the target at (x', y')
- Simple to implement, but the target locations (x', y') are not integer pixel coordinates in general

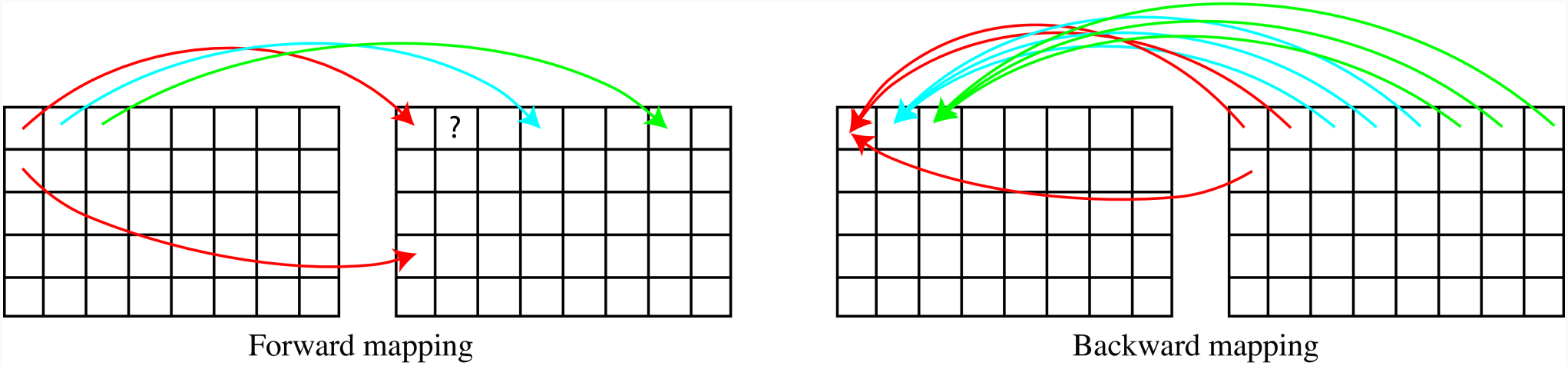


Figure 13: source: visionbook

Why Forward Mapping Doesn't Work Well

- Non-integer target coordinates: need to round, causing **holes** (uncovered target pixels)
- Multiple source pixels can map to the same target pixel (overlap), or none at all (gaps)
- It's even more problematic with non-rigid transforms, where the mapping is non-uniform

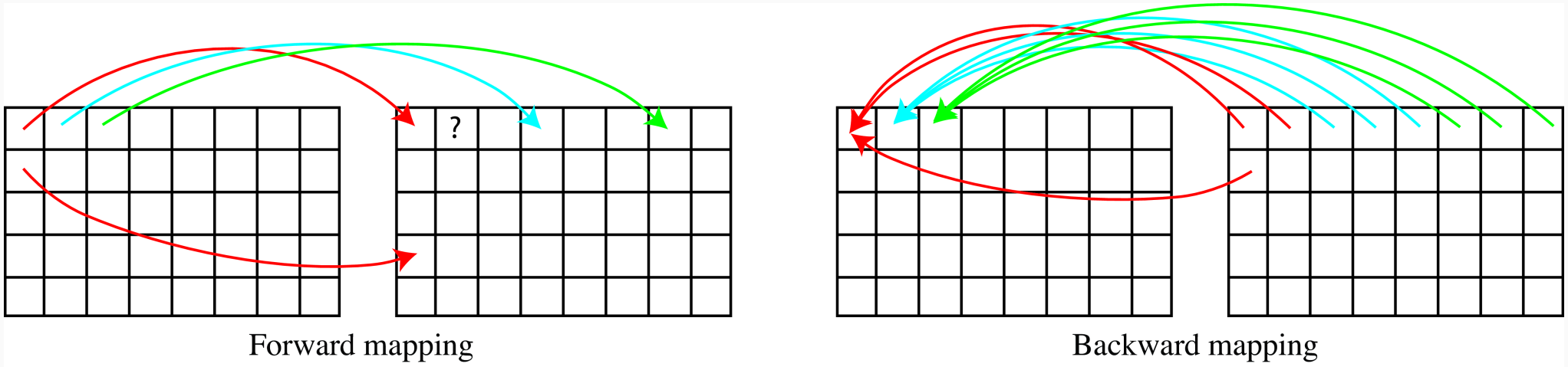


Figure 14: source: visionbook

Backward Mapping

- For each **target** pixel (x', y'), compute the source location $M^{-1}(x', y')$ and interpolate the source image there
- Every target pixel gets exactly one value (no holes)
- Requires the inverse transform M^{-1}, which is easy to obtain by inverting the matrix

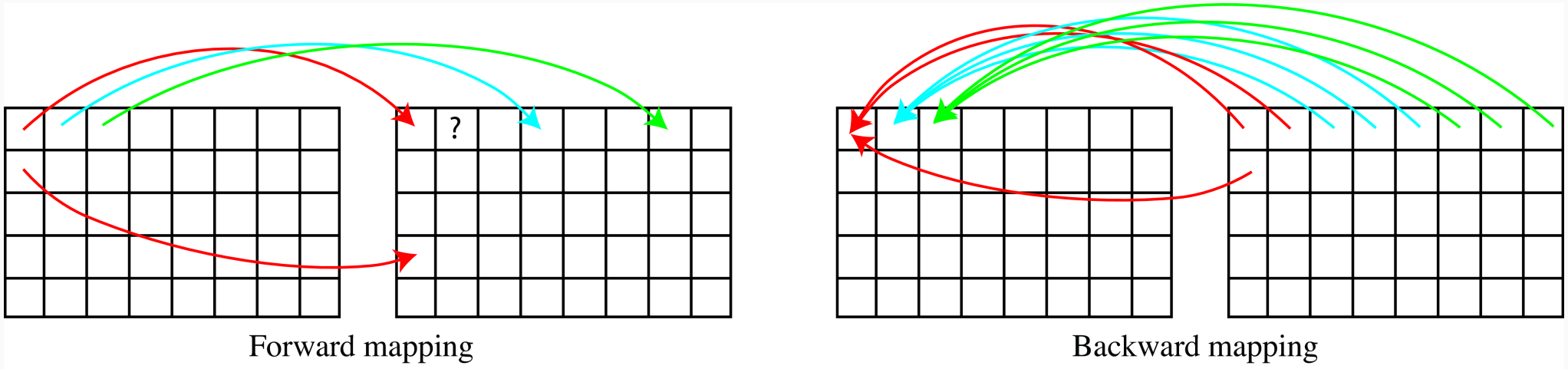
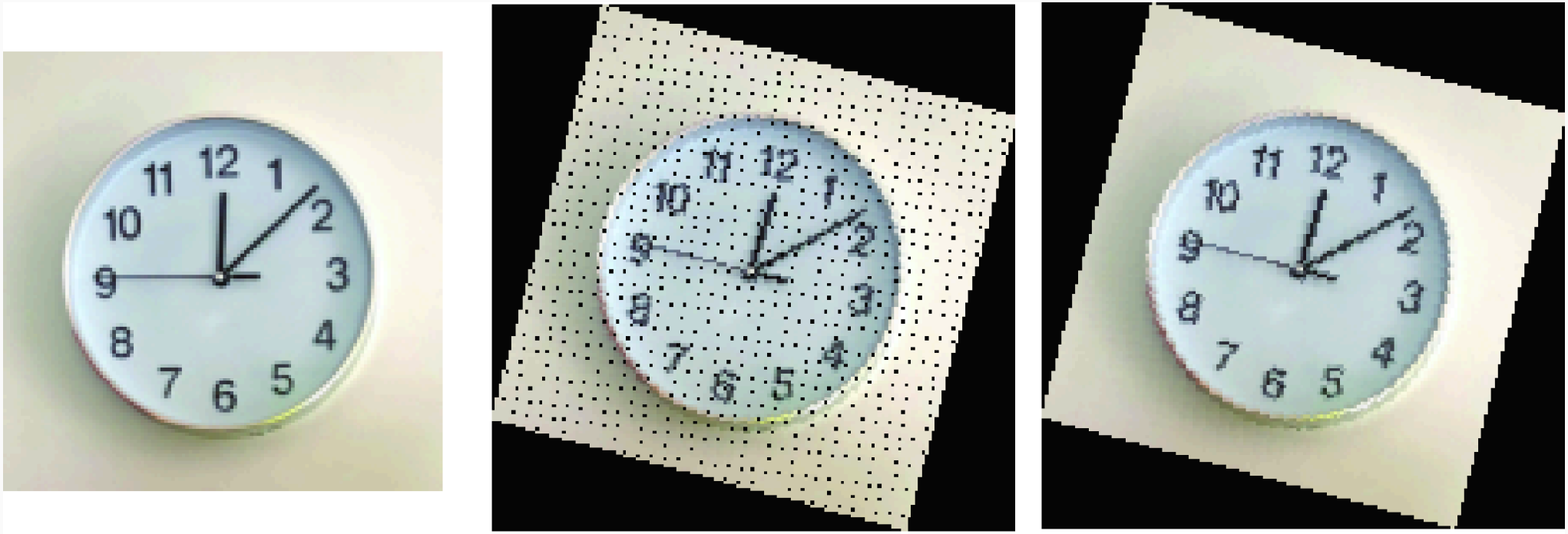


Figure 15: source: visionbook

Example



Forward mapping

Backward mapping

Figure 16: source: visionbook

High-Quality Warping

- Nearest neighbor is not the only (or the optimal) interpolation choice
  - nearest-neighbor (blocky), bilinear, higher-order kernels, ...
- **MIP-mapping**: uses a multiscale pyramid to improve the interpolation

Camera Model in Homogeneous Coordinates

Camera Model

We cannot always align image and world coordinates to have simple perspective equations:

- we may have multiple cameras in different positions and orientations. The origin can be located at most in one of them.
- the camera can move over time



Figure 17: source: visionbook

The Pinhole Projection Equations: from World to Camera to Image

- World point $\boldsymbol{P} = [X, Y, Z]^T$ (heterogeneous) or $[X, Y, Z, 1]^T$ (homogeneous)
- Image pixel $\boldsymbol{p} = [x, y]^T$ (heterogeneous) or $[x, y, 1]^T$ (homogeneous)
- Full pipeline:
  - **world** (3D, meters) coordinates
  - **camera** (2D, meters) coordinates in 2D (extrinsics R, T)
  - **pixel** (3D, pixels) coordinates (intrinsics K)

The final result we will get is a matrix that combines all the parameters:

$$\boldsymbol{p} = K[R \mid -RT]\boldsymbol{P}_W$$

Intrinsic and Extrinsic Parameters

- **Intrinsics** K: how camera coordinates map to pixels (depends only on the camera/sensor, not its pose)
- **Extrinsics** R, T: how world coordinates map to camera coordinates (depends only on camera pose)
- Full projection: $\boldsymbol{p} = K[R \mid -RT]\boldsymbol{P}_W$ (extrinsics first, then intrinsics)

3D Camera Projections with Homogeneous Coordinates

Camera Coordinate Frame: Setup and Conventions

- Camera coordinate frame:
  - origin at the camera center
  - Z-axis along the optical axis (perpendicular to the image)
  - camera looks in the direction of **positive** Z
- Image/projection plane sits at distance f (focal length) along Z

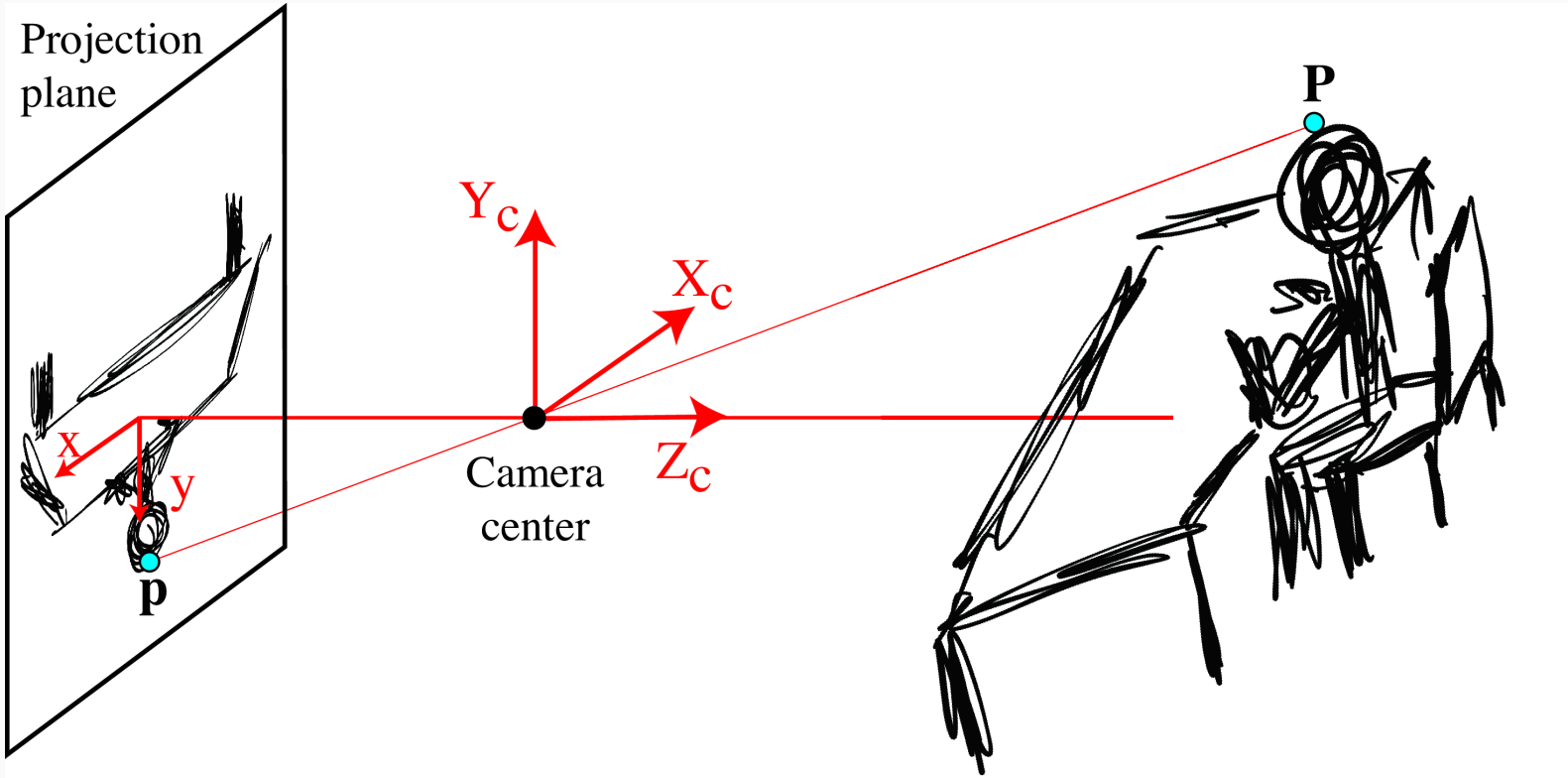


Figure 18: source: visionbook

Camera Center and Projection Plane, f

- Camera center: the pinhole, where all projection rays converge
- Projection (image) plane at $Z = f$
  - note, in the figure, the projection plane is located at $Z = -f$
  - at this point, the image is mirrored
  - if we compute the perspective projection at $Z = f$ instead, we get the same result but without the mirroring
- A 3D point $\boldsymbol{P} = (X, Y, Z)$ projects onto the plane at the intersection with the ray from the camera center through $\boldsymbol{P}$

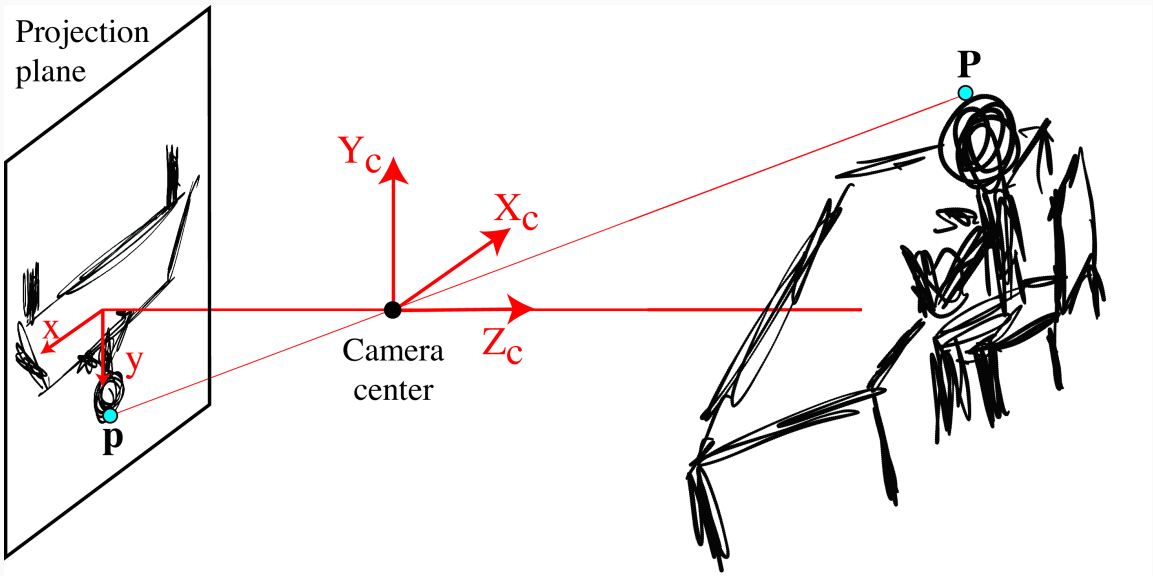


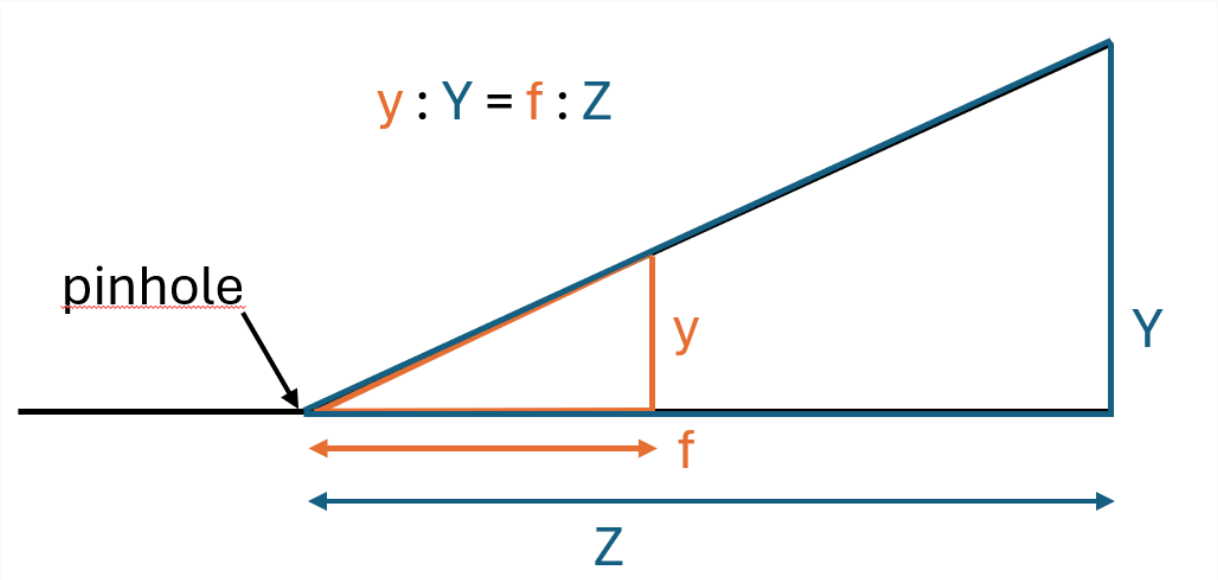
Figure 19: source: visionbook

Simple Case with "Aligned Camera": Perspective and Similar Triangles

- **Perspective Projection**: for a point at world coordinates (X, Y, Z), similar triangles give:

$$x = f\frac{X}{Z}, \quad y = f\frac{Y}{Z}$$

- these are image coordinates. The unit is meters (not pixels)
- the division by Z is the only nonlinear part
  - this equation is still using heterogeneous coordinates. We need to convert it to homogeneous coordinates now



Perspective with Homogeneous Coordinates

- given a world point $\boldsymbol{P} = [X, Y, Z, 1]$ in homogeneous coordinates
- given the focal length f
- with heterogeneous coordinates we have $x = f\frac{X}{Z}, \quad y = f\frac{Y}{Z}$
- with homogeneous coordinates we use the matrix $\boldsymbol{K}$, the projection matrix

$$\mathbf{KP} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} = \lambda \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Perspective with Homogeneous Coordinates

$$\mathbf{KP} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} = \lambda \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

we can show that $\boldsymbol{K}$ is the projection matrix by convert $\mathbf{KP}$ to heterogeneous coordinates

$$\mathbf{KP} \to \begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} = \begin{pmatrix} x\frac{F}{Z} \\ f\frac{Y}{Z} \end{pmatrix} = \boldsymbol{p}$$

notice that $\boldsymbol{p}$ follows the perspective equations $x = f\frac{X}{Z}, \quad y = f\frac{Y}{Z}$

Perspective with Homogeneous Coordinates

$$\mathbf{KP} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} = \lambda \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Since homogeneous coordinates are equivalent up to any arbitrary uniform rescaling, the matrix is often rescaled as

$$\mathbf{K} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{f} & 0 \end{bmatrix}$$

Intrinsic Parameters

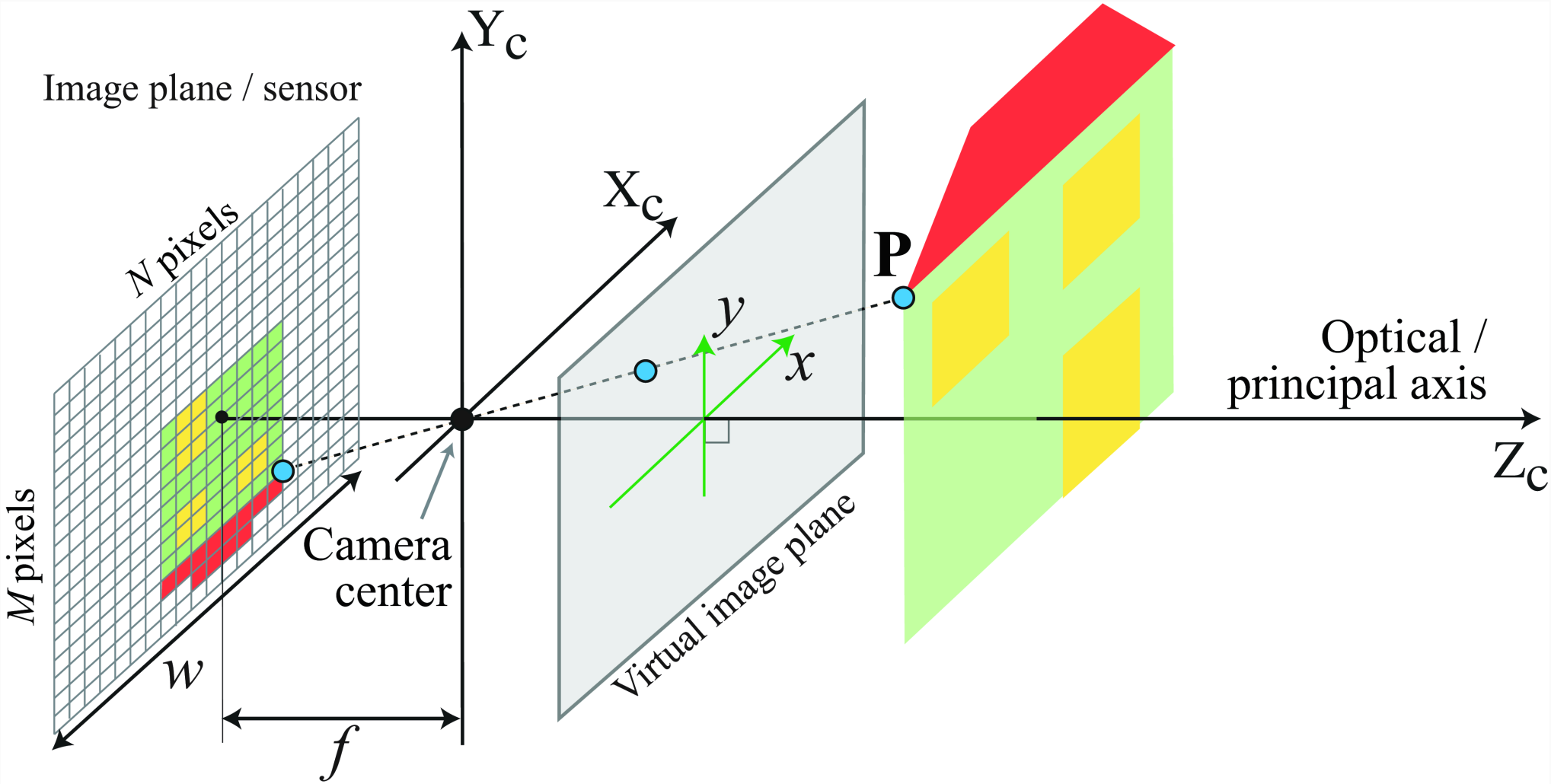


Figure 21: source: visionbook

Focal Length in Pixels vs. Physical Units

- in physical units we have the focal length f (meters)
- sensor width w (meters)
- image width N pixels
- $\frac{N}{w}$ is the number of pixels per meter in the sensor
- the resulting scaled (pixel) focal length is $a = f\frac{N}{w}$
- in matrix form (homogeneous) we get

$$\mathbf{K} = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Now $\mathbf{K}$ gives us the 2D coordinates in pixels instead of meters

Principal Point

- (c_x, c_y): pixel location where the optical axis intersects the sensor plane
  - the principal point is in the center of the image
  - the origin of the coordinate system in pixels is usually the top-left corner
  - we need to translate the coordinates
- For a sensor centered on the optical axis we have $c_x = \frac{N}{2}, c_y = \frac{M}{2}$

$$\mathbf{K} = \begin{bmatrix} a & 0 & c_x & 0 \\ 0 & a & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

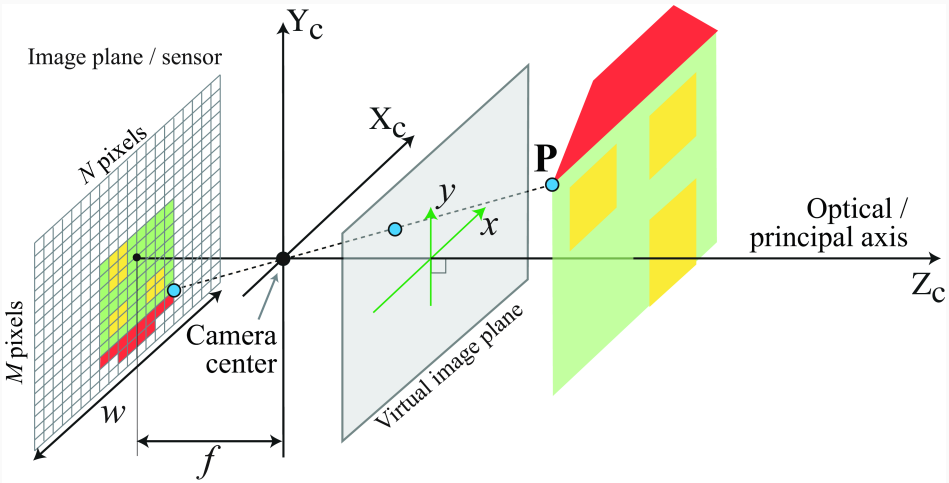


Figure 22: source: visionbook

Pixel Aspect Ratio: Non-Square Pixels

- Aspect ratio is captured by having $a \neq b$ when pixels are non-square (different physical width/height)
- a and b differ through $\frac{N}{w}$ vs. $\frac{M}{h}$ if sensor width $\neq$ height

$$\mathbf{K} = \begin{bmatrix} a & 0 & c_x & 0 \\ 0 & b & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Putting It Together: Full Form of K

Final result of camera-to-pixel mapping:

$$\begin{bmatrix} a & 0 & c_x & 0 \\ 0 & b & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \rightarrow \begin{pmatrix} a\frac{X}{Z} + c_x \\ b\frac{Y}{Z} + c_y \end{pmatrix} = \begin{pmatrix} n \\ m \end{pmatrix}$$

- 4 intrinsic parameters: a, b, c_x, c_y

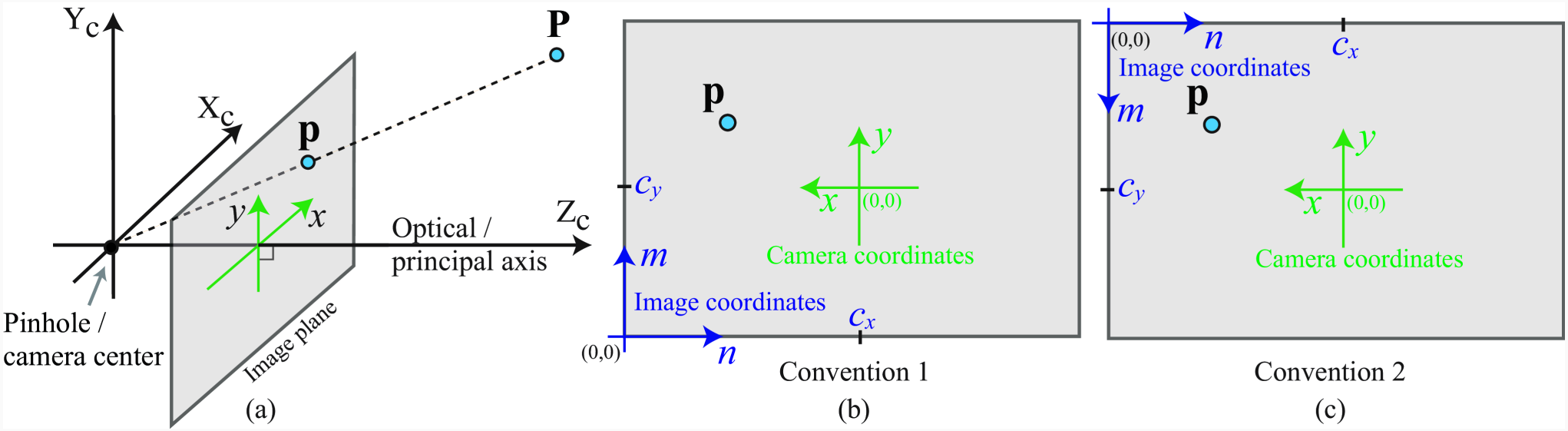


Figure 23: source: visionbook

Sign of a and b

- the sign of a and b can be negative if we switch the direction of the axis from image to pixel coordinates
- in the standard convention, where the pixel origin is at the top-left (convention 2 in the figure), both a and b are negative
- the book uses convention 1, where only a is negative
- notice that the x axis of the virtual image plane (green x arrow) is inverted in the sensor plane due to projection

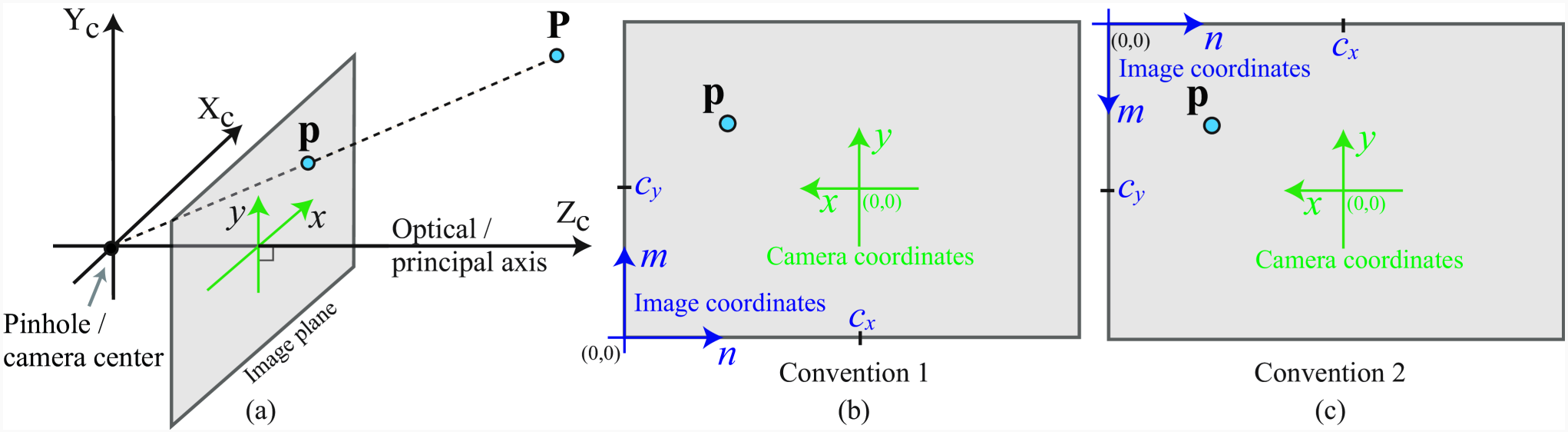


Figure 24: source: visionbook

Other Camera Parameters

We ignored some additional complexities:

- pixels might be arranged along non-perpendicular axes, which requires introducing a skew parameter
- with radial distortion introduced by lenses, straight lines will appear curved
- standardized tools for camera calibrations take into account all these different aspects
- https://docs.opencv.org/5.0/main_modules/calib.html

Simple, Unreliable Calibration

Simple, Unreliable Calibration



Figure 25: source: visionbook

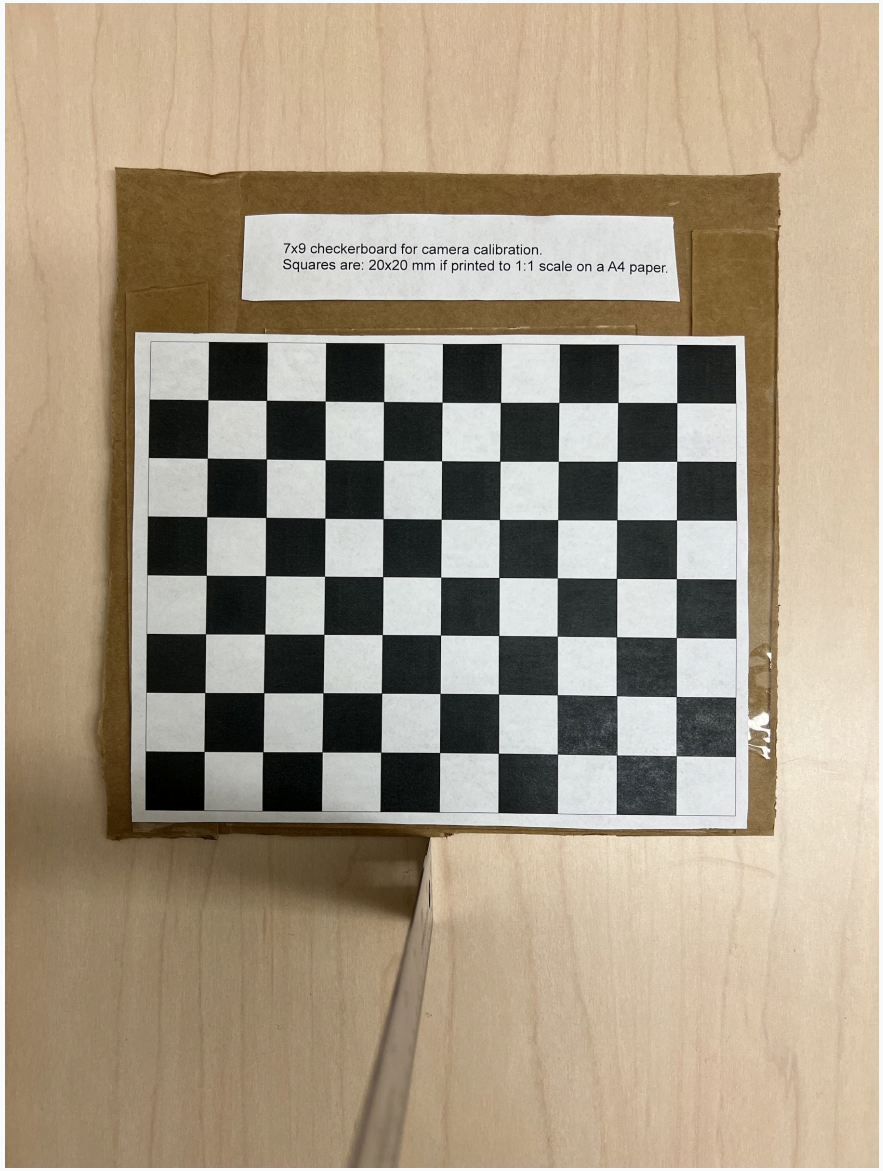


Figure 26: source: visionbook

Calibration: Setup

- object of known size. The chessboard is a standard. Width $W = 20cm$
- known distance between camera and object $Z = 31cm$
- a picture of the object taken by the camera. Width of the chessboard $L = 2002$ pixels



Figure 27: source: visionbook

Calibration: Estimation

- chessboard width (world) $W = 20cm$
- camera-object distance $Z = 31cm$
- chessboard width (image) $L = 2002$ pixels
- these quantities are related to: $a = Z\frac{L}{W}$
- solving for $a = \frac{31 \times 2002}{20} = 3103.1$
- the picture size is 4032×3024 pixels
  - so, we have $c_x = \frac{3024}{2}$ and $c_y = \frac{4032}{2}$. Putting all together

$$\texttt{bold}\ (K) = \begin{bmatrix} 3103.1 & 0 & 1512 & 0 \\ 0 & 30103.1 & 2016 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$



Figure 28: source: visionbook

Calibration: Accuracy

How accurate is this method?

- estimated $a = 3103.1$
- correct $a = 3024$ (from hardware specs)

side note: by calibrating in this way, we do not need to know the actual hardware parameters, which are often hard to find, affected by post-processing, and they need to be verified anyway.

Extrinsic Parameters

World Coordinate Frame vs. Camera Coordinate Frame

Up to now, we made to big simplifications:

- camera origin = world origin
- optical axis aligned with world axis

World Coordinate Frame vs. Camera Coordinate Frame

Up to now, we made to big simplifications:

- camera origin = world origin
- optical axis aligned with world axis

Separate coordinate systems:

- **World frame**: fixed reference frame for the scene, shared across all cameras/views
- **Camera frame**: origin at the camera center, axes aligned with the optical axis
- **Extrinsic** parameters describe the rigid transform that maps world coordinates into the camera frame

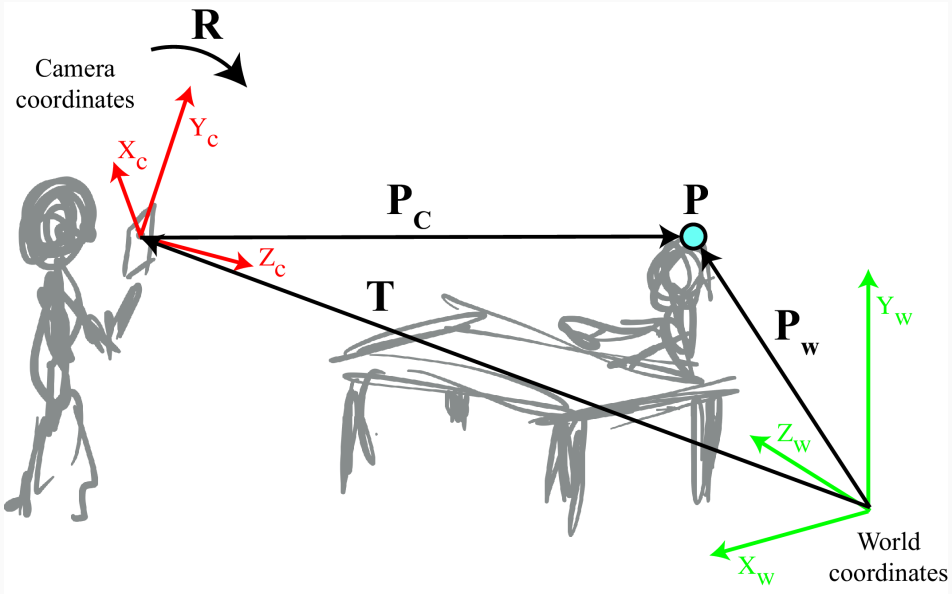


Figure 29: source: visionbook

Transformation Parameters

the world coordinates (green) and camera frame (red) are related by

- a translation by a vector $\boldsymbol{T}$ (from world to camera origin)
- axes are rotated by $\boldsymbol{R}^T$ (rotation to convert from world to camera orientation)
- before applying the intrinsic parameters, we need to convert the world coordinates to be aligned with camera frame

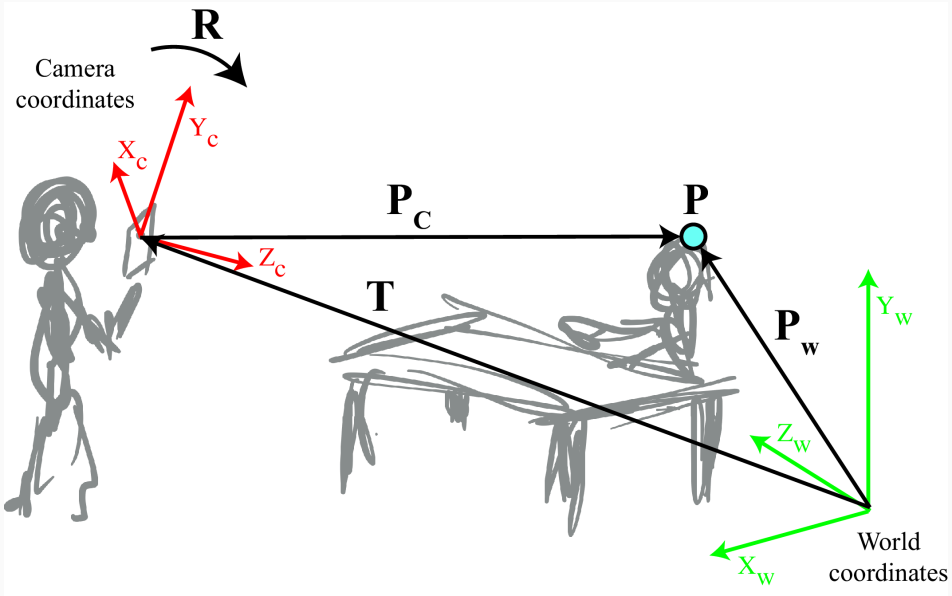


Figure 30: source: visionbook

World Coordinate Frame vs. Camera Coordinate Frame

How do map a point $\boldsymbol{P}_W$ from world to camera frame?

- $\boldsymbol{P}_W$ is in world coordinates (green axes)
- we translate by $-\boldsymbol{T}$ so that the origin of the two frames are the same
- we rotate by $\boldsymbol{R}$

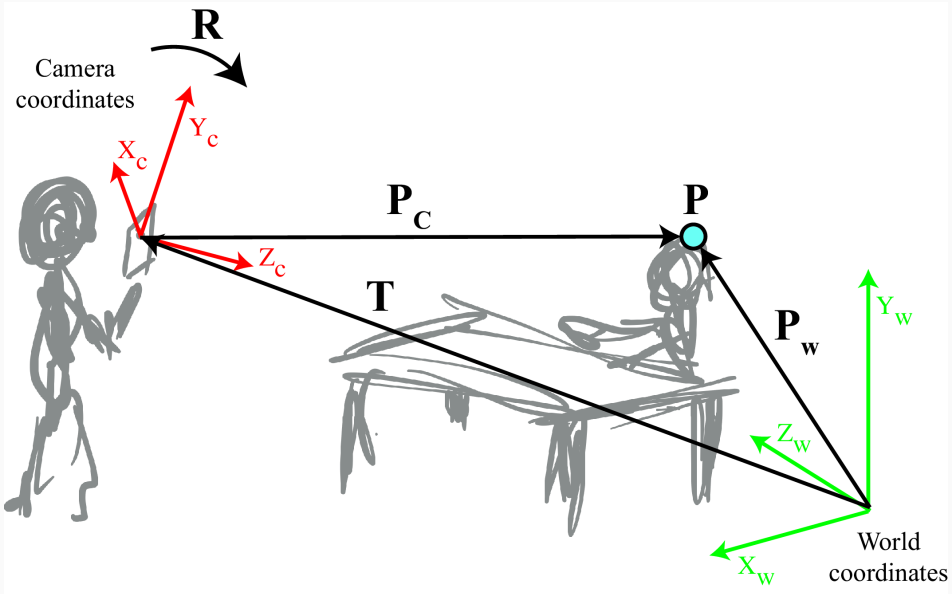


Figure 31: source: visionbook

World Coordinate Frame vs. Camera Coordinate Frame

How do map a point $\boldsymbol{P}_W$ from world to camera frame?

- $\boldsymbol{P}_W$ is in world coordinates (green axes)
- we translate by $-\boldsymbol{T}$ so that the origin of the two frames are the same
- we rotate by $\boldsymbol{R}$

in heterogeneous coordinates:

$$\boldsymbol{P}_c = \boldsymbol{R}(\boldsymbol{P}_W - \boldsymbol{T}) = \boldsymbol{R}\boldsymbol{P}_W - \boldsymbol{R}\boldsymbol{T}$$

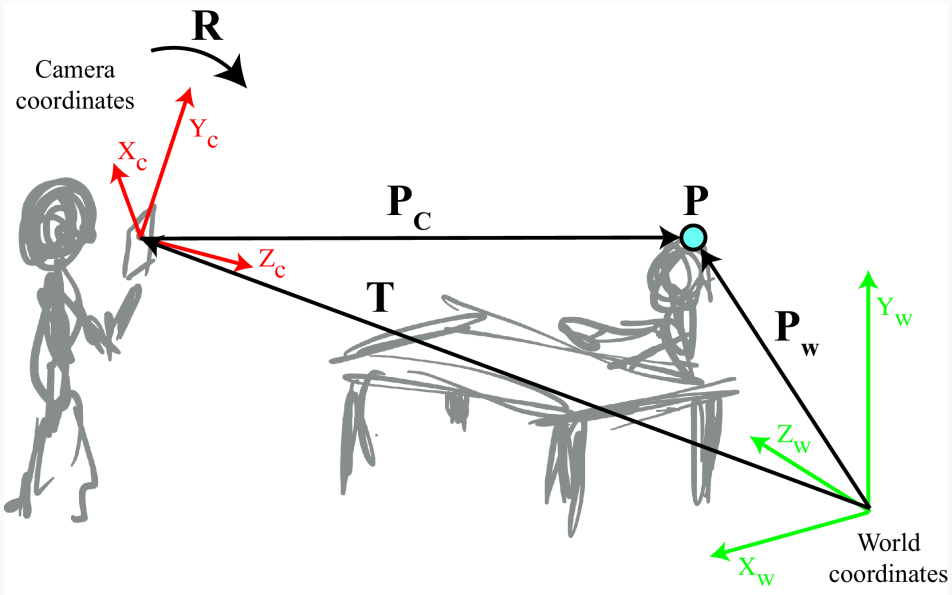


Figure 31: source: visionbook

World Coordinate Frame vs. Camera Coordinate Frame

How do map a point P_W from world to camera frame?

- P_W is in world coordinates (green axes)
- we translate by $-T$ so that the origin of the two frames are the same
- we rotate by R

in heterogeneous coordinates:

$$P_c = R(P_W - T) = RP_W - RT$$

now we have:

- same origin
- Z axis aligned

we can apply the intrinsic transformation to get the pixel coordinates

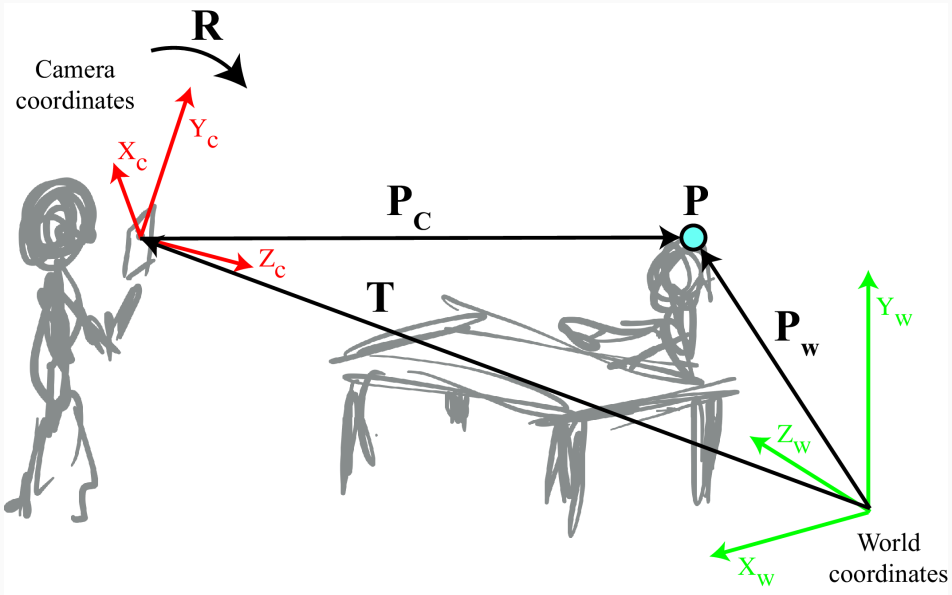


Figure 31: source: visionbook

Translation

$$M_1 = \begin{pmatrix} 1 & 0 & 0 & -T_X \\ 0 & 1 & 0 & -T_Y \\ 0 & 0 & 1 & -T_Z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation

$$M_2 = \begin{pmatrix} R_1 & R_2 & R_3 & 0 \\ R_4 & R_5 & R_6 & 0 \\ R_7 & R_8 & R_9 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

full transformation (first translate, then rotate):

$$\boldsymbol{P}_C = \boldsymbol{M}_2 \boldsymbol{M}_1 \boldsymbol{P}_W$$

Translation

$$M_1 = \begin{pmatrix} 1 & 0 & 0 & -T_X \\ 0 & 1 & 0 & -T_Y \\ 0 & 0 & 1 & -T_Z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation

$$M_2 = \begin{pmatrix} R_1 & R_2 & R_3 & 0 \\ R_4 & R_5 & R_6 & 0 \\ R_7 & R_8 & R_9 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$M_2 M_1 = \begin{pmatrix} R_1 & R_2 & R_3 & -\boldsymbol{r}_1 \boldsymbol{T} \\ R_4 & R_5 & R_6 & -\boldsymbol{r}_2 \boldsymbol{T} \\ R_7 & R_8 & R_9 & -\boldsymbol{r}_3 \boldsymbol{T} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \boldsymbol{R} & -\boldsymbol{R}\boldsymbol{T} \\ \boldsymbol{0}^T & 1 \end{pmatrix}$$

Summary

Parameters: rotation $\boldsymbol{R}$ and translation $\boldsymbol{T}$

$$\boldsymbol{P}_C = \begin{pmatrix} \boldsymbol{R} & -\boldsymbol{R}\boldsymbol{T} \\ \mathbf{0}^T & 1 \end{pmatrix} \boldsymbol{P}_W$$

The Full Camera Matrix

Combining Intrinsic and Extrinsic Parameters

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \lambda \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ w \end{pmatrix} = \boldsymbol{K}\boldsymbol{M}_2\boldsymbol{M}_1 \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

- note: right-to-left application. First extrinsic, then intrinsic
- to convert back to heterogeneous, you just divide by w

usually written as:

$$\boldsymbol{p} = \boldsymbol{K}[\boldsymbol{R} \mid -\boldsymbol{R}\boldsymbol{T}]\boldsymbol{P}_W$$

Degrees of Freedom of P

- Intrinsic: $a, b, c_x, c_y \rightarrow 4$ (often there is also the skew so they become 5)
- Extrinsic: 3 (rotation) + 3 (translation) = 6
- Total: the 3×4 matrix is defined up to scale so it has 11 degrees of freedom

Conclusions

Common Pitfalls

- Always think explicitly about the coordinate systems (world, camera, image)
- be careful about the order of operations
- be careful about inverted axis and sign changes (e.g. inverting axis from camera to image)

Key Takeaways

- Homogeneous coordinates turn projection, translation, and other transforms into a single matrix multiplication
- Camera model $\boldsymbol{p} = \boldsymbol{K}[\boldsymbol{R} \mid -\boldsymbol{RT}]\boldsymbol{P}_W$ split into intrinsic and extrinsic parameters

References

- MIT Vision Book, Chapter 38: *Representing Images and Geometry*
- MIT Vision Book, Chapter 39: *Camera Modeling and Calibration*

Next Lecture

- Estimate K, R, T (and distortion coefficients) from real images
- Direct Linear Transform (DLT) calibration from point correspondences
- Multiplane calibration with a chessboard chart
- Reprojection error minimization: $\sum_i \|\boldsymbol{p}_i - \pi(K[R|-RT]\boldsymbol{P}_i)\|^2$