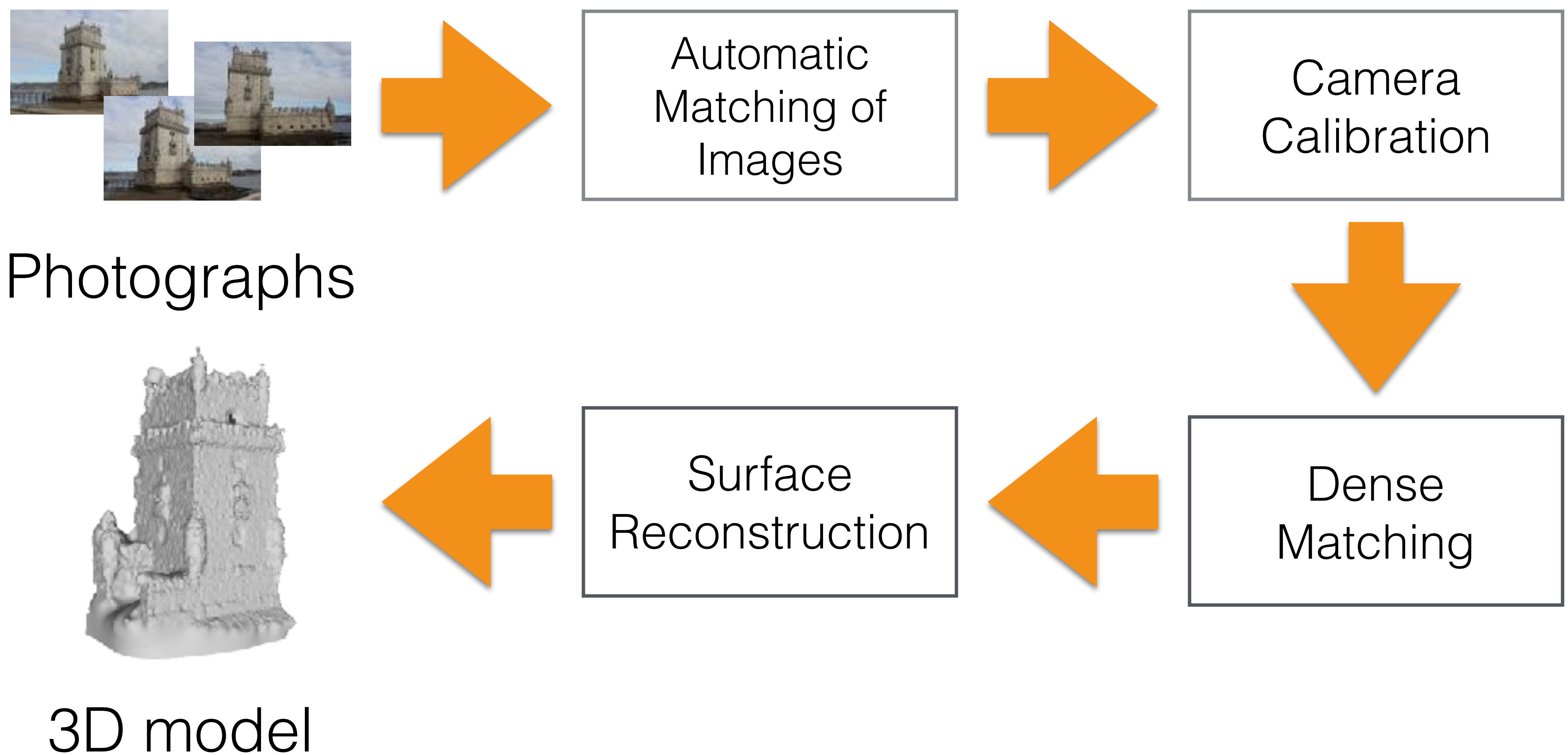
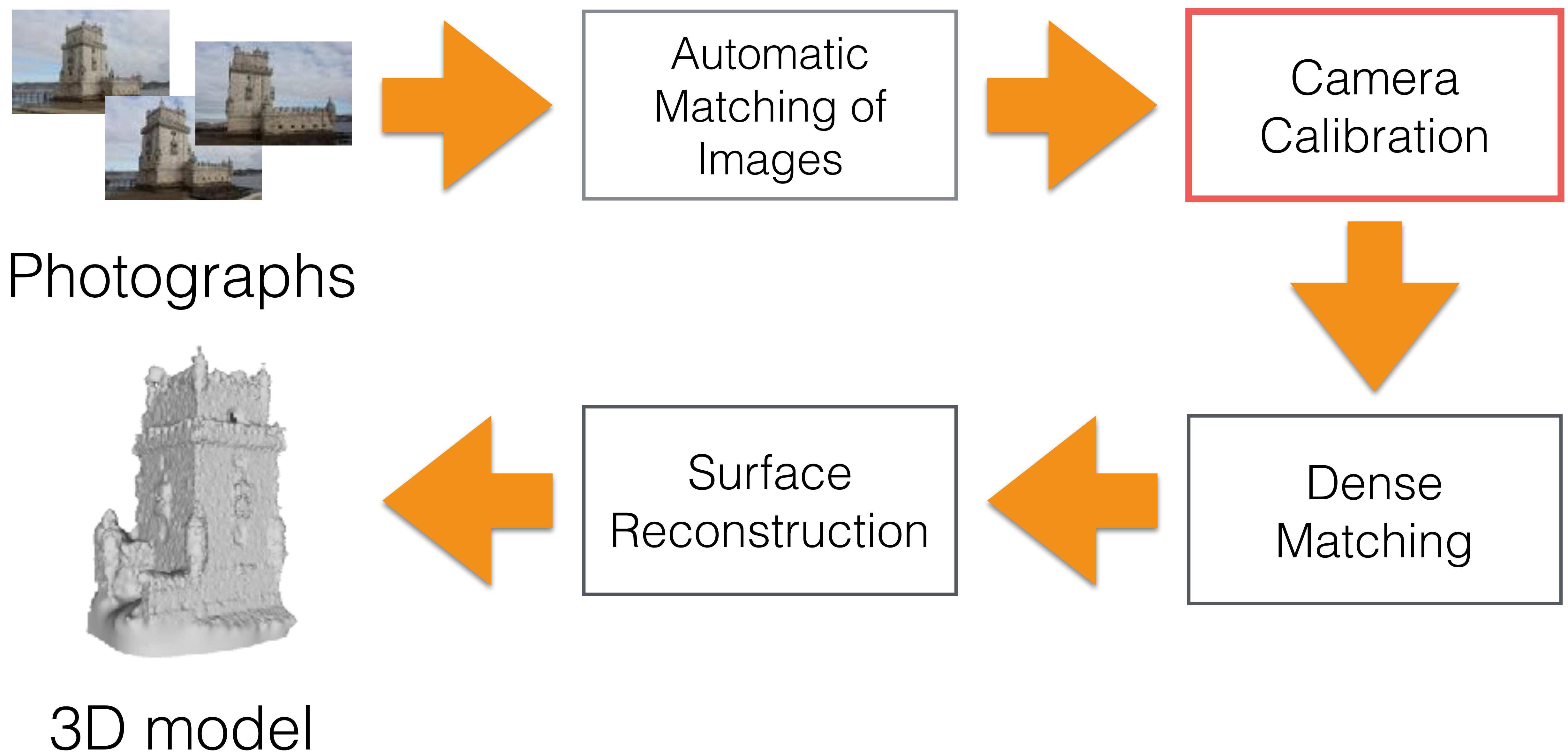
3D from Photographs: Camera Calibration

Dr Francesco Banterle
francesco.banterle@isti.cnr.it

3D from Photographs

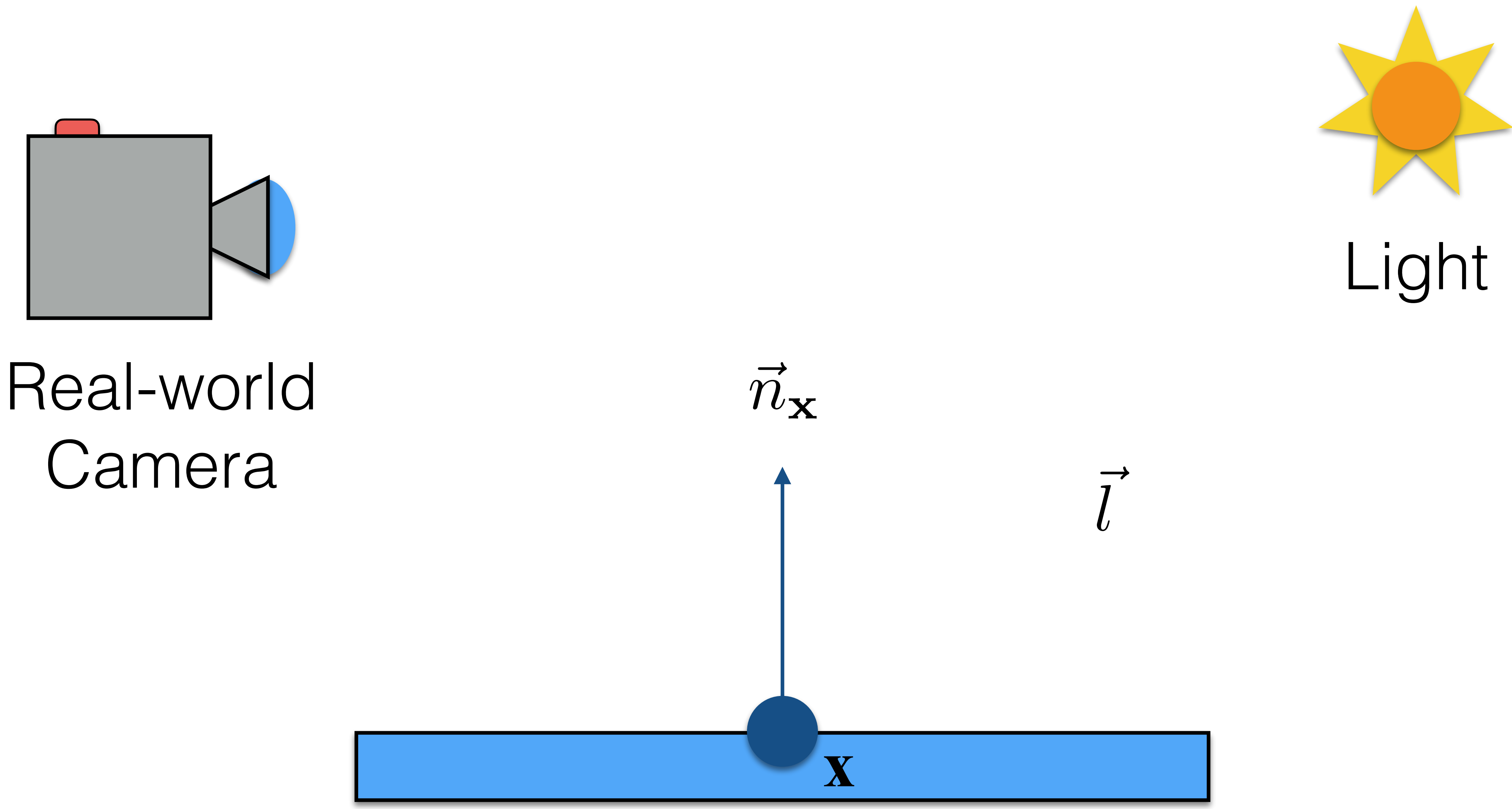


3D from Photographs

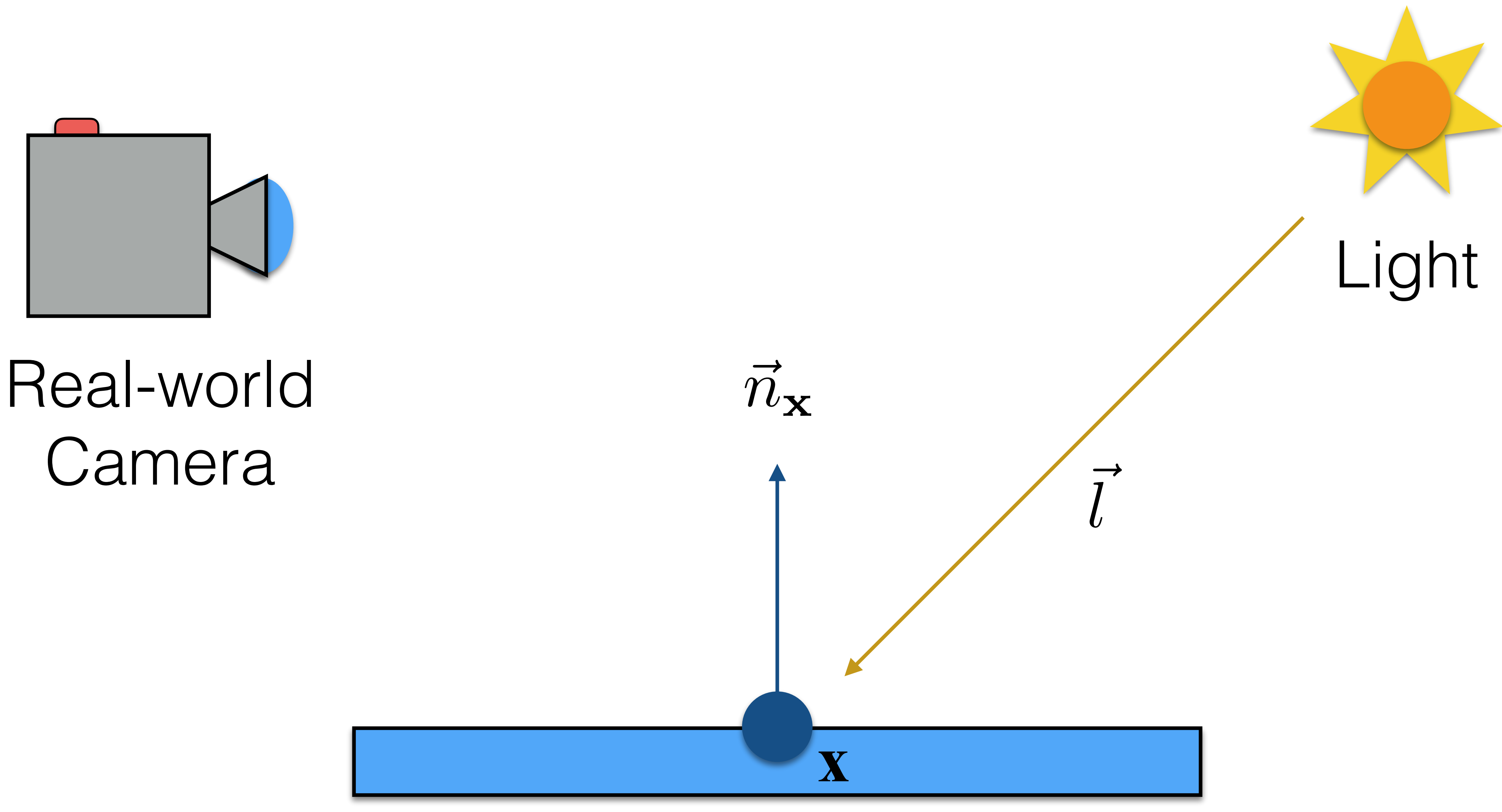


Back to the
Camera Model

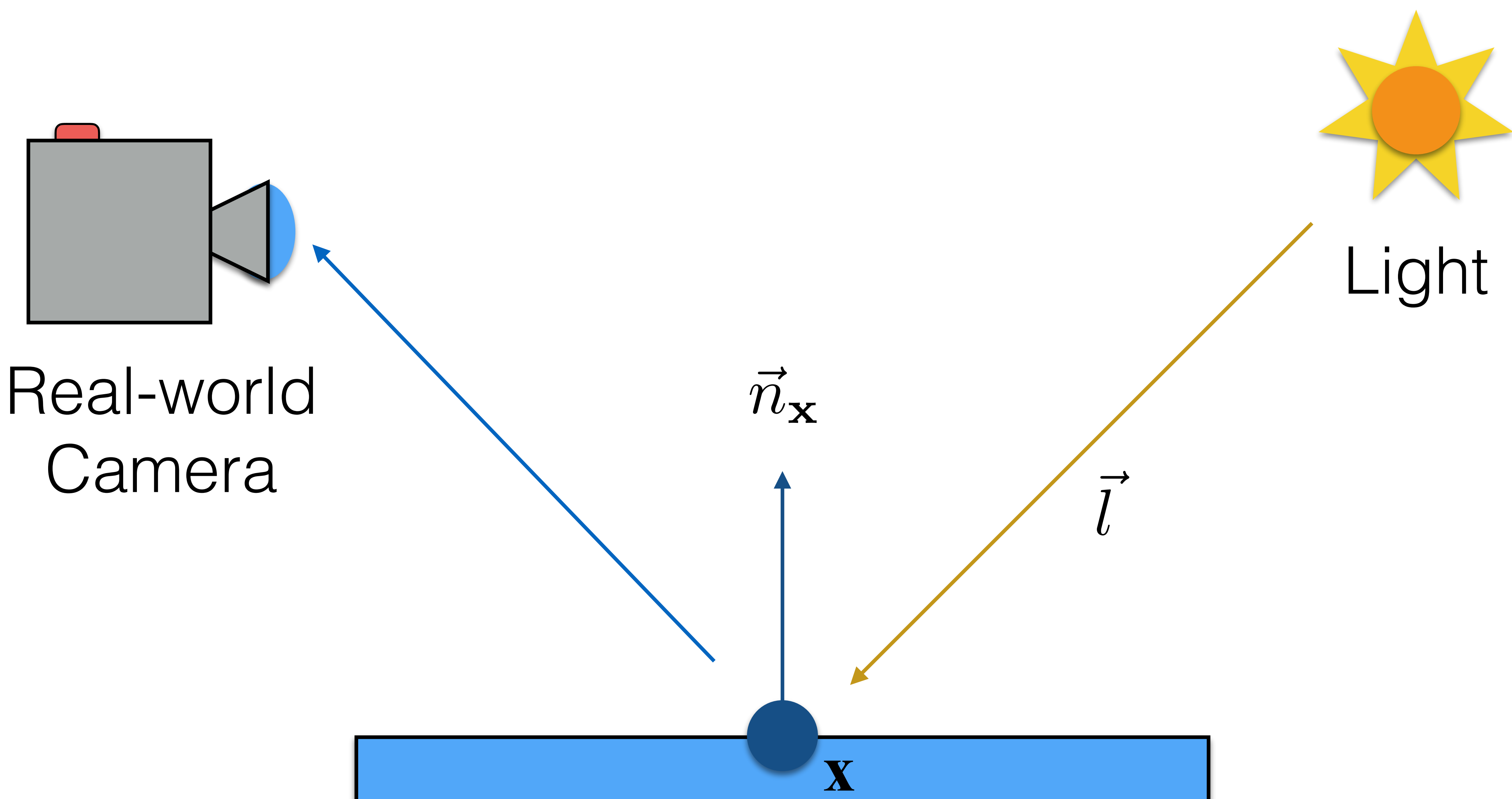
Camera Model: Image Formation



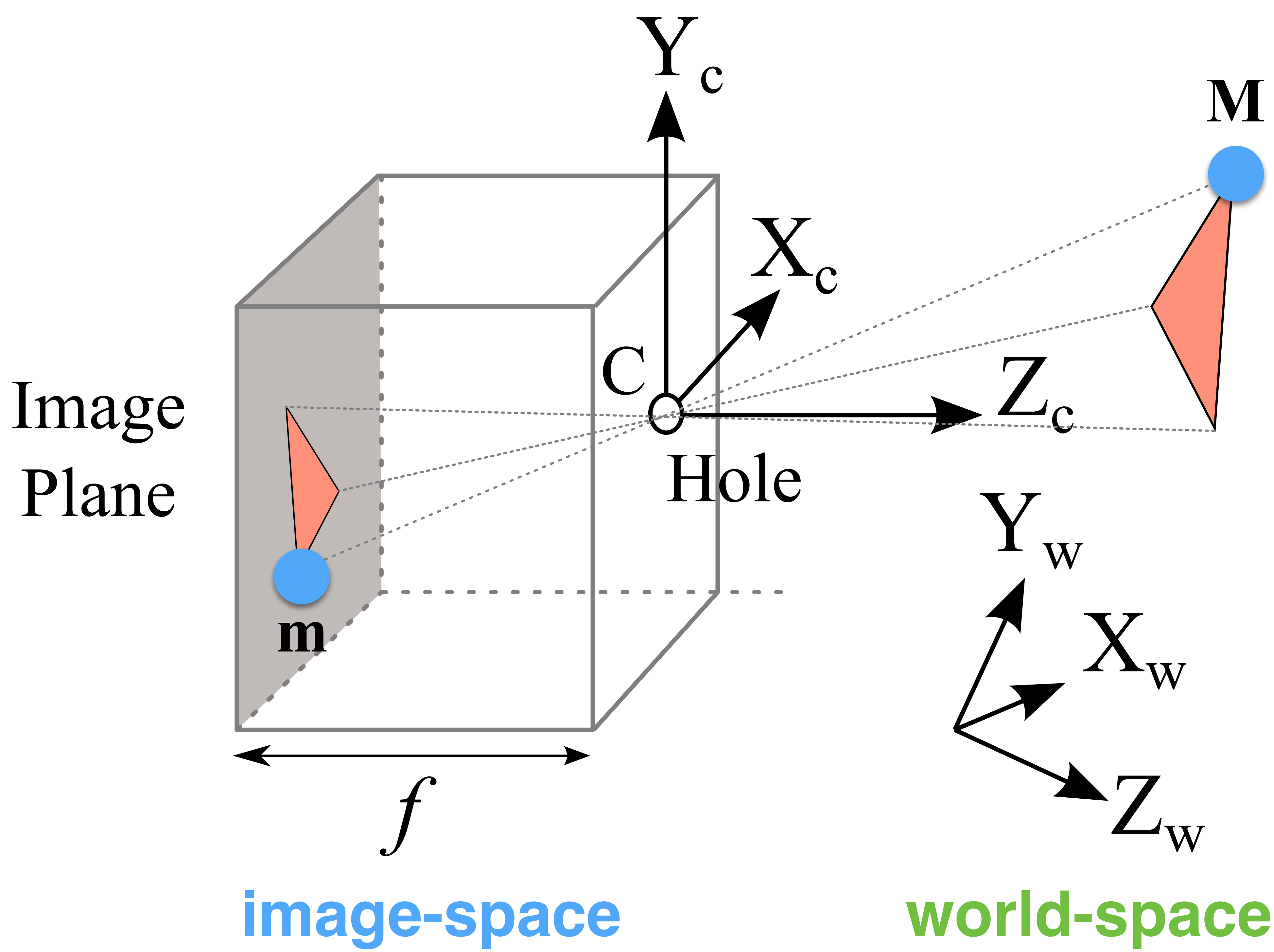
Camera Model: Image Formation



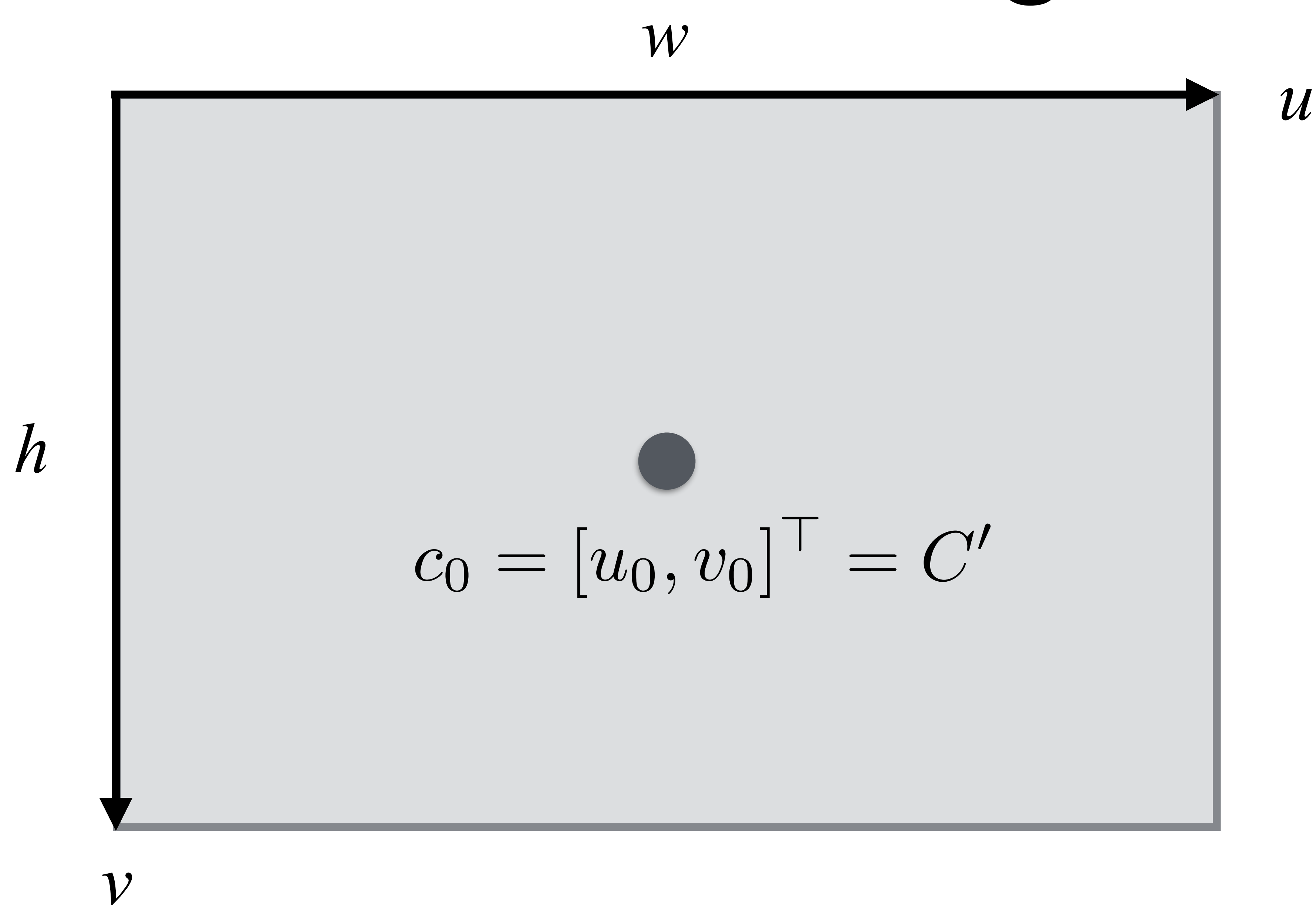
Camera Model: Image Formation



Camera Model: Pinhole Camera



Camera Model: Image Plane



- Pixels have different height and width; i.e., (k_u, k_v).
- c_0 is called the principal point.
- The image plane has a finite size: w (width) and h (height)

Camera Model

- $\mathbf{M}$ is a point in the 3D world, and it is defined as:

$$\mathbf{M} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- $\mathbf{m}$ is a 2D point, the projection of $\mathbf{M}$. $\mathbf{m}$ lives in the image plane UV:

$$\mathbf{m} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Camera Model

- By analyzing the two triangles (real-world and projected one), the following relationship emerges:

$$\frac{f}{z} = -\frac{u}{x} = -\frac{v}{y}$$

- This means that:

$$\begin{cases} u = -\frac{f}{z} \cdot x \\ v = -\frac{f}{z} \cdot y \end{cases}$$

Camera Model: Intrinsic Parameters

- If we take all into account of the optical center, and pixel size we obtain:

$$\begin{cases} u = -\frac{f}{z} \cdot x \cdot k_u + u_0 \\ v = -\frac{f}{z} \cdot y \cdot k_v + v_0 \end{cases}$$

- If we put this in matrix form, we obtain:

$$P = \begin{bmatrix} -fk_u & 0 & u_0 & 0 \\ 0 & -fk_v & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = K[I|\mathbf{0}] \qquad K = \begin{bmatrix} -fk_u & 0 & u_0 \\ 0 & -fk_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{m}z = P \cdot \mathbf{M}$$

Camera Model: Pinhole Camera

- The perspective projection is defined as:

$$\mathbf{m}z = P \cdot \mathbf{M}$$

$$P = K[I|\mathbf{0}]G = K[R|\mathbf{t}]$$

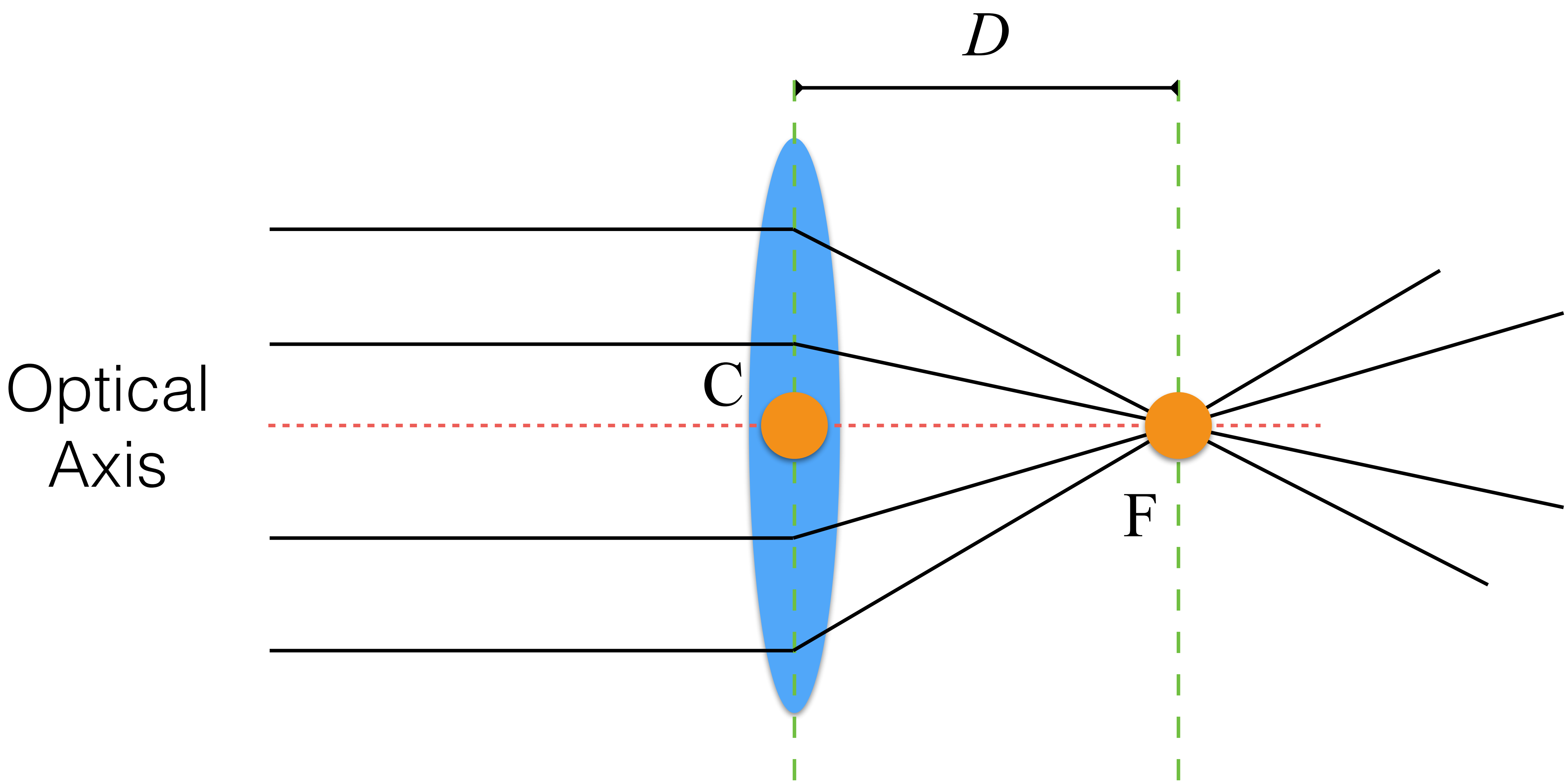
$$K = \begin{bmatrix} -fk_u & 0 & u_0 \\ 0 & -fk_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

Intrinsic Matrix

$$\mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \qquad R = \begin{bmatrix} \mathbf{r}_1^\top \\ \mathbf{r}_2^\top \\ \mathbf{r}_3^\top \end{bmatrix}$$

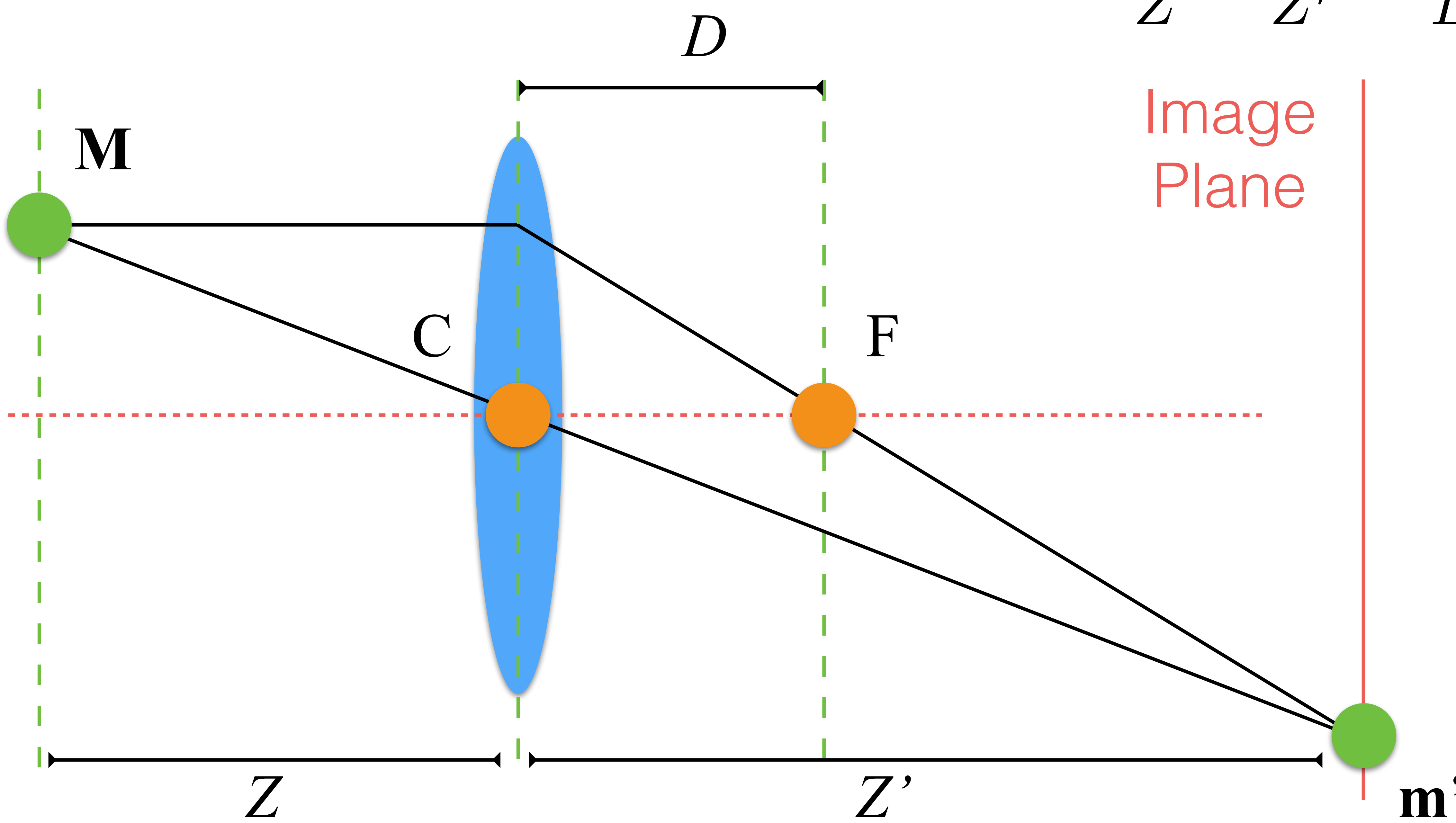
Extrinsic Matrix

Camera Model: Thin Lens



Camera Model:
Thin Lens

$$\frac{1}{Z} + \frac{1}{Z'} = \frac{1}{D}$$



Camera Model: Thin Lens

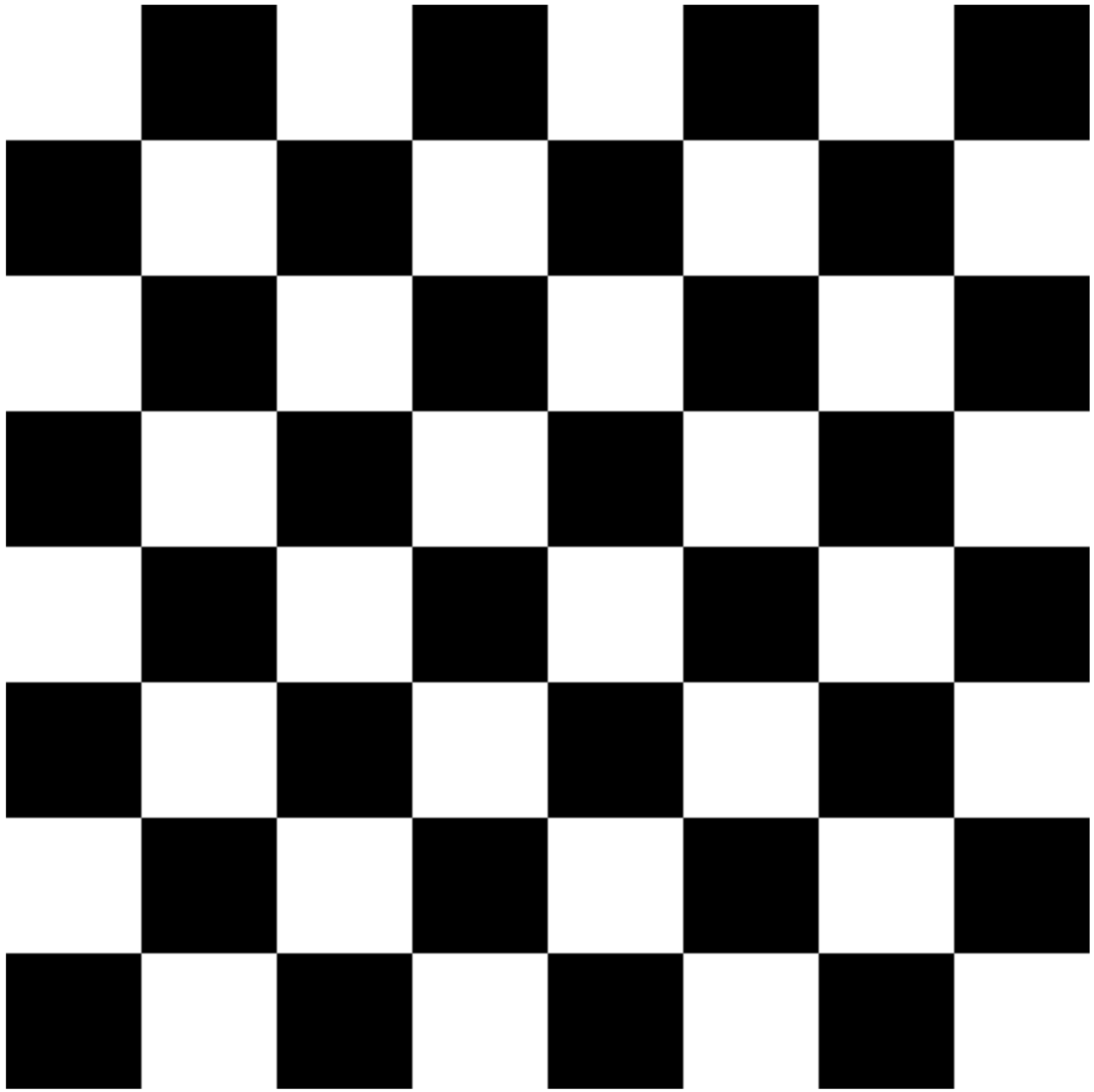
$$\mathbf{m}z = P \cdot \mathbf{M} \quad \Rightarrow \quad \mathbf{m} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

$$\begin{cases} u' = (u - u_0) \cdot (1 + k_1 r_d^2 + k_2 r_d^4 + \ldots + k_n r_d^{2n}) + u_0 \\ v' = (v - v_0) \cdot (1 + k_1 r_d^2 + k_2 r_d^4 + \ldots + k_n r_d^{2n}) + v_0 \end{cases}$$

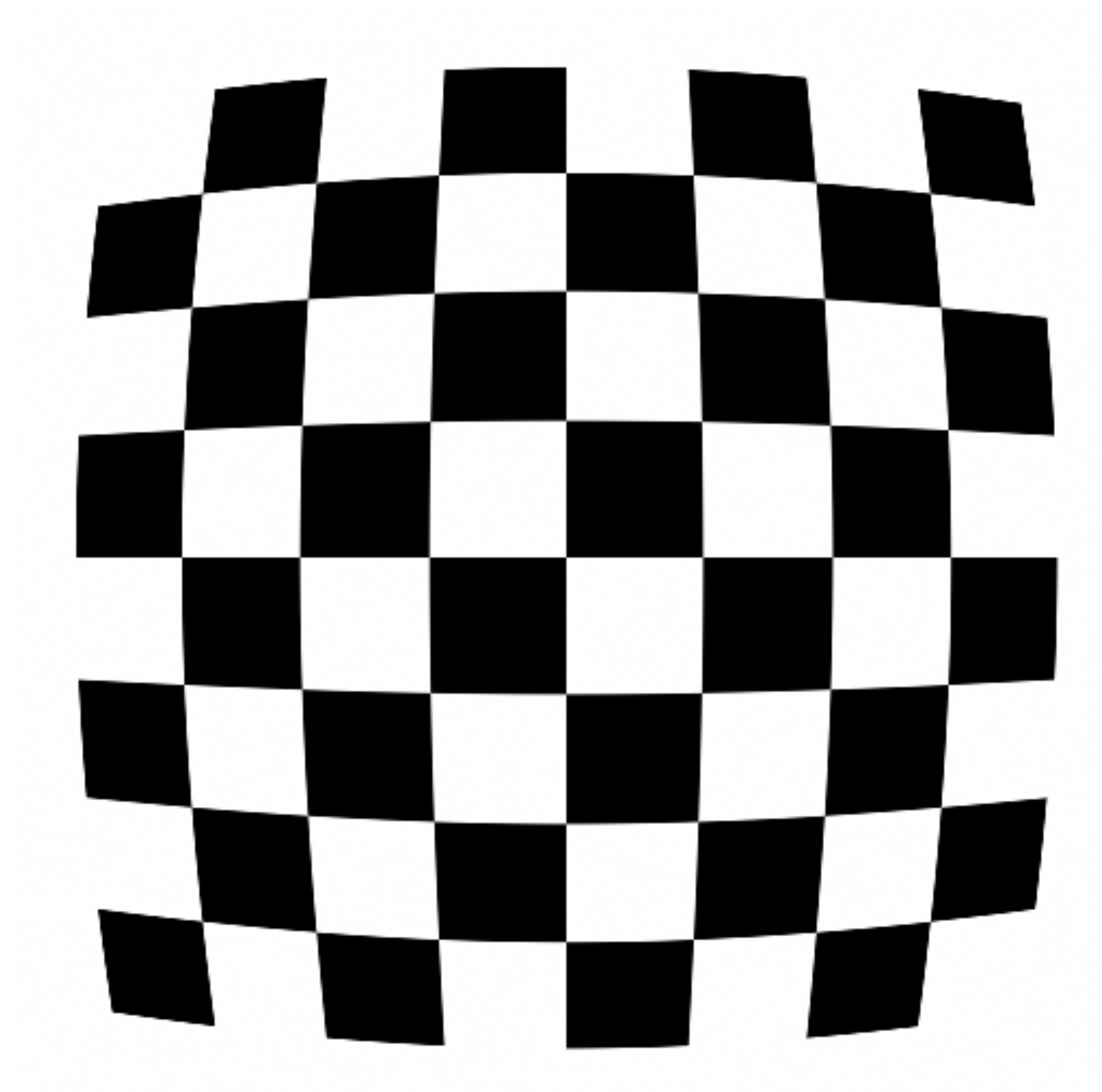
n is set maximum to 3.

$$r_d^2 = \left(\frac{(u - u_0)}{\alpha_u}\right)^2 + \left(\frac{(v - v_0)}{\alpha_v}\right)^2 \quad \alpha_u = -f \cdot k_u \quad \alpha_u = -f \cdot k_v$$

Camera Model: Thin Lens

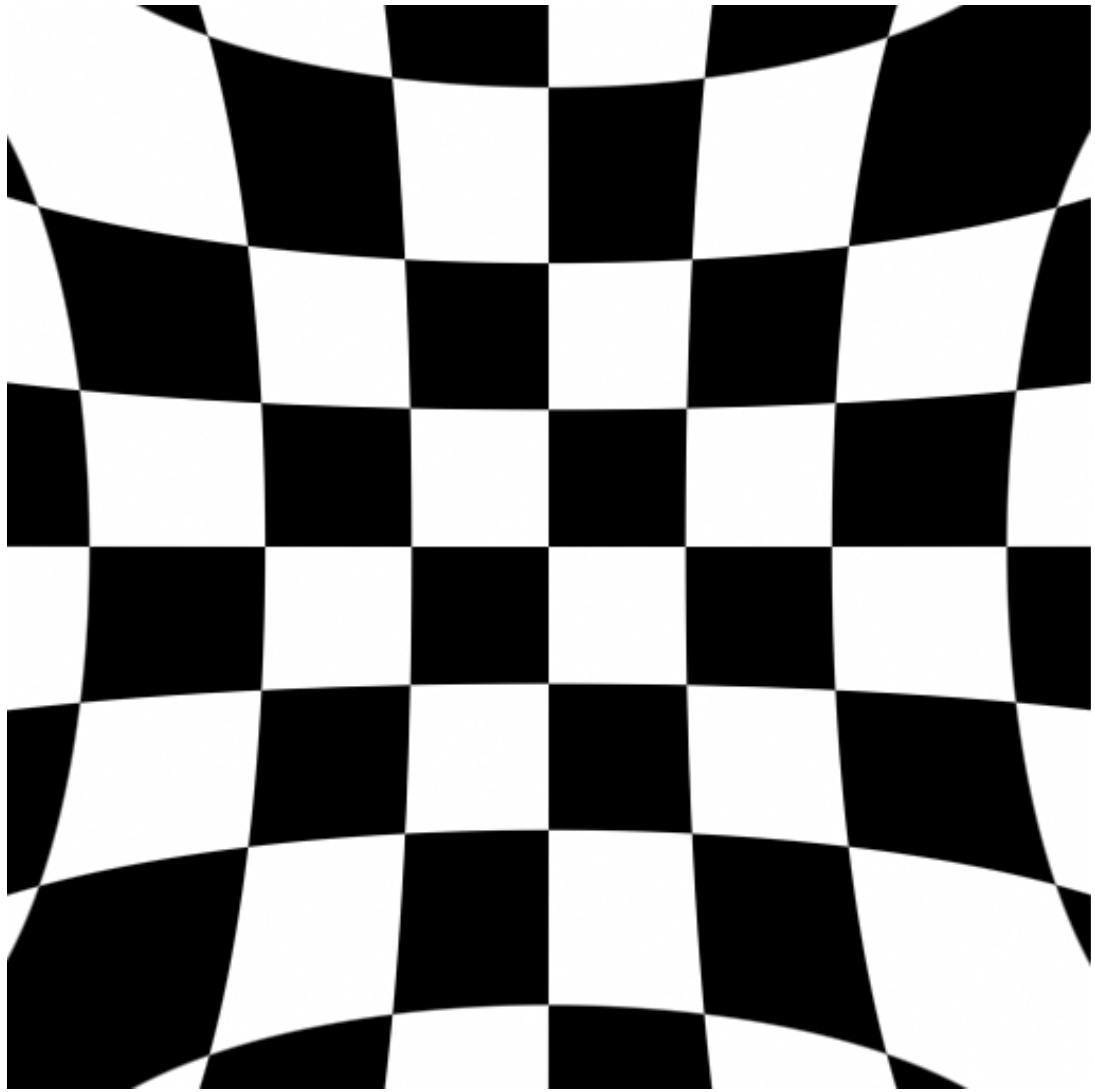


Camera Model: Thin Lens



Barrel distortion

Camera Model: Thin Lens



Pincushion distortion

Camera
Pre-Calibration

Pre-Calibration: Why?

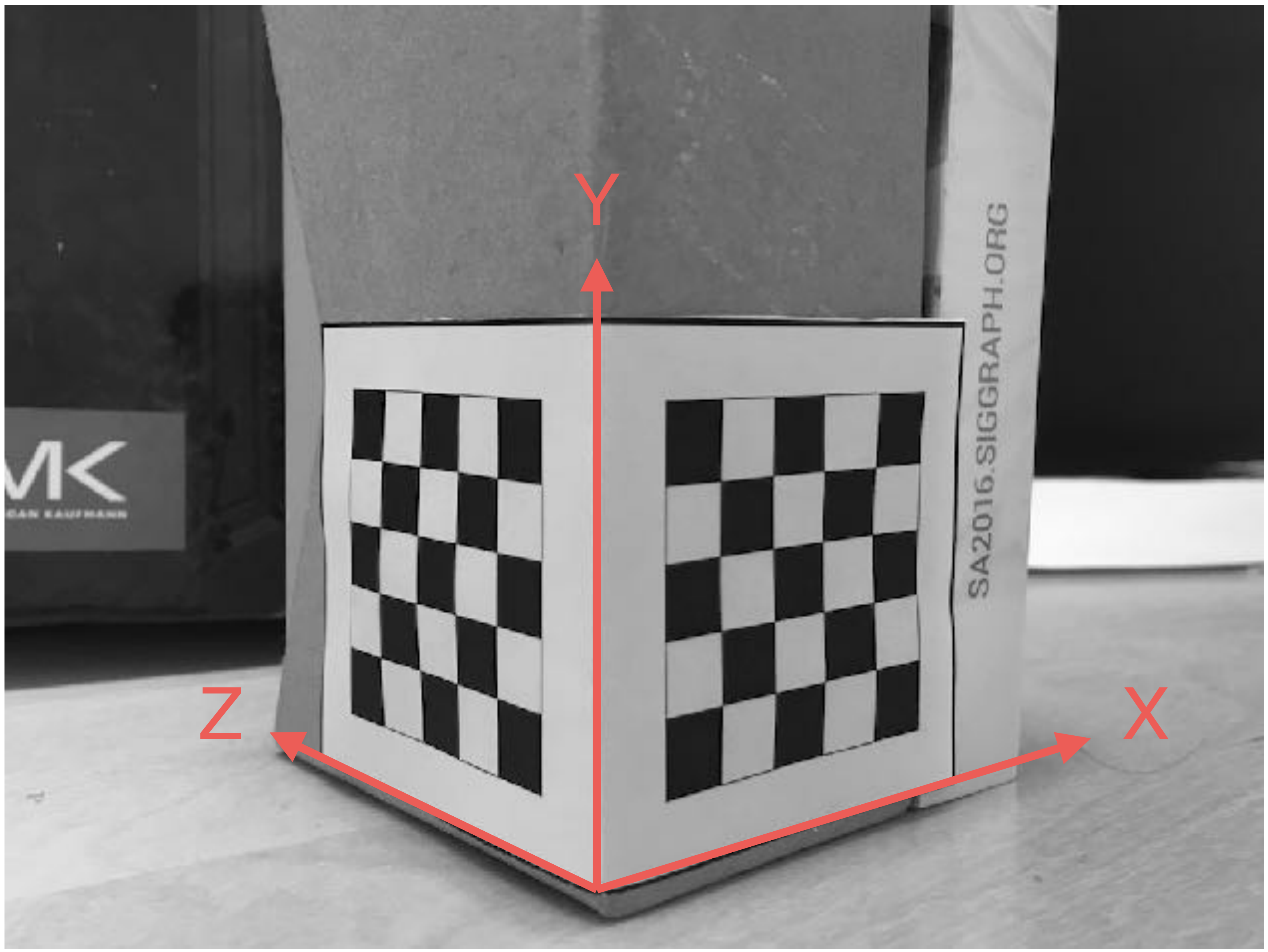
- In some cases, when we know the camera, it is useful to avoid intrinsics matrix estimation:
    - It is more precise.
    - We reduce computations.

DLT:
Direct Linear Transform

DLT: Direct Linear Transform

- **Input**: a photograph of a non-coplanar calibration with m 2D points with known 3D coordinates.
- **Output**: K of the camera. We can optionally recover $[R|\,\boldsymbol{t}]$.

DLT: Direct Linear Transform



DLT: Idea

$$\mathbf{m}_i = [u_i, v_i, 1]^\top \leftrightarrow \mathbf{M}_i = [x, y, z, 1]^\top$$

2D-3D matches

DLT: Idea

- At this point, if we get the projection equation back, we can notice that we know something:

$$P = \begin{bmatrix} \mathbf{p}_1^\top \\ \mathbf{p}_2^\top \\ \mathbf{p}_3^\top \end{bmatrix} \qquad \begin{cases} u = \dfrac{\mathbf{p}_1^\top \cdot \mathbf{M}}{\mathbf{p}_3^\top \cdot \mathbf{M}} \\ v = \dfrac{\mathbf{p}_2^\top \cdot \mathbf{M}}{\mathbf{p}_3^\top \cdot \mathbf{M}} \end{cases}$$

DLT: Idea

$$\begin{cases} \mathbf{p}_1^\top \cdot \mathbf{M}_i - u_i \mathbf{p}_3^\top \cdot \mathbf{M}_i = 0 \\ \mathbf{p}_2^\top \cdot \mathbf{M}_i - v_i \mathbf{p}_3^\top \cdot \mathbf{M}_i = 0 \end{cases}$$

$$\mathbf{m}_i = [u_i, v_i, 1]^\top \leftrightarrow \mathbf{M}_i = [x, y, z, 1]^\top$$

2D-3D matches

DLT: Linear System

- This leads to a matrix:

$$\begin{bmatrix} \mathbf{M}_i^\top & \mathbf{0} & -u_i\mathbf{M}_i^\top \\ \mathbf{0} & -\mathbf{M}_i^\top & v_i\mathbf{M}_i^\top \end{bmatrix} \cdot \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} = \mathbf{0}$$

- For each point, we need to stack this equations obtaining a matrix A.
- We obtain a $2m \times 12$ linear system to solve.
- The minimum number of points to solve it is 6, but more points are required to have robust and stable solutions.

What’s the problem with this method?

DLT: Direct Linear Transform

- DLT minimizes an algebraic error!
- It does not have geometric meaning!!
- Hang on, is it all wrong?
    - Nope, we can use it as input for a non-linear method.

DLT: Non-linear Refinement

- The non-linear refinement minimizes (at least squares) the distance between 2D points of the image ($\mathbf{m}_i$) and projected 3D points ($\mathbf{M}_i$):

$$\arg\min_{P} \sum_{i=1}^{m} \left(\frac{\mathbf{p}_1^\top \cdot \mathbf{M}_i}{\mathbf{p}_3^\top \cdot \mathbf{M}_i} - u_i \right)^2 + \left(\frac{\mathbf{p}_2^\top \cdot \mathbf{M}_i}{\mathbf{p}_3^\top \cdot \mathbf{M}_i} - v_i \right)^2$$

- Different methods for solving it such as Gradient Descent (we need gradients!), Nelder-Mead's method (MATLAB's **fminsearch**), etc.

Now we have a nice matrix P…

DLT: Direct Linear Transform

- Let's recap:
  - K has to be upper-triangular.
  - R is orthogonal.
  - $P = K[R|\mathbf{t}] = [K \cdot R|K \cdot \mathbf{t}] = [P'|\mathbf{p}_4]$

DLT: Direct Linear Transform

- QR decomposition of a matrix A:

$$A = O \cdot T$$

  - where:
    - O is **orthogonal**.
    - T is **upper-triangular**.

- In our case, we have:

$$P' = K \cdot R \rightarrow (P')^{-1} = R^{-1} \cdot K^{-1}$$

DLT: Direct Linear Transform

- QR decomposition to *P'*:

$$[P']_{QR} = O \cdot T$$

- In our case, we have:

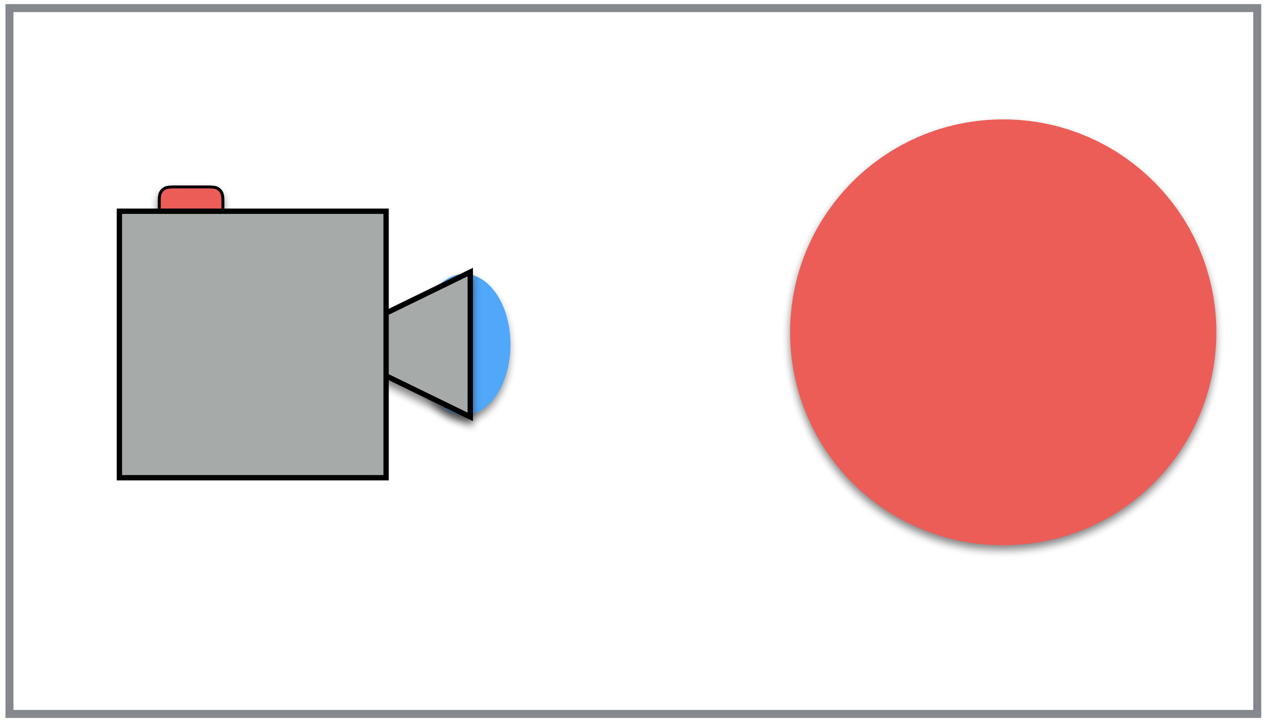
$$R = O^{-1} \quad K = T^{-1}$$

- Note that there is a scale factor!

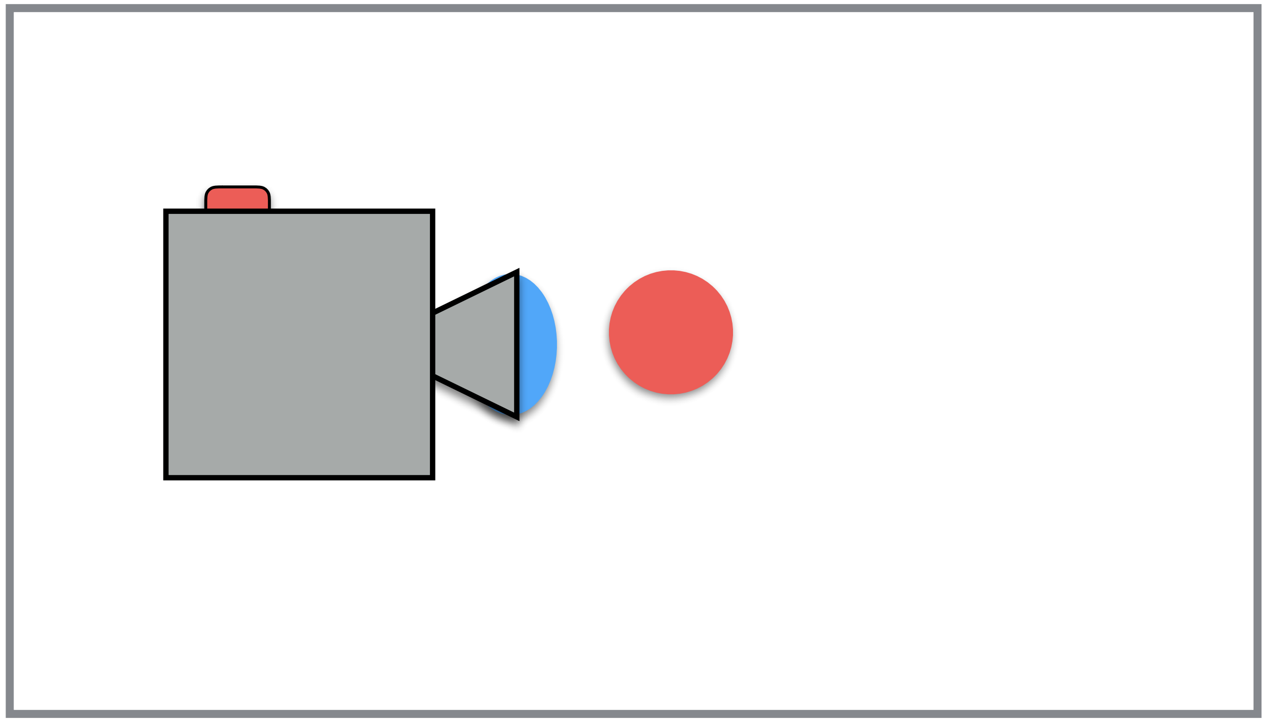
DLT: Direct Linear Transform

- This scale factor is due to the fact we do not know if we took a photograph of:
  - a **large** object **far away** from the camera.
  - a **small** object **near** the camera.

DLT: Direct Linear Transform



Case 1



Case 2

DLT: Direct Linear Transform

- It makes sense to fix the scale in K because R has to be an orthogonal matrix:
  - This affects also $\boldsymbol{t}$.
  - How do we compute t?

DLT: Direct Linear Transform

- It makes sense to fix the scale in K because R has to be an orthogonal matrix:
  - This affects also $\boldsymbol{t}$.
  - How do we compute t?

$$\mathbf{t} = K^{-1} \cdot \mathbf{p}_4$$

The Sanity Check

- If we can have an "***estimation***" of K from camera parameters that are available in the camera specifications:

$$K = \begin{bmatrix} a & 0 & u_0 \\ a & b & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

The Sanity Check

- What do we need?
  - Focal length of the camera in mm (f).
  - Resolution of the picture in pixels (w, h).
  - CCD/CMOS sensor size in mm (w_s, h_s).

The Sanity Check

- $a = (f \times w) / w_s$.
- $b = (f \times h) / h_s$.
- $u_0 = w / 2$.
- $v_0 = h / 2$.

The Sanity Check

- $a = (f \times w) / w_s$.
- $b = (f \times h) / h_s$.
- $u_0 = w / 2$.
- $v_0 = h / 2$.

Assuming it in the center!

and what’s about the radial distortion?

Estimating Radial Distortion

- Let's start with simple radial distortion; i.e., only a coefficient:

$$\begin{cases} u' = (u - u_0) \cdot (1 + k_1 r_d^2) + u_0 \\ v' = (v - v_0) \cdot (1 + k_1 r_d^2) + v_0 \end{cases}$$

$$r_d^2 = \left(\frac{(u - u_0)}{\alpha_u} \right)^2 + \left(\frac{(v - v_0)}{\alpha_v} \right)^2 \quad \alpha_u = -f \cdot k_u \quad \alpha_u = -f \cdot k_v$$

- Can we solve it?

Estimating Radial Distortion

- We have only one unknown, which is linear; i.e., k_1:

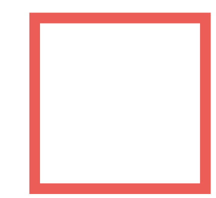
$$\begin{cases} \dfrac{u' - u}{(u - u_0) \cdot r_d^2} = k_1 \\ \dfrac{v' - v}{(v - v_0) \cdot r_d^2} = k_1 \end{cases}$$

- In theory, ***a single point is enough***, but it is better to use more points to get a more robust solution.

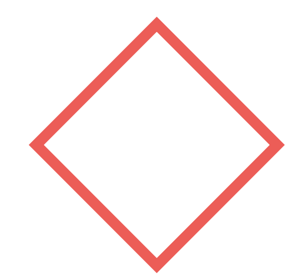
Homography

2D Transformations

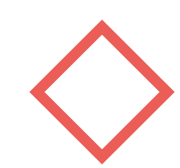
- We can have different type of transformation (defined by a matrix) of 2D points:
  - Translation (2 degree of freedom [DoF]):



    - It preserves orientation.
  - Rigid/Euclidian (3 DoF); translation, and rotation:



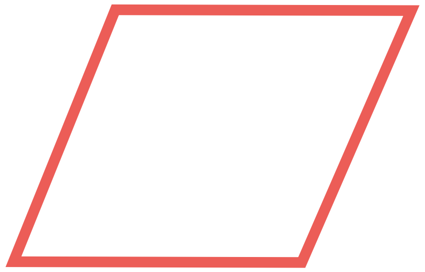
    - It preserves lengths.
  - Similarity (4DoF); translation, rotation, and scaling:

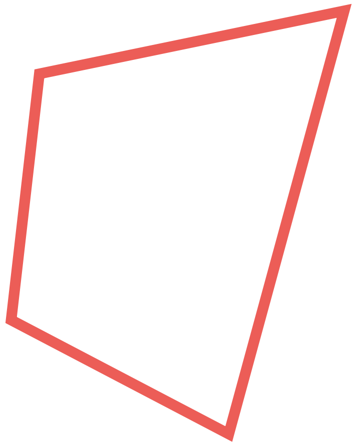


    - It preserves angles.

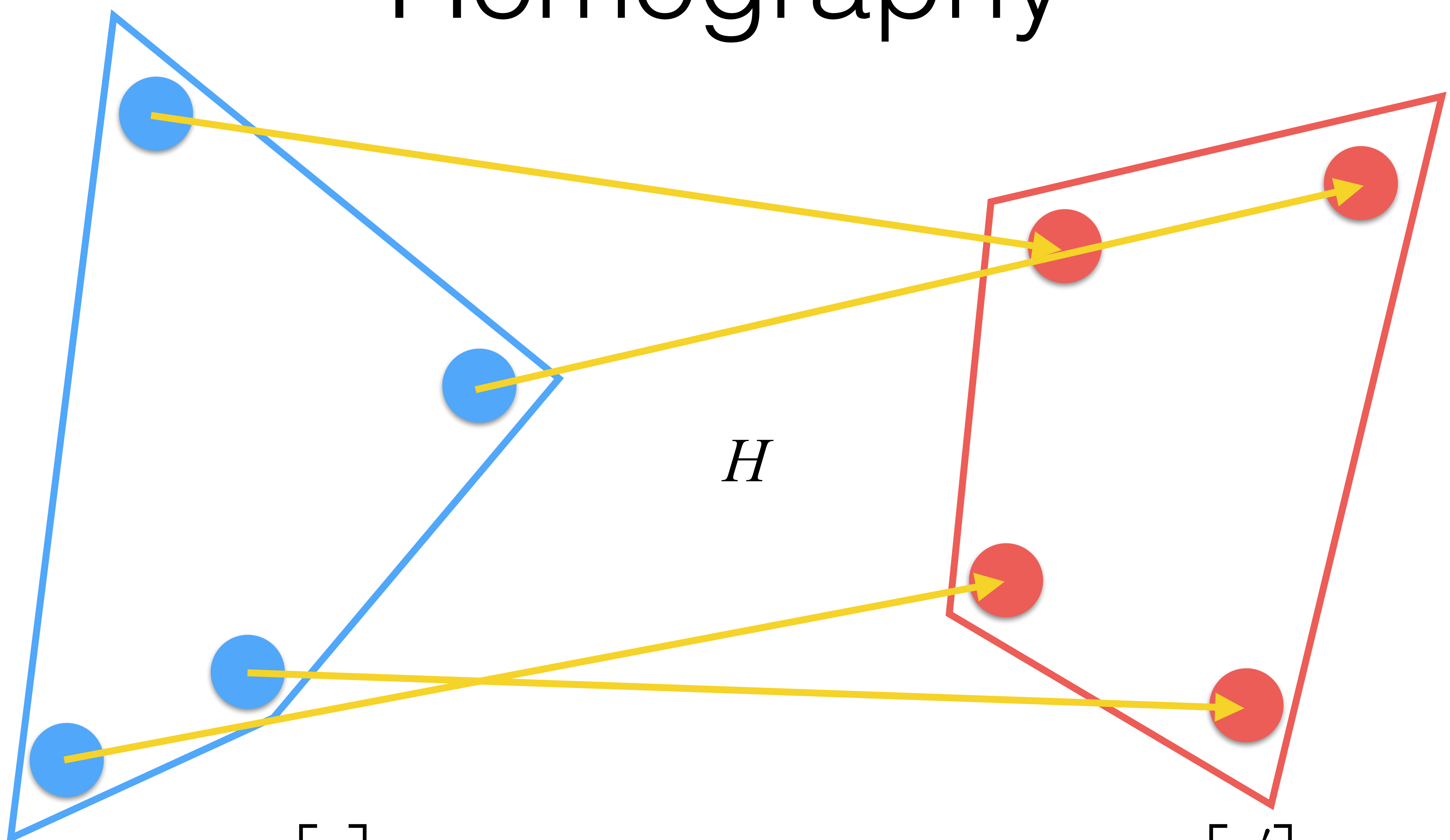
2D Transformations

- Affine (6 degree of freedom [DoF]):
  - It reserves parallelism.
- Projective (8 DoF):
  - It preserves straight lines.





2D Transformations: Homography



$$\mathbf{M} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad \mathbf{m} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

2D Transformations: Homography

- Homography is defined as

$$\mathbf{m}' = H \cdot \mathbf{M} \quad \Rightarrow \quad \mathbf{m} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \mathbf{m}'/m_3$$

- This is typically expressed as

$$\mathbf{m} \sim H \cdot \mathbf{M}$$

- where H is a 3×3 non-singular matrix with 8 DoF.

Homography Estimation

$$\mathbf{m} \sim H \cdot \mathbf{M} \qquad \mathbf{m} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \quad \mathbf{M} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Homography Estimation

$$\mathbf{m} \sim H \cdot \mathbf{M} \qquad \mathbf{m} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \quad \mathbf{M} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Homography Estimation

$$\mathbf{m} \sim H \cdot \mathbf{M} \qquad \mathbf{m} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \quad \mathbf{M} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$x' = \frac{h_{11}x_1 + h_{12}y_1 + h_{13}}{h_{31}x_1 + h_{32}y_1 + h_{33}}$$

$$y' = \frac{h_{21}x_1 + h_{22}y_1 + h_{23}}{h_{31}x_1 + h_{32}y_1 + h_{33}}$$

Homography Estimation

$$(h_{31}x_1 + h_{32}y_1 + h_{33}) \cdot x' - (h_{11}x_1 + h_{12}y_1 + h_{33}) = 0$$
$$(h_{31}x_1 + h_{32}y_1 + h_{33}) \cdot y' - (h_{21}x_1 + h_{22}y_1 + h_{23}) = 0$$

Stacking multiple equations;
one for each match (at least 5!)

Homography Estimation

$$(h_{31}x_1 + h_{32}y_1 + h_{33}) \cdot x' - (h_{11}x_1 + h_{12}y_1 + h_{33}) = 0$$
$$(h_{31}x_1 + h_{32}y_1 + h_{33}) \cdot y' - (h_{21}x_1 + h_{22}y_1 + h_{23}) = 0$$

Stacking multiple equations;
one for each match (at least 5!)

$$A \cdot \mathrm{vec}(H) = \mathbf{0}$$ A is $2n \times 9$

Homography Estimation

- Again, we have minimized an algebraic error!!
- Technically speaking, we should run a non-linear optimization:

$$\arg\min_{H} \sum_{I=1}^{m} \left(x_i' - \frac{\mathbf{h}_1^\top \cdot \mathbf{M}_i}{\mathbf{h}_3^\top \cdot \mathbf{M}_i} \right)^2 + \left(y_i' - \frac{\mathbf{h}_2^\top \cdot \mathbf{M}_i}{\mathbf{h}_3^\top \cdot \mathbf{M}_i} \right)^2$$

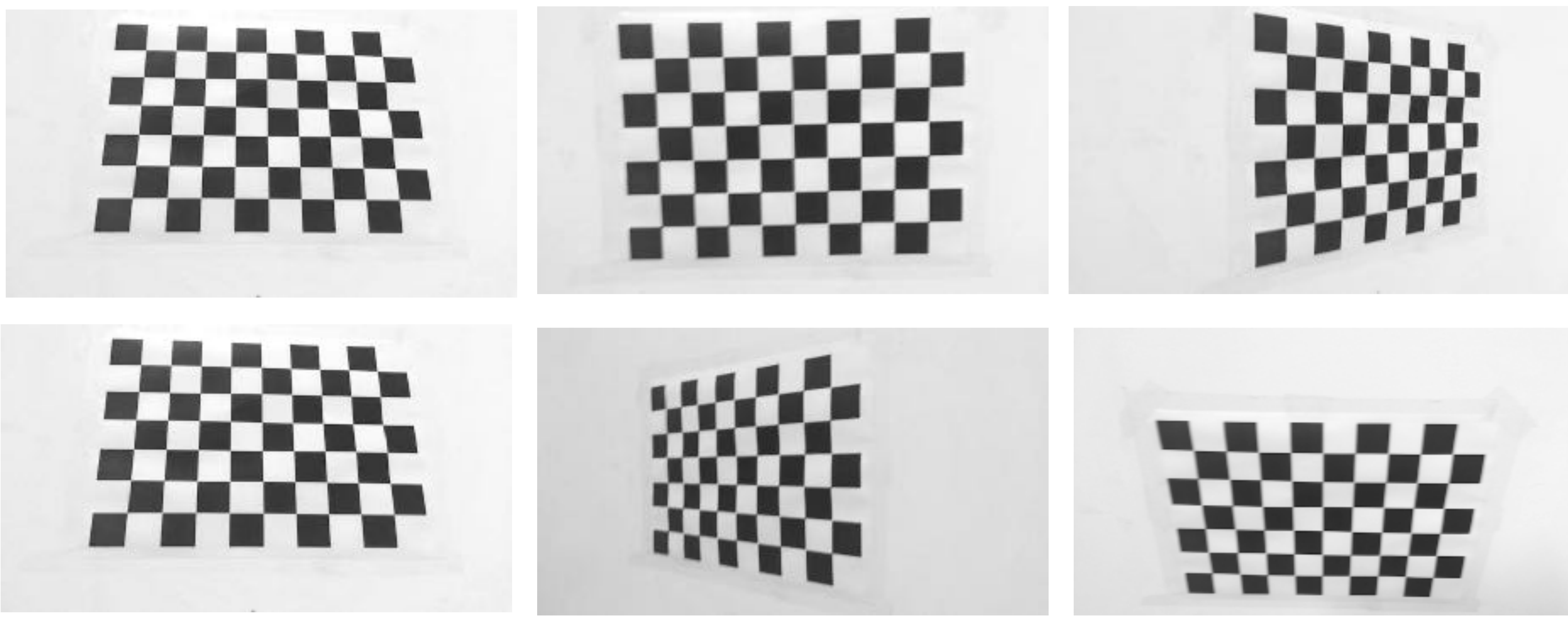
- where $H = [\mathbf{h}_1 \ \mathbf{h}_2 \ \mathbf{h}_3]$.

Zhang's Algorithm

Zhang's Algorithm

- **Input**: a set of n photographs of a checkboard or other patterns. From these, we have to extract m points in each photograph!
- **Output**: K of the camera. We can optionally compute $[R|\mathbf{t}]$ for each photographs.

Zhang's Algorithm



A set of input images

Zhang's Algorithm

$$K = \begin{bmatrix} -fk_u & 0 & u_0 \\ 0 & -fk_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

Zhang's Algorithm

$$K = \begin{bmatrix} \alpha & c & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

Zhang's Algorithm

$$K = \begin{bmatrix} \alpha & \boxed{c} & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

Zhang's Algorithm

- **Assumption:**
  - We have a set of photographs of a plane so Z is equal 0.
  - So we have 3D points defined as

$$\mathbf{M} = \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix}$$

Zhang's Algorithm

- This means that we have:

$$\mathbf{m} = P \cdot \mathbf{M} =$$

$$= K \cdot [R|\mathbf{t}] \cdot \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} =$$

Zhang's Algorithm

$$\mathbf{m} = K \cdot [\mathbf{r}_1 \mathbf{r}_2 \mathbf{r}_3 | \mathbf{t}] \cdot \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} =$$

$$= K \cdot [\mathbf{r}_1 \mathbf{r}_2 | \mathbf{t}] \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Zhang's Algorithm

$$\mathbf{m} = K \cdot [\mathbf{r}_1 \mathbf{r}_2 \mathbf{r}_3 | \mathbf{t}] \cdot \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} =$$

$$= K \cdot [\mathbf{r}_1 \mathbf{r}_2 | \mathbf{t}] \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Zhang's Algorithm

$$\mathbf{m} = K \cdot [\mathbf{r}_1 \mathbf{r}_2 \mathbf{r}_3 | \mathbf{t}] \cdot \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} =$$

$$= K \cdot [\mathbf{r}_1 \mathbf{r}_2 | \mathbf{t}] \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$H = K \cdot [\mathbf{r}_1 \mathbf{r}_2 | \mathbf{t}]$$

Zhang's Algorithm

$$\mathbf{m} = K \cdot [\mathbf{r}_1 \mathbf{r}_2 \mathbf{r}_3 | \mathbf{t}] \cdot \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} =$$

$$= K \cdot [\mathbf{r}_1 \mathbf{r}_2 | \mathbf{t}] \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

It is a homography!

$$H = K \cdot [\mathbf{r}_1 \mathbf{r}_2 | \mathbf{t}]$$

Zhang's Algorithm

$$\mathbf{m} = K \cdot [\mathbf{r}_1 \mathbf{r}_2 \mathbf{r}_3 | \mathbf{t}] \cdot \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} =$$

$$= K \cdot [\mathbf{r}_1 \mathbf{r}_2 | \mathbf{t}] \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

It is a homography!

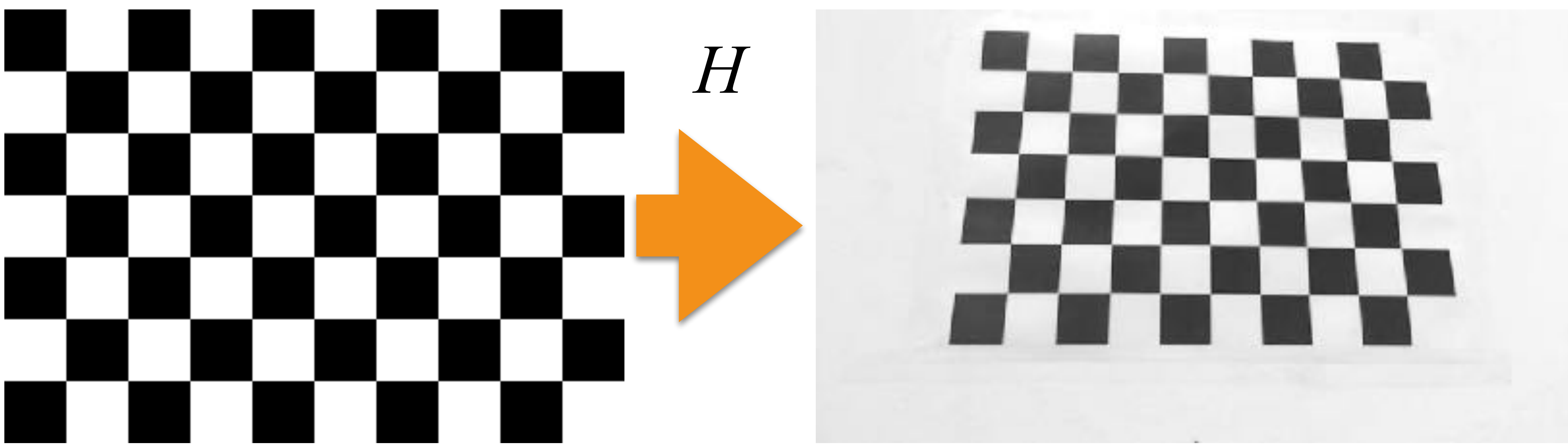
$$H = K \cdot [\mathbf{r}_1 \mathbf{r}_2 | \mathbf{t}]$$

$$H = \begin{bmatrix} \mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3 \end{bmatrix}$$

Zhang's Algorithm

- What to do?
- For each photograph:
  - We compute the homography H between photographed checkerboard corners and its model.

Zhang's Algorithm



Model

Photograph

Zhang's Algorithm

- Given that $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are orthonormal, we have that:

$$\mathbf{h}_1^\top K^{-\top} K^{-1} \mathbf{h}_2 = 0$$

$$\mathbf{h}_1^\top K^{-\top} K^{-1} \mathbf{h}_1 = \mathbf{h}_2^\top K^{-\top} K^{-1} \mathbf{h}_2$$

Zhang's Algorithm

$$B = K^{-\top}K^{-1} = \begin{bmatrix} \frac{1}{\alpha^2} & -\frac{c}{\alpha^2\beta} & \frac{cv_0 - u_0\beta}{\alpha^2\beta} \\ -\frac{c}{\alpha^2\beta} & \frac{c^2}{\alpha^2\beta^2} + \frac{1}{\beta^2} & -\frac{c(cv_0 - u_0\beta)}{\alpha^2\beta^2} - \frac{v_0}{\beta^2} \\ \frac{cv_0 - u_0\beta}{\alpha^2\beta} & -\frac{c(cv_0 - u_0\beta)}{\alpha^2\beta^2} - \frac{v_0}{\beta^2} & \frac{(cv_0 - u_0\beta)^2}{\alpha^2\beta^2} + \frac{v_0^2}{\beta^2} + 1 \end{bmatrix}$$

B is symmetric —> defined only by six values:

$$\mathbf{b} = [B_{1,1}, B_{1,2}, B_{2,2}, B_{1,3}, B_{2,3}, B_{3,3}]^\top$$

Zhang's Algorithm

$$B = K^{-\top}K^{-1} = \begin{bmatrix} \frac{1}{\alpha^2} & -\frac{c}{\alpha^2\beta} & \frac{cv_0 - u_0\beta}{\alpha^2\beta} \\ -\frac{c}{\alpha^2\beta} & \frac{c^2}{\alpha^2\beta^2} + \frac{1}{\beta^2} & -\frac{c(cv_0 - u_0\beta)}{\alpha^2\beta^2} - \frac{v_0}{\beta^2} \\ \frac{cv_0 - u_0\beta}{\alpha^2\beta} & -\frac{c(cv_0 - u_0\beta)}{\alpha^2\beta^2} - \frac{v_0}{\beta^2} & \frac{(cv_0 - u_0\beta)^2}{\alpha^2\beta^2} + \frac{v_0^2}{\beta^2} + 1 \end{bmatrix}$$

B is symmetric —> defined only by six values:

$$\mathbf{b} = [B_{1,1}, B_{1,2}, B_{2,2}, B_{1,3}, B_{2,3}, B_{3,3}]^\top$$

Zhang's Algorithm

$$\mathbf{h}_i^\top \cdot B \cdot \mathbf{h}_j = \mathbf{v}_{i,j}^\top \cdot \mathbf{b}$$

Zhang's Algorithm

$$\mathbf{h}_i^\top \cdot B \cdot \mathbf{h}_j = \mathbf{v}_{i,j}^\top \cdot \mathbf{b}$$

$$H = \begin{bmatrix} \mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3 \end{bmatrix}$$

$$\mathbf{h}_i = \begin{bmatrix} h_{i1} & h_{i2} & h_{i3} \end{bmatrix}^\top$$

$$\mathbf{v}_{ij} = \begin{bmatrix} h_{i1}h_{j1} \\ h_{i1}h_{j2} + h_{i2}h_{j1} \\ h_{i2}h_{j2} \\ h_{i3}h_{j1} + h_{i1}h_{j3} \\ h_{i3}h_{j2} + h_{i2}h_{j3} \\ h_{i3}h_{j3} \end{bmatrix}$$

Zhang's Algorithm

- Given that $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are orthonormal, we have that:

$$\mathbf{h}_1^\top K^{-\top} K^{-1} \mathbf{h}_2 = 0$$

$$\mathbf{h}_1^\top K^{-\top} K^{-1} \mathbf{h}_1 = \mathbf{h}_2^\top K^{-\top} K^{-1} \mathbf{h}_2$$

$$\begin{bmatrix} \mathbf{v}_{12}^T \\ (\mathbf{v}_{11} - \mathbf{v}_{22})^T \end{bmatrix} \mathbf{b} = \mathbf{0}$$

Zhang's Algorithm

- If n images of the model plane are observed, by stacking n of such equations:

$$\begin{bmatrix} \mathbf{v}_{12}^T \\ (\mathbf{v}_{11} - \mathbf{v}_{22})^T \end{bmatrix} \mathbf{b} = \mathbf{0}$$

- We obtain:

$$V \cdot \mathbf{b} = \mathbf{0}$$

V is $2n \times 6$ matrix, so we need $n > 2$

Zhang's Algorithm

- At this point, we can compute elements of K as

$$
\begin{aligned}
v_0 &= (B_{12}B_{13} - B_{11}B_{23})/(B_{11}B_{22} - B_{12}^2) \\
\lambda &= B_{33} - [B_{13}^2 + v_0(B_{12}B_{13} - B_{11}B_{23})]/B_{11} \\
\alpha &= \sqrt{\lambda/B_{11}} \\
\beta &= \sqrt{\lambda B_{11}/(B_{11}B_{22} - B_{12}^2)} \\
c &= -B_{12}\alpha^2\beta/\lambda \\
u_0 &= cv_0/\alpha - B_{13}\alpha^2/\lambda \; .
\end{aligned}
$$

Zhang's Algorithm: Camera Pose

- Furthermore, we can extract the pose as

$$\mathbf{r}_1 = \lambda \cdot K^{-1}\mathbf{h}_1$$

$$\mathbf{r}_2 = \lambda \cdot K^{-1}\mathbf{h}_2$$

$$\mathbf{r}_3 = \mathbf{r}_1 \times \mathbf{r}_2$$

$$\mathbf{t} = \lambda K^{-1}\mathbf{h}_3$$

Zhang's Algorithm: Non-Linear Refinement

- So far, we have obtained a solution through minimizing an algebraic distance that is not physically meaningful!

- From that solution, we can use a non-linear method for minimizing the following error:

$$\sum_{i=1}^{n}\sum_{j=1}^{m} \|\mathbf{m}_{i,j} - \tilde{\mathbf{m}}(K, R_i, \mathbf{t}_i, \mathbf{M}_j)\|^2$$

Zhang's Algorithm: Non-Linear Refinement

- So far, we have obtained a solution through minimizing an algebraic distance that is not physically meaningful!
- From that solution, we can use a non-linear method for minimizing the following error:

$$\sum_{i=1}^{n} \sum_{j=1}^{m} \|\mathbf{m}_{i,j} - \tilde{\mathbf{m}}(K, R_i, \mathbf{t}_i, \mathbf{M}_j)\|^2$$

This is a function projecting $\mathbf{M}_j$ points given intrinsics and the pose!

Zhang's Algorithm: Optical Distortion

- What's about the parameters for modeling the radial distortion?
- As before, first algebraic solution, and then a non-linear solution.

Zhang's Algorithm: Optical Distortion

$$\begin{bmatrix} (u-u_0)r_d^2 & (u-u_0)r_d^4 \\ (v-v_0)r_d^2 & (v-v_0)r_d^4 \end{bmatrix} \cdot \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} = \begin{bmatrix} u'-u \\ v'-v \end{bmatrix}$$

$$D \cdot \mathbf{k} = \mathbf{d}$$

$$\mathbf{k} = (D^\top \cdot D)^{-1} \cdot D^\top \cdot \mathbf{d}$$

Zhang's Algorithm: Non-Linear Refinement

- We extend the previous non-linear model to include optical distortion:

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\|\mathbf{m}_{i,j} - \tilde{\mathbf{m}}(K, R_i, \mathbf{t}_i, \mathbf{k}, \mathbf{M}_j)\|^2$$

Zhang's Algorithm: Non-Linear Refinement

- We extend the previous non-linear model to include optical distortion:

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\|\mathbf{m}_{i,j} - \boxed{\tilde{\mathbf{m}}(K, R_i, \mathbf{t}_i, \mathbf{k}, \mathbf{M}_j)}\|^2$$

This is a function projecting M_j points given intrinsics and the pose!

that’s all folks!