Industrial Computer Vision
- Epipolar Geometry

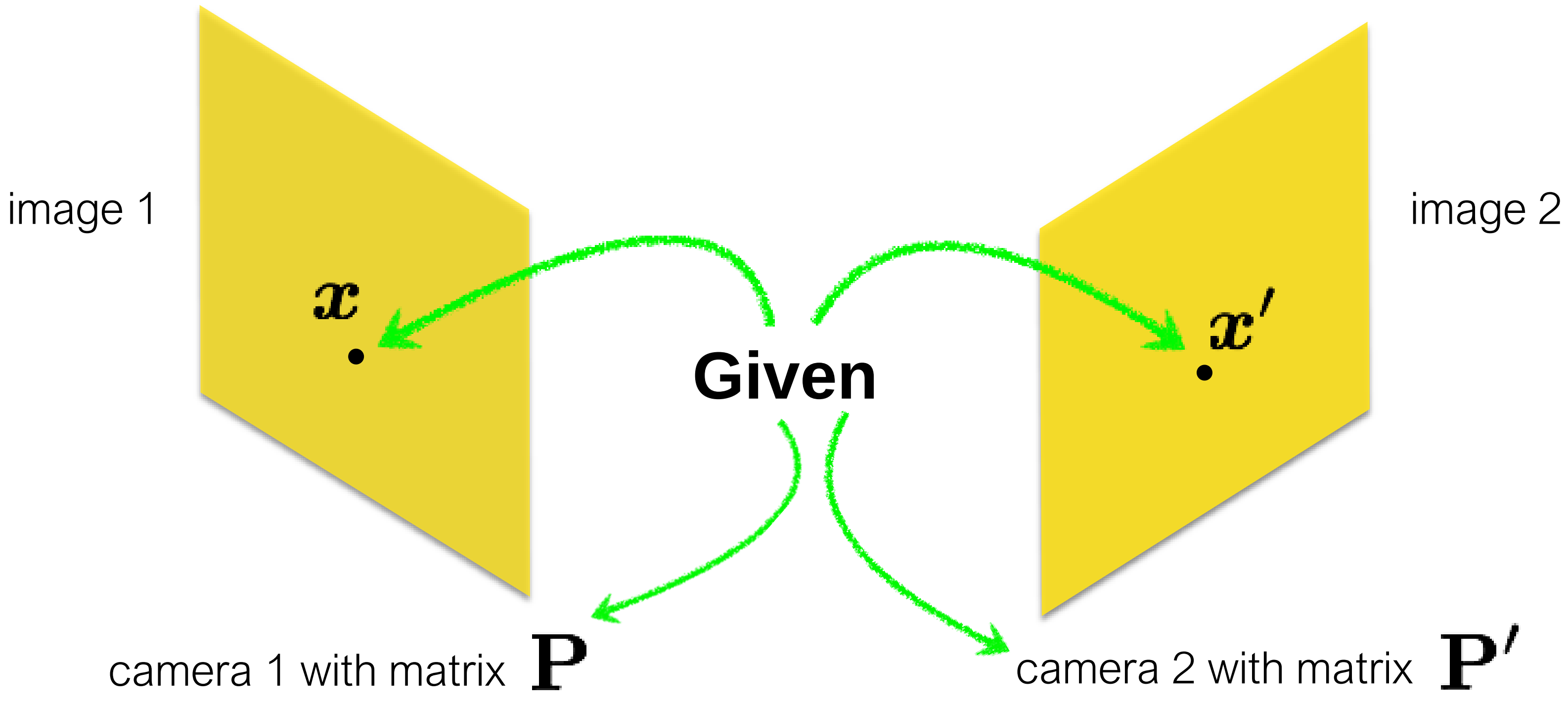
12th lecture, 2022.11.30

Lecturer: Youngbae Hwang

Contents

- Triangulation
- Epipolar Geometry
- Essential/Fundamental Matrix
- Stereo Rectification

Triangulation

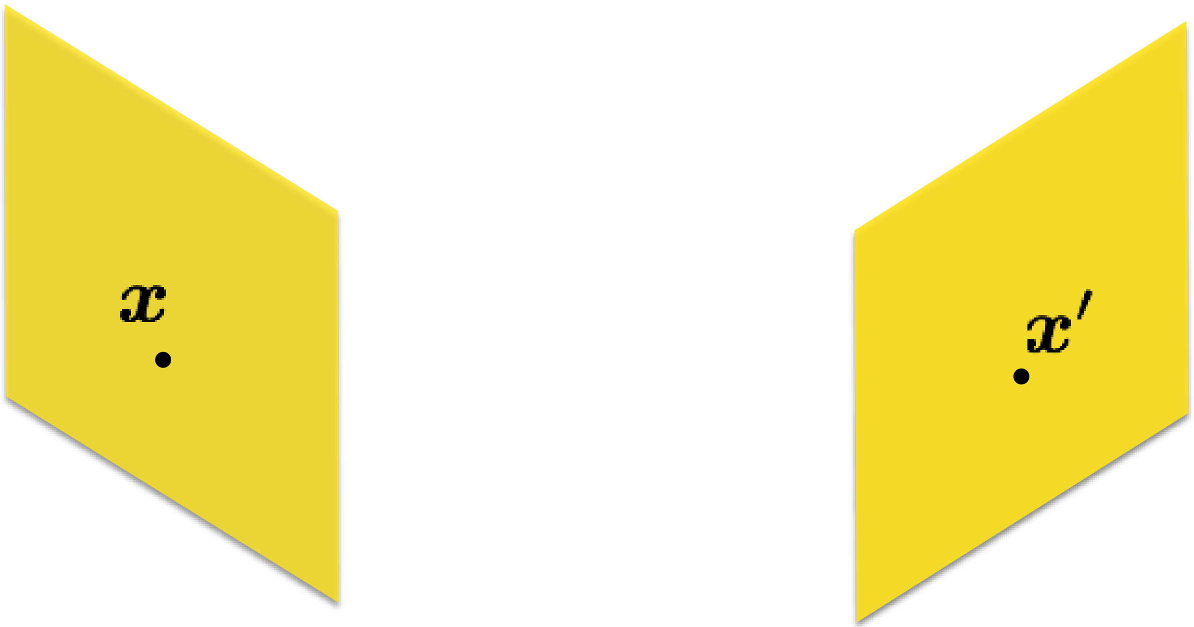


Triangulation

Which 3D points map to **x**?

image 1

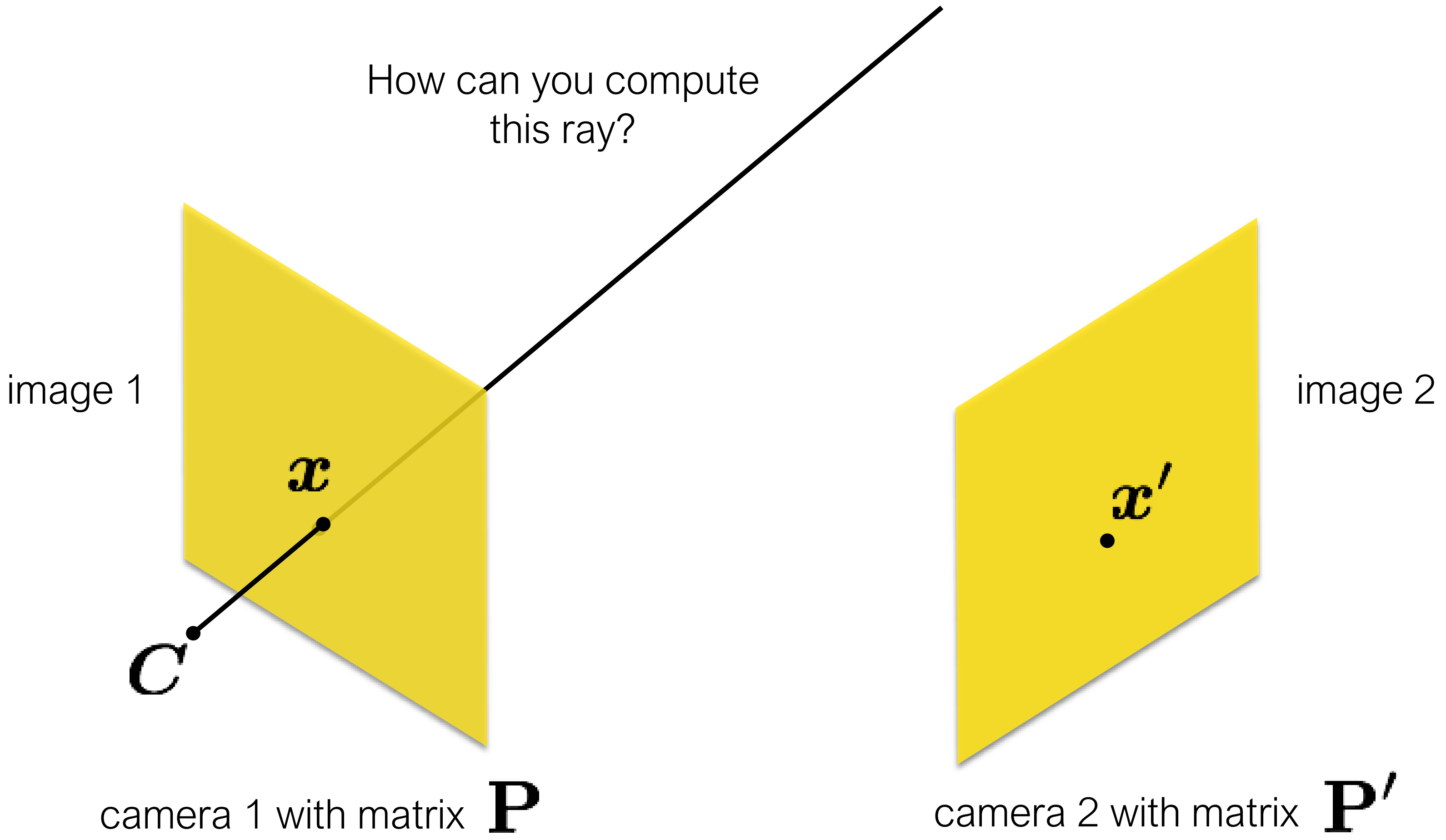
image 2



camera 1 with matrix $\mathbf{P}$

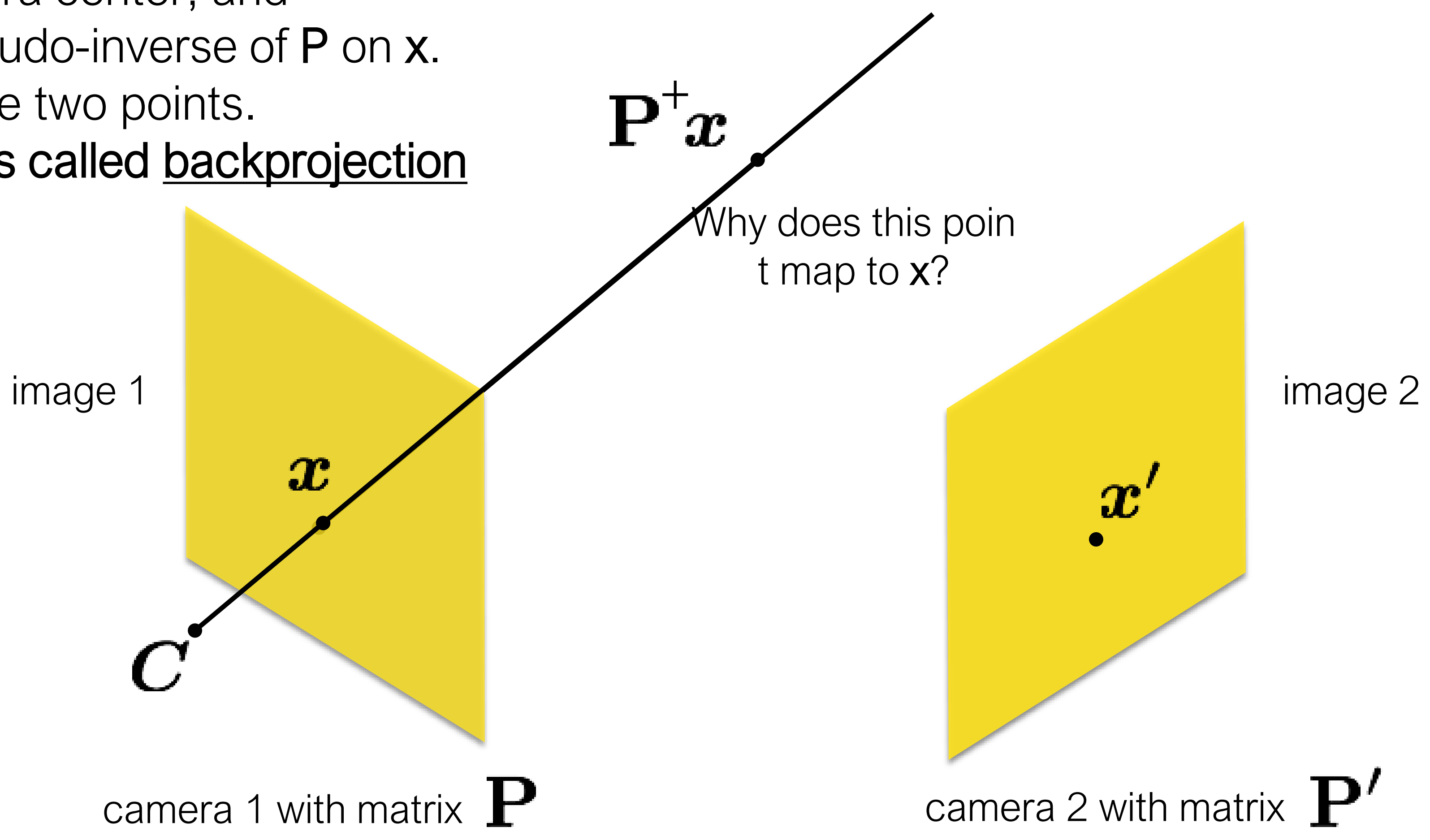
camera 2 with matrix $\mathbf{P}'$

Triangulation

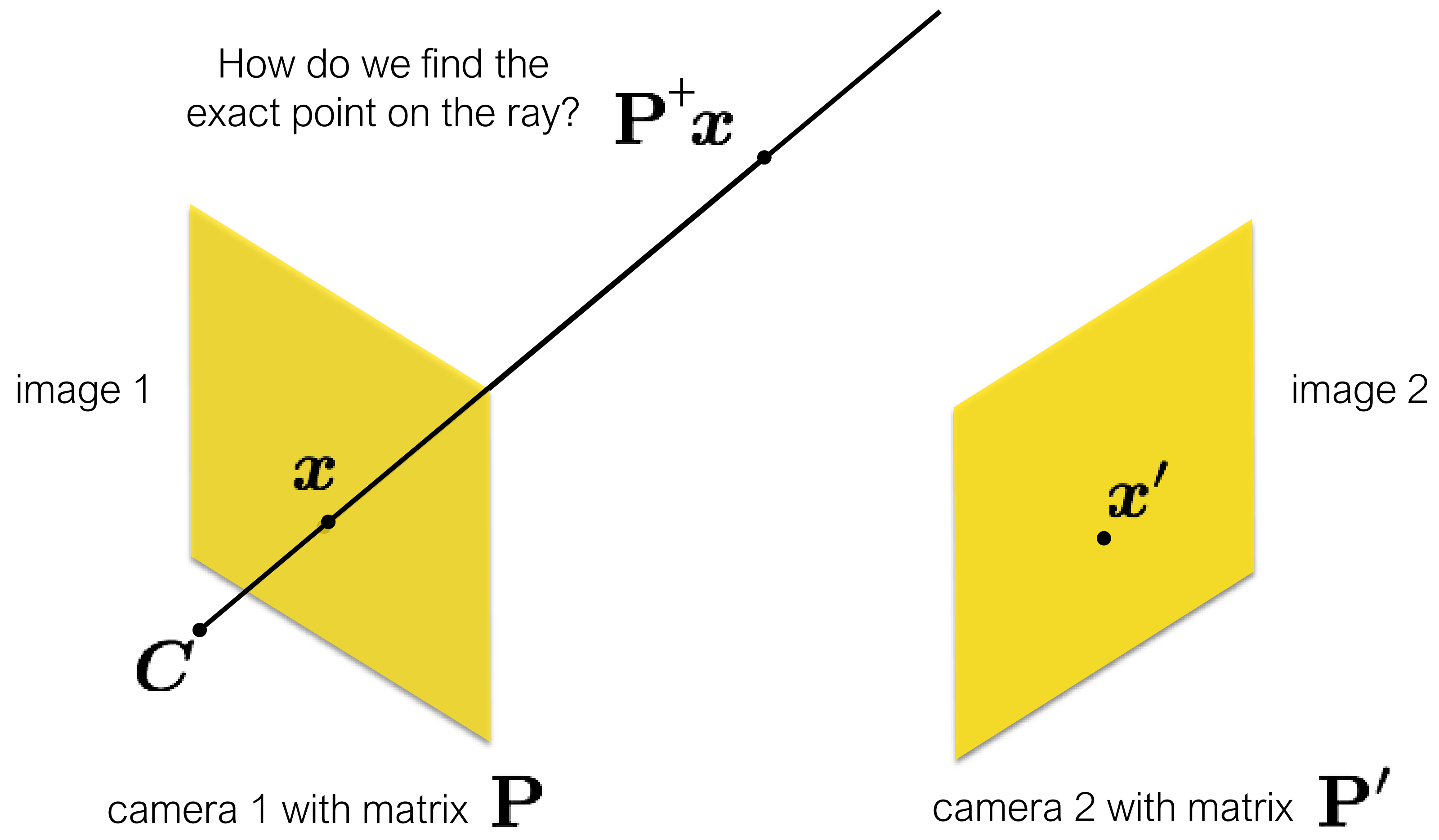


Triangulation

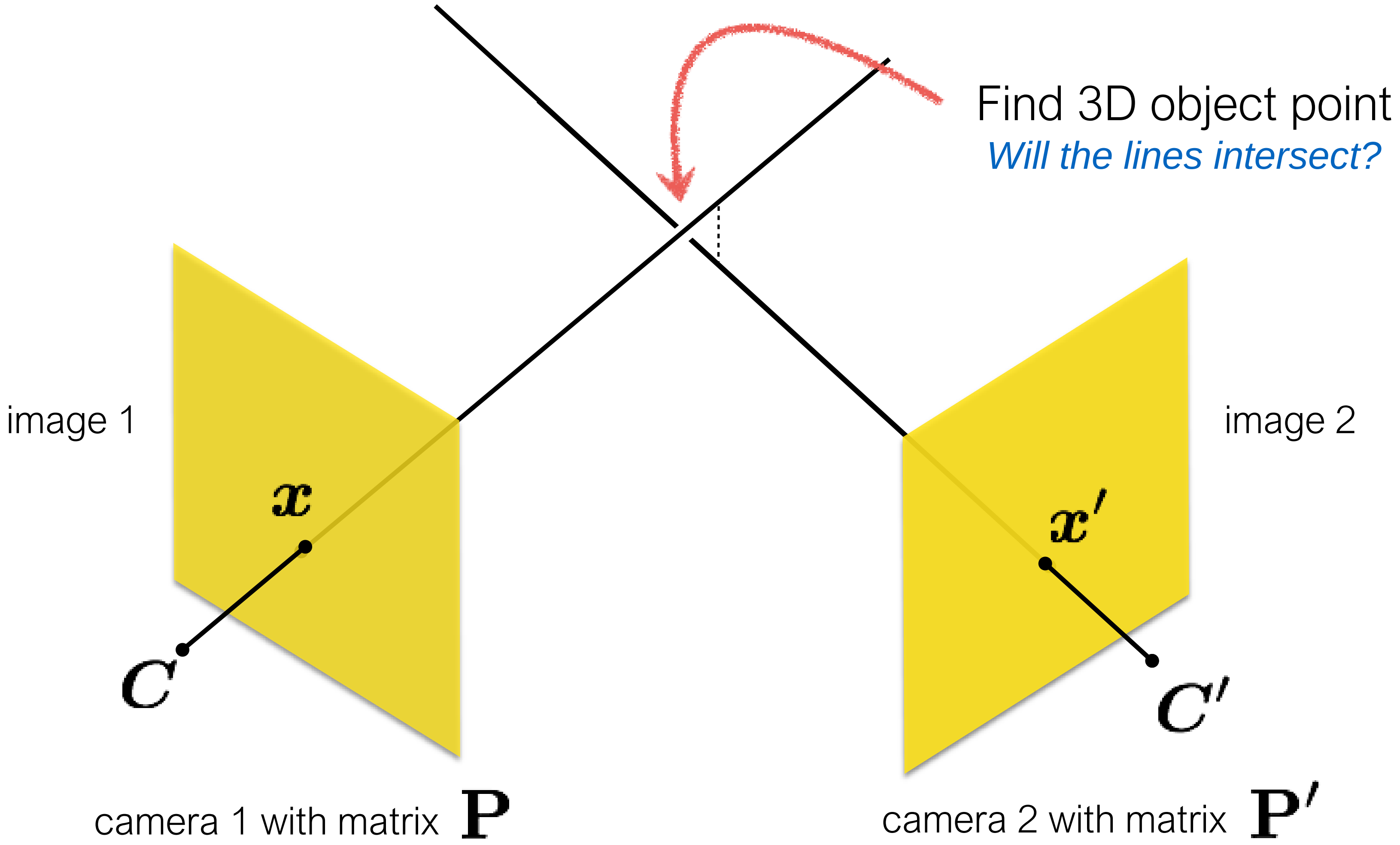
Create two points on the ray:
1) find the camera center; and
2) apply the pseudo-inverse of P on **x**.
Then connect the two points.
This procedure is called <u>backprojection</u>



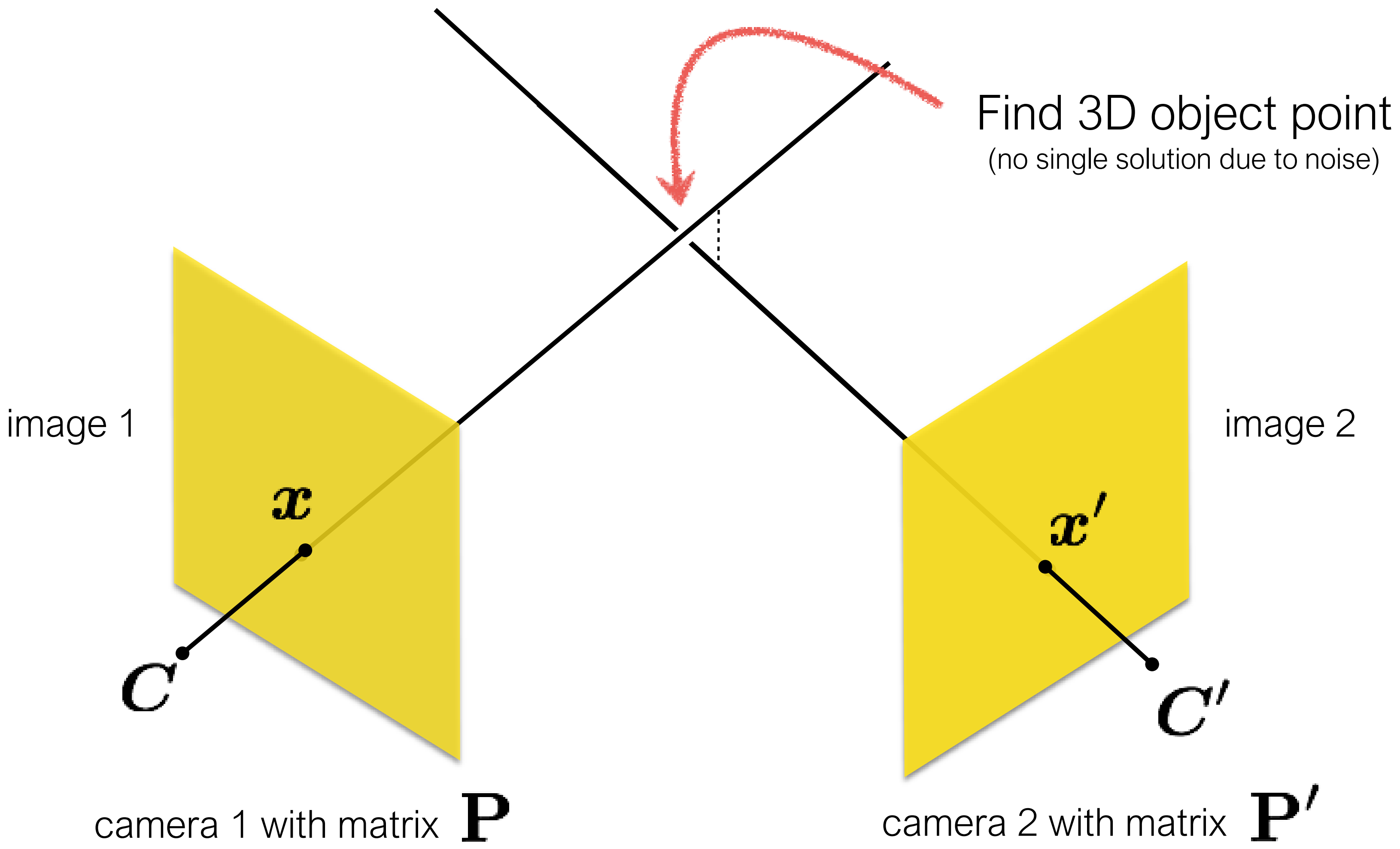
Triangulation



Triangulation



Triangulation



Triangulation

Given a set of (noisy) matched points

$$\{\boldsymbol{x}_i, \boldsymbol{x}'_i\}$$

and camera matrices

$$\mathbf{P}, \mathbf{P}'$$

Estimate the 3D point

$$\mathbf{X}$$

Triangulation

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

known known

*Can we compute **X** from a single correspondence **x**?*

Triangulation

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

(homogeneous co
ordinate)

This is a similarity relation because it involves homogeneous coordinates

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

(homorogeneous
coordinate)

Same ray direction but differs by a scale factor

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

How do we solve for unknowns in a similarity relation?

Triangulation

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

(homogeneous co
ordinate)

Also, this is a similarity relation because it involves homogeneous coordinates

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

(inhomogeneous
coordinate)

Same ray direction but differs by a scale factor

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

How do we solve for unknowns in a similarity relation?

Triangulation

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

Same direction but differs by a scale factor

$$\mathbf{x} \times \mathbf{P} \boldsymbol{X} = \mathbf{0}$$

Cross product of two vectors of same direction is zero
(this equality removes the scale factor)

Triangulation

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

Do the same after first expanding out the camera matrix and points

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

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \alpha \begin{bmatrix} \boldsymbol{p}_1^\top \boldsymbol{X} \\ \boldsymbol{p}_2^\top \boldsymbol{X} \\ \boldsymbol{p}_3^\top \boldsymbol{X} \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \times \begin{bmatrix} \boldsymbol{p}_1^\top \boldsymbol{X} \\ \boldsymbol{p}_2^\top \boldsymbol{X} \\ \boldsymbol{p}_3^\top \boldsymbol{X} \end{bmatrix} = \begin{bmatrix} y\boldsymbol{p}_3^\top \boldsymbol{X} - \boldsymbol{p}_2^\top \boldsymbol{X} \\ \boldsymbol{p}_1^\top \boldsymbol{X} - x\boldsymbol{p}_3^\top \boldsymbol{X} \\ x\boldsymbol{p}_2^\top \boldsymbol{X} - y\boldsymbol{p}_1^\top \boldsymbol{X} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Triangulation

Using the fact that the cross product should be zero

$$\mathbf{x} \times \mathbf{P}\boldsymbol{X} = \mathbf{0}$$

$$\left[\begin{array}{c} y\boldsymbol{p}_3^\top \boldsymbol{X} - \boldsymbol{p}_2^\top \boldsymbol{X} \\ \boldsymbol{p}_1^\top \boldsymbol{X} - x\boldsymbol{p}_3^\top \boldsymbol{X} \\ x\boldsymbol{p}_2^\top \boldsymbol{X} - y\boldsymbol{p}_1^\top \boldsymbol{X} \end{array} \right] = \left[\begin{array}{c} 0 \\ 0 \\ 0 \end{array} \right]$$

Third line is a linear combination of the first and second lines.
(x times the first line plus y times the second line)

One 2D to 3D point correspondence give you equations

Triangulation

Using the fact that the cross product should be zero

$$\mathbf{x} \times \mathbf{P}\boldsymbol{X} = \mathbf{0}$$

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

Third line is a linear combination of the first and second lines.
(x times the first line plus y times the second line)

One 2D to 3D point correspondence give you 2 equations

Triangulation

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

Remove third row, and rear range as system on unkno wns

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

$$\mathbf{A}_i \boldsymbol{X} = \mathbf{0}$$

Now we can make a system of linear equations
(two lines for each 2D point correspondence)

Triangulation

Concatenate the 2D points from both images

Two rows from camera one

Two rows from camera two

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

sanity check! dimensions?

$$\mathbf{A}\boldsymbol{X} = \mathbf{0}$$

How do we solve homogeneous linear system?

Triangulation

Concatenate the 2D points from both images

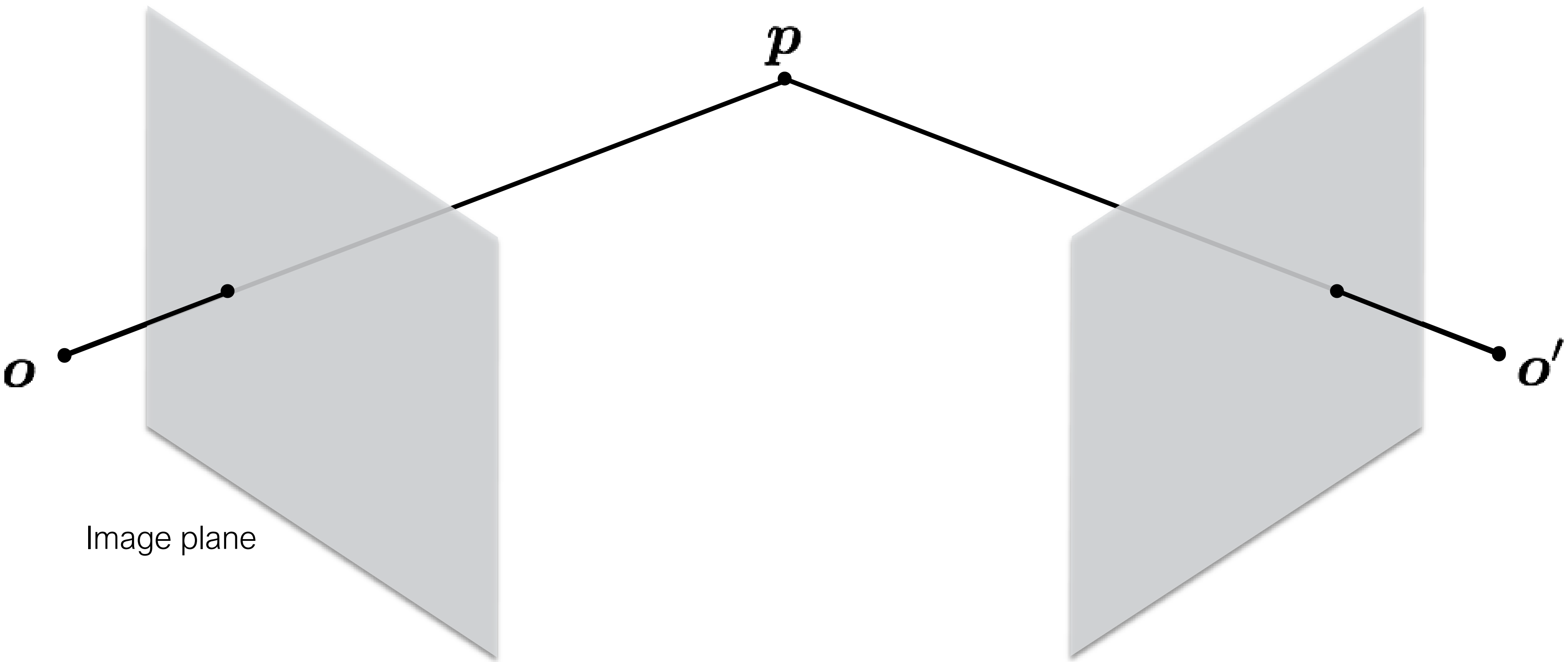
$$\begin{bmatrix} y\boldsymbol{p}_3^\top - \boldsymbol{p}_2^\top \\ \boldsymbol{p}_1^\top - x\boldsymbol{p}_3^\top \\ y'\boldsymbol{p}_3'^\top - \boldsymbol{p}_2'^\top \\ \boldsymbol{p}_1'^\top - x'\boldsymbol{p}_3'^\top \end{bmatrix} \boldsymbol{X} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{A}\boldsymbol{X} = \mathbf{0}$$

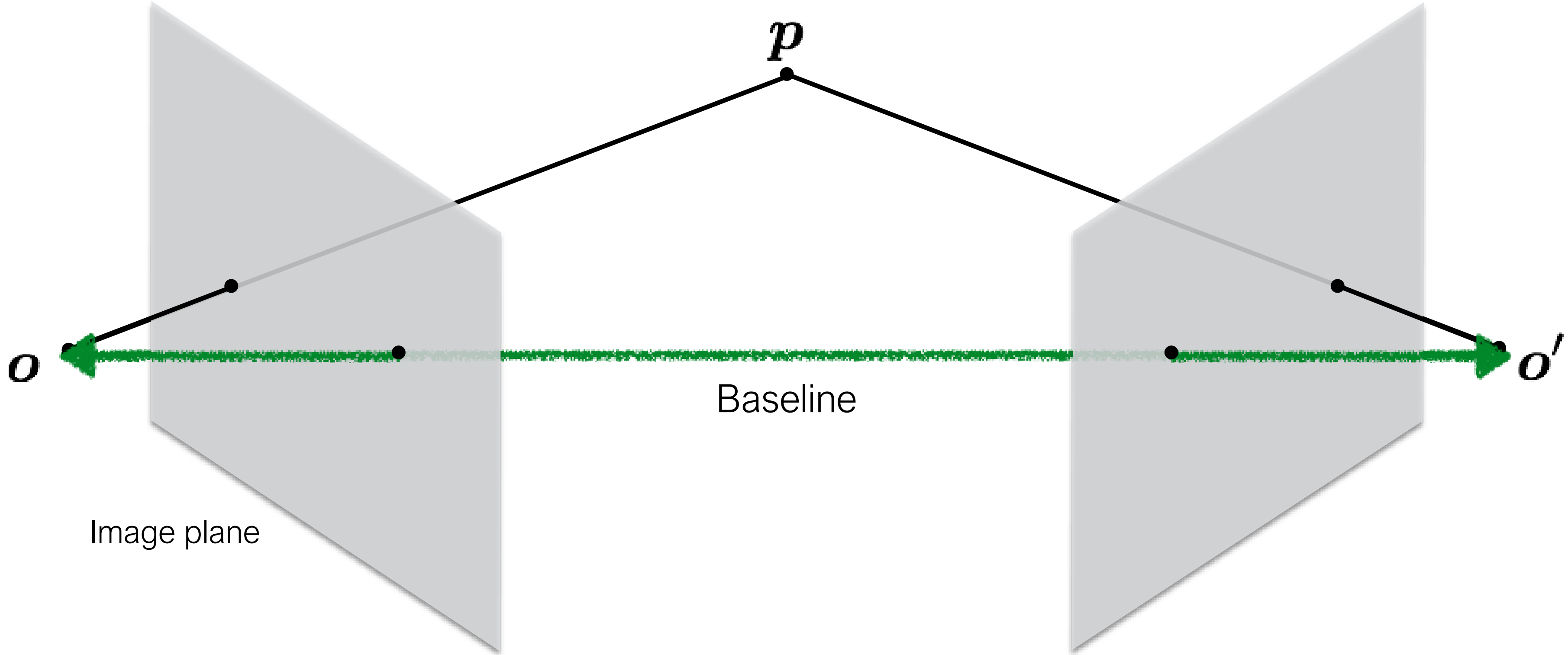
How do we solve homogeneous linear system?

S V D !

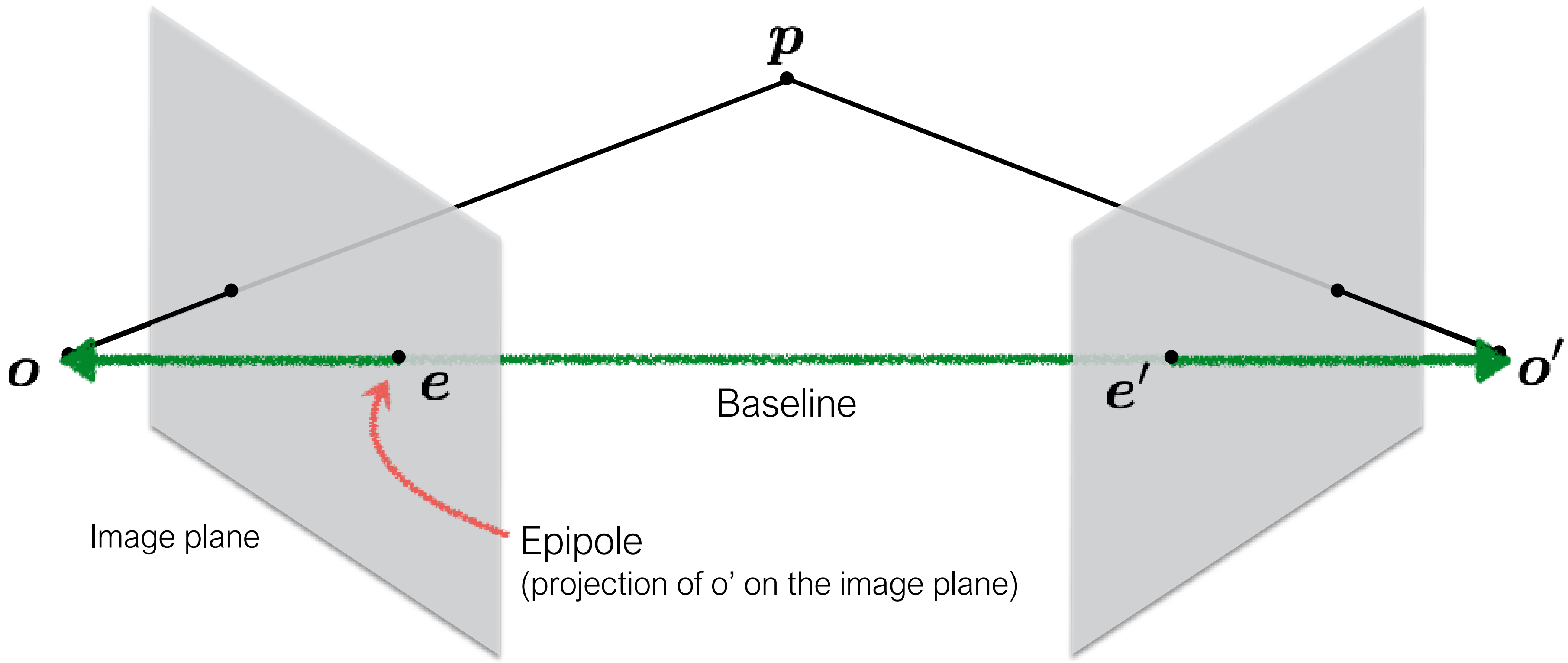
Epipolar geometry



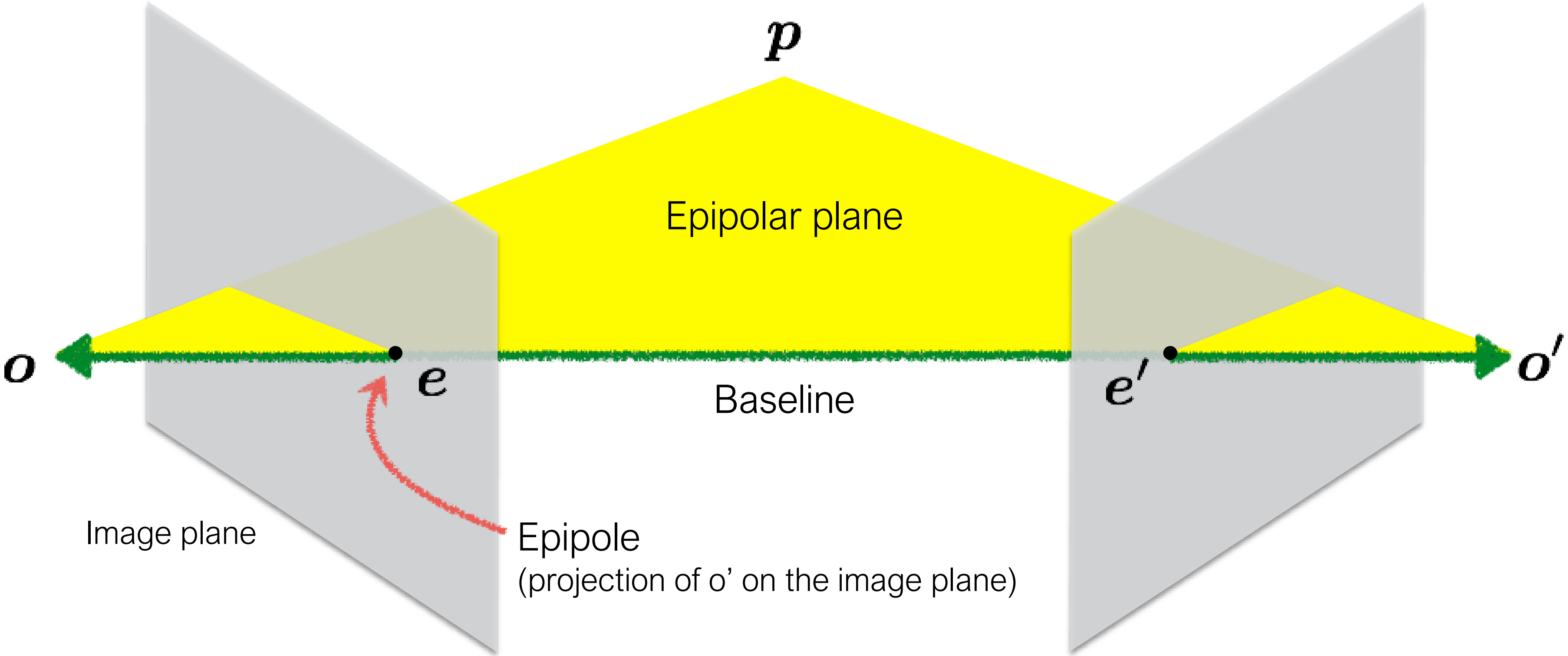
Epipolar geometry



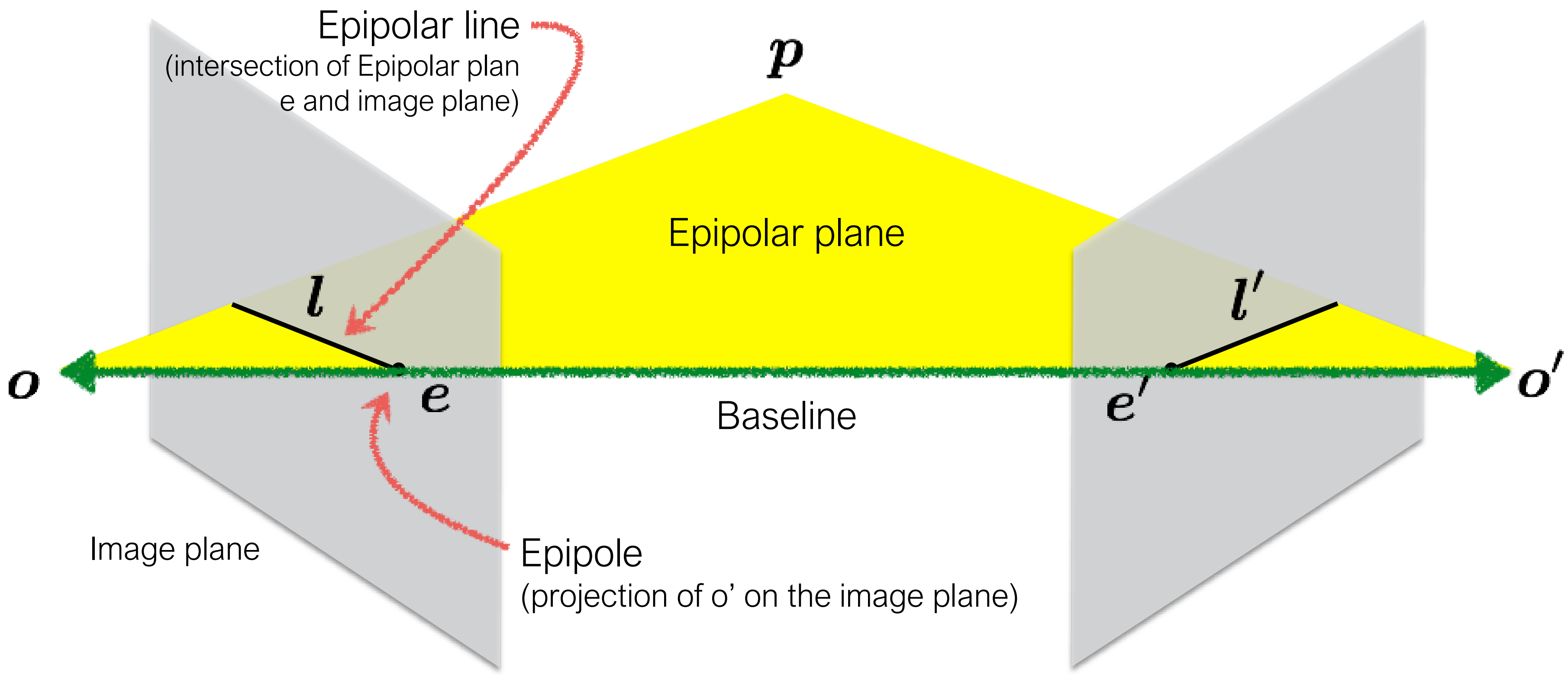
Epipolar geometry



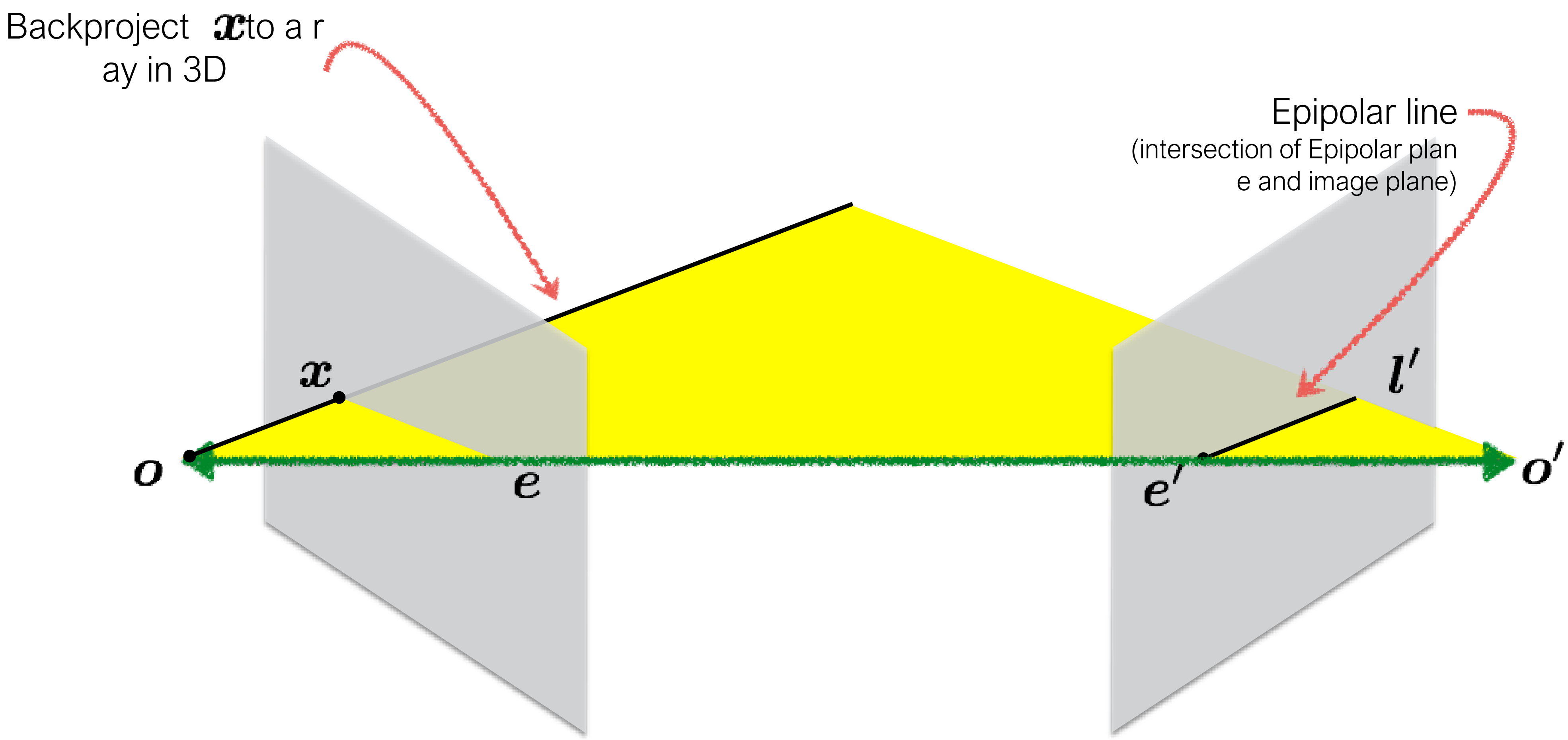
Epipolar geometry



Epipolar geometry

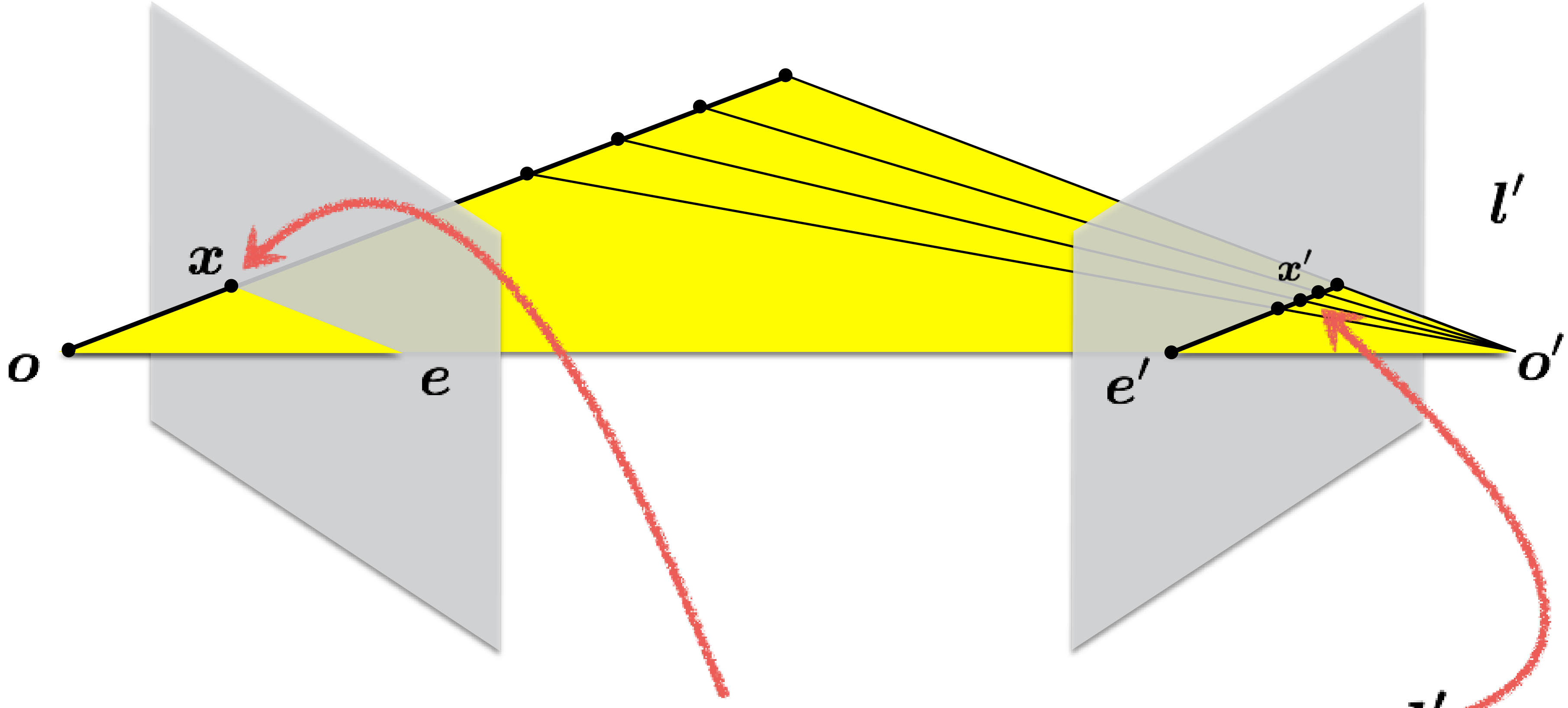


Epipolar constraint



Another way to construct the epipolar plane, this time given $\boldsymbol{x}$

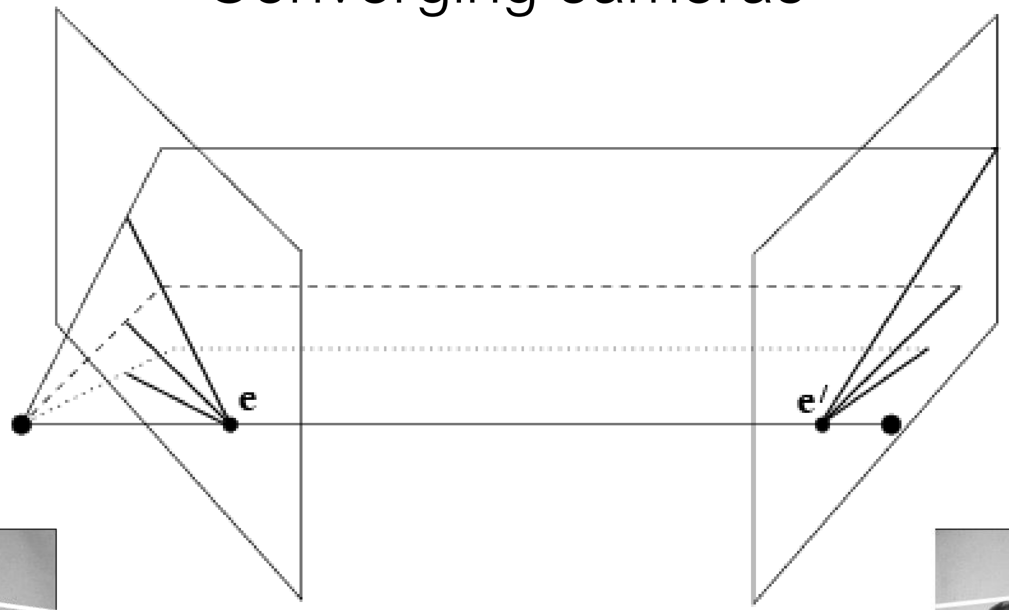
Epipolar constraint

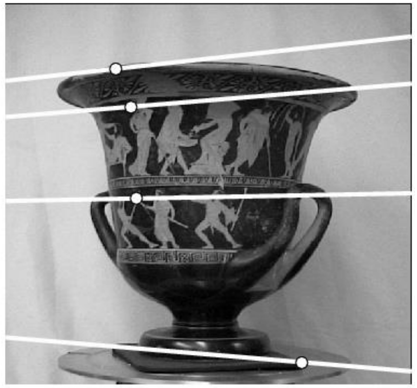


Potential matches for x lie on the epipolar line l'

Epipolar constraint

Converging cameras

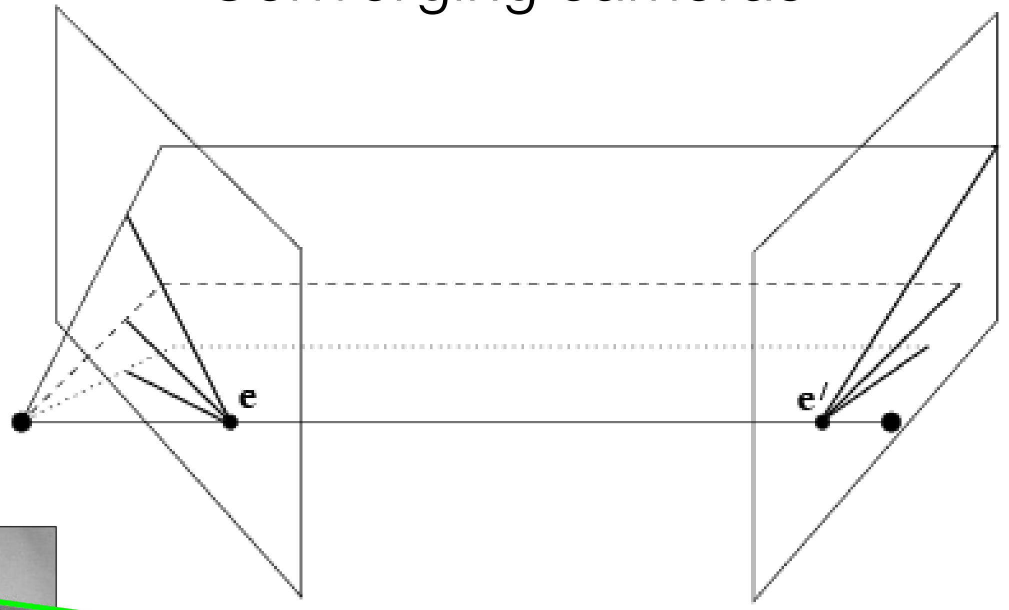


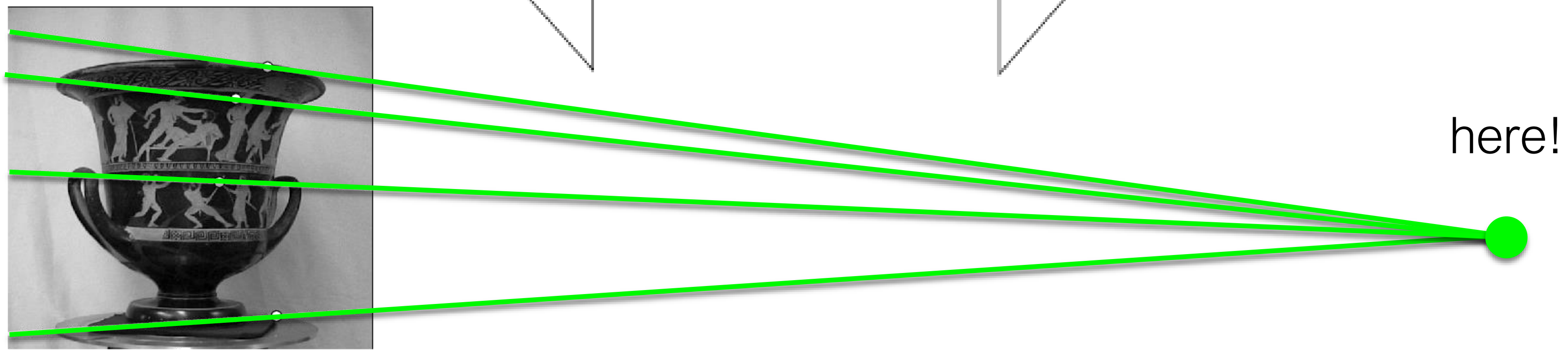


Where is the epipole in this image?

Epipolar constraint

Converging cameras

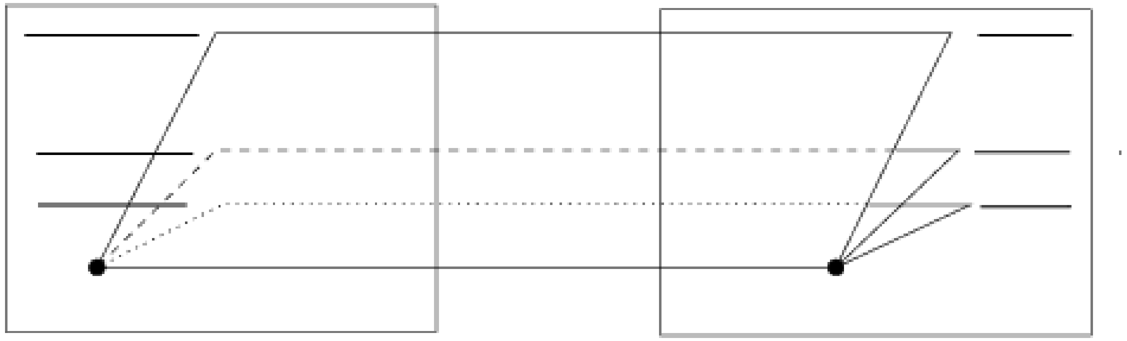


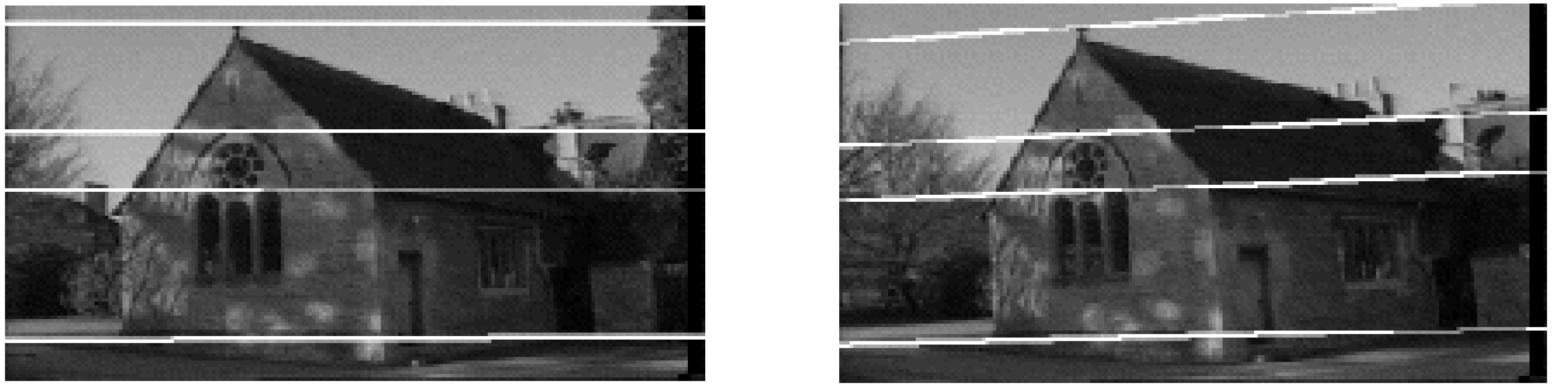


Where is the epipole in this image?

It's not always in the image

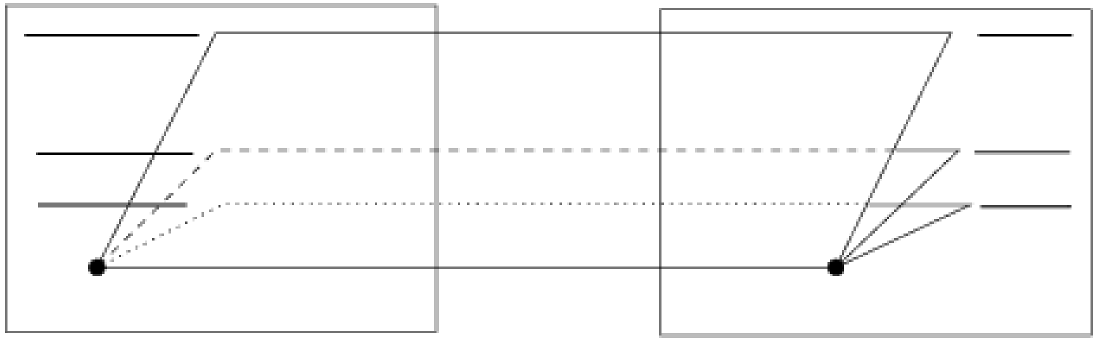
Parallel cameras

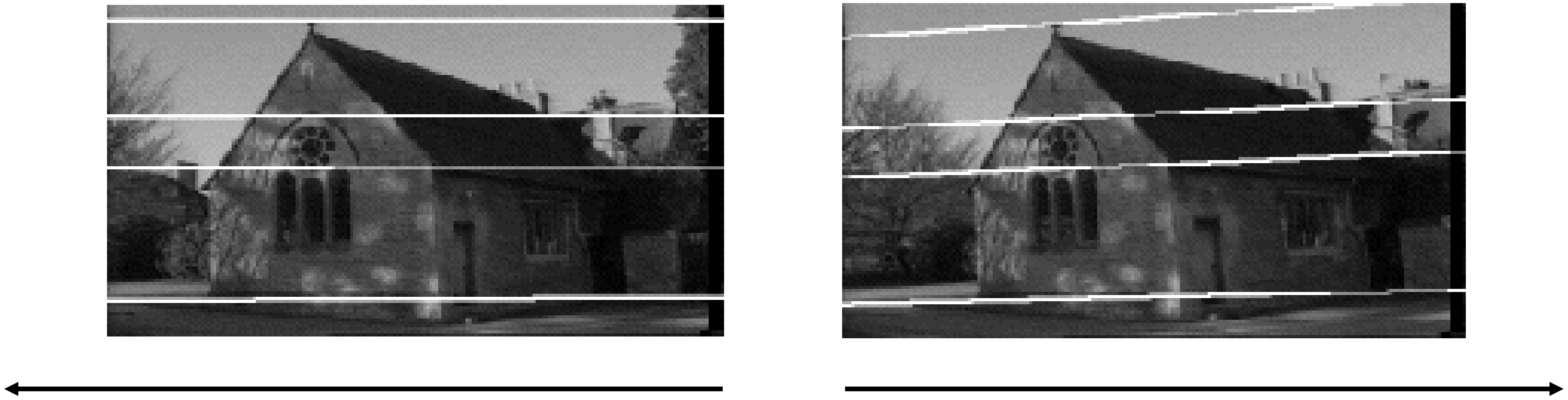




Where is the epipole?

Parallel cameras



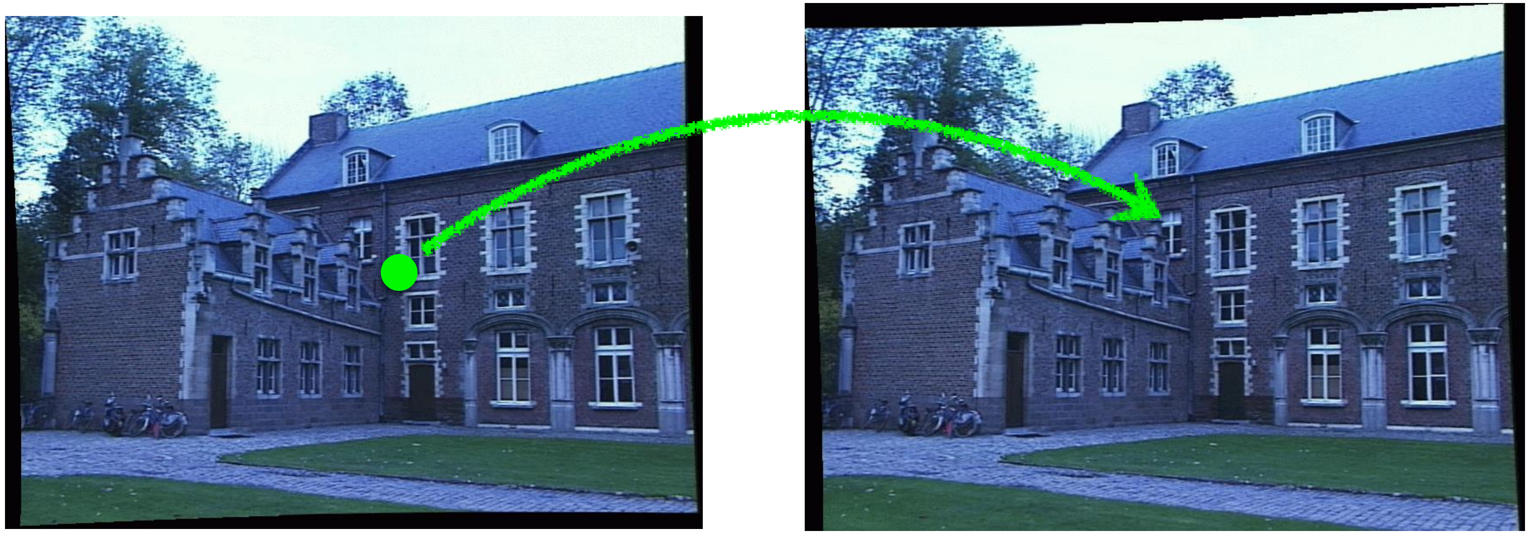


epipole at infinity

Epipolar constraint

The epipolar constraint is an important concept for stereo vision

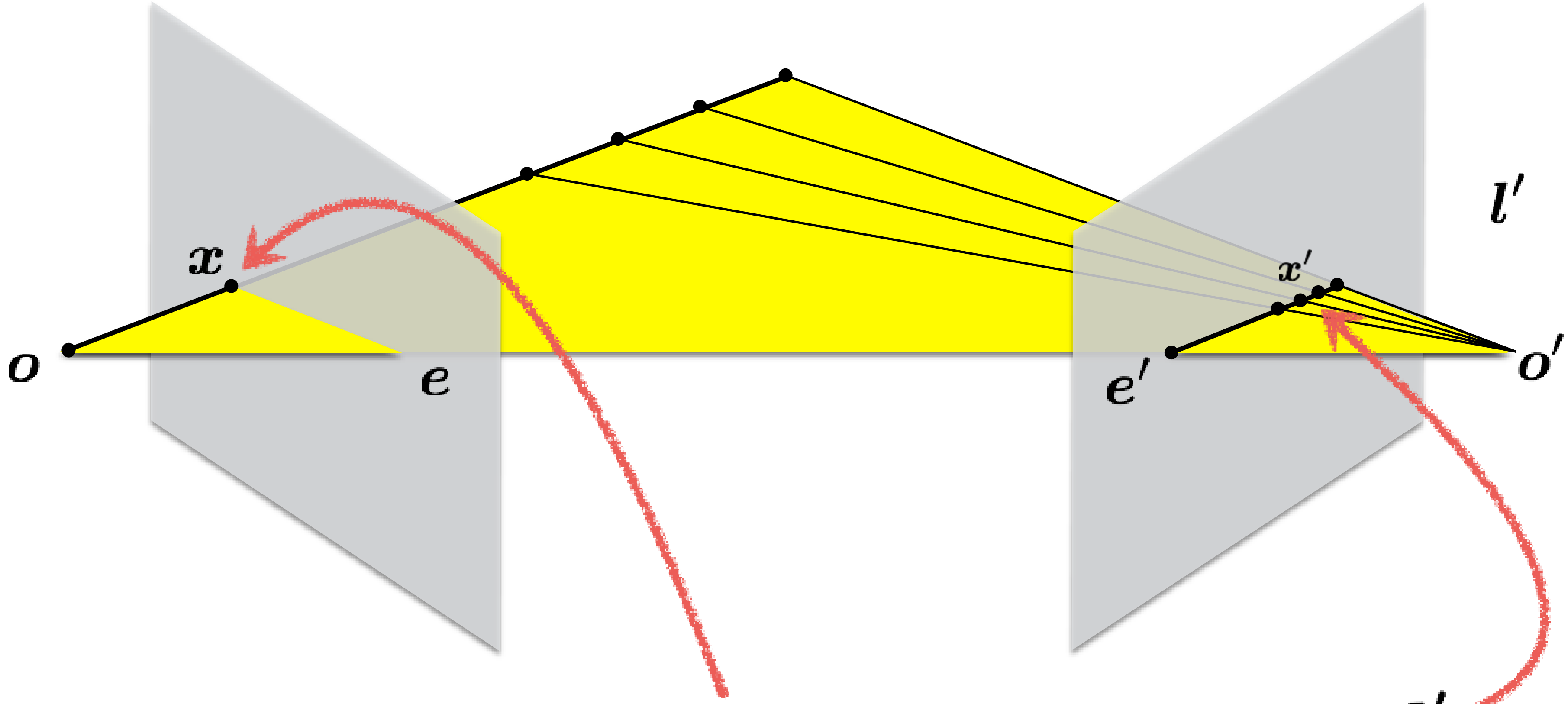
Task: Match point in left image to point in right image



Left image Right image

How would you do it?

Epipolar constraint

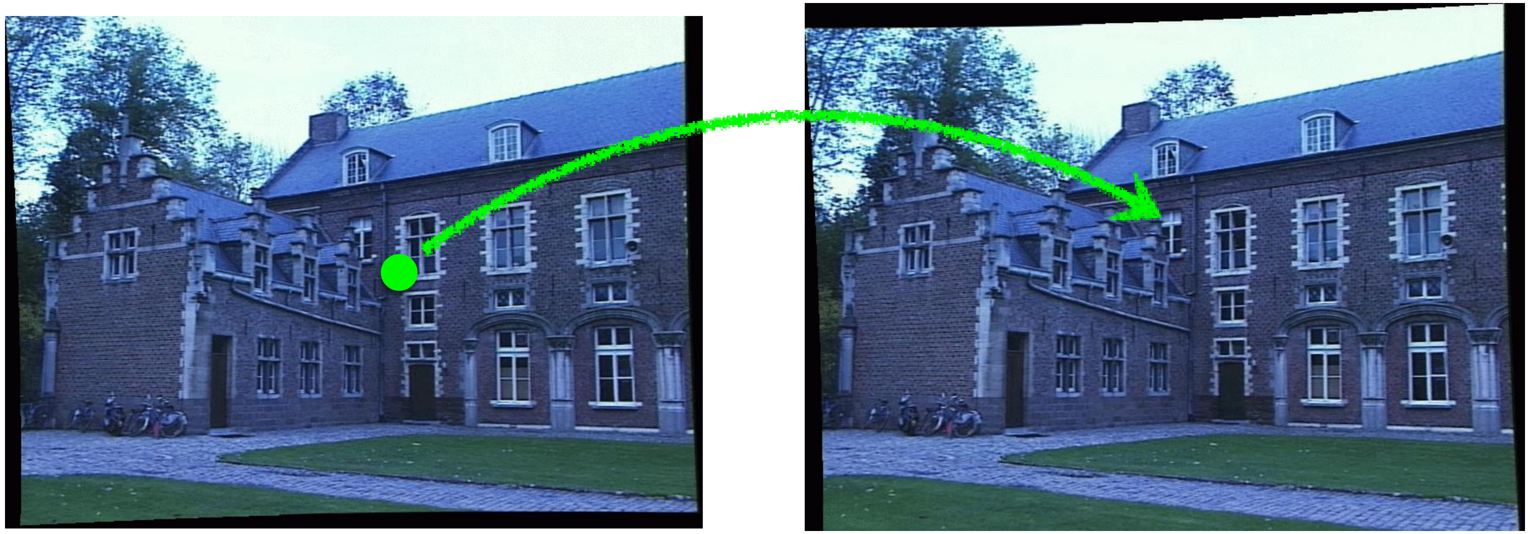


Potential matches for x lie on the epipolar line l'

Epipolar constraint

The epipolar constraint is an important concept for stereo vision

Task: Match point in left image to point in right image



Left image

Right image

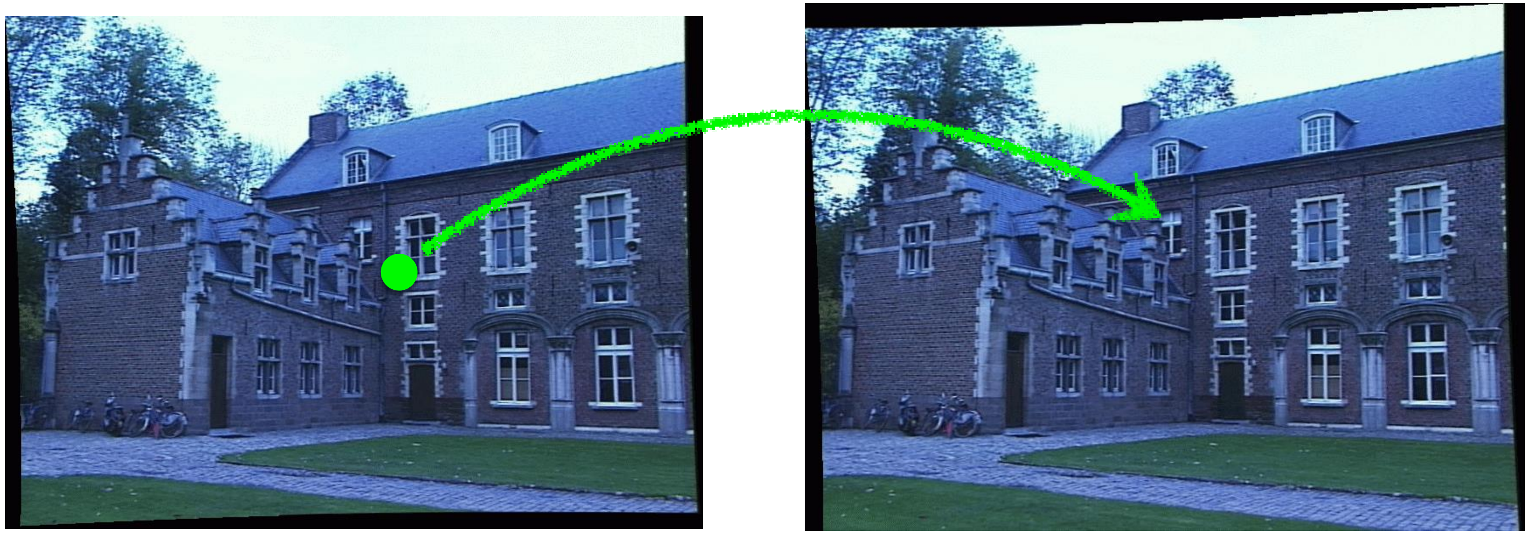
Want to avoid search over entire image

Epipolar constraint reduces search to a single line

Epipolar constraint

The epipolar constraint is an important concept for stereo vision

Task: Match point in left image to point in right image



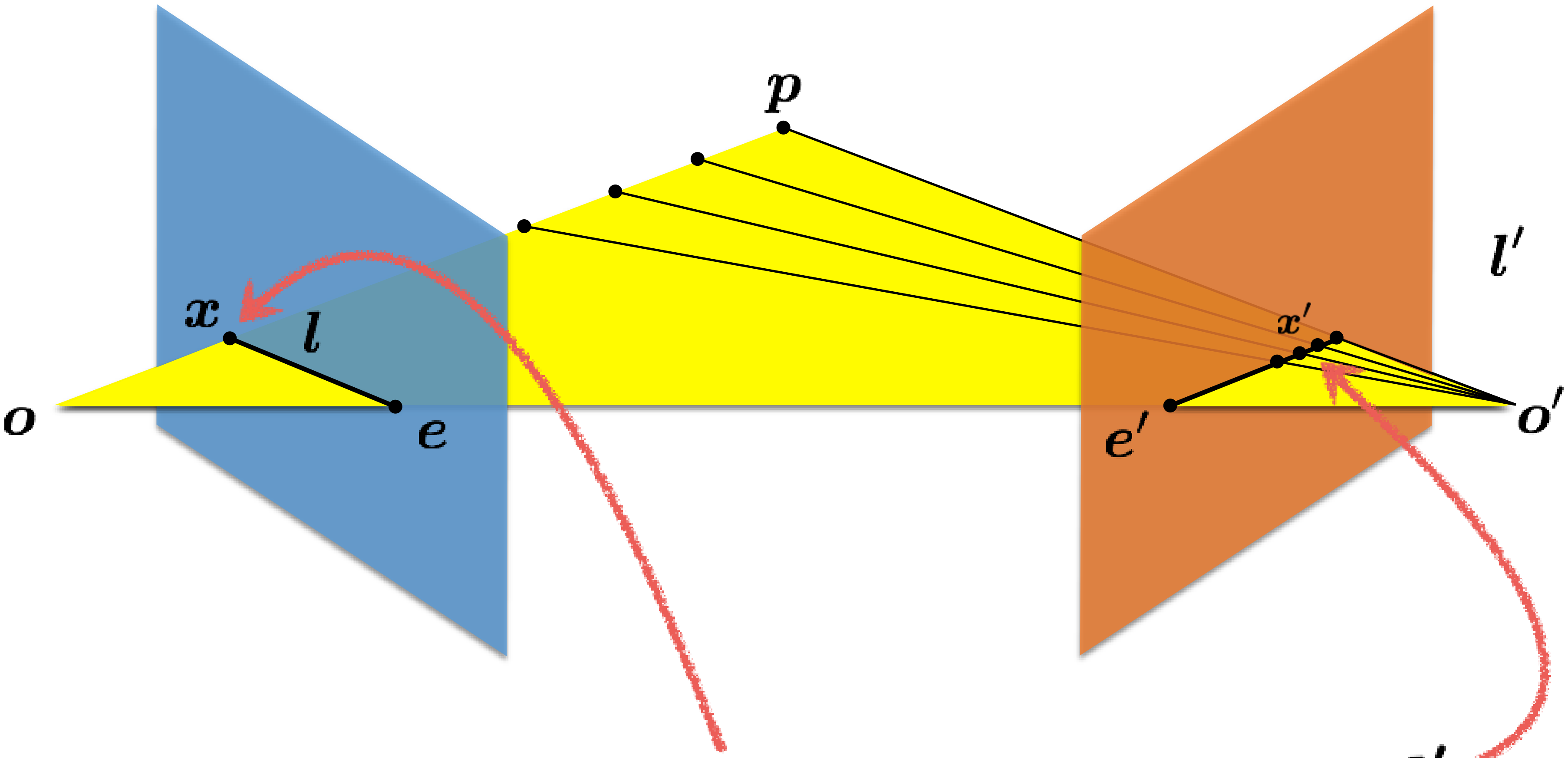
Left image Right image

Want to avoid search over entire image

Epipolar constraint reduces search to a single line

How do you compute the epipolar line?

Recall:Epipolar constraint

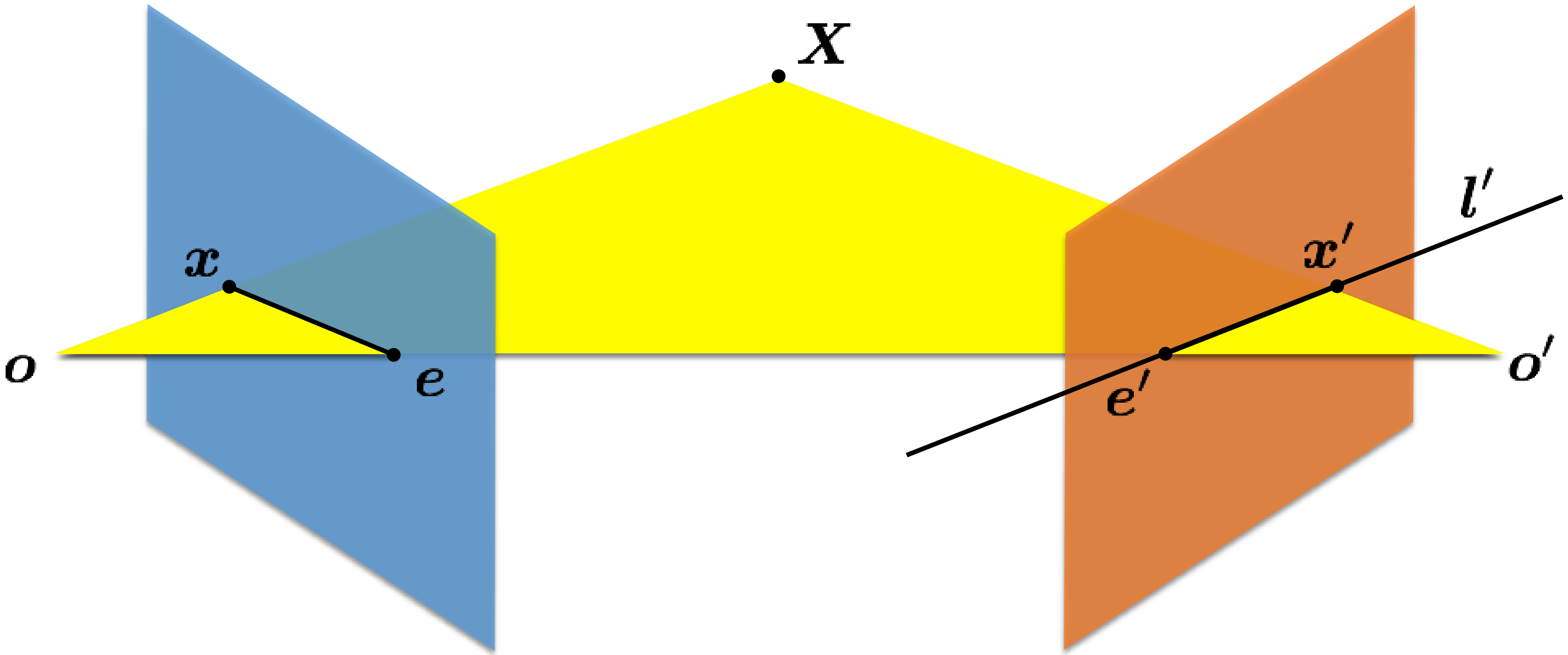


Potential matches for x lie on the epipolar line l'

Recall:Epipolar constraint

Given a point in one image,
multiplying by the **essential matrix** will tell us
the **epipolar line** in the second view.

$$\boxed{\mathbf{E}\boldsymbol{x} = \boldsymbol{l}'}$$



Motivation

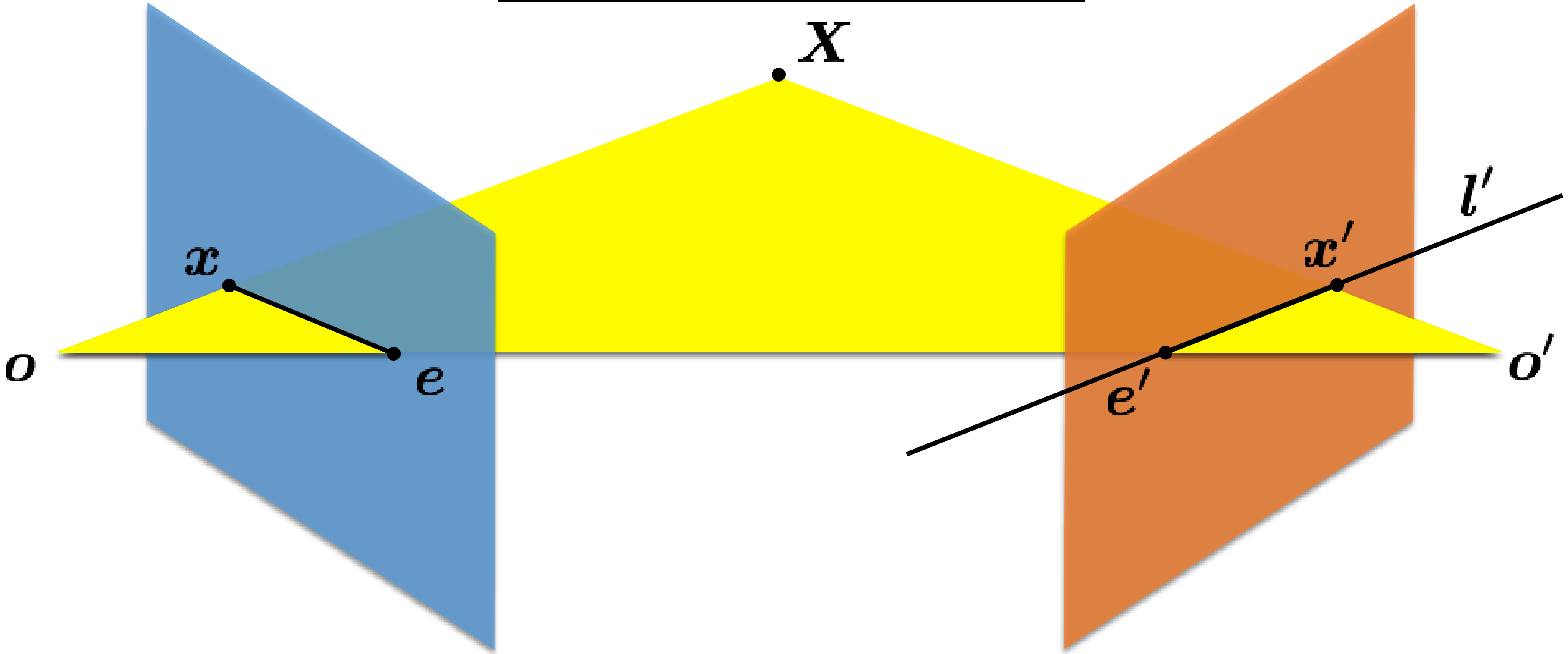
The Essential Matrix is a 3 x 3 matrix that en
codes **epipolar geometry**

Given a point in one image,
multiplying by the **essential matrix** will tell us
the **epipolar line** in the second image.

Epipolar Line

So if $\boldsymbol{x}'^{\top}\boldsymbol{l}' = 0$ and $\mathbf{E}\boldsymbol{x} = \boldsymbol{l}'$ then

$$\boxed{\boldsymbol{x}'^{\top}\mathbf{E}\boldsymbol{x} = 0}$$



Essential Matrix vs Homography

What's the difference between the essential matrix and a homography?

They are both 3 x 3 matrices but …

$$\boldsymbol{l}' = \mathbf{E}\boldsymbol{x}$$

Essential matrix maps a
point to a **line**

$$\boldsymbol{x}' = \mathbf{H}\boldsymbol{x}$$

Homography maps a
point to a **point**

properties of the E matrix

Longuet-Higgins equation $\boldsymbol{x}'^{\top}\mathbf{E}\boldsymbol{x} = 0$

(2D points expressed in camera coordinate system)

properties of the E matrix

Longuet-Higgins equation $\boldsymbol{x}'^{\top}\mathbf{E}\boldsymbol{x} = 0$

Epipolar lines

$\boldsymbol{x}^{\top}\boldsymbol{l} = 0$ $\boldsymbol{x}'^{\top}\boldsymbol{l}' = 0$

$\boldsymbol{l}' = \mathbf{E}\boldsymbol{x}$ $\boldsymbol{l} = \mathbf{E}^{T}\boldsymbol{x}'$

(2D points expressed in camera coordinate system)

properties of the E matrix

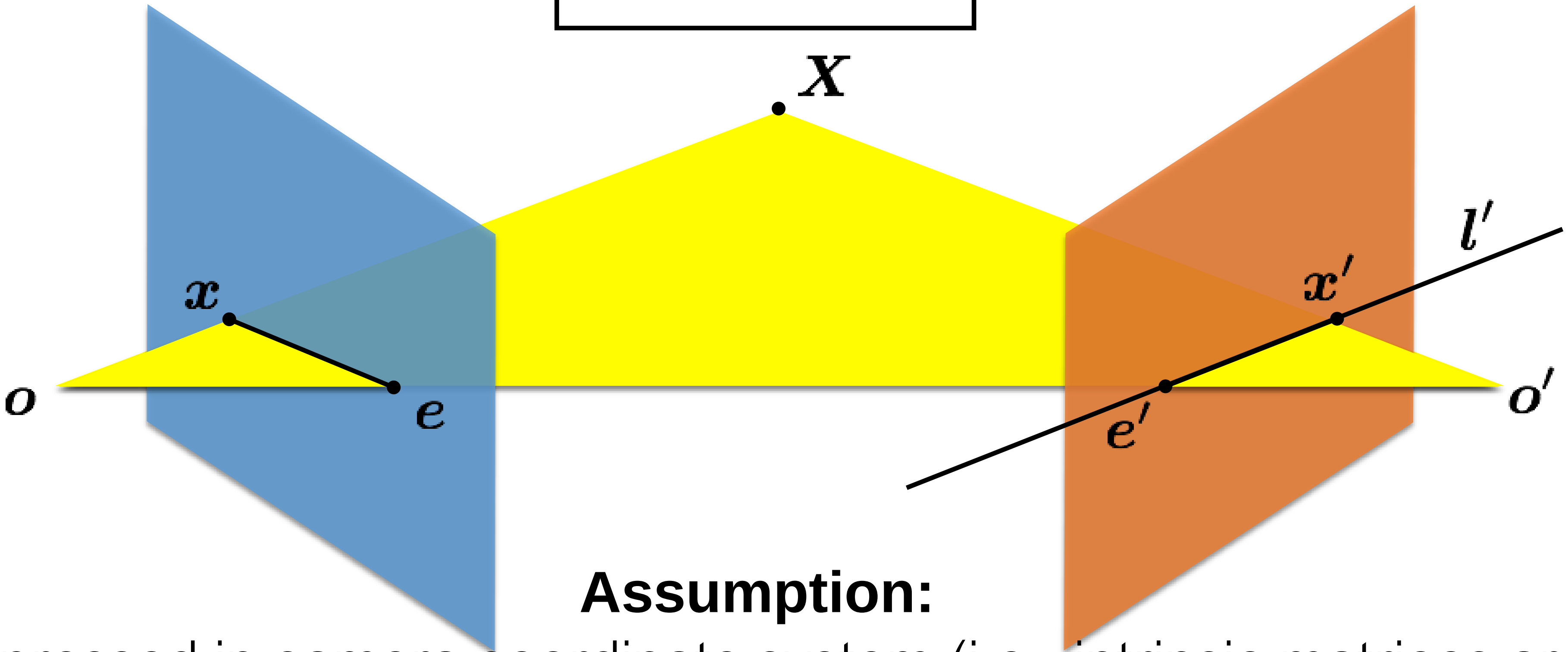
Longuet-Higgins equation	$\boldsymbol{x}'^{\top}\mathbf{E}\boldsymbol{x} = 0$	
Epipolar lines	$\boldsymbol{x}^{\top}\boldsymbol{l} = 0$ $\boldsymbol{l}' = \mathbf{E}\boldsymbol{x}$	$\boldsymbol{x}'^{\top}\boldsymbol{l}' = 0$ $\boldsymbol{l} = \mathbf{E}^{T}\boldsymbol{x}'$
Epipoles	$\boldsymbol{e}'^{\top}\mathbf{E} = \mathbf{0}$	$\mathbf{E}\boldsymbol{e} = \mathbf{0}$

(2D points expressed in <u>camera</u> coordinate system)

Essential Matrix

Given a point in one image,
multiplying by the **essential matrix** will tell us
the **epipolar line** in the second view.

$$\boxed{\mathbf{E}\boldsymbol{x} = \boldsymbol{l}'}$$



Assumption:
2D points expressed in camera coordinate system (i.e., intrinsic matrices are identities)

How do you generalize to non-identity intrinsic matrices?

Fundamental matrix

The

fundamental matrix

is a

generalization

of the

essential matrix,

where the assumption of

Identity matrices

is removed

Fundamental matrix

$$\hat{\boldsymbol{x}}'^{\top}\mathbf{E}\hat{\boldsymbol{x}} = 0$$

The essential matrix operates on image points expressed in
2D coordinates expressed in the camera coordinate system

$$\hat{\boldsymbol{x}}' = \mathbf{K}'^{-1}\boldsymbol{x}'$$

$$\underbrace{\hat{\boldsymbol{x}}}_{\text{camera point}} = \mathbf{K}^{-1}\underbrace{\boldsymbol{x}}_{\text{image point}}$$

$$\hat{\boldsymbol{x}}'^{\top}\mathbf{E}\hat{\boldsymbol{x}} = 0$$

The essential matrix operates on image points expressed in
2D coordinates expressed in the camera coordinate system

$$\hat{\boldsymbol{x}}' = \mathbf{K}'^{-1}\boldsymbol{x}'$$

$$\hat{\boldsymbol{x}} = \mathbf{K}^{-1}\boldsymbol{x}$$

camera point

image point

Writing out the epipolar constraint in terms of image coordinates

$$\boldsymbol{x}'^{\top}\mathbf{K}'^{-\top}\mathbf{E}\mathbf{K}^{-1}\boldsymbol{x} = 0$$

$$\boldsymbol{x}'^{\top}(\mathbf{K}'^{-\top}\mathbf{E}\mathbf{K}^{-1})\boldsymbol{x} = 0$$

$$\boldsymbol{x}'^{\top}\mathbf{F}\boldsymbol{x} = 0$$

Fundamental matrix

Same equation works in image coordinates!

$$\boldsymbol{x}'^{\top}\mathbf{F}\boldsymbol{x} = 0$$

it maps pixels to epipolar lines

properties of the E matrix

Longuet-Higgins equation	$\boldsymbol{x}'^{\top}\mathbf{E}\boldsymbol{x} = 0$	
Epipolar lines	$\boldsymbol{x}^{\top}\boldsymbol{l} = 0$	$\boldsymbol{x}'^{\top}\boldsymbol{l}' = 0$
	$\boldsymbol{l}' = \mathbf{E}\boldsymbol{x}$	$\boldsymbol{l} = \mathbf{E}^{T}\boldsymbol{x}'$
Epipoles	$\boldsymbol{e}'^{\top}\mathbf{E} = \mathbf{0}$	$\mathbf{E}\boldsymbol{e} = \mathbf{0}$

(points in **image** coordinates)

8-point algorithm

Assume you have *M* matched *image* points

$$\{\boldsymbol{x}_m, \boldsymbol{x}'_m\} \qquad m = 1, \ldots, M$$

Each correspondence should satisfy

$$\boldsymbol{x}'^{\top}_m \mathbf{F} \boldsymbol{x}_m = 0$$

How would you solve for the 3 x 3 ***F*** *matrix?*

Set up a homogeneous linear system with 9 unknowns

8-point algorithm

$$\boldsymbol{x}'^{\top}_m \mathbf{F} \boldsymbol{x}_m = 0$$

$$\begin{bmatrix} x'_m & y'_m & 1 \end{bmatrix} \begin{bmatrix} f_1 & f_2 & f_3 \\ f_4 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{bmatrix} \begin{bmatrix} x_m \\ y_m \\ 1 \end{bmatrix} = 0$$

How many equation do you get from one correspondence?

8-point algorithm

$$\begin{bmatrix} x'_m & y'_m & 1 \end{bmatrix} \begin{bmatrix} f_1 & f_2 & f_3 \\ f_4 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{bmatrix} \begin{bmatrix} x_m \\ y_m \\ 1 \end{bmatrix} = 0$$

ONE correspondence gives you ONE equation

$$\begin{aligned} x_m x'_m f_1 + x_m y'_m f_2 + x_m f_3 + \\ y_m x'_m f_4 + y_m y'_m f_5 + y_m f_6 + \\ x'_m f_7 + y'_m f_8 + f_9 = 0 \end{aligned}$$

8-point algorithm

$$\begin{bmatrix} x'_m & y'_m & 1 \end{bmatrix} \begin{bmatrix} f_1 & f_2 & f_3 \\ f_4 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{bmatrix} \begin{bmatrix} x_m \\ y_m \\ 1 \end{bmatrix} = 0$$

Set up a homogeneous linear system with 9 unknowns

$$\begin{bmatrix} x_1x'_1 & x_1y'_1 & x_1 & y_1x'_1 & y_1y'_1 & y_1 & x'_1 & y'_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_Mx'_M & x_My'_M & x_M & y_Mx'_M & y_My'_M & y_M & x'_M & y'_M & 1 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \\ f_8 \\ f_9 \end{bmatrix} = \mathbf{0}$$

How many equations do you need?

8-point algorithm

Each point pair (according to epipolar constraint) contributes only one scalar equation

$$\boldsymbol{x}_m'^{\top} \mathbf{F} \boldsymbol{x}_m = 0$$

Note: This is different from the Homography estimation where each point pair contributes 2 equations.

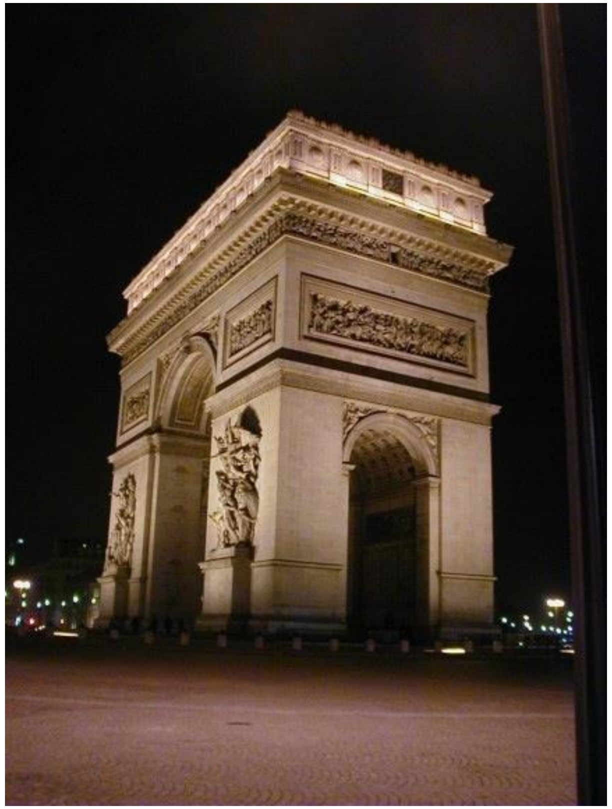
We need at least 8 points

Hence, the 8 point algorithm!

Eight-Point Algorithm

0. (Normalize points)
1. Construct the M x 9 matrix **A**
2. Find the SVD of **A**
3. Entries of **F** are the elements of column of **V** corresponding to the least singular value
4. (Enforce rank 2 constraint on F)
5. (Un-normalize F)

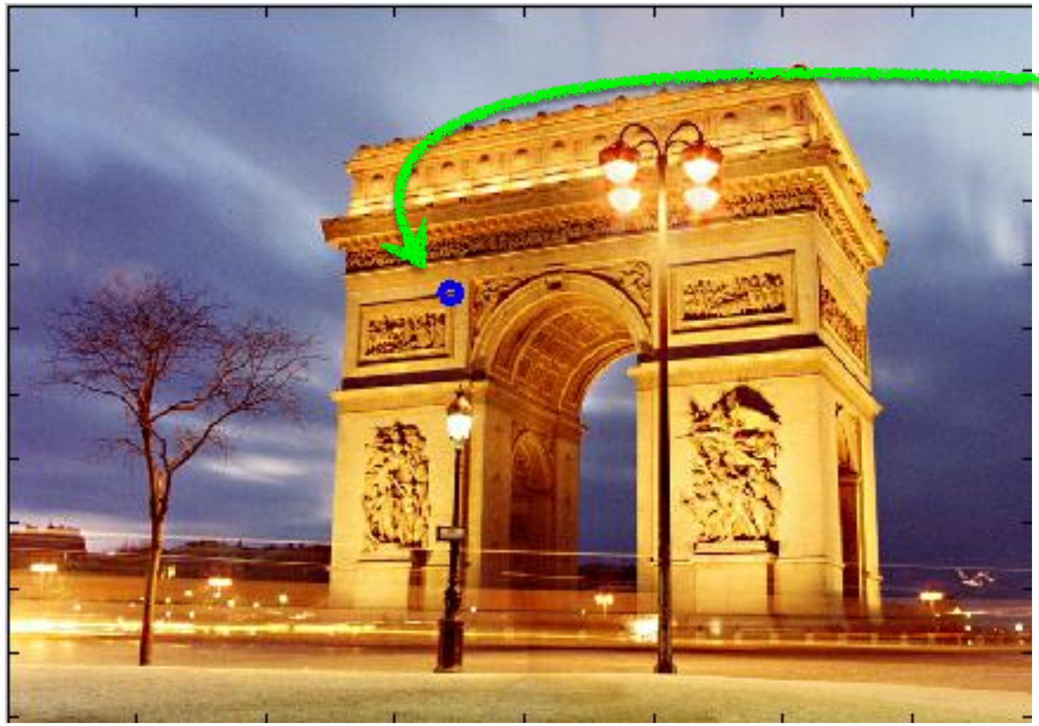
Example



Epipolar lines

Computation

$$\mathbf{F} = \begin{bmatrix} -0.00310695 & -0.0025646 & 2.96584 \\ -0.028094 & -0.00771621 & 56.3813 \\ 13.1905 & -29.2007 & -9999.79 \end{bmatrix}$$



$$\boldsymbol{x} = \begin{bmatrix} 343.53 \\ 221.70 \\ 1.0 \end{bmatrix}$$

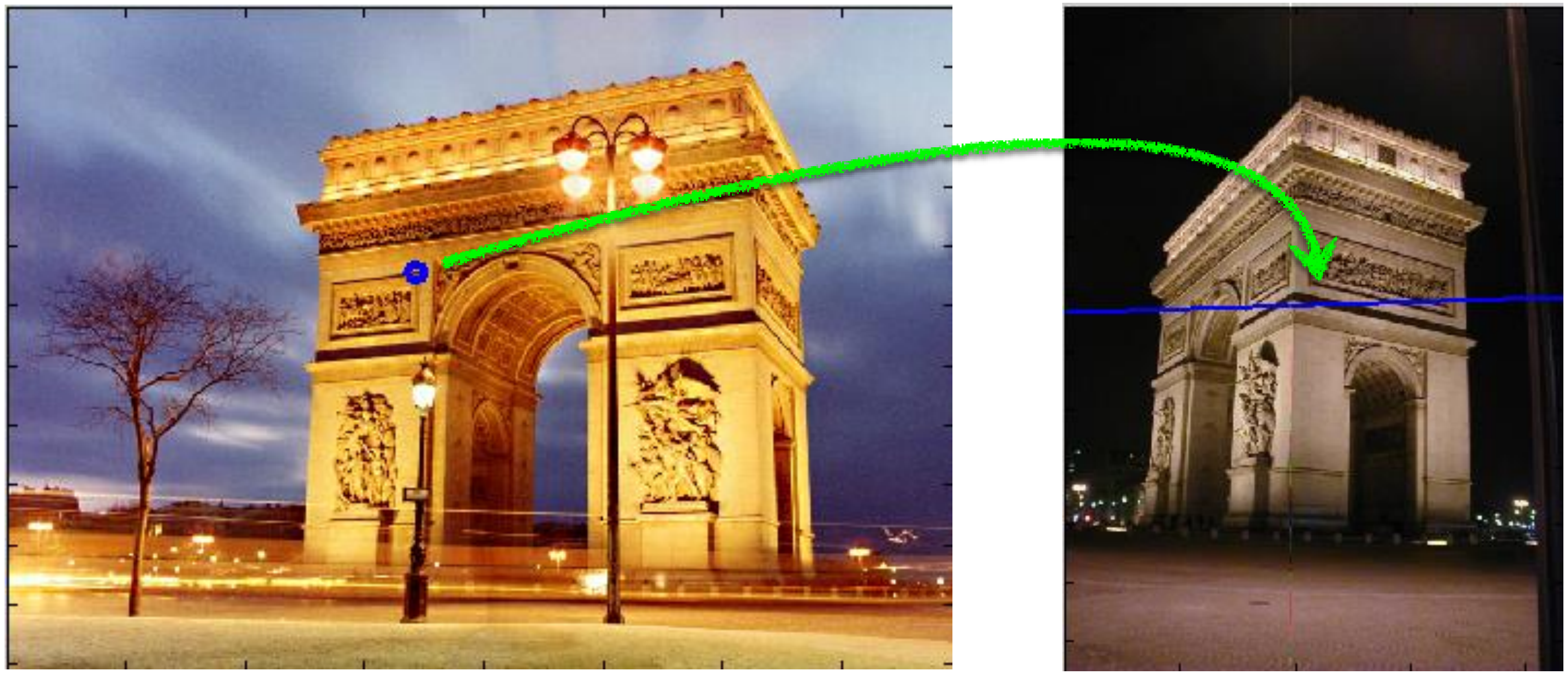
$$\boldsymbol{l}' = \mathbf{F}\boldsymbol{x}$$

$$= \begin{bmatrix} 0.0295 \\ 0.9996 \\ -265.1531 \end{bmatrix}$$

Computation

$$\boldsymbol{l}' = \mathbf{F}\boldsymbol{x}$$

$$= \begin{bmatrix} 0.0295 \\ 0.9996 \\ -265.1531 \end{bmatrix}$$



Where is the epipole?

How would you compute it?

Where is the epipole?

$$\mathbf{F}\boldsymbol{e} = \mathbf{0}$$

The epipole is in the right null space of **F**

How would you solve for the epipole?

Where is the epipole?

$$\mathbf{F}\boldsymbol{e} = \mathbf{0}$$

The epipole is in the right null space of **F**

How would you solve for the epipole?

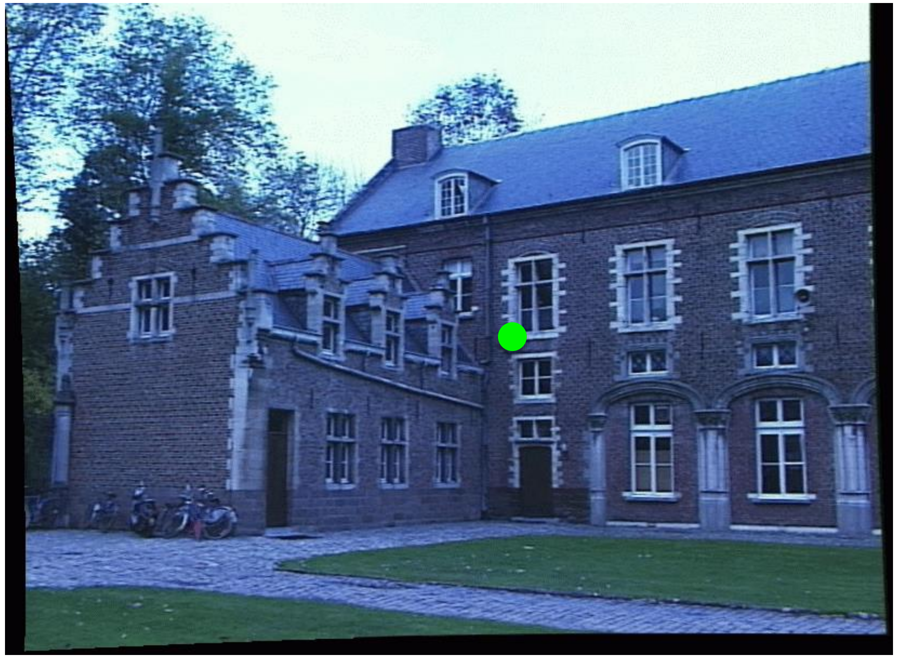
S V D !

How would you reconstruct 3D points?

Left image

Right image

How would you reconstruct 3D points?

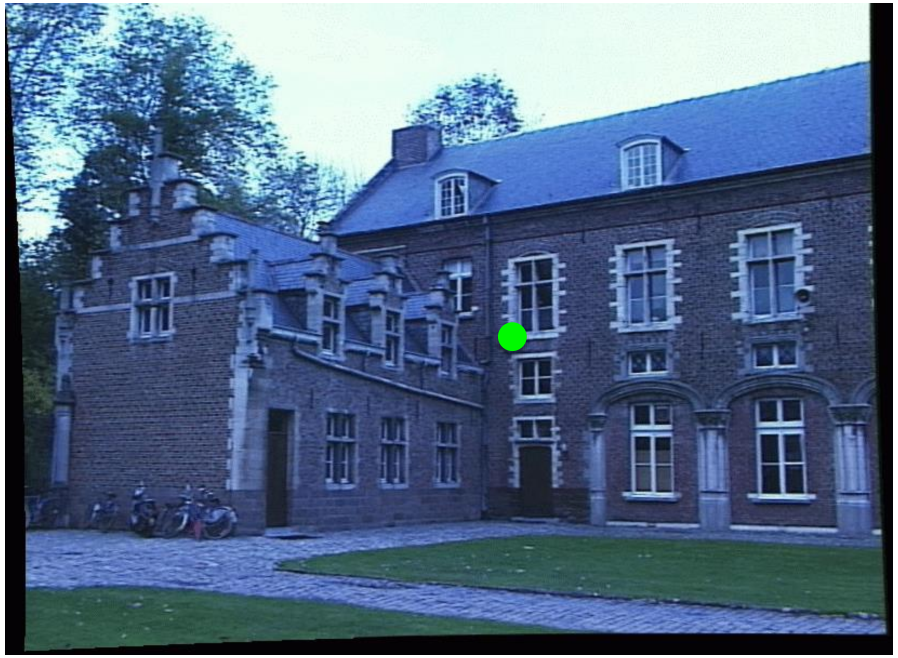


Left image

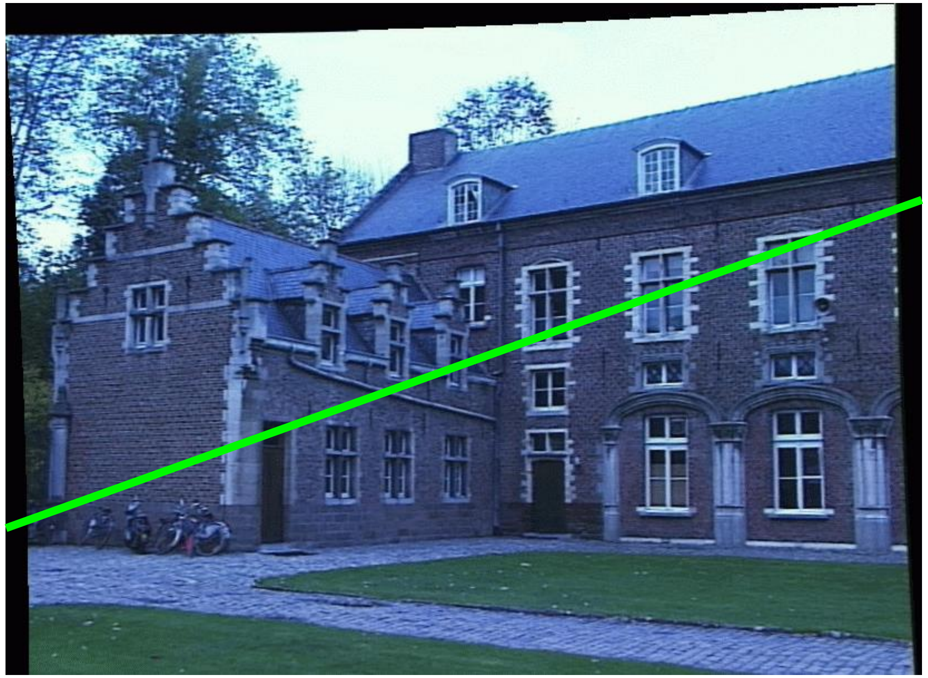
Right image

1. Select point in one image (how?)

How would you reconstruct 3D points?



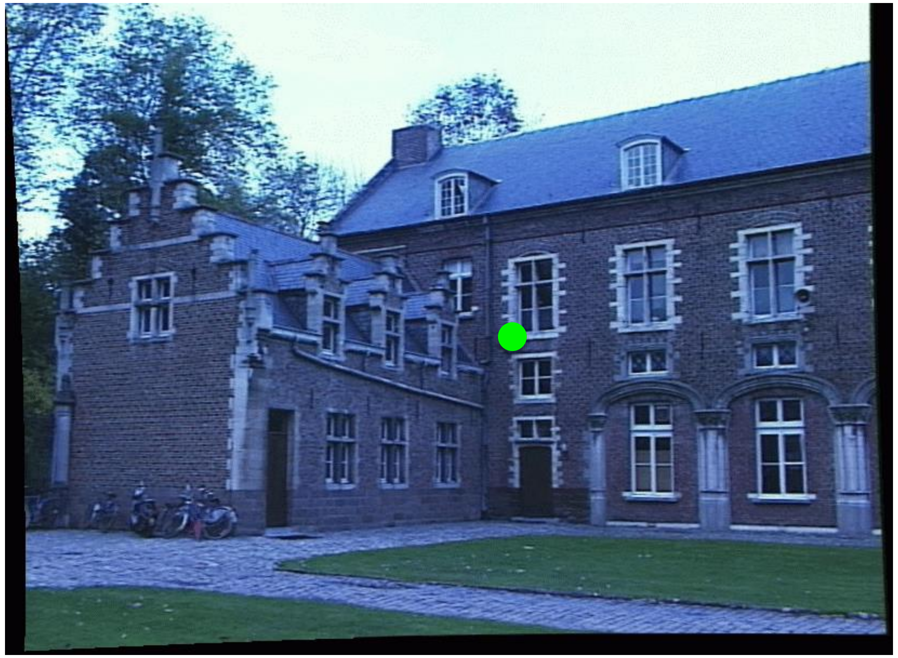
Left image



Right image

1. Select point in one image (how?)
2. Form epipolar line for that point in second image (how?)

How would you reconstruct 3D points?

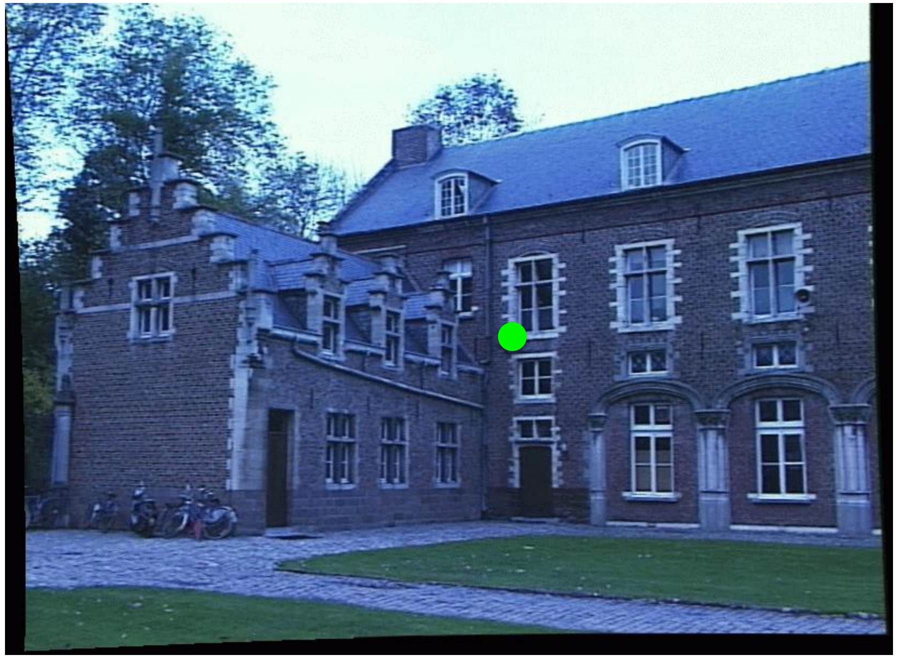


Left image

Right image

1. Select point in one image (how?)
2. Form epipolar line for that point in second image (how?)
3. Find matching point along line (how?)

How would you reconstruct 3D points?

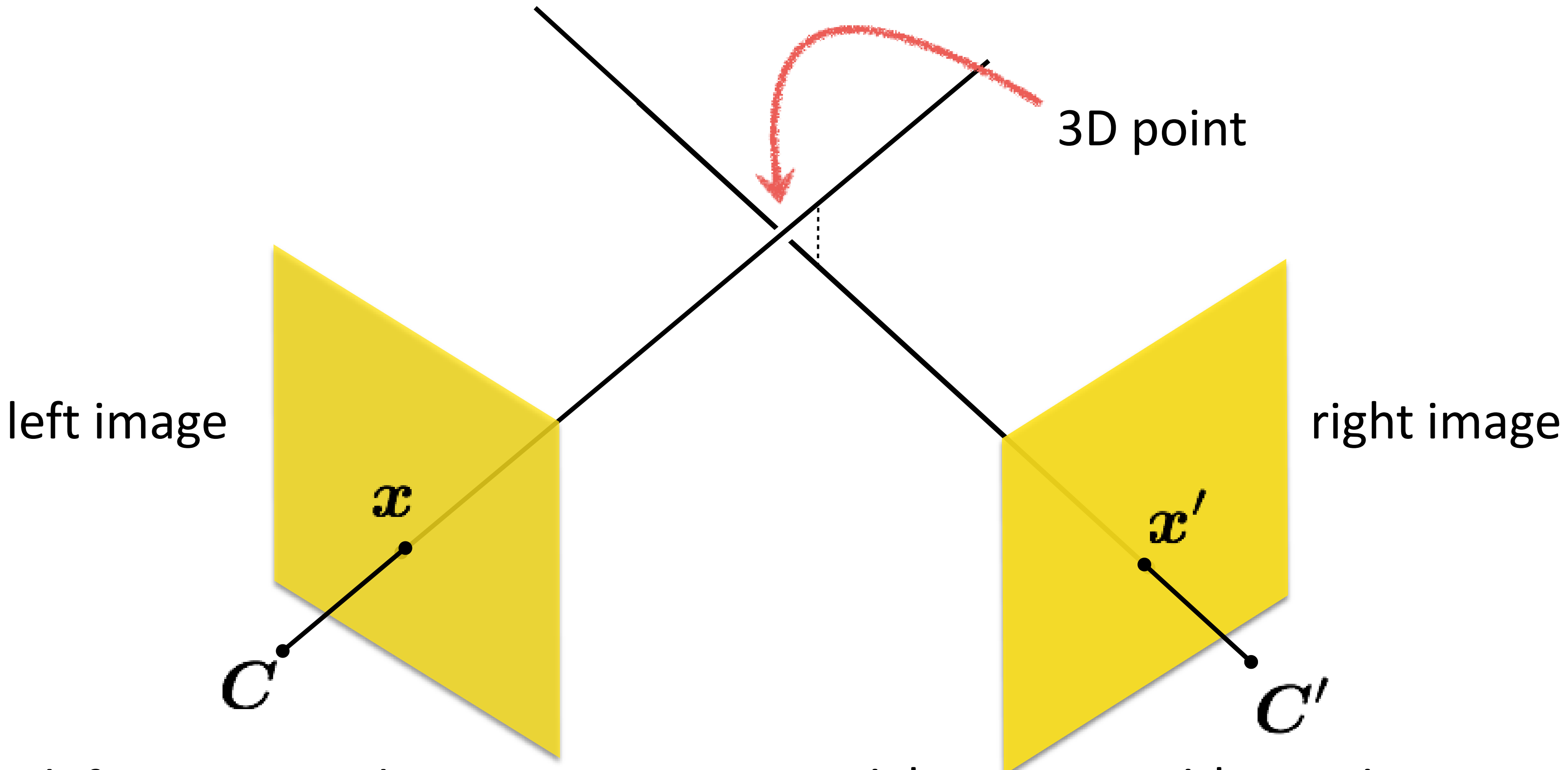


Left image

Right image

1. Select point in one image (how?)
2. Form epipolar line for that point in second image (how?)
3. Find matching point along line (how?)
4. Perform triangulation (how?)

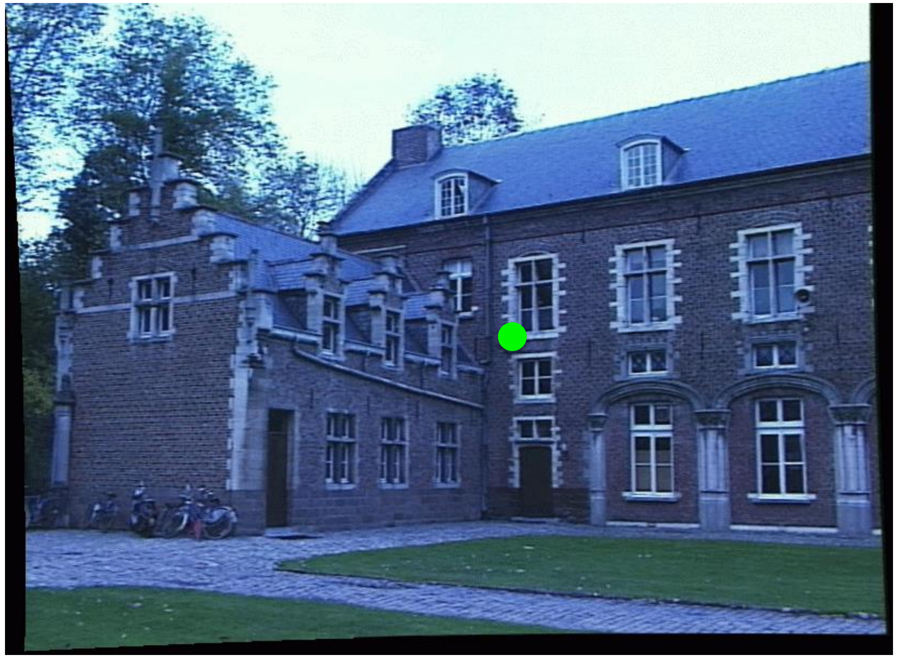
Triangulation



left camera with matri $\mathbf{P}$
x

right camera with matri $\mathbf{P}'$
x

How would you reconstruct 3D points?



Left image

Right image

1. Select point in one image (how?)
2. Form epipolar line for that point in second image (how?)
3. Find matching point along line (how?)
4. Perform triangulation (how?)

What are the disadvantages of this procedure?

Stereo rectification

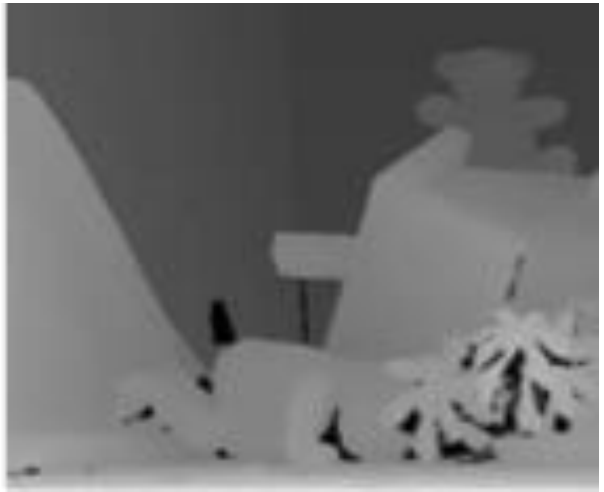
What's different between these two images?

Stereo rectification

Objects that are close move more or less?

Stereo rectification

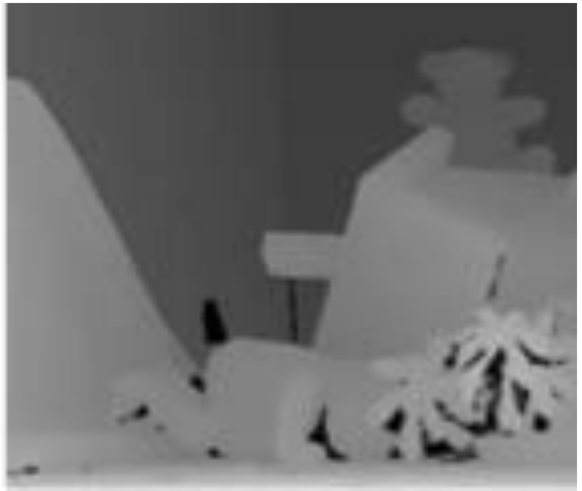
The amount of horizontal movement is inversely proportional to …



Stereo rectification

The amount of horizontal movement is inversely proportional to …

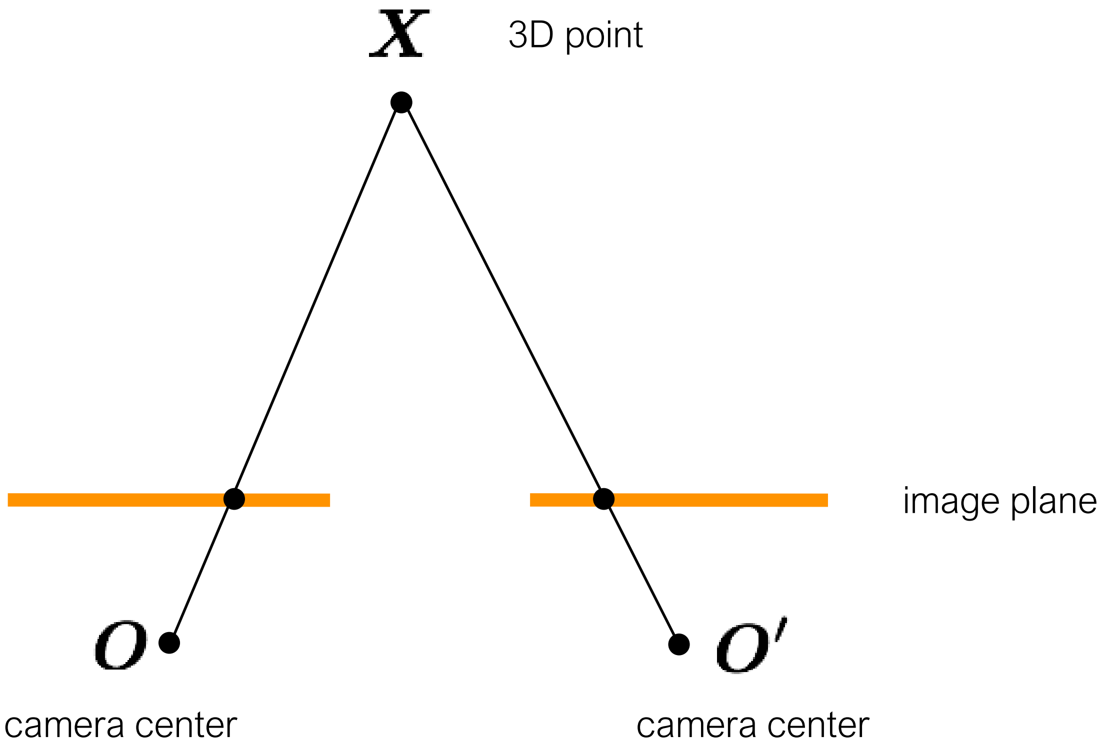




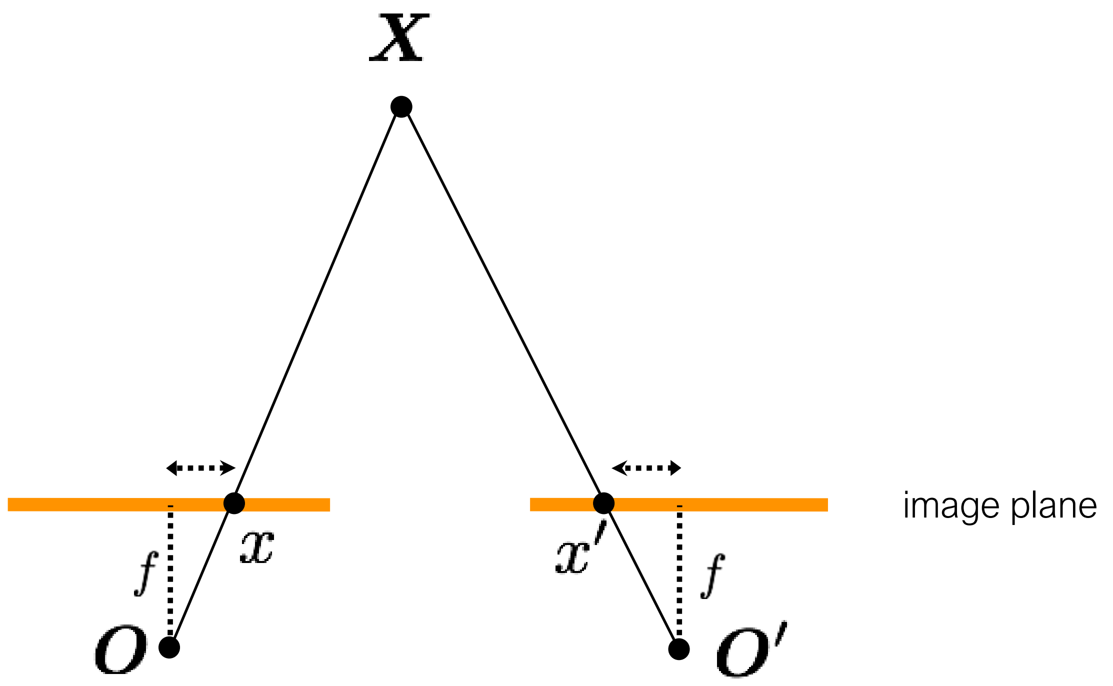
… the distance from the camera.

More formally…

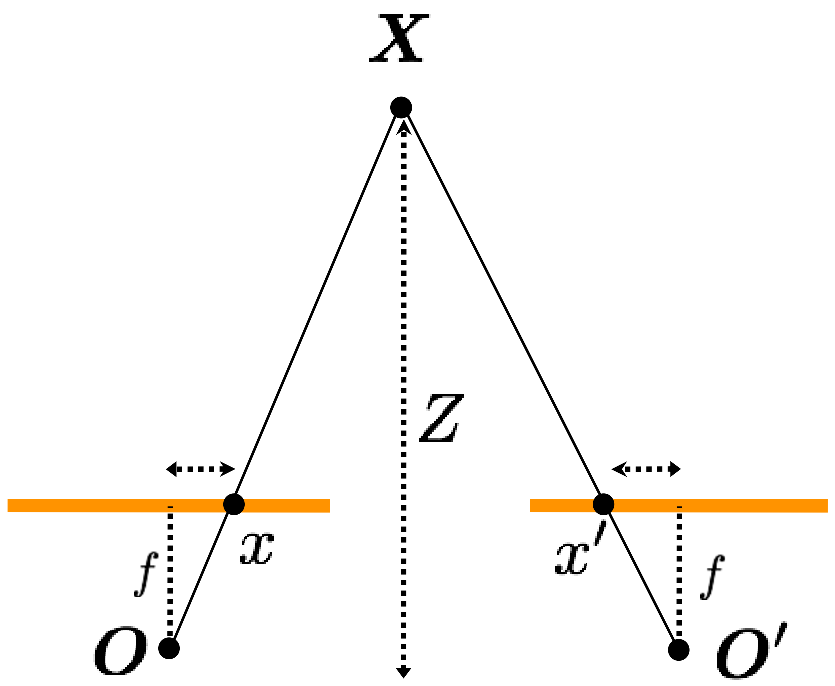
Stereo rectification



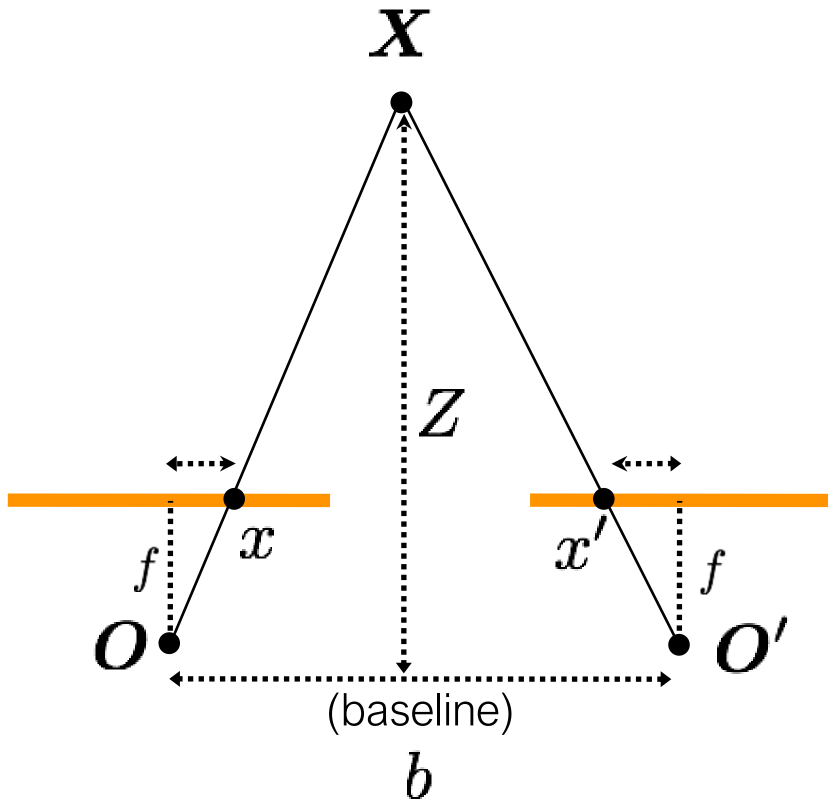
Stereo rectification



Stereo rectification

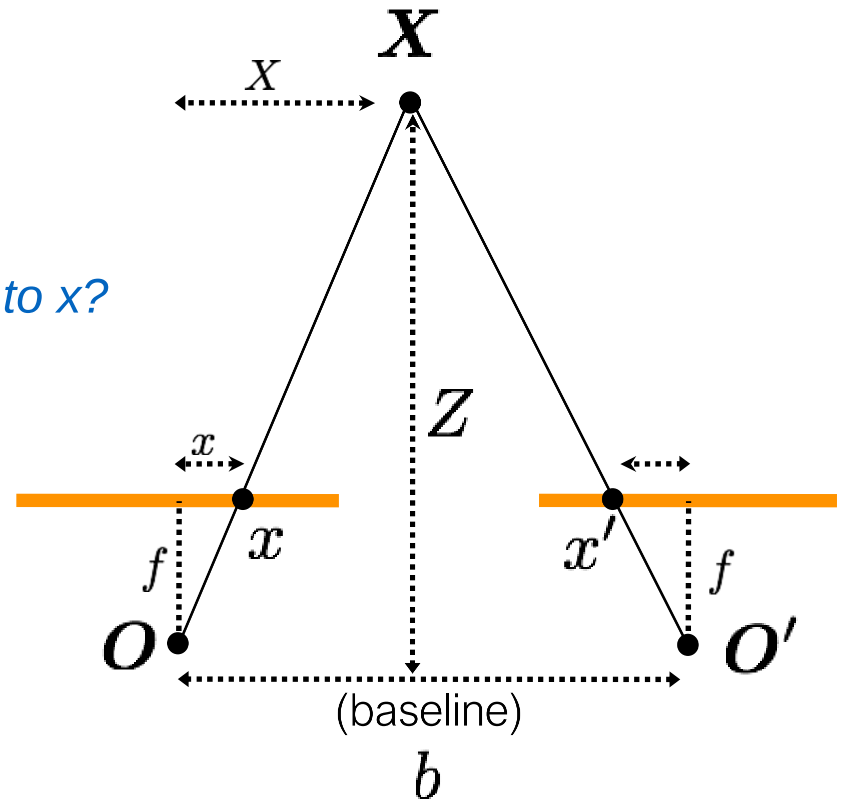


Stereo rectification



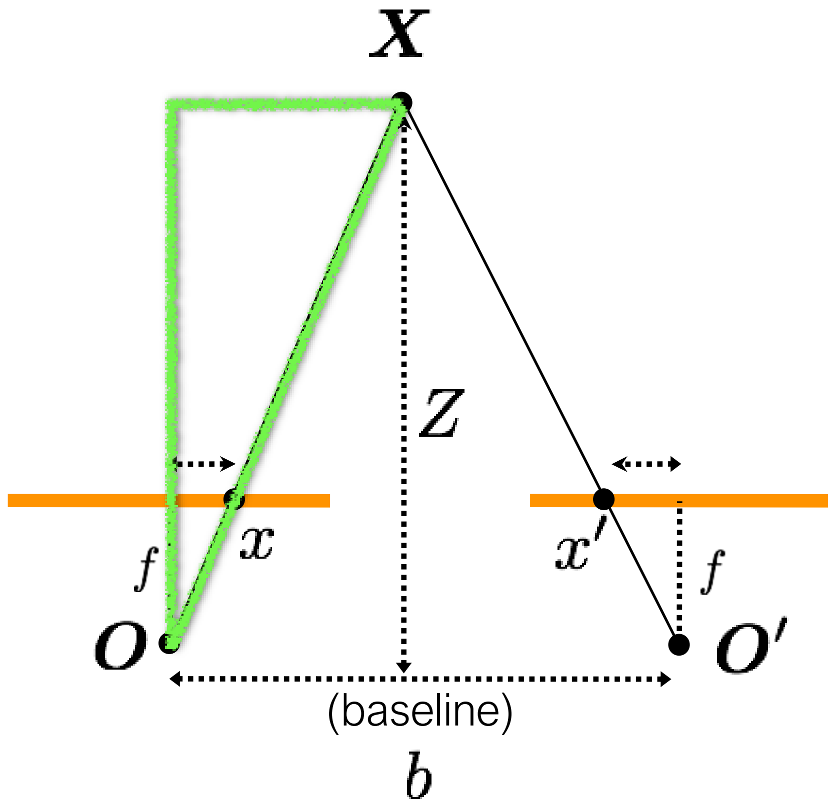
Stereo rectification

How is X related to x?



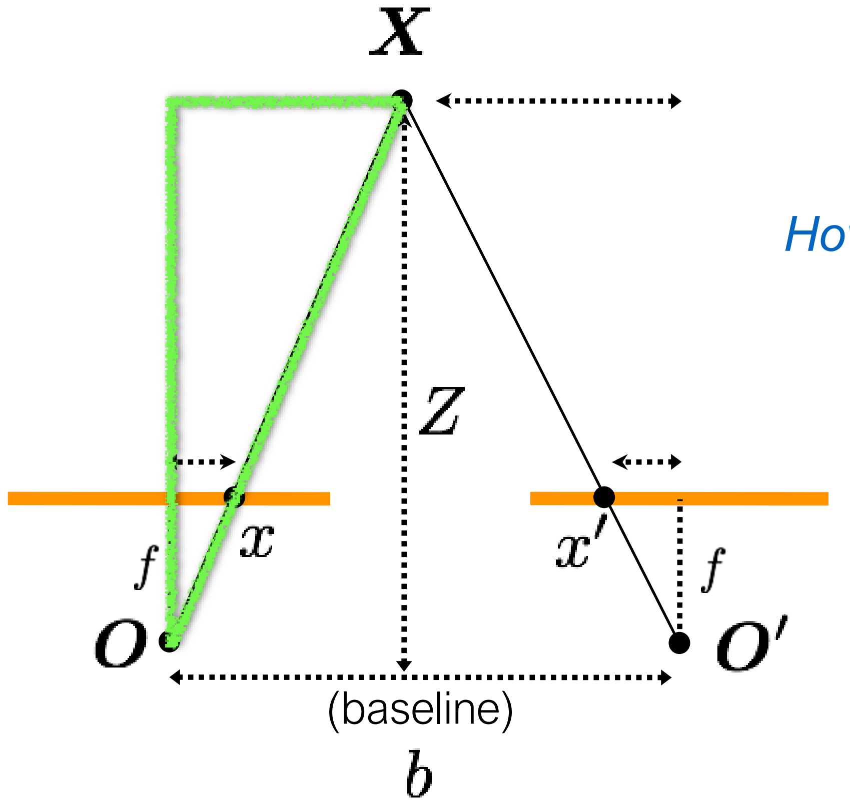
Stereo rectification

$$\frac{X}{Z} = \frac{x}{f}$$



Stereo rectification

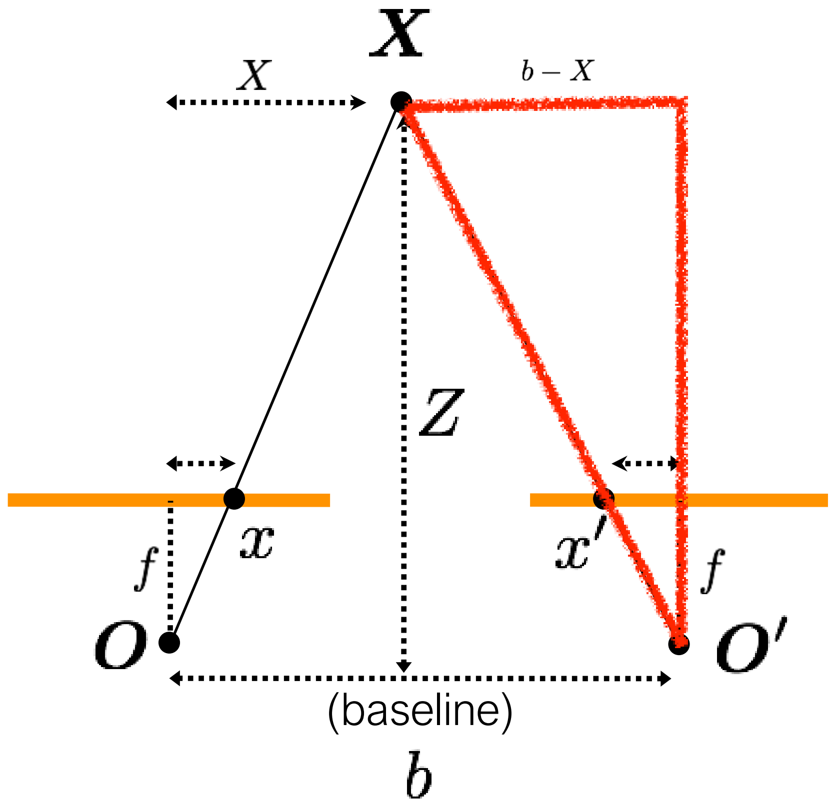
$$\frac{X}{Z} = \frac{x}{f}$$



How is X related to x'?

Stereo rectification

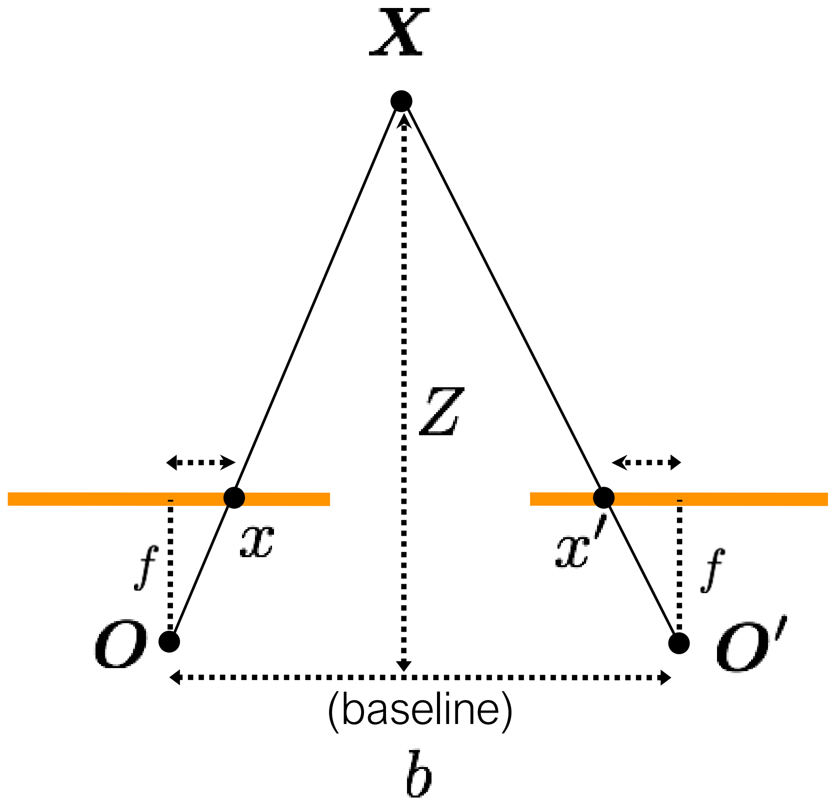
$$\frac{X}{Z} = \frac{x}{f}$$



$$\frac{b - X}{Z} = \frac{x'}{f}$$

Stereo rectification

$$\frac{X}{Z} = \frac{x}{f}$$



$$\frac{b - X}{Z} = \frac{x'}{f}$$

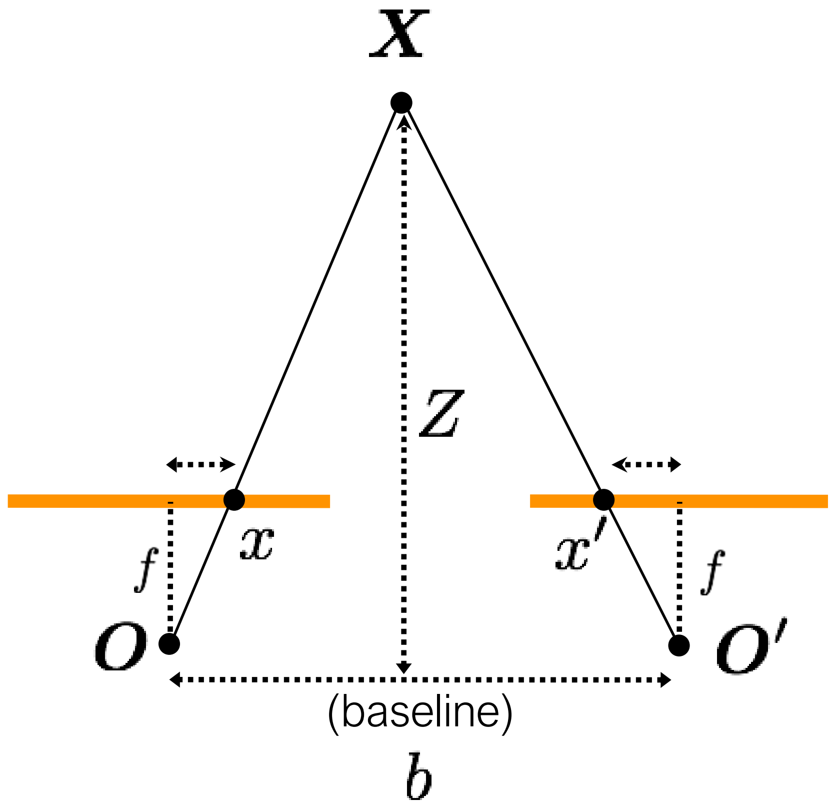
Disparity

$$d = x - x'$$ (wrt to camera origin of image plane)

$$= \frac{bf}{Z}$$

Stereo rectification

$$\frac{X}{Z} = \frac{x}{f}$$



$$\frac{b - X}{Z} = \frac{x'}{f}$$

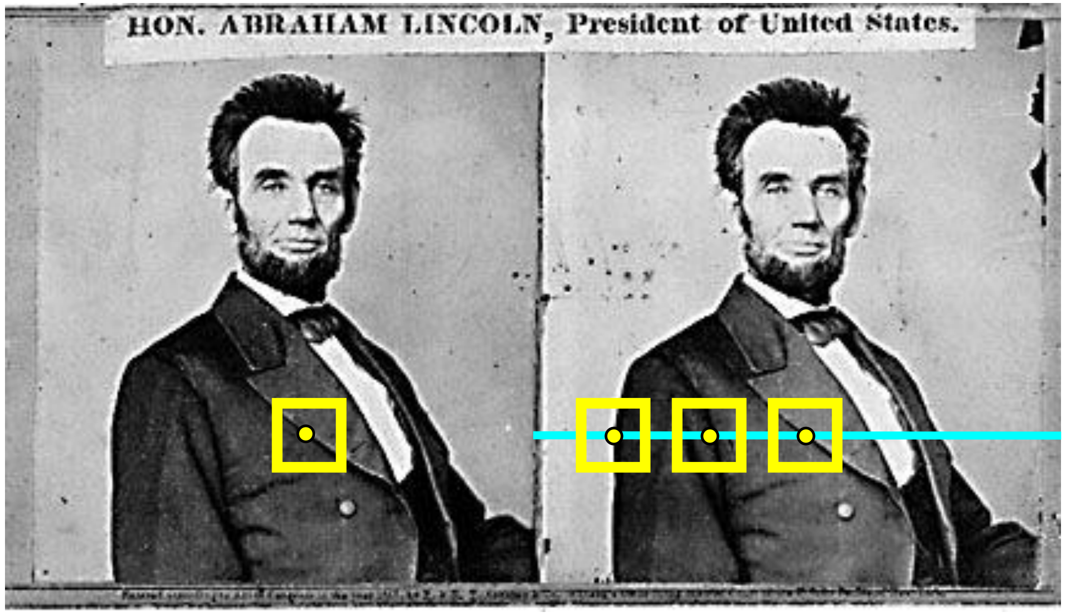
Disparity

$$d = x - x'$$

$$= \frac{bf}{Z}$$

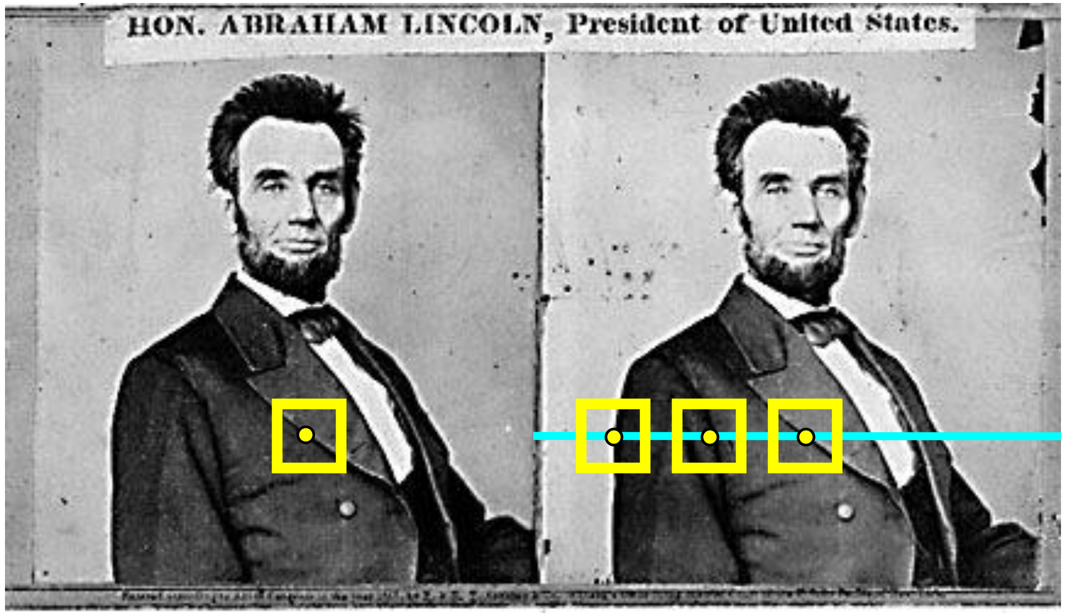
inversely proportional
to depth

Stereo depth sensing



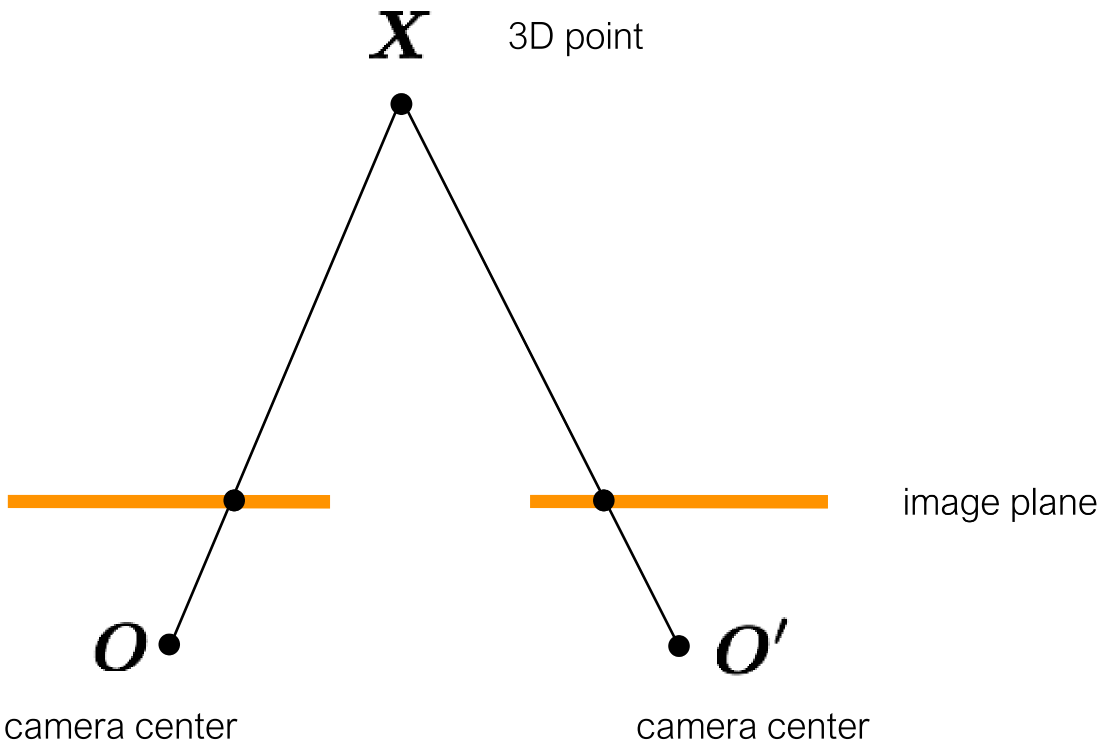
1. Rectify images
   (make epipolar lines horizontal)
2. For each pixel
   a. Find epipolar line
   b. Scan line for best match
   c. Compute depth from disparity $Z = \frac{bf}{d}$

Stereo depth sensing



How can you make the epipolar lines horizontal?

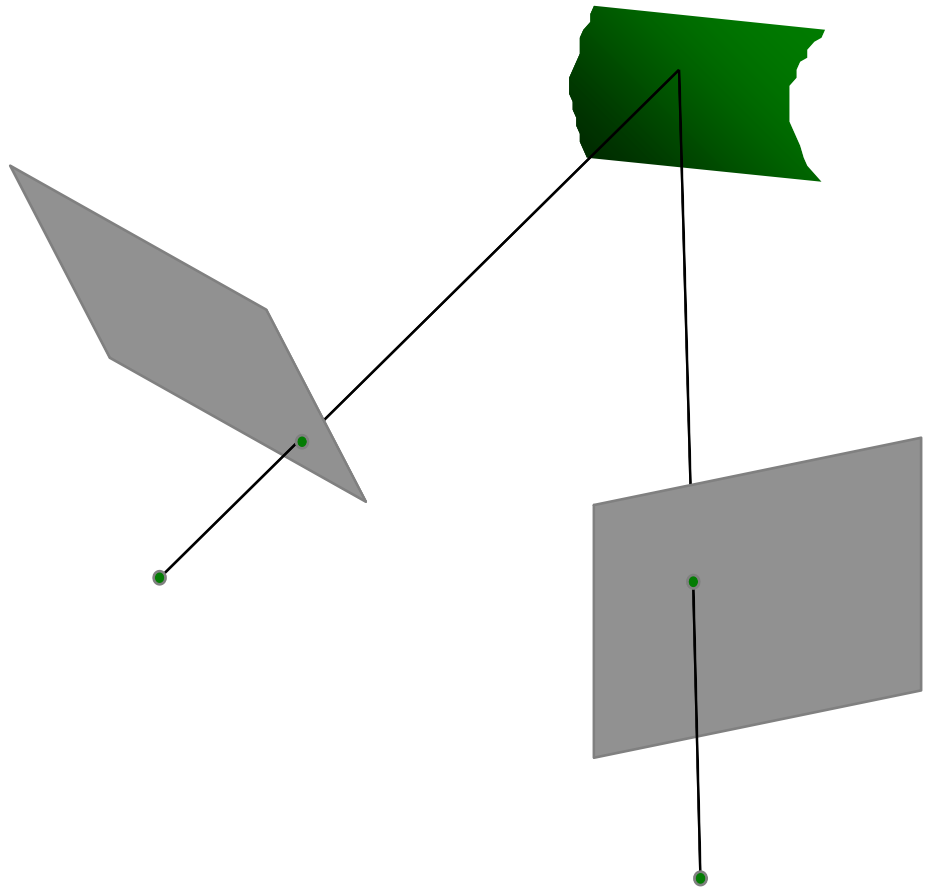
Stereo depth sensing



What's special about these two cameras?

Stereo rectification

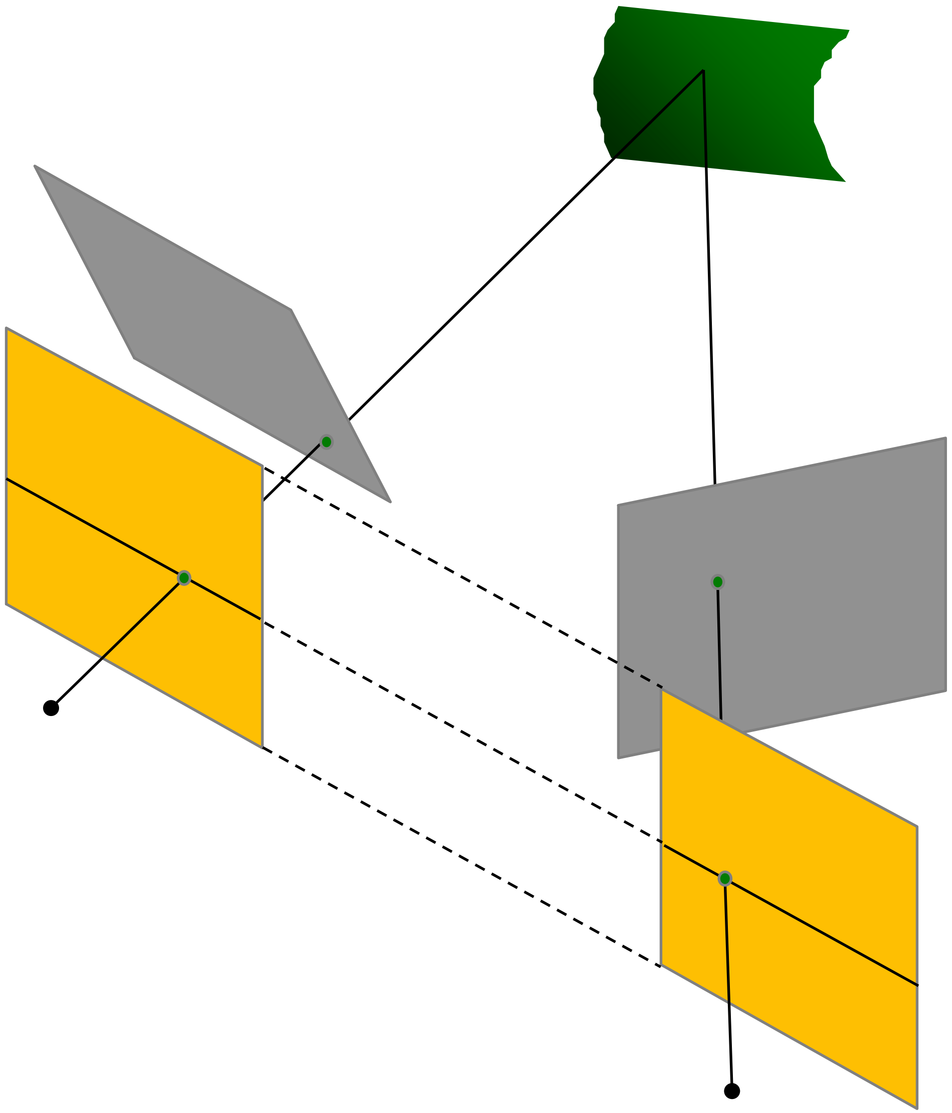
What is stereo rectification?



Stereo rectification

What is stereo rectification?

Reproject image planes onto a common plane parallel to the line between camera centers



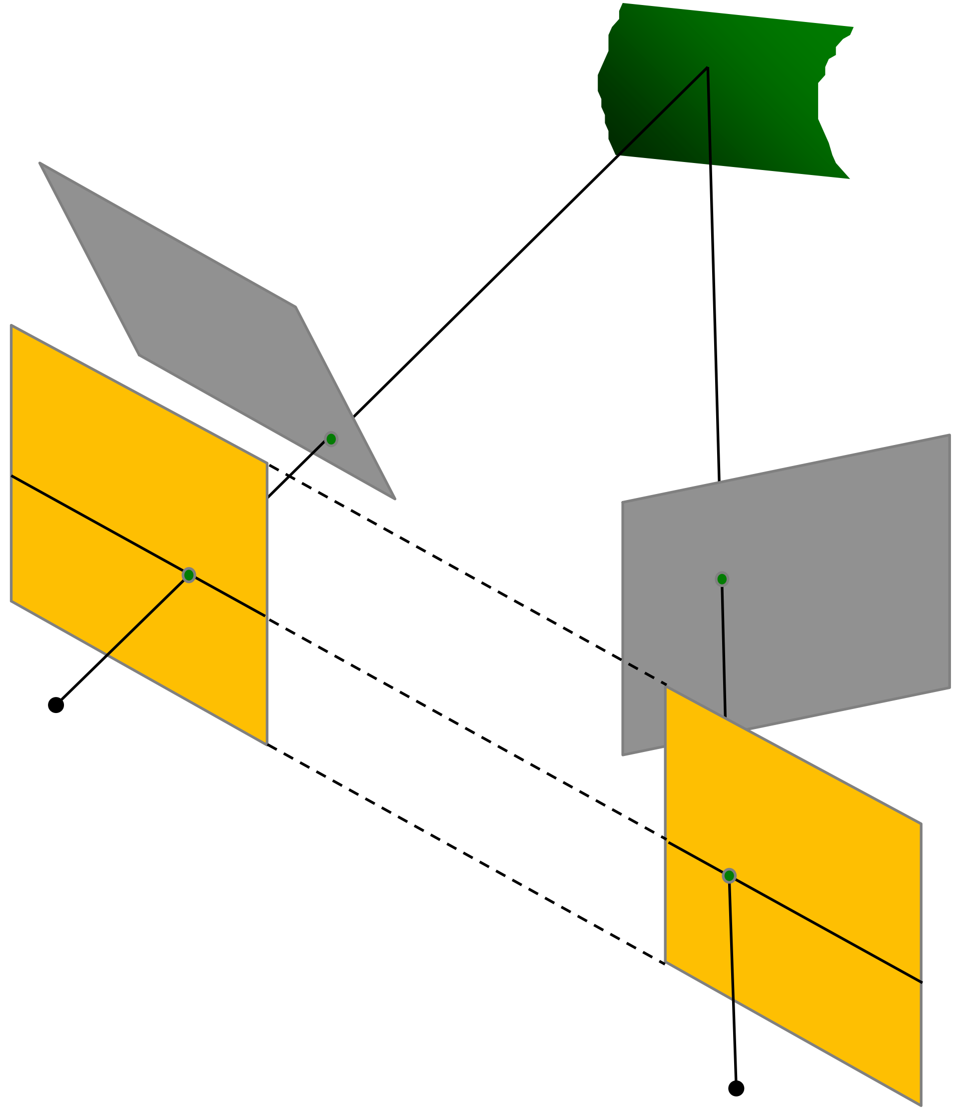
How can you do this?

Stereo rectification

What is stereo rectification?

Reproject image planes onto a common plane parallel to the line between camera centers

Need two homographies (3x3 transform), one for each input image reprojection



C. Loop and Z. Zhang. Computing Rectifying Homographies for Stereo Vision.Computer Vision and Pattern Recognition, 1999.

Stereo Rectification

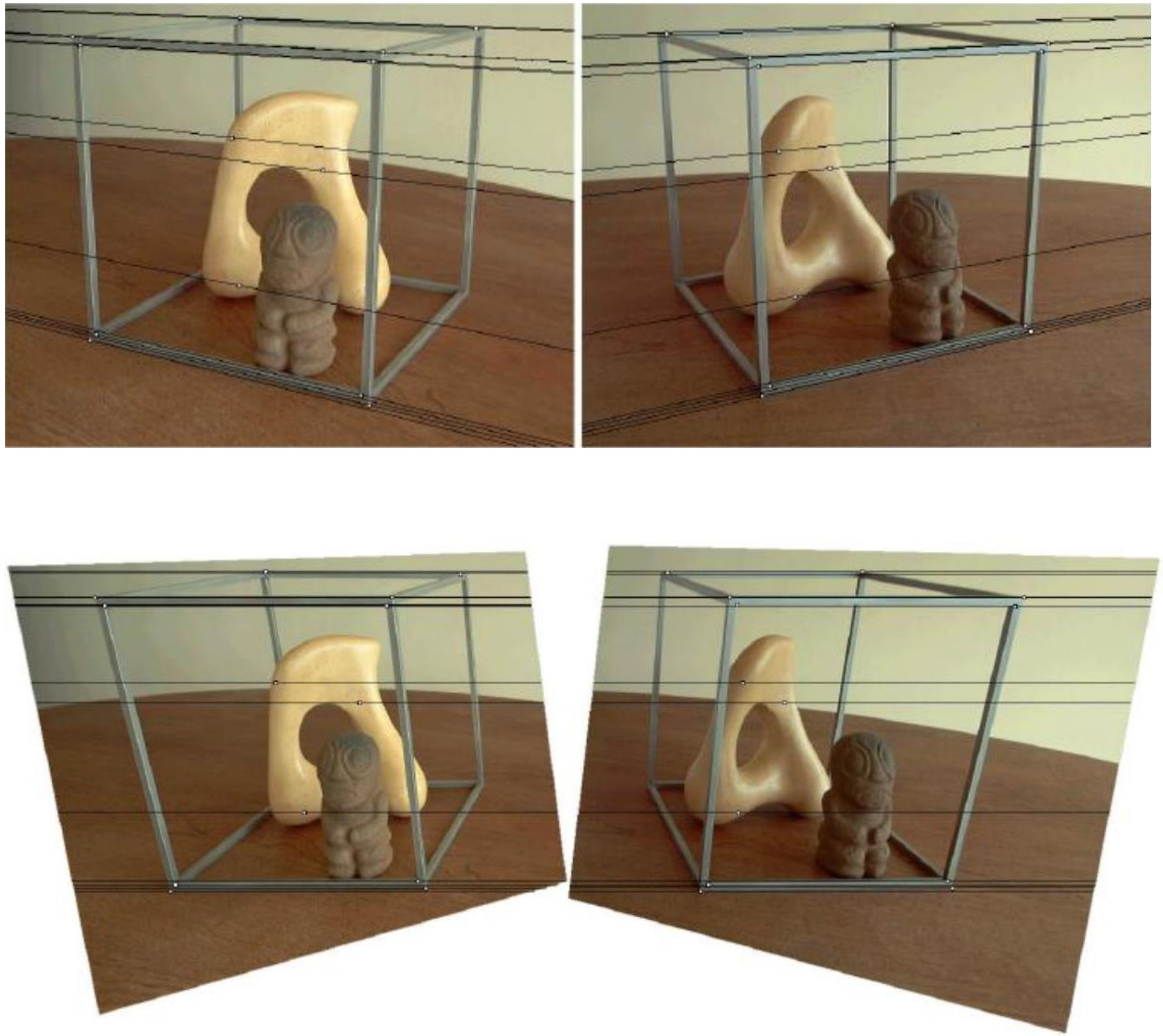
1. **Rotate** the right camera by **R**
   (aligns camera coordinate system orientation only)

2. Rotate (**rectify**) the left camera so that the epipole is at infinity

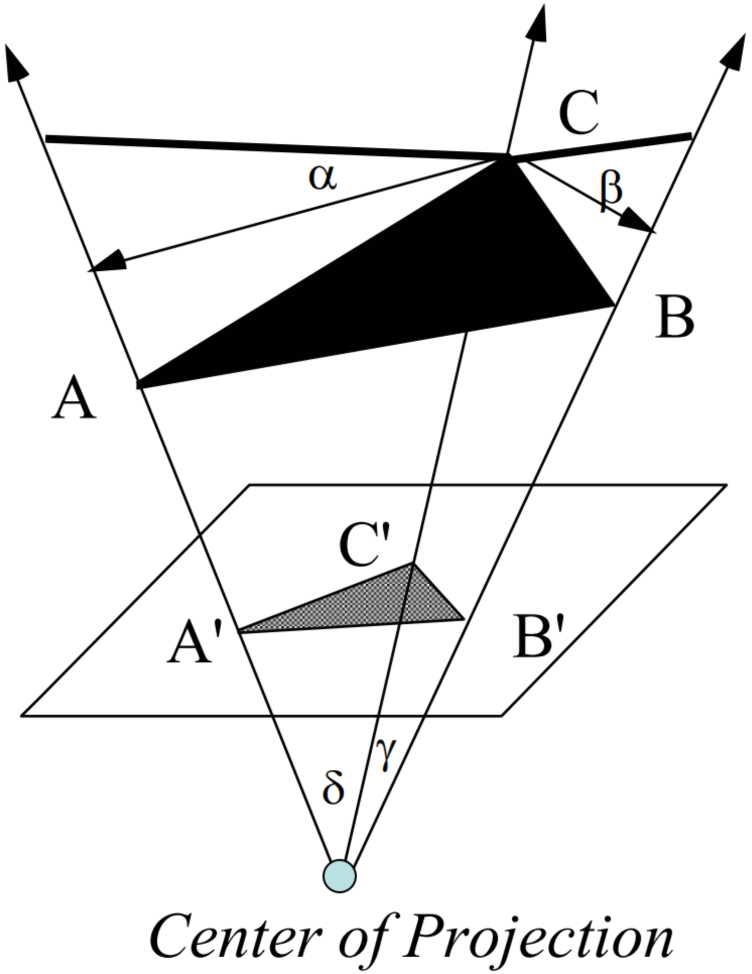
3. Rotate (**rectify**) the right camera so that the epipole is at infinity

4. Adjust the **scale**

What can we do after rectification?

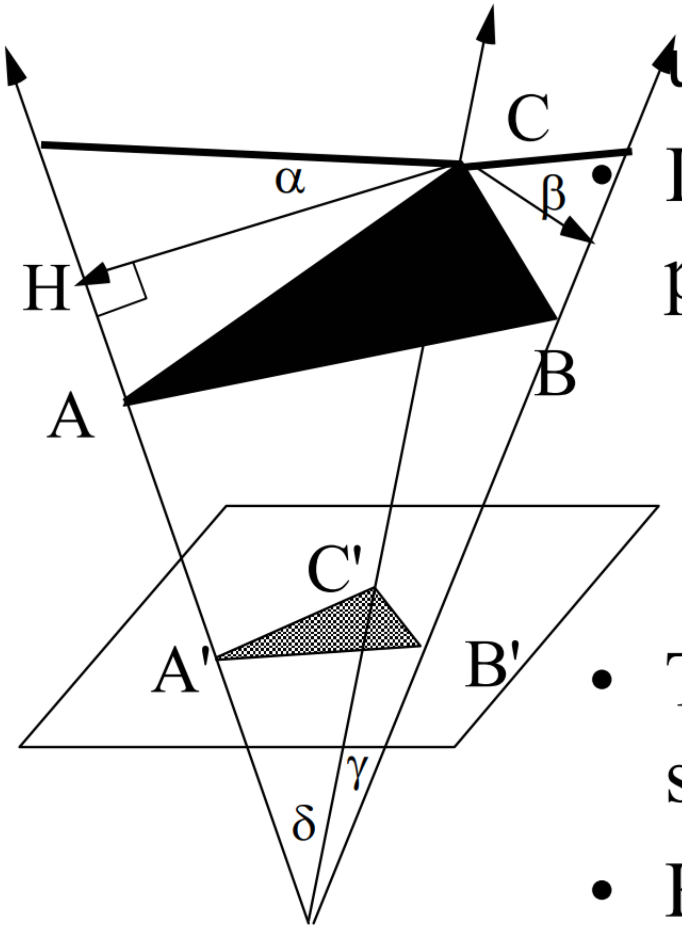


Perspective 3 Point Algorithm



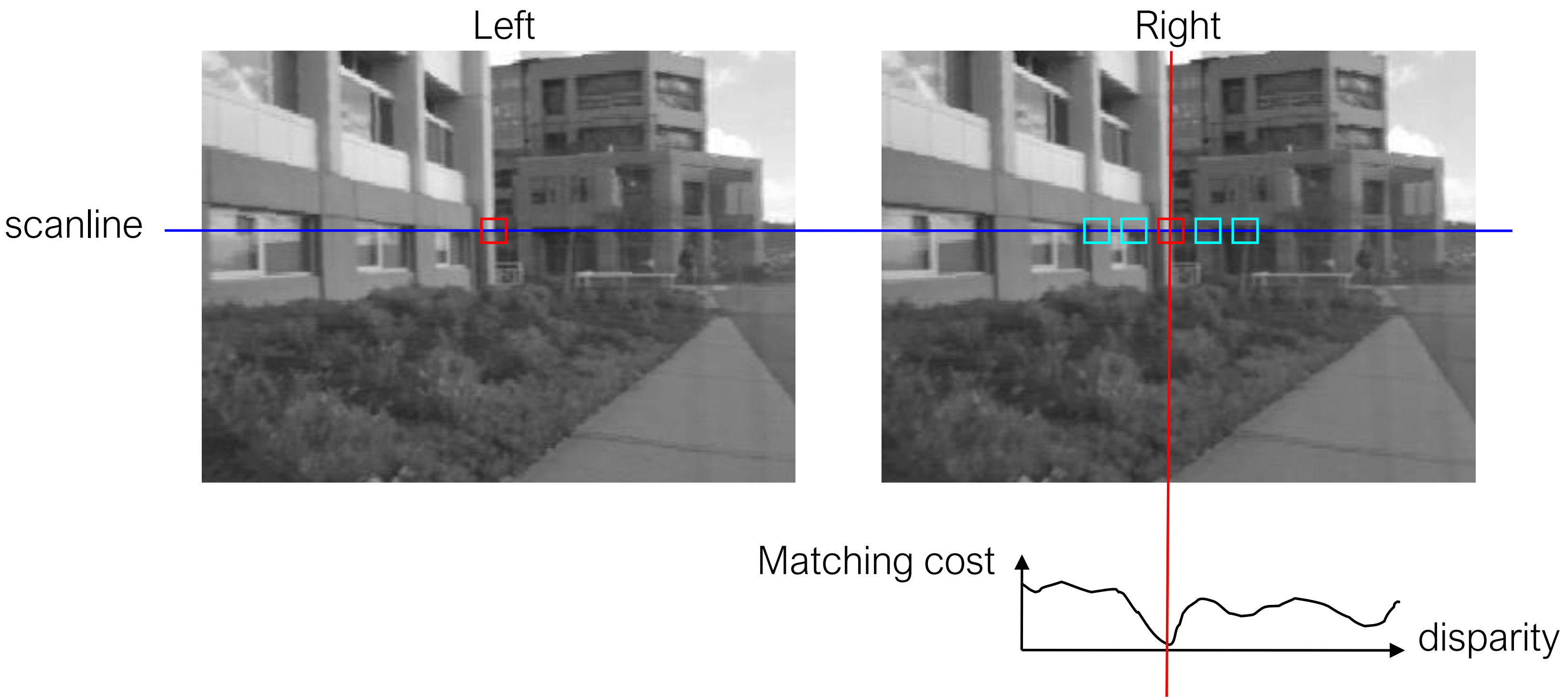
- If distance R_c to C is known, then possible locations of A (and B) can be computed
  - they lie on the intersections of the line of sight through A' and the sphere of radius AC centered at C
  - Once A and B are located, their distance can be computed and compared against the actual distance AB

Perspective 3 Point Algorithm



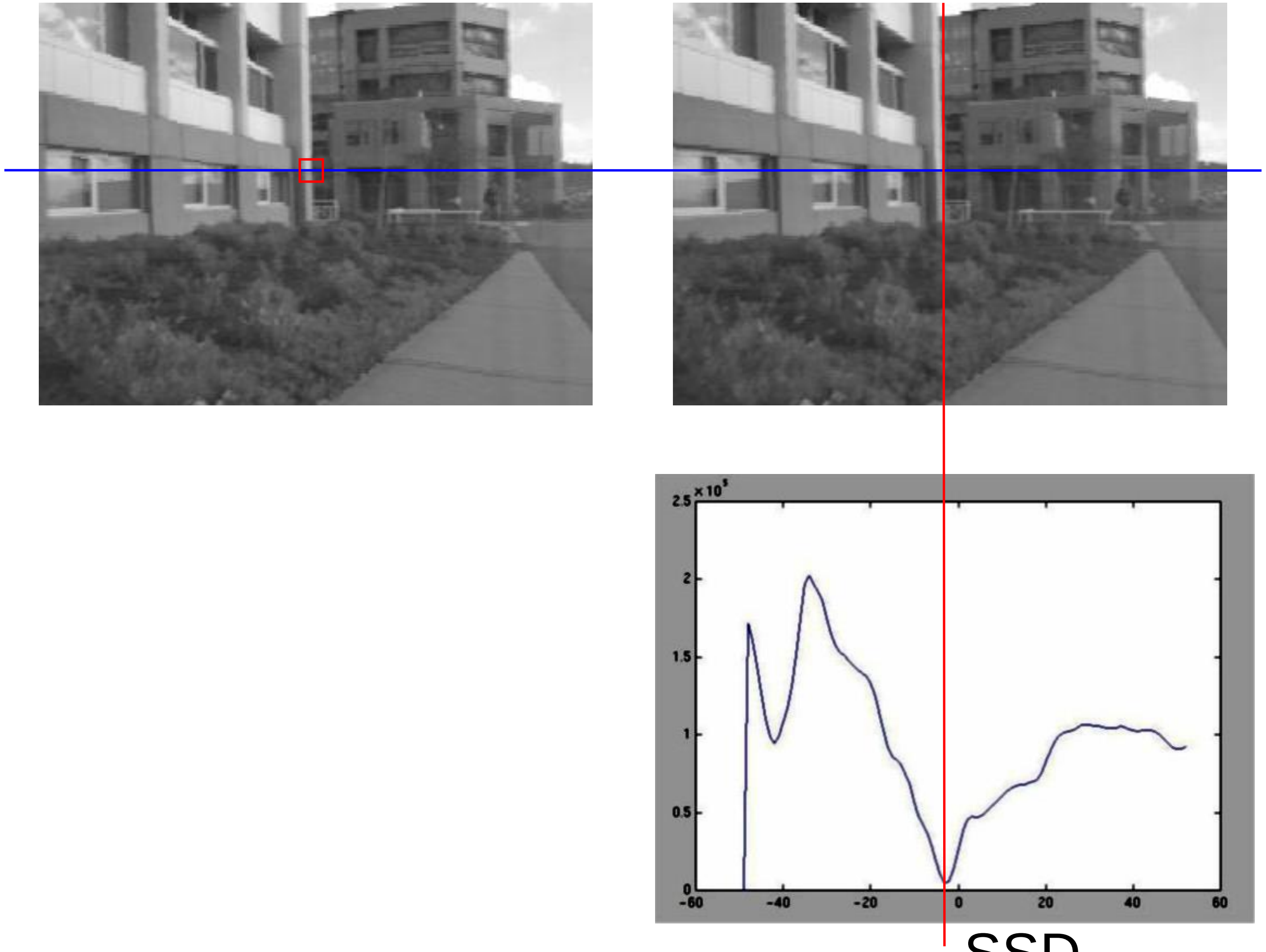
- Not practical to search on R_c since it is unbounded
- Instead, search on one angular pose parameter, α.
  - $R_c = AC \cos\alpha / \sin\delta$
  - $R_a = R_c \cos\delta \pm AC \sin\alpha$
  - $R_b = R_c \cos\gamma \pm [(BC^2 - (RC \sin\gamma)^2]^{1/2}$
- This results in four possible lengths for side AB
- Keep poses with the right AB length

Stereo Block Matching

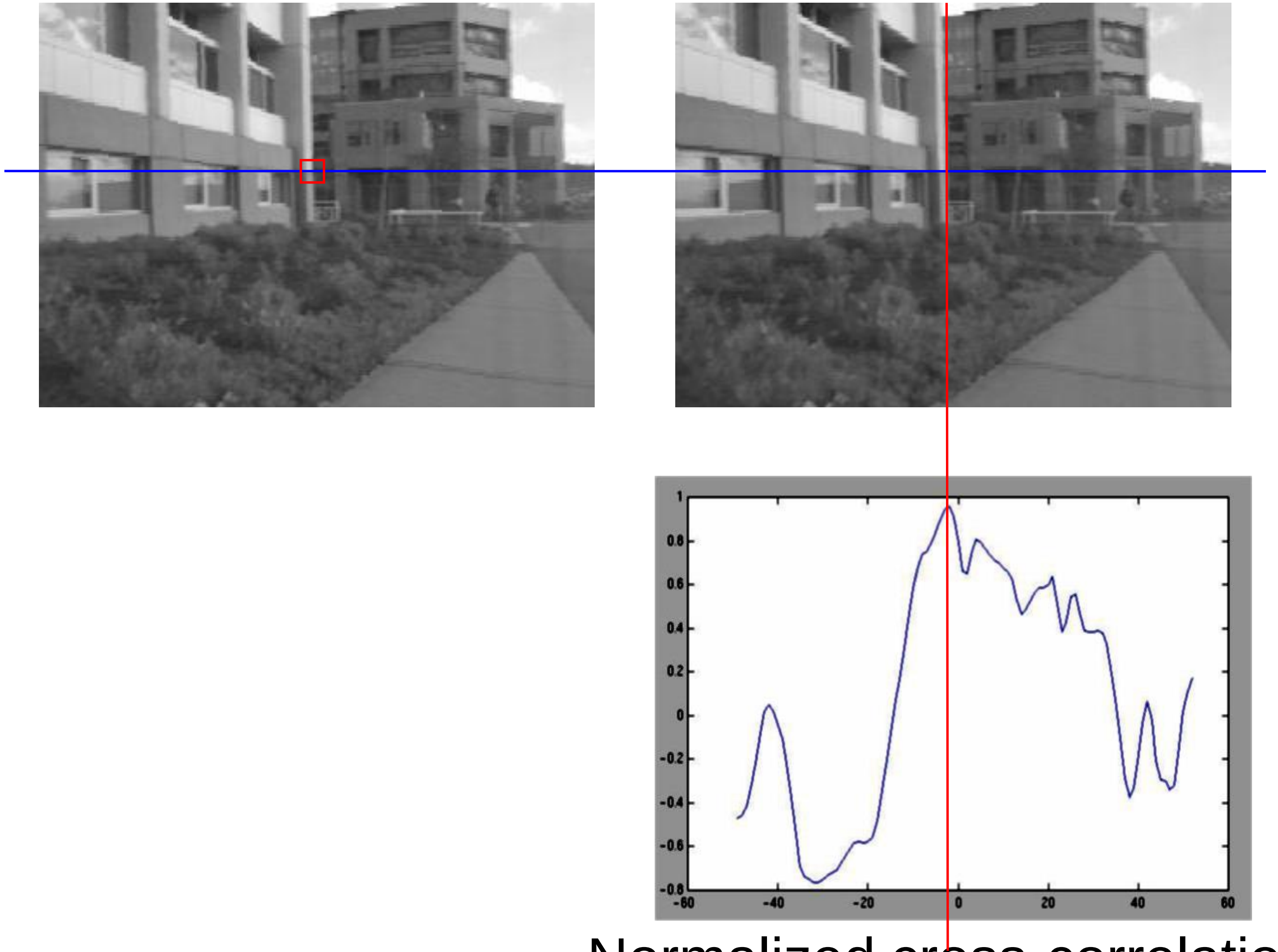


- Slide a window along the epipolar line and compare contents of th at window with the reference window in the left image
- Matching cost: SSD or normalized correlation

Stereo Block Matching



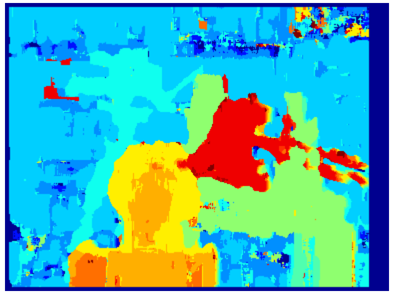
Stereo Block Matching

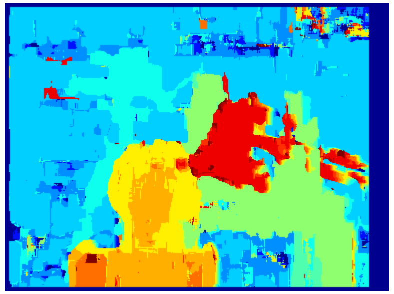


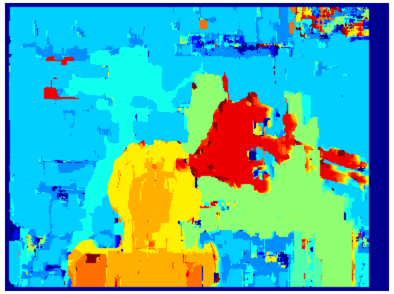
Normalized cross-correlation

Stereo Block Matching

Similarity Measure	Formula
Sum of Absolute Differences (SAD)	$\sum_{(i,j)\in W} \lvert I_1(i,j) - I_2(x+i, y+j) \rvert$
Sum of Squared Differences (SSD)	$\sum_{(i,j)\in W} \left(I_1(i,j) - I_2(x+i, y+j)\right)^2$
Zero-mean SAD	$\sum_{(i,j)\in W} \lvert I_1(i,j) - \bar{I}_1(i,j) - I_2(x+i,y+j) + \bar{I}_2(x+i,y+j) \rvert$
Locally scaled SAD	$\sum_{(i,j)\in W} \lvert I_1(i,j) - \frac{\bar{I}_1(i,j)}{\bar{I}_2(x+i,y+j)} I_2(x+i,y+j) \rvert$
Normalized Cross Correlation (NCC)	$\frac{\sum_{(i,j)\in W} I_1(i,j).I_2(x+i,y+j)}{\sqrt[2]{\sum_{(i,j)\in W} I_1^2(i,j).\sum_{(i,j)\in W} I_2^2(x+i,y+j)}}$


SAD

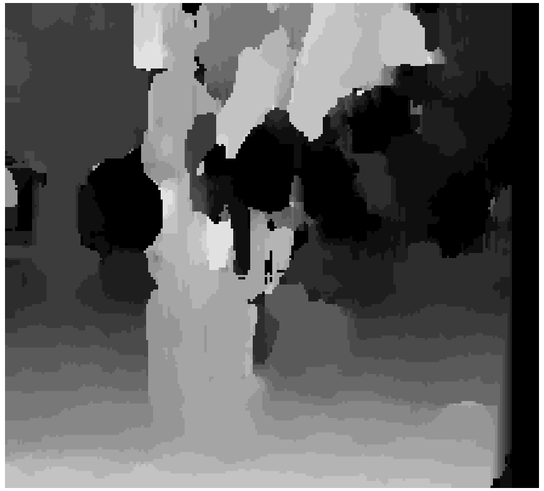

SSD


NCC


Ground truth

Effect of window size

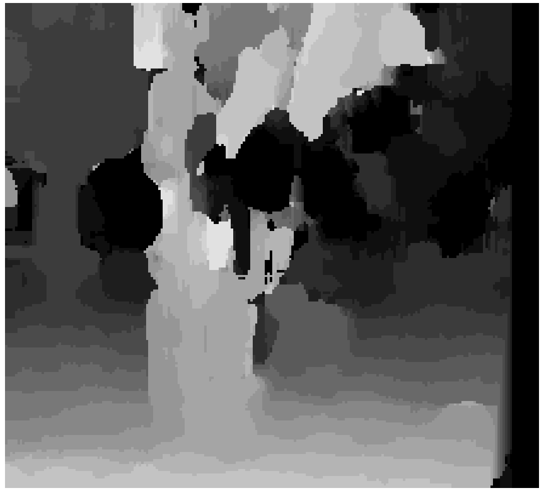
W = 3



W = 20

Effect of window size

W = 3



W = 20

Smaller window

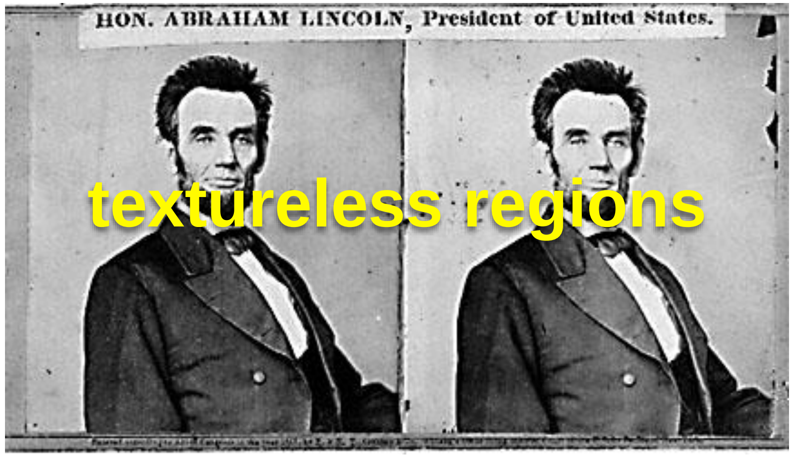
+ More detail
- More noise

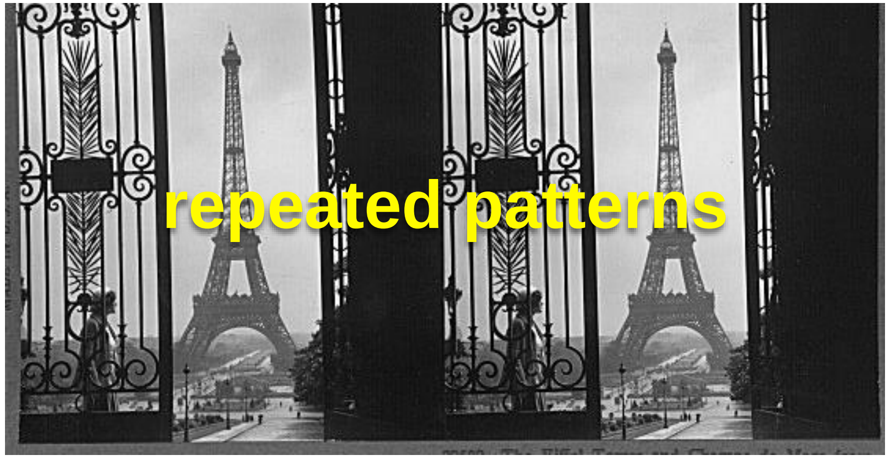
Larger window

+ Smoother disparity maps
- Less detail
- Fails near boundaries

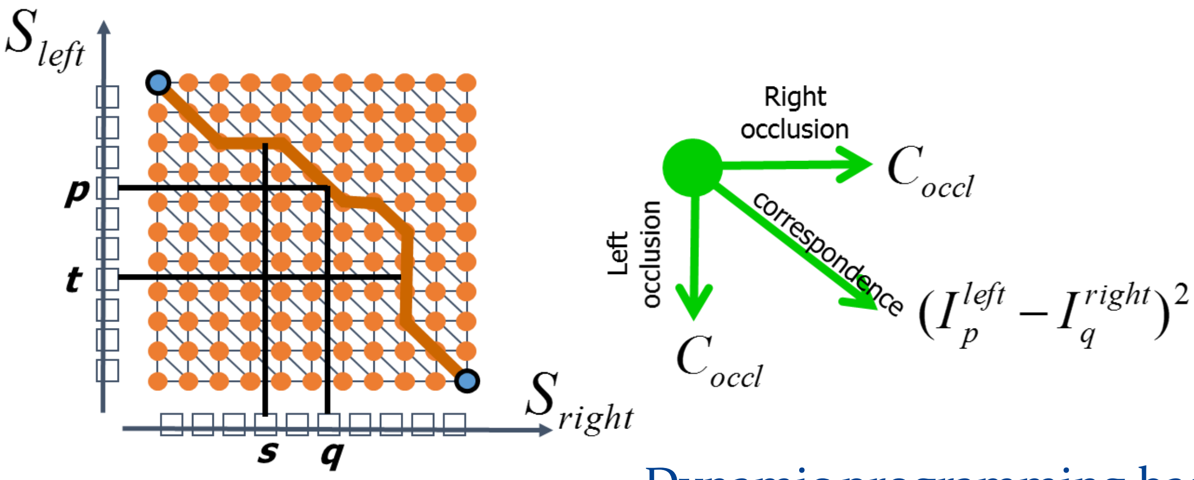
Stereo Matching

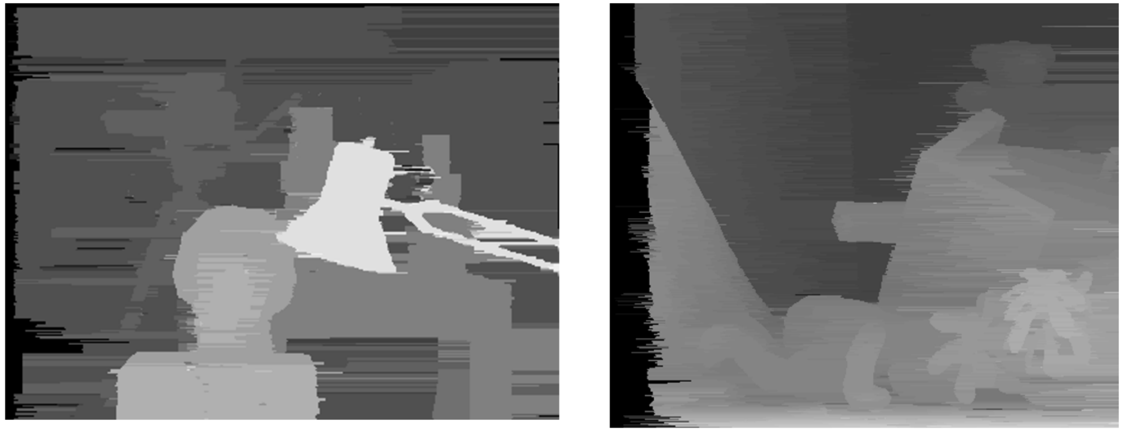
When will stereo block matching fail?





Semi-Global Matching





Dynamic programming based Stereo matching

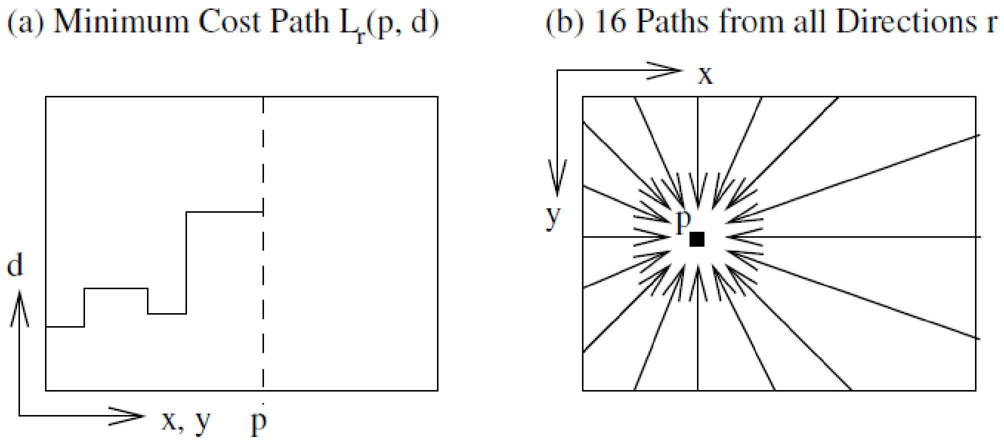
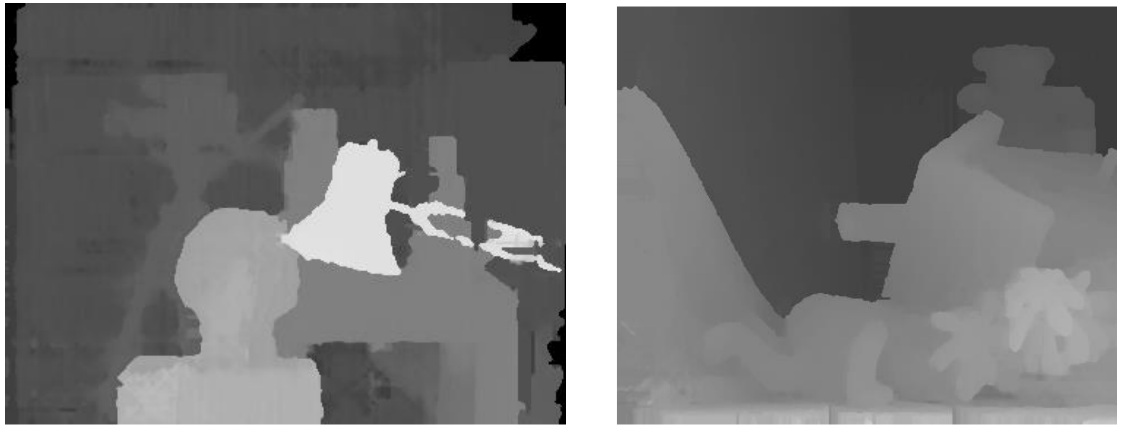


Figure 1. Aggregation of costs.



3D triangulation

```python
import cv2
import numpy as np

P1 = np.eye(3, 4, dtype=np.float32)
P2 = np.eye(3, 4, dtype=np.float32)
P2[0, 3] = -1

N = 5
points3d = np.empty((4, N), np.float32)
points3d[:3, :] = np.random.randn(3, N)
points3d[3, :] = 1

points1 = P1 @ points3d
points1 = points1[:2, :] / points1[2, :]
points1[:2, :] += np.random.randn(2, N) * 1e-2

points2 = P2 @ points3d
points2 = points2[:2, :] / points2[2, :]
points2[:2, :] += np.random.randn(2, N) * 1e-2

points3d_reconstr = cv2.triangulatePoints(P1, P2, points1, points2)
points3d_reconstr /= points3d_reconstr[3, :]

print('Original points')
print(points3d[:3].T)
print('Reconstructed points')
print(points3d_reconstr[:3].T)
```

Find relative camera-object pose using PnP algorithm

```
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)

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

ret, rvec, tvec = cv2.solvePnP(pattern_points, corners, camera_matrix, dist_coefs,
                               None, None, False, cv2.SOLVEPNP_ITERATIVE)

img_points, _ = cv2.projectPoints(pattern_points, rvec, tvec, camera_matrix, dist_coefs)

for c in img_points.squeeze():
    cv2.circle(img, tuple(c), 10, (0, 255, 0), 2)

cv2.imshow('points', img)
cv2.waitKey()

cv2.destroyAllWindows()
```

Stereo rectification

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
matplotlib.rcParams.update({'font.size': 20})
np_load_old = np.load
np.load = lambda *a,**k: np_load_old(*a, allow_pickle=True, **k)

data = np.load('../data/stereo/case1/stereo.npy').item()
Kl, Dl, Kr, Dr, R, T, img_size = data['Kl'], data['Dl'], data['Kr'], data['Dr'], \
                                 data['R'], data['T'], data['img_size']

left_img = cv2.imread('../data/stereo/case1/left14.png')
right_img = cv2.imread('../data/stereo/case1/right14.png')

R1, R2, P1, P2, Q, validRoi1, validRoi2 = cv2.stereoRectify(Kl, Dl, Kr, Dr, img_size, R, T)

xmap1, ymap1 = cv2.initUndistortRectifyMap(Kl, Dl, R1, Kl, img_size, cv2.CV_32FC1)
xmap2, ymap2 = cv2.initUndistortRectifyMap(Kr, Dr, R2, Kr, img_size, cv2.CV_32FC1)

left_img_rectified = cv2.remap(left_img, xmap1, ymap1, cv2.INTER_LINEAR)
right_img_rectified = cv2.remap(right_img, xmap2, ymap2, cv2.INTER_LINEAR)
```

```python
plt.figure(0, figsize=(12,10))
plt.subplot(221)
plt.title('left original')
plt.imshow(left_img, cmap='gray')
plt.subplot(222)
plt.title('right original')
plt.imshow(right_img, cmap='gray')
plt.subplot(223)
plt.title('left rectified')
plt.imshow(left_img_rectified, cmap='gray')
plt.subplot(224)
plt.title('right rectified')
plt.imshow(right_img_rectified, cmap='gray')
plt.tight_layout()
plt.show()
```

Fundamental matrix computation

```
import cv2
import numpy as np

np_load_old = np.load
np.load = lambda *a,**k: np_load_old(*a, allow_pickle=True, **k)

data = np.load('../data/stereo/case1/stereo.npy').item()
Kl, Kr, Dl, Dr, left_pts, right_pts, E_from_stereo, F_from_stereo = \
data['Kl'], data['Kr'], data['Dl'], data['Dr'], data['left_pts'], data['right_pts'], data['E'], data['F']

left_pts = np.vstack(left_pts)
right_pts = np.vstack(right_pts)

left_pts = cv2.undistortPoints(left_pts, Kl, Dl, P=Kl)
right_pts = cv2.undistortPoints(right_pts, Kr, Dr, P=Kr)

F, mask = cv2.findFundamentalMat(left_pts, right_pts, cv2.FM_LMEDS)

E = Kr.T @ F @ Kl

print('Fundamental matrix:')
print(F)
print('Essential matrix:')
print(E)
```

Essential decomposition into rotation and translation

```
import cv2
import numpy as np

np_load_old = np.load
np.load = lambda *a,**k: np_load_old(*a, allow_pickle=True, **k)

data = np.load('../data/stereo/case1/stereo.npy').item()
E = data['E']

R1, R2, T = cv2.decomposeEssentialMat(E)

print('Rotation 1:')
print(R1)
print('Rotation 2:')
print(R2)
print('Translation:')
print(T)
```

Estimating disparity map for stereo images

```
import cv2
import matplotlib.pyplot as plt
import matplotlib
matplotlib.rcParams.update({'font.size': 20})

left_img = cv2.imread('../data/stereo/left.png')
right_img = cv2.imread('../data/stereo/right.png')

stereo_bm = cv2.StereoBM_create(32)
dispmap_bm = stereo_bm.compute(cv2.cvtColor(left_img, cv2.COLOR_BGR2GRAY),
                               cv2.cvtColor(right_img, cv2.COLOR_BGR2GRAY))

stereo_sgbm = cv2.StereoSGBM_create(0, 32)
dispmap_sgbm = stereo_sgbm.compute(left_img, right_img)

plt.figure(figsize=(12,10))
plt.subplot(221)
plt.title('left')
plt.imshow(left_img[:,:,[2,1,0]])
plt.subplot(222)
plt.title('right')
plt.imshow(right_img[:,:,[2,1,0]])
plt.subplot(223)
plt.title('BM')
plt.imshow(dispmap_bm, cmap='gray')
plt.subplot(224)
plt.title('SGBM')
plt.imshow(dispmap_sgbm, cmap='gray')
plt.show()
```