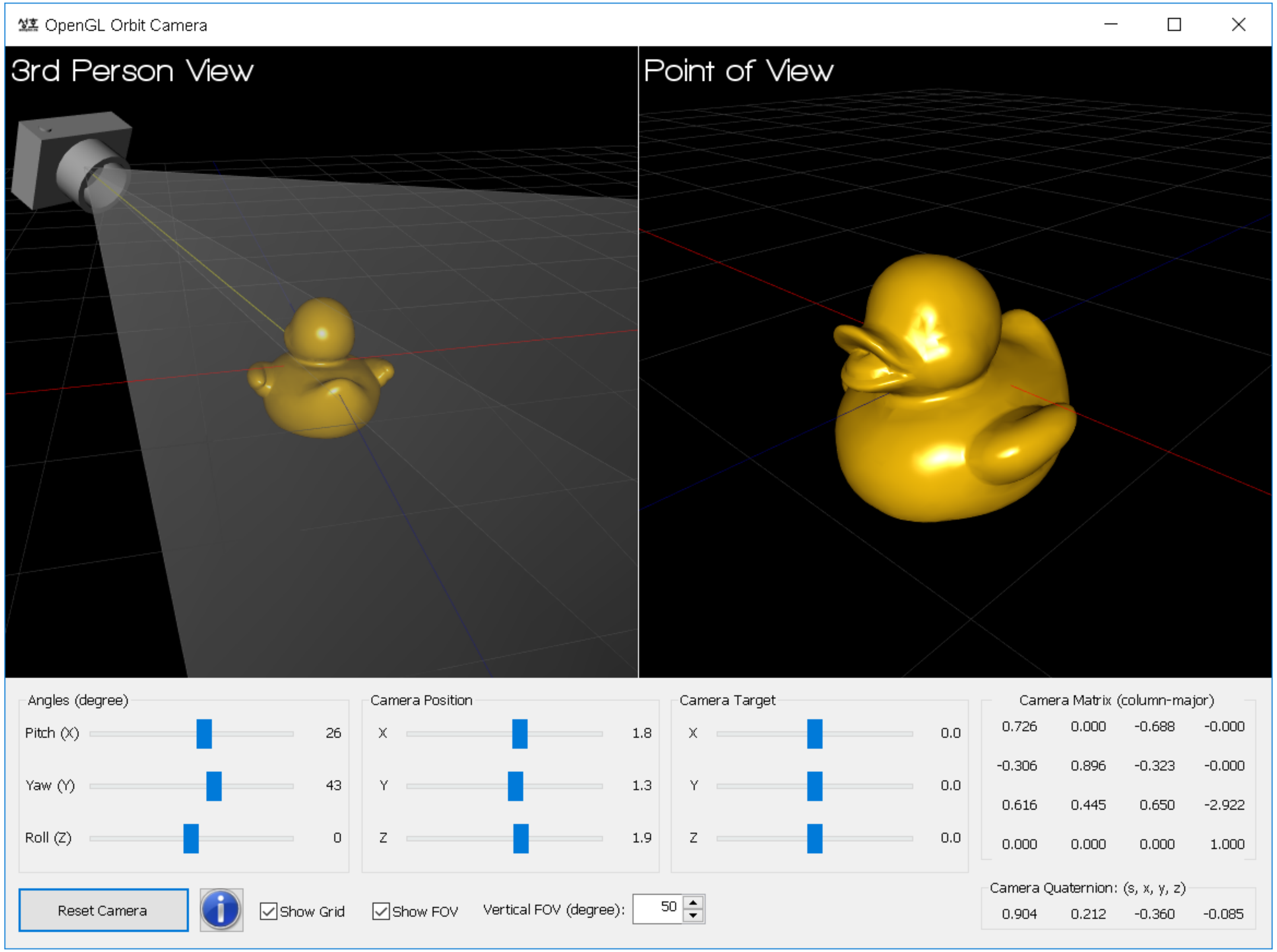
Camera Calibration

Jiayuan Gu

gujy1@shanghaitech.edu.cn

How does a camera take images?



https://www.songho.ca/opengl/gl_camera.html

Three Components

- In the context of rendering geometries (computer graphics)

- We know
  - Camera
  - 3D points (model)

- We render
  - 2D pixels (image)

Review: Pinhole Camera

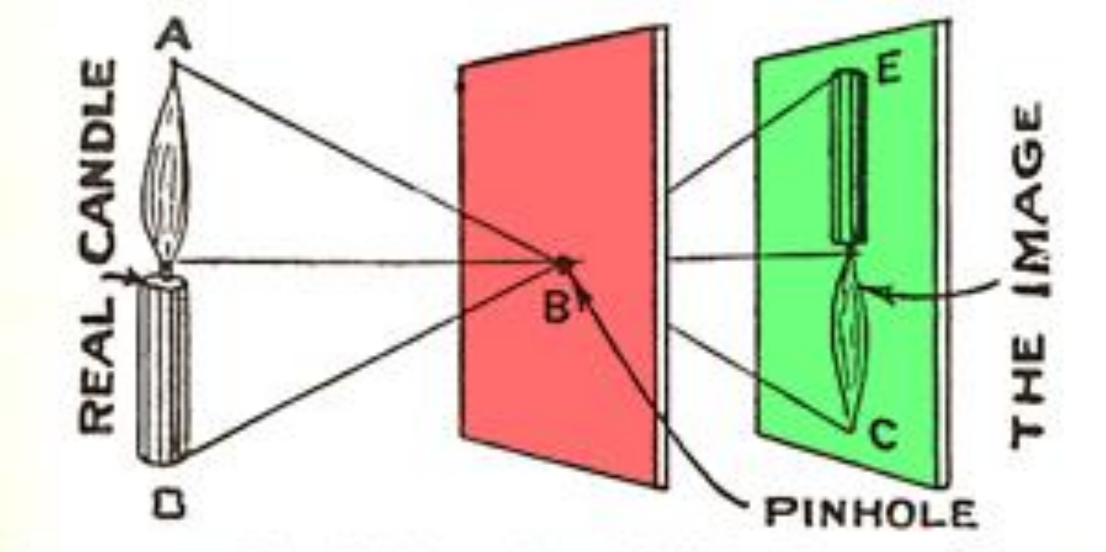
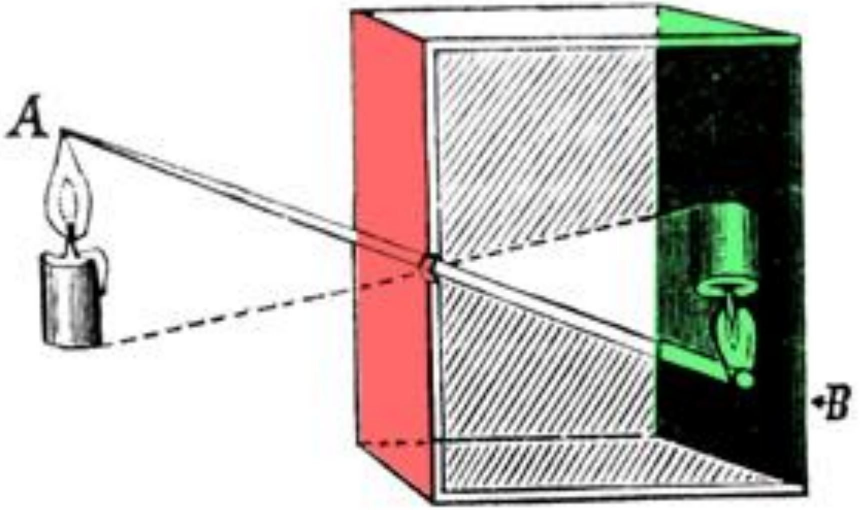


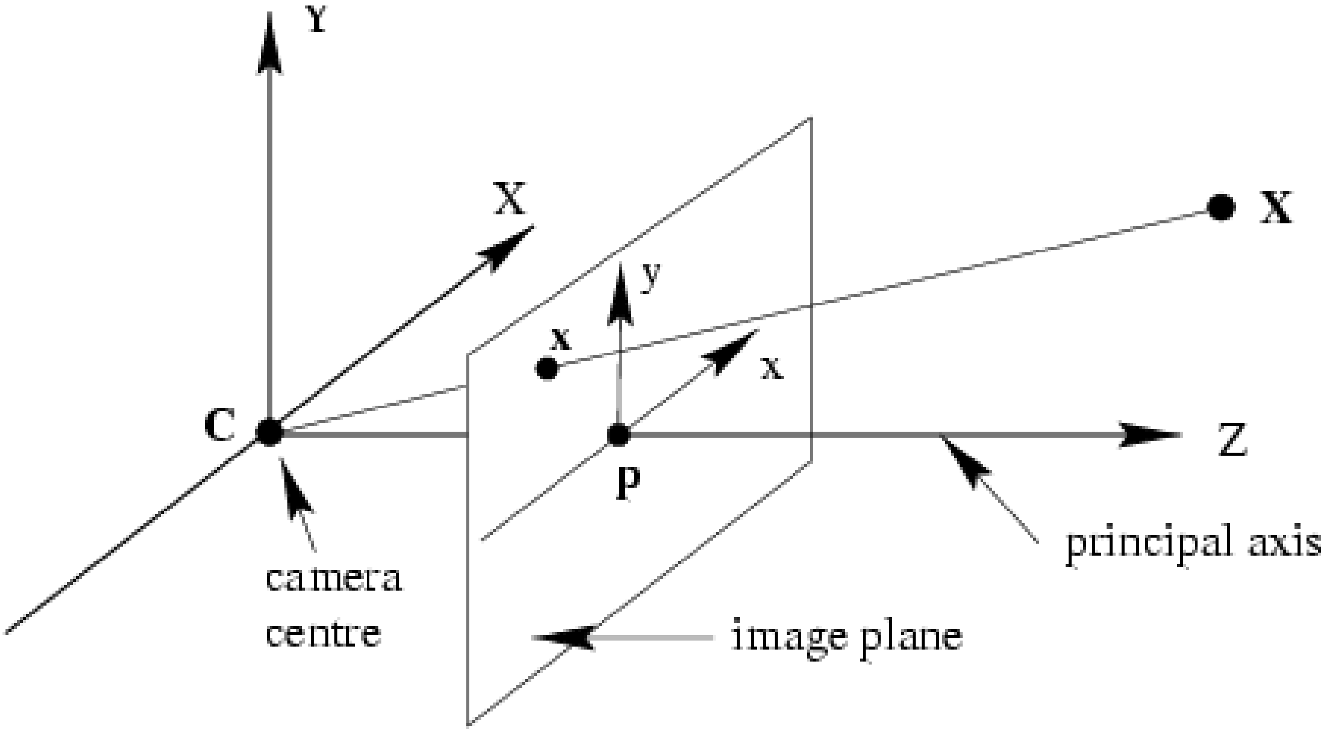
FIG. 131.—How Light and a Pinhole Form an Image.



Illustrated in 1925, in *The Boy Scientist*

Figure from [tutorial]

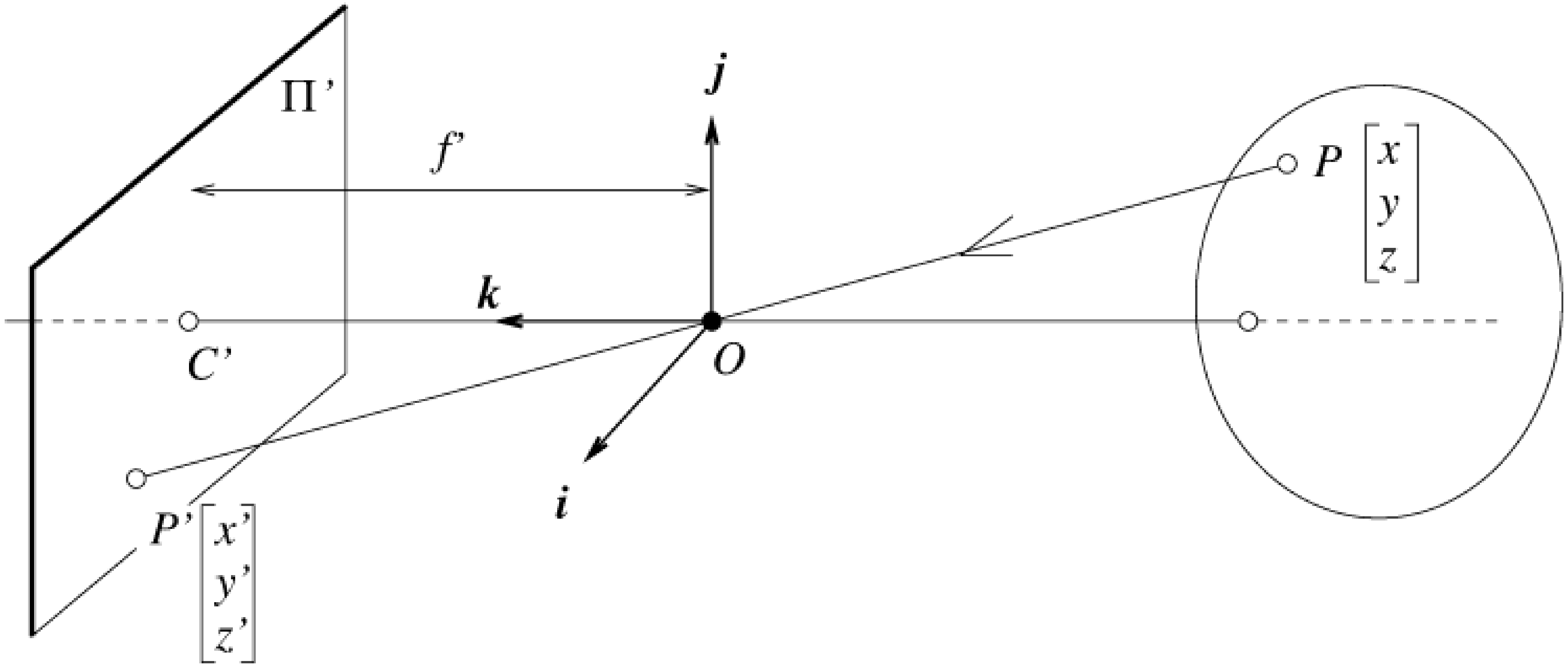
Review: Pinhole Camera



Normalized (camera) coordinate system: camera center is at the origin, the **principal axis** is the z-axis, x and y axes of the image plane are parallel to x and y axes of the camera

Figure from "[Multiple View Geometry in Computer Vision]"

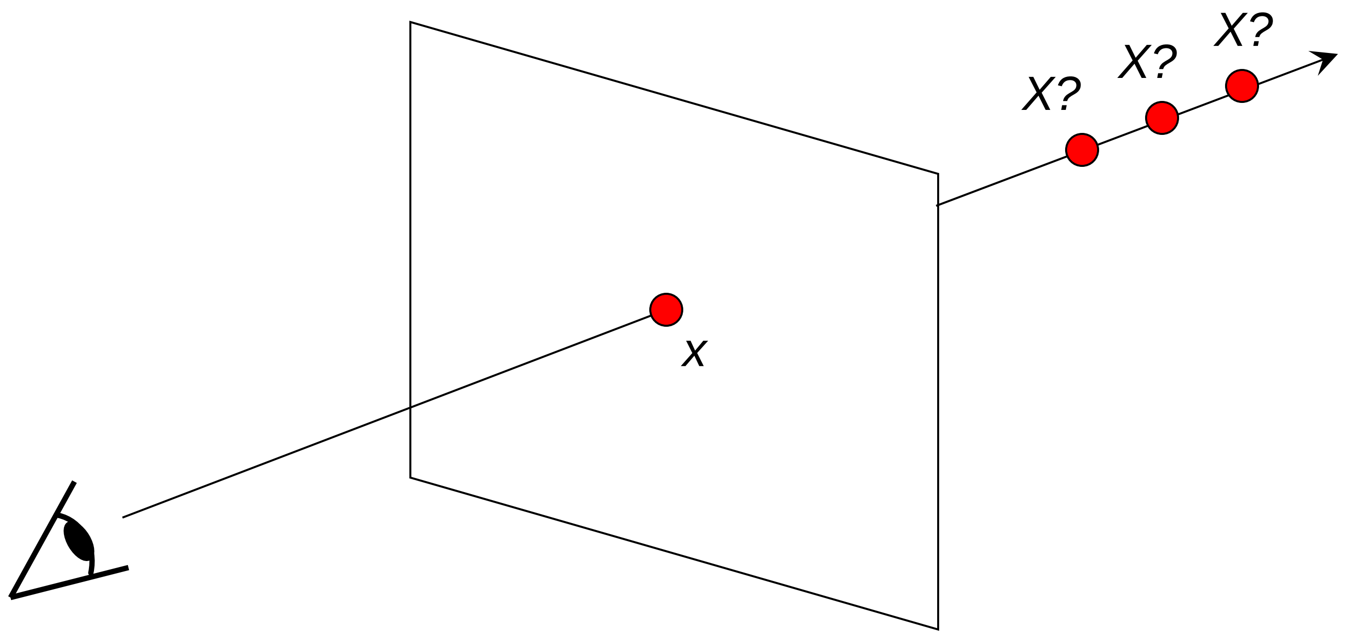
Perspective Projection



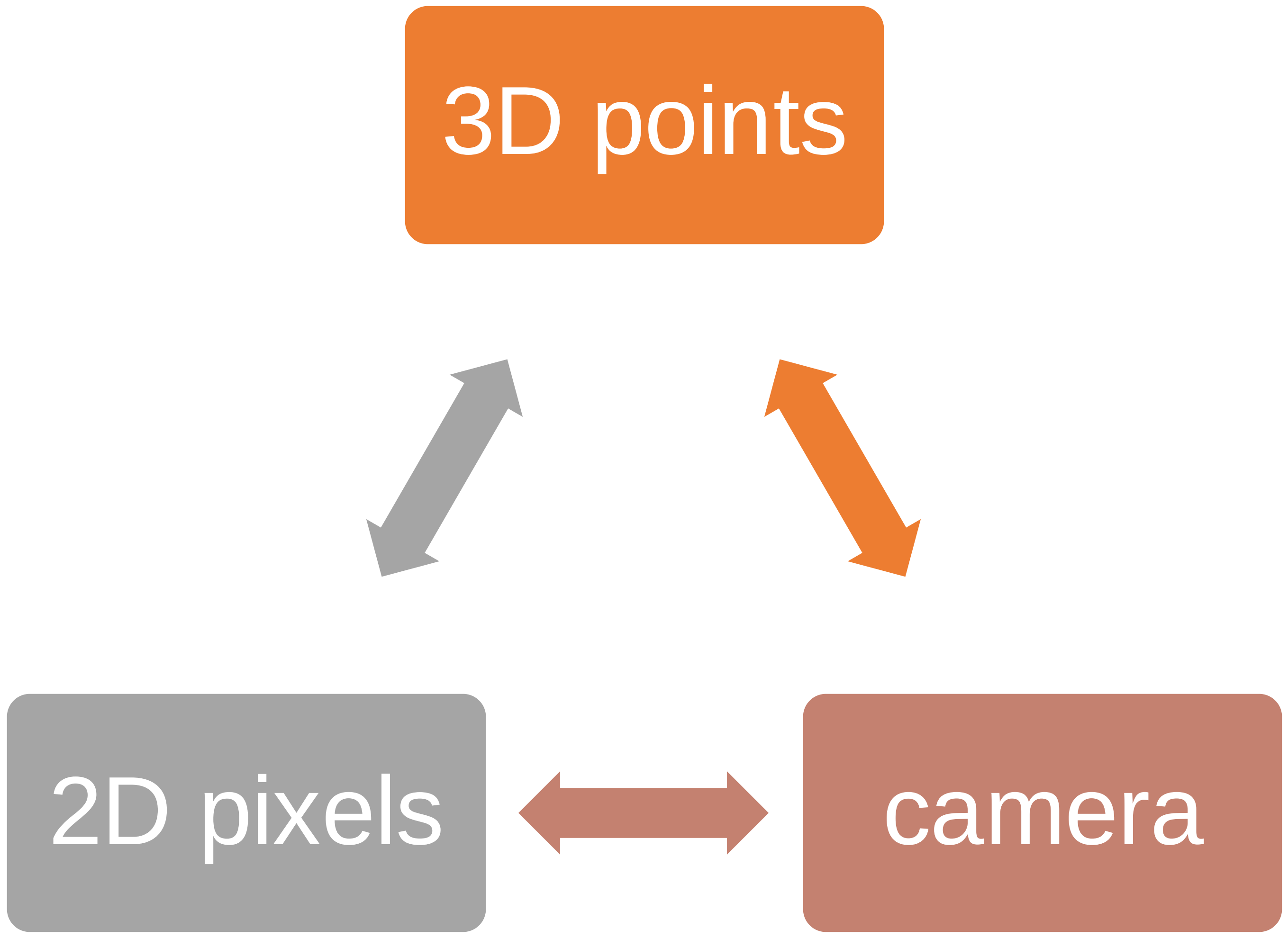
$$\frac{x'}{x} = \frac{y'}{y} = \frac{z'}{z}$$

Source: Bahadir K. Gunturk

Question: Can we recover 3D from images?



Core: Pixels, Points, Camera

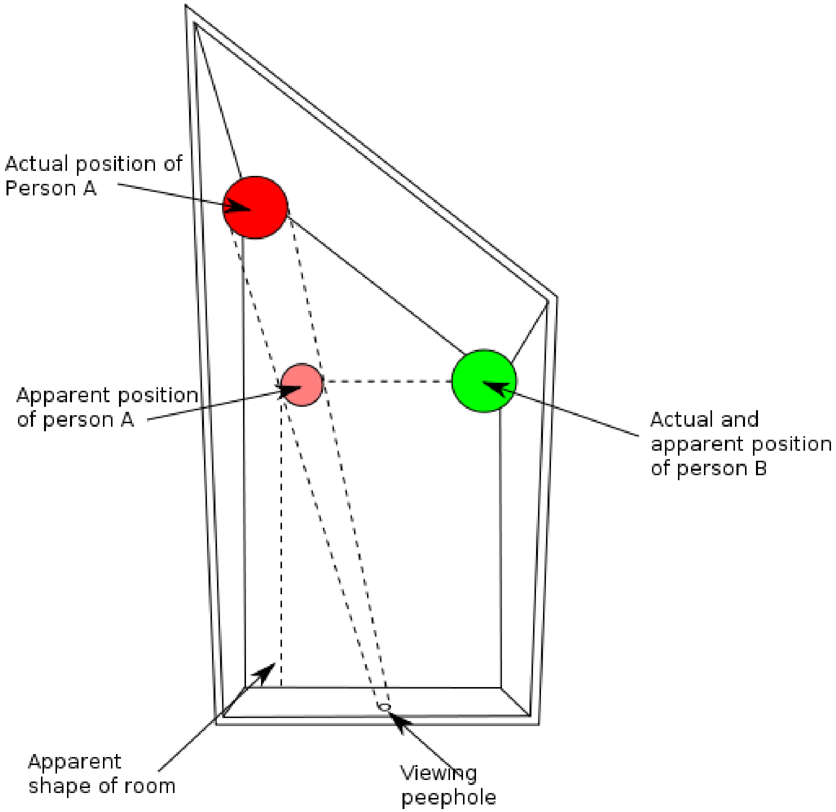


Single-View Ambiguity

Single-View Ambiguity

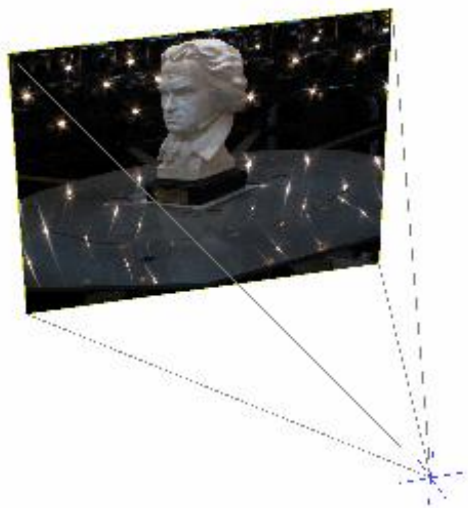
Rashad Alakbarov shadow sculptures

Single-View Ambiguity



Ames room

Multi-view Geometry

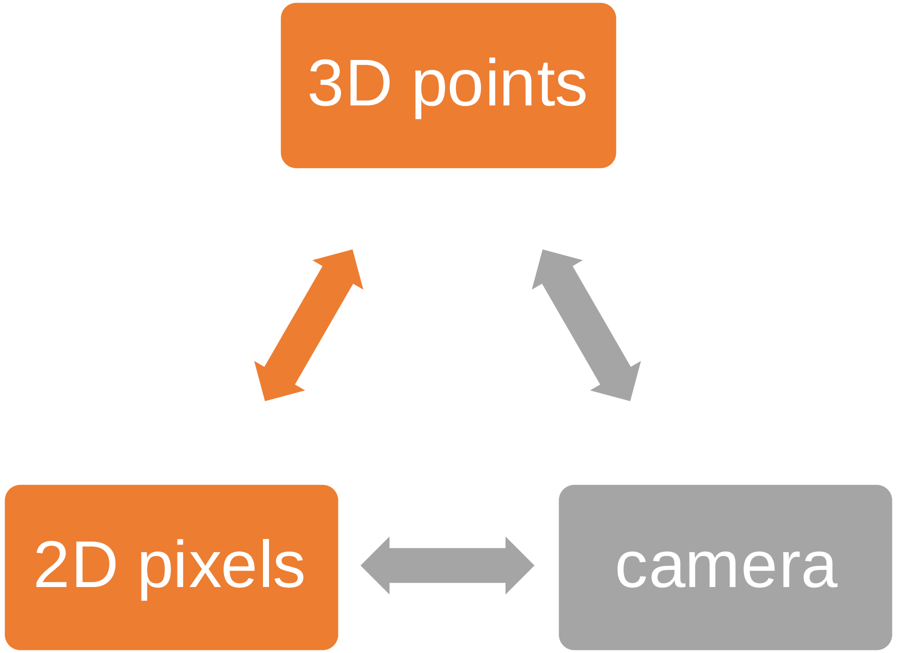


Animation from [TUM]

Camera Calibration

- Known: 3D points and their 2D pixels

- To solve: camera parameters



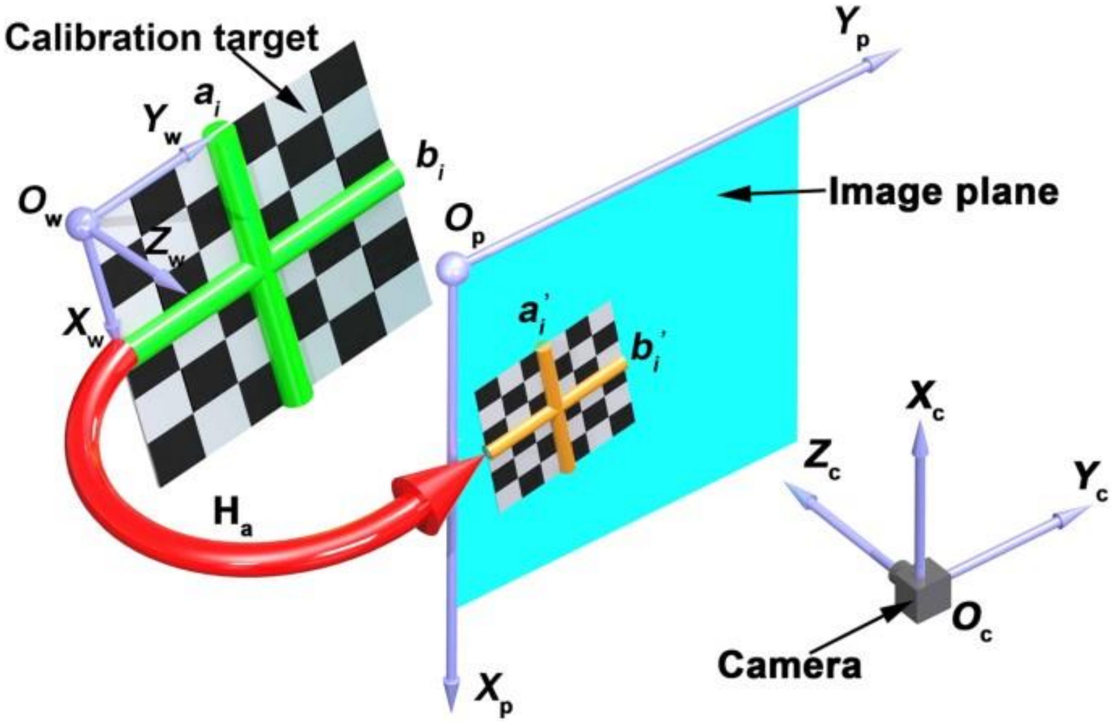
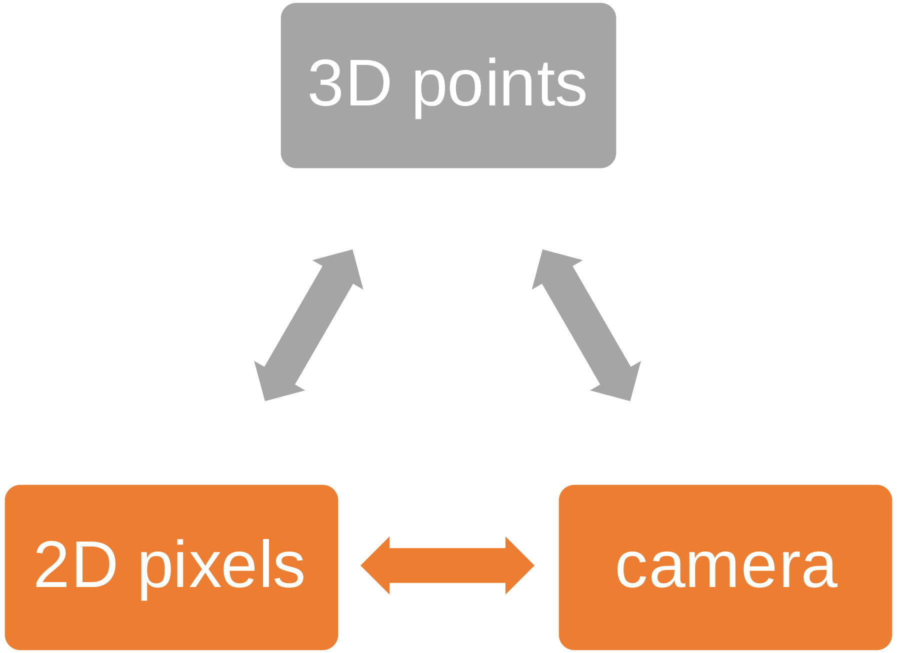
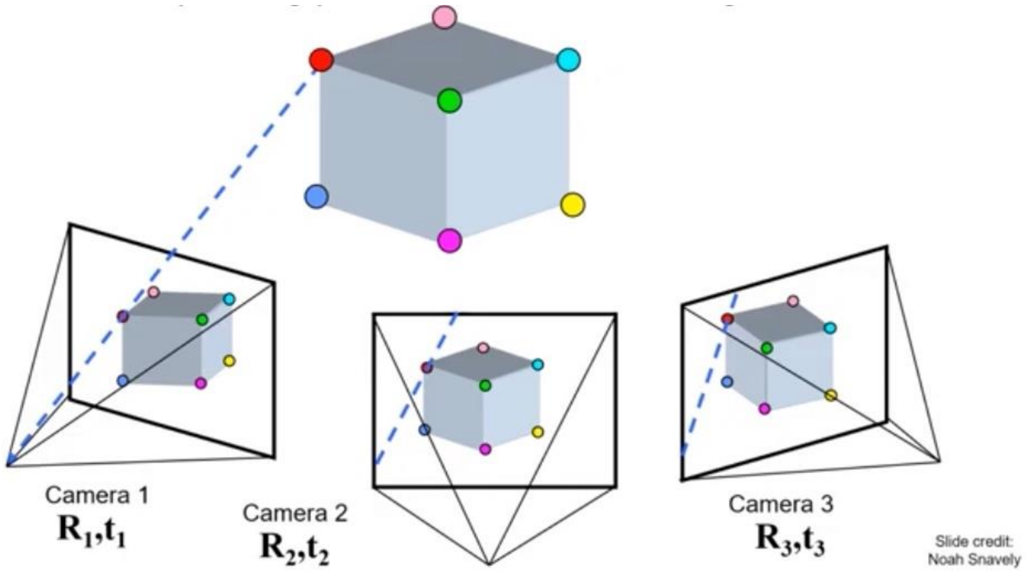


Figure from Nature

Multi-View Stereo (MVS)

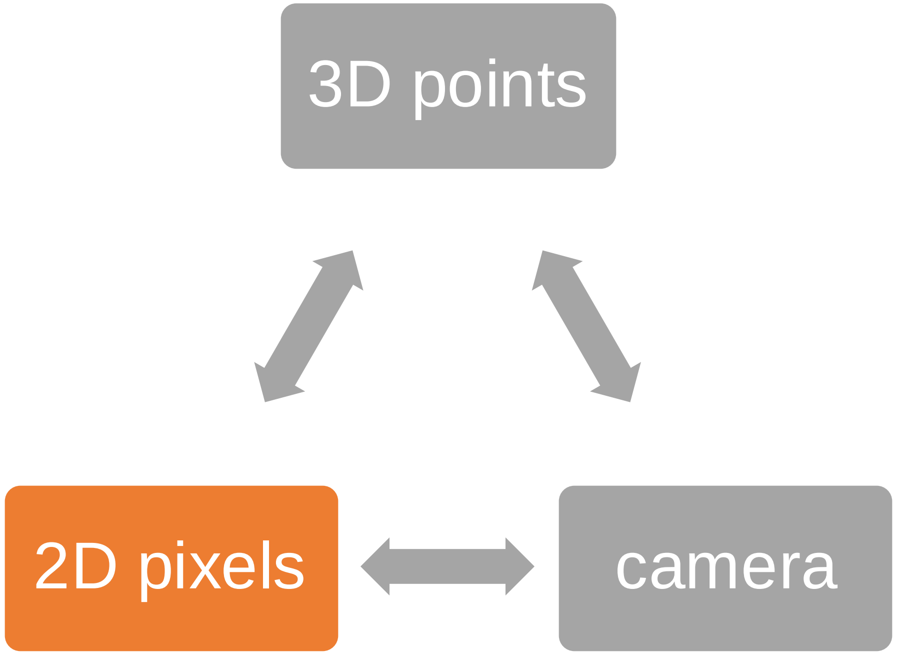
- Known:
  - multiple images (corresponding pixels in each view)
  - camera parameters (including relative poses between views)
- To solve: 3D points (corresponding to pixels in each view)

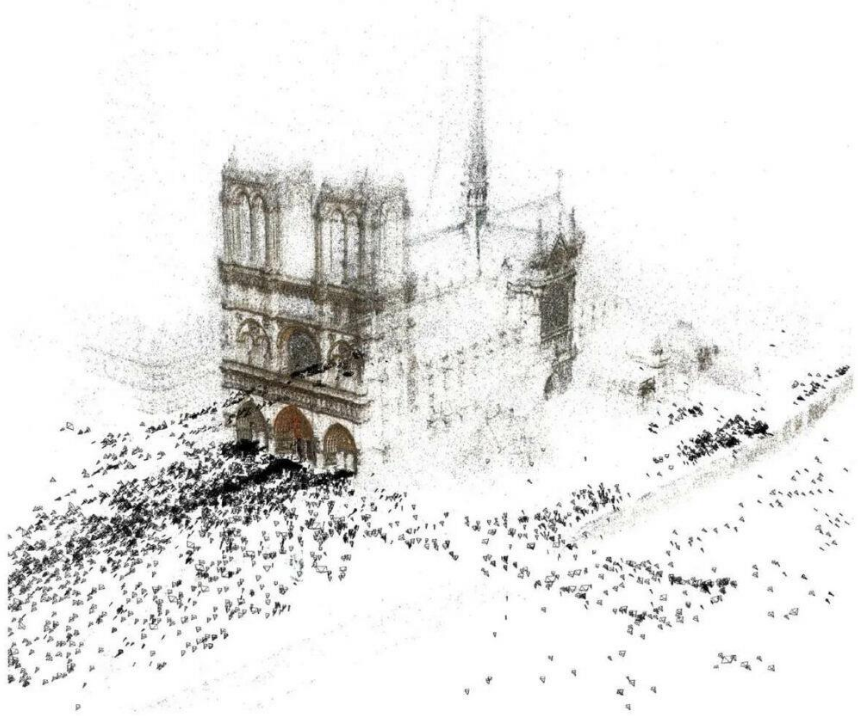




Structure from Motion (SFM)

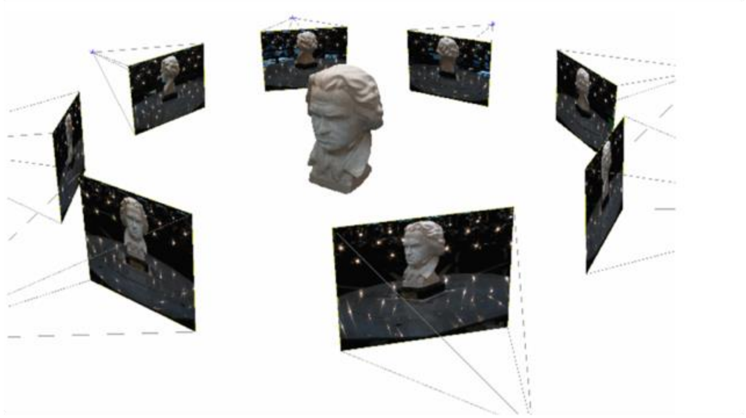
- Known:
  - multiple images (corresponding pixels in each view)
- To solve:
  - 3D points (corresponding to pixels in each view)
  - camera parameters (including relative poses between views)





MVS vs. SFM

Multi-View Stereo

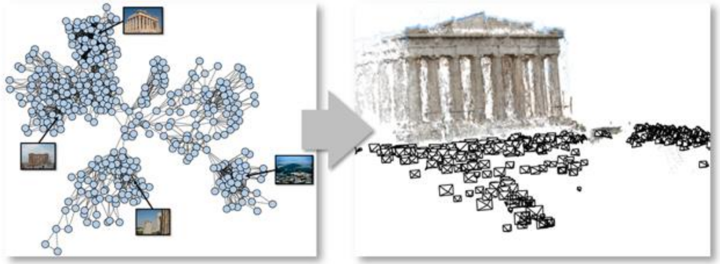


✅ Camera parameters known

✅ Camera position known

📐 Based on Epipolar Geometry

Structure From Motion

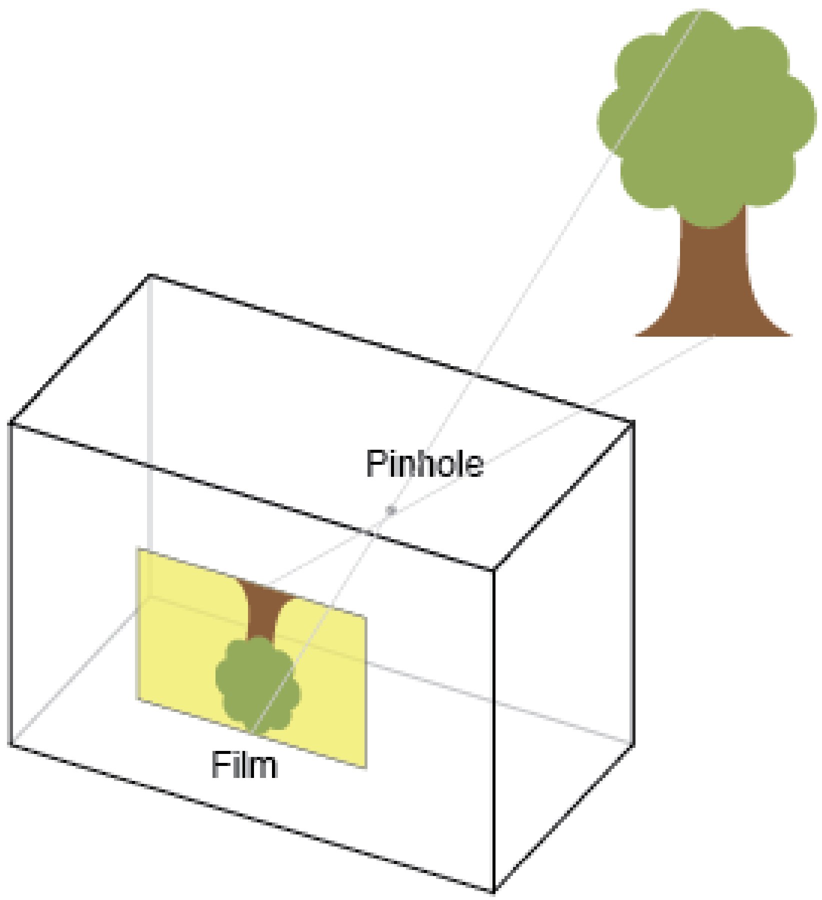


❌ Camera parameters unknown

❌ Camera positions unknown

🔗 Based on Feature Matching

Today’s Topic: Camera Calibration

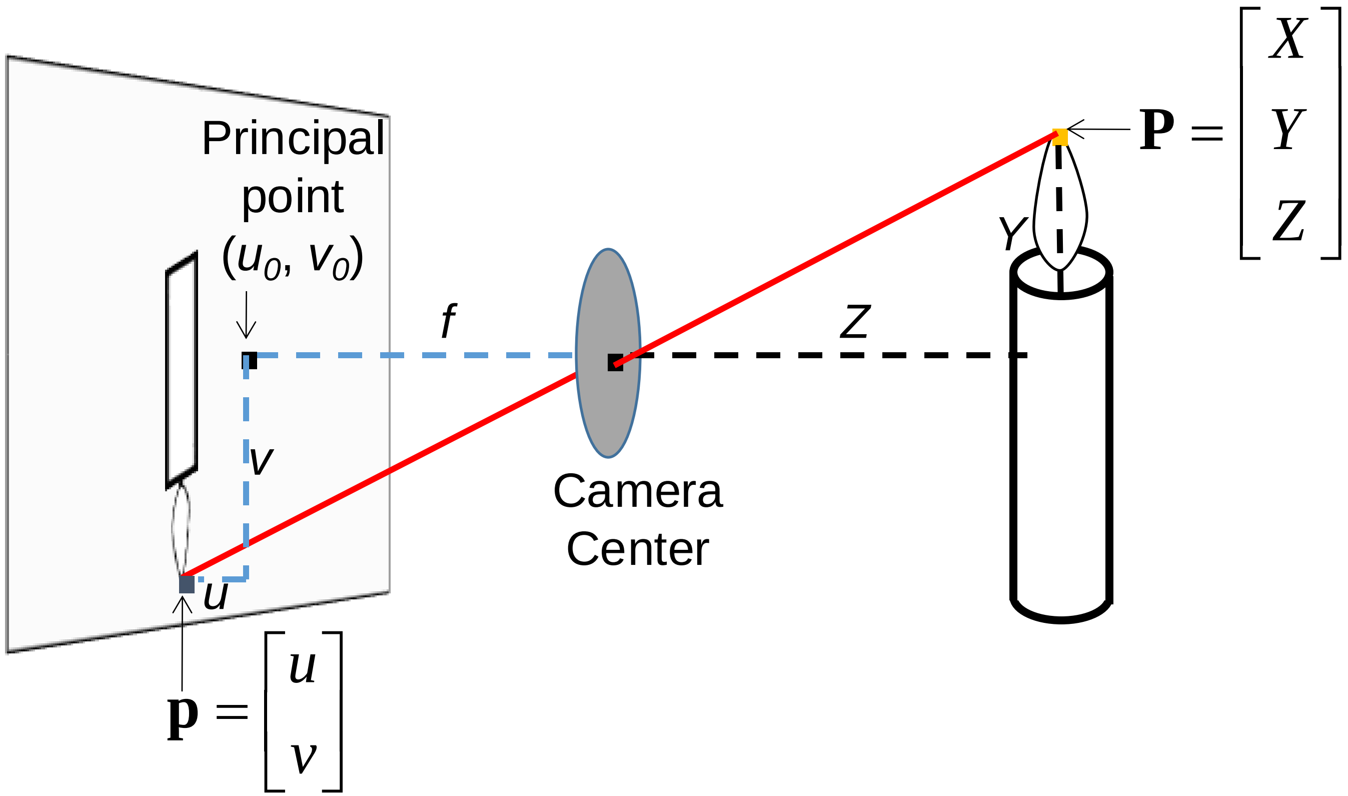


How to parameterize a camera?

How to solve camera parameters?

Figure from blog

From Camera Space to Image Space



$$u = f\frac{x}{z}$$

$$v = f\frac{y}{z}$$

Need a new math tool to conveniently describe the conversion

Source: Derek Hoiem

Homogeneous Coordinates

Converting **to** homogeneous coordinates

$$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

homogeneous image coordinates

$$(x, y, z) \Rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

homogeneous scene coordinates

Converting **from** homogeneous coordinates

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow (x/w, y/w)$$

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \Rightarrow (x/w, y/w, z/w)$$

Homogeneous Coordinates

$$u = f\frac{x}{z}$$

$$v = f\frac{y}{z}$$

$$\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} fx \\ fy \\ z \end{bmatrix}$$

Non-linear formula with inhomogeneous coordinates (Cartesian)

Linear formula with homogeneous coordinates

Homogeneous Coordinates

$$k\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} kx \\ ky \\ kw \end{bmatrix} \Rightarrow \begin{bmatrix} \frac{kx}{kw} \\ \frac{ky}{kw} \end{bmatrix} = \begin{bmatrix} \frac{x}{w} \\ \frac{y}{w} \end{bmatrix}$$

Homogeneous Coordinates Cartesian Coordinates

A point in Cartesian space is a ray in projective space

Homogeneous: invariant to scaling

Geometry in Homogeneous Coordinates

2D line

$$ax + by + c = 0$$

$$[a, b, c]\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0$$

2D line passing two points

$$ax_1 + by_1 + c = 0$$
$$ax_2 + by_2 + c = 0$$

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \times \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$$

Cartesian Coordinates

Homogeneous Coordinates

Geometry in Homogeneous Coordinates

Intersection of two lines

$$a_1 x + b_1 y + c_1 = 0$$
$$a_2 x + b_2 y + c_2 = 0$$

Cartesian Coordinates

$$[a_1, b_1, c_1]\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0$$

$$[a_2, b_2, c_2]\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0$$

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} a_1 \\ b_1 \\ c_1 \end{bmatrix} \times \begin{bmatrix} a_2 \\ b_2 \\ c_2 \end{bmatrix}$$

Homogeneous Coordinates

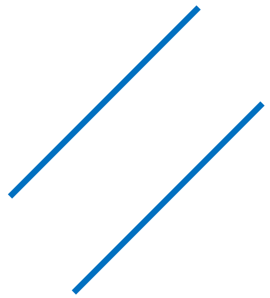
Infinity in Homogeneous Coordinates

A very interesting property of homogeneous coordinates is to represent points at infinity

A point at infinity is $[x, y, 0]^T$

A line at infinity is $[0,0, c]^T$

Two parallel lines will intersect at an infinity point



E.g., Lines $[1,1,1]^T$ and $[1,1,-1]^T$ will intersect at [-2, 2, 0]

Parallel Lines Intersect in Projective Space

https://www.songho.ca/math/homogeneous/homogeneous.html

Projection Matrix (Intrinsic)

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

$$\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} fx \\ fy \\ z \end{bmatrix}$$

$\mathrm{x} \in \mathcal{R}^{3\times 1}$: 2D point (homogeneous)

$$x = K[I\ 0]X$$

$X \in \mathcal{R}^{4\times 1}$: 3D point (homogeneous)

$K \in \mathcal{R}^{3\times 3}$: Intrinsic matrix

Note that the 3D point is represented in the camera space

More General Intrinsic Matrix

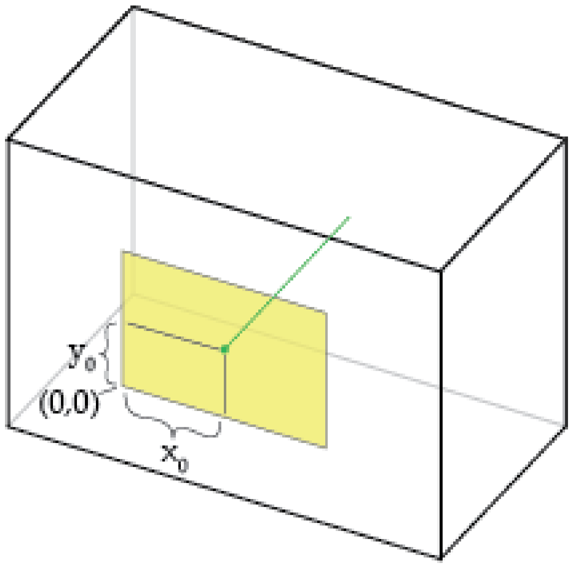
$$K = \begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}$$

A common practice is to measure the intrinsic matrix in **pixels** rather than physical units

Previous assumption:
- Unit aspect ratio
- No skew
- Principal point at (0, 0)

Principal Point Offset

$$w \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & u_0 & 0 \\ 0 & f & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

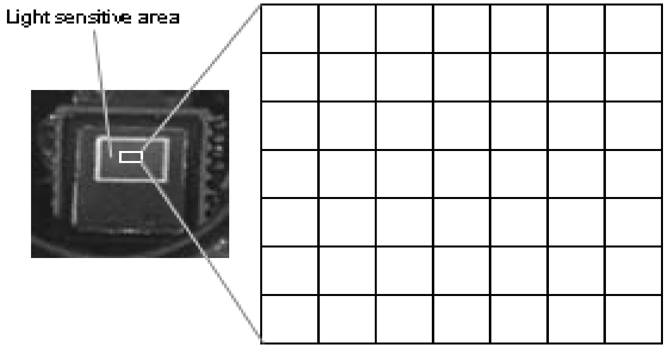


Commonly, the offset is (W/2, H/2)
so that a 3D point on the principal axis will be at the image center

Figure from blog

Focal Length and Aspect Ratio

$$w\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_0 & 0 \\ 0 & f_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



The focal lengths differ for a number of reason:

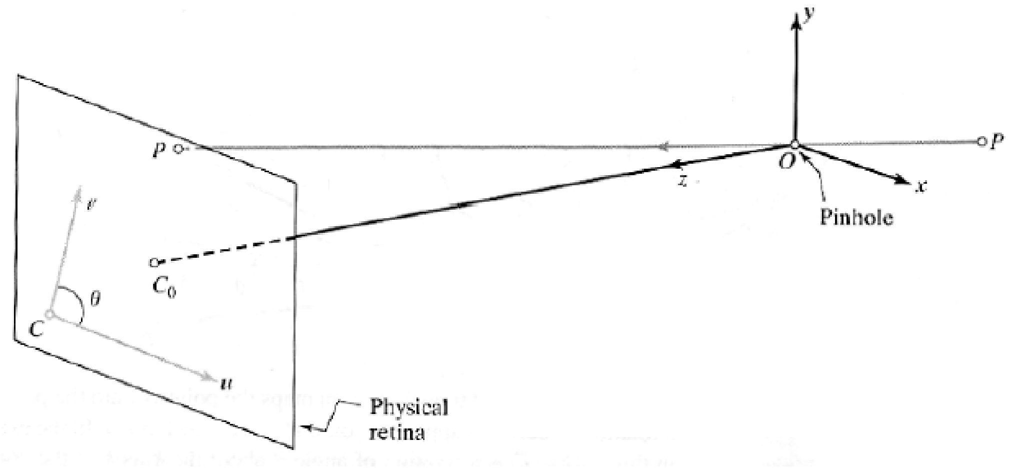
- Flaws in the digital camera sensor.
- The image has been non-uniformly scaled in post-processing.
- The camera's lens introduces unintentional distortion.
- The camera uses an anamorphic format, where the lens compresses a widescreen scene into a standard-sized sensor.
- Errors in camera calibration.

Source: Bahadir K. Gunturk

Axis Skew

$$w \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & -f_x \cot\theta & u_0 & 0 \\ 0 & f_y / \sin\theta & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$w \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & u_0 & 0 \\ 0 & f_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



Due to manufacturing error

Source: Bahadir K. Gunturk

Summary: Intrinsic Matrix

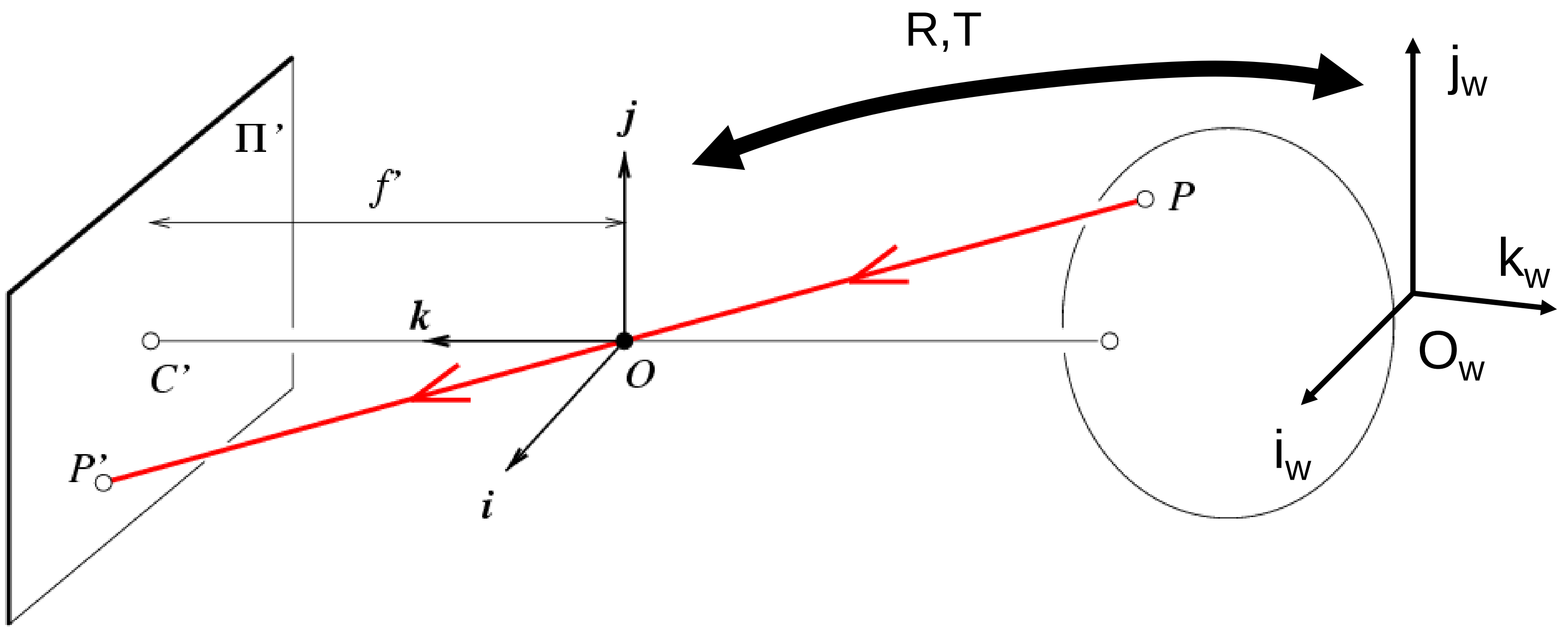
$$K = \begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \underbrace{\begin{pmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Translation}} \times \underbrace{\begin{pmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Scaling}} \times \underbrace{\begin{pmatrix} 1 & s/f_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Shear}}$$

5 parameters:
- Focal length: fx, fy
- Skew: s
- Principal point offset: (x0, y0)

There are other ways to parameterize, like FOV (field of view)

Camera Extrinsic

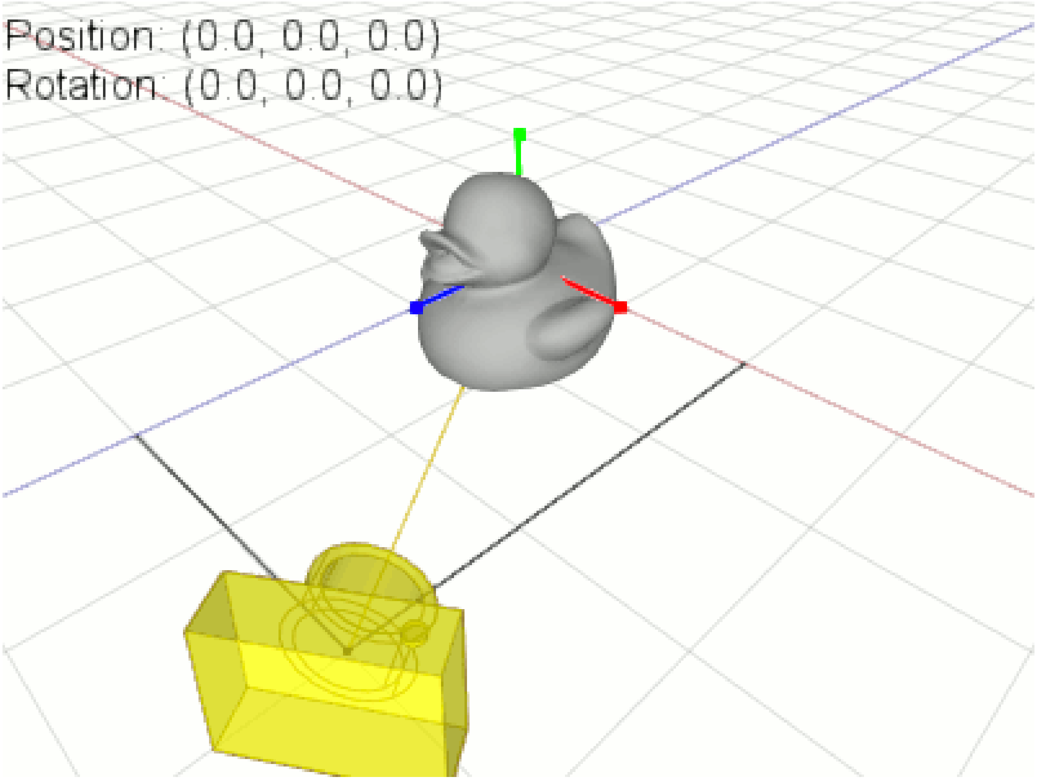


Previously, coordinates are in camera space

Now, we want to record coordinates in **world space** (convenient for multiple cameras)

Source: Derek Hoiem

Camera Extrinsic



Transform the world coordinate system to the camera coordinate system
so that the camera frame aligns with the world frame

Animation from blog

Camera Extrinsic

$$x = K[I\ 0]X_C$$

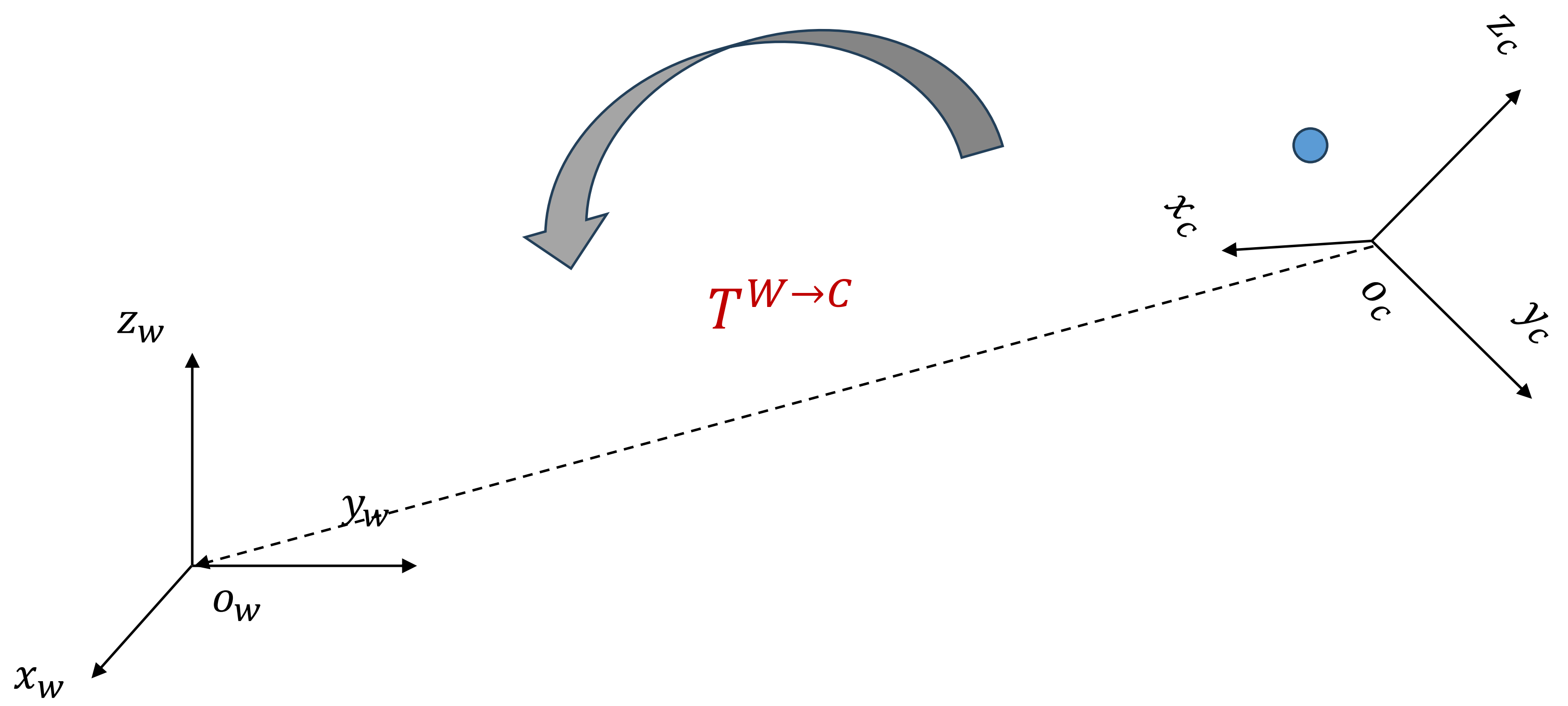
$$x = K[I\ 0]T_{W \to C}X_W$$

X_C is in the camera space

X_W is in the world space

$T_{W \to C}$ is the coordinate transformation from world to camera (the motion from camera frame to world frame)

Coordinate Transformation

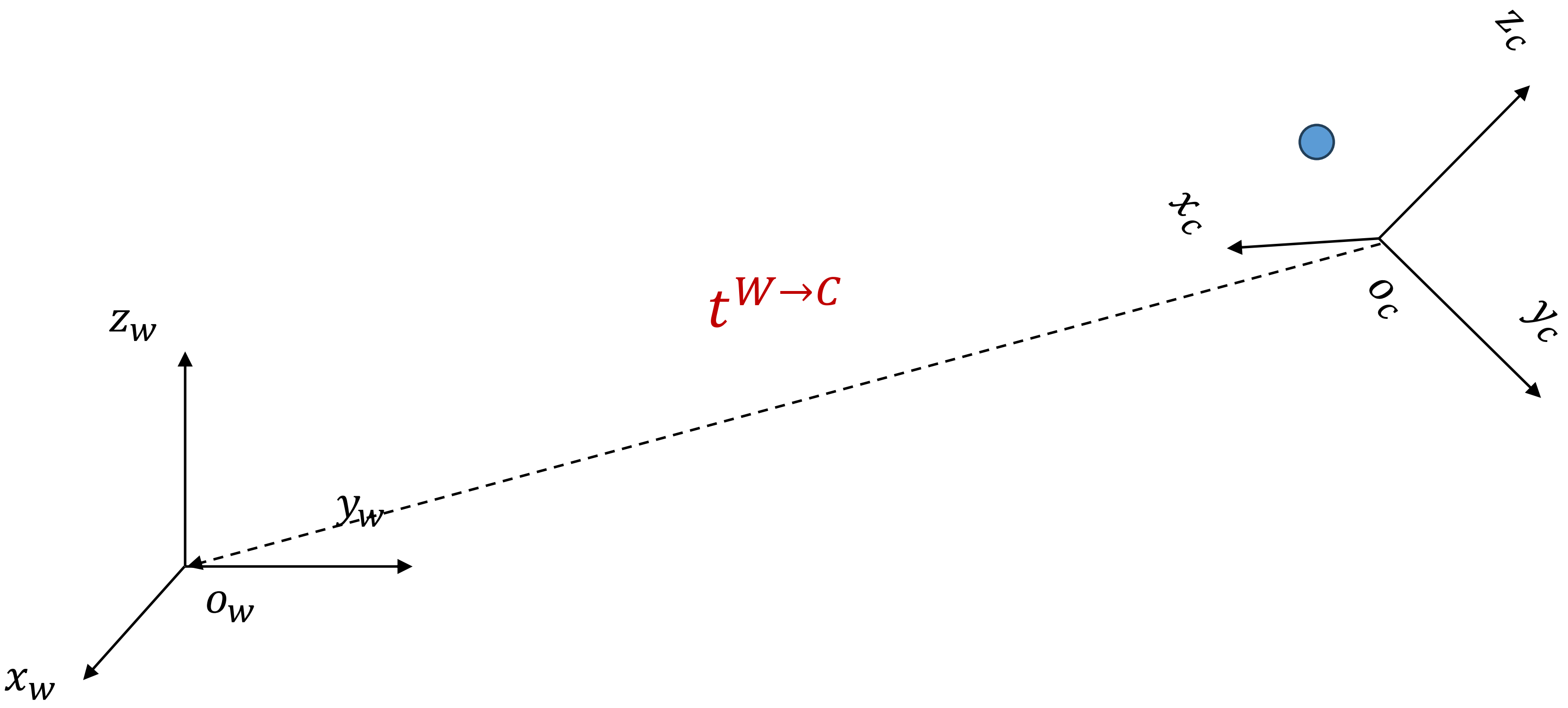


$$X_C = T_{W\to C} X_W$$

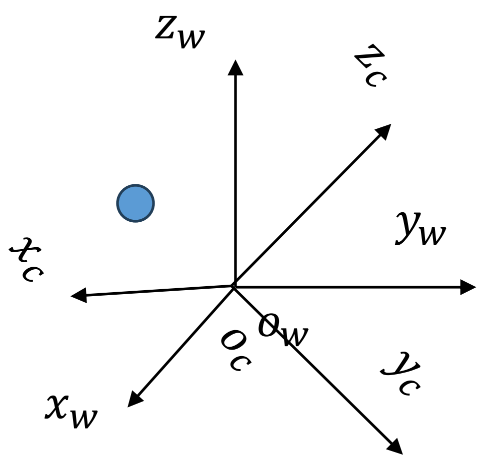
Point in camera space

Point in world space

Translation

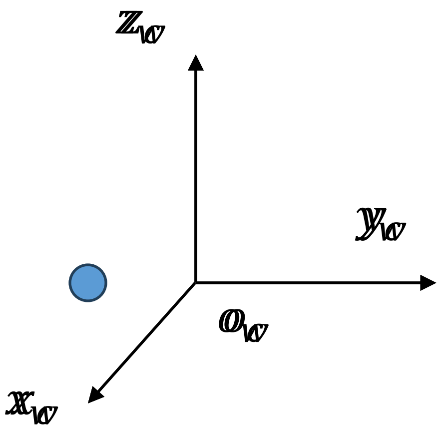


Translation



$$x^c = x^W + t^{W \to C}$$

Rotation



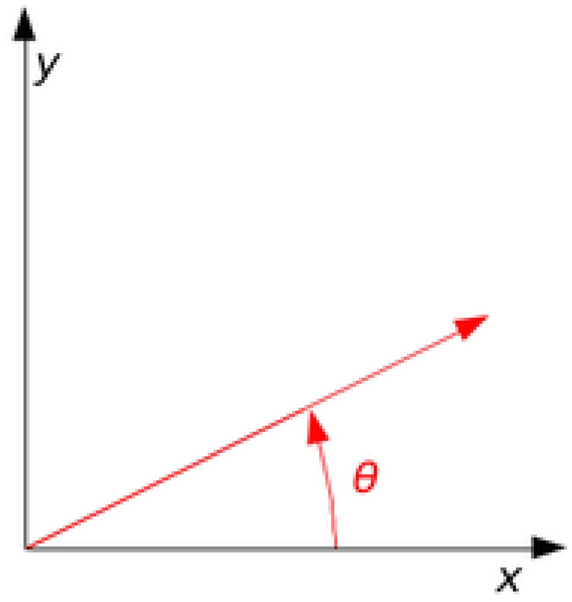
$R^{W \to C}$

$$x^c = R^{W \to C} x^W + t^{W \to C}$$

Rotation Matrix (2D)

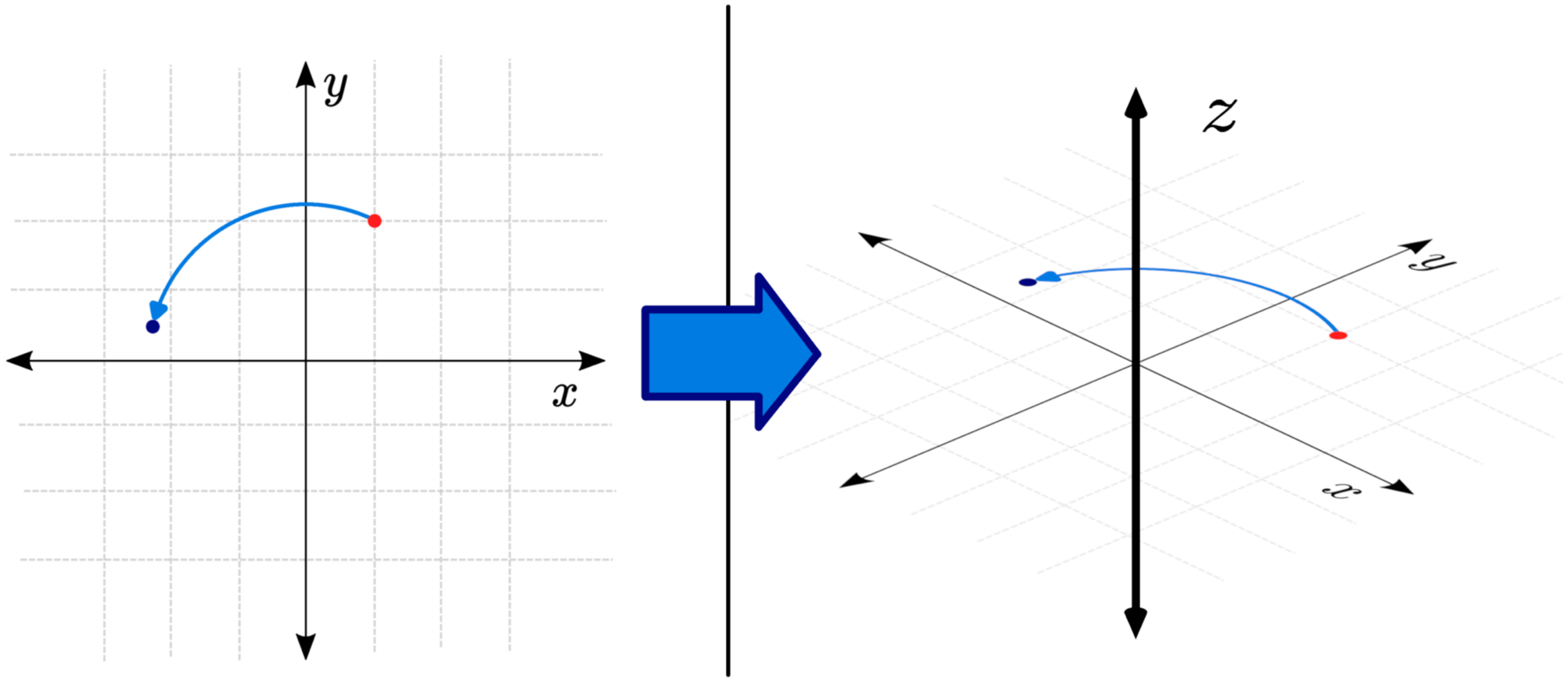
$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



From wiki

Rotation Matrix (3D)

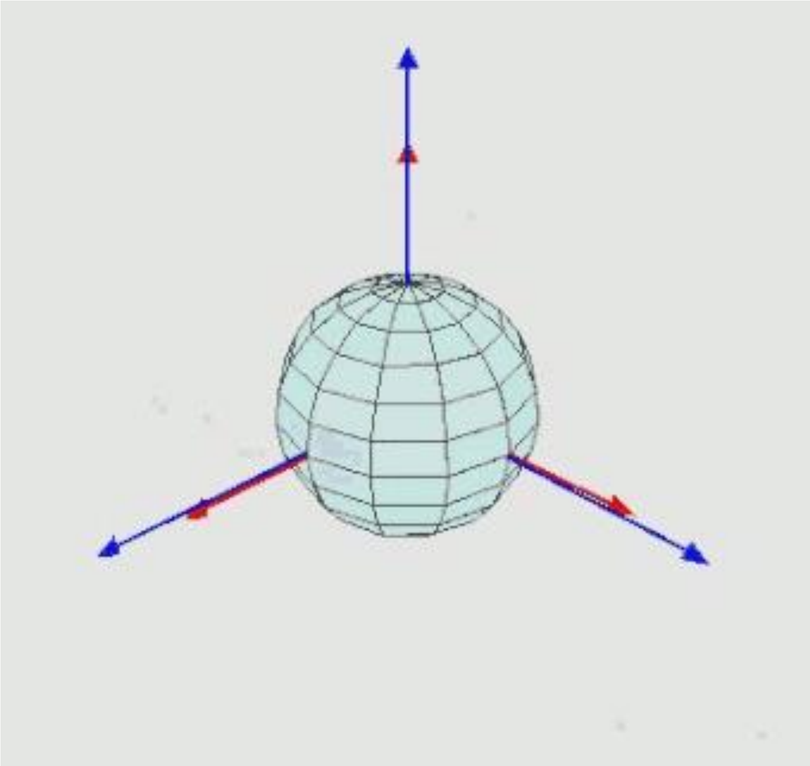


$$\begin{bmatrix} x_{new} \\ y_{new} \\ z_{new} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Example: 3D rotation along z-axis

From tutorial

Rotation Representation: Euler Angles



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

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

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

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

From [Euler Angles Wiki]

Rotation Matrix is in SO(3)

- SO(3): Special Orthogonal Group
  - “Group”: roughly, closed under matrix multiplication
  - “Orthogonal”: $R^T R = I$
  - “Special”: $\det(R) = 1$

$$\mathbb{SO}(n) = \{R \in \mathbb{R}^{n \times n} : \det(R) = 1, RR^T = I\}$$

Camera Extrinsic

$$x = K[I\ 0]X_C$$

$$x = K[I\ 0]T_{W \to C}X_W$$

X_C is in the camera space

X_W is in the world space

$T_{W \to C}$ is the coordinate transformation from world to camera

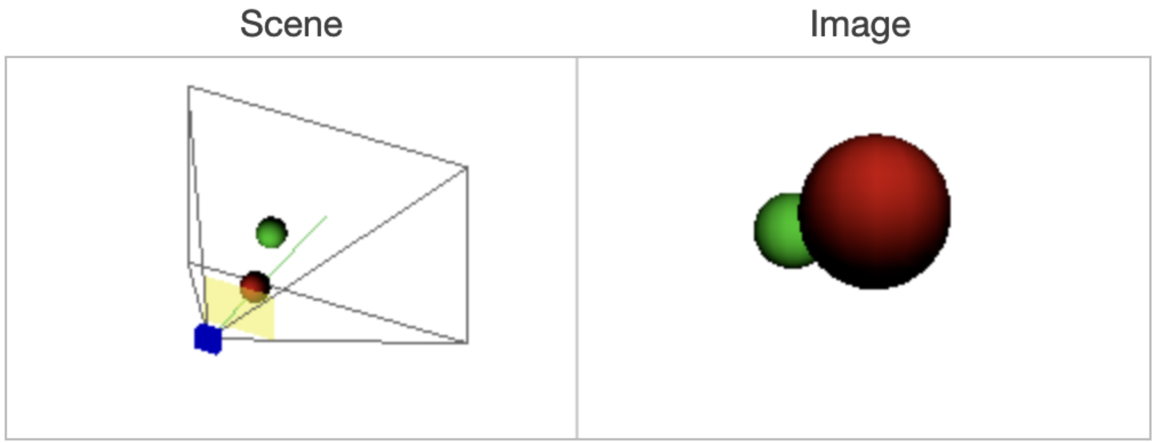
$$T_{W \to C} = \begin{bmatrix} R_{W \to C} & t_{W \to C} \\ 0 & 1 \end{bmatrix} \in \mathcal{R}^{4 \times 4}$$

Decomposition of Projection Matrix

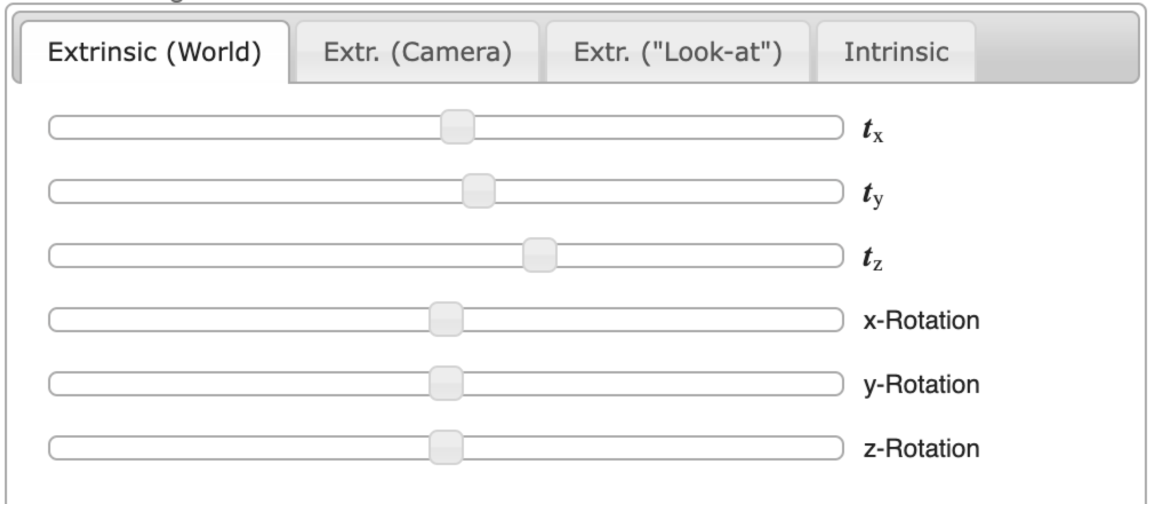
$$P = \overbrace{K}^{\text{Intrinsic Matrix}} \times \overbrace{[R \mid \mathbf{t}]}^{\text{Extrinsic Matrix}}$$

$$= \overbrace{\underbrace{\begin{pmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Translation}} \times \underbrace{\begin{pmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Scaling}} \times \underbrace{\begin{pmatrix} 1 & s/f_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Shear}}}^{\text{Intrinsic Matrix}} \times \overbrace{\underbrace{\left(I \mid \mathbf{t} \right)}_{\text{3D Translation}} \times \underbrace{\left(\begin{array}{c|c} R & 0 \\ \hline 0 & 1 \end{array} \right)}_{\text{3D Rotation}}}^{\text{Extrinsic Matrix}}$$

Demo: WebGL

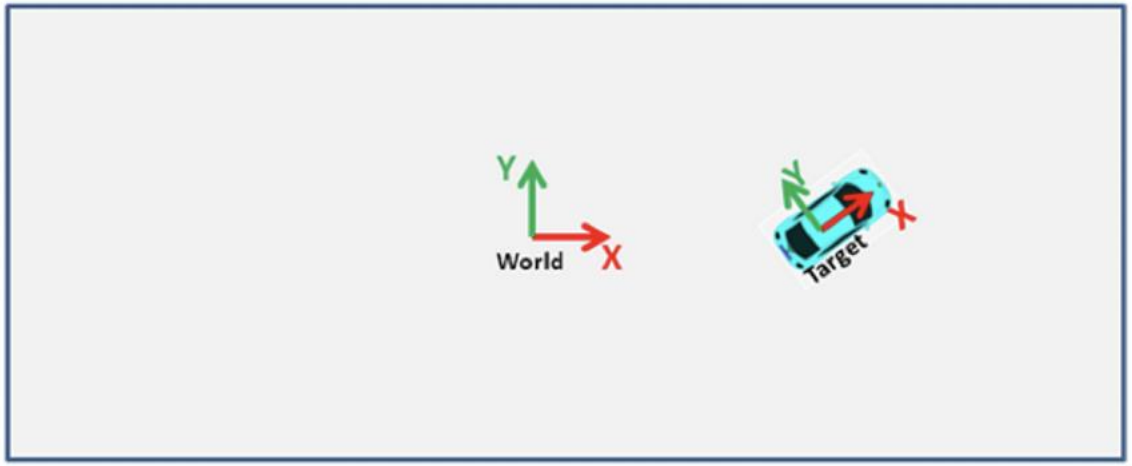


Left: scene with camera and viewing volume. Virtual image plane is shown in yellow. *Right*: camera's image.

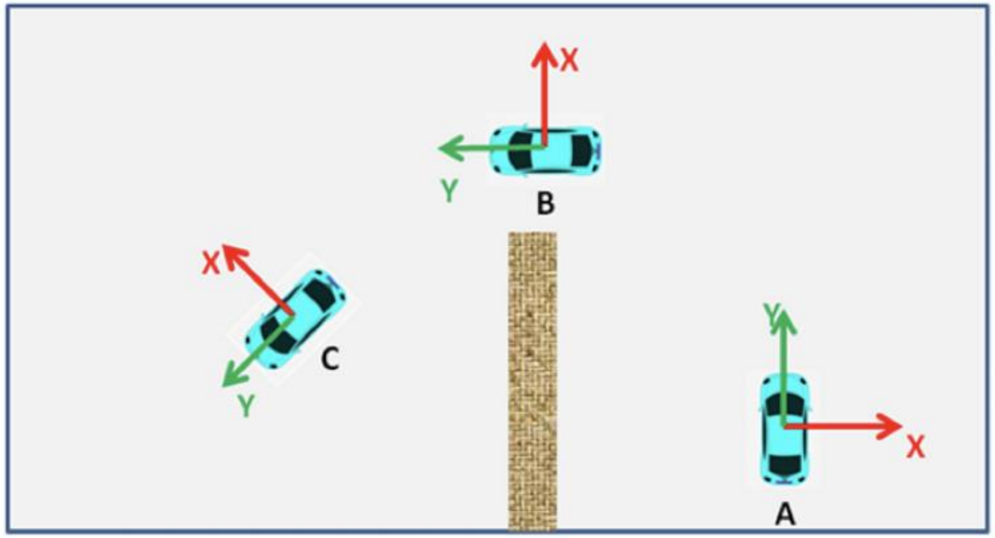


https://ksimek.github.io/2012/08/22/extrinsic/

Pose: Transformation between Frames



Where is the car in the world?



Where is the car B observed by the driver of the car A?

https://www.rosroboticslearning.com/rigid-body-transformations

Extrinsic Matrix from Camera Pose

Consider the fact:
The origin of camera coordinate system is the camera center in world coordinate system
The axis of camera coordinate system is the forward direction in world coordinate system

Thus, the camera pose is equivalent to the coordinate transformation from camera to world

$$R_C \overset{\text{def}}{=} R_{C\to W}, C \overset{\text{def}}{=} t^{C\to W}$$

We can invert $T_{C\to W}$ to compute $T_{W\to C} = T_{C\to W}^{-1}$

Inverting Rigid Transformation

$$
\left[\begin{array}{c|c} R & \boldsymbol{t} \\ \hline \mathbf{0} & 1 \end{array}\right] = \left[\begin{array}{c|c} R_c & C \\ \hline \mathbf{0} & 1 \end{array}\right]^{-1}
$$

$$
= \left[\left[\begin{array}{c|c} I & C \\ \hline \mathbf{0} & 1 \end{array}\right] \left[\begin{array}{c|c} R_c & 0 \\ \hline \mathbf{0} & 1 \end{array}\right]\right]^{-1} \qquad \text{(decomposing rigid transform)}
$$

$$
= \left[\begin{array}{c|c} R_c & 0 \\ \hline \mathbf{0} & 1 \end{array}\right]^{-1} \left[\begin{array}{c|c} I & C \\ \hline \mathbf{0} & 1 \end{array}\right]^{-1} \qquad \text{(distributing the inverse)}
$$

$$
= \left[\begin{array}{c|c} R_c^T & 0 \\ \hline \mathbf{0} & 1 \end{array}\right] \left[\begin{array}{c|c} I & -C \\ \hline \mathbf{0} & 1 \end{array}\right] \qquad \text{(applying the inverse)}
$$

$$
= \left[\begin{array}{c|c} R_c^T & -R_c^T C \\ \hline \mathbf{0} & 1 \end{array}\right] \qquad \text{(matrix multiplication)}
$$

Animation from blog

Camera Calibration

$$x = PX = K[R\ t]X$$

equal sign under homogeneous coordinates

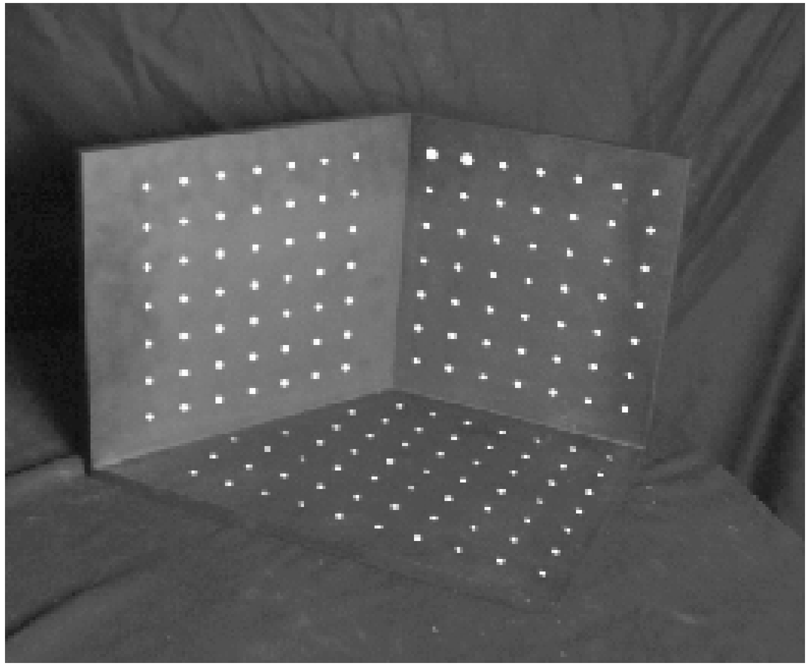
$$\begin{bmatrix} wu \\ wv \\ w \end{bmatrix} = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

How many degrees of freedom (DoF)?

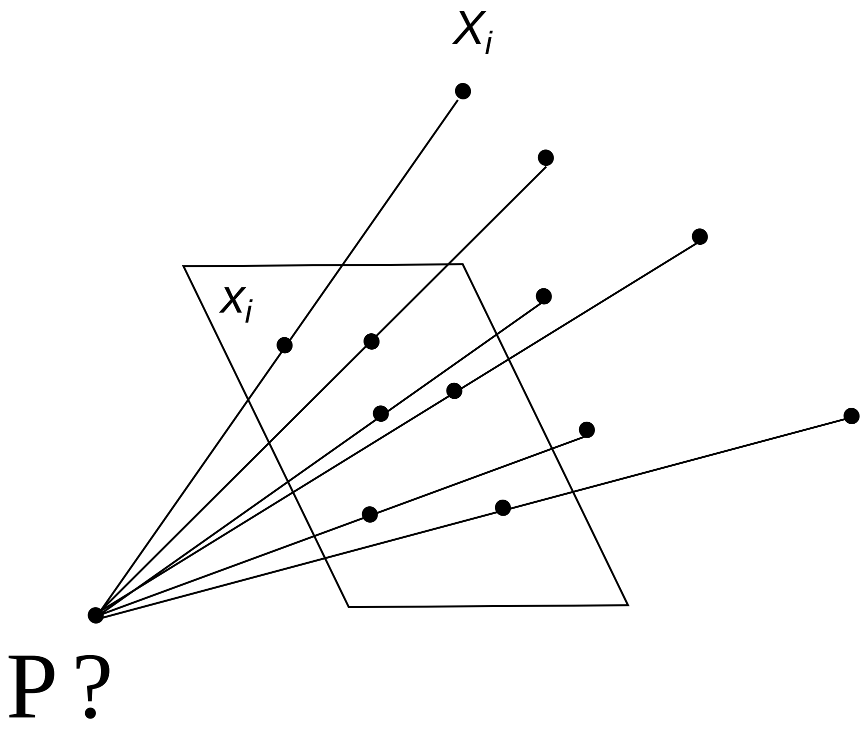
Source: Derek Hoiem

Camera Calibration

- Given n points with known 3D coordinates X_i and known image projections x_i, estimate the camera parameters



A calibration grid with known geometry



Camera Calibration

$$\begin{bmatrix} wu \\ wv \\ w \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Known Unknown Known

One pair of corresponding 3D point and 2D pixel provides an equation (2 constraints)

Direct Linear Transformation (DLT)

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -u_1X_1 & -u_1Y_1 & -u_1Z_1 & -u_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -v_1X_1 & -v_1Y_1 & -v_1Z_1 & -v_1 \\ & & & & & & \vdots & & & & & \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -u_nX_n & -u_nY_n & -u_nZ_n & -u_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -v_nX_n & -v_nY_n & -v_nZ_n & -v_n \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \\ m_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

2n x 12

12 x1

Reorganize the equation with the format Ax=0

Least Square Solution

$$\min_x x^T A^T A x \text{, s.t. } ||\mathrm{x}|| = 1$$

Least square objective

$$L(x, \lambda) = x^T A^T A x - \lambda(x^T x - 1)$$

Lagrange multiplier

$$\frac{\partial L}{\partial x} = 2A^T A x - 2\lambda x = 0$$

$$A^T A x = \lambda x$$

x is the eigenvector with the smallest eigenvalue of $A^T A$

Lagrange Multiplier

The basic idea is to convert a constrained problem into a form such that the derivative test of an unconstrained problem can still be applied.

$$\min_x f(x), s.t.\ g(x) = 0$$

constrained

$$\Downarrow$$

$$\mathcal{L}(x, \lambda) \triangleq f(x) + \lambda g(x)$$

unconstrained

$$\frac{\partial \mathcal{L}(x, \lambda)}{\partial x} = 0, \frac{\partial \mathcal{L}(x, \lambda)}{\partial \lambda} = 0$$

Refer to [wiki]

Pros and Cons of DLT

- Pros:
  - Easy to formulate and solve
  - No need for any initialization, which non-linear methods need

- Cons
  - Projection matrix is got, but decomposition is not
  - Can not impose constraints (like known focal length)
  - Does not minimize projection error

Decomposing Projection Matrix

$$P = K[R\ t] \equiv [KR\ Kt]$$

$M \stackrel{\text{def}}{=} KR$, K is upper triangle and R is SO(3)

We can use RQ decomposition to find K, R

QR Decomposition

Any real square matrix A may be decomposed as

$$A = QR,$$

where Q is an orthogonal matrix (its columns are orthogonal unit vectors meaning $Q^\mathsf{T} = Q^{-1}$) and R is an upper triangular matrix (also called right triangular matrix). If A is invertible, then the factorization is unique if we require the diagonal elements of R to be positive.

$$A = QR$$

$$A = [a_1, a_2, \dots, a_n]$$

where:

$$Q = \begin{bmatrix} \mathbf{e}_1 & \cdots & \mathbf{e}_n \end{bmatrix}$$

and

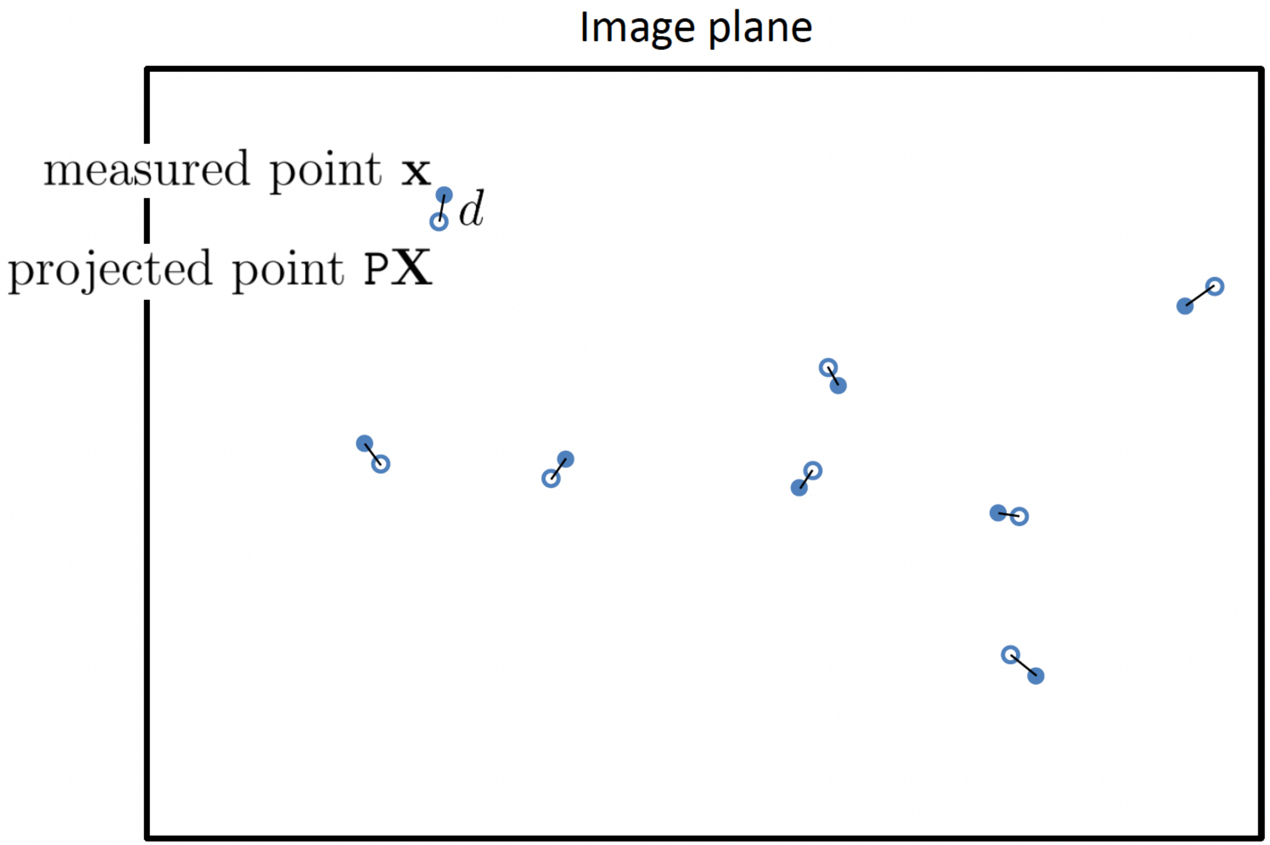
$$R = \begin{bmatrix} \langle \mathbf{e}_1, \mathbf{a}_1 \rangle & \langle \mathbf{e}_1, \mathbf{a}_2 \rangle & \langle \mathbf{e}_1, \mathbf{a}_3 \rangle & \cdots & \langle \mathbf{e}_1, \mathbf{a}_n \rangle \\ 0 & \langle \mathbf{e}_2, \mathbf{a}_2 \rangle & \langle \mathbf{e}_2, \mathbf{a}_3 \rangle & \cdots & \langle \mathbf{e}_2, \mathbf{a}_n \rangle \\ 0 & 0 & \langle \mathbf{e}_3, \mathbf{a}_3 \rangle & \cdots & \langle \mathbf{e}_3, \mathbf{a}_n \rangle \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \langle \mathbf{e}_n, \mathbf{a}_n \rangle \end{bmatrix}$$

Using the Gram–Schmidt process

Refer to wiki

Re-Projection Error

- Error in image point measurements



Linear projection of points in homogeneous coordinates

$$\mathbf{x} = \mathtt{P}\mathbf{X}$$

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ T \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} Xp_{11} + Yp_{12} + Zp_{13} + Tp_{14} \\ Xp_{21} + Yp_{22} + Zp_{23} + Tp_{24} \\ Xp_{31} + Yp_{32} + Zp_{33} + Tp_{34} \end{bmatrix}$$

Nonlinear projection of points in inhomogeneous coordinates

$$\tilde{x} = \frac{x}{w} = \frac{Xp_{11} + Yp_{12} + Zp_{13} + Tp_{14}}{Xp_{31} + Yp_{32} + Zp_{33} + Tp_{34}} \text{ and}$$
$$\tilde{y} = \frac{y}{w} = \frac{Xp_{21} + Yp_{22} + Zp_{23} + Tp_{24}}{Xp_{31} + Yp_{32} + Zp_{33} + Tp_{34}}$$

Do not forget this is a nonlinear projection

Non-Linear Optimization for Camera Calibration

Optimization problem

- Measurement vector x (2D pixels)
- Parameter vector P (projection matrix)
- Non-linear mapping $f: P \mapsto x$
- Note that f actually involves 3D points X: $x = f_X(P)$

An example objective to minimize projection error

$$\min_P \left|\left| f_X(P) - x \right|\right|$$

Nonlinear estimation using iterative estimation methods

Measurement vector $\mathbf{X}$ Not 3D point

Parameter vector $\mathbf{P}$ Not camera projection matrix

Nonlinear function f, where $\mathbf{X} = f(\mathbf{P})$

Given an estimate of the parameter vector $\hat{\mathbf{P}}$ and assuming f is approximately linear in the region about $\hat{\mathbf{P}}$

$$\mathbf{X} \approx f(\hat{\mathbf{P}}) + \frac{\partial \hat{\mathbf{X}}}{\partial \hat{\mathbf{P}}}(\mathbf{P} - \hat{\mathbf{P}})$$

$$\mathbf{X} \approx \hat{\mathbf{X}} + \mathtt{J}(\mathbf{P} - \hat{\mathbf{P}}), \text{ where } \hat{\mathbf{X}} = f(\hat{\mathbf{P}}) \text{ and } \mathtt{J} = \frac{\partial \hat{\mathbf{X}}}{\partial \hat{\mathbf{P}}}$$

$$\mathbf{X} - \hat{\mathbf{X}} \approx \mathtt{J}(\mathbf{P} - \hat{\mathbf{P}})$$

$$\boldsymbol{\epsilon} \approx \mathtt{J}\boldsymbol{\delta}, \text{ solve for } \boldsymbol{\delta} \text{ where } \boldsymbol{\epsilon} = \mathbf{X} - \hat{\mathbf{X}} \text{ and } \boldsymbol{\delta} = \mathbf{P} - \hat{\mathbf{P}}$$

Then

$$\boldsymbol{\delta} = \mathbf{P} - \hat{\mathbf{P}}$$

$$\hat{\mathbf{P}} + \boldsymbol{\delta} = \mathbf{P}$$

Adjustment

Perform again (i.e., iterate) using resulting $\mathbf{P}$ as $\hat{\mathbf{P}}$

Nonlinear estimation using the Levenberg-Marquardt algorithm

Given

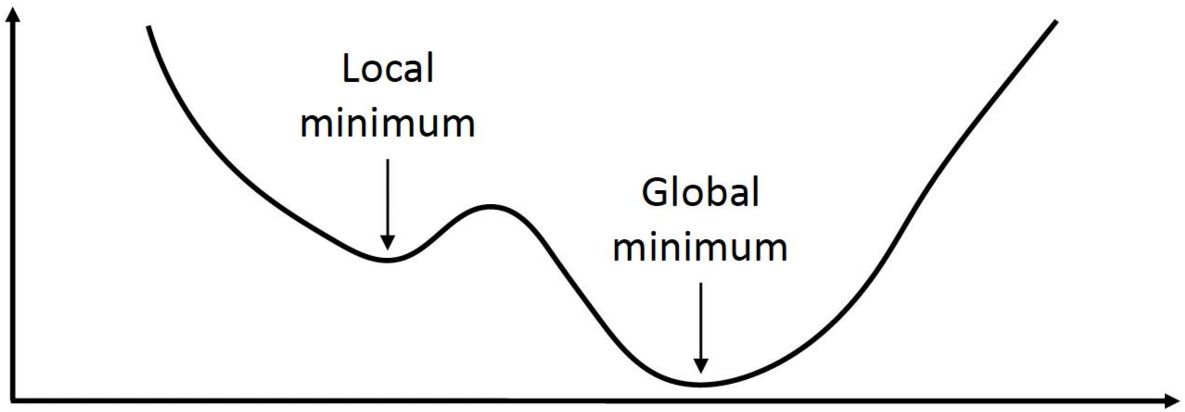
Measurement vector $\mathbf{X}$ with associated covariance matrix $\Sigma_{\mathbf{X}}$

Parameter vector $\hat{\mathbf{P}}$ (initial estimate)

Nonlinear function f, where $\hat{\mathbf{X}} = f(\hat{\mathbf{P}})$

Objective

Find $\hat{\mathbf{P}}$ that minimizes $\boldsymbol{\epsilon}^\top \Sigma_{\mathbf{X}}^{-1} \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} = \mathbf{X} - \hat{\mathbf{X}}$



Nonlinear estimation using the Levenberg-Marquardt algorithm

Algorithm

1. $\lambda = 0.001$ and $\boldsymbol{\epsilon} = \mathbf{X} - \hat{\mathbf{X}}$
2. Jacobian $\mathtt{J} = \dfrac{\partial \hat{\mathbf{X}}}{\partial \hat{\mathbf{P}}}$
3. Weighted least-squares normal equations $\left(\mathtt{J}^\top \Sigma_{\mathbf{X}}^{-1} \mathtt{J}\right) \boldsymbol{\delta} = \mathtt{J}^\top \Sigma_{\mathbf{X}}^{-1} \boldsymbol{\epsilon}$
4. Augmented normal equations $\left(\mathtt{J}^\top \Sigma_{\mathbf{X}}^{-1} \mathtt{J} + \lambda \mathtt{I}\right) \boldsymbol{\delta} = \mathtt{J}^\top \Sigma_{\mathbf{X}}^{-1} \boldsymbol{\epsilon}$, solve for $\boldsymbol{\delta}$
5. Candidate parameter vector $\hat{\mathbf{P}}_0 = \hat{\mathbf{P}} + \boldsymbol{\delta}$ — Adjustment
6. $\hat{\mathbf{X}}_0 = f(\hat{\mathbf{P}}_0)$, then $\boldsymbol{\epsilon}_0 = \mathbf{X} - \hat{\mathbf{X}}_0$
7. If candidate cost $\boldsymbol{\epsilon}_0^\top \Sigma_{\mathbf{X}}^{-1} \boldsymbol{\epsilon}_0 \geq \boldsymbol{\epsilon}^\top \Sigma_{\mathbf{X}}^{-1} \boldsymbol{\epsilon}$, then

    $\lambda = 10\lambda$

    Go to step 4 (this does not count as an iteration)

   else

    $\hat{\mathbf{P}} = \hat{\mathbf{P}}_0$ and $\boldsymbol{\epsilon} = \boldsymbol{\epsilon}_0$

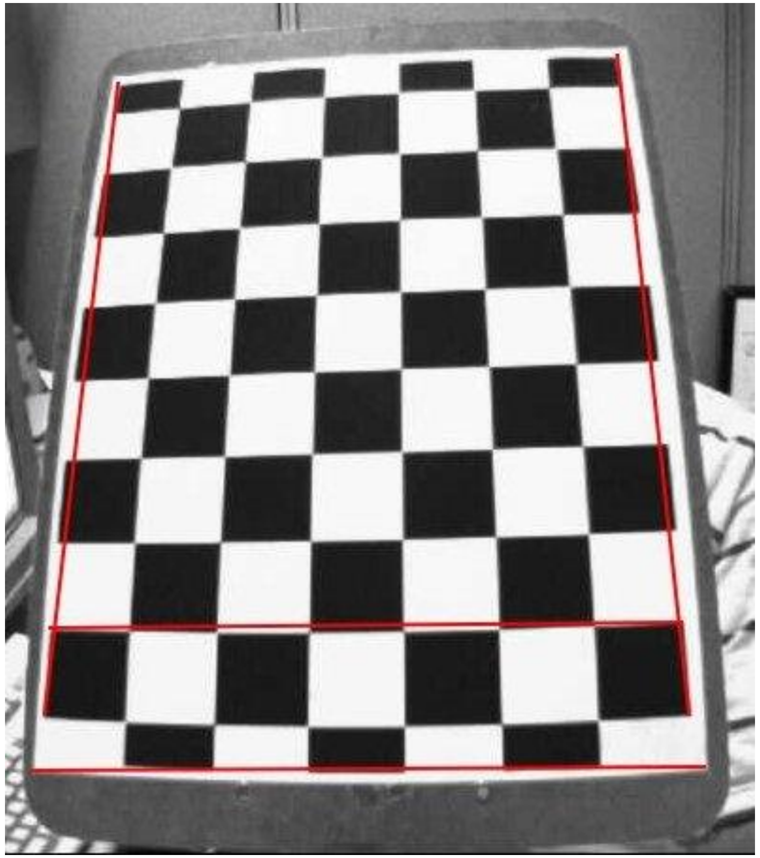
    $\lambda = \lambda/10$

    Go to step 2 or terminate (this counts as an iteration)

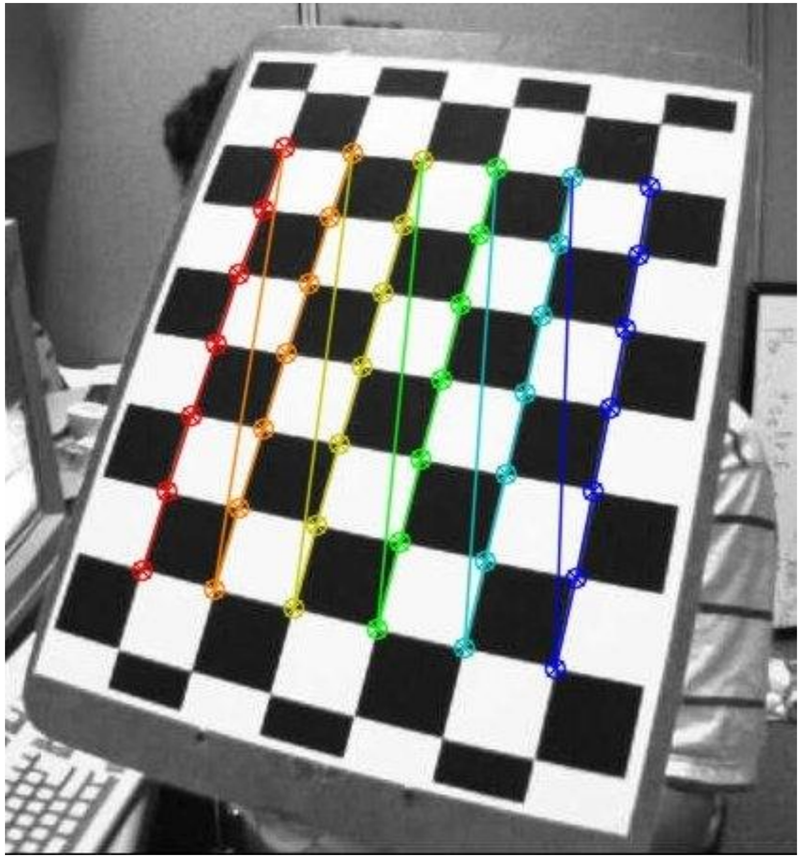
Inner loop

Outer loop

Real-World Camera Calibration



Use checkboard as the known geometry



Find the corners

From [OpenCV tutorial]

Beyond Pinhole: Radial Distortion

- Common in wide-angle lenses
- Create non-linear terms in projection
- Usually handled by solving for non-linear terms and then correcting image

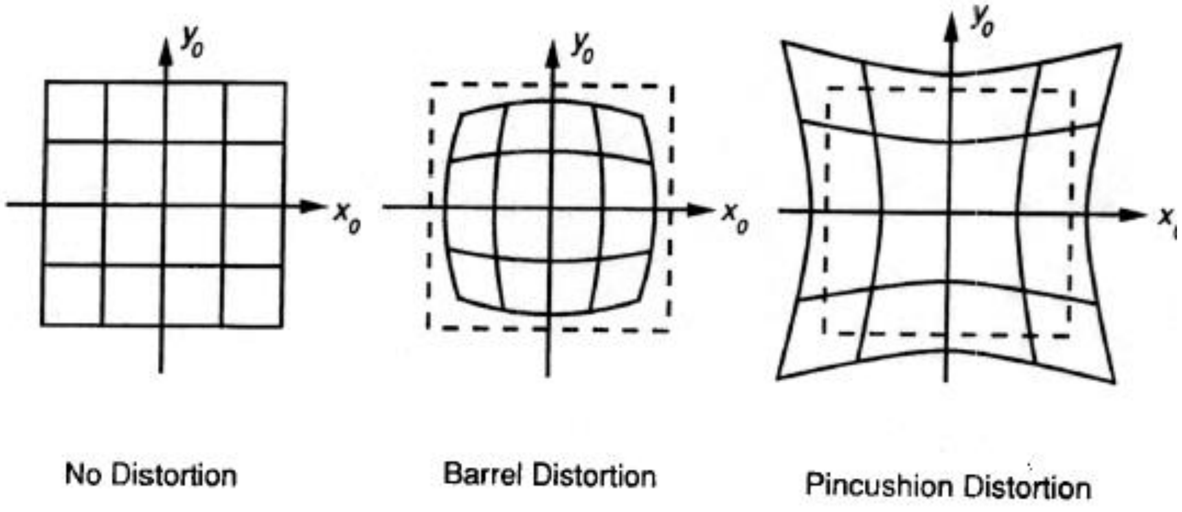


Image from Martin Habbecke

Real-World Camera Calibration

Undistort the image

From [OpenCV tutorial]

Real-World Camera Calibration

Calibration

Now that we have our object points and image points, we are ready to go for calibration. We can use the function, **cv.calibrateCamera()** which returns the camera matrix, distortion coefficients, rotation and translation vectors etc.

```
ret, mtx, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None, None)
```

The algorithm performs the following steps:

- Compute the initial intrinsic parameters (the option only available for planar calibration patterns) or read them from the input parameters. The distortion coefficients are all set to zeros initially unless some of CALIB_FIX_K? are specified.
- Estimate the initial camera pose as if the intrinsic parameters have been already known. This is done using **solvePnP** .
- Run the global Levenberg-Marquardt optimization algorithm to minimize the reprojection error, that is, the total sum of squared distances between the observed feature points imagePoints and the projected (using the current estimates for camera parameters and the poses) object points objectPoints. See **projectPoints** for details.

From OpenCV tutorial

Summary

- Homogeneous coordinates
- Projection matrix: $x = PX = K[R\ t]X$
- Find the projection matrix
  - Direct linear transformation
  - Levenberg-Marquardt optimization