



Industrial Computer Vision


- Geometric Camera Models

11th lecture, 2021.11.17
Lecturer: Youngbae Hwang

Contents

- Pinhole camera model vs. Lens camera model
- Camera Projection Matrix
- Geometric Camera Calibration
- Radial Distortion

Let's say we have a sensor...



digital sensor
(CCD or CMOS)

... and an object we like to photograph

real-world
object



digital sensor
(CCD or CMOS)



What would an image taken like this look like?

Bare-sensor imaging

real-world
object

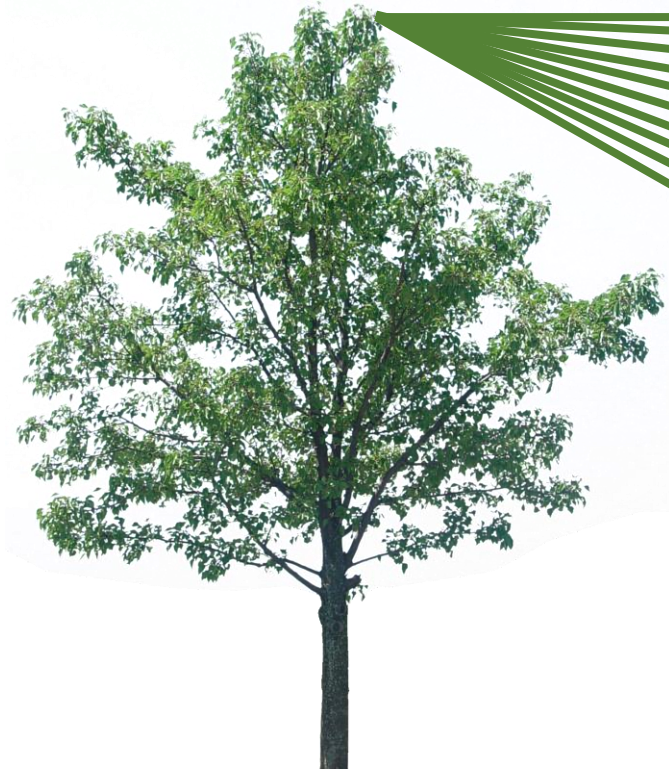


digital sensor
(CCD or CMOS)



Bare-sensor imaging

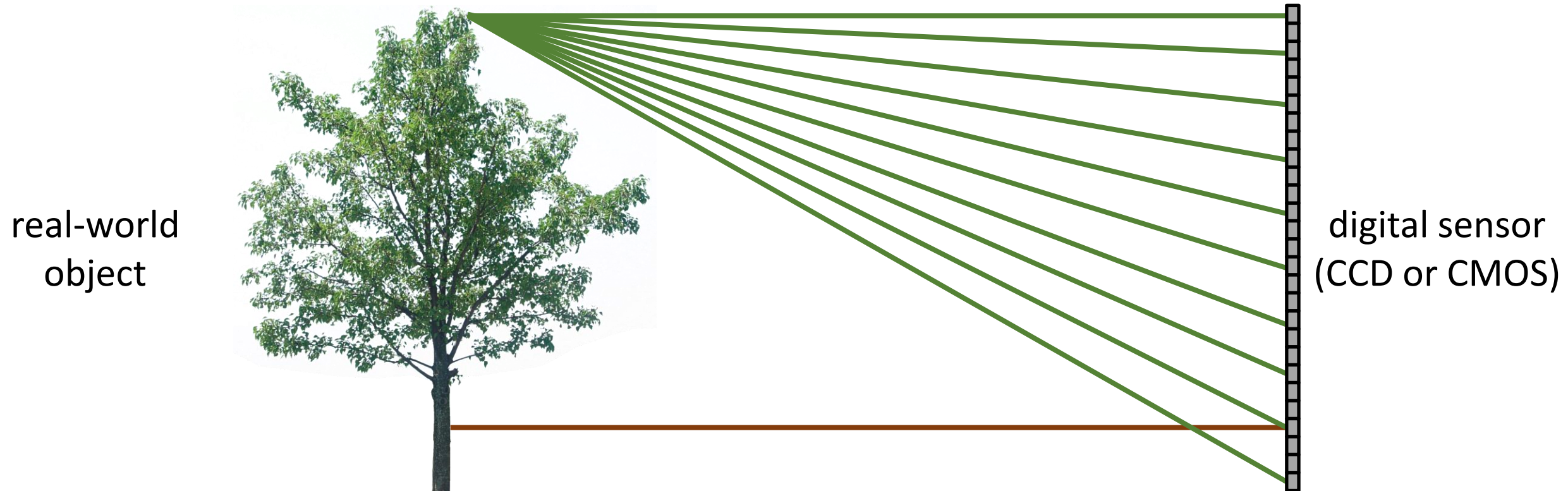
real-world
object



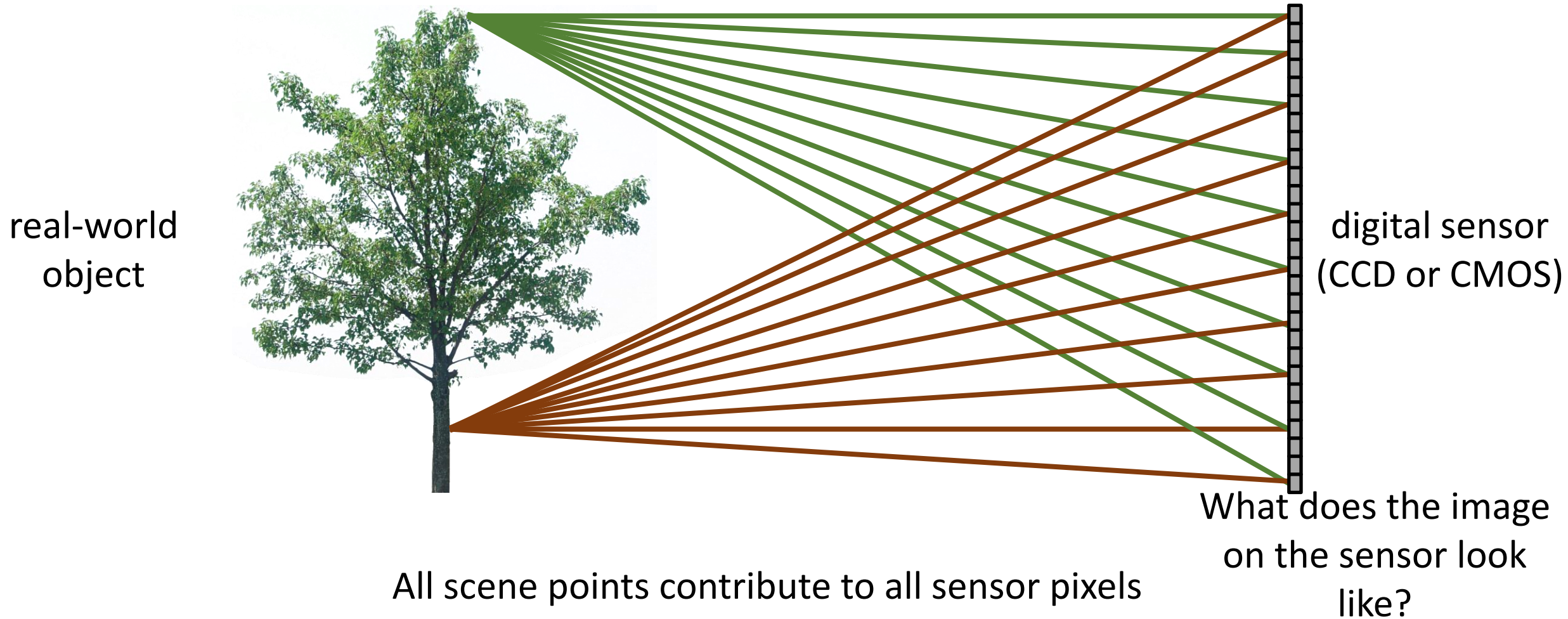
digital sensor
(CCD or CMOS)



Bare-sensor imaging



Bare-sensor imaging



Bare-sensor imaging



All scene points contribute to all sensor pixels

Let's add something to this scene

real-world
object



barrier (diaphragm)



pinhole
(aperture)



digital sensor
(CCD or CMOS)

What would an image taken like this look like?

Pinhole imaging

real-world
object



most rays
are blocked

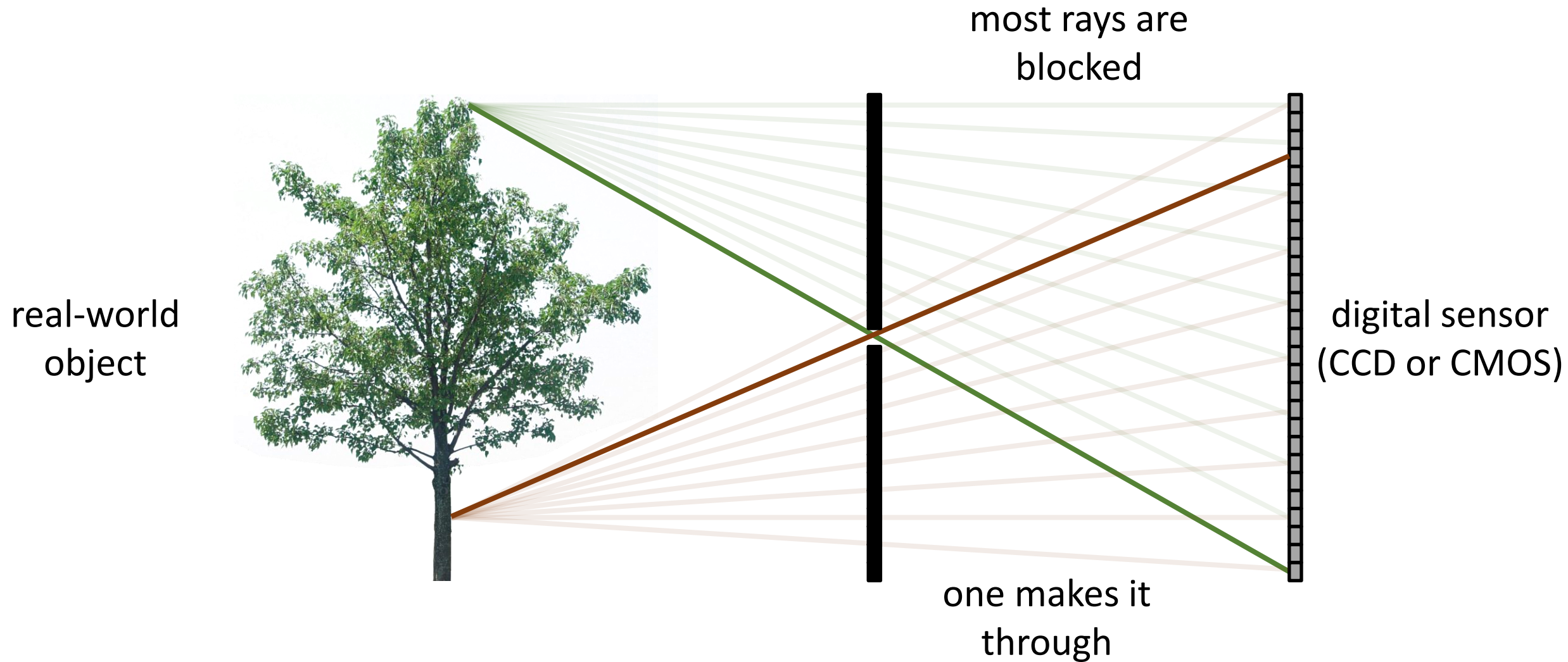


one makes it
through

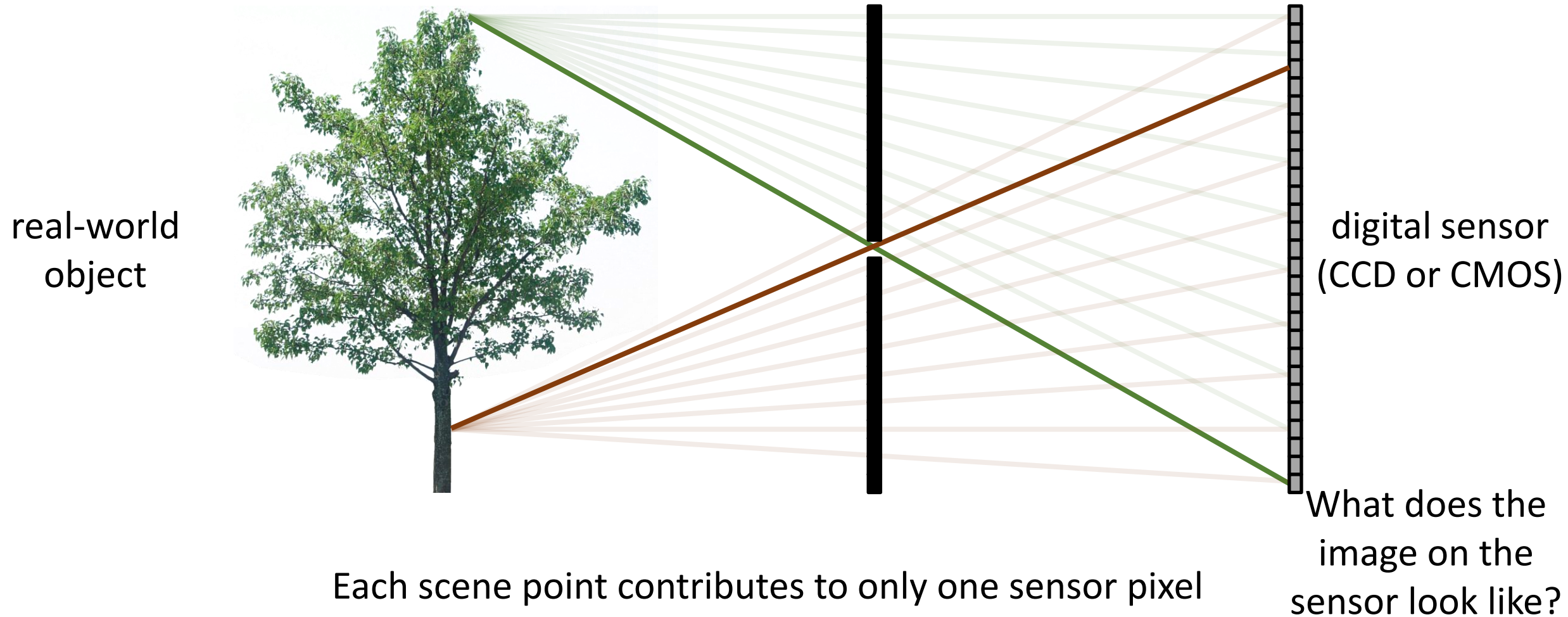
digital sensor
(CCD or CMOS)



Pinhole imaging

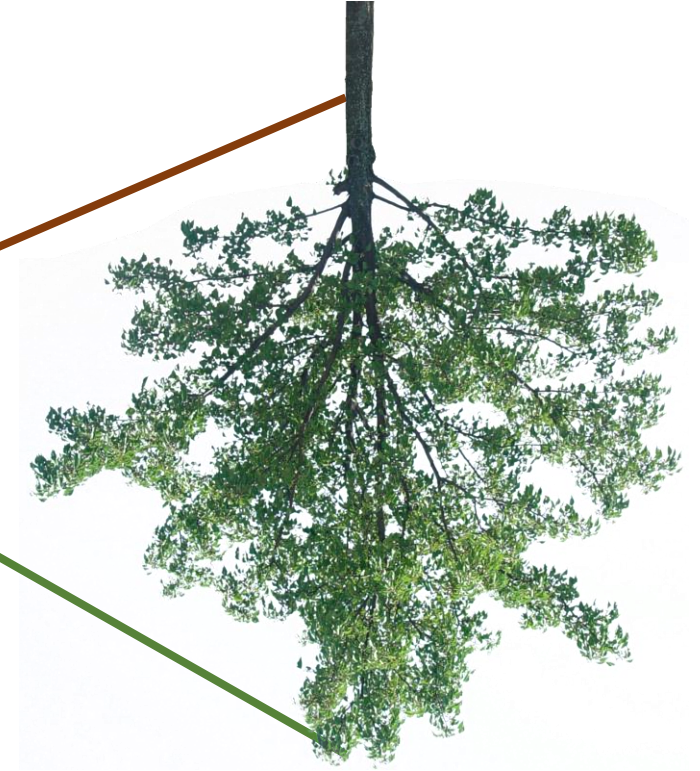
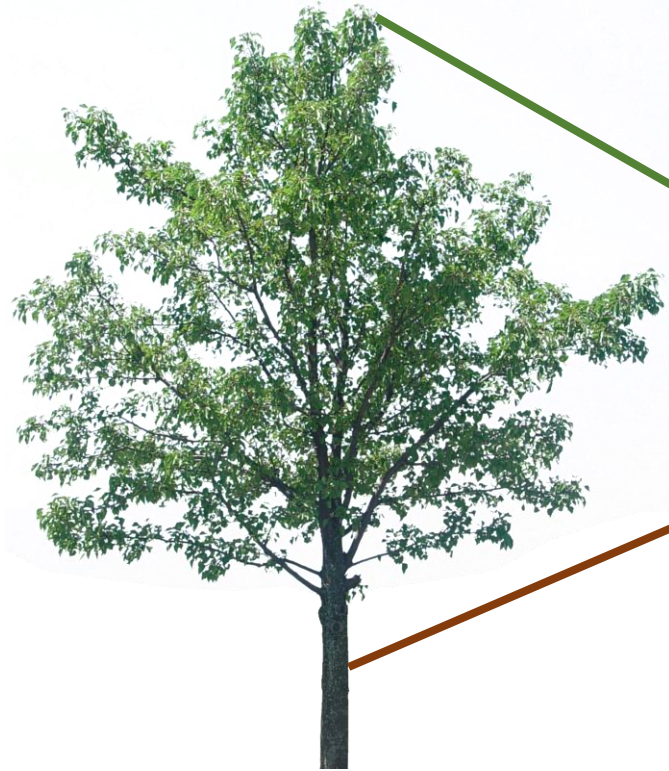


Pinhole imaging



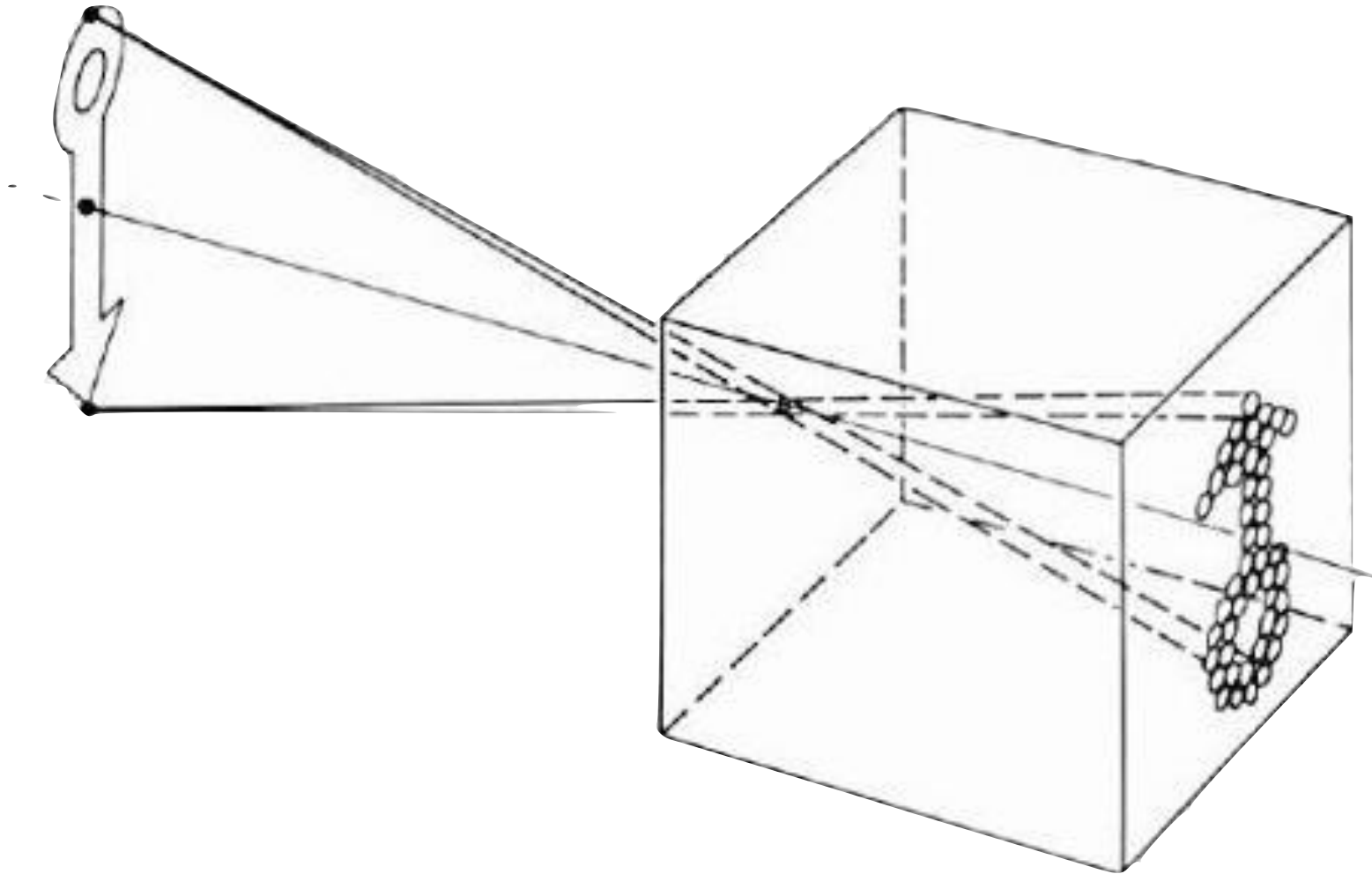
Pinhole imaging

real-world
object



copy of real-world object
(inverted and scaled)

Pinhole camera a.k.a. camera obscura



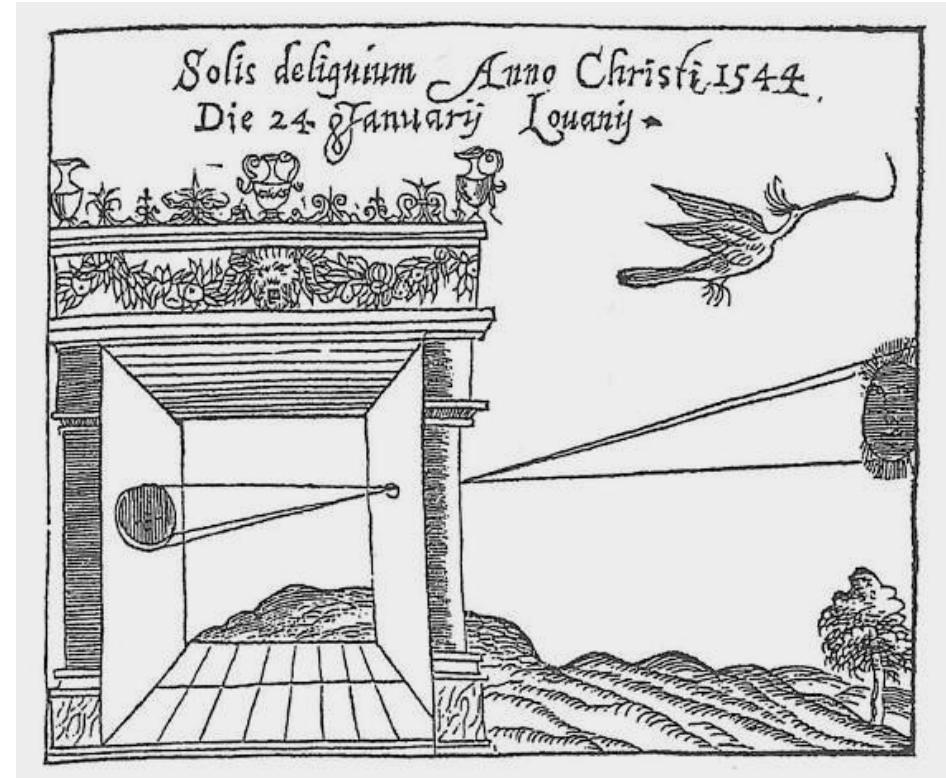
Pinhole camera a.k.a. camera obscura

First mention ...



Chinese philosopher Mozi
(470 to 390 BC)

First camera ...



Greek philosopher Aristotle
(384 to 322 BC)

Pinhole camera terms

real-world
object



barrier (diaphragm)



pinhole
(aperture)



digital sensor
(CCD or CMOS)

Pinhole camera terms

real-world
object



barrier (diaphragm)



pinhole
(aperture)



camera center
(center of projection)

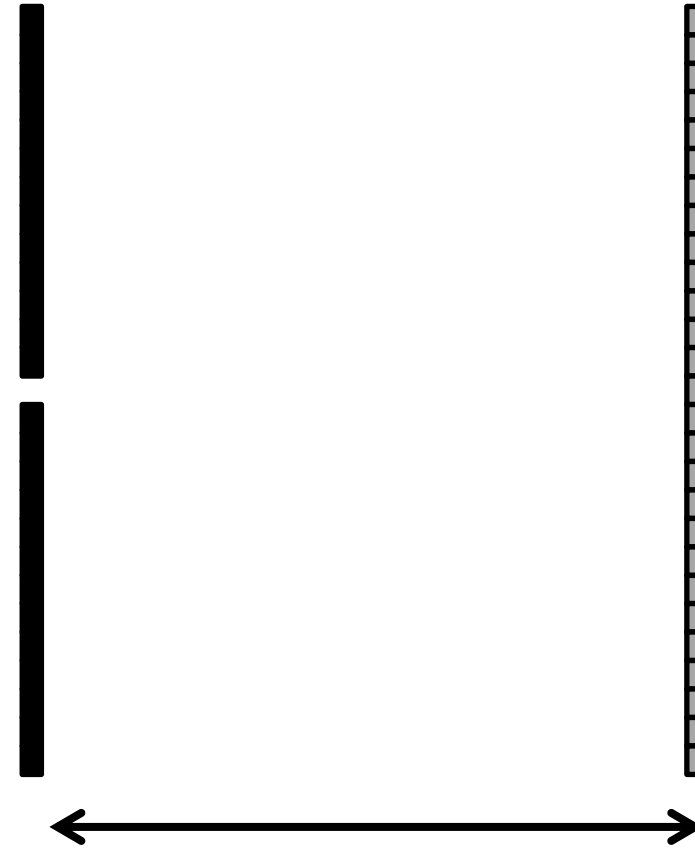
image plane



digital sensor
(CCD or CMOS)

Focal length

real-world
object

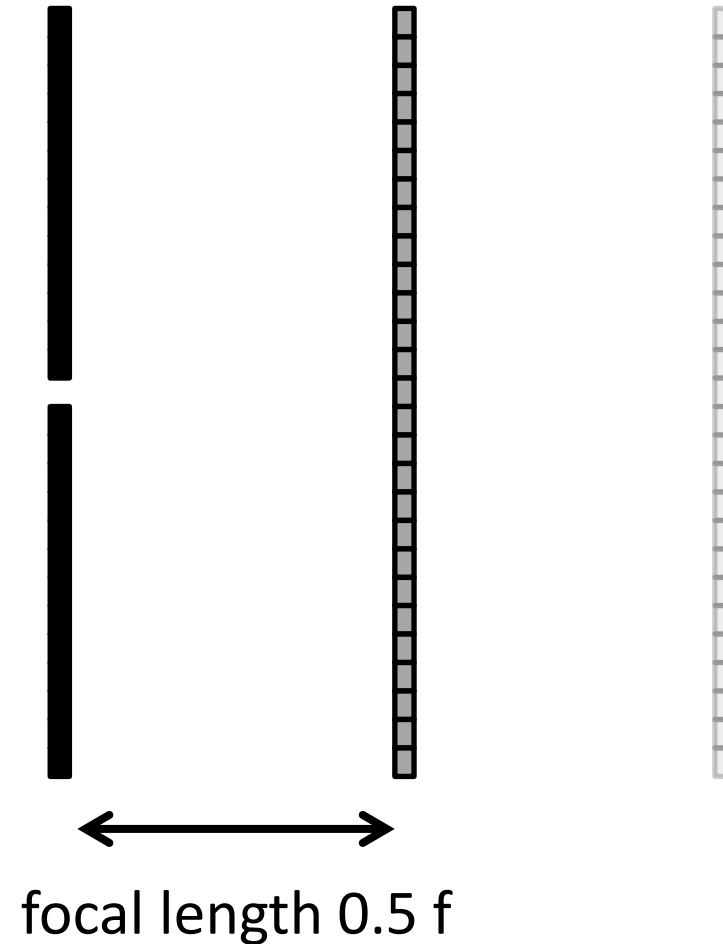


focal length f

Focal length

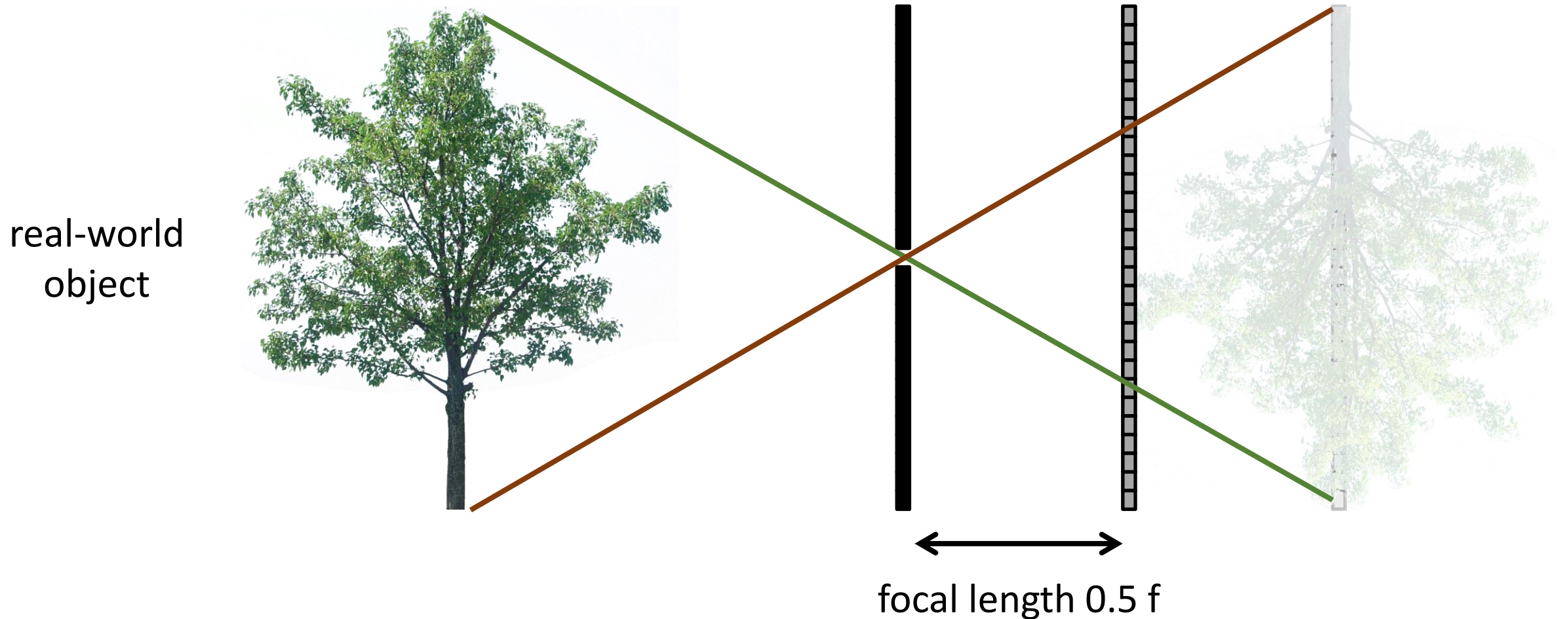
- What happens as we change the focal length?

real-world
object



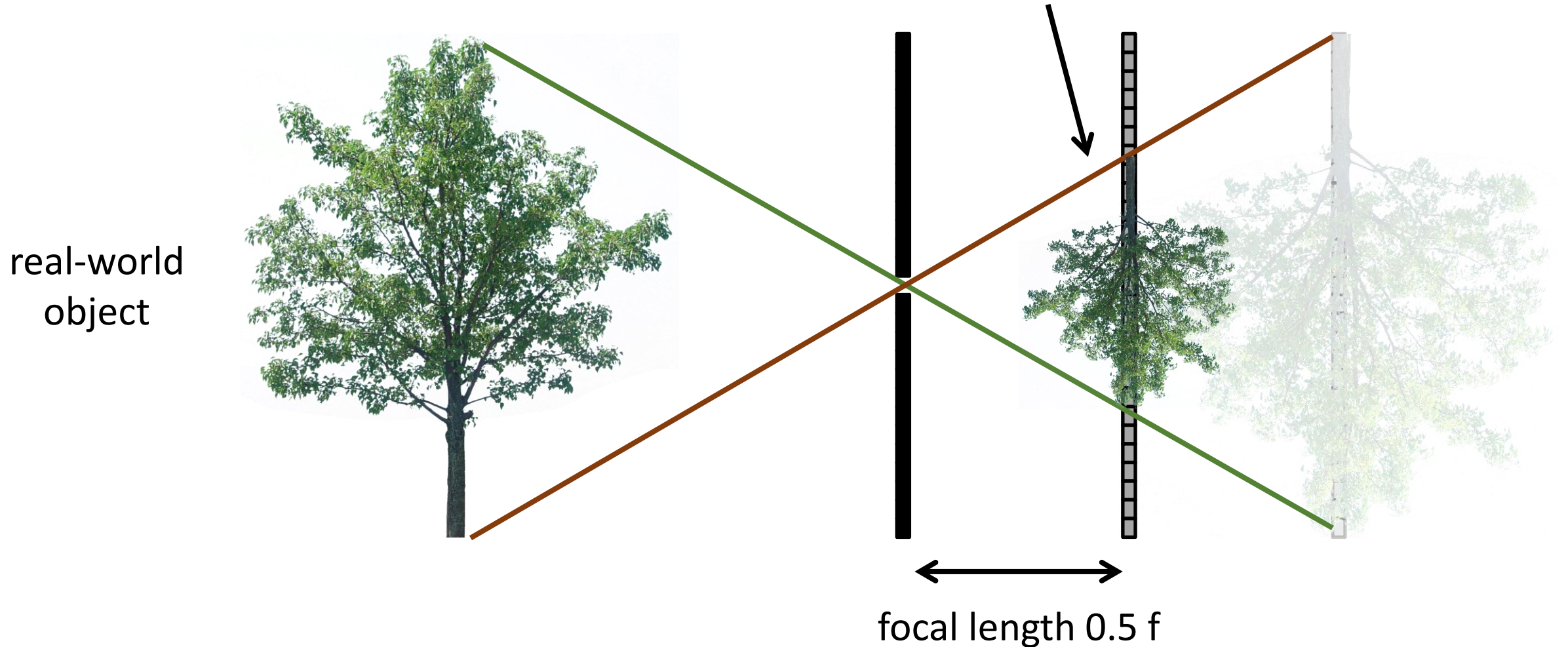
Focal length

- What happens as we change the focal length?



Focal length

- What happens as we change the focal length? object projection is half the size



Pinhole size

real-world
object



pinhole
diameter



Ideal pinhole has infinitesimally small size

- In practice that is impossible.

Pinhole size

- What happens as we change the pinhole diameter?

real-world
object

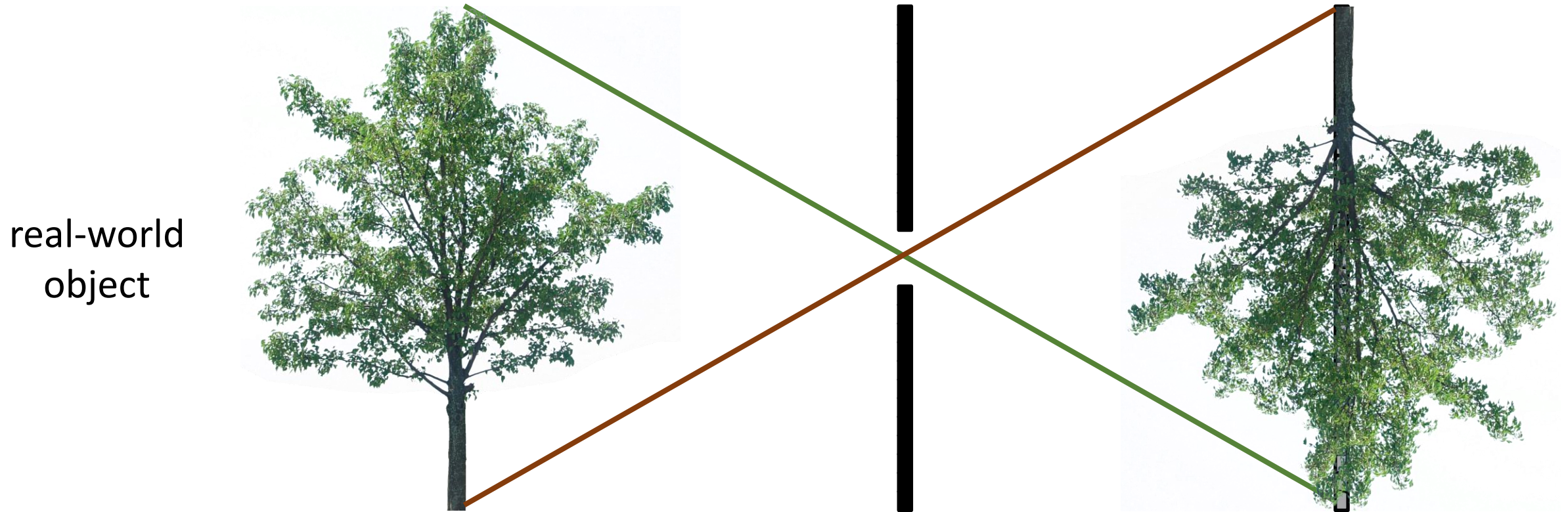


pinhole
diameter



Pinhole size

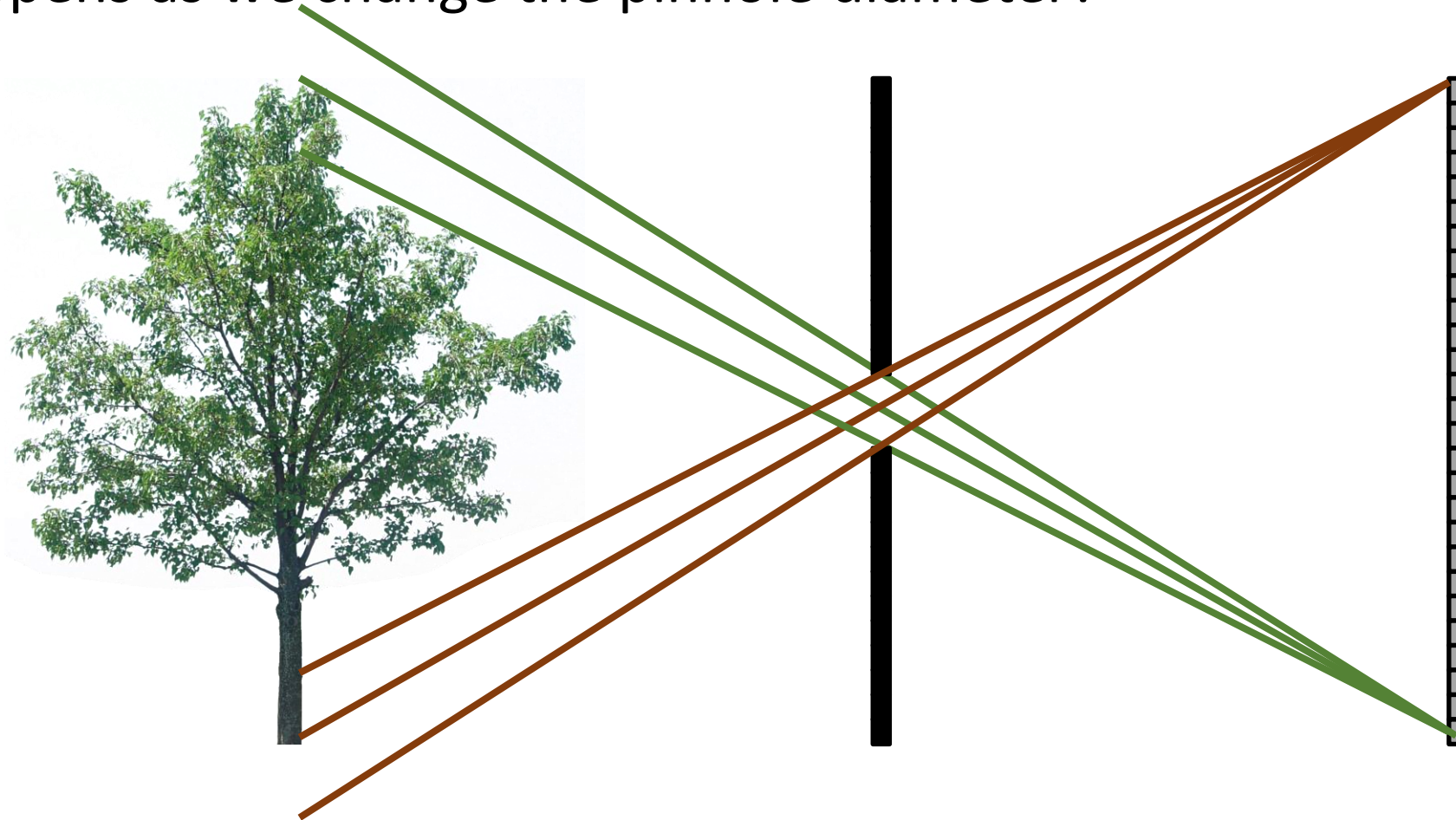
- What happens as we change the pinhole diameter?



Pinhole size

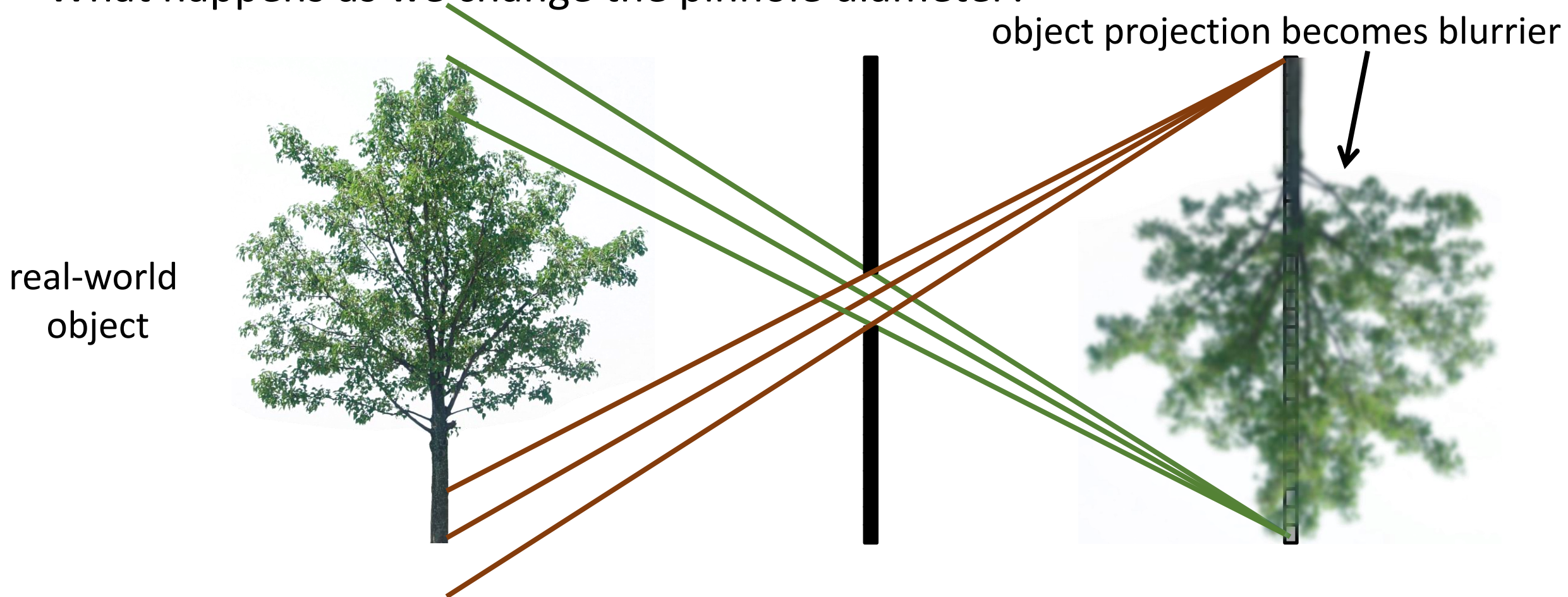
- What happens as we change the pinhole diameter?

real-world
object



Pinhole size

- What happens as we change the pinhole diameter?



What about light efficiency?

real-world
object



pinhole
diameter



focal length f



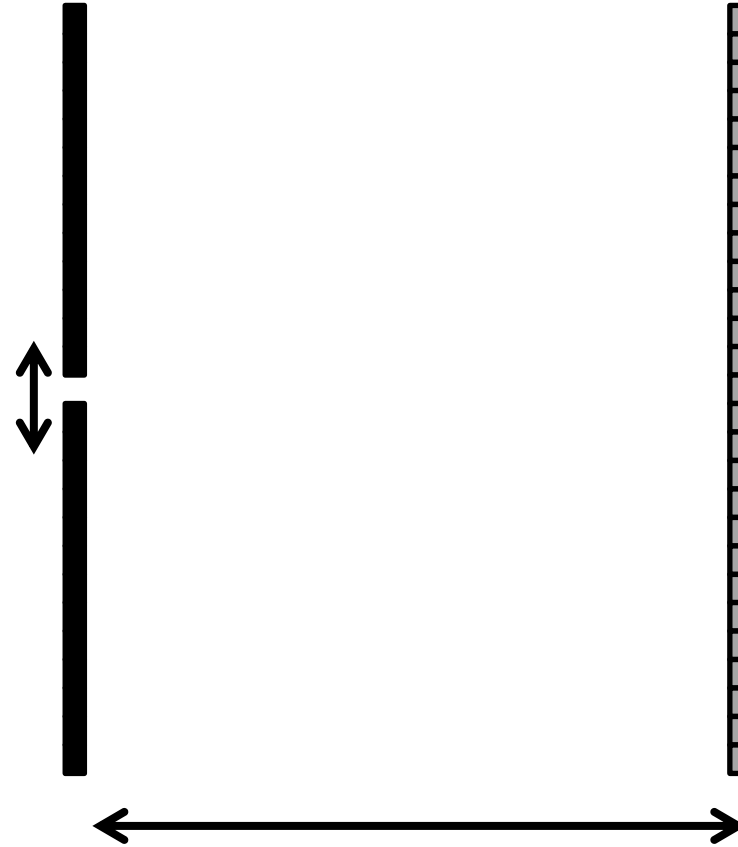
- What is the effect of doubling the pinhole diameter?
- What is the effect of doubling the focal length?

What about light efficiency?

real-world
object

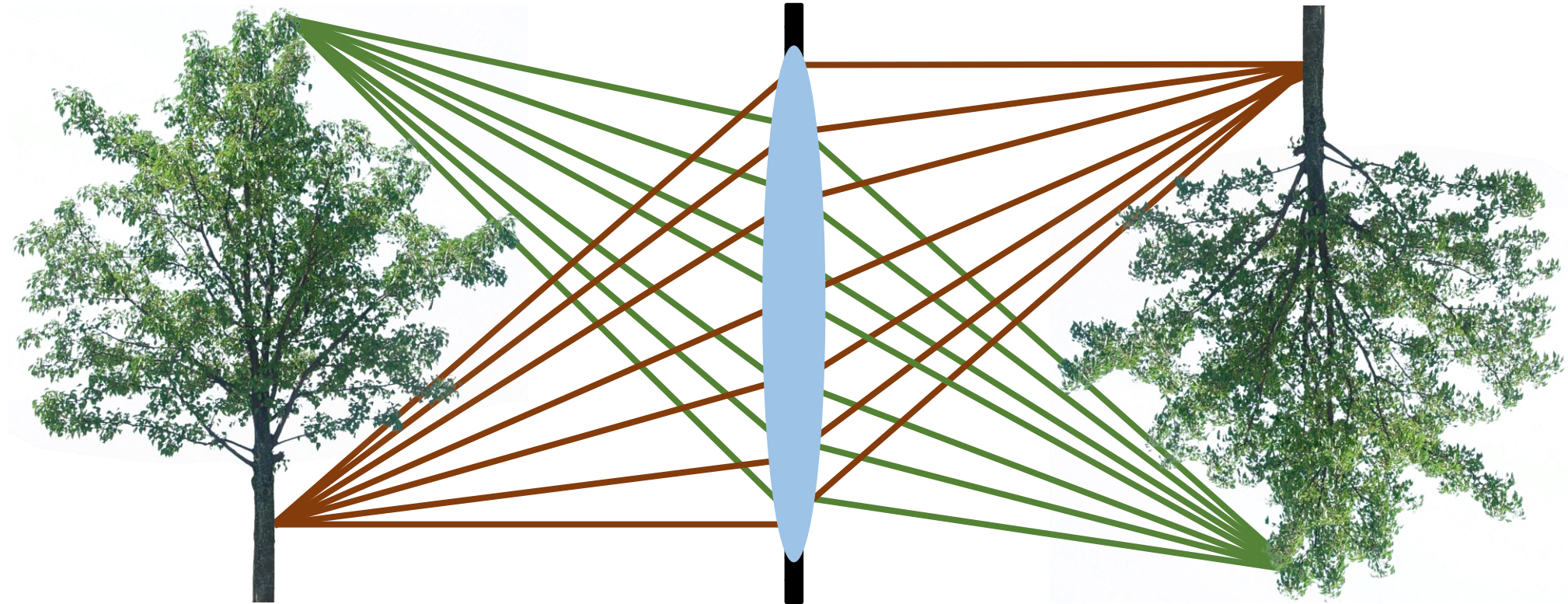


pinhole
diameter



- 2x pinhole diameter \rightarrow 4x light
- 2x focal length \rightarrow $\frac{1}{4}$ x light

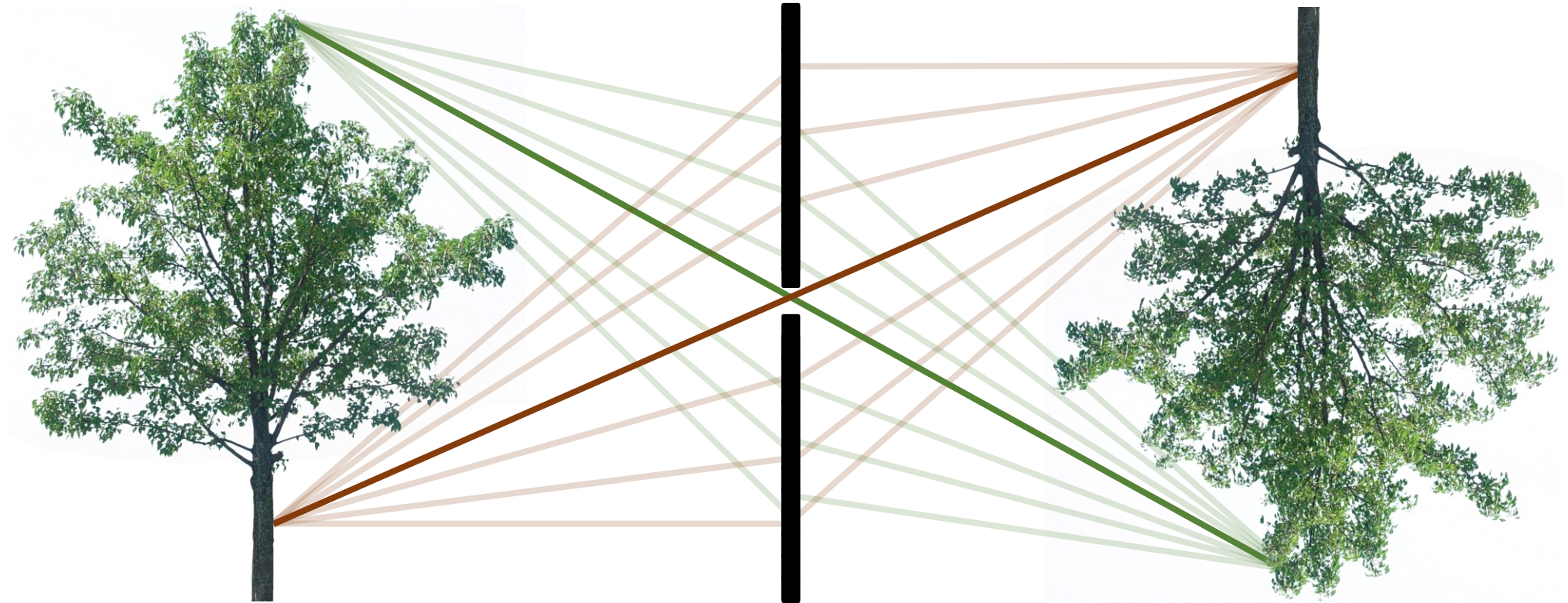
The lens camera



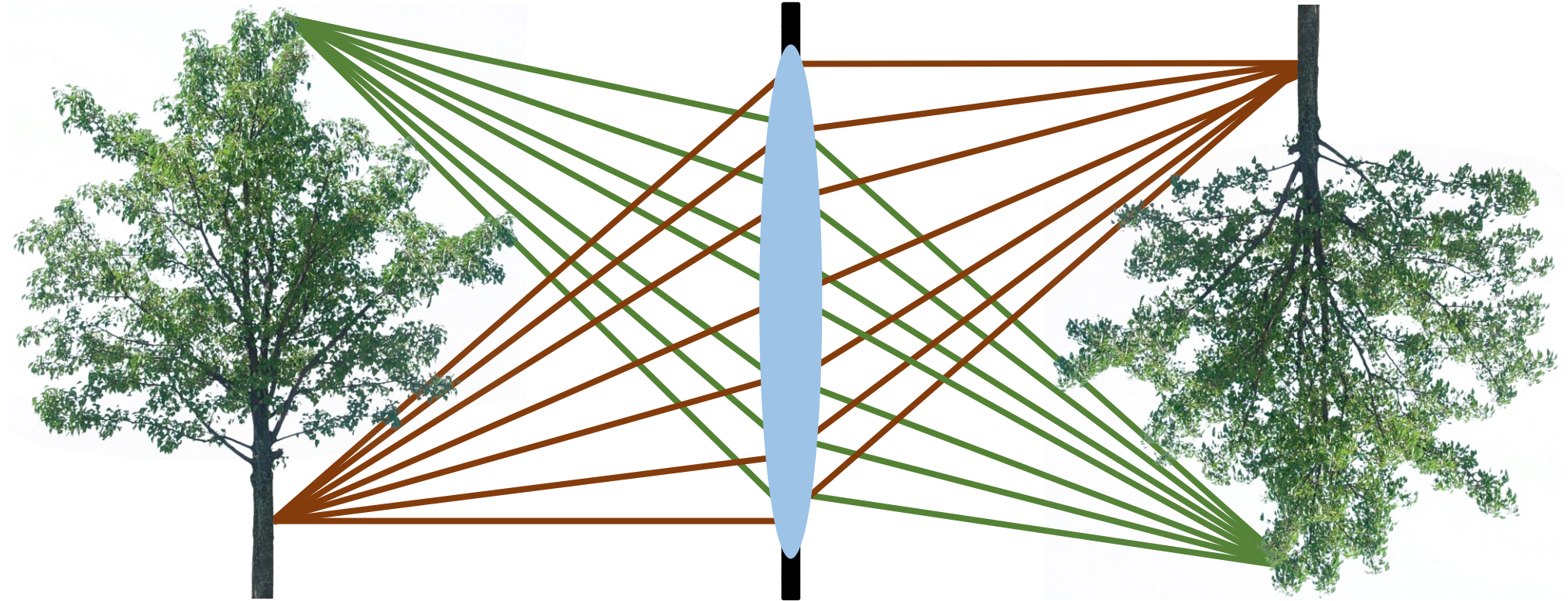
Lenses map “bundles” of rays from points on the scene to the sensor.

How does this mapping work exactly?

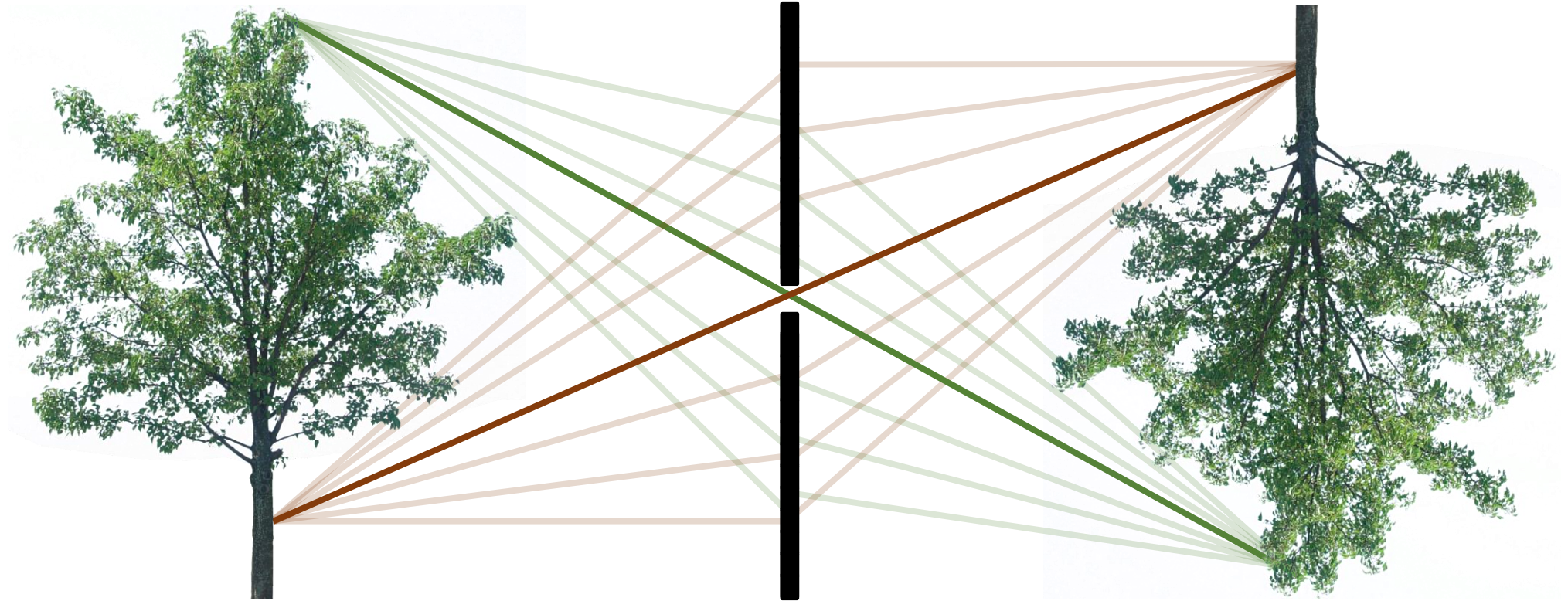
The pinhole camera



The lens camera

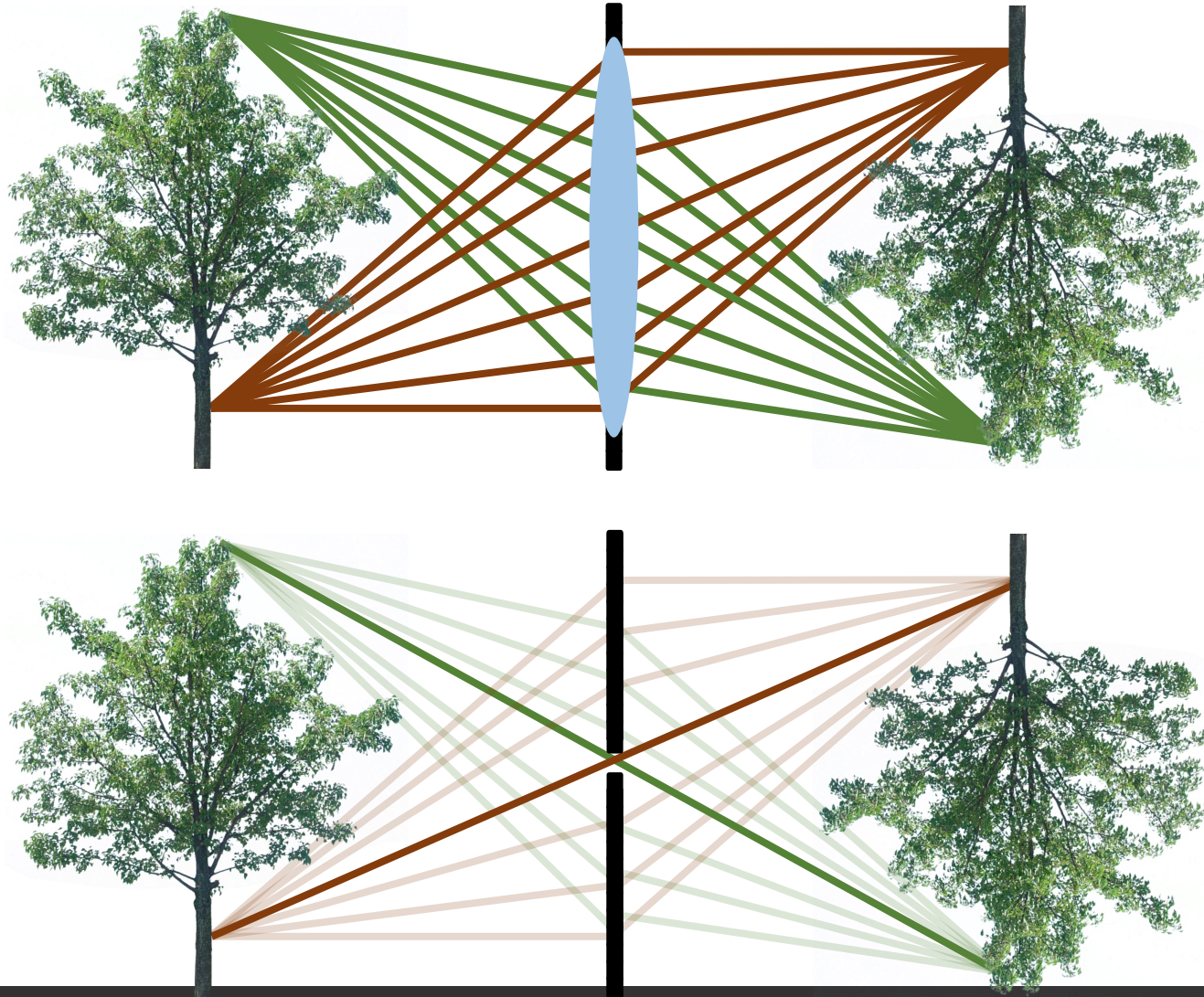


The pinhole camera



Central rays propagate in the same way for both models!

Describing both lens and pinhole cameras

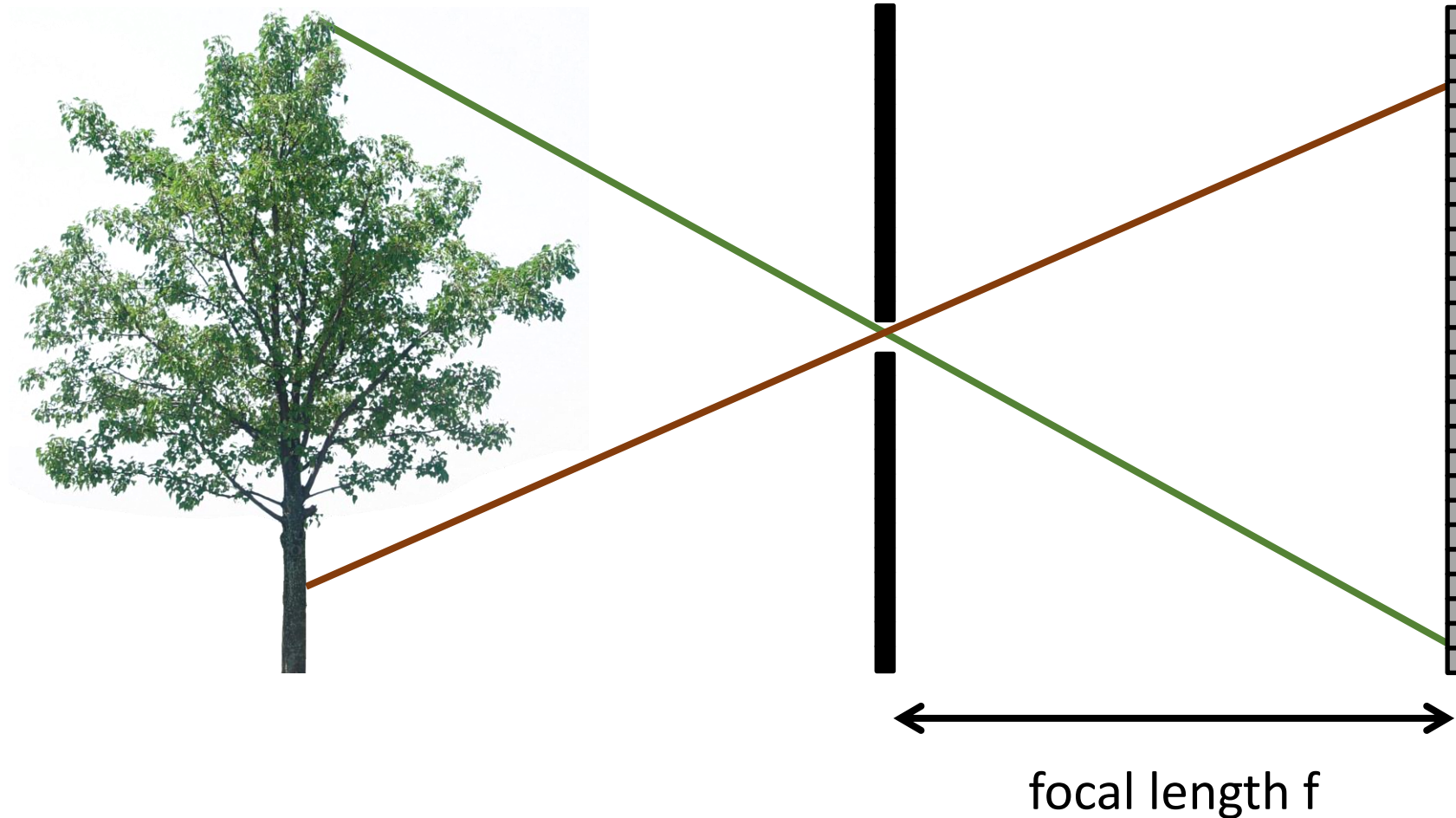


We can derive properties and descriptions that hold for both camera models if:

- We use only central rays.
- We assume the lens camera is in focus.

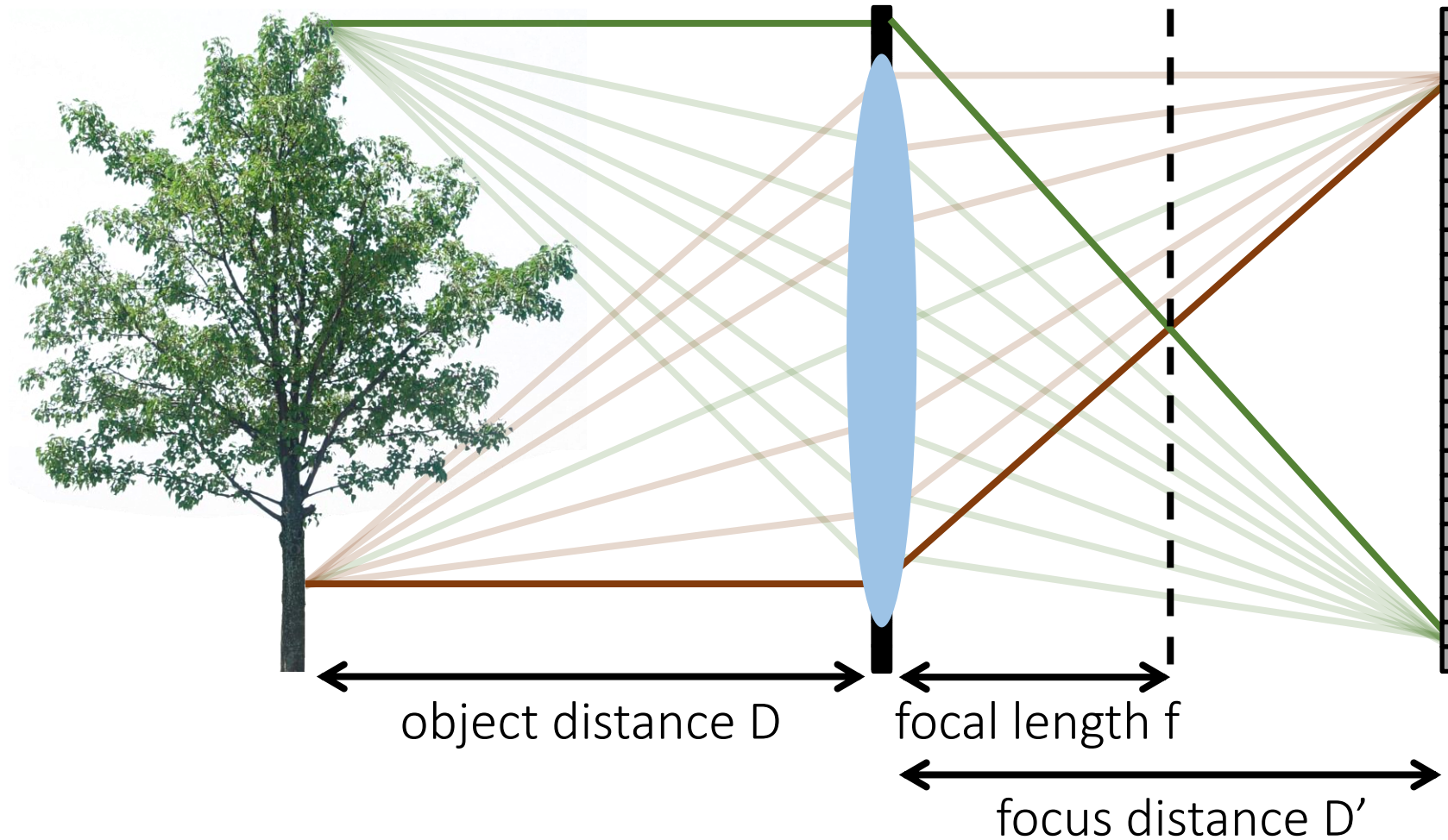
Important difference: focal length

- In a pinhole camera, focal length is distance between aperture and sensor

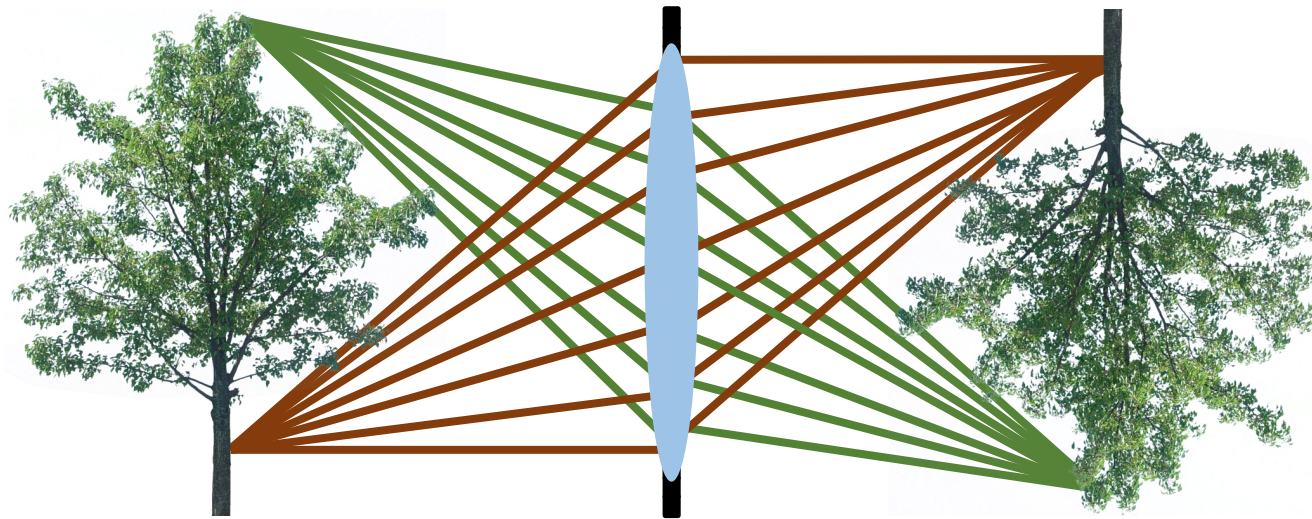


Important difference: focal length

- In a lens camera, focal length is distance where parallel rays intersect

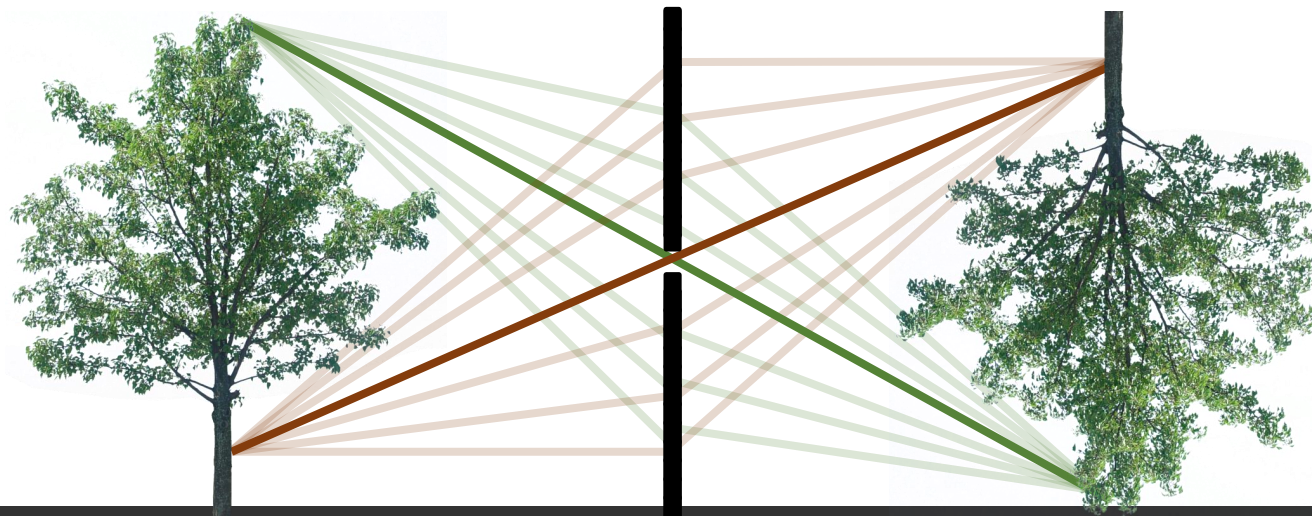


Describing both lens and pinhole cameras



We can derive properties and descriptions that hold for both camera models if:

- We use only central rays.
- We assume the lens camera is in focus.
- We assume that the focus distance of the lens camera is equal to the focal length of the pinhole camera.



Remember: *focal length* f refers to different things for lens and pinhole cameras.

- In this lecture, we use it to refer to the aperture-sensor distance, as in the pinhole camera case.

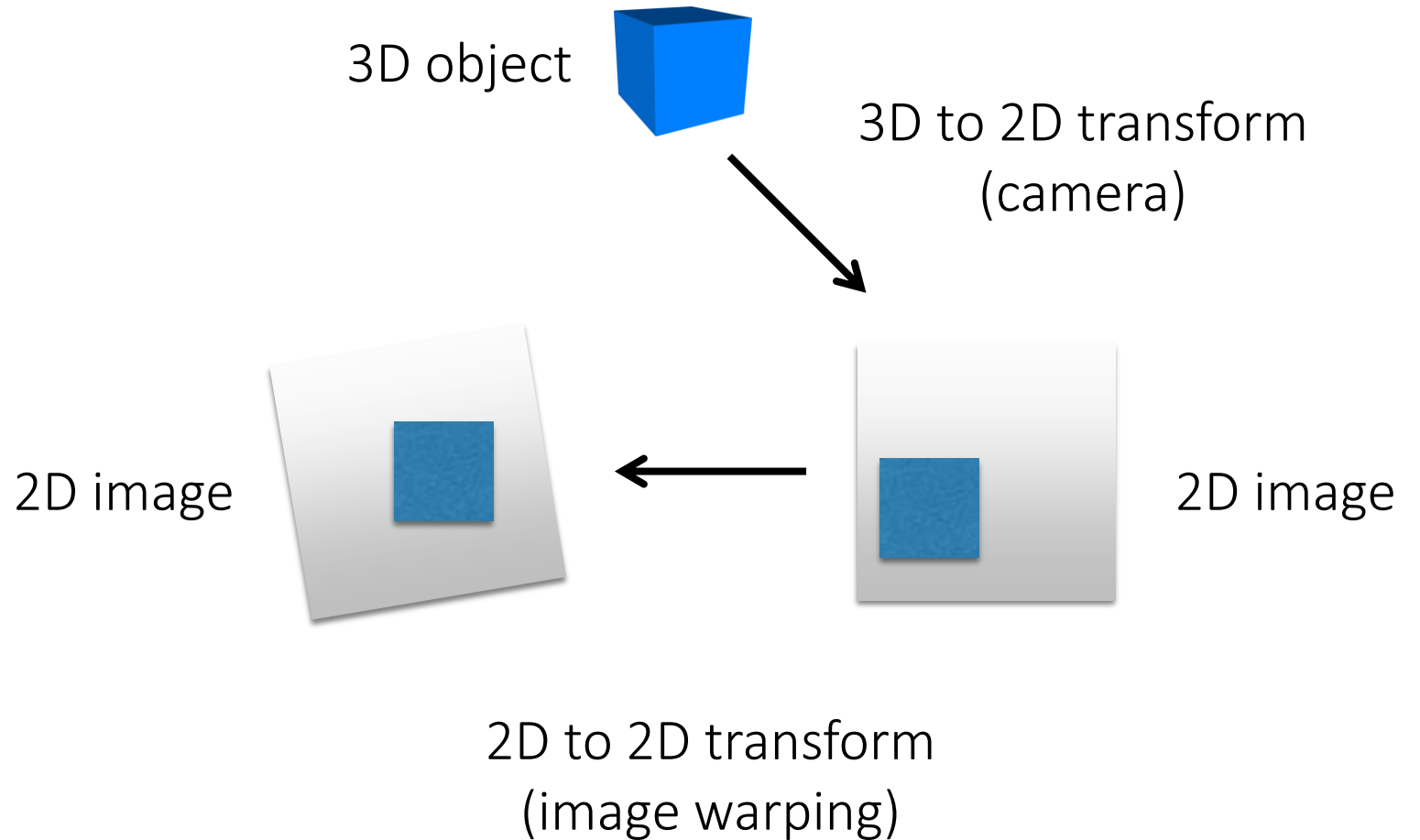
The camera as a coordinate transformation

A camera is a mapping from:

the 3D world

to:

a 2D image



The camera as a coordinate transformation

A camera is a mapping from:

the 3D world

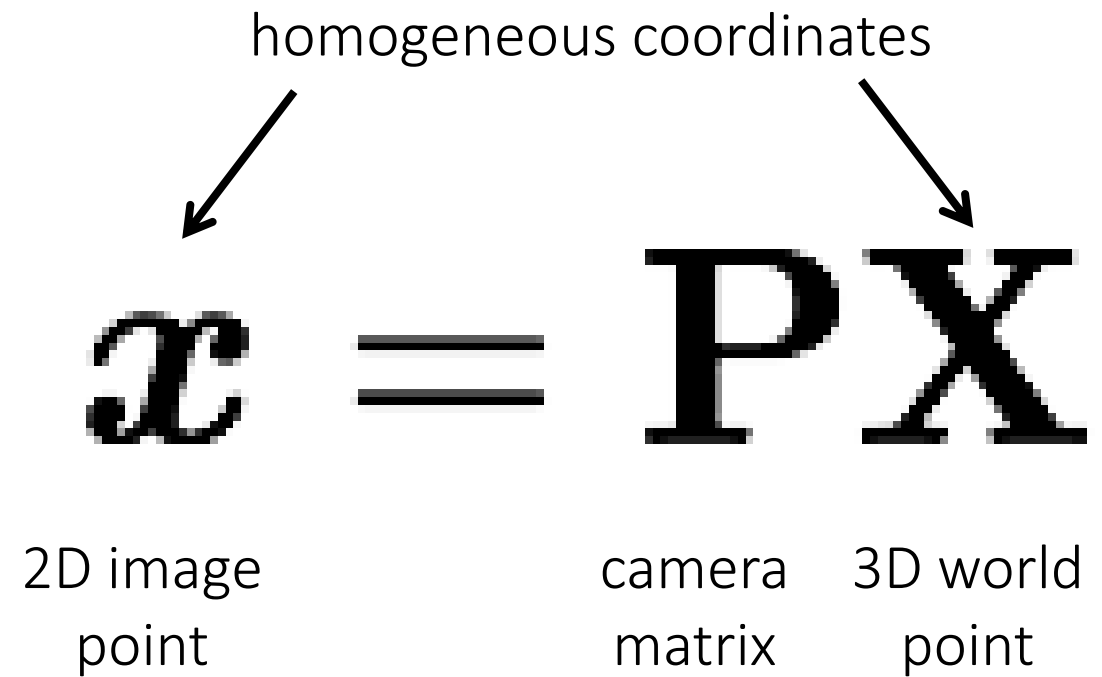
to:

a 2D image

homogeneous coordinates

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

2D image point camera matrix 3D world point



What are the dimensions of each variable?

The camera as a coordinate transformation

$$x = PX$$

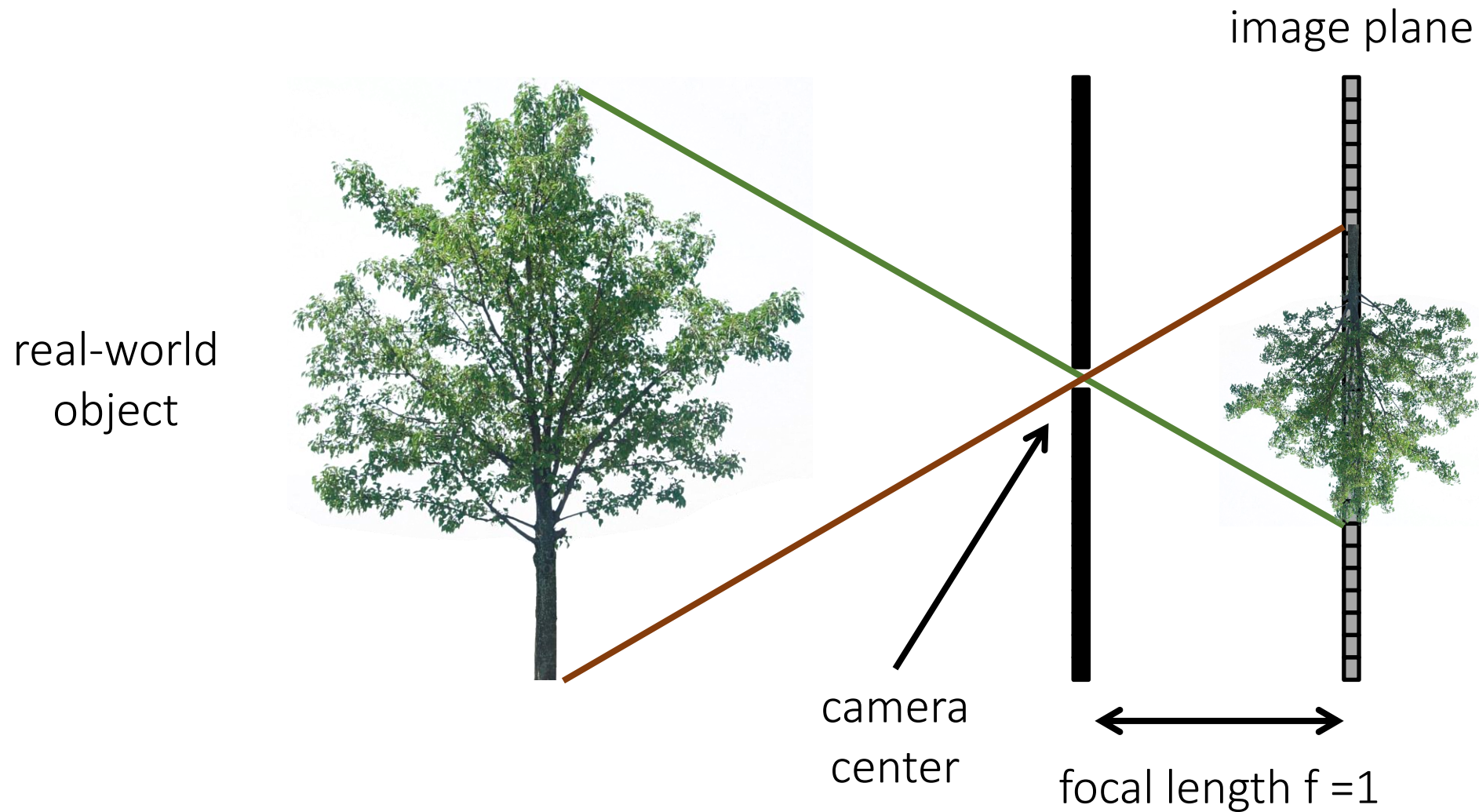
$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

homogeneous
image coordinates
3 x 1

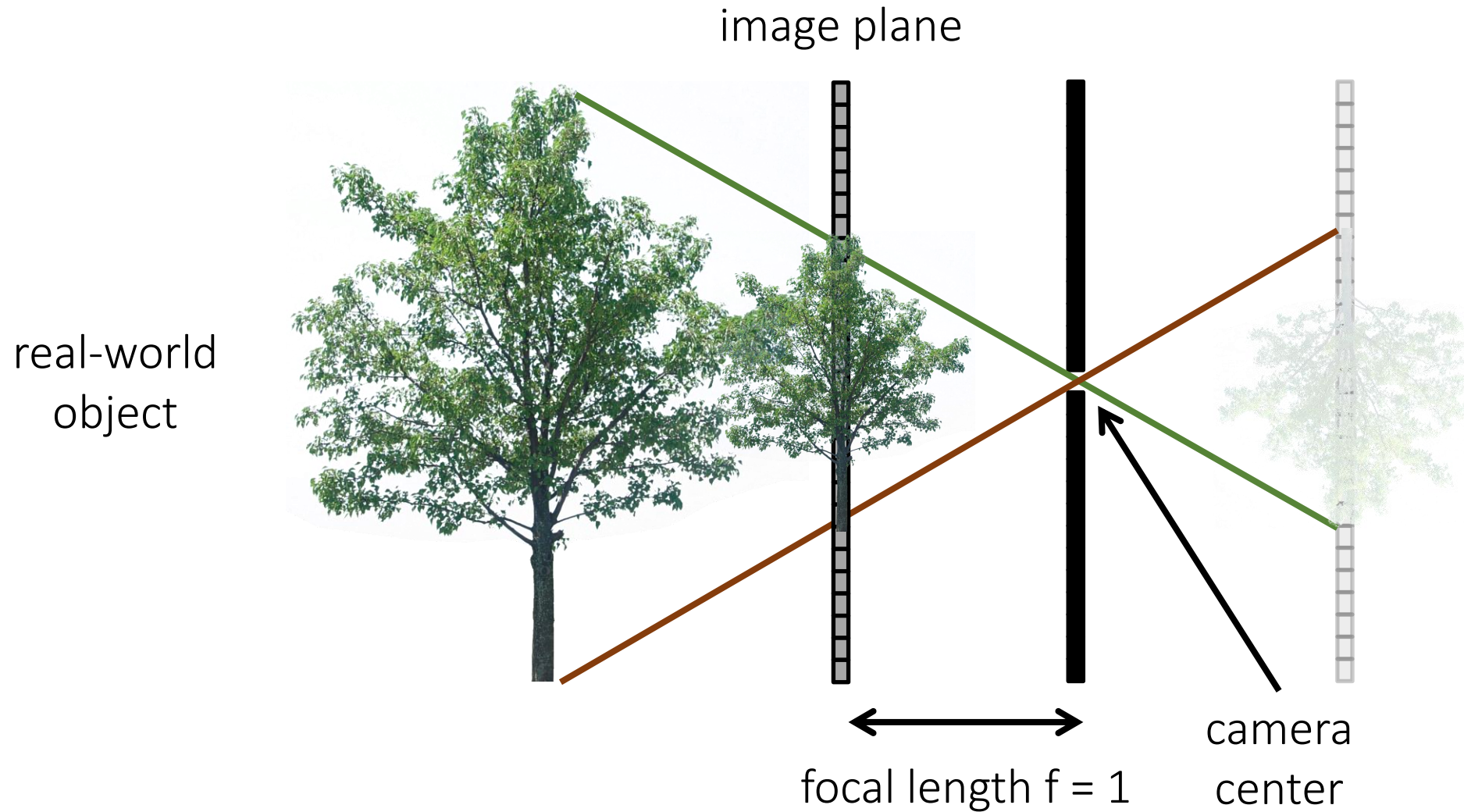
camera
matrix
3 x 4

homogeneous
world coordinates
4 x 1

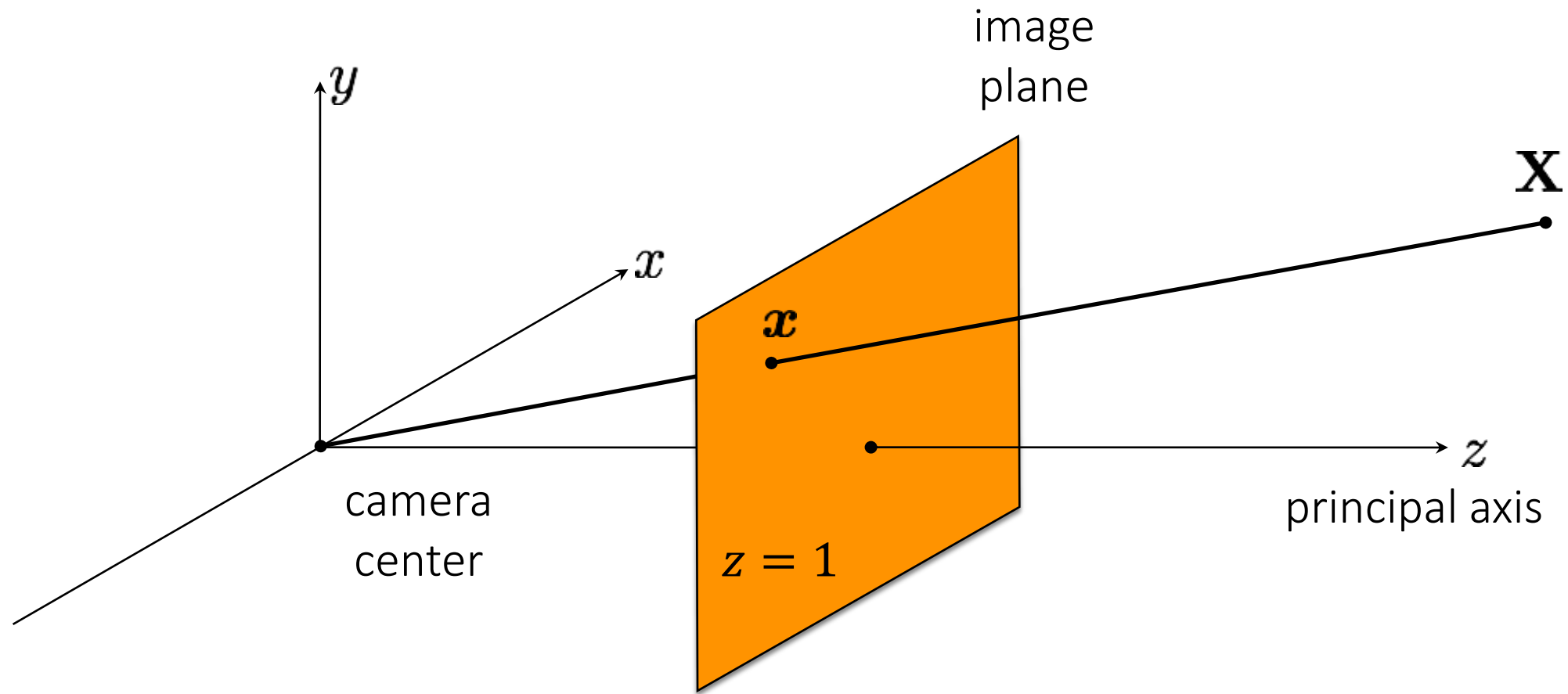
The pinhole camera



The (rearranged) pinhole camera

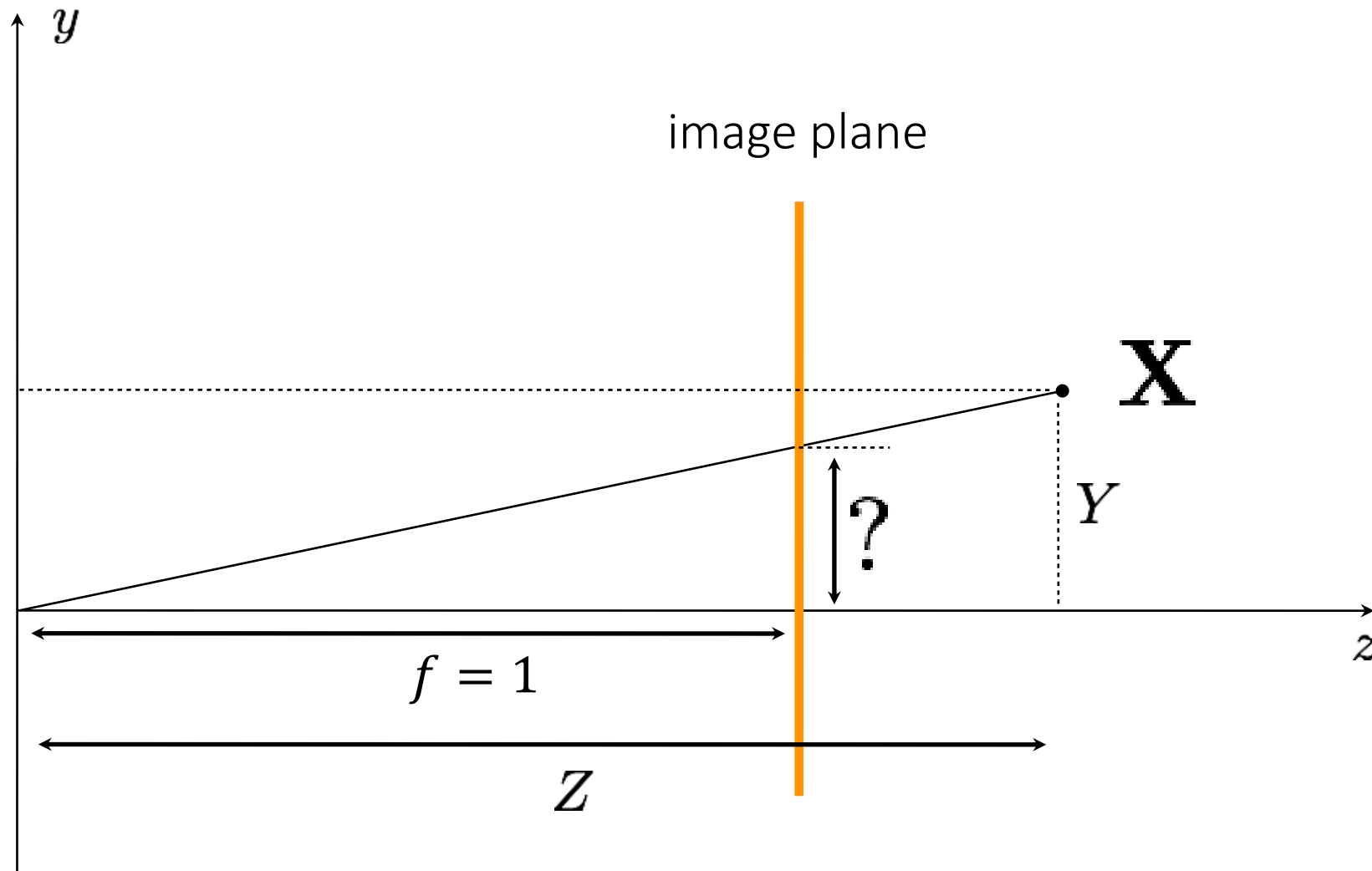


The (rearranged) pinhole camera



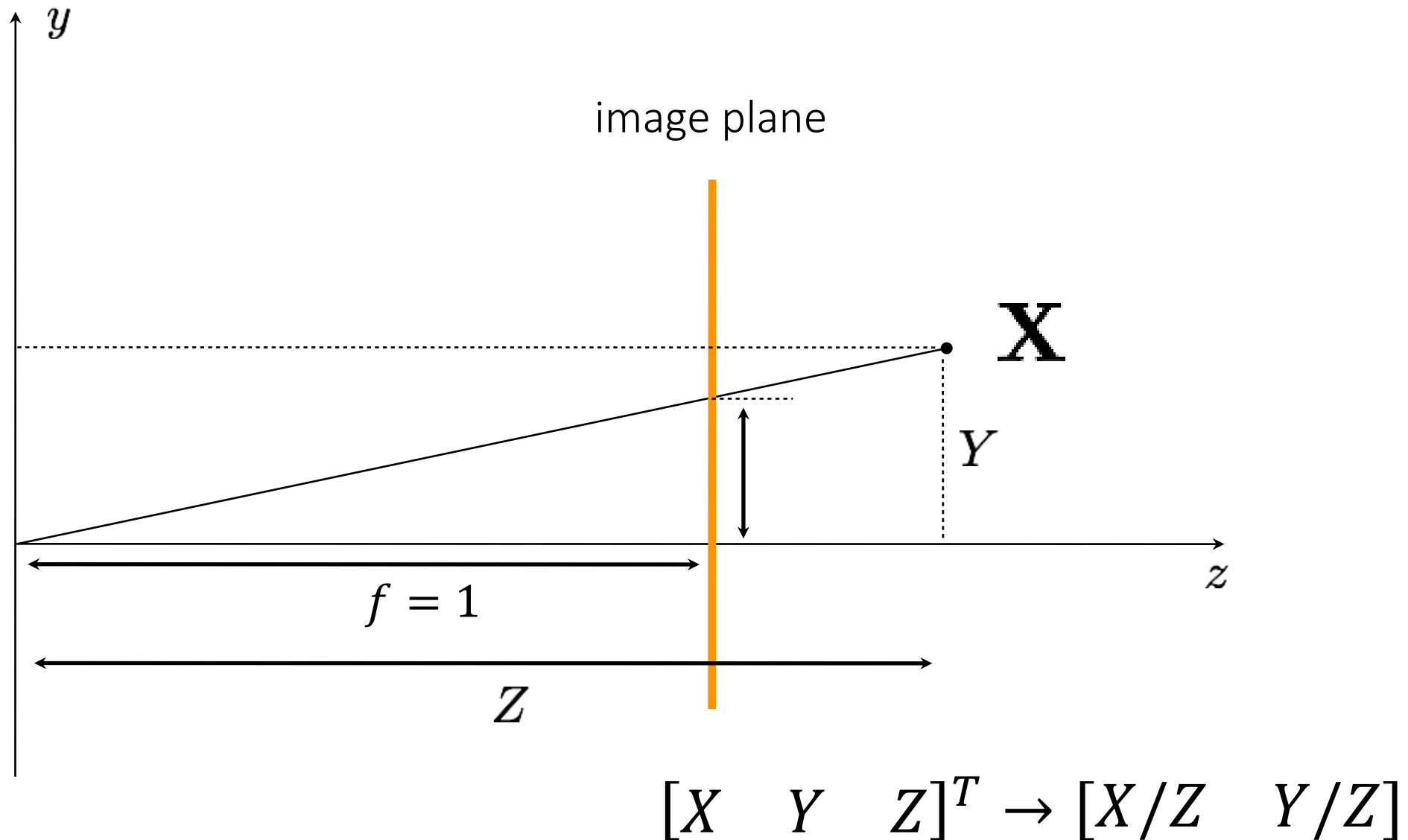
What is the equation for image coordinate \mathbf{x} in terms of \mathbf{X} ?

The 2D view of the (rearranged) pinhole camera

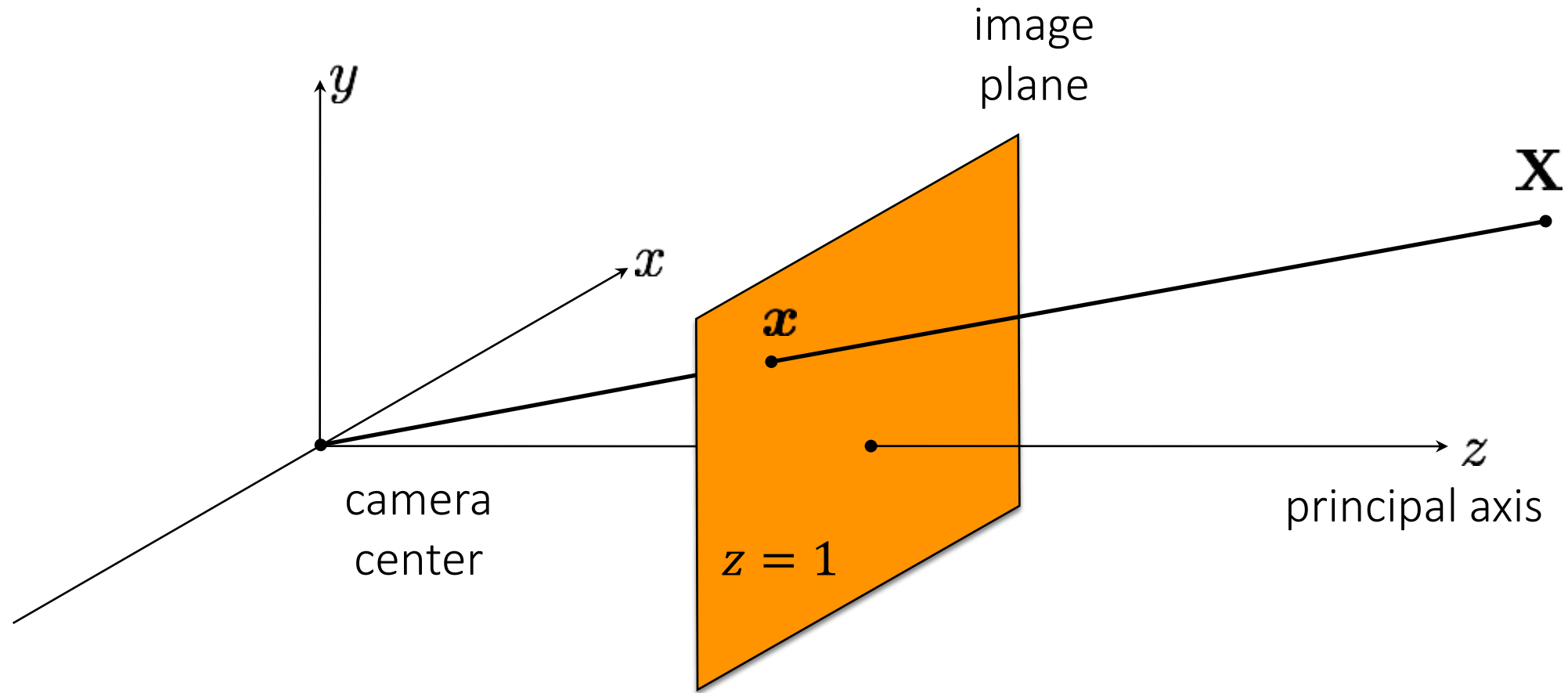


What is the equation for image coordinate x in terms of X ?

The 2D view of the (rearranged) pinhole camera



The (rearranged) pinhole camera



What is the camera matrix \mathbf{P} for a pinhole camera? $\mathbf{x} = \mathbf{P}\mathbf{X}$

The pinhole camera matrix

Relationship from similar triangles:

$$\begin{bmatrix} X & Y & Z \end{bmatrix}^T \rightarrow \begin{bmatrix} X/Z & Y/Z \end{bmatrix}$$

General camera model in *homogeneous coordinates*:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

What does the pinhole camera projection look like?

$$\mathbf{P} = \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$$

The pinhole camera matrix

Relationship from similar triangles:

$$\begin{bmatrix} X & Y & Z \end{bmatrix}^T \rightarrow \begin{bmatrix} X/Z & Y/Z \end{bmatrix}$$

General camera model *in homogeneous coordinates*:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

What does the pinhole camera projection look like?

The perspective
projection matrix

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

The pinhole camera matrix

Relationship from similar triangles:

$$\begin{bmatrix} X & Y & Z \end{bmatrix}^T \rightarrow \begin{bmatrix} X/Z & Y/Z \end{bmatrix}$$

General camera model *in homogeneous coordinates*:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

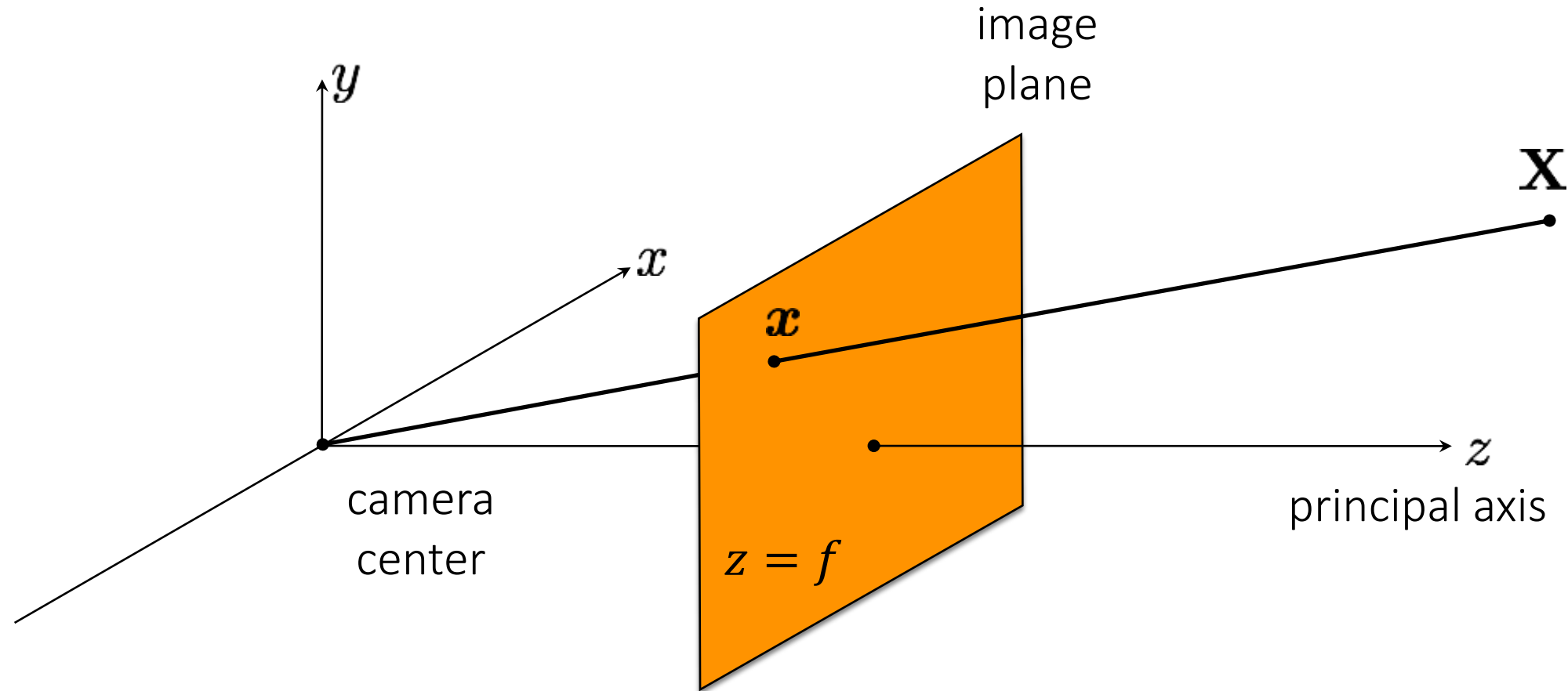
What does the pinhole camera projection look like?

The perspective
projection matrix

$$\mathbf{P} = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array} \right] = [\mathbf{I} \mid \mathbf{0}]$$

alternative way to write
the same thing

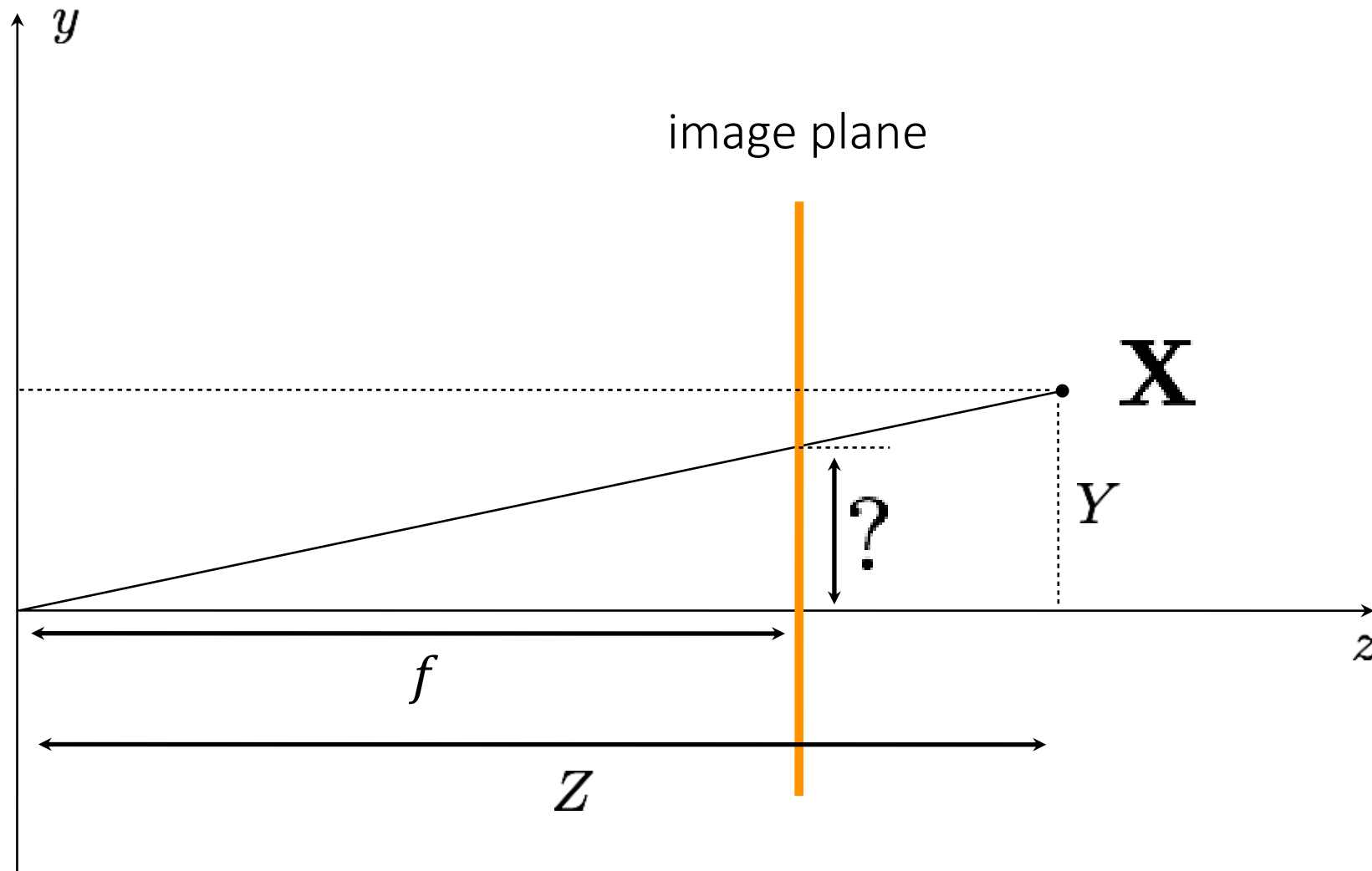
More general case: arbitrary focal length



What is the camera matrix \mathbf{P} for a pinhole camera?

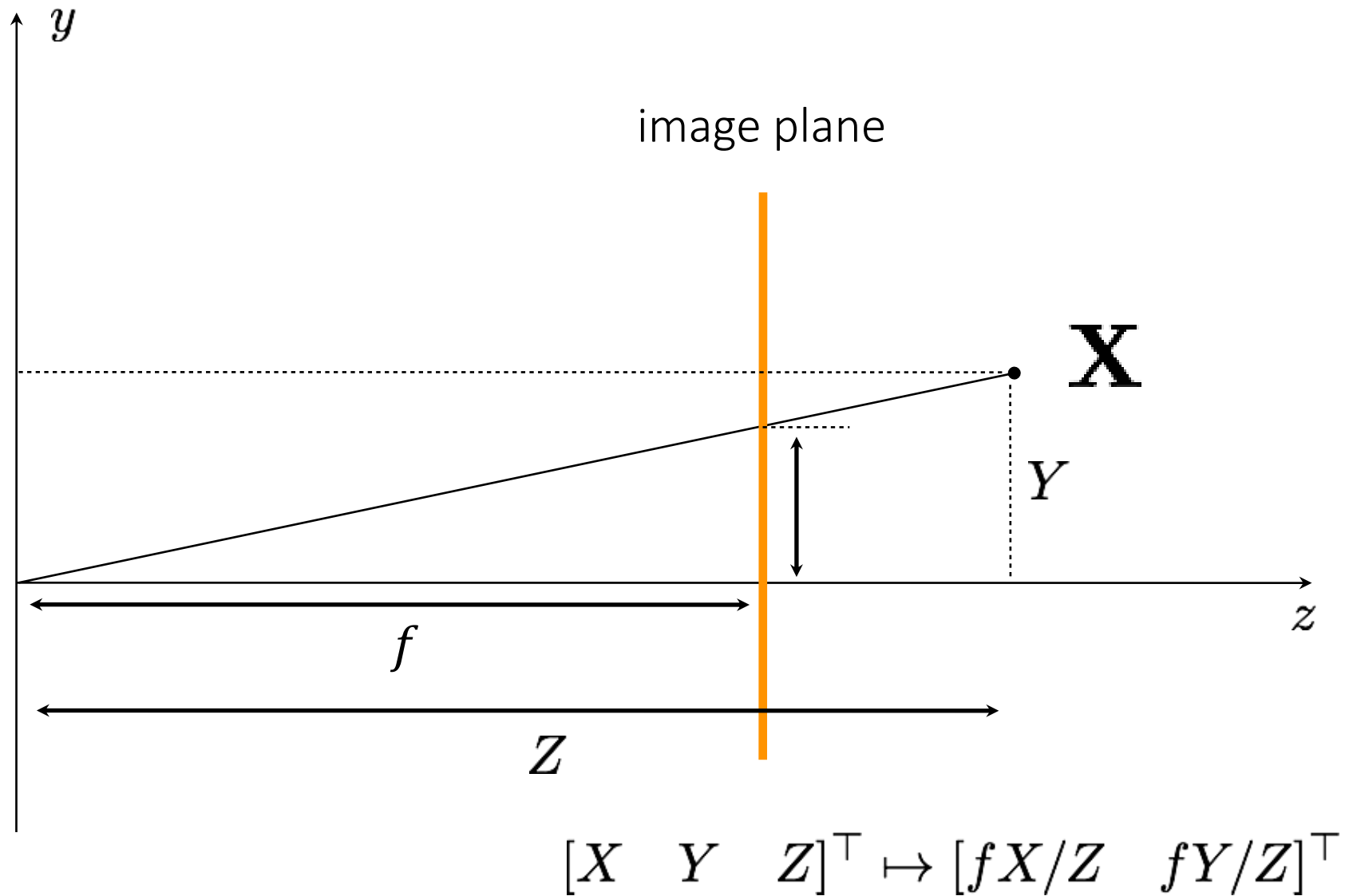
$$\mathbf{x} = \mathbf{P}\mathbf{X}$$

More general (2D) case: arbitrary focal length



What is the equation for image coordinate x in terms of \mathbf{X} ?

More general (2D) case: arbitrary focal length



The pinhole camera matrix for arbitrary focal length

Relationship from similar triangles:

$$[X \quad Y \quad Z]^T \mapsto [fX/Z \quad fY/Z]^T$$

General camera model *in homogeneous coordinates*:

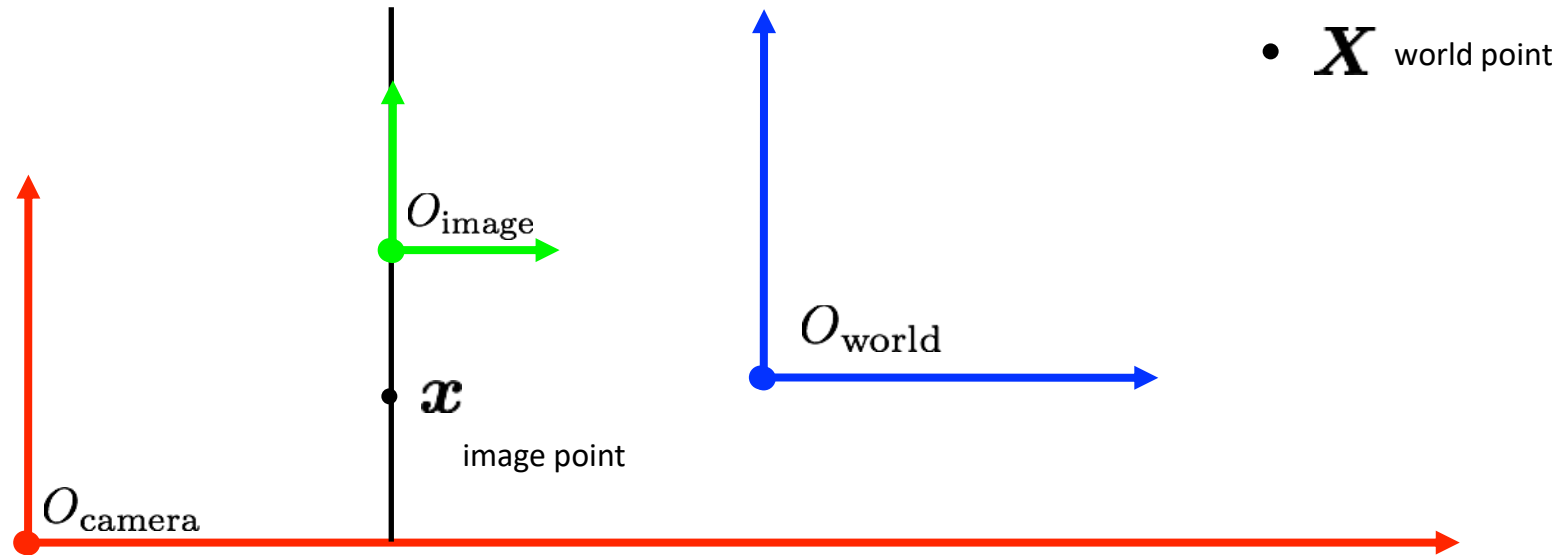
$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

What does the pinhole camera projection look like?

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

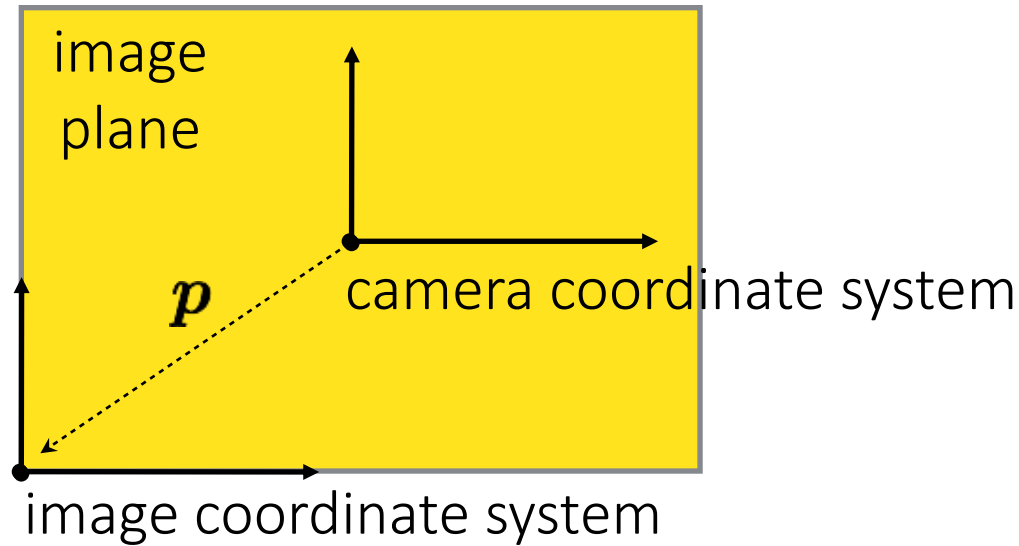
Generalizing the camera matrix

In general, the camera and image have *different* coordinate systems.



Generalizing the camera matrix

In particular, the camera origin and image origin may be different:

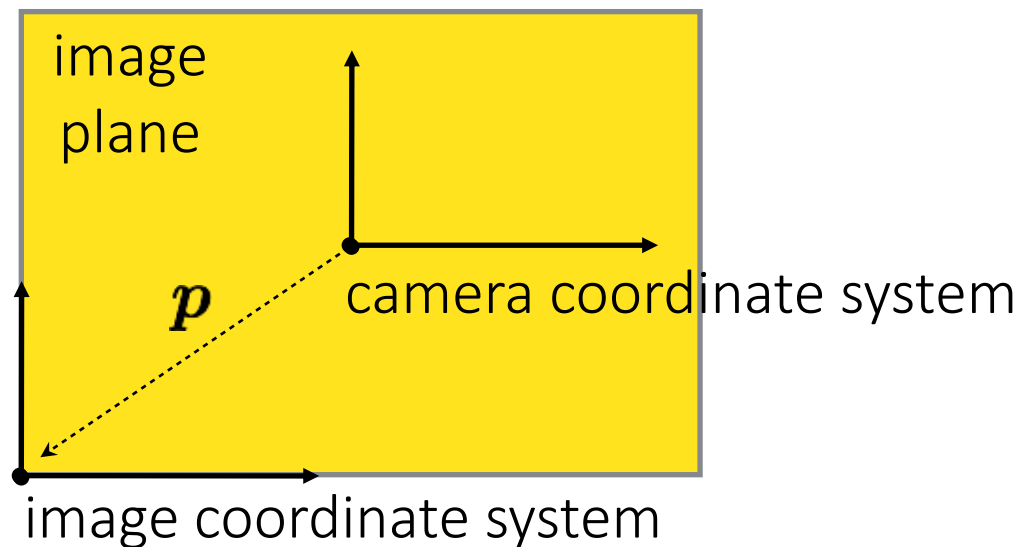


How does the camera matrix change?

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

Generalizing the camera matrix

In particular, the camera origin and image origin may be different:



How does the camera matrix change?

$$\mathbf{P} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

shift vector
transforming
camera origin to
image origin

Camera matrix decomposition


We can decompose the camera matrix like this:

$$\mathbf{P} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & | & 0 \\ 0 & 1 & 0 & | & 0 \\ 0 & 0 & 1 & | & 0 \end{bmatrix}$$

What does each part of the matrix represent?

Camera matrix decomposition

We can decompose the camera matrix like this:

$$\mathbf{P} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & | & 0 \\ 0 & 1 & 0 & | & 0 \\ 0 & 0 & 1 & | & 0 \end{bmatrix}$$


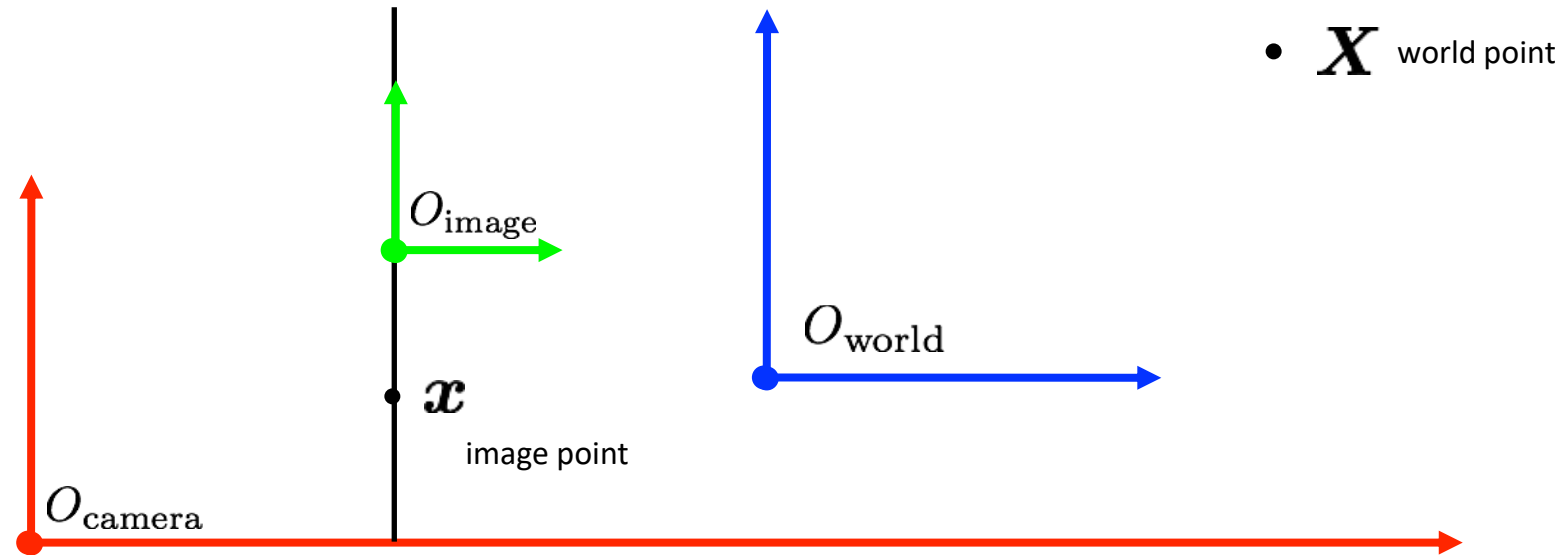
(homogeneous) transformation
from 2D to 2D, accounting for not
unit focal length and origin shift

(homogeneous) perspective projection
from 3D to 2D, assuming image plane at
 $z = 1$ and shared camera/image origin

Also written as: $\mathbf{P} = \mathbf{K}[\mathbf{I}|\mathbf{0}]$ where $\mathbf{K} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}$

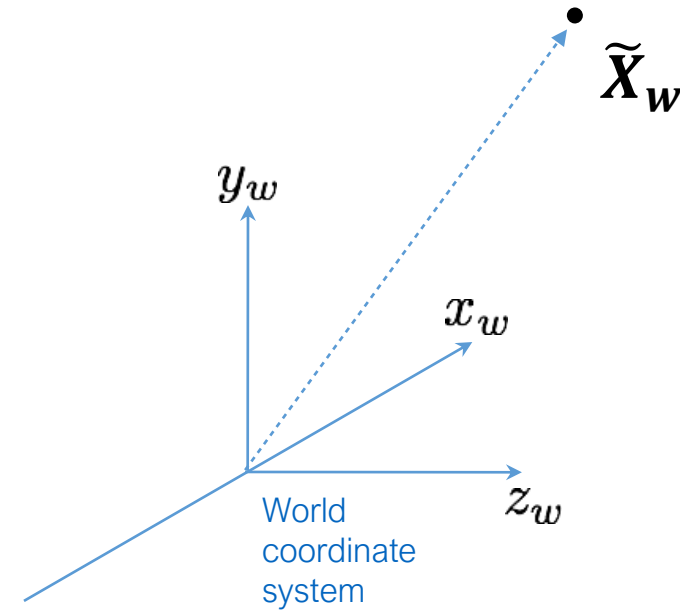
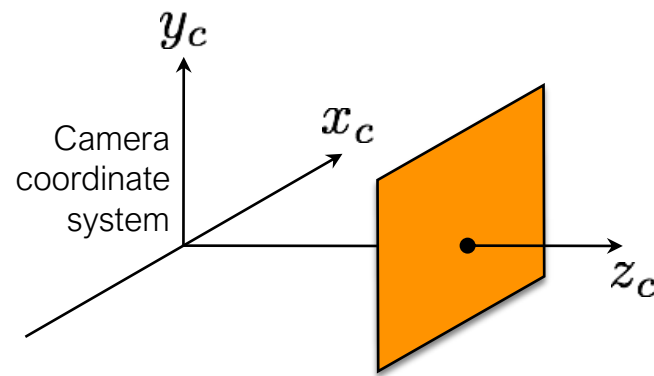
Generalizing the camera matrix

In general, there are *three*, generally different, coordinate systems.



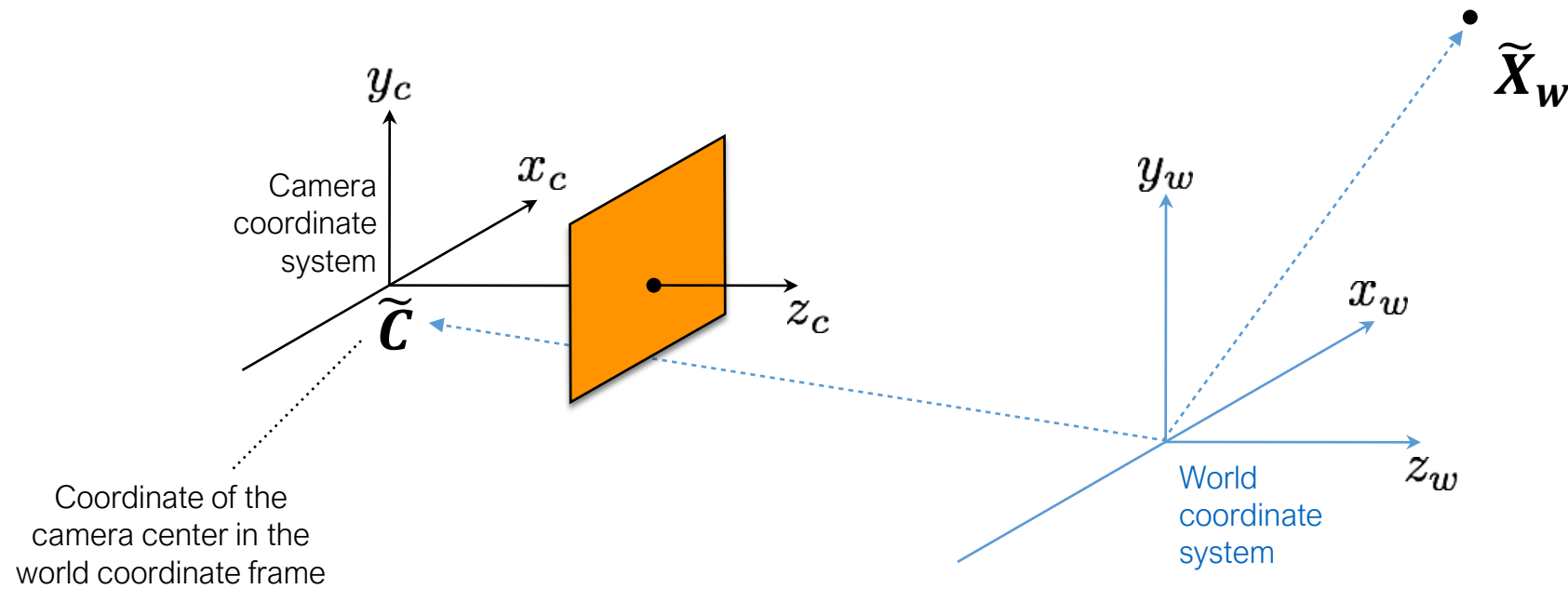
We need to know the transformations between them.

World-to-camera coordinate system transformation

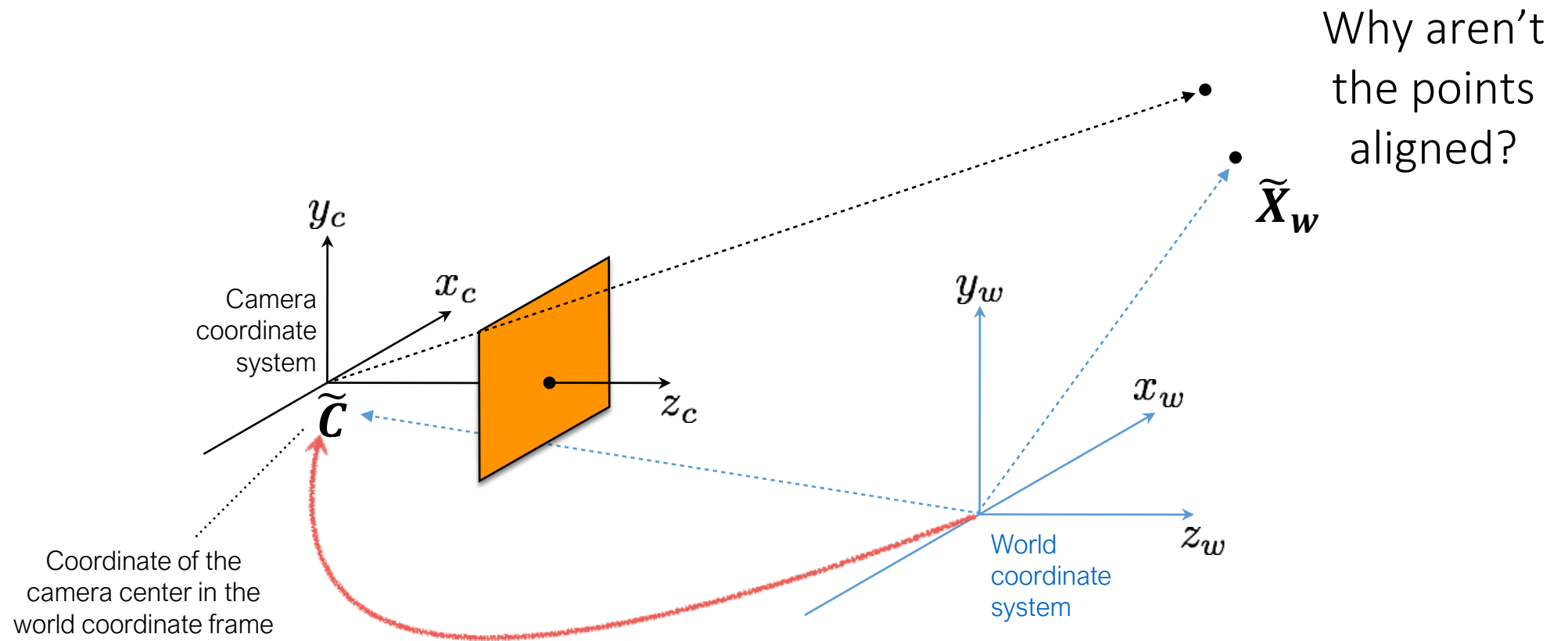


tilde means
heterogeneous
coordinates

World-to-camera coordinate system transformation

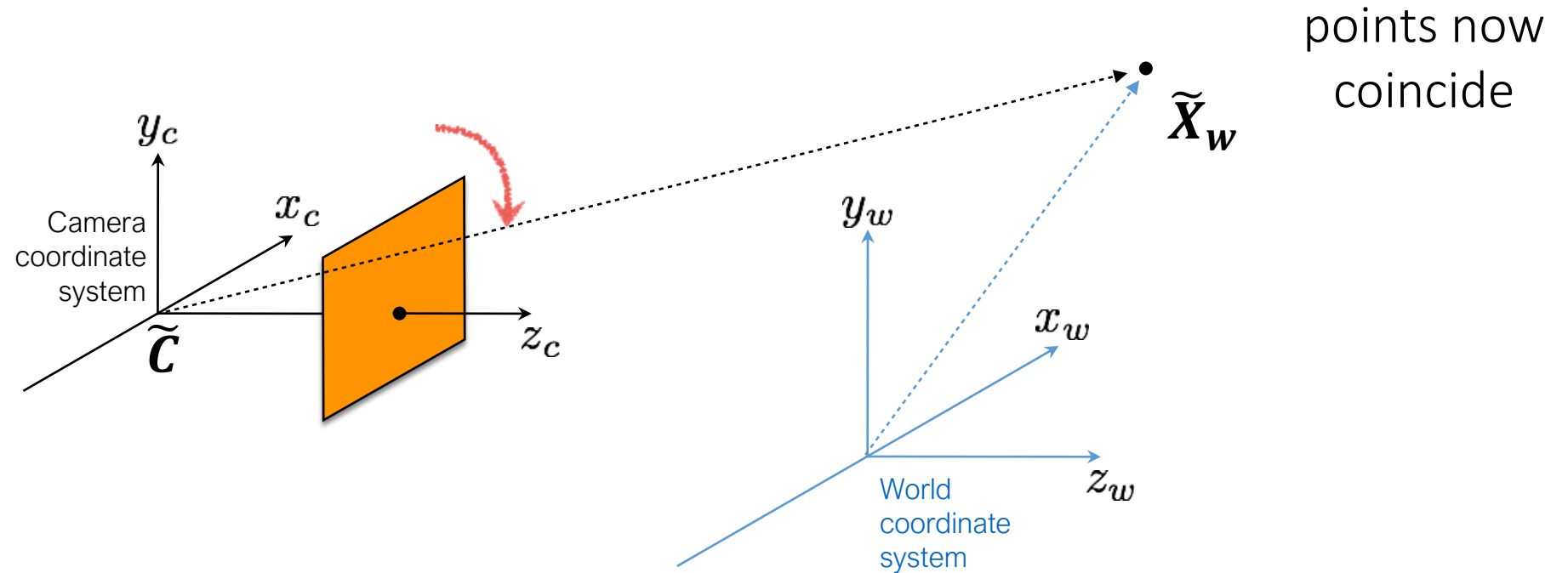


World-to-camera coordinate system transformation



translate

World-to-camera coordinate system transformation



$$\underset{\text{rotate}}{R} \cdot (\underset{\text{translate}}{\tilde{\mathbf{X}}_w - \tilde{\mathbf{c}}})$$

Modeling the coordinate system transformation

In heterogeneous coordinates, we have:

$$\tilde{\mathbf{X}}_c = \mathbf{R} \cdot (\tilde{\mathbf{X}}_w - \tilde{\mathbf{C}})$$

How do we write this transformation in homogeneous coordinates?

Modeling the coordinate system transformation

In heterogeneous coordinates, we have:

$$\tilde{\mathbf{X}}_c = \mathbf{R} \cdot (\tilde{\mathbf{X}}_w - \tilde{\mathbf{C}})$$

In homogeneous coordinates, we have:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & -\mathbf{RC} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \quad \text{or} \quad \mathbf{X}_c = \begin{bmatrix} \mathbf{R} & -\mathbf{RC} \\ \mathbf{0} & 1 \end{bmatrix} \mathbf{X}_w$$

Incorporating the transform in the camera matrix

The previous camera matrix is for homogeneous 3D coordinates in camera coordinate system:

$$\mathbf{x} = \mathbf{P}\mathbf{X}_c = \mathbf{K}[\mathbf{I}|\mathbf{0}]\mathbf{X}_c$$

We also just derived:

$$\mathbf{X}_c = \begin{bmatrix} \mathbf{R} & -\mathbf{R}\tilde{\mathbf{C}} \\ \mathbf{0} & 1 \end{bmatrix} \mathbf{X}_w$$

Putting it all together

We can write everything into a single projection:

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

The camera matrix now looks like:

$$\mathbf{P} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} [\mathbf{I} \mid \mathbf{0}] \begin{bmatrix} \mathbf{R} & -\mathbf{R}\tilde{\mathbf{C}} \\ \mathbf{0} & 1 \end{bmatrix}$$

intrinsic parameters (3 x 3):
correspond to camera
internals (image-to-image
transformation)

perspective projection (3 x 4):
maps 3D to 2D points
(camera-to-image
transformation)

extrinsic parameters (4 x 4):
correspond to camera
externals (world-to-camera
transformation)

Putting it all together


We can write everything into a single projection:

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


The camera matrix now looks like:

$$\mathbf{P} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \left[\mathbf{R} \mid -\mathbf{R}\mathbf{C} \right]$$

intrinsic parameters (3 x 3):
correspond to camera internals
(sensor not at $f = 1$ and origin shift)



extrinsic parameters (3 x 4):
correspond to camera externals
(world-to-image transformation)



General pinhole camera matrix

We can decompose the camera matrix like this:

$$\mathbf{P} = \mathbf{KR}[\mathbf{I} | -\mathbf{C}]$$

(translate first then rotate)

Another way to write the mapping:

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

where $\mathbf{t} = -\mathbf{RC}$

(rotate first then translate)

General pinhole camera matrix

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

$$\mathbf{P} = \underbrace{\begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}}_{\substack{\text{intrinsic} \\ \text{parameters}}} \underbrace{\begin{bmatrix} r_1 & r_2 & r_3 & | & t_1 \\ r_4 & r_5 & r_6 & | & t_2 \\ r_7 & r_8 & r_9 & | & t_3 \end{bmatrix}}_{\substack{\text{extrinsic} \\ \text{parameters}}}$$

$$\mathbf{R} = \begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{bmatrix} \quad \mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}$$

3D rotation

3D translation

More general camera matrices

The following is the standard camera matrix we saw.

$$\mathbf{P} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \quad \left[\mathbf{R} \mid -\mathbf{RC} \right]$$

More general camera matrices

CCD camera: pixels may not be square.

$$\mathbf{P} = \begin{bmatrix} \alpha_x & 0 & p_x \\ 0 & \alpha_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \left[\mathbf{R} \mid -\mathbf{RC} \right]$$

How many degrees of freedom?

More general camera matrices

CCD camera: pixels may not be square.

$$\mathbf{P} = \begin{bmatrix} \alpha_x & 0 & p_x \\ 0 & \alpha_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \left[\mathbf{R} \mid -\mathbf{RC} \right]$$

How many degrees of freedom?

10 DOF

More general camera matrices

Finite projective camera: sensor be skewed.

$$\mathbf{P} = \begin{bmatrix} \alpha_x & s & p_x \\ 0 & \alpha_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \left[\mathbf{R} \mid -\mathbf{RC} \right]$$

How many degrees of freedom?

More general camera matrices

Finite projective camera: sensor be skewed.

$$\mathbf{P} = \begin{bmatrix} \alpha_x & s & p_x \\ 0 & \alpha_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \left[\mathbf{R} \mid -\mathbf{RC} \right]$$

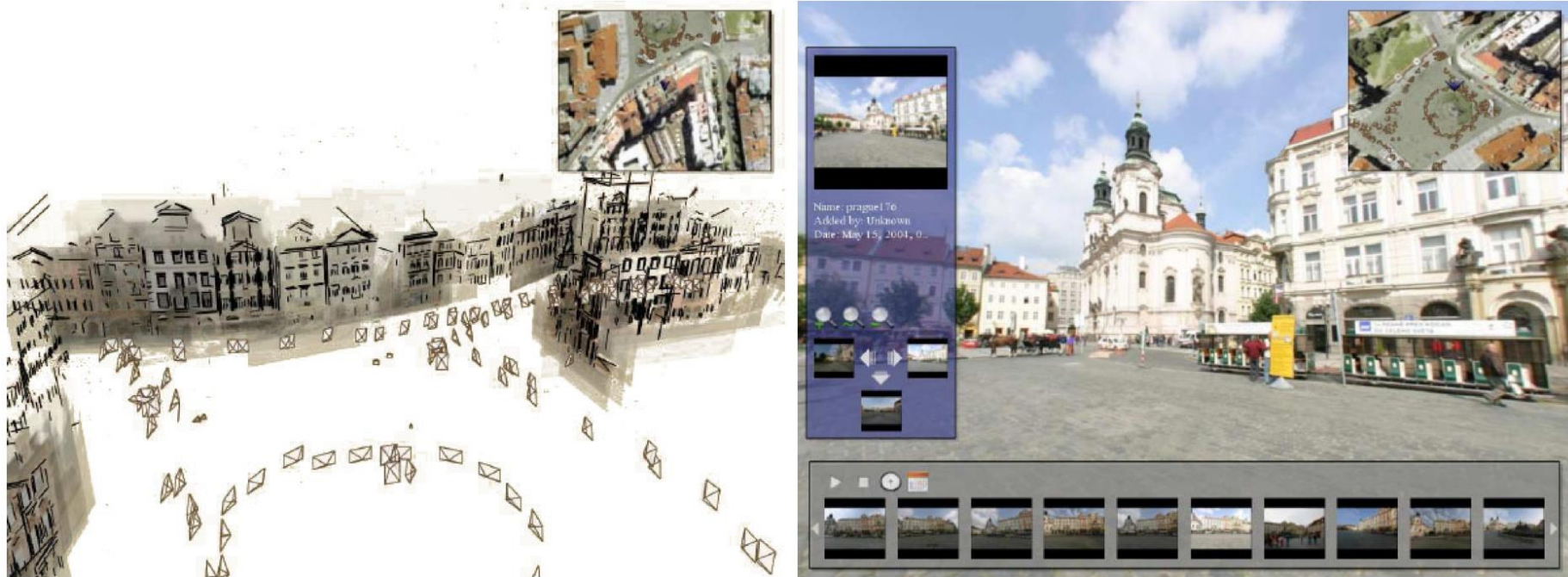
How many degrees of freedom?

11 DOF

Geometric camera calibration

	Structure (scene geometry)	Motion (camera geometry)	Measurements
Camera Calibration (a.k.a. Pose Estimation)	known	estimate	3D to 2D correspondences
Triangulation	estimate	known	2D to 2D coorespondences
Reconstruction	estimate	estimate	2D to 2D coorespondences

Pose Estimation



Given a single image,
estimate the exact position of the photographer

Geometric camera calibration

Given a set of matched points

$$\{\mathbf{X}_i, \mathbf{x}_i\}$$

point in 3D
space

point in the
image

and camera model

$$\mathbf{x} = \mathbf{f}(\mathbf{X}; \mathbf{p}) = \mathbf{P}\mathbf{X}$$

projection
model

parameters

Camera
matrix

Find the (pose) estimate of

P

We'll use a **perspective** camera
model for pose estimation

Same setup as homography estimation
(slightly different derivation here)

Mapping between 3D point and image points

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

What are the unknowns?

Mapping between 3D point and image points

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \text{---} & \mathbf{p}_1^\top & \text{---} \\ \text{---} & \mathbf{p}_2^\top & \text{---} \\ \text{---} & \mathbf{p}_3^\top & \text{---} \end{bmatrix} \begin{bmatrix} | \\ \mathbf{X} \\ | \end{bmatrix}$$

Heterogeneous coordinates

$$x' = \frac{\mathbf{p}_1^\top \mathbf{X}}{\mathbf{p}_3^\top \mathbf{X}} \quad y' = \frac{\mathbf{p}_2^\top \mathbf{X}}{\mathbf{p}_3^\top \mathbf{X}}$$

(non-linear relation between coordinates)

How can we make these relations linear?

Mapping between 3D point and image points

$$x' = \frac{\mathbf{p}_1^\top \mathbf{X}}{\mathbf{p}_3^\top \mathbf{X}} \quad y' = \frac{\mathbf{p}_2^\top \mathbf{X}}{\mathbf{p}_3^\top \mathbf{X}}$$

Make them linear with algebraic manipulation...

$$\mathbf{p}_2^\top \mathbf{X} - \mathbf{p}_3^\top \mathbf{X} y' = 0$$

$$\mathbf{p}_1^\top \mathbf{X} - \mathbf{p}_3^\top \mathbf{X} x' = 0$$

Now we can setup a system of linear equations with multiple point correspondences

Mapping between 3D point and image points

$$\mathbf{p}_2^\top \mathbf{X} - \mathbf{p}_3^\top \mathbf{X} y' = 0$$

$$\mathbf{p}_1^\top \mathbf{X} - \mathbf{p}_3^\top \mathbf{X} x' = 0$$

How do we proceed?

Mapping between 3D point and image points

$$\mathbf{p}_2^\top \mathbf{X} - \mathbf{p}_3^\top \mathbf{X} y' = 0$$

$$\mathbf{p}_1^\top \mathbf{X} - \mathbf{p}_3^\top \mathbf{X} x' = 0$$

In matrix form ...

$$\begin{bmatrix} \mathbf{X}^\top & \mathbf{0} & -x' \mathbf{X}^\top \\ \mathbf{0} & \mathbf{X}^\top & -y' \mathbf{X}^\top \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} = \mathbf{0}$$

How do we proceed?

Mapping between 3D point and image points

$$\mathbf{p}_2^\top \mathbf{X} - \mathbf{p}_3^\top \mathbf{X} y' = 0$$

$$\mathbf{p}_1^\top \mathbf{X} - \mathbf{p}_3^\top \mathbf{X} x' = 0$$

In matrix form ...

$$\begin{bmatrix} \mathbf{X}^\top & \mathbf{0} & -x' \mathbf{X}^\top \\ \mathbf{0} & \mathbf{X}^\top & -y' \mathbf{X}^\top \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \mathbf{0}$$

For N points ...

$$\begin{bmatrix} \mathbf{X}_1^\top & \mathbf{0} & -x'_1 \mathbf{X}_1^\top \\ \mathbf{0} & \mathbf{X}_1^\top & -y'_1 \mathbf{X}_1^\top \\ \vdots & \vdots & \vdots \\ \mathbf{X}_N^\top & \mathbf{0} & -x'_N \mathbf{X}_N^\top \\ \mathbf{0} & \mathbf{X}_N^\top & -y'_N \mathbf{X}_N^\top \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \mathbf{0}$$

*How do we solve
this system?*

Mapping between 3D point and image points

Solve for camera matrix by

$$\hat{\mathbf{x}} = \arg \min_{\mathbf{x}} \|\mathbf{A}\mathbf{x}\|^2 \text{ subject to } \|\mathbf{x}\|^2 = 1$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{X}_1^\top & \mathbf{0} & -x' \mathbf{X}_1^\top \\ \mathbf{0} & \mathbf{X}_1^\top & -y' \mathbf{X}_1^\top \\ \vdots & \vdots & \vdots \\ \mathbf{X}_N^\top & \mathbf{0} & -x' \mathbf{X}_N^\top \\ \mathbf{0} & \mathbf{X}_N^\top & -y' \mathbf{X}_N^\top \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

SVD!

Mapping between 3D point and image points

Solve for camera matrix by

$$\hat{\mathbf{x}} = \arg \min_{\mathbf{x}} \|\mathbf{A}\mathbf{x}\|^2 \text{ subject to } \|\mathbf{x}\|^2 = 1$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{X}_1^\top & \mathbf{0} & -x' \mathbf{X}_1^\top \\ \mathbf{0} & \mathbf{X}_1^\top & -y' \mathbf{X}_1^\top \\ \vdots & \vdots & \vdots \\ \mathbf{X}_N^\top & \mathbf{0} & -x' \mathbf{X}_N^\top \\ \mathbf{0} & \mathbf{X}_N^\top & -y' \mathbf{X}_N^\top \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

Solution \mathbf{x} is the column of \mathbf{V}
corresponding to smallest singular
value of

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$$

Mapping between 3D point and image points

Solve for camera matrix by

$$\hat{\mathbf{x}} = \arg \min_{\mathbf{x}} \|\mathbf{A}\mathbf{x}\|^2 \text{ subject to } \|\mathbf{x}\|^2 = 1$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{X}_1^\top & \mathbf{0} & -x' \mathbf{X}_1^\top \\ \mathbf{0} & \mathbf{X}_1^\top & -y' \mathbf{X}_1^\top \\ \vdots & \vdots & \vdots \\ \mathbf{X}_N^\top & \mathbf{0} & -x' \mathbf{X}_N^\top \\ \mathbf{0} & \mathbf{X}_N^\top & -y' \mathbf{X}_N^\top \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

Equivalently, solution \mathbf{x} is the Eigenvector corresponding to smallest Eigenvalue of

$$\mathbf{A}^\top \mathbf{A}$$

Mapping between 3D point and image points

Now we have:

$$\mathbf{P} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix}$$

Are we done?

Mapping between 3D point and image points

Almost there ...

$$\mathbf{P} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix}$$

How do you get the intrinsic and extrinsic parameters from the projection matrix?

Decomposition of the Camera Matrix

$$\mathbf{P} = \left[\begin{array}{ccc|c} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{array} \right]$$

Decomposition of the Camera Matrix

$$\mathbf{P} = \left[\begin{array}{ccc|c} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{array} \right]$$

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

Decomposition of the Camera Matrix

$$\mathbf{P} = \left[\begin{array}{ccc|c} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{array} \right]$$

$$\begin{aligned} \mathbf{P} &= \mathbf{K}[\mathbf{R}|\mathbf{t}] \\ &= \mathbf{K}[\mathbf{R}| -\mathbf{R}\mathbf{c}] \\ &= [\mathbf{M}| -\mathbf{M}\mathbf{c}] \end{aligned}$$

Decomposition of the Camera Matrix

$$\mathbf{P} = \left[\begin{array}{ccc|c} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{array} \right]$$

$$\begin{aligned} \mathbf{P} &= \mathbf{K}[\mathbf{R}|\mathbf{t}] \\ &= \mathbf{K}[\mathbf{R} | -\mathbf{R}\mathbf{c}] \\ &= [\mathbf{M} | -\mathbf{M}\mathbf{c}] \end{aligned}$$

Find the camera center \mathbf{C}

What is the projection of the camera center?

Find intrinsic \mathbf{K} and rotation \mathbf{R}

Decomposition of the Camera Matrix

$$\mathbf{P} = \left[\begin{array}{ccc|c} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{array} \right]$$

$$\begin{aligned} \mathbf{P} &= \mathbf{K}[\mathbf{R}|\mathbf{t}] \\ &= \mathbf{K}[\mathbf{R} | -\mathbf{R}\mathbf{c}] \\ &= [\mathbf{M} | -\mathbf{M}\mathbf{c}] \end{aligned}$$

Find the camera center \mathbf{c}

$$\mathbf{P}\mathbf{c} = \mathbf{0}$$

How do we compute the camera center from this?

Find intrinsic \mathbf{K} and rotation \mathbf{R}

Decomposition of the Camera Matrix

$$\mathbf{P} = \left[\begin{array}{ccc|c} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{array} \right]$$

$$\begin{aligned} \mathbf{P} &= \mathbf{K}[\mathbf{R}|\mathbf{t}] \\ &= \mathbf{K}[\mathbf{R}|-\mathbf{R}\mathbf{c}] \\ &= [\mathbf{M}|-\mathbf{M}\mathbf{c}] \end{aligned}$$

Find the camera center \mathbf{c}

$$\mathbf{P}\mathbf{c} = \mathbf{0}$$

SVD of \mathbf{P} !

\mathbf{c} is the Eigenvector corresponding to
smallest Eigenvalue

Find intrinsic \mathbf{K} and rotation \mathbf{R}

Decomposition of the Camera Matrix

$$\mathbf{P} = \left[\begin{array}{ccc|c} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{array} \right]$$

$$\begin{aligned}\mathbf{P} &= \mathbf{K}[\mathbf{R}|\mathbf{t}] \\ &= \mathbf{K}[\mathbf{R} | -\mathbf{R}\mathbf{c}] \\ &= [\mathbf{M} | -\mathbf{M}\mathbf{c}]\end{aligned}$$

Find the camera center \mathbf{c}

$$\mathbf{P}\mathbf{c} = \mathbf{0}$$

SVD of \mathbf{P} !

\mathbf{c} is the Eigenvector corresponding to
smallest Eigenvalue

Find intrinsic \mathbf{K} and rotation \mathbf{R}

$$\mathbf{M} = \mathbf{K}\mathbf{R}$$

*Any useful properties of \mathbf{K}
and \mathbf{R} we can use?*

Decomposition of the Camera Matrix

$$\mathbf{P} = \left[\begin{array}{ccc|c} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{array} \right]$$

$$\begin{aligned}\mathbf{P} &= \mathbf{K}[\mathbf{R}|\mathbf{t}] \\ &= \mathbf{K}[\mathbf{R} | -\mathbf{R}\mathbf{c}] \\ &= [\mathbf{M} | -\mathbf{M}\mathbf{c}]\end{aligned}$$

Find the camera center \mathbf{c}

$$\mathbf{P}\mathbf{c} = \mathbf{0}$$

SVD of \mathbf{P} !

\mathbf{c} is the Eigenvector corresponding to
smallest Eigenvalue

Find intrinsic \mathbf{K} and rotation \mathbf{R}

$$\mathbf{M} = \mathbf{K}\mathbf{R}$$

right upper
triangle

orthogonal

*How do we find \mathbf{K}
and \mathbf{R} ?*

Decomposition of the Camera Matrix

$$\mathbf{P} = \left[\begin{array}{ccc|c} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{array} \right]$$

$$\begin{aligned}\mathbf{P} &= \mathbf{K}[\mathbf{R}|\mathbf{t}] \\ &= \mathbf{K}[\mathbf{R} | -\mathbf{R}\mathbf{c}] \\ &= [\mathbf{M} | -\mathbf{M}\mathbf{c}]\end{aligned}$$

Find the camera center \mathbf{c}

$$\mathbf{P}\mathbf{c} = \mathbf{0}$$

SVD of \mathbf{P} !

\mathbf{c} is the Eigenvector corresponding to
smallest Eigenvalue

Find intrinsic \mathbf{K} and rotation \mathbf{R}

$$\mathbf{M} = \mathbf{K}\mathbf{R}$$

QR decomposition

Geometric camera calibration

Given a set of matched points

$$\{\mathbf{X}_i, \mathbf{x}_i\}$$

point in 3D
space

point in the
image

*Where do we get these
matched points from?*

and camera model

$$\mathbf{x} = \mathbf{f}(\mathbf{X}; \mathbf{p}) = \mathbf{P}\mathbf{X}$$

projection
model

parameters

Camera
matrix

Find the (pose) estimate of

\mathbf{P}

We'll use a **perspective** camera
model for pose estimation

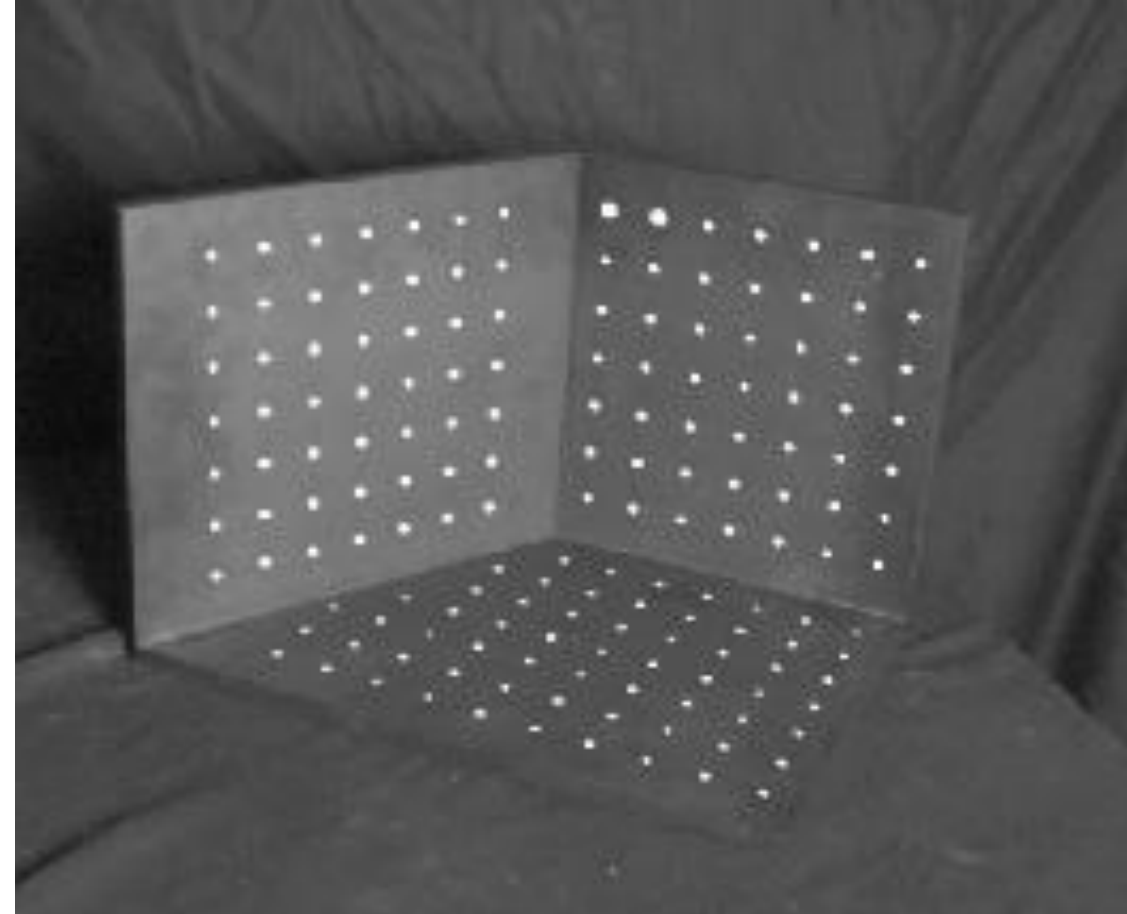
Calibration using a reference object

Place a known object in the scene:

- identify correspondences between image and scene
- compute mapping from scene to image

Issues:

- must know geometry very accurately
- must know 3D->2D correspondence



Geometric camera calibration

Advantages:

- Very simple to formulate.
- Analytical solution.

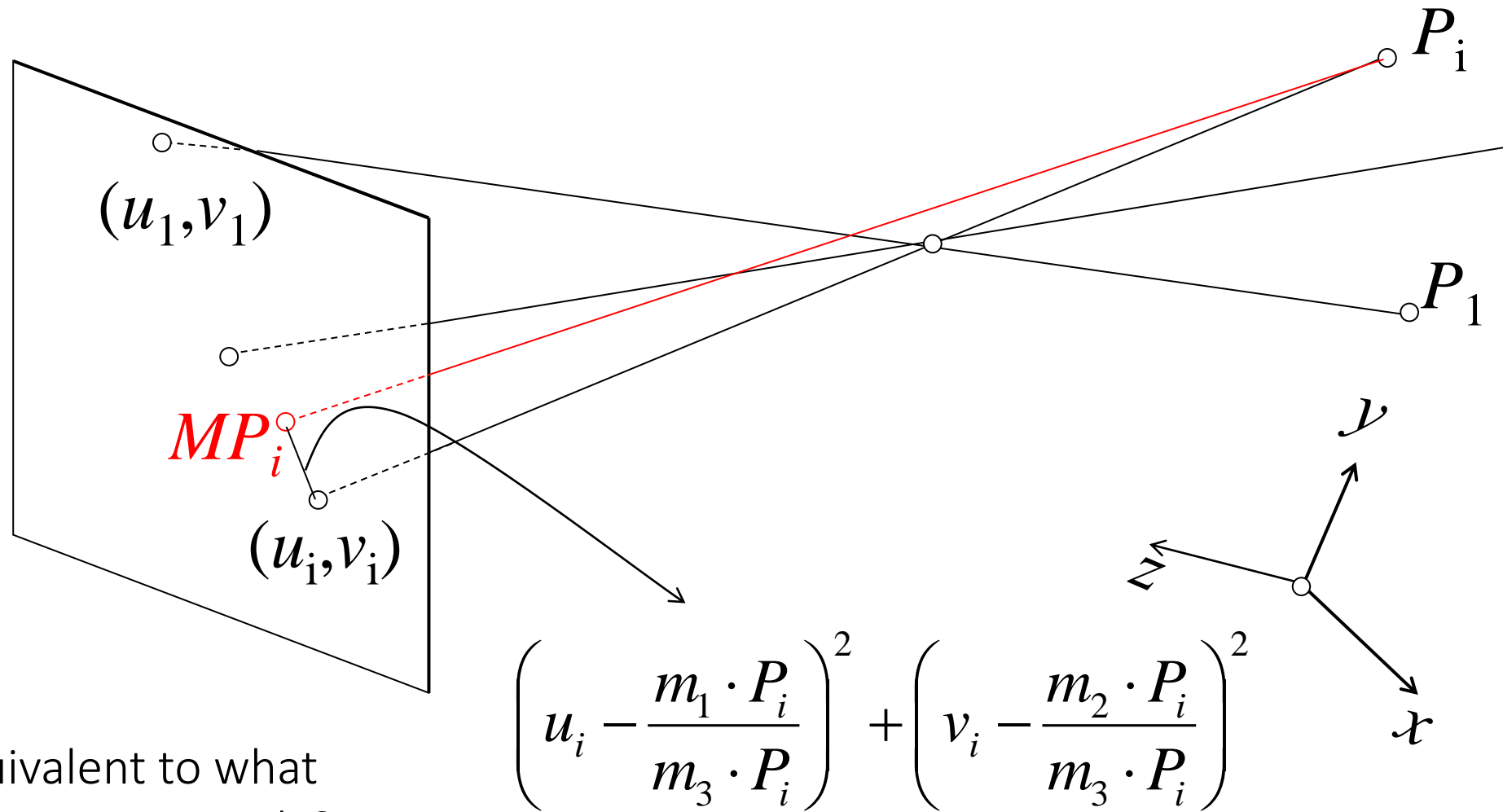
Disadvantages:

- Doesn't model radial distortion.
- Hard to impose constraints (e.g., known f).
- Doesn't minimize the correct error function.

For these reasons, *nonlinear methods* are preferred

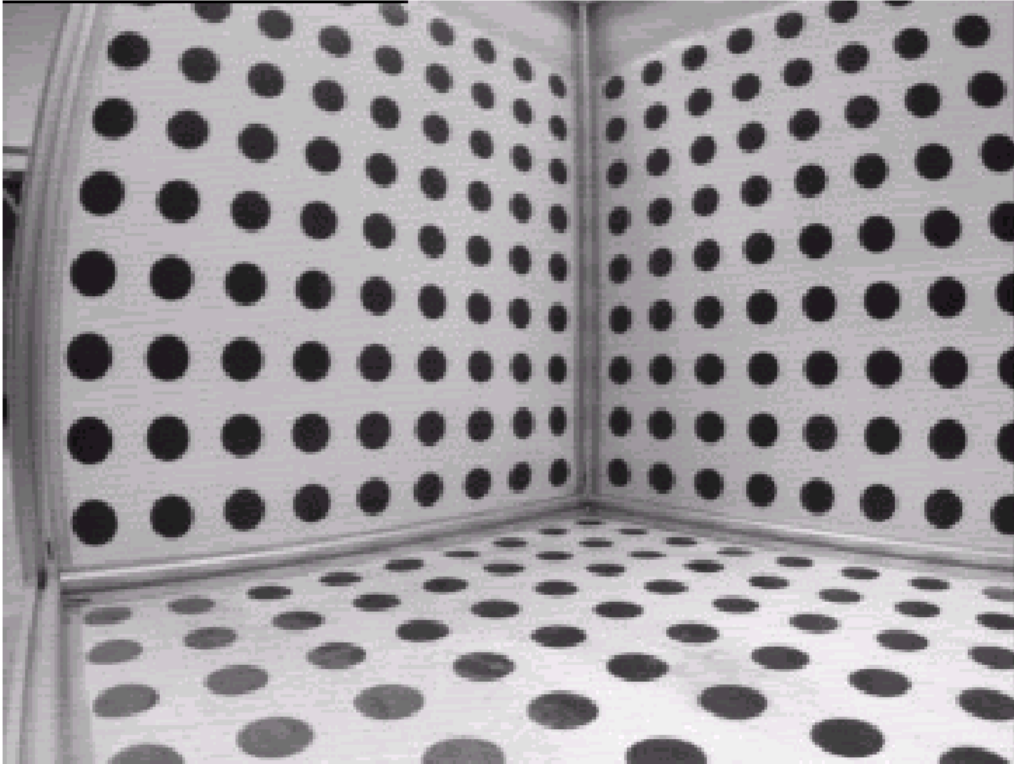
- Define error function E between projected 3D points and image positions
 - E is nonlinear function of intrinsics, extrinsics, radial distortion
- Minimize E using nonlinear optimization techniques

Minimizing reprojection error

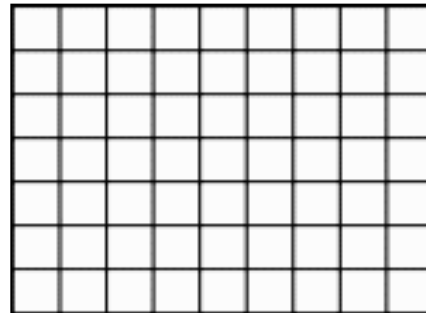


Is this equivalent to what we were doing previously?

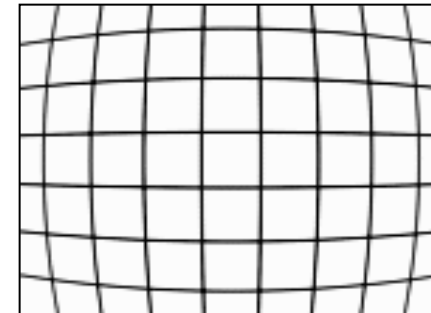
Radial distortion



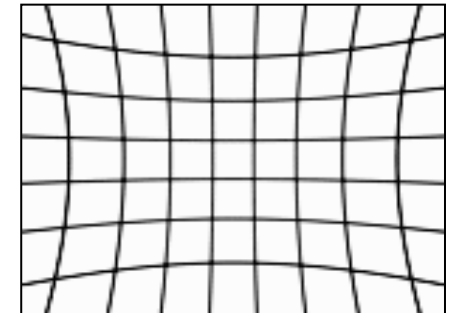
What causes this distortion?



no distortion

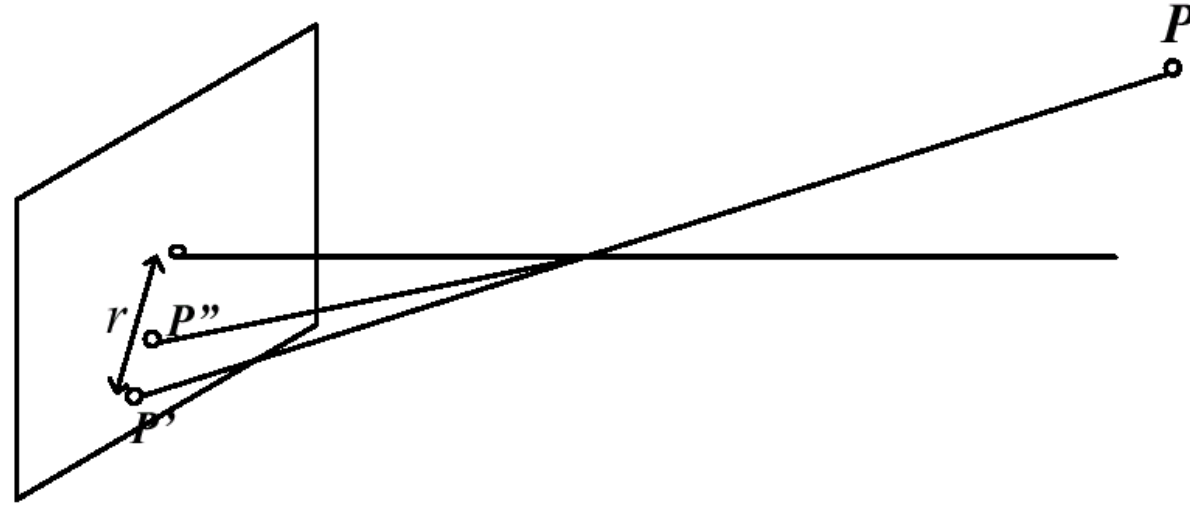


barrel
distortion



pincushion
distortion

Radial distortion model



Ideal:

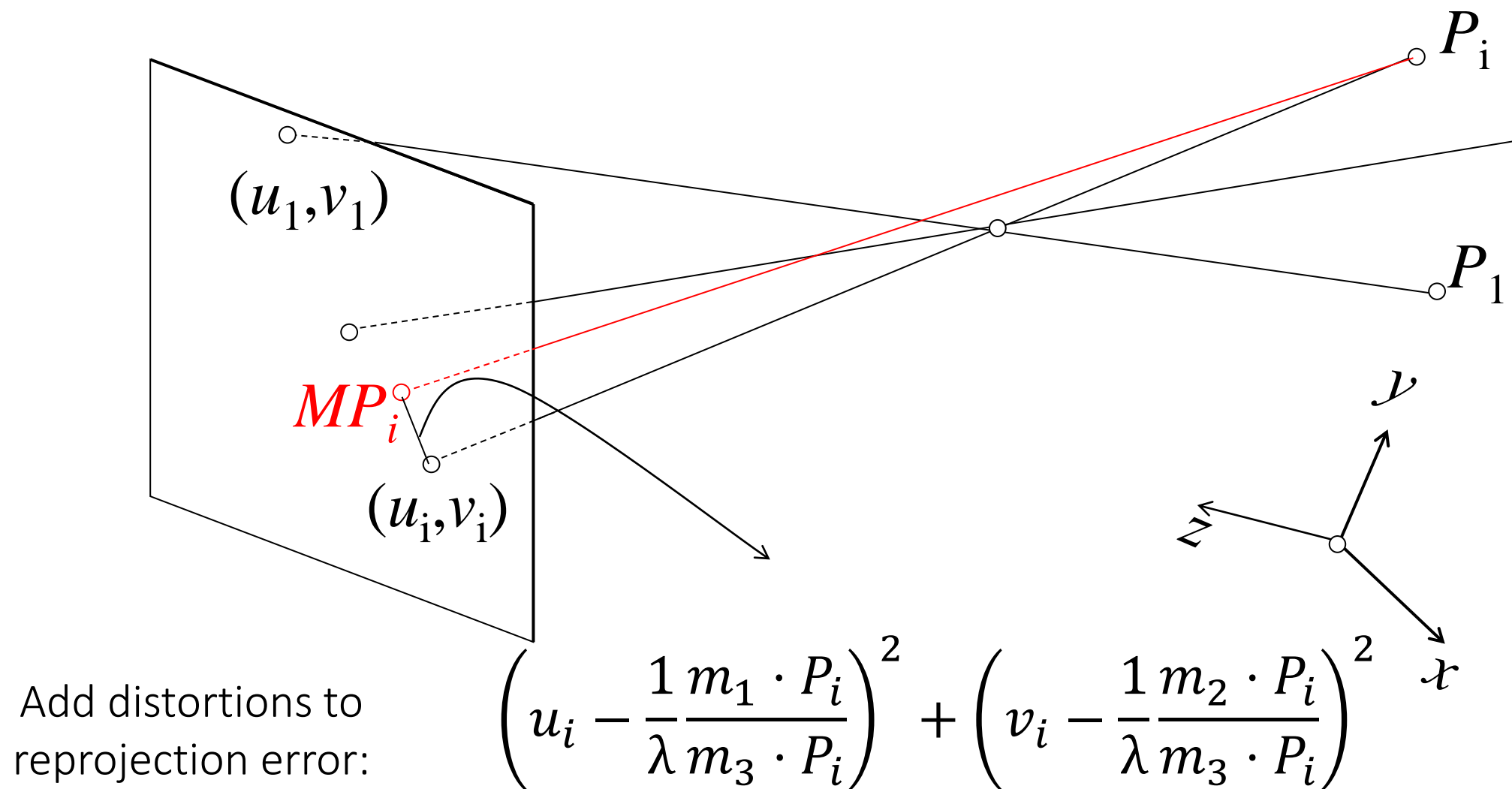
$$x' = f \frac{x}{z}$$
$$y' = f \frac{y}{z}$$

Distorted:

$$x'' = \frac{1}{\lambda} x'$$
$$y'' = \frac{1}{\lambda} y'$$

$$\lambda = 1 + k_1 r^2 + k_2 r^4 + \dots$$

Minimizing reprojection error with radial distortion



Correcting radial distortion

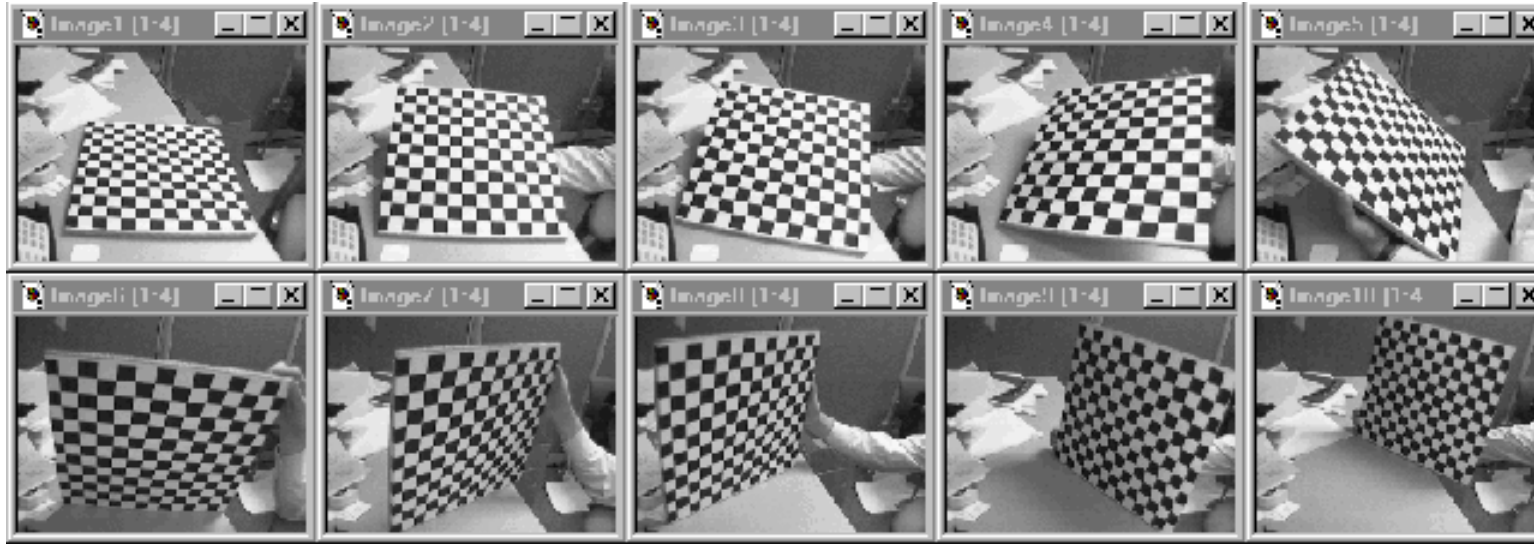


before



after

Alternative: Multi-plane calibration

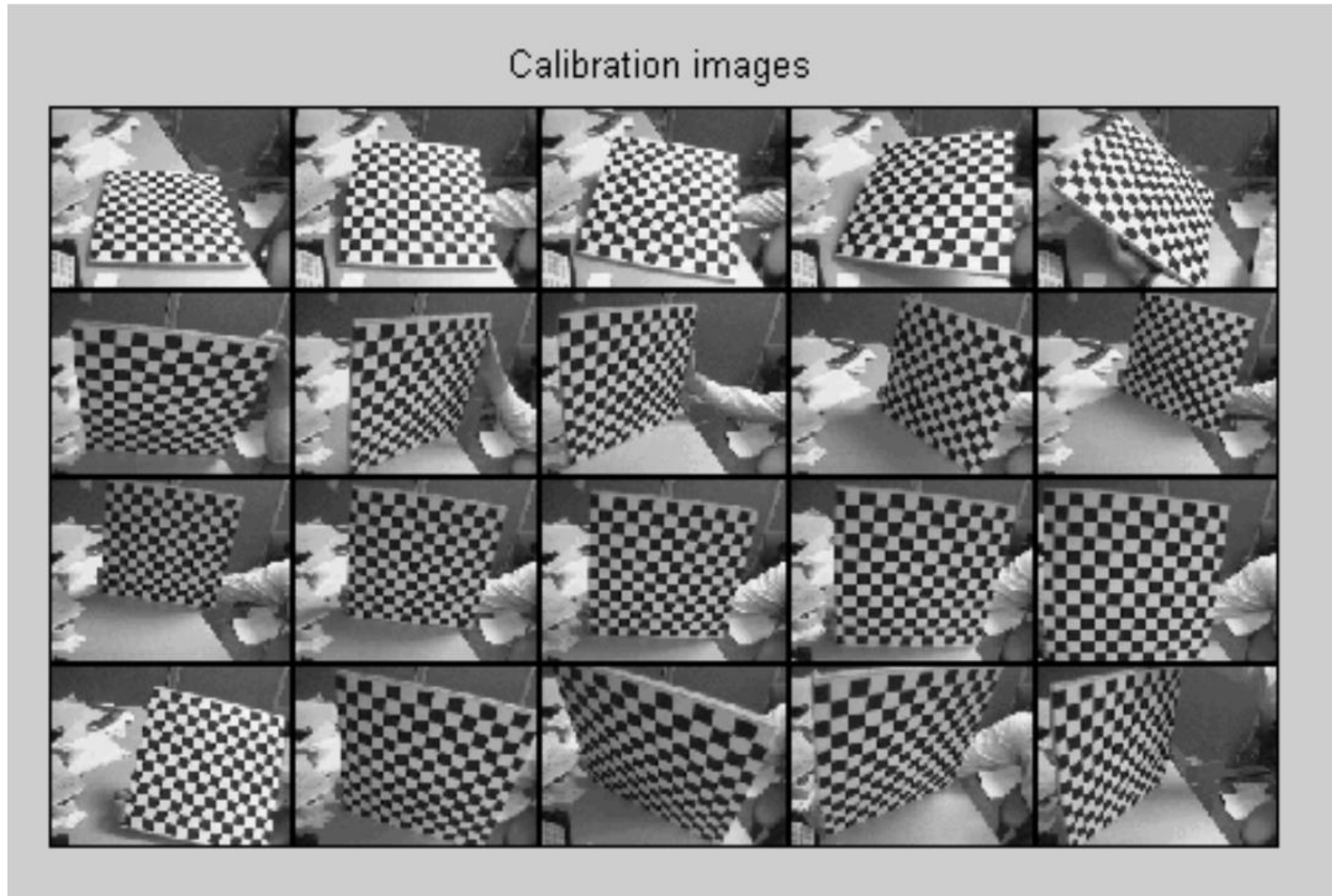


Advantages:

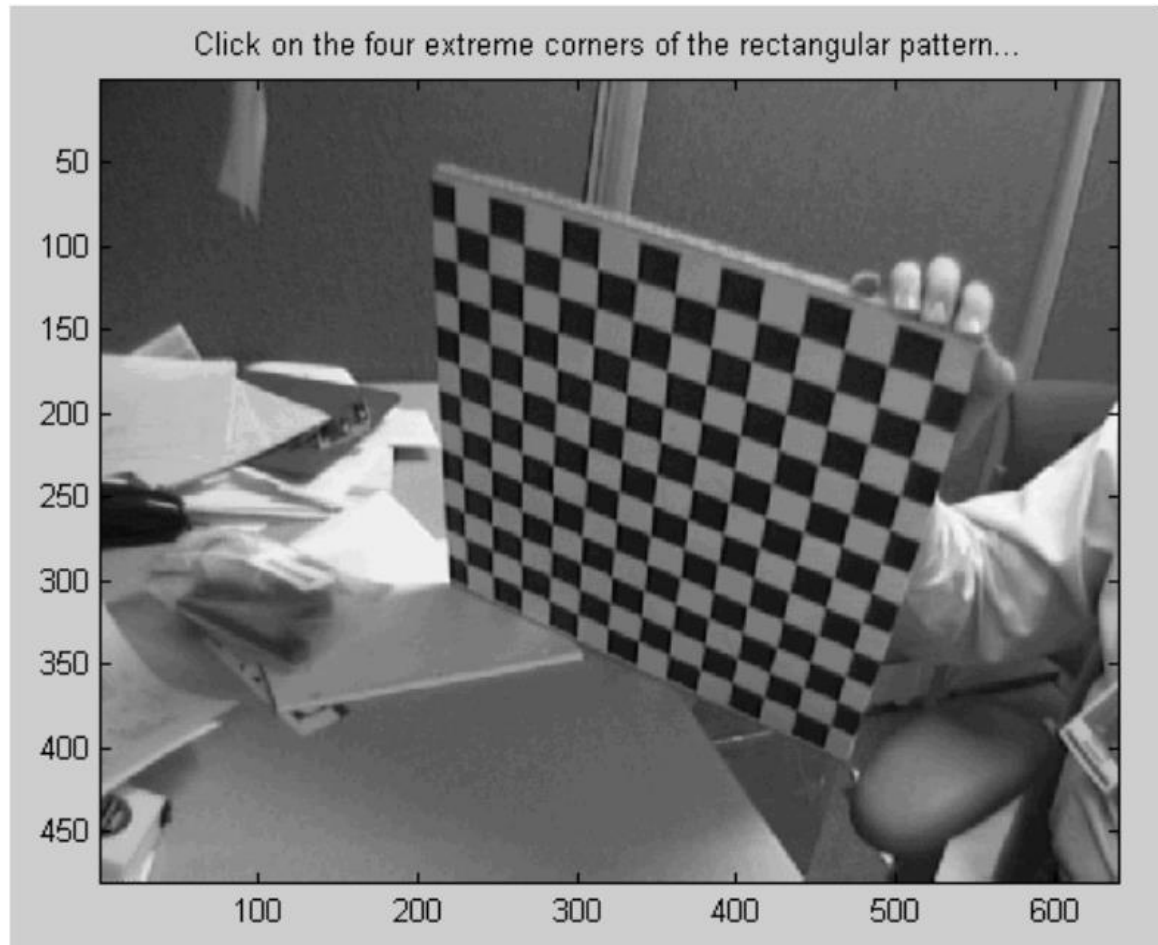
- Only requires a plane
- Don't have to know positions/orientations
- Great code available online!
 - Matlab version: http://www.vision.caltech.edu/bouguetj/calib_doc/index.html
 - Also available on OpenCV.

Disadvantage: Need to solve non-linear optimization problem.

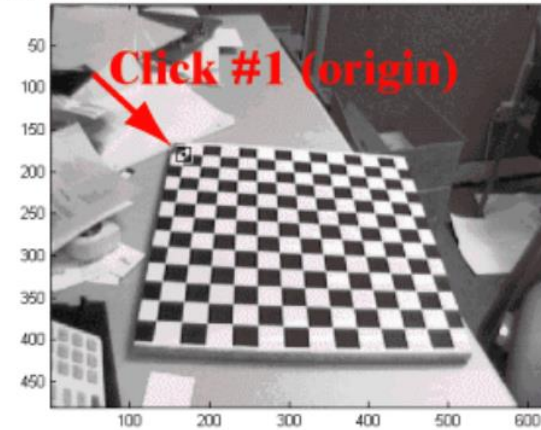
Step-by-step demonstration



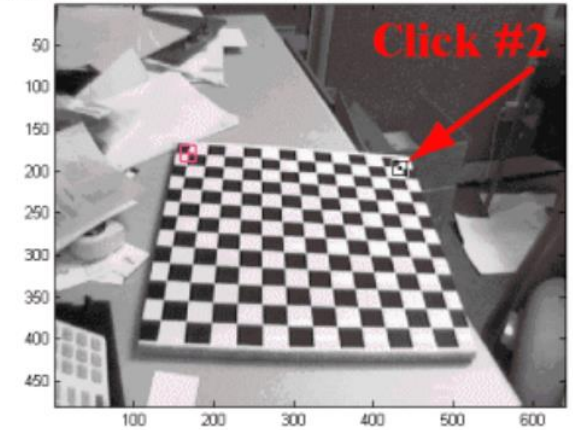
Step-by-step demonstration



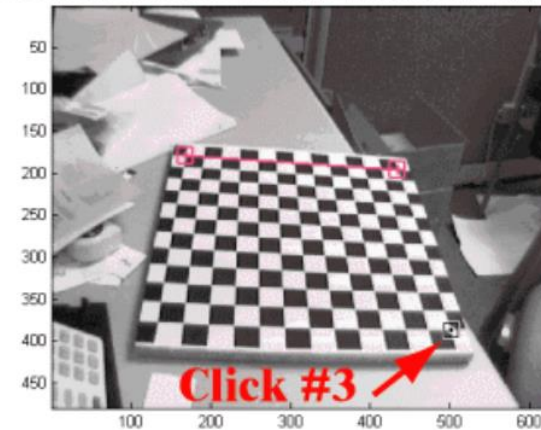
Click on the four extreme corners of the rectangular pattern (first corner = origin)... Image 1



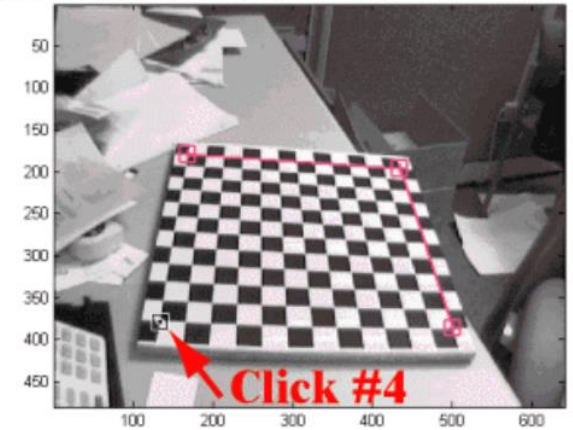
Click on the four extreme corners of the rectangular pattern (first corner = origin)... Image 1



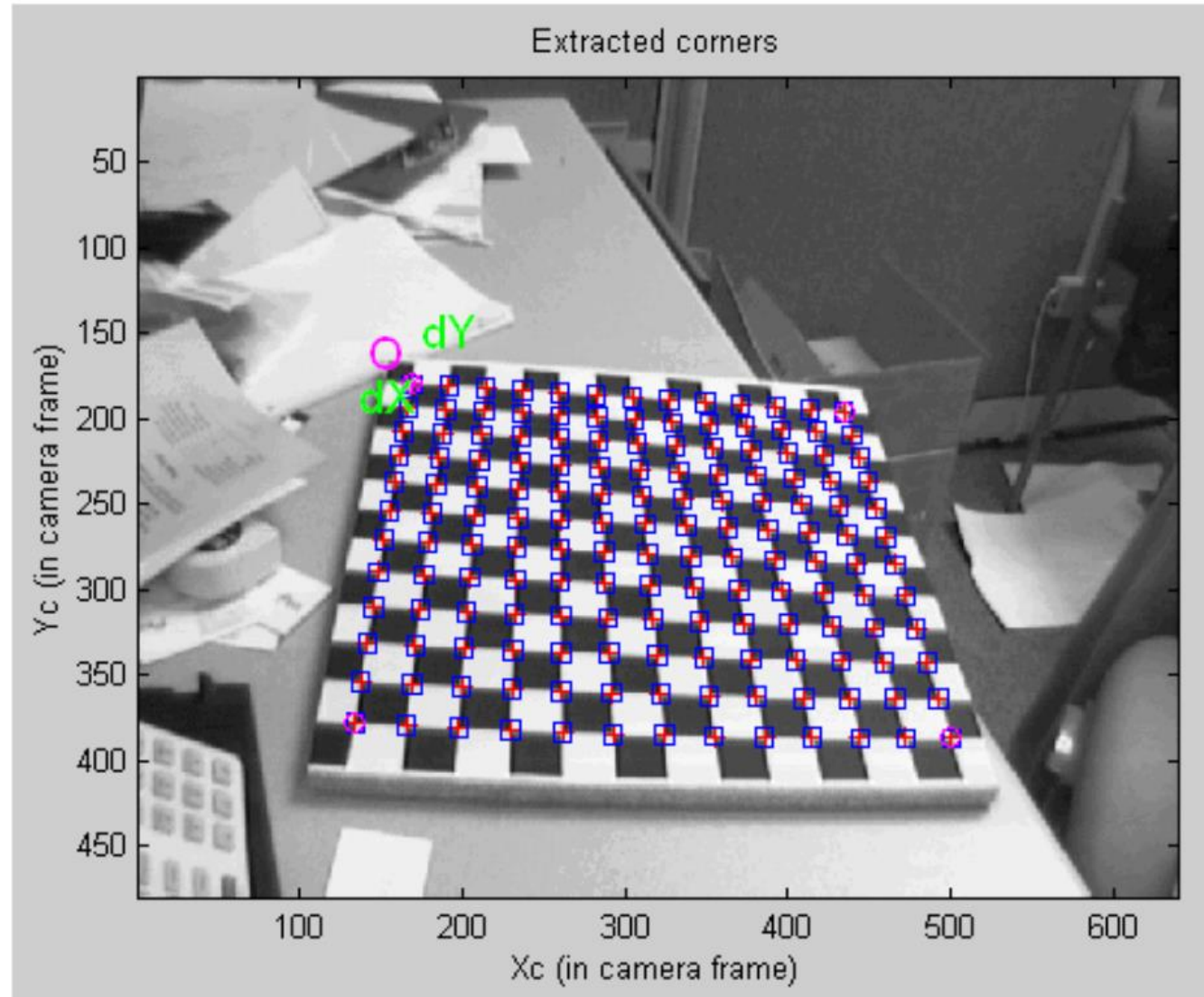
Click on the four extreme corners of the rectangular pattern (first corner = origin)... Image 1



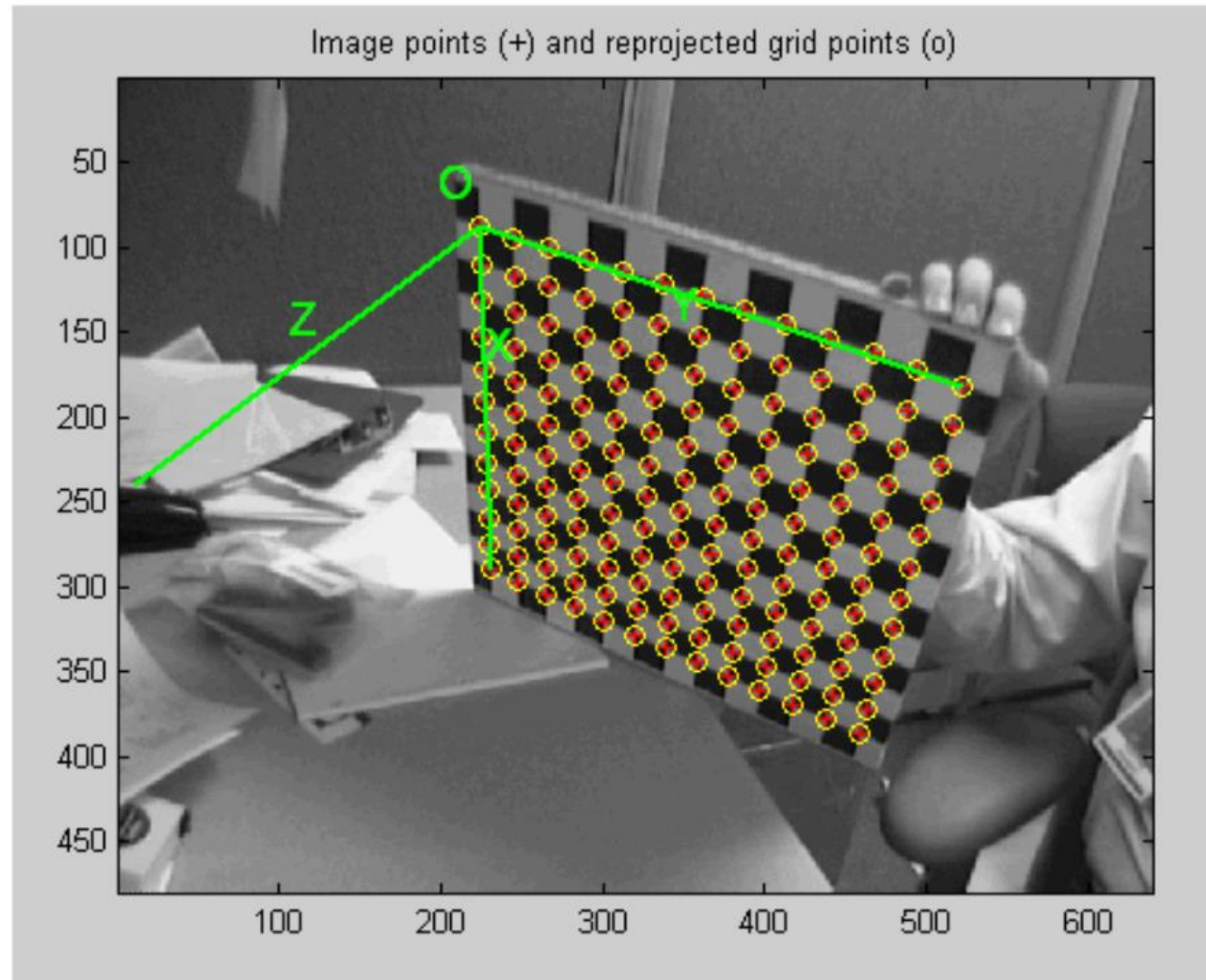
Click on the four extreme corners of the rectangular pattern (first corner = origin)... Image 1



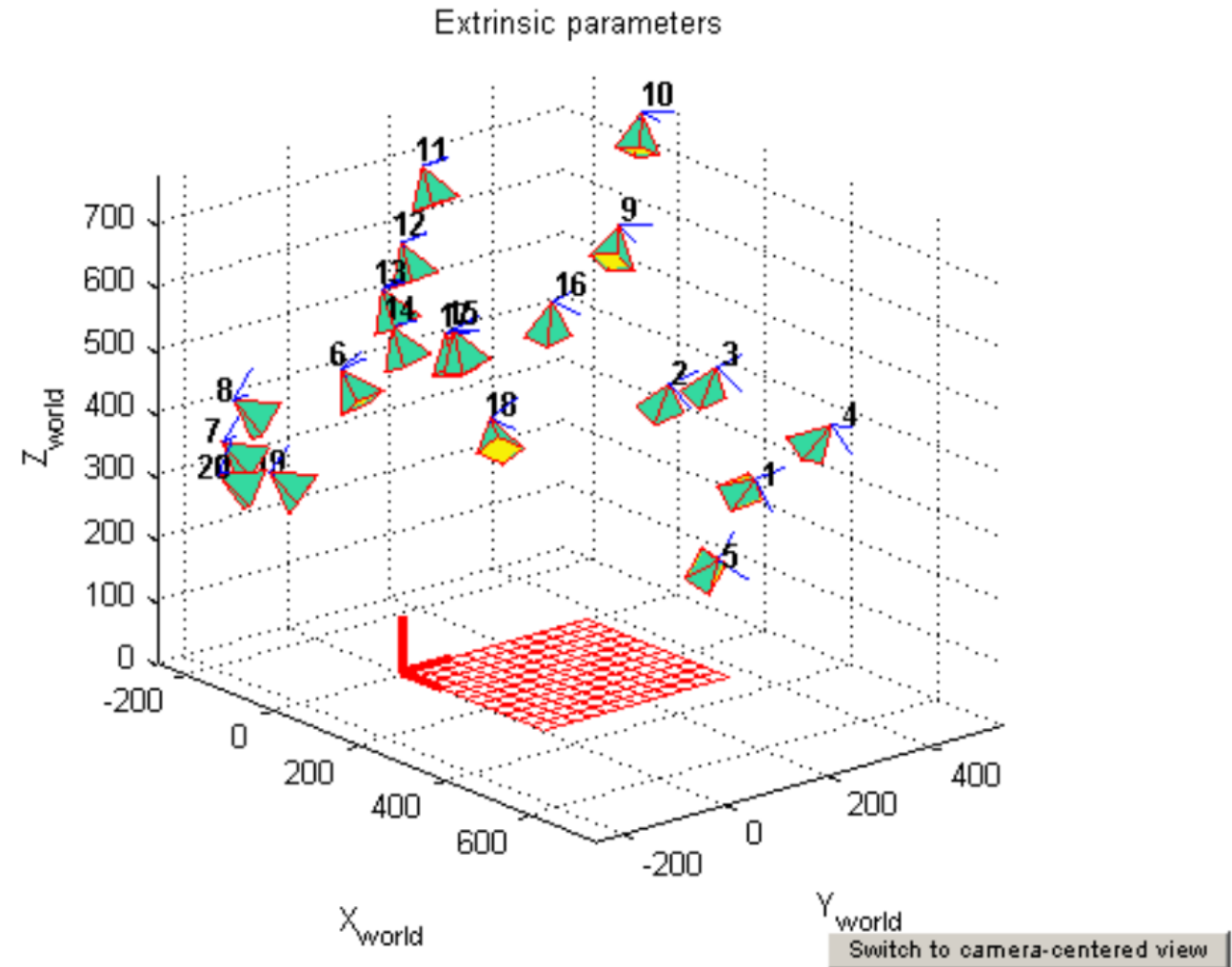
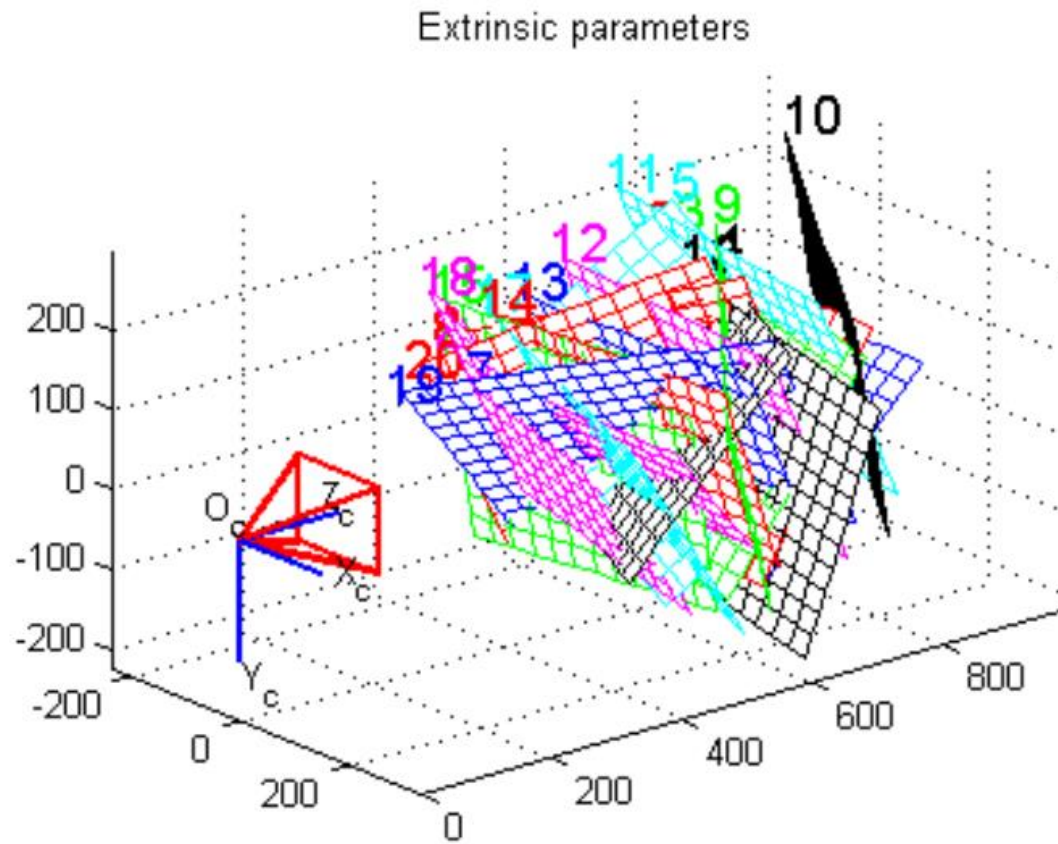
Step-by-step demonstration



Step-by-step demonstration



Step-by-step demonstration



What does it mean to “calibrate a camera”?

Many different ways to calibrate a camera:

- Radiometric calibration.
- Color calibration.
- Geometric calibration.
- Noise calibration.
- Lens (or aberration) calibration.

Pinhole camera model calibration

```
import cv2
import numpy as np
import os

pattern_size = (10, 7)
samples = []
file_list = os.listdir('../data/pinhole_calib')
img_file_list = [file for file in file_list if file.startswith('img')]

for filename in img_file_list:

    frame = cv2.imread(os.path.join('../data/pinhole_calib', filename))
    res, corners = cv2.findChessboardCorners(frame, pattern_size)

    img_show = np.copy(frame)
    cv2.drawChessboardCorners(img_show, pattern_size, corners, res)
    cv2.putText(img_show, 'Samples captured: %d' % len(samples), (0, 40),
                cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0, 255, 0), 2)
    cv2.imshow('chessboard', img_show)

    wait_time = 0 if res else 30
    k = cv2.waitKey(wait_time)

    if k == ord('s') and res:
        samples.append((cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY), corners))
    elif k == 27:
        break

cv2.destroyAllWindows()

criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 1e-3)

for i in range(len(samples)):
    img, corners = samples[i]
    corners = cv2.cornerSubPix(img, corners, (10, 10), (-1,-1), criteria)

pattern_points = np.zeros((np.prod(pattern_size), 3), np.float32)
pattern_points[:, :2] = np.indices(pattern_size).T.reshape(-1, 2)

images, corners = zip(*samples)

pattern_points = [pattern_points]*len(corners)

rms, camera_matrix, dist_coefs, rvecs, tvecs = cv2.calibrateCamera(
    pattern_points, corners, images[0].shape, None, None)

np.save('camera_mat.npy', camera_matrix)
np.save('dist_coefs.npy', dist_coefs)

print(np.load('camera_mat.npy'))
print(np.load('dist_coefs.npy'))
```



Fisheye camera model calibration

```
import cv2
import numpy as np
import os

pattern_size = (8, 6)
samples = []

file_list = os.listdir('../data/fisheyes')
img_file_list = [file for file in file_list if file.startswith('Fisheye1_')]

for filename in img_file_list:

    frame = cv2.imread(os.path.join('../data/fisheyes', filename))
    res, corners = cv2.findChessboardCorners(frame, pattern_size)

    res, corners = cv2.findChessboardCorners(frame, pattern_size)

    img_show = np.copy(frame)
    cv2.drawChessboardCorners(img_show, pattern_size, corners, res)
    cv2.putText(img_show, 'Samples captured: %d' % len(samples), (0, 40),
                cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0, 255, 0), 2)
    cv2.imshow('chessboard', img_show)

    wait_time = 0 if res else 30
    k = cv2.waitKey(wait_time)

    if k == ord('s') and res:
        samples.append((cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY), corners))
    elif k == 27:
        break

cv2.destroyAllWindows()
```

```
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 1e-3)

for i in range(len(samples)):
    img, corners = samples[i]
    corners = cv2.cornerSubPix(img, corners, (10, 10), (-1, -1), criteria)

pattern_points = np.zeros((1, np.prod(pattern_size), 3), np.float32)
pattern_points[0, :, :2] = np.indices(pattern_size).T.reshape(-1, 2)

images, corners = zip(*samples)

pattern_points = [pattern_points]*len(corners)

print(len(pattern_points), pattern_points[0].shape, pattern_points[0].dtype)
print(len(corners), corners[0].shape, corners[0].dtype)

rms, camera_matrix, dist_coefs, rvecs, tvecs = cv2.fisheye.calibrate(
    pattern_points, corners, images[0].shape, None, None)

np.save('camera_mat.npy', camera_matrix)
np.save('dist_coefs.npy', dist_coefs)

print(np.load('camera_mat.npy'))
print(np.load('dist_coefs.npy'))
```



Stereo rig calibration

```
import cv2
import glob
import numpy as np

PATTERN_SIZE = (9, 6)
left_imgs = list(sorted(glob.glob('../data/stereo/case1/left*.png')))
right_imgs = list(sorted(glob.glob('../data/stereo/case1/right*.png')))
assert len(left_imgs) == len(right_imgs)

criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 1e-3)
left_pts, right_pts = [], []
img_size = None

for left_img_path, right_img_path in zip(left_imgs, right_imgs):
    left_img = cv2.imread(left_img_path, cv2.IMREAD_GRAYSCALE)
    right_img = cv2.imread(right_img_path, cv2.IMREAD_GRAYSCALE)
    if img_size is None:
        img_size = (left_img.shape[1], left_img.shape[0])

    res_left, corners_left = cv2.findChessboardCorners(left_img, PATTERN_SIZE)
    res_right, corners_right = cv2.findChessboardCorners(right_img, PATTERN_SIZE)

    corners_left = cv2.cornerSubPix(left_img, corners_left, (10, 10), (-1, -1),
                                    criteria)
    corners_right = cv2.cornerSubPix(right_img, corners_right, (10, 10), (-1, -1),
                                    criteria)

    left_pts.append(corners_left)
    right_pts.append(corners_right)
```

```
pattern_points = np.zeros((np.prod(PATTERN_SIZE), 3), np.float32)
pattern_points[:, :2] = np.indices(PATTERN_SIZE).T.reshape(-1, 2)
pattern_points = [pattern_points] * len(left_imgs)

err, K1, D1, Kr, Dr, R, T, E, F = cv2.stereoCalibrate(
    pattern_points, left_pts, right_pts, None, None, None, None, img_size, flags=0)

print('Left camera:')
print(K1)
print('Left camera distortion:')
print(D1)
print('Right camera:')
print(Kr)
print('Right camera distortion:')
print(Dr)
print('Rotation matrix:')
print(R)
print('Translation:')
print(T)

np.save('stereo.npy', {'K1': K1, 'D1': D1, 'Kr': Kr, 'Dr': Dr, 'R': R, 'T': T, 'E': E, 'F': F,
                       'img_size': img_size, 'left_pts': left_pts, 'right_pts': right_pts})
```



Distorting and undistorting points

```
import cv2
import numpy as np

camera_matrix = np.load('../data/pinhole_calib/camera_mat.npy')
dist_coefs = np.load('../data/pinhole_calib/dist_coefs.npy')

img = cv2.imread('../data/pinhole_calib/img_00.png')
pattern_size = (10, 7)
res, corners = cv2.findChessboardCorners(img, pattern_size)
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 1e-3)
corners = cv2.cornerSubPix(cv2.cvtColor(img, cv2.COLOR_BGR2GRAY),
                           corners, (10, 10), (-1,-1), criteria)

h_corners = cv2.undistortPoints(corners, camera_matrix, dist_coefs)
h_corners = np.c_[h_corners.squeeze(), np.ones(len(h_corners))]

img_pts, _ = cv2.projectPoints(h_corners, (0, 0, 0), (0, 0, 0), camera_matrix, None)
```

```
for c in corners:
    cv2.circle(img, tuple(c[0]), 10, (0, 255, 0), 2)

for c in img_pts.squeeze().astype(np.float32):
    cv2.circle(img, tuple(c), 5, (0, 0, 255), 2)

cv2.imshow('undistorted corners', img)
cv2.waitKey()
cv2.destroyAllWindows()

img_pts, _ = cv2.projectPoints(h_corners, (0, 0, 0), (0, 0, 0), camera_matrix, dist_coefs)

for c in img_pts.squeeze().astype(np.float32):
    cv2.circle(img, tuple(c), 2, (255, 255, 0), 2)

cv2.imshow('reprojected corners', img)
cv2.waitKey()
cv2.destroyAllWindows()
```


Removing lens distortion effects

```
import cv2
import numpy as np

camera_matrix = np.load('../data/pinhole_calib/camera_mat.npy')
dist_coefs = np.load('../data/pinhole_calib/dist_coefs.npy')
img = cv2.imread('../data/pinhole_calib/img_00.png')

cv2.imshow('original image', img)

ud_img = cv2.undistort(img, camera_matrix, dist_coefs)
cv2.imshow('undistorted image1', ud_img)

opt_cam_mat, valid_roi = cv2.getOptimalNewCameraMatrix(camera_matrix, dist_coefs, img.shape[:2][::-1], 0)
ud_img = cv2.undistort(img, camera_matrix, dist_coefs, None, opt_cam_mat)
cv2.imshow('undistorted image2', ud_img)

cv2.waitKey(0)
cv2.destroyAllWindows()
```