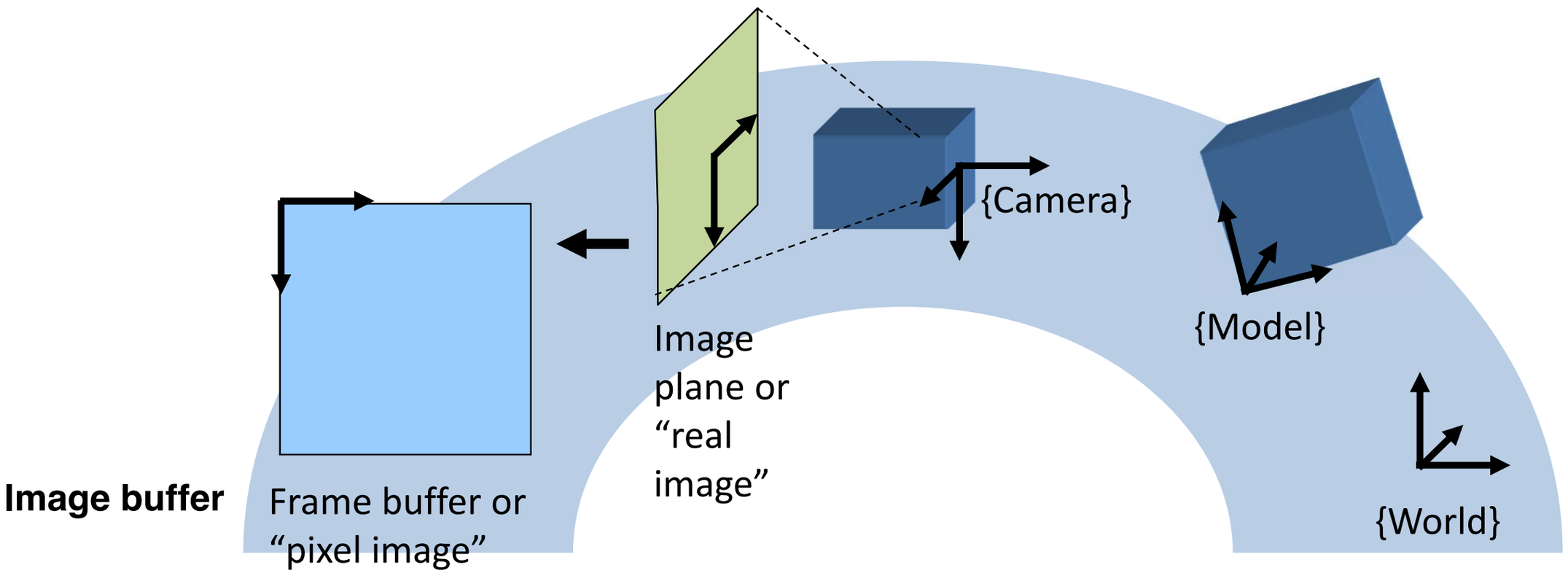
Last time..

Frames of Reference

- Image frames are 2D; others are 3D
- The “pose” (position and orientation) of a 3D rigid body has 6 degrees of freedom



Transformations

«From» frame {B} «to» frame {A}

$$ {}^{A}\mathbf{P} = {}^{A}_{B}\mathbf{R}\ {}^{B}\mathbf{P} + \mathbf{t} $$

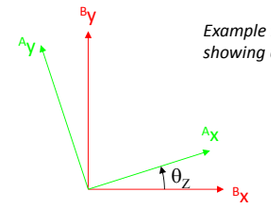
$$ {}^{A}\mathbf{P} = {}^{A}_{B}\mathbf{H}\ {}^{B}\mathbf{P} $$

Two ways to think about a transformation:

- **A transform mapping a point in the «from» frame {B} to its representation in the «to» frame {A}.**
- **A description of the «from» frame {B} relative to the «to» frame {A}**

Note on Direction of Rotation

- The rotation matrix ${}^{B}_{A}\mathbf{R}$ describes the orientation of the {A} frame with respect to the {B} frame
- Example:
  - Consider a rotation about the Z axis
  - We start with the {A} frame aligned with the {B} frame
  - Then rotate about Z until you get to the desired orientation



Example rotation about the Z axis, showing a rotation of +20 degrees

$$ \begin{pmatrix} {}^{B}x \\ {}^{B}y \\ {}^{B}z \end{pmatrix} = \begin{pmatrix} \cos 20° & -\sin 20° & 0 \\ \sin 20° & \cos 20° & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} {}^{A}x \\ {}^{A}y \\ {}^{A}z \end{pmatrix} $$

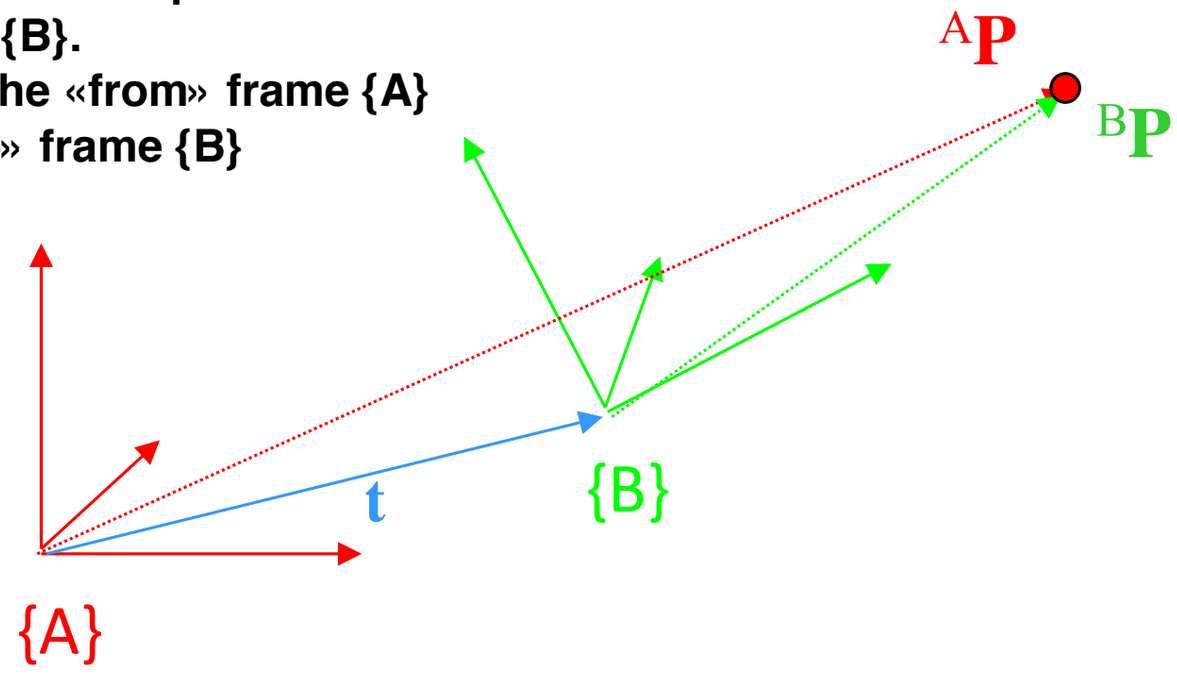
«From» frame {A} «to» frame {B}

$$ {}^{B}\mathbf{P} = {}^{B}_{A}\mathbf{R}\ {}^{A}\mathbf{P} + {}^{B}\mathbf{t}_{Aorg} $$

$$ {}^{B}\mathbf{P} = {}^{B}_{A}\mathbf{H}\ {}^{A}\mathbf{P} $$

Two ways to think about a transformation:

- **A transform mapping a point in the «from» frame {A} to its representation in the «to» frame {B}.**
- **A description of the «from» frame {A} relative to the «to» frame {B}**



Complete Perspective Projection

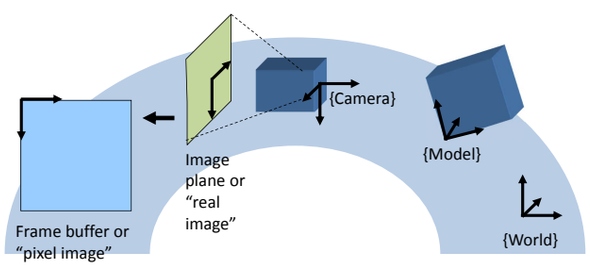
- Projection of a 3D point ${}^{W}\mathbf{P}$ in the world to a point in the pixel image (x_{im}, y_{im})

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \mathbf{K}\,\mathbf{M}_{ext} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}^{W}, \qquad x_{im} = x_1 / x_3, \;\; y_{im} = x_2 / x_3$$

  - where **K** is the intrinsic camera parameter matrix
  - and $\mathbf{M}_{ext}$ is the 3x4 matrix given by

$$\mathbf{M}_{ext} = \left({}^{C}_{W}\mathbf{R} \quad {}^{C}\mathbf{t}_{Worg} \right) = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_X \\ r_{21} & r_{22} & r_{23} & t_Y \\ r_{31} & r_{32} & r_{33} & t_Z \end{pmatrix}$$

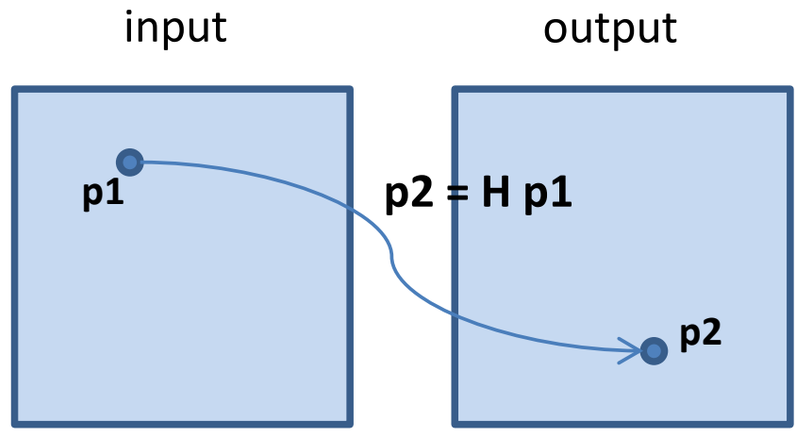
$$\mathbf{K} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$



Generating another image using a transform

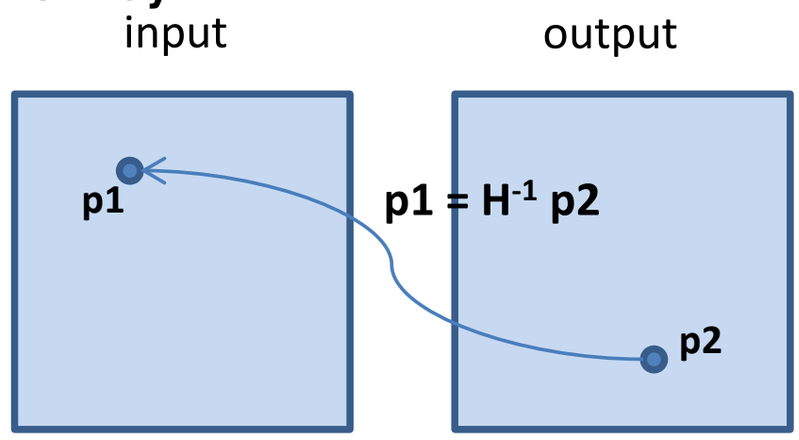
- We know the transformation from input image to output image, **p2 = H p1**
- But instead we use the inverse, **p1 = H^{-1} p2**
- Then we scan through every point **p2** in the output image, and calculate the point **p1** in the input image where we should get the intensity value to use
  - This makes sure that we don't miss assigning any pixels in the output image
  - If p1 falls at a non-integer location, we just take the value at the nearest integer point (a better way to do it is to interpolate among the neighbors)

One way:



for every input pixel -> not filled out

Another way:

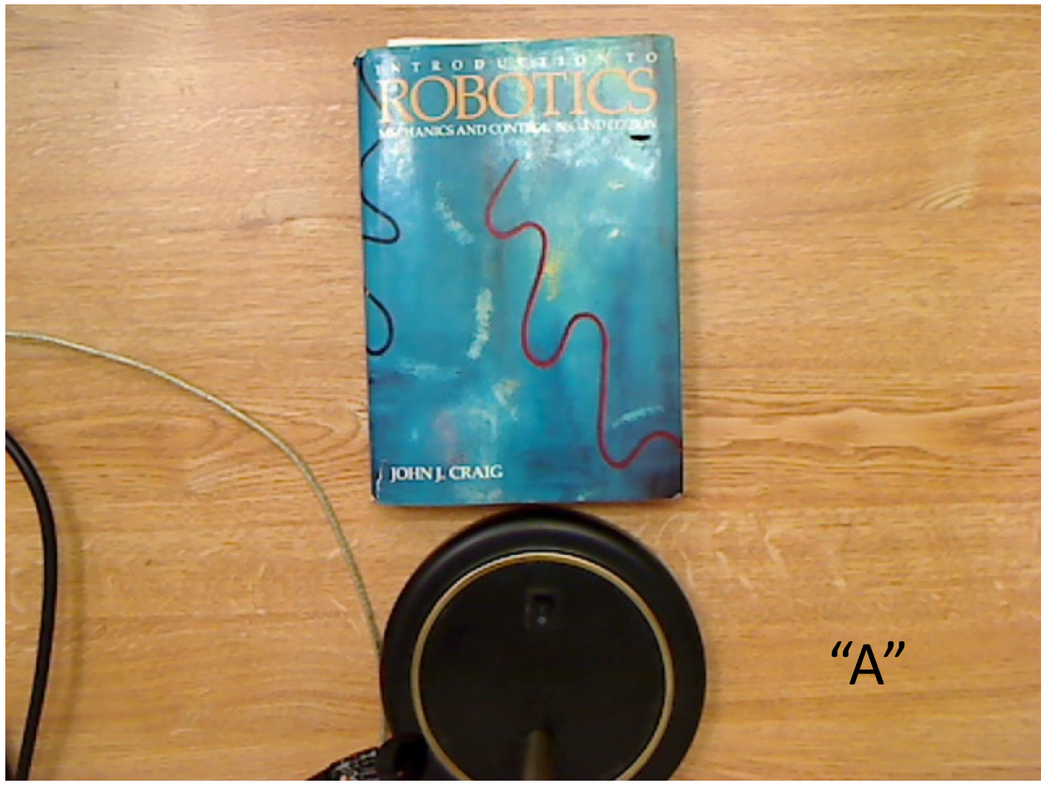


for every output pixel

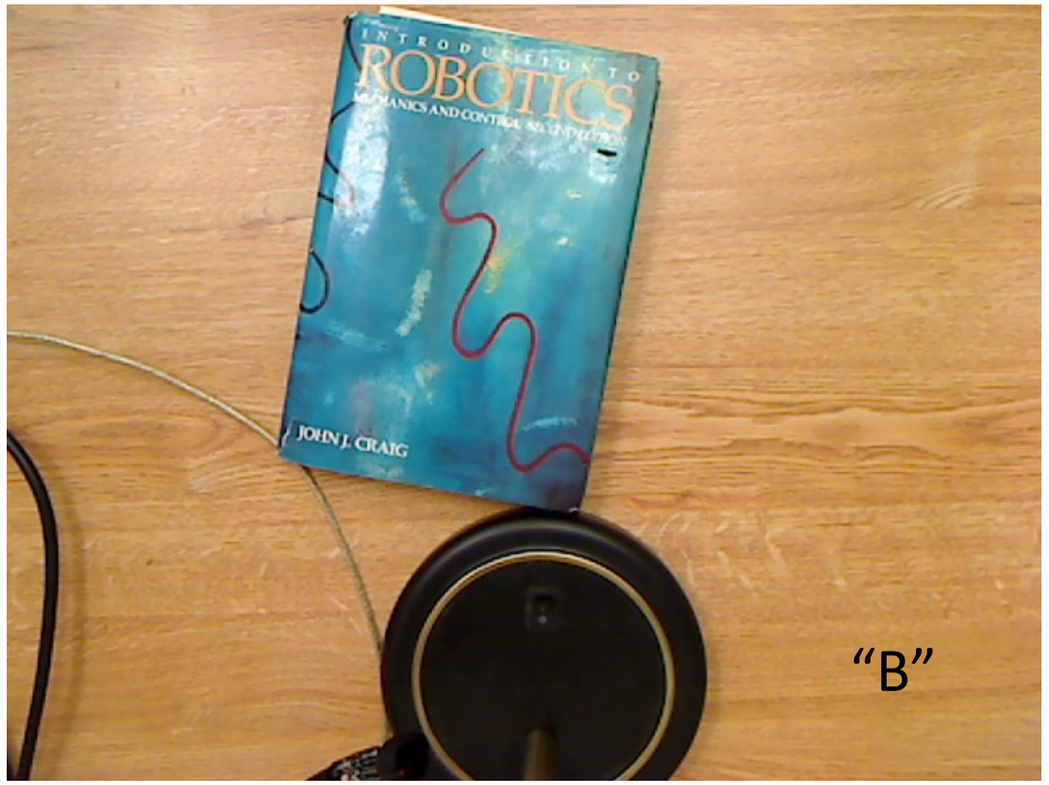
Alignment using **Linear** Least Squares

- If you know a set of point correspondences, you can estimate the parameters of the transform
- Example – find the rotation and translation of the book in the images below

```
% Using imtool, we manually find
% corresponding points (x;y), which are
% the four corners of the book
pA = [
    221 413  416  228;
    31   20  304  308];
pB = [
    214 404  352  169;
    7    34  314  280];
```



"A"



"B"

Example (continued)

- A 2D rigid transform is

$$\begin{pmatrix} x_B \\ y_B \\ 1 \end{pmatrix} = \begin{pmatrix} c & -s & t_x \\ s & c & t_t \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_A \\ y_A \\ 1 \end{pmatrix}, \quad \text{where } c = \cos\theta,\ s = \sin\theta$$

- Or

$$x_B = cx_A - sy_A + t_x$$
$$y_B = sx_A + cy_A + t_y$$

b = Ax

- We put into the form **Ax = b**, where

$$\mathbf{A} = \begin{pmatrix} x_A^{(1)} & -y_A^{(1)} & 1 & 0 \\ y_A^{(1)} & x_A^{(1)} & 0 & 1 \\ \hline \vdots & & & \\ y_A^{(N)} & x_A^{(N)} & 0 & 1 \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} c \\ s \\ t_x \\ t_y \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} x_B^{(1)} \\ y_B^{(1)} \\ \hline \vdots \\ y_B^{(N)} \end{pmatrix}$$

Note: c and s are not really independent variables; however we treat them as independent so that we get a system of linear equations

$$\mathbf{x} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{b}$$

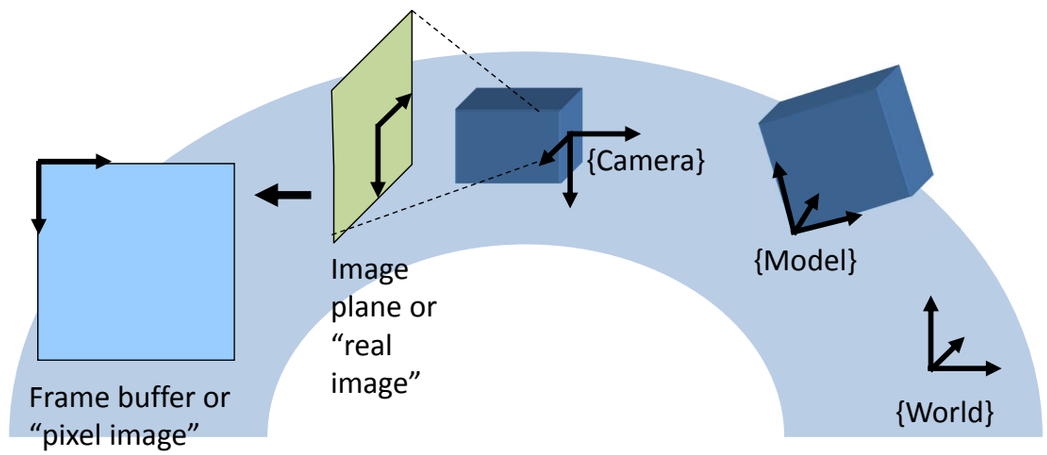
Today:

Calibration, Stereo, Epipolar geometry, Essential and Fundamental matrix

Camera Calibration

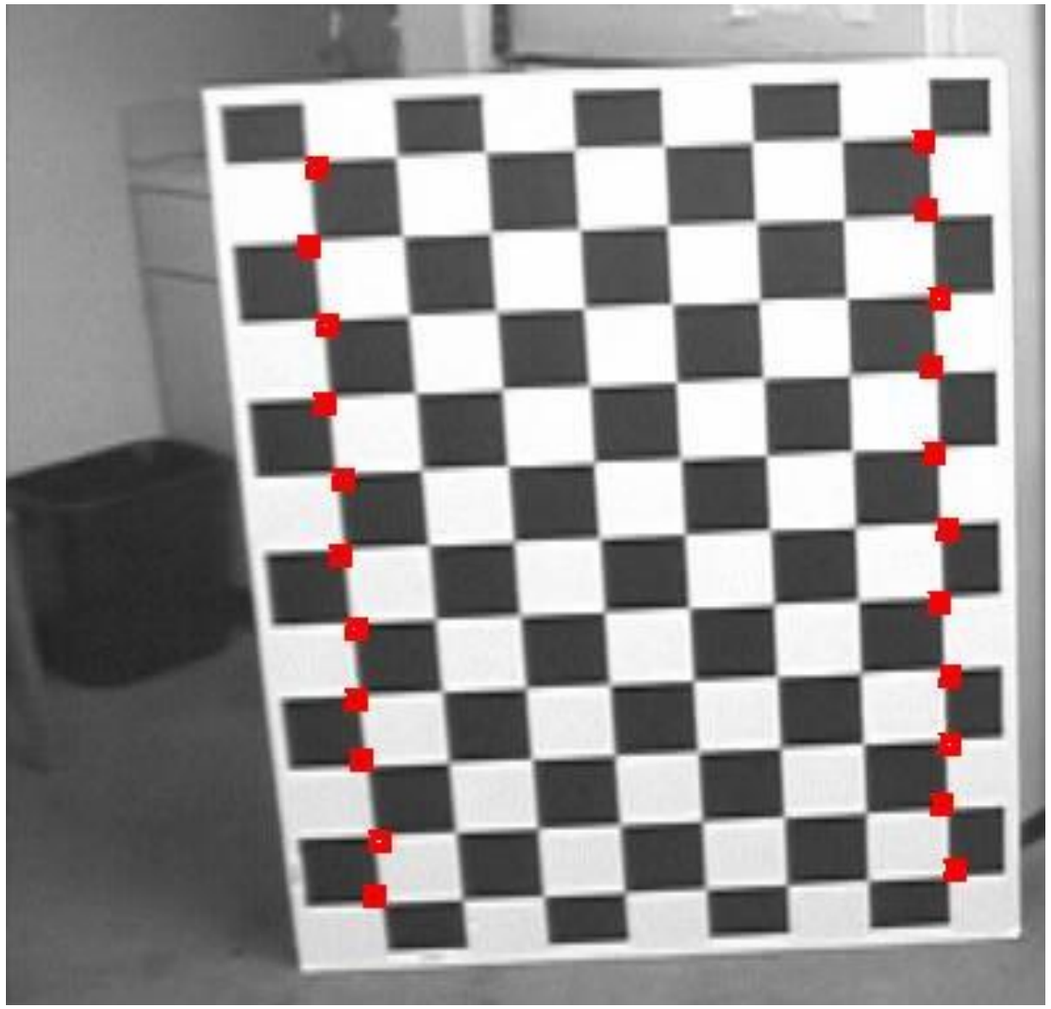
Camera Calibration

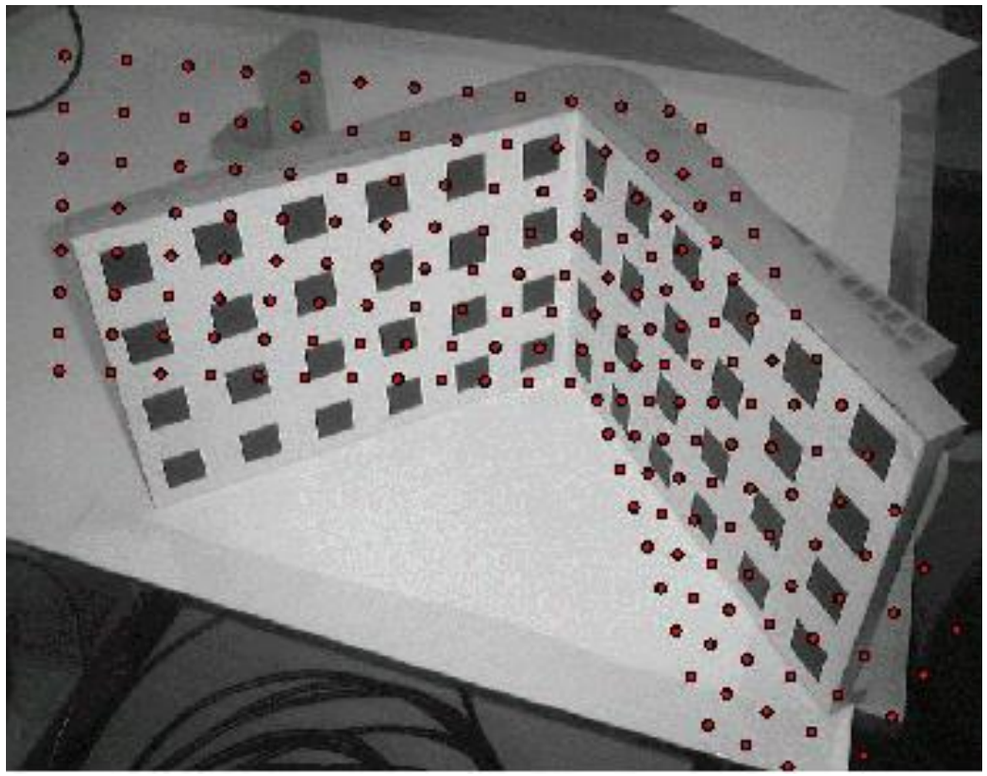
- Needed for most machine vision and photogrammetry tasks (object recognition, pose estimation, …).
- *Calibration* means estimating extrinsic (external) and intrinsic (internal) parameters
- Extrinsic parameters
  - position and orientation (pose) of camera
- Intrinsic parameters
  - Focal length
  - Pixel size
  - (Distortion coefficients)
  - Image center



Example Calibration Patterns

- Geometry of target is known
- Pose of target is not known





Simple Perspective Projection

- A 3x4 camera projection matrix M projects 3D points onto 2D image points

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \mathbf{M} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

3D point in world coords

- This matrix models:
  - rotation and translation
  - focal length
  - ratio of pixel height and width
  - image center

2D image point (x,y)

$$x = u / w, \quad y = v / w$$

- It doesn't model lens distortion

$$\mathbf{M} = \mathbf{K}\,\mathbf{M}_{ext}$$

3x3 intrinsic parameter matrix

3x4 extrinsic parameter matrix, contains world to camera pose

Perspective projection

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

- If the target is planar, we can use Z=0 for all points on the target
- So equation simplifies to

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{14} \\ m_{21} & m_{22} & m_{24} \\ m_{31} & m_{32} & m_{34} \end{pmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}$$

- This 3x3 matrix is a homography that maps the planar target's calibration points (*X*, *Y*, 0) into image coordinates (*x,y*), where *x=u/w*, *y=v/w*

Perspective projection of a plane (continued)

- We can solve for the 9 unknowns (m11..m34) by observing a set of known points on a calibration planar target, then do least squares fitting
  - This is called the “direct linear transformation” (DLT)

- To do this, write:

$$x = \frac{u}{w} = \frac{m_{11}X + m_{12}Y + m_{14}}{m_{31}X + m_{32}Y + m_{34}}, \quad y = \frac{v}{w} = \frac{m_{21}X + m_{22}Y + m_{14}}{m_{31}X + m_{32}Y + m_{34}}$$

- Then

$$\begin{aligned} m_{11}X + m_{12}Y + m_{14} - x\left(m_{31}X + m_{32}Y + m_{34}\right) = 0 \\ m_{21}X + m_{22}Y + m_{14} - y\left(m_{31}X + m_{32}Y + m_{34}\right) = 0 \end{aligned}$$

Am = 0

Direct Linear Transformation

- We collect all the unknowns (m_{ij}) into a vector (9x1), so **Am**=0
- **Am** = 0 is a homogeneous system of equations
- You can only solve for **m** up to an unknown scale factor
- The solution is the eigenvector corresponding to the zero eigenvalue of $\mathbf{A}^T\mathbf{A}$
- You can also find this using Singular Value Decomposition (SVD)
- Recall that we can take the SVD of **A**; ie., $\mathbf{A} = \mathbf{U}\,\mathbf{D}\,\mathbf{V}^T$
  - And **x** is the column of **V** corresponding to the zero singular value of **A**
  - (Since the columns are ordered, this is the rightmost column of **V**)
- We can repeat for other poses

NB: Knowing how to find the unknowns or extract the parameters (next slide) is not part of the syllabus

Extracting parameters

- We next extract the intrinsic and extrinsic camera parameters from **M**, where **M** = **K** $\mathbf{M}_{ext}$
  - The intrinsic parameters (in **K**) are *fx, fy, cx, cy*
  - The extrinsic parameters (in $\mathbf{M}_{ext}$) are *r11, r12, …, r33, tx, ty, tz*

- The procedure requires some linear algebra but is fairly straightforward. It is described clearly in:

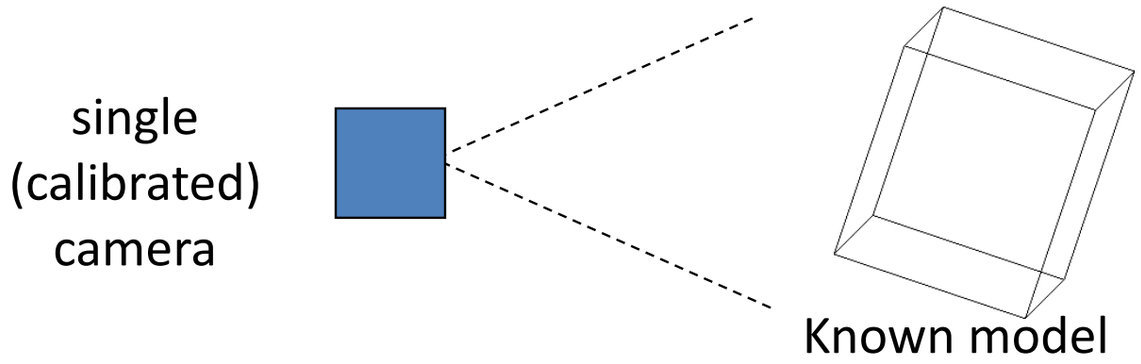
  Zhang, Z. (2000). A flexible new technique for camera calibration. IEEE Trans on Pattern Analysis &Machine Intel, 22(11):1330–1334.

- The final result is that we have solved for all intrinsic and extrinsic parameters except lens distortion

Stereo Vision

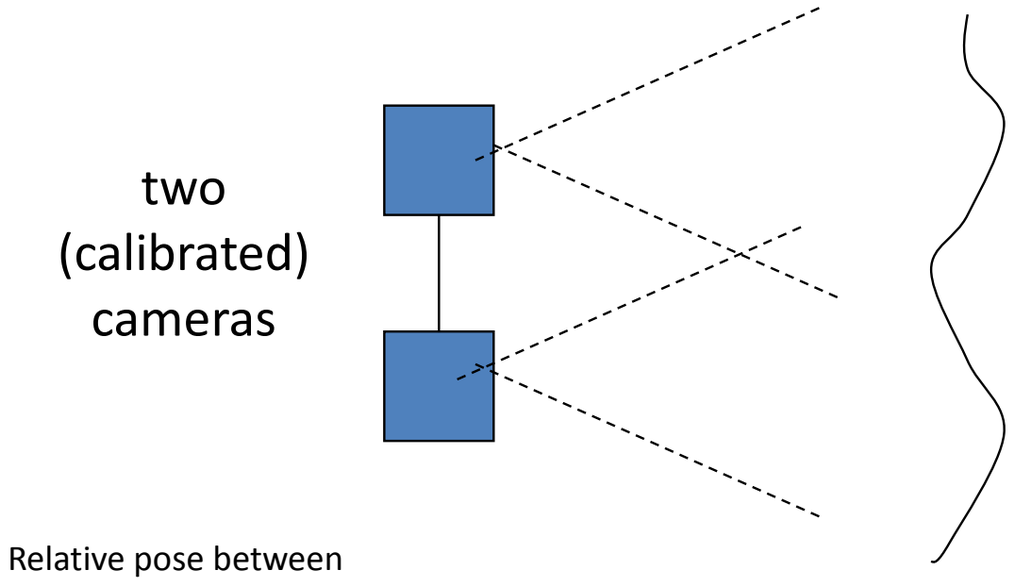
Inferring 3D from 2D

- Model based pose estimation



-> Can determine the pose of the model

- Stereo vision



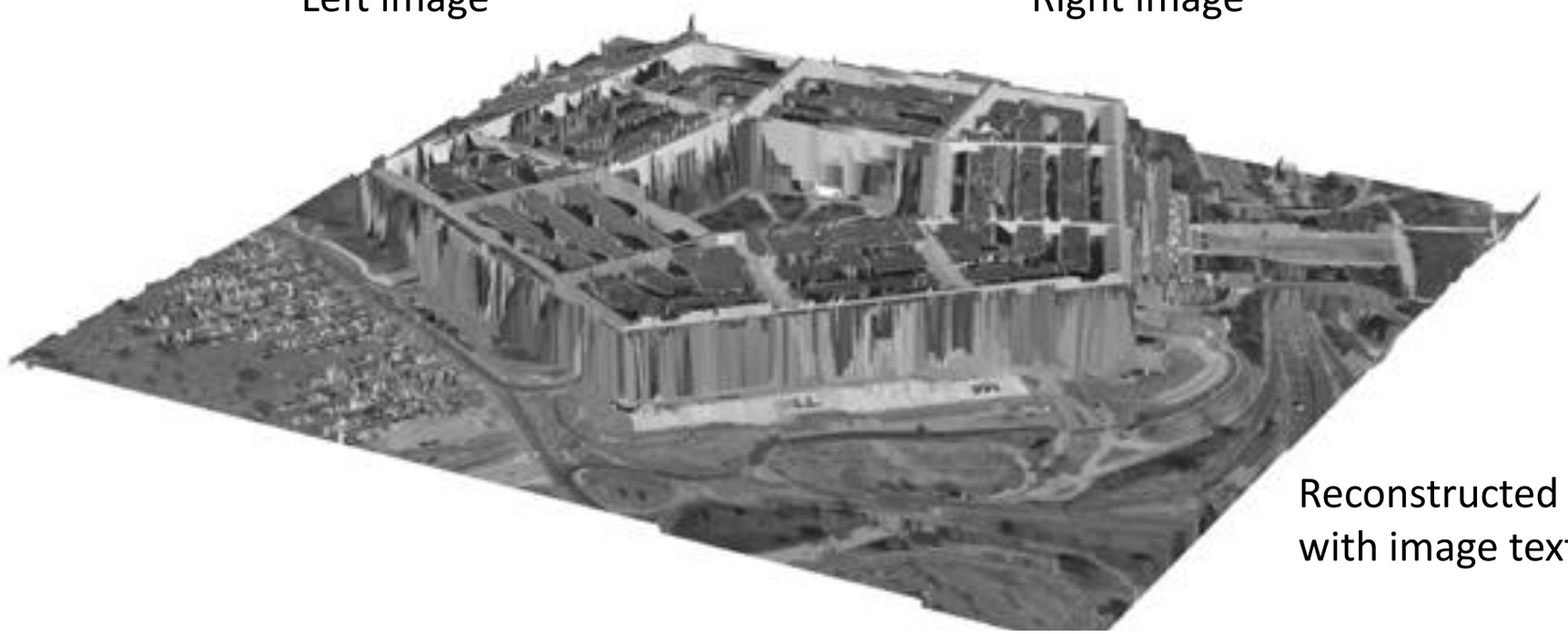
-> Can determine the positions of points in the scene

- A way of getting <u>depth</u> (3-D) information about a scene from two (or more) 2-D images
  - Used by <u>humans</u> and animals, now computers

Example

Left image

Right image



Reconstructed surface
with image texture

Davi Geiger

Stereo Displays

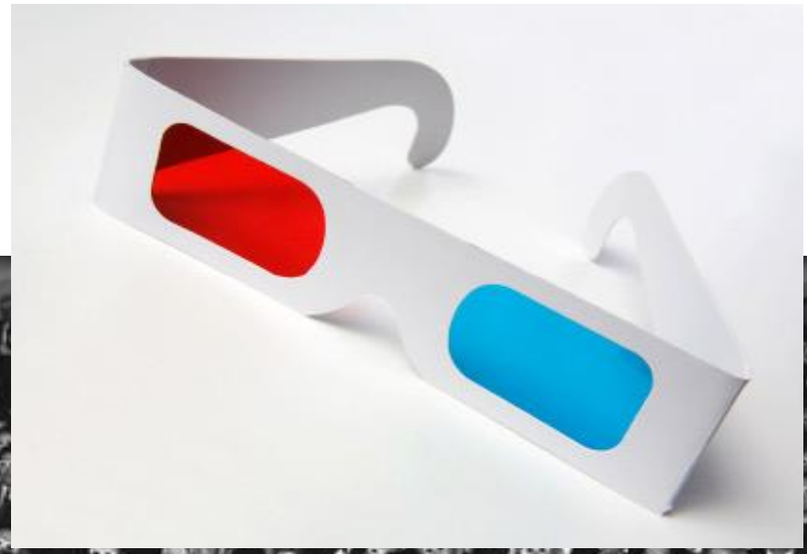
- Stereograms were popular in the early 1900's
- A special viewer was needed to display two different images to the left and right eyes

http://www.columbia.edu/itc/mealac/pritchett/00routesdata/1700_1799/jaipur/jaipurcity/jaipurcity.html

Stereo Displays

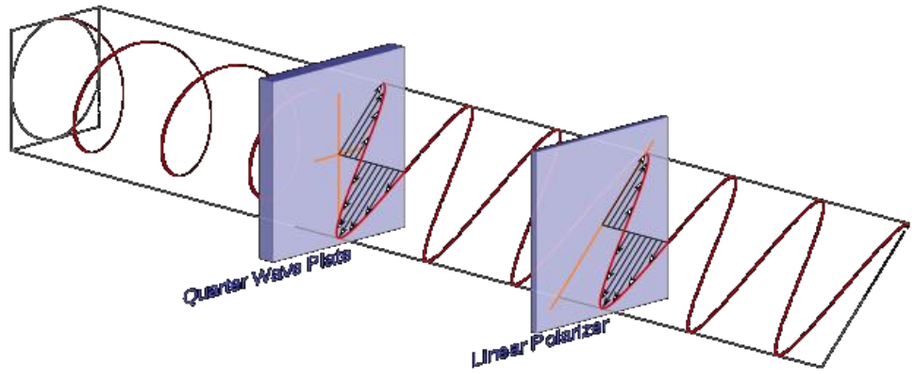
- 3D movies were popular in the 1950's
- The left and right images were displayed as red and blue

http://j-walkblog.com/index.php?/weblog/posts/swimmers/



Stereo Displays

- Current technology for 3D movies and computer displays is to use polarized glasses
- The viewer wears eyeglasses which contain circular polarizers of opposite handedness

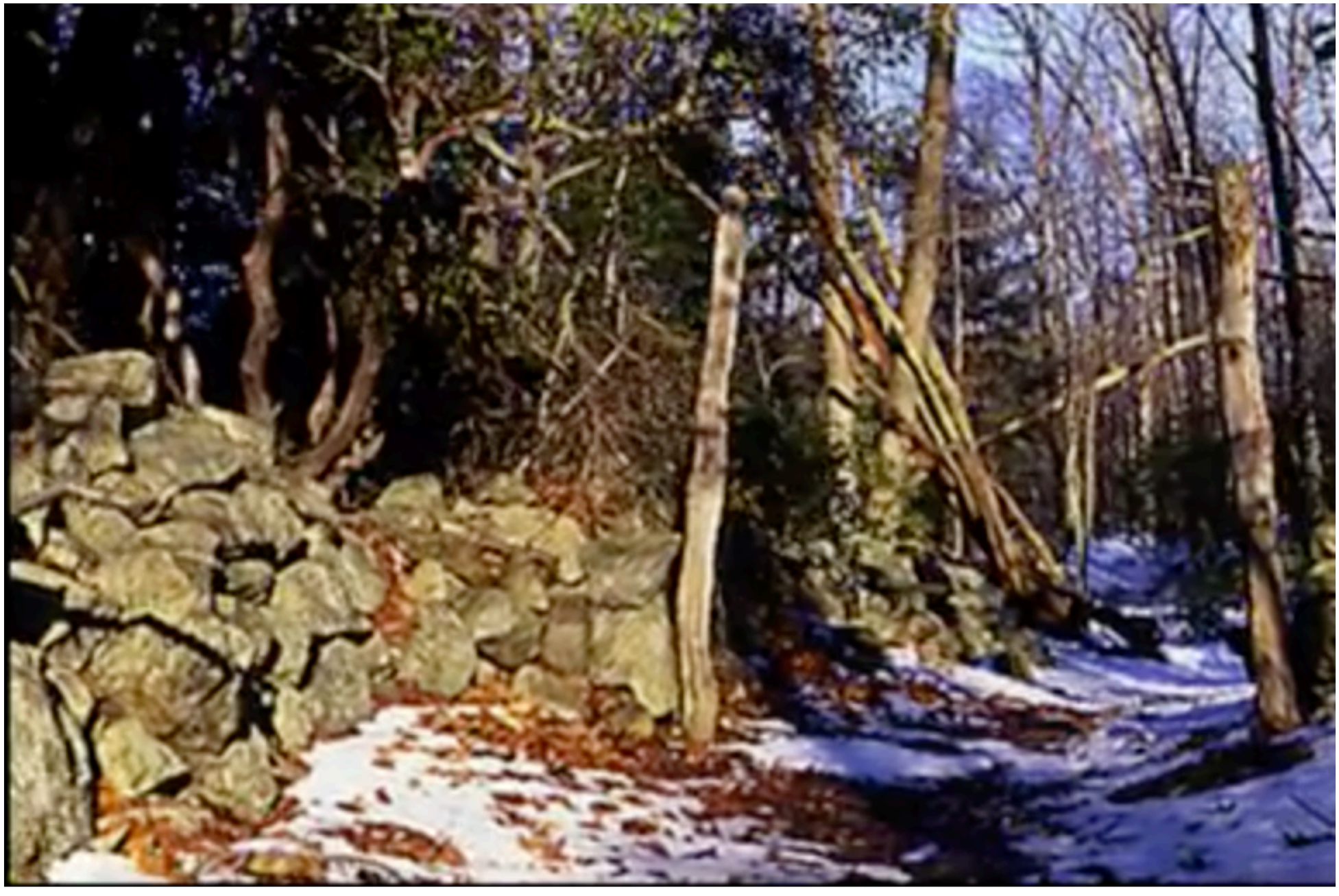


http://www.3dsgamenews.com/2011/01/3ds-to-feature-3d-movies/

Today: HTC Vive, Oculus Rift & Touch, Hololens (AR)

Basic idea: Two slightly different images (views)

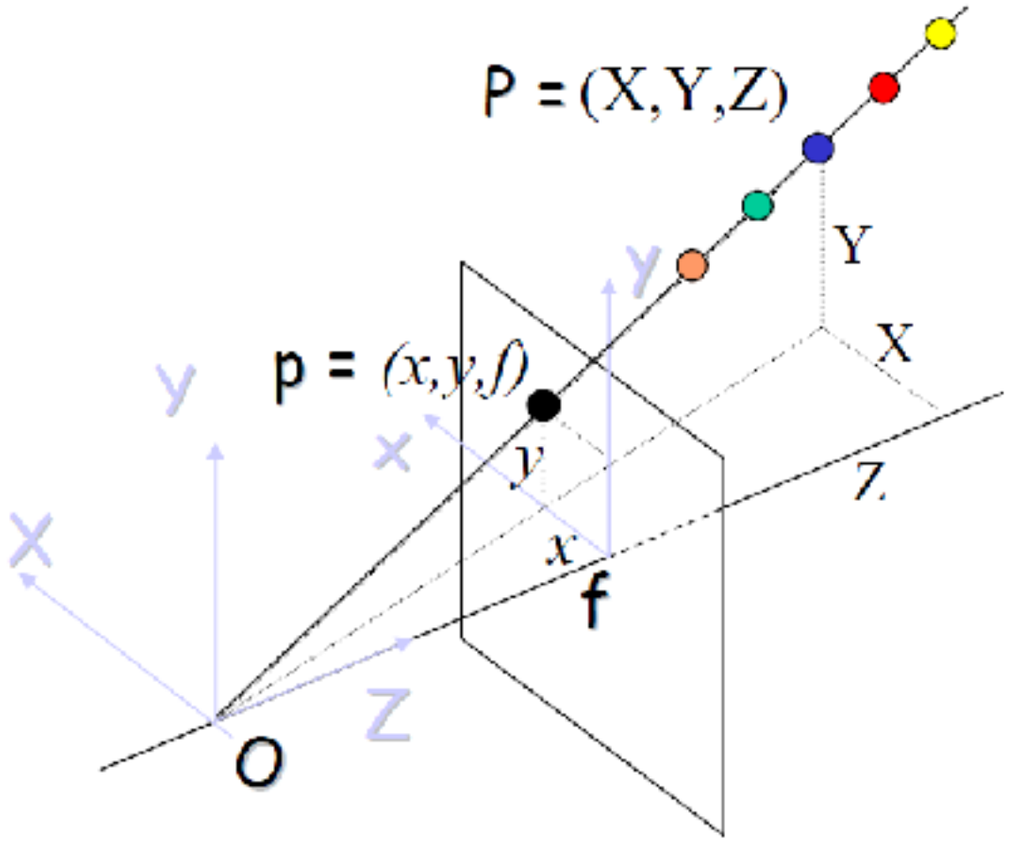
Basic idea: Two slightly different images (views)

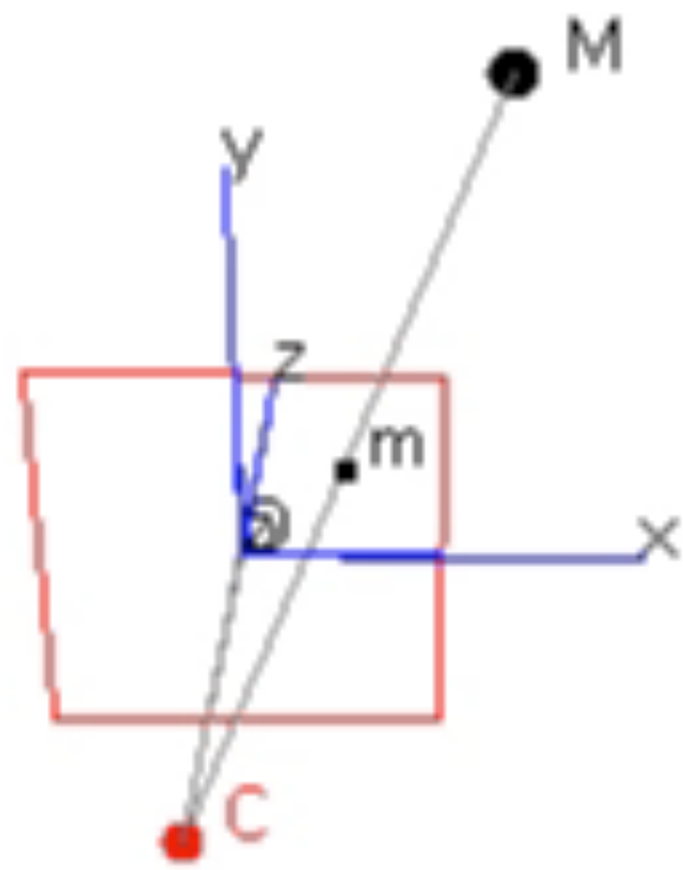


Basic idea: Two slightly different images (views) (2)

Basic idea: Two slightly different images (views) (2)

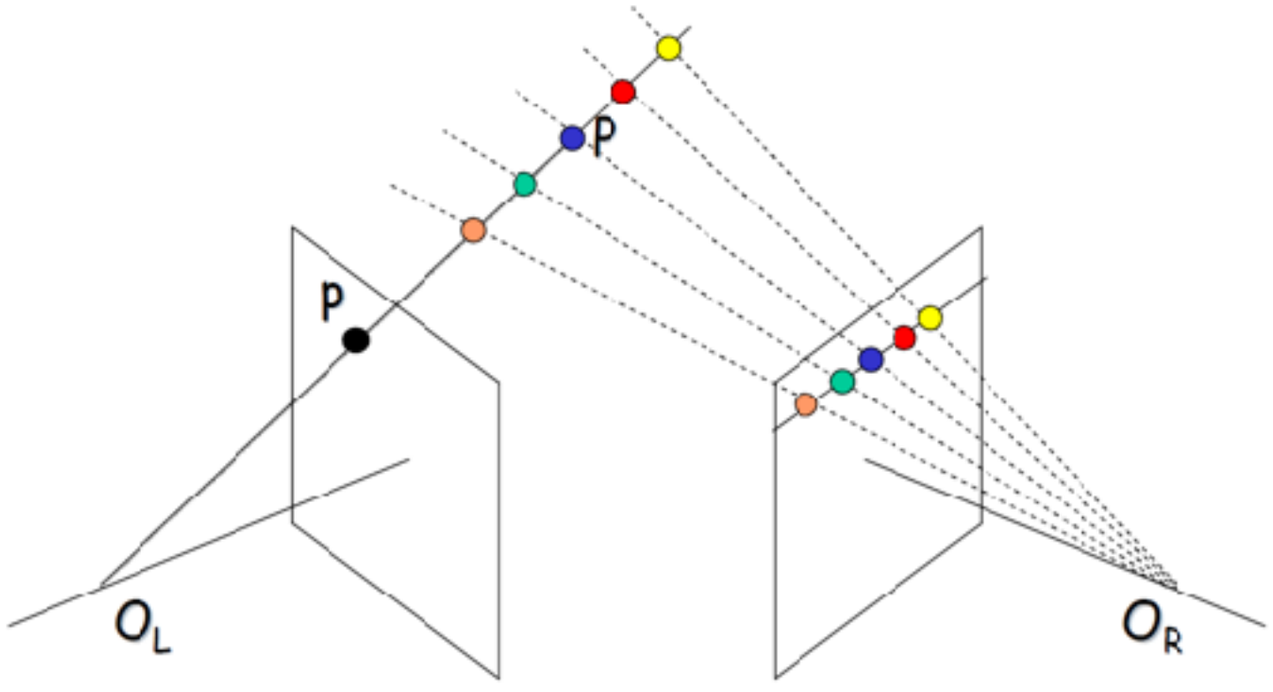
Single Camera / View

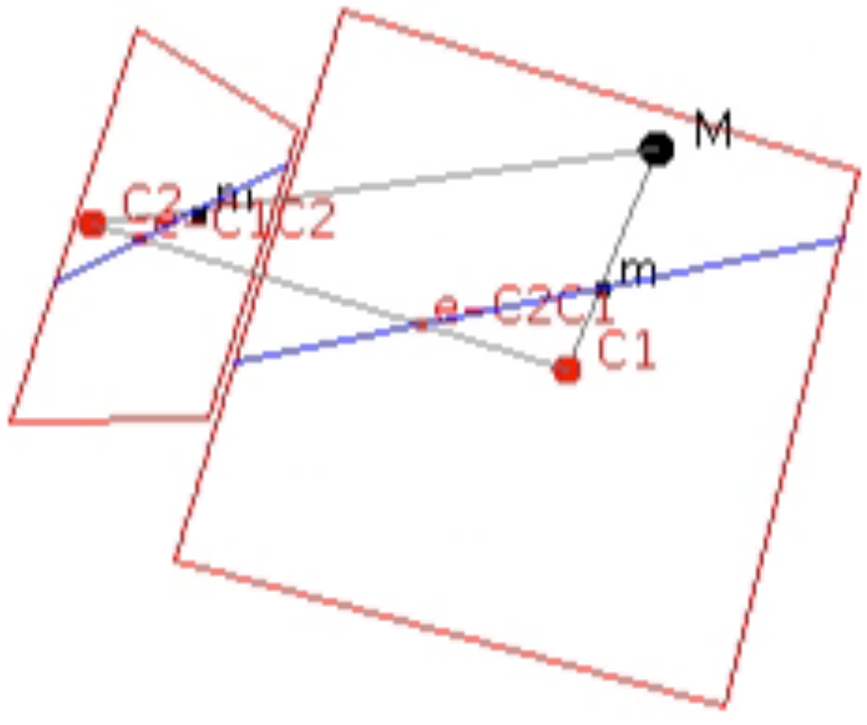




Fundamental ambiguity: Any scene point on the ray OP has image point p

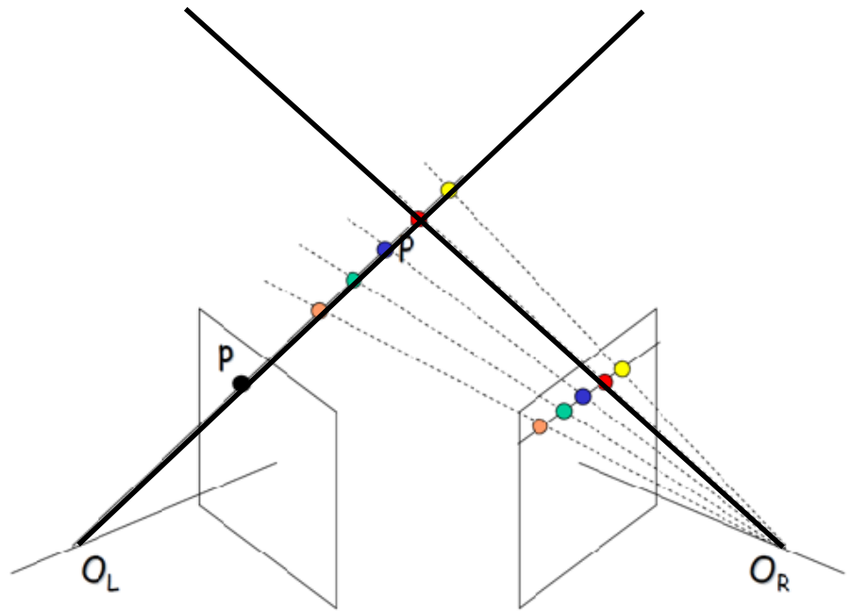
Stereo





A second camera / view can resolve the ambiguity, enabling measurements of depth via triangulation

Stereo Principle



- If you know
  - intrinsic parameters of each camera
  - the relative pose between the cameras
- If you measure
  - An image point in the left camera
  - The corresponding point in the right camera

- Each image point corresponds to a ray emanating from that camera
- You can intersect the rays (triangulate) to find the absolute point position

Stereo Geometry – Simple Case

Formulas derived work in the general case as well

- Assume image planes are coplanar
- There is only a translation in the *X* direction between the two coordinate frames
- *b* is the baseline distance between the cameras

$$Z_L = Z_R = Z$$

$$X_L = X_R + b$$

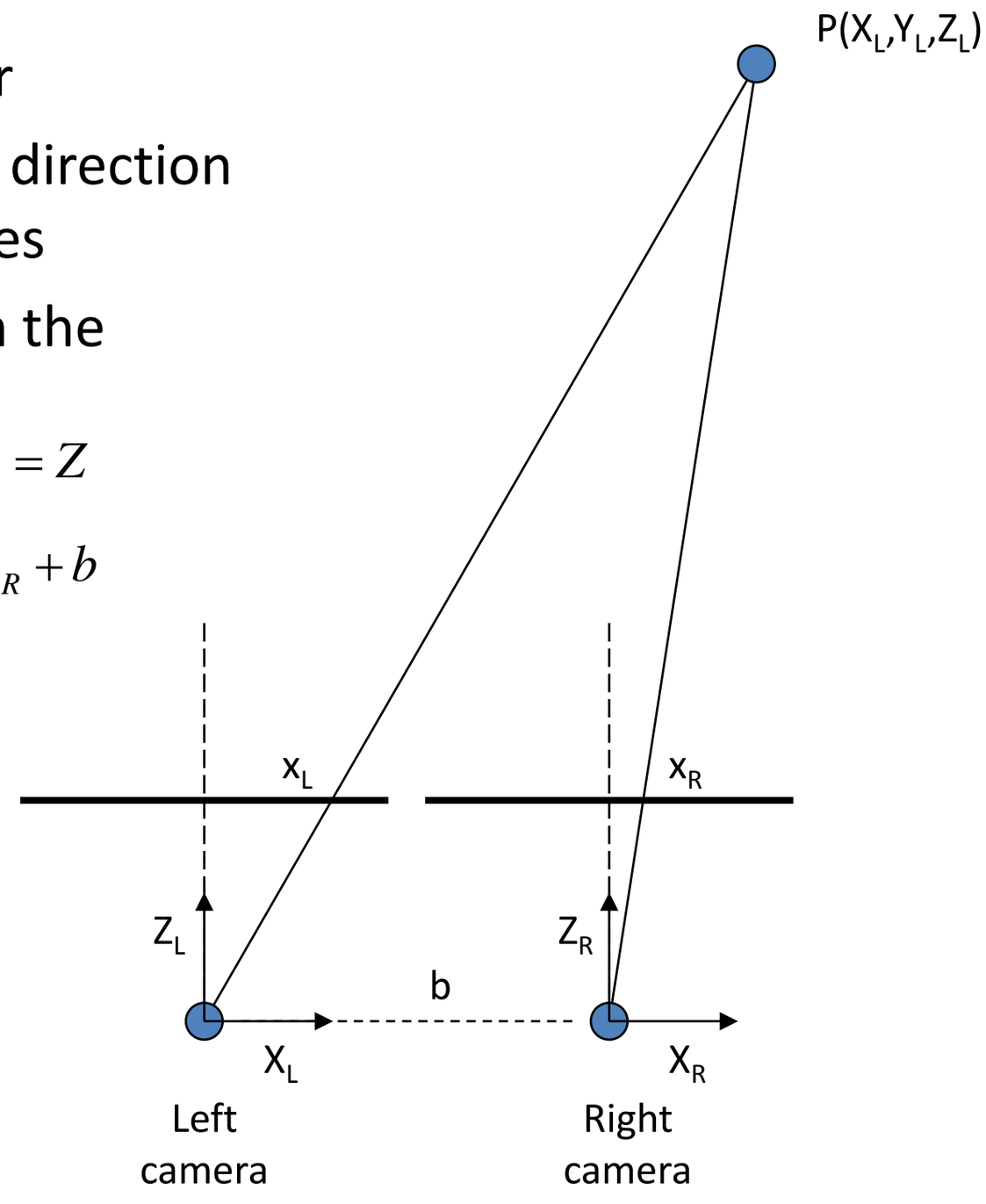
$$x_L = f\frac{X_L}{Z_L},\ x_R = f\frac{X_R}{Z_R}$$

$$\rightarrow \quad x_L = f\frac{X_R + b}{Z}$$

$$d = x_L - x_R = f\frac{(X_R + b) - X_R}{Z} = f\frac{b}{Z}$$

$$\rightarrow \quad Z = f\frac{b}{d}$$

Disparity d = x_L - x_R



Z increases -> d decreases, Z decreases -> d increases **d increases -> Z closer, d decreases -> Z further away**

Stereo Process

- Extract features from the left and right images
- Match the left and right image features, to get their disparity in position (the “correspondence problem”)
- Use stereo disparity to compute depth (the reconstruction problem)

http://vision.middlebury.edu/stereo/data/scenes2003/

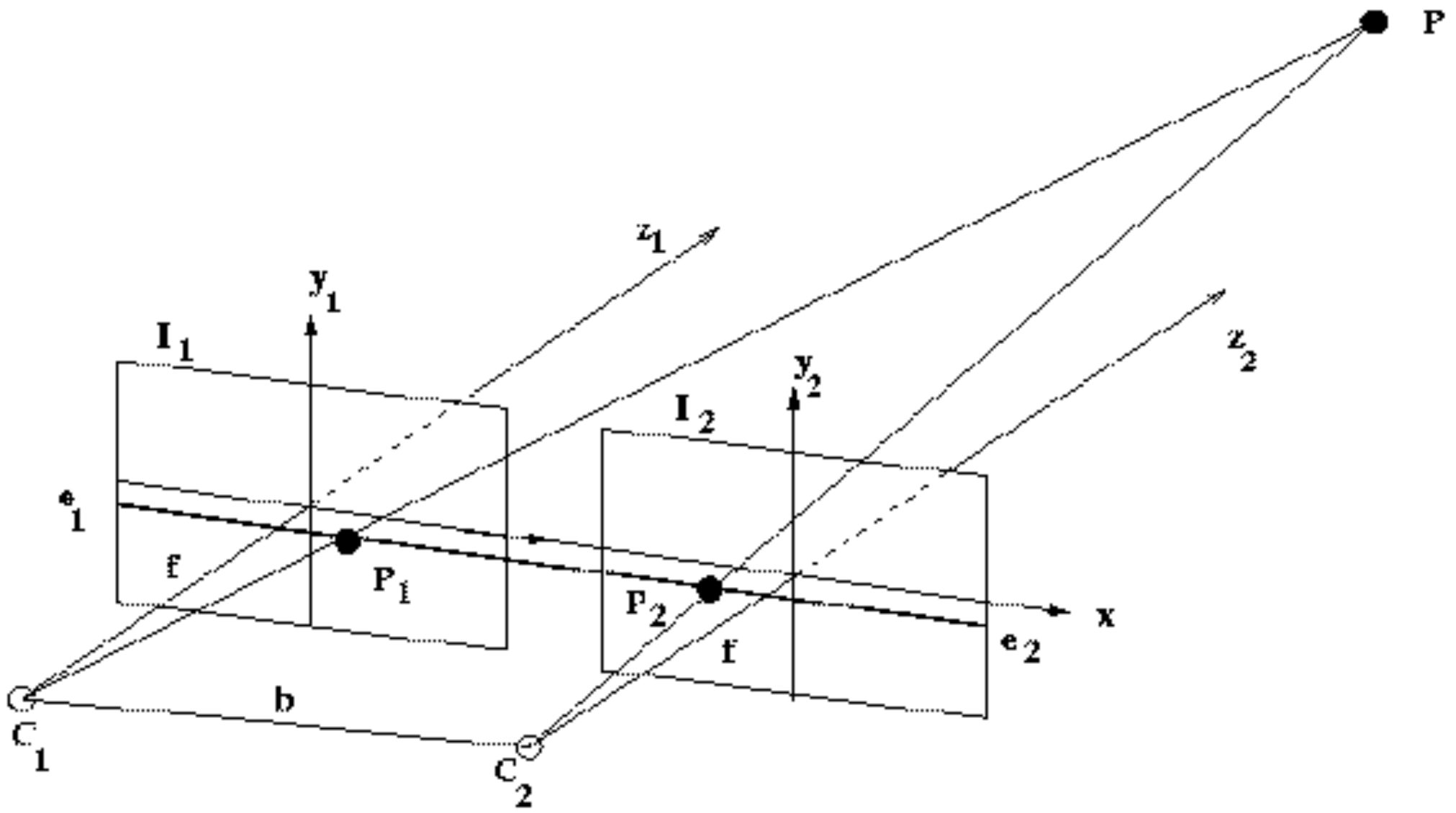
- The correspondence problem is the most difficult

Correspondence Problem – Most difficult part of stereo vision

- For every point in the left image, there are many possible matches in the right image
- Locally, many points look similar -> matches are ambiguous

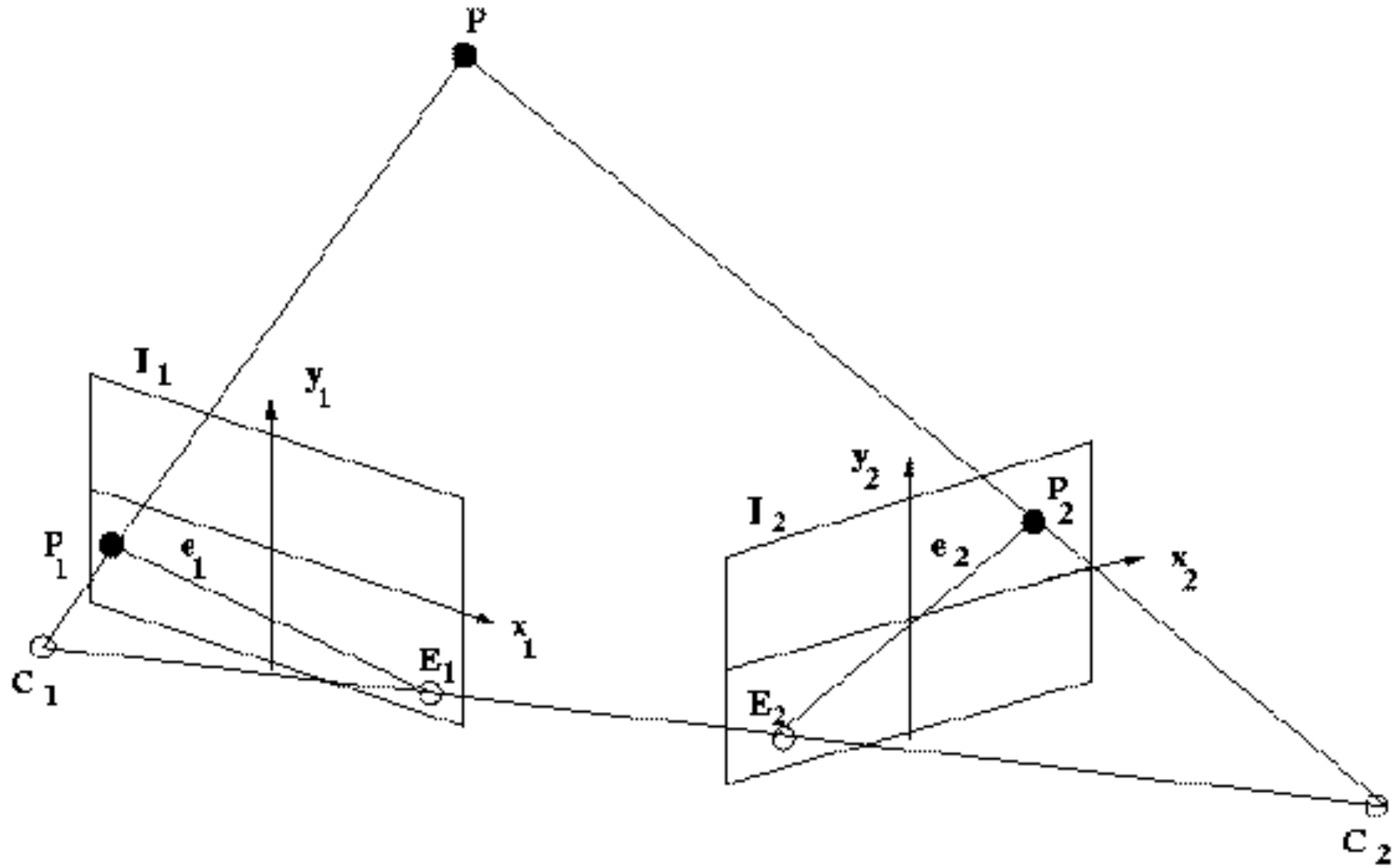
- We can use the (known) geometry of the cameras to help limit the search for matches
- The most important constraint is the epipolar constraint
  - We can limit the search for a match to be along a certain line in the other image

Epipolar Constraint



With aligned cameras, search for corresponding point is 1D along corresponding row of other camera.

Epipolar constraint for non baseline stereo computation



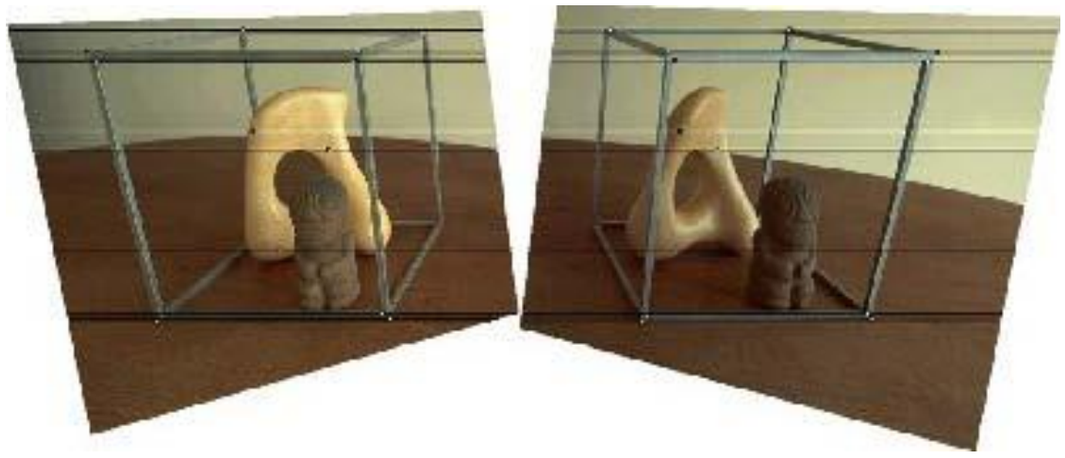
If cameras are not aligned, a 1D search can still be determined for the corresponding point. P1, C1, C2 determine a plane that cuts image I2 in a line: P2 will be on that line.

Rectification

- If relative camera pose is known, it is possible to rectify the images – effectively rotate both cameras so that they are looking perpendicular to the line joining the camera centers

Original image pair overlaid with several epipolar lines

- These means that epipolar lines will be horizontal, and matching algorithms will be more efficient



Images rectified so that epipolar lines are horizontal and in vertical correspondence

From Richard Szeliski, Computer Vision: Algorithms and Applications, Springer, 2010

Correspondence Problem

- Even using the epipolar constraint, there are many possible matches

Still a difficult problem..

- Worst case scenarios
  - A white board (no features)
  - A checkered wallpaper (ambiguous matches)
- The problem is under constrained

- To solve, we need to impose assumptions about the real world:
  - Disparity limits
  - Appearance
  - Uniqueness
  - Ordering
  - Smoothness

Methods for Correspondence

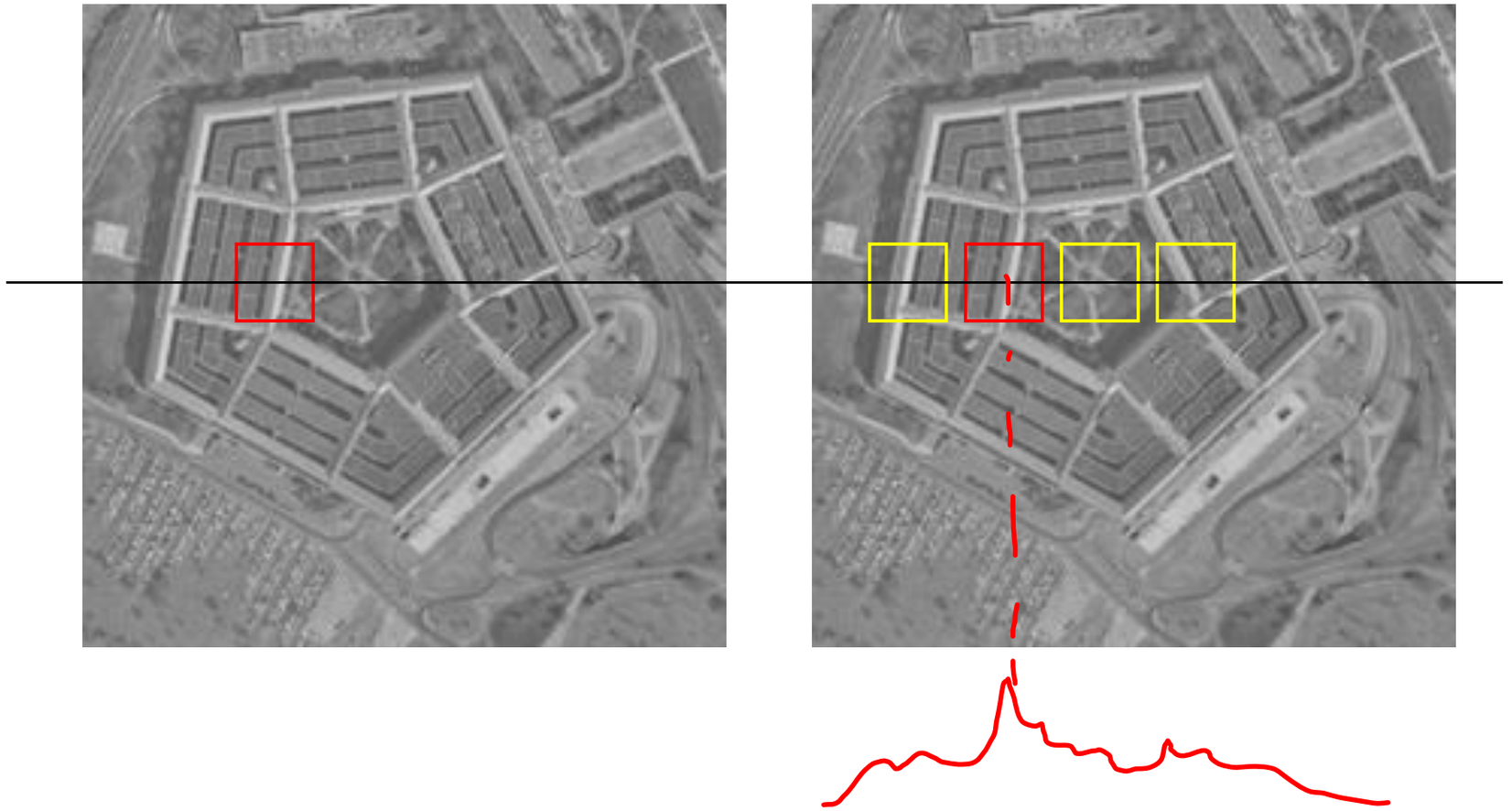
- Match points based on local similarity between images
- Two general approaches

- Correlation-based approaches
  - Matches image patches using correlation
  - Assumes only a translational difference between the two local patches (no rotation, or differences in appearance due to perspective)
  - A good assumption if patch covers a single surface, and surface is far away compared to baseline between cameras
  - Works well for scenes with lots of texture

- Feature-based approaches
  - Matches edges, lines, or corners
  - Gives a sparse reconstruction
  - May be better for scenes with little texture

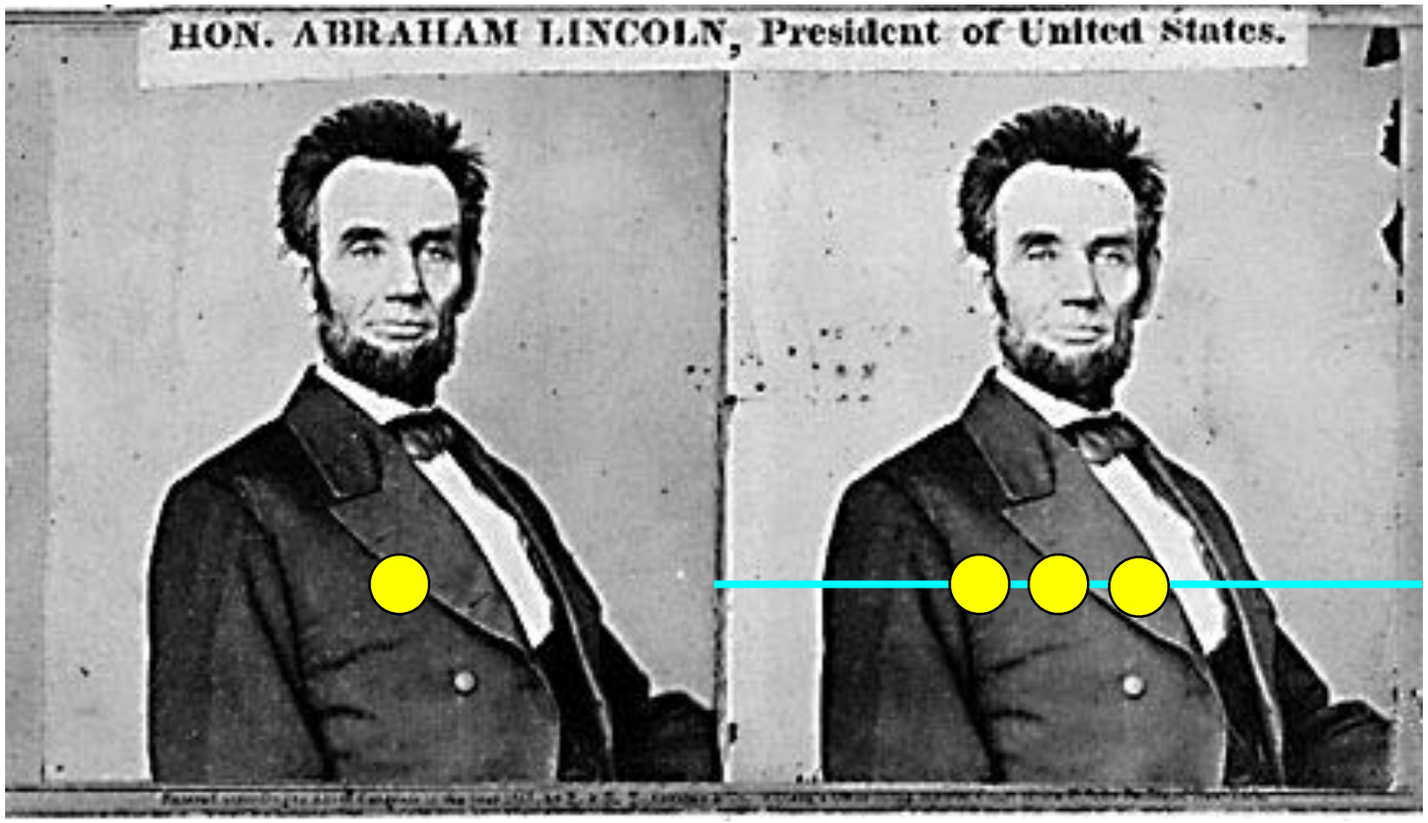
Correlation Approach

- Select a range of disparities to search
- For each patch in the left image, compute cross correlation score for every point along the epipolar line
- Find maximum correlation score along that line



Summery:

Stereo, correspondence, disparity and depth



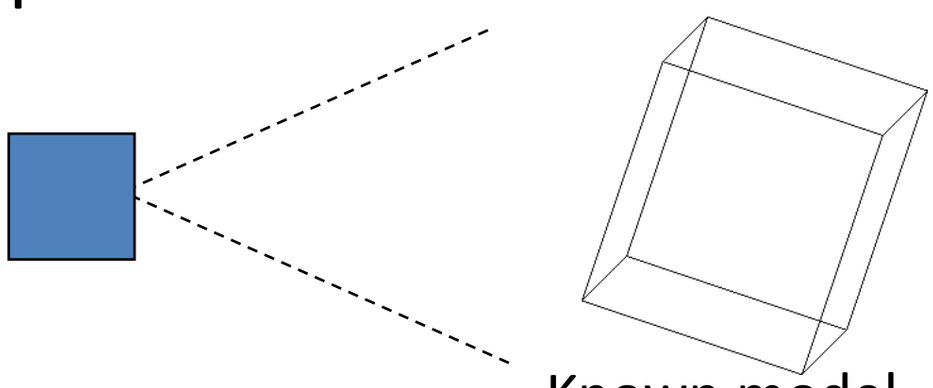
- **For each pixel** x in the first image
  - Find corresponding epipolar scanline in the right image
  - Examine all pixels on the scanline and pick the best match x’
  - Compute disparity x-x’ and set depth(x) = b*f/(x-x’)

Epipolar Geometry and the Essential Matrix

Inferring 3D from 2D

- Possible methods to obtain 3D information:
  - Model based pose estimation

single (calibrated) camera

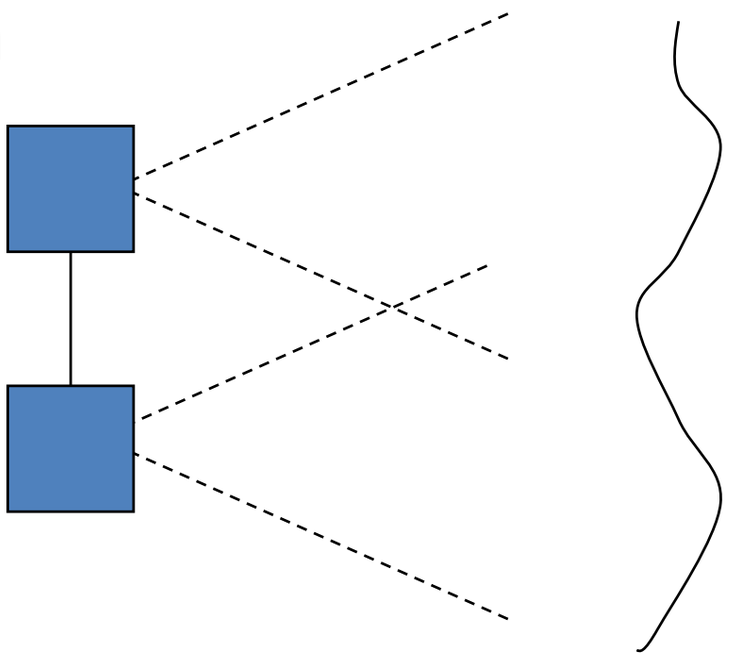


Known model

-> Can determine the pose of the model wrt camera

  - Stereo vision

two (calibrated) cameras



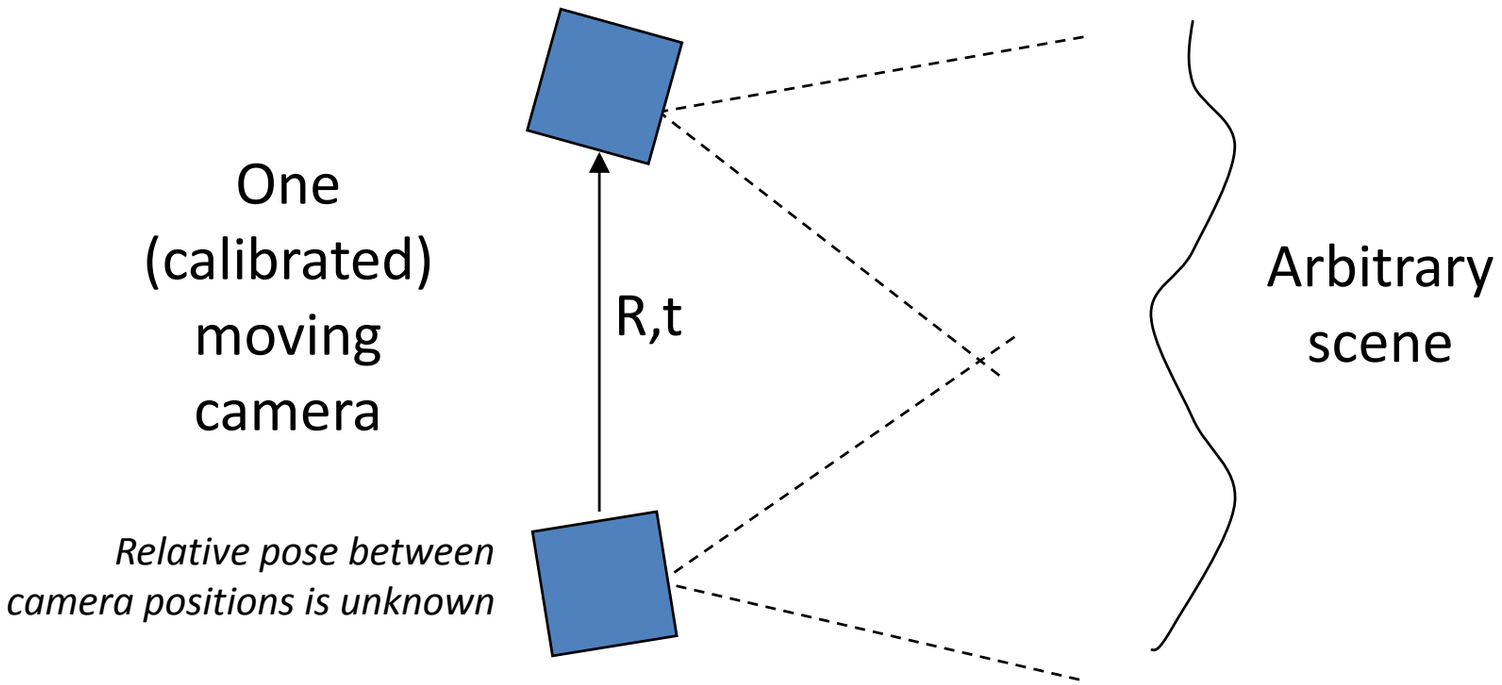
Relative pose between cameras is also known

Arbitrary scene

-> Can determine the positions of points in the scene

Inferring 3D from 2D

- Another method
  - Structure-from-motion (might be better termed "structure-and-motion from a moving camera")



-> Can determine the positions of points in the scene, as well as the motion of the camera (R,t)

However, we will see that the positions of points and the translation of the camera have an unknown scale factor

Outline

First:

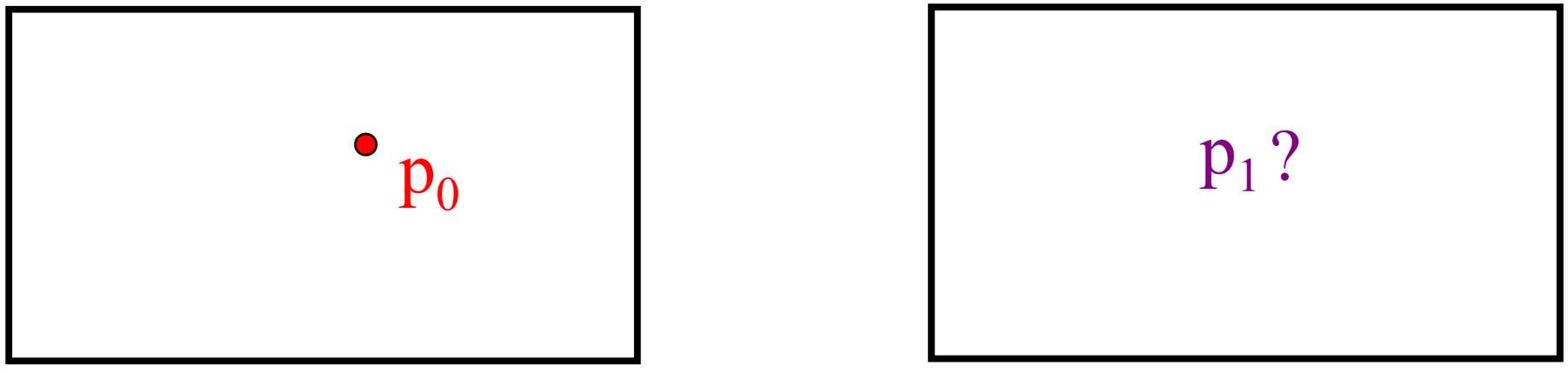
- First we review the geometry and representation of epipolar lines
- Next we derive the essential matrix and show how it can predict the locations of epipolar lines

Then:

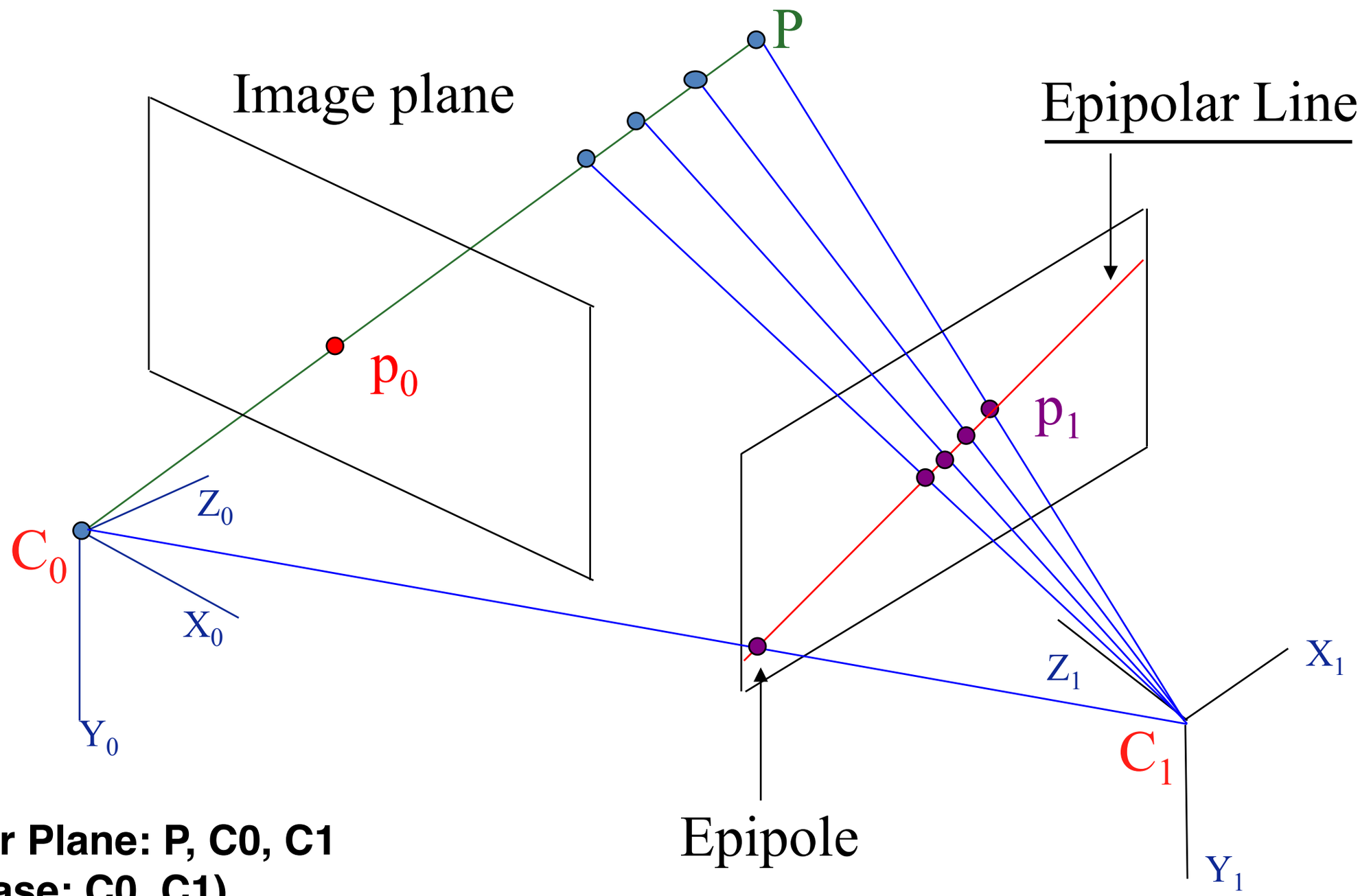
- Show how the essential matrix can be estimated from point correspondences
- Show how to recover rotation and translation
- Show how to recover point positions

Epipolar Geometry

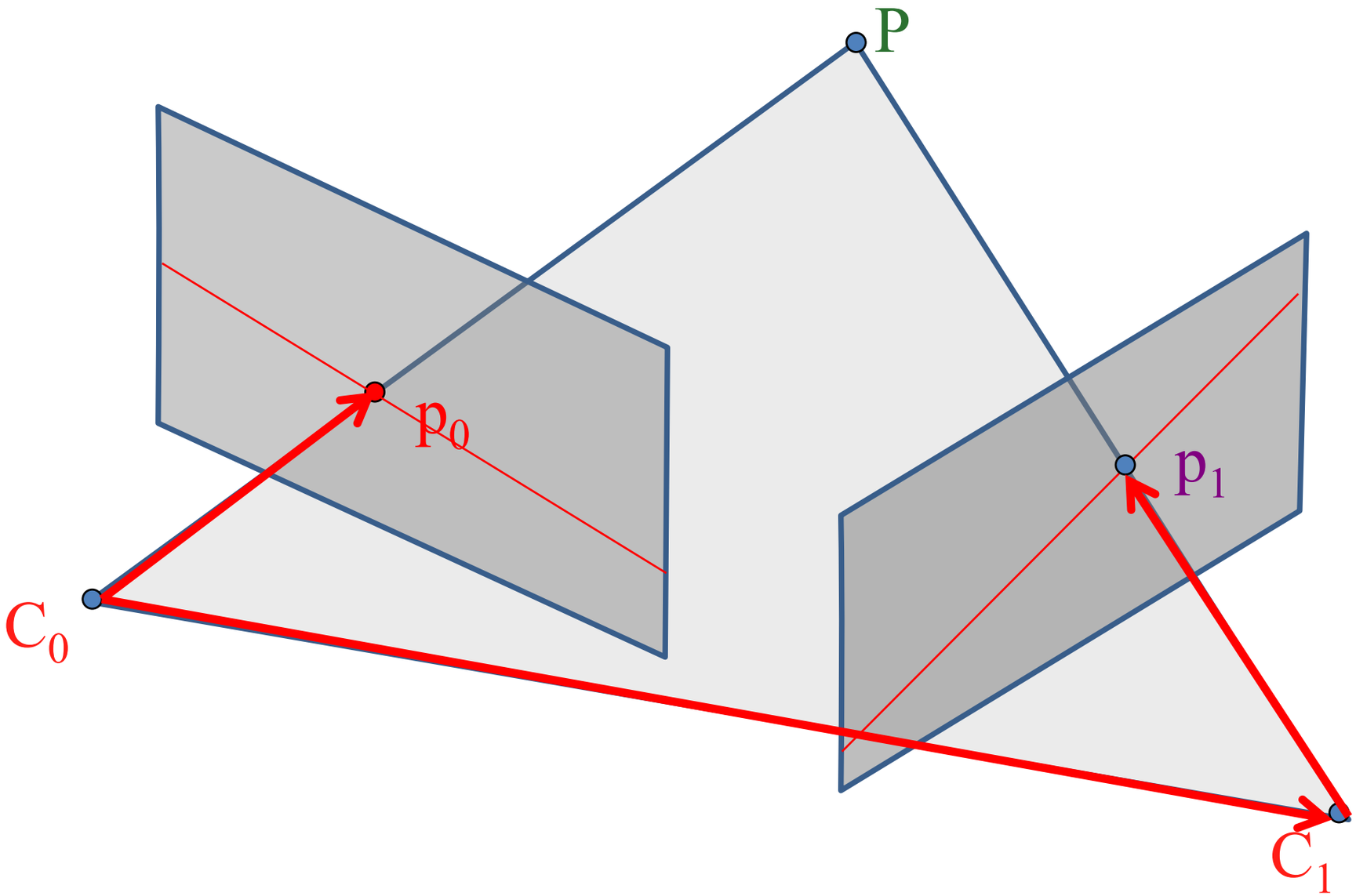
- We have two views of a scene, taken from different viewpoints
- We see an image point **p** in one image, which is the projection of a 3D point



Given $\mathbf{p}_0$ in the first image, where can the corresponding point $\mathbf{p}_1$ in the second image be?



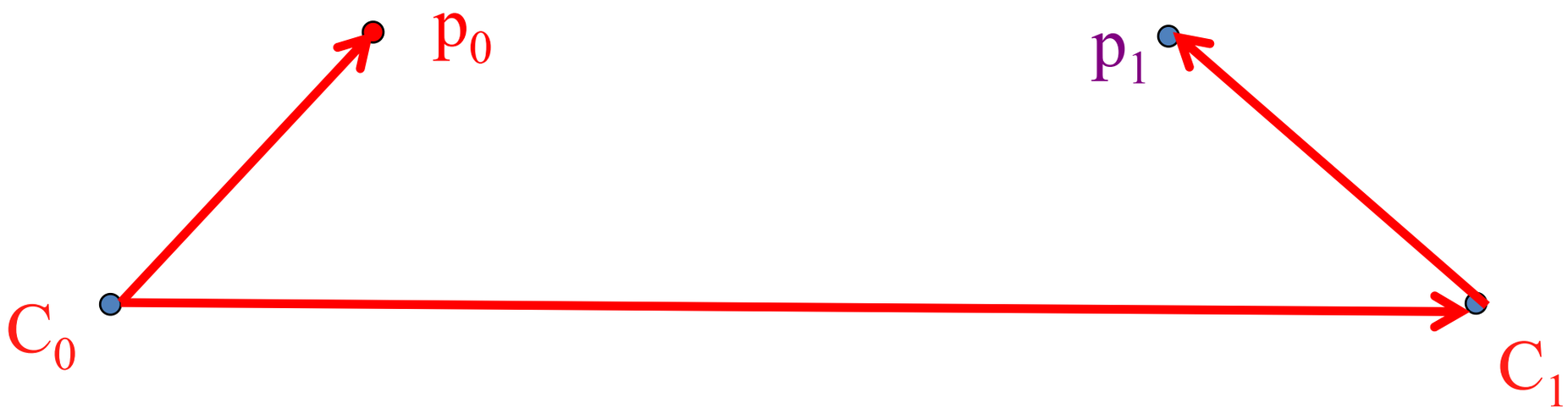
Epipolar Plane: P, C0, C1
Base: C0, C1)



- The optical centers of the two cameras, a point **P**, and the image points $\mathbf{p}_0$ and $\mathbf{p}_1$ of **P** all lie in the same plane (epipolar plane)
- These vectors are co-planar: $\overrightarrow{\mathbf{C}_0\mathbf{p}_0}$, $\overrightarrow{\mathbf{C}_1\mathbf{p}_1}$, $\overrightarrow{\mathbf{C}_0\mathbf{C}_1}$

Intersection: 3D epipolar plane & 2D image plane

Epipolar line corresponding to point in other image and vice versa



- Another way to write the fact they are co-planar is

$$\overrightarrow{\mathbf{C}_0\mathbf{p}_0} \cdot \left(\overrightarrow{\mathbf{C}_0\mathbf{C}_1} \times \overrightarrow{\mathbf{C}_1\mathbf{p}_1} \right) = 0$$

- **Cross product of two of them, perpendicular to both vectors**
- **Dot product with first vector should be 0**

- Now, instead of treating $\mathbf{p}_0$ as a point, treat it as a 3D direction vector*

  We assume "normalized" image coordinates; ie effective focal length=1

$$\mathbf{p}_0 = {}^{C_0}\begin{pmatrix} x_0 \\ y_0 \\ 1 \end{pmatrix}$$

This is defined with respect to the coordinate frame of camera 0

3D vector in camera 0

- $\mathbf{p}_1$ is also a direction vector

$$\mathbf{p}_1 = {}^{C_1}\begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix}$$

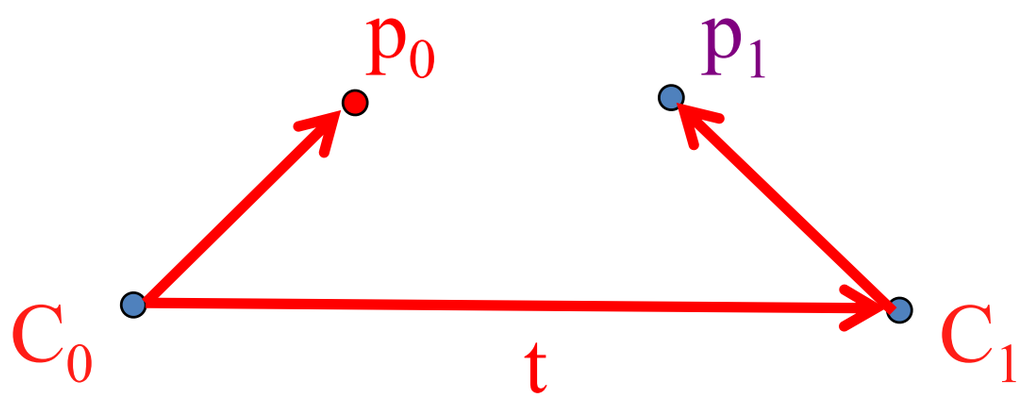
This is defined with respect to the coordinate frame of camera 1

- The direction of $\mathbf{p}_1$ in camera 0 coordinates is

$${}^{C_0}_{C_1}\mathbf{R}\,\mathbf{p}_1$$

Namely, we apply the rotation matrix from the camera 1 to camera 0 pose

*A direction vector is a vector whose starting point (tail) doesn't matter, just its direction



$$\overrightarrow{\mathbf{C}_0\mathbf{p}_0} \cdot \left(\overrightarrow{\mathbf{C}_0\mathbf{C}_1} \times \overrightarrow{\mathbf{C}_1\mathbf{p}_1} \right) = 0$$

- So we can write the coplanar constraint as

$$\mathbf{p}_0 \cdot \left(\mathbf{t} \times \mathbf{R}\mathbf{p}_1 \right) = 0$$

- Where **R** is the rotation of camera 1 wrt camera 0

$$^{C_0}_{C_1}\mathbf{R}$$

- And **t** is the translation of the camera 1 origin wrt camera 0

$$^{C_0}\mathbf{t}_{C_1 org}$$

- Remember that the pose of camera 1 wrt camera 0 is

$$^{C_0}_{C_1}\mathbf{H} = \begin{pmatrix} ^{C_0}_{C_1}\mathbf{R} & ^{C_0}\mathbf{t}_{C_1 org} \\ \mathbf{0} & 1 \end{pmatrix}$$

Cross Product as Matrix Multiplication

- The cross product of a vector **a** with a vector **b, a x b,** can be represented as a 3x3 matrix times the vector **b**:
  - $[\mathbf{a}]_x \mathbf{b}$, where $[\mathbf{a}]_x$ is a skew symmetric matrix

$$\vec{a} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \quad \vec{b} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

- It is easy to show that

$$\vec{a} \times \vec{b} = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} = \begin{pmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{pmatrix}$$

$$[\mathbf{a}]_\times = \begin{pmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{pmatrix}$$

(Show this!)

$$[a]_x \vec{b} = \begin{pmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

$$= \begin{pmatrix} -a_3 b_2 + a_2 b_3 \\ a_3 b_1 - a_1 b_3 \\ -a_2 b_1 + a_1 b_2 \end{pmatrix} \quad \checkmark$$

Matrix Form of Epipolar Constraint

- We have

$$\mathbf{p}_0 \cdot \left(\mathbf{t} \times \mathbf{R}\mathbf{p}_1\right) = 0$$

- Or

$$\mathbf{p}_0^T \left[\mathbf{t}\right]_\times \mathbf{R}\mathbf{p}_1 = 0$$

- Where $[\mathbf{t}]_x$ is the 3x3 skew symmetric matrix for **t** corresponding to the cross product operator

- Let $\mathbf{E} = [\mathbf{t}]_x \mathbf{R}$ (which is a 3x3 matrix)
- Then

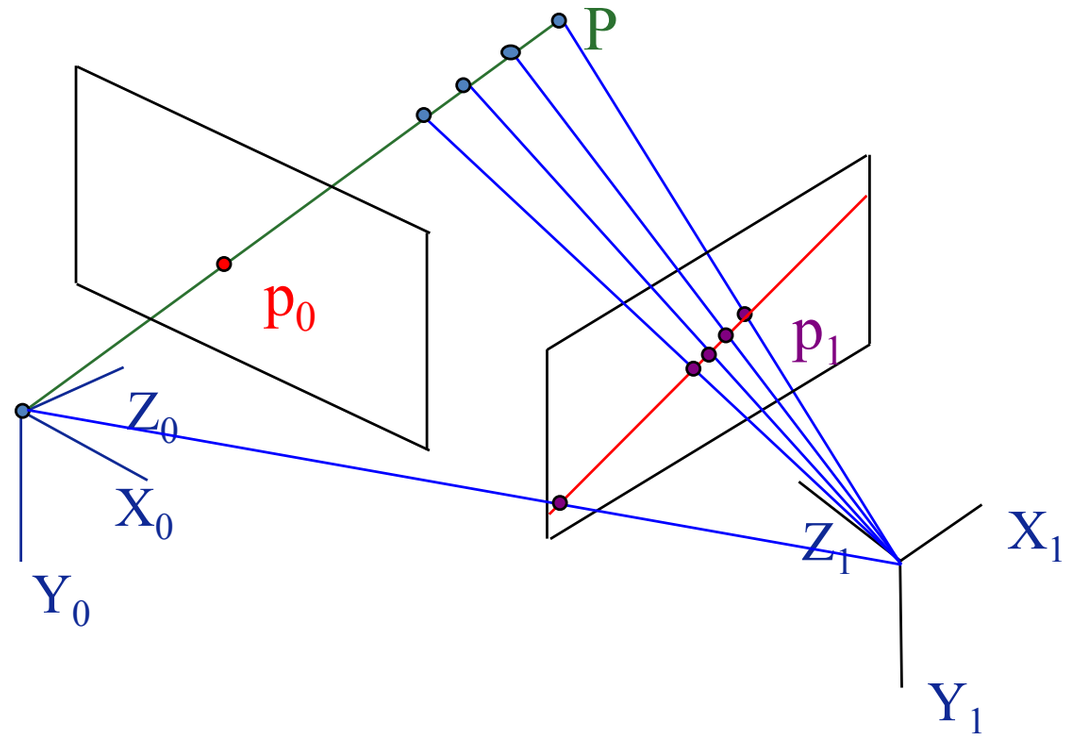
$$\mathbf{p}_0^T \mathbf{E} \mathbf{p}_1 = 0 \qquad \begin{pmatrix} x_0 & y_0 & 1 \end{pmatrix} \begin{pmatrix} E_{11} & E_{12} & E_{13} \\ E_{21} & E_{22} & E_{23} \\ E_{31} & E_{32} & E_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} = 0$$

Epipolar constraint as a cross product -> as a matrix product

The Essential Matrix

- Is the matrix **E**, that relates the image of a point in one camera to its image in the other camera, given a translation and rotation

$$\mathbf{p}_0^T \, \mathbf{E} \, \mathbf{p}_1 = 0$$



- where

$$\mathbf{E} = [\mathbf{t}]_x \, \mathbf{R}$$

Essential matrix

- The essential matrix is $\mathbf{E} = [\mathbf{t}]_x \mathbf{R}$
  - where
    - $[\mathbf{t}]_x$ is the skew symmetric matrix corresponding to **t**
    - **t** is the translation of camera 2 with respect to camera 1; i.e., ${}^{c1}\mathbf{P}_{c2org}$
    - **R** is rotation of camera 2 with respect to camera 1; i.e., ${}^{c1}{}_{c2}\mathbf{R}$

```
% Calculate essential matrix
t = Pc2org_c1;
E = [ 0 -t(3) t(2); t(3) 0 -t(1); -t(2) t(1) 0] * R_c2_c1;
```

```
E =
         0   -1.0000         0
   -0.3615         0   -3.1415
         0    3.0000         0
```

Representation of a line

- Equation of a line in the (x,y) plane is $ax + by + c = 0$
- We get the same line with the equation

  $(ka)x + (kb)y + (kc) = 0$, for any non-zero constant k

- So a line may be represented by the homogeneous coordinates

  $\mathbf{l} = (a,b,c)^T$

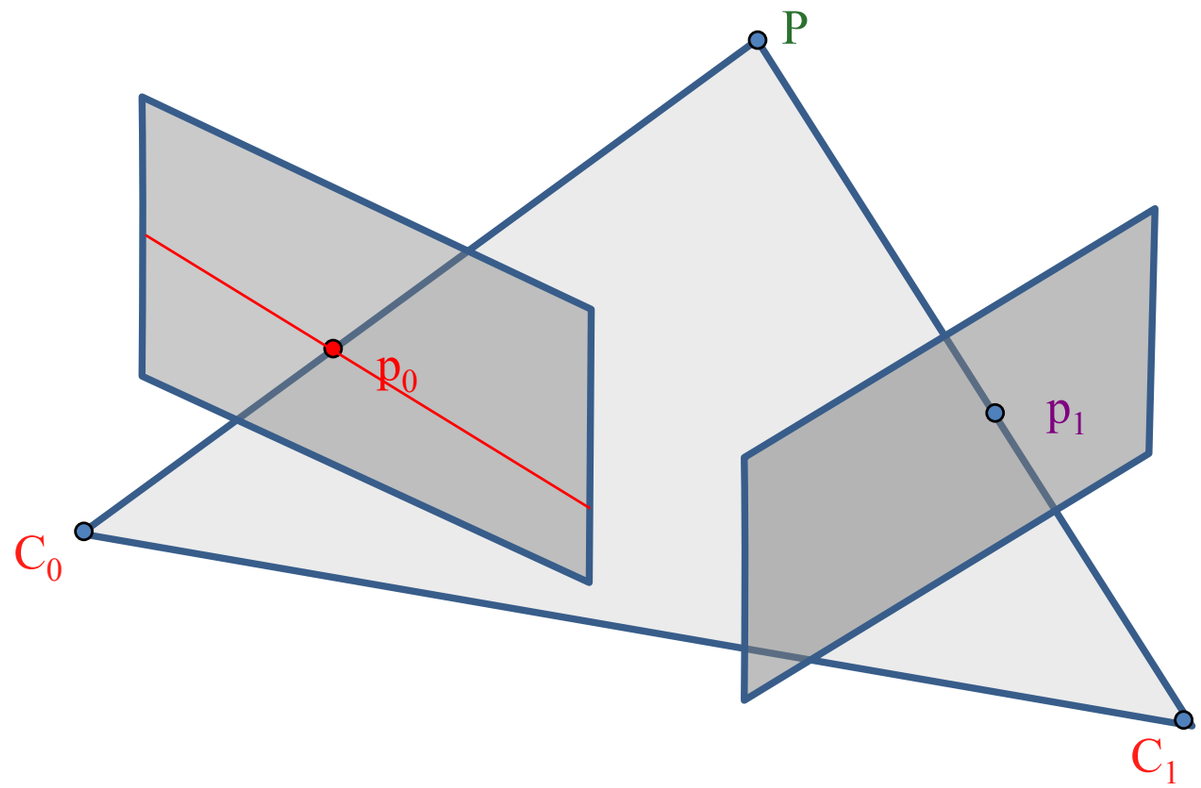
- Note: The point $\mathbf{p}$ lies on the line $\mathbf{l}$ if and only if $\mathbf{p}^T\,\mathbf{l} = 0$

$$p^T l = \begin{pmatrix} x & y & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = ax + by + c = 0$$

Epipolar Lines

- Recall $\mathbf{p}_0^T \, \mathbf{E} \, \mathbf{p}_1 = 0$

- So $\mathbf{E}\,\mathbf{p}_1$ is the epipolar line corresponding to $\mathbf{p}_1$ in the camera 0 image



- Or, writing another way,

$$\begin{pmatrix} x_0 & y_0 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = ax_0 + by_0 + c = 0$$

where $\mathbf{l} = (a,b,c)^T = \mathbf{E}\,\mathbf{p}_1$ are the parameters of the line

Visualization

- Pick a point $\mathbf{p}_1$ in the second image

- Calculate the corresponding epipolar line in the first image, $\mathbf{l}=\mathbf{E}\,\mathbf{p}_1$
  - where $\mathbf{l}=(a,b,c)$
  - $ax_0+by_0+c=0$ is the equation of the line

- Draw the line on the first image
  - Find two points (xa,ya) and (xb,yb) on the line, and draw a line between them
  - Let xa = -1, solve for ya
  - Let xb = +1, solve for yb

ax + by + c = 0, x = xa og xb

$$y = (-c - ax)/b$$

Visualization (continued)

- Similarly, can view the corresponding set of epipolar lines on image 1 **(second image)**
- Find the essential matrix going the other way; $\mathbf{E}' = [\mathbf{t}]_x \mathbf{R}$
  - where
    - $[\mathbf{t}]_x$ is the skew symmetric matrix corresponding to **t**
    - **t** is the translation of camera 0 with respect to camera 1; i.e., ${}^{c1}P_{c0org}$
    - **R** is rotation of camera 0 with respect to camera 1; i.e., ${}^{c1}{}_{c0}R$
- Pick a point p in the first image

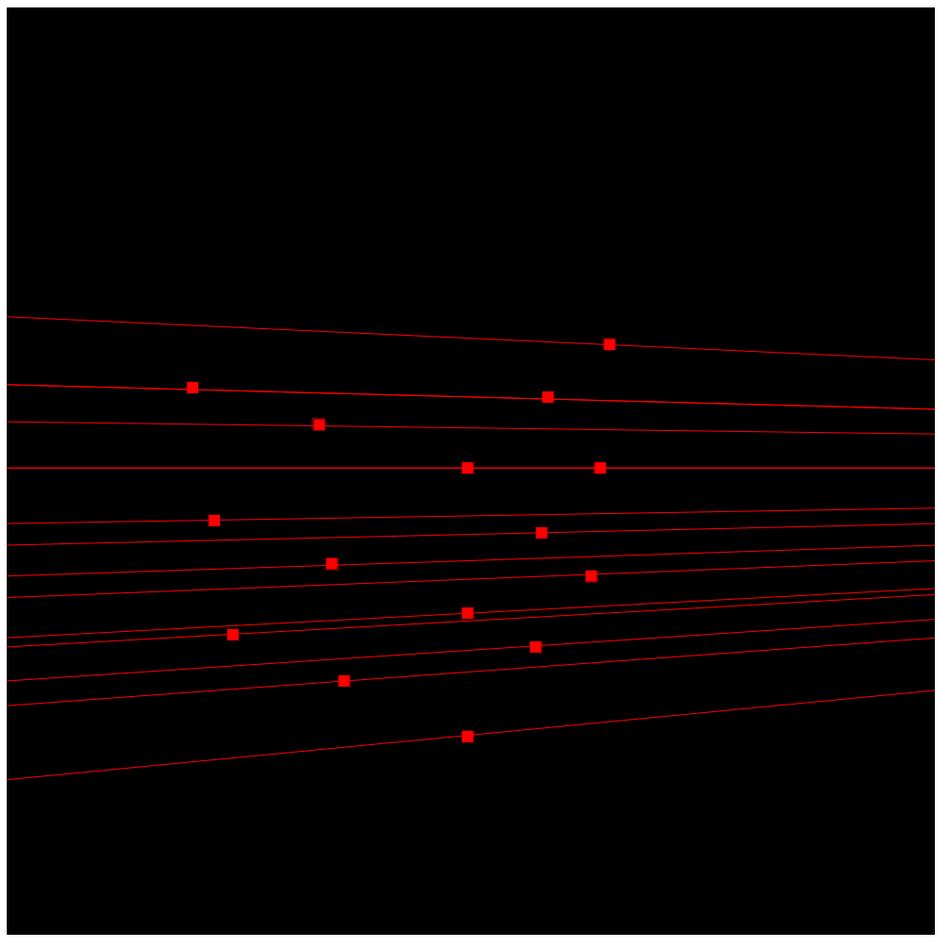
- Calculate the corresponding epipolar line in the second image, l=E'p
  - where l=(a,b,c)
  - ax+by+c=0 is the equation of the line

- Draw the line on the second image
  - Find two points (xa,ya) and (xb,yb) on the line, and draw a line between them
  - Let xa = -1, solve for ya
  - Let xb = +1, solve for yb

```
% Draw epipolar lines
for i=1:length(p2)
    figure(1);
    % The product l=E*p2 is the equation of the epipolar line corresponding
    % to p2, in the first image.  Here, l=(a,b,c), and the equation of the
    % line is ax + by + c = 0.
    l = E * p2(:,i);

    % Let's find two points on this line. First set x=-1 and solve
    % for y, then set x=1 and solve for y.
    pLine0 = [-1; (-l(3)-l(1)*(-1))/l(2); 1];
    pLine1 = [1; (-l(3)-l(1))/l(2); 1];

    % Convert from normalized to unnormalized coords
    pLine0 = Mint * pLine0;
    pLine1 = Mint * pLine1;

    line([pLine0(1) pLine1(1)], [pLine0(2) pLine1(2)], 'Color', 'r');
    pause
end
```

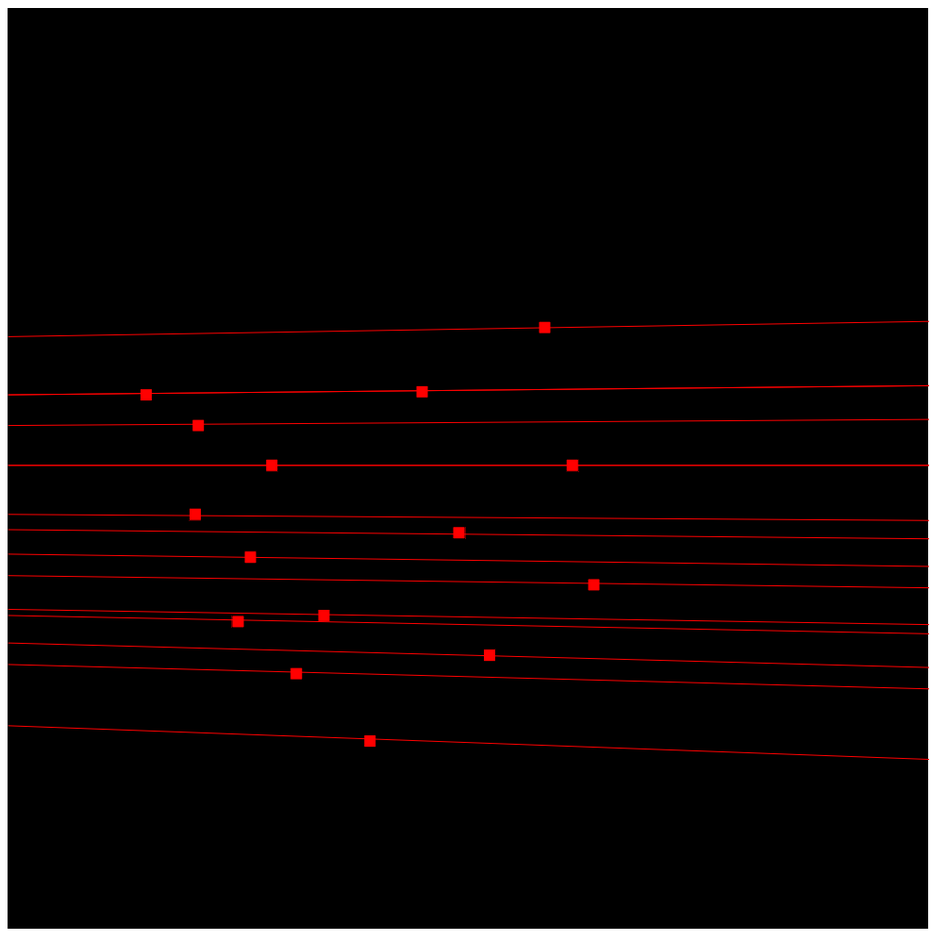
Results

View 1



Epipolar lines corresponding to points in the second image, projected onto the first image

View 2



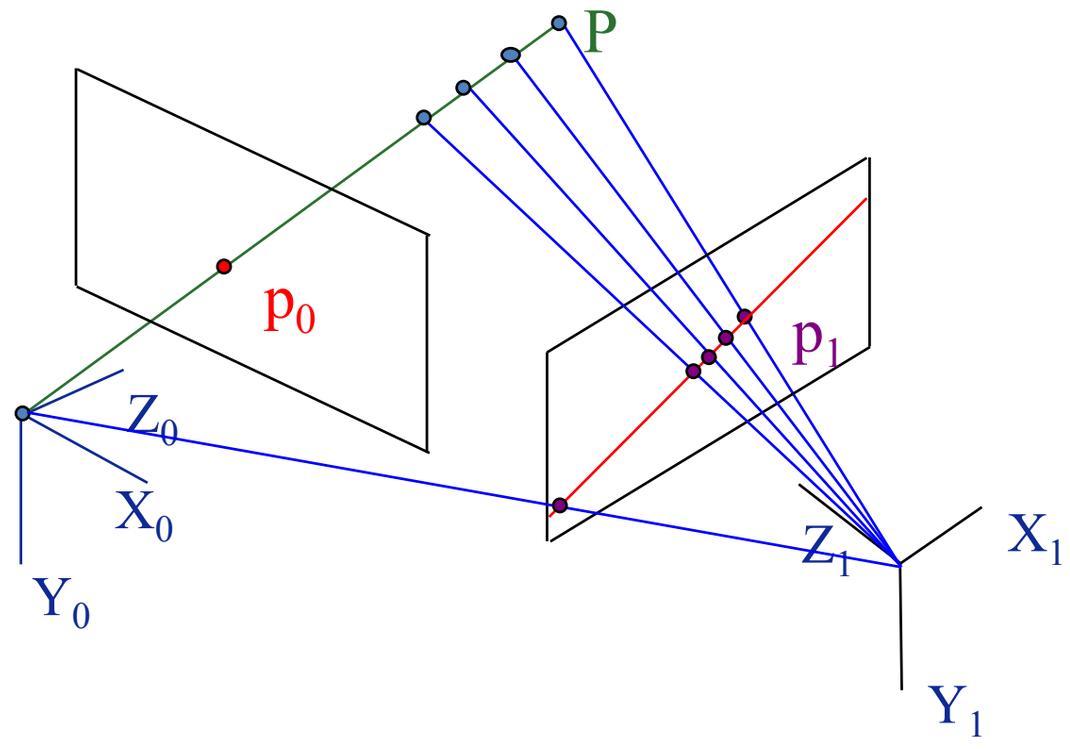
Epipolar lines corresponding to points in the first image, projected onto the second image

Structure From Motion

The Essential Matrix

- Is the matrix **E**, that relates the image of a point in one camera to its image in the other camera, given a translation and rotation

$$\mathbf{p}_0^T \, \mathbf{E} \, \mathbf{p}_1 = 0$$



- where

$$\mathbf{E} = [\mathbf{t}]_x \, \mathbf{R}$$

Recall **a** x **b** = $[\mathbf{a}_x]\mathbf{b}$, where

$$[\mathbf{a}]_\times = \begin{pmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{pmatrix}$$

- We can calculate E if we know the pose between the two views
- Or, we can calculate E from a set of known point correspondences

Calculating the Essential Matrix

- We have

$$\mathbf{p}_0^T \, \mathbf{E} \, \mathbf{p}_1 = 0$$

- We have a set of known point correspondences $\mathbf{p}_0$ and $\mathbf{p}_1$
- We have one equation for each point correspondence
- We can solve for the unknowns in **E**

- Note that **E** is a 3x3 matrix with 9 unknowns
  - It's only known up to a scale factor
  - We really only have 8 unknowns
- We need at least 8 equations – we can compute **E** from eight or more point correspondences

Calculating the Essential Matrix

- We have

$$\mathbf{p}_0^T \, \mathbf{E} \, \mathbf{p}_1 = 0$$

$$\begin{pmatrix} x_0 & y_0 & 1 \end{pmatrix} \begin{pmatrix} E_{11} & E_{12} & E_{13} \\ E_{21} & E_{22} & E_{23} \\ E_{31} & E_{32} & E_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} = 0$$

- Write out as equation

$$\begin{pmatrix} x_0 & y_0 & 1 \end{pmatrix} \begin{pmatrix} E_{11}x_1 + E_{12}y_1 + E_{13} \\ E_{21}x_1 + E_{22}y_1 + E_{13} \\ E_{31}x_1 + E_{32}y_1 + E_{33} \end{pmatrix} = 0$$

$$\begin{aligned} &E_{11}x_0x_1 + E_{12}x_0y_1 + E_{13}x_0 + \\ &E_{21}y_0x_1 + E_{22}y_0y_1 + E_{13}y_0 + \\ &E_{31}x_1 + E_{32}y_1 + E_{33} = 0 \end{aligned}$$

- Write as **A x = 0**, where **x** = (E11, E12, E13, … , E33)

$$\begin{pmatrix} x_0x_1 & x_0y_1 & x_0 & y_0x_1 & y_0y_1 & y_0 & x_1 & y_1 & 1 \end{pmatrix} \begin{pmatrix} E_{11} \\ E_{12} \\ E_{13} \\ \vdots \\ E_{33} \end{pmatrix} = 0$$

Actually, A will have one row for each point correspondence (x0,y0) – (x1,y1)

Solving for E

- We have **A x = 0**
- This is a system of homogeneous equations
- Ignoring the trivial solution **x** = 0, you can find a unique solution for **x** that gives the least squares solution for **x**; i.e., the solution that minimizes $\sum (\mathbf{p}_0^T \mathbf{E}\, \mathbf{p}_1)^2$

- It is proportional to the only zero eigenvalue of $\mathbf{A}^T\mathbf{A}$

- You can use Singular Value Decomposition to find it:

  $\mathbf{A} = \mathbf{U}\, \mathbf{D}\, \mathbf{V}^T$

- The solution **x** is the column of **V** corresponding to the only null singular value of **A**
- This is the rightmost column of **V**

Orientation

Finding E using 8-point linear algorithm

- Solve **A x** = 0 using Singular Value Decomposition (SVD):

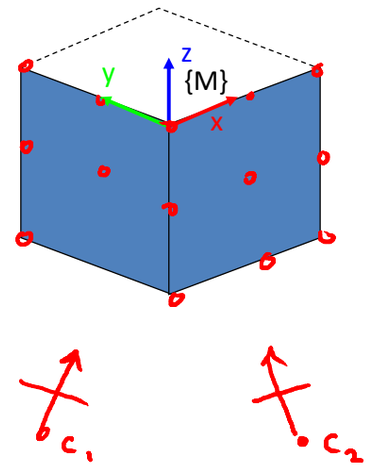
  A = U D V^T

- The solution **x** is the rightmost column of **V**

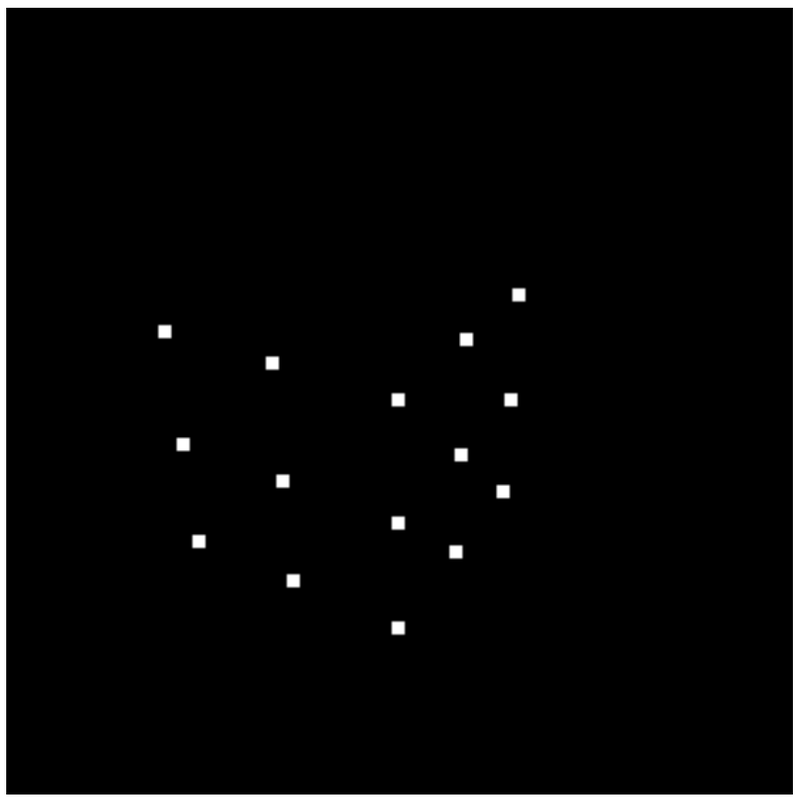
```
% Compute essential matrix E from point correspondences.
% We know that p1' E p2 = 0, where p1,p2 are the normalized image coords.
% We write out the equations in the unknowns E(i,j)
%   A x = 0
A = [p1s(1,:)'.*p2s(1,:)'   p1s(1,:)'.*p2s(2,:)'  p1s(1,:)' ...
        p1s(2,:)'.*p2s(1,:)'   p1s(2,:)'.*p2s(2,:)'  p1s(2,:)' ...
        p2s(1,:)'                p2s(2,:)'  ones(length(p1s),1)];

% The solution to Ax=0 is the singular vector of A corresponding to the
% smallest singular value; that is, the last column of V in A=UDV'
[U,D,V] = svd(A);
x = V(:,size(V,2));                    % get last column of V

% Put unknowns into a 3x3 matrix.  Transpose because Matlab's "reshape"
% uses the order E11 E21 E31 E12 ...
Escale = reshape(x,3,3)';
```

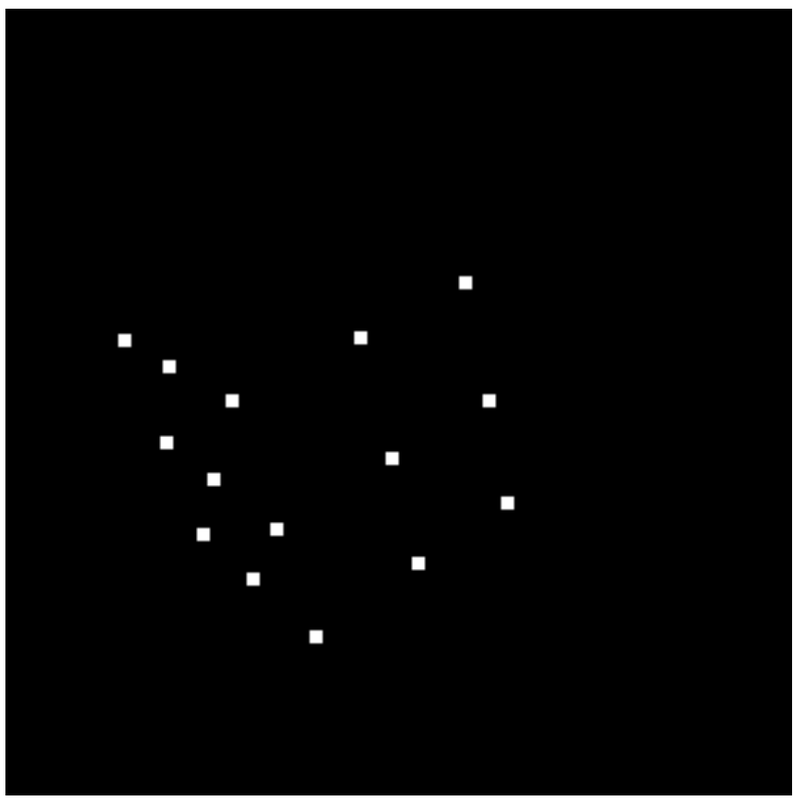
Orientation

Generate Two Synthetic Images



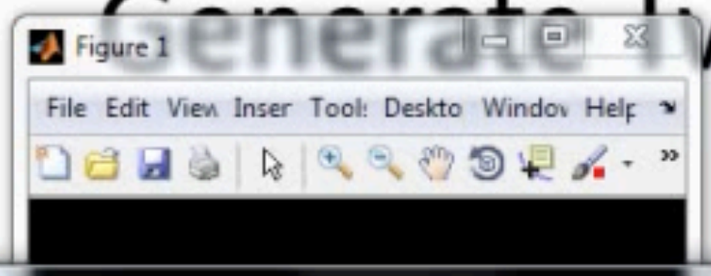
- Run program “createpts.m”
  - Create some points on the face of cube
  - Render image from two views
  - Saves results: “I1.tif, I2.tif”

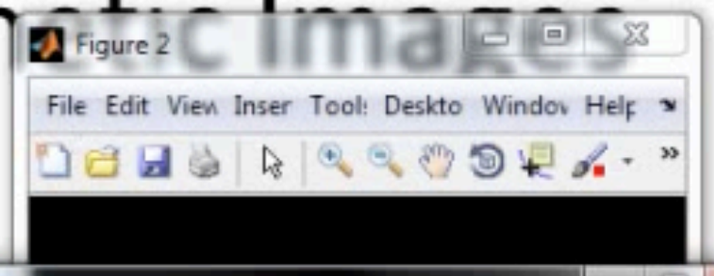




Orientation

Generate Two Synthetic Images





ATLAB 7.11.0 (R2010b)

Edit Debug Desktop Window Help

Current Folder: C:\Users\whoff\Documents\Teaching\Eggn512\2011\demos\26-StructureFromMotion

ortcuts How to Add What's New

ew to MATLAB? Watch this Video, see Demos, or read Getting Started.

```
>> createpts
Points in image 1:
  Columns 1 through 10
   61.4195  102.1798  150.0000   68.3768  106.2098  150.0000   74.3208  109.6134  150.0000  176.0870
  124.4290  136.1955  150.0000  167.2490  181.1490  197.2377  203.8325  219.1146  236.6025  127.4080
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
  Columns 11 through 15
  196.1538  174.0000  192.8571  172.2222  190.0000
  110.0296  170.7846  150.0000  207.7350  184.6410
    1.0000    1.0000    1.0000    1.0000    1.0000
Points in image 2:
  Columns 1 through 10
   45.5272   63.4568   86.9447   61.6620   80.2653  104.1468   75.7981   94.7507  118.6606  135.5451
  126.5989  136.7293  150.0000  165.9997  180.2731  198.5963  200.5196  217.7991  239.5982  125.7710
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
  Columns 11 through 15
  176.3357  147.5739  184.5258  157.8633  191.6139
  105.4355  172.3407  150.0000  212.1766  188.5687
    1.0000    1.0000    1.0000    1.0000    1.0000
True essential matrix:
         0   -1.0000         0
   -0.3615         0   -3.1415
         0    3.0000         0
>>
```

Orientation

art

OVR

```
% Create an image pair with known rotation and translation between the
% views, and corresponding image points.

clear all
close all

L = 300;        % size of image in pixels
I1 = zeros(L,L);

% Define f, u0, v0
f = L;
u0 = L/2;
v0 = L/2;

%  Create the matrix of intrinsic camera parameters
K = [ f  0  u0;
      0  f  v0;
      0  0   1];

DEG_TO_RAD = pi/180;

% Create some points on the face of a cube
P_M = [
    0   0   0   0   0   0   0   0   0   1   2   1   2   1   2;
    2   1   0   2   1   0   2   1   0   0   0   0   0   0   0;
    0   0   0   -1  -1  -1  -2  -2  -2  0   0   -1  -1  -2  -2;
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1;
    ];
NPTS = length(P_M);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define pose of model with respect to camera1
ax = 120 * DEG_TO_RAD;
ay = 0 * DEG_TO_RAD;
az = 60 * DEG_TO_RAD;

Rx = [ 1    0            0;
       0    cos(ax)     -sin(ax);
       0    sin(ax)      cos(ax) ];
Ry = [ cos(ay)    0     sin(ay);
       0          1     0;
      -sin(ay)    0     cos(ay) ];
Rz = [ cos(az)    -sin(az)   0;
       sin(az)     cos(az)   0;
       0           0         1 ];
R_m_c1 = Rx * Ry * Rz;
Pmorg_c1 = [0; 0; 5];   % translation of model wrt camera

M = [ R_m_c1 Pmorg_c1 ];    % Extrinsic camera parameter matrix

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Render image 1
p1 = M * P_M;

p1(1,:) = p1(1,:) ./ p1(3,:);
p1(2,:) = p1(2,:) ./ p1(3,:);
p1(3,:) = p1(3,:) ./ p1(3,:);

u1 = K * p1;  % Convert image points from normalized to unnormalized
for i=1:length(u1)
    x = round(u1(1,i));   y = round(u1(2,i));
    I1(y-2:y+2, x-2:x+2) = 255;
end
figure(1), imshow(I1, []), title('View 1');
pause
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Set up second view.
% Define rotation of camera1 with respect to camera2
ax = 0 * DEG_TO_RAD;
ay = -25 * DEG_TO_RAD;
az = 0;

Rx = [ 1    0            0;
       0    cos(ax)     -sin(ax);
       0    sin(ax)      cos(ax) ];
Ry = [ cos(ay)    0     sin(ay);
       0          1     0;
      -sin(ay)    0     cos(ay) ];
Rz = [ cos(az)    -sin(az)   0;
       sin(az)     cos(az)   0;
       0           0         1 ];
R_c2_c1 = Rx * Ry * Rz;

% Define translation of camera2 with respect to camera1
Pc2org_c1 = [3; 0; 1];

% Figure out pose of model wrt camera 2.
H_m_c1 = [ R_m_c1 Pmorg_c1 ;  0 0 0 1];
H_c2_c1 = [ R_c2_c1 Pc2org_c1 ;   0 0 0 1];
H_c1_c2 = inv(H_c2_c1);
H_m_c2 = H_c1_c2 * H_m_c1;

R_m_c2 = H_m_c2(1:3,1:3);
Pmorg_c2 = H_m_c2(1:3,4);

% Extrinsic camera parameter matrix
M = [ R_m_c2 Pmorg_c2 ];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Render image 2
I2 = zeros(L,L);
p2 = M * P_M;

p2(1,:) = p2(1,:) ./ p2(3,:);
p2(2,:) = p2(2,:) ./ p2(3,:);
p2(3,:) = p2(3,:) ./ p2(3,:);

% Convert image points from normalized to unnormalized
u2 = K * p2;
for i=1:length(u2)
    x = round(u2(1,i));   y = round(u2(2,i));
    I2(y-2:y+2, x-2:x+2) = 255;
end
figure(2), imshow(I2, []), title('View 2');

disp('Points in image 1:');
disp(u1);
disp('Points in image 2:');
disp(u2);
imwrite(I1, 'I1.tif');
imwrite(I2, 'I2.tif');


% This is the "true" essental matrix between the views
t = Pc2org_c1;
Etrue = [ 0 -t(3) t(2); t(3) 0 -t(1); -t(2) t(1) 0] * R_c2_c1;
disp('True essential matrix:');
disp(Etrue);
```

Orientation

Results

- True relative pose, H_c2_c1:

```
H_c2_c1 =
    0.9063         0   -0.4226    3.0000
         0    1.0000         0         0
    0.4226         0    0.9063    1.0000
         0         0         0    1.0000
```

- Corresponding points in images:

```
Points in image 1:
  Columns 1 through 10
   61.4195  102.1798  150.0000   68.3768  106.2098  150.0000   74.3208  109.6134  150.0000  176.0870
  124.4290  136.1955  150.0000  167.2490  181.1490  197.2377  203.8325  219.1146  236.6025  127.4080
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
  Columns 11 through 15
  196.1538  174.0000  192.8571  172.2222  190.0000
  110.0296  170.7846  150.0000  207.7350  184.6410
    1.0000    1.0000    1.0000    1.0000    1.0000
Points in image 2:
  Columns 1 through 10
   45.5272   63.4568   86.9447   61.6620   80.2653  104.1468   75.7981   94.7507  118.6606  135.5451
  126.5989  136.7293  150.0000  165.9997  180.2731  198.5963  200.5196  217.7991  239.5982  125.7710
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
  Columns 11 through 15
  176.3357  147.5739  184.5258  157.8633  191.6139
  105.4355  172.3407  150.0000  212.1766  188.5687
    1.0000    1.0000    1.0000    1.0000    1.0000
```

- Essential matrix:

True essential matrix:

```
         0   -1.0000         0
   -0.3615         0   -3.1415
         0    3.0000         0
```

Orientation

Observations

- Results can be unstable, due to poor numerical conditioning
- We can improve results by:
  - Preconditioning: We will first translate and scale the data points so they are centered at the origin and the average distance to the origin is √2
  - Postconditioning: The values of E are not independent. There are only five independent parameters. E must have rank=2 … we will enforce this

Orientation

Preconditioning

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Scale and translate image points so that the centroid of
% the points is at the origin, and the average distance of the points to the
% origin is equal to sqrt(2).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xn = p1(1:2,:);                 % xn is a 2xN matrix
N = size(xn,2);
t = (1/N) * sum(xn,2);          % this is the (x,y) centroid of the points
xnc = xn - t*ones(1,N);         % center the points; xnc is a 2xN matrix
dc = sqrt(sum(xnc.^2));         % dist of each new point to 0,0; dc is 1xN vector
davg = (1/N)*sum(dc);           % average distance to the origin
s = sqrt(2)/davg;               % the scale factor, so that avg dist is sqrt(2)
T1 = [s*eye(2), -s*t ; 0 0 1];
p1s = T1 * p1;

xn = p2(1:2,:);                 % xn is a 2xN matrix
N = size(xn,2);
t = (1/N) * sum(xn,2);          % this is the (x,y) centroid of the points
xnc = xn - t*ones(1,N);         % center the points; xnc is a 2xN matrix
dc = sqrt(sum(xnc.^2));         % dist of each new point to 0,0; dc is 1xN vector
davg = (1/N)*sum(dc);           % average distance to the origin
s = sqrt(2)/davg;               % the scale factor, so that avg dist is sqrt(2)
T2 = [s*eye(2), -s*t ; 0 0 1];
p2s = T2 * p2;
```

Orientation

Postconditioning

- Enforce the property that the essential matrix has only two non-zero eigenvalues, and that they are equal
- You can force this by taking the SVD of E

  $\mathbf{E = U\,S\,V^T}$

- then reconstruct E with only its first two eigenvalues

  $\mathbf{E' = U\,diag(1,1,0)\,V^T}$

- In Matlab

```
[U,D,V] = svd(Escale);
Escale = U*diag([1 1 0])*V';
```

Orientation

Undoing Preconditioning

- After computing the essential matrix **Escale**, you then have to adjust the result to undo the effect of point scaling. This can be done by **E** = **T1**^{T} **Escale T2**.

```
E = T1' * Escale * T2;      % Undo scaling
```

Orientation

Complete Code for computing Essential Matrix (1)

```
% Calculate the essential matrix.

clear all
close all

K = [ 300  0  150;   % intrinsic camera parameters
      0  300  150;
      0  0   1];

% These are the points in image 1
u1 = [
   61.4195  102.1798  150.0000   68.3768  106.2098  150.0000   74.3208  109.6134  150.0000  176.0870  196.1538  174.0000  192.8571  172.2222  190.0000;
  124.4290  136.1955  150.0000  167.2490  181.1490  197.2377  203.8325  219.1146  236.6025  127.4080  110.0296  170.7846  150.0000  207.7350  184.6410;
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000 ];

% These are the corresponding points in image 2
u2 = [
   45.5272   63.4568   86.9447   61.6620   80.2653  104.1468   75.7981   94.7507  118.6606  135.5451  176.3357  147.5739  184.5258  157.8633  191.6139;
  126.5989  136.7293  150.0000  165.9997  180.2731  198.5963  200.5196  217.7991  239.5982  125.7710  105.4355  172.3407  150.0000  212.1766  188.5687;
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000 ];

I1 = imread('I1.tif');
I2 = imread('I2.tif');
```

Read images and corresponding points

```
% Display points on the images for visualization
imshow(I1, []);
for i=1:length(u1)
    x = round(u1(1,i));     y = round(u1(2,i));
    rectangle('Position', [x-4 y-4 8 8], 'EdgeColor', 'r');
    text(x+4, y+4, sprintf('%d', i), 'Color', 'r');
end
figure, imshow(I2, []);
for i=1:length(u2)
    x = round(u2(1,i));     y = round(u2(2,i));
    rectangle('Position', [x-4 y-4 8 8], 'EdgeColor', 'r');
    text(x+4, y+4, sprintf('%d', i), 'Color', 'r');
end
```

Display images and points

-

Orientation

Complete Code for computing Essential Matrix (2)

```
% Get normalized image points
p1 = inv(K)*u1;
p2 = inv(K)*u2;
```

Normalize points

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Scale and translate image points so that the centroid of
% the points is at the origin, and the average distance of the points to the
% origin is equal to sqrt(2).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xn = p1(1:2,:);             % xn is a 2xN matrix
N = size(xn,2);
t = (1/N) * sum(xn,2);      % this is the (x,y) centroid of the points
xnc = xn - t*ones(1,N);     % center the points; xnc is a 2xN matrix
dc = sqrt(sum(xnc.^2));     % dist of each new point to 0,0; dc is 1xN vector
davg = (1/N)*sum(dc);       % average distance to the origin
s = sqrt(2)/davg;           % the scale factor, so that avg dist is sqrt(2)
T1 = [s*eye(2), -s*t ; 0 0 1];
p1s = T1 * p1;

xn = p2(1:2,:);             % xn is a 2xN matrix
N = size(xn,2);
t = (1/N) * sum(xn,2);      % this is the (x,y) centroid of the points
xnc = xn - t*ones(1,N);     % center the points; xnc is a 2xN matrix
dc = sqrt(sum(xnc.^2));     % dist of each new point to 0,0; dc is 1xN vector
davg = (1/N)*sum(dc);       % average distance to the origin
s = sqrt(2)/davg;           % the scale factor, so that avg dist is sqrt(2)
T2 = [s*eye(2), -s*t ; 0 0 1];
p2s = T2 * p2;
```

Scale and translate points

```
% Compute essential matrix E from point correspondences.
% We know that p1s' E p2s = 0, where p1s,p2s are the scaled image coords.
% We write out the equations in the unknowns E(i,j)
%   A x = 0
A = [p1s(1,:)'.*p2s(1,:)'   p1s(1,:)'.*p2s(2,:)'  p1s(1,:)' ...
        p1s(2,:)'.*p2s(1,:)'   p1s(2,:)'.*p2s(2,:)'  p1s(2,:)' ...
        p2s(1,:)'             p2s(2,:)'  ones(length(p1s),1)];

% The solution to Ax=0 is the singular vector of A corresponding to the
% smallest singular value; that is, the last column of V in A=UDV'
[U,D,V] = svd(A);
x = V(:,size(V,2));                  % get last column of V

% Put unknowns into a 3x3 matrix.  Transpose because Matlab's "reshape"
% uses the order E11 E21 E31 E12 ...
Escale = reshape(x,3,3)';
```

Compute E

```
% Force rank=2 and equal eigenvalues
[U,D,V] = svd(Escale);
Escale = U*diag([1 1 0])*V';
```

Force E to have rank 2

```
% Undo scaling
E = T1' * Escale * T2;
```

Undo scaling and translation

```
disp('Calculated essential matrix:');
disp(E);

save E
```

Orientation

Results

- Run program “essential.m”
  - This inputs the corresponding points, and calculates the essential matrix
- Verify that calculated essential matrix equals the “true” essential matrix (to within a scale factor)

True essential matrix:

0	-1.0000	0
-0.3615	0	-3.1415
0	3.0000	0

Calculated essential matrix:

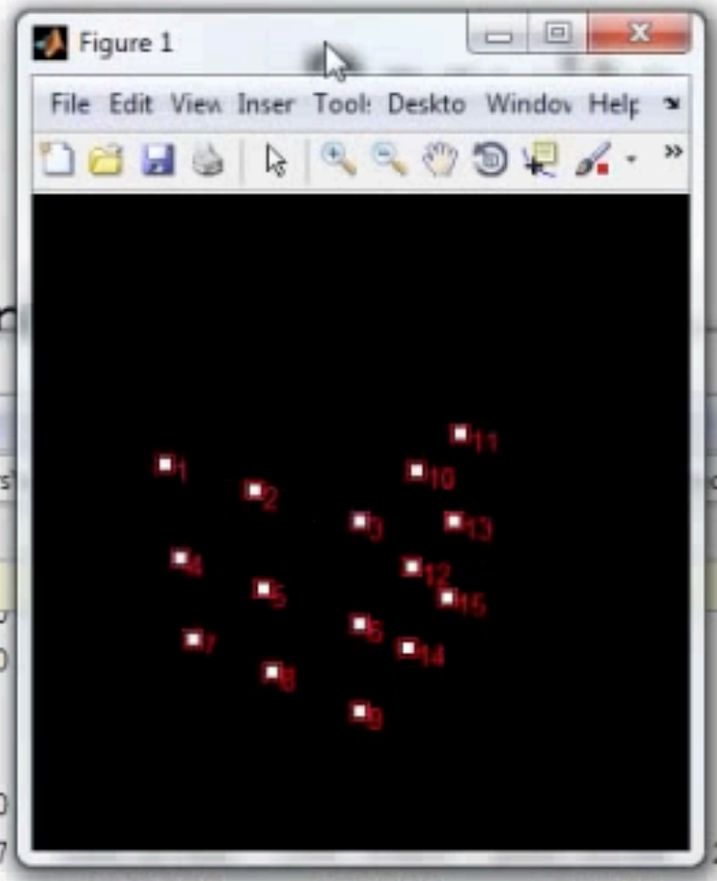
0.0001	2.4639	0.0010
0.8929	0.0834	7.7585
-0.0004	-7.3917	-0.0031

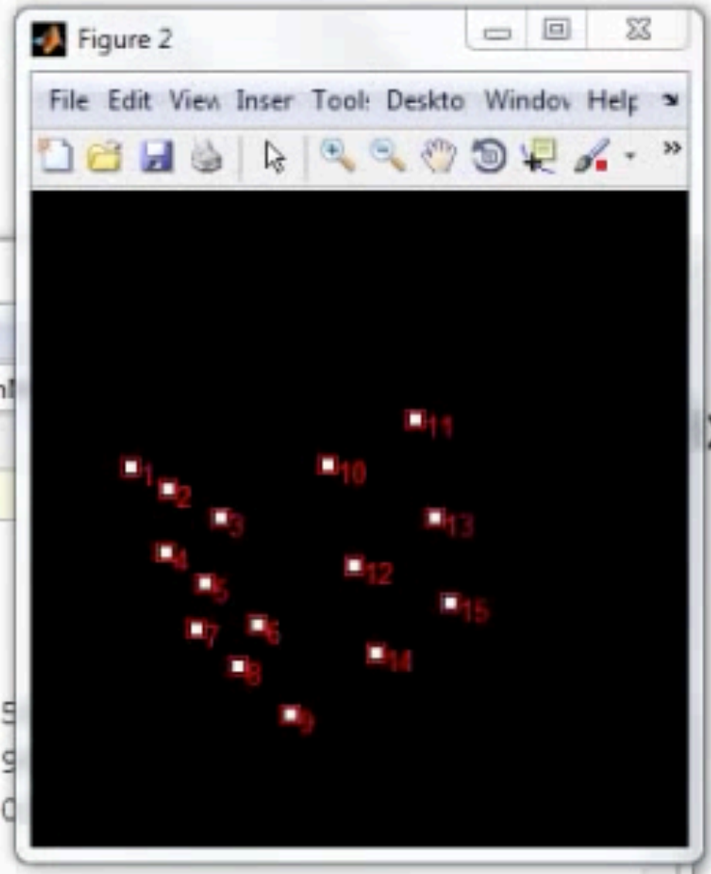
Scaled essential matrix:

-0.0000	-1.0000	-0.0004
-0.3624	-0.0338	-3.1489
0.0001	3.0000	0.0013

Orientation

- Run program "esser





ATLAB 7.11.0 (R2010b)

Edit Debug Desktop Window Help

Current Folder: C:\Users

ctureFrom

ortcuts How to Add What's New

ew to MATLAB? Watch this Video, see Demos, or read Getting Started.

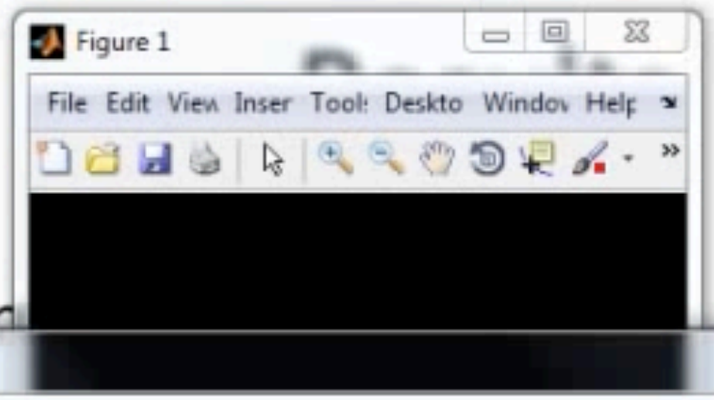
```
   1.0000    1.0000    1.0000    1.0000
Points in image 2:
  Columns 1 through 10
   45.5272   63.4568   86.9447   61.6620                                   94.75
  126.5989  136.7293  150.0000  165.9997                                  217.79
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.00
  Columns 11 through 15
  176.3357  147.5739  184.5258  157.8633  191.6139
  105.4355  172.3407  150.0000  212.1766  188.5687
    1.0000    1.0000    1.0000    1.0000    1.0000
True essential matrix:
         0   -1.0000         0
   -0.3615         0   -3.1415
         0    3.0000         0
>>
>>
>>
>> essential
Calculated essential matrix:
   -0.0001   -2.4639   -0.0010
   -0.8929   -0.0834   -7.7585
    0.0004    7.3917    0.0031
>>
```

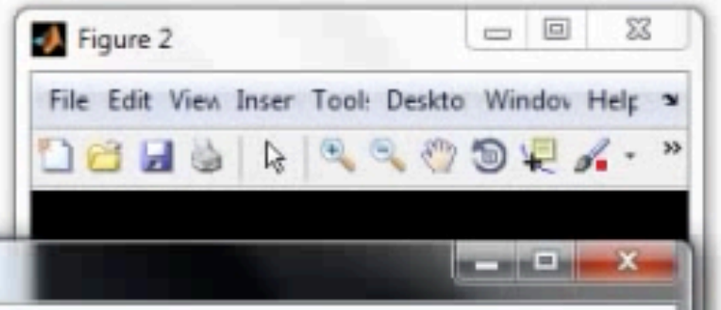
art

OVR

Orientation

- Run program "esser





ATLAB 7.11.0 (R2010b)

Edit Debug Desktop Window Help

Current Folder: C:\Users\whoff\Documents\Teaching\Eggn512\2011\demos\26-StructureFromMotion

ortcuts How to Add What's New

ew to MATLAB? Watch this Video, see Demos, or read Getting Started.

```
   110.0290  170.7846  150.0000  207.7330  184.6410
    1.0000    1.0000    1.0000    1.0000    1.0000
Points in image 2:
  Columns 1 through 10
   45.5272   63.4568   86.9447   61.6620   80.2653  104.1468   75.7981   94.7507  118.6606  135.5451
  126.5989  136.7293  150.0000  165.9997  180.2731  198.5963  200.5196  217.7991  239.5982  125.7710
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
  Columns 11 through 15
  176.3357  147.5739  184.5258  157.8633  191.6139
  105.4355  172.3407  150.0000  212.1766  188.5687
    1.0000    1.0000    1.0000    1.0000    1.0000
True essential matrix:
         0   -1.0000         0
   -0.3615         0   -3.1415
         0    3.0000         0
>>
>>
>>
>> essential
Calculated essential matrix:
   -0.0001   -2.4639   -0.0010
   -0.8929   -0.0834   -7.7585
    0.0004    7.3917    0.0031
>>
```

Orientation

art

OVR

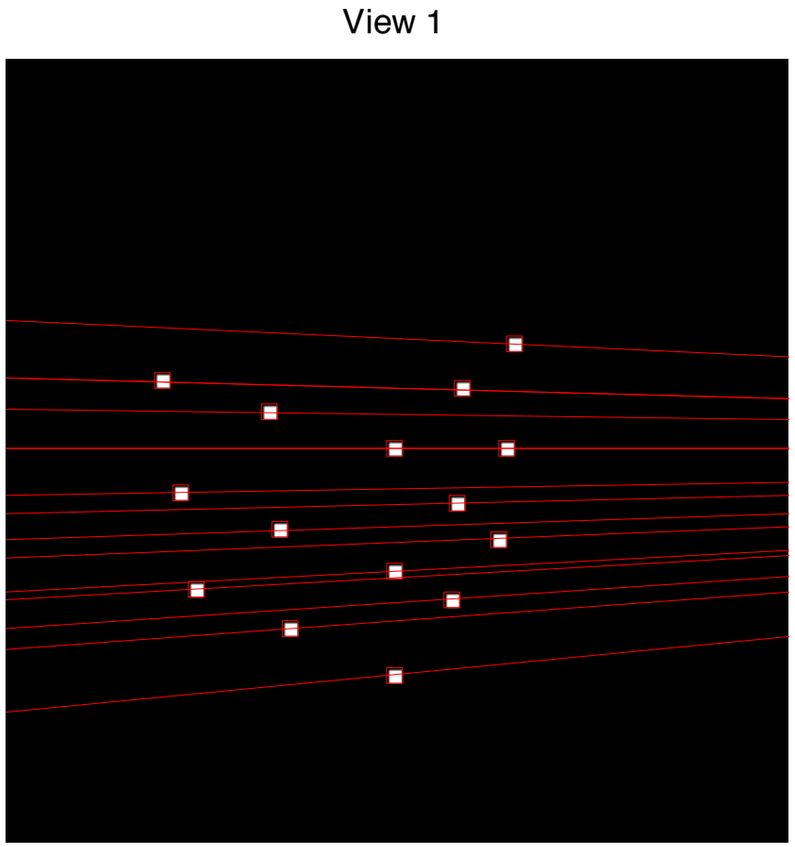
Visualization of epipolar lines

- Draw epipolar lines to verify that corresponding points lie on these lines
- Representation of lines
  - A line ($ax + by + c = 0$) is represented by the homogeneous coordinates
    $\mathbf{l} = (a,b,c)^T$
  - A point $\mathbf{p}$ lies on the line $\mathbf{l}$ if and only if $\mathbf{p}^T \mathbf{l} = 0$

- Epipolar lines
  - $\mathbf{E}\mathbf{p}_1$ is the epipolar line in the first view corresponding to $\mathbf{p}_1$ in the second view
  - $\mathbf{E}^T\mathbf{p}_0$ is the epipolar line in the second view corresponding to $\mathbf{p}_0$ in the first view

Example

- Run program “drawepipolar.m”
  - This inputs a pair of images, a set of corresponding points, and an essential matrix
  - It draws epipolar lines in the images



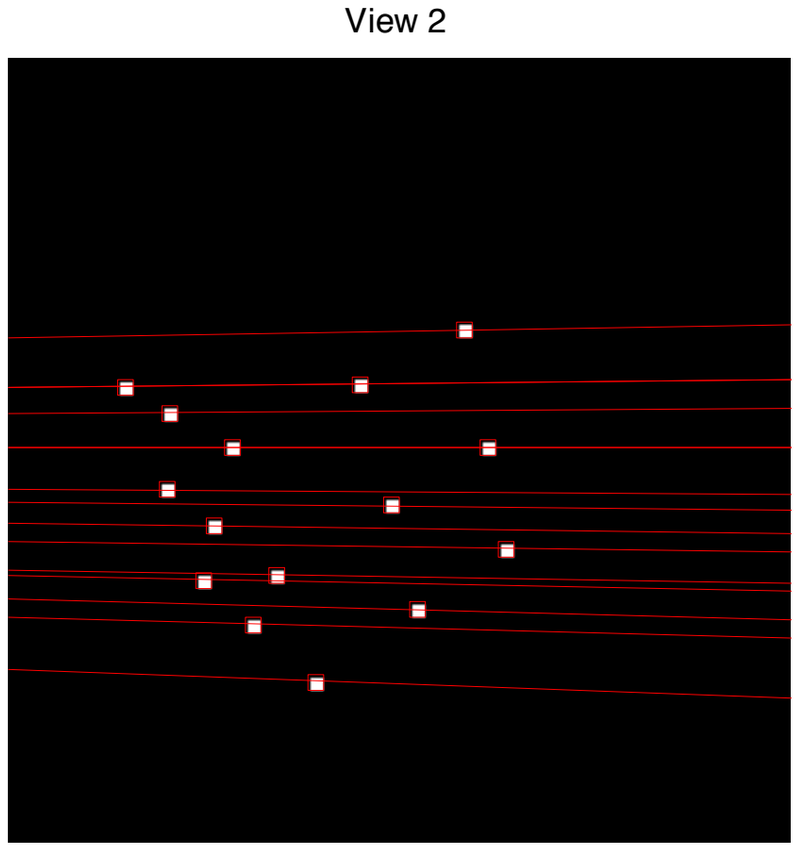
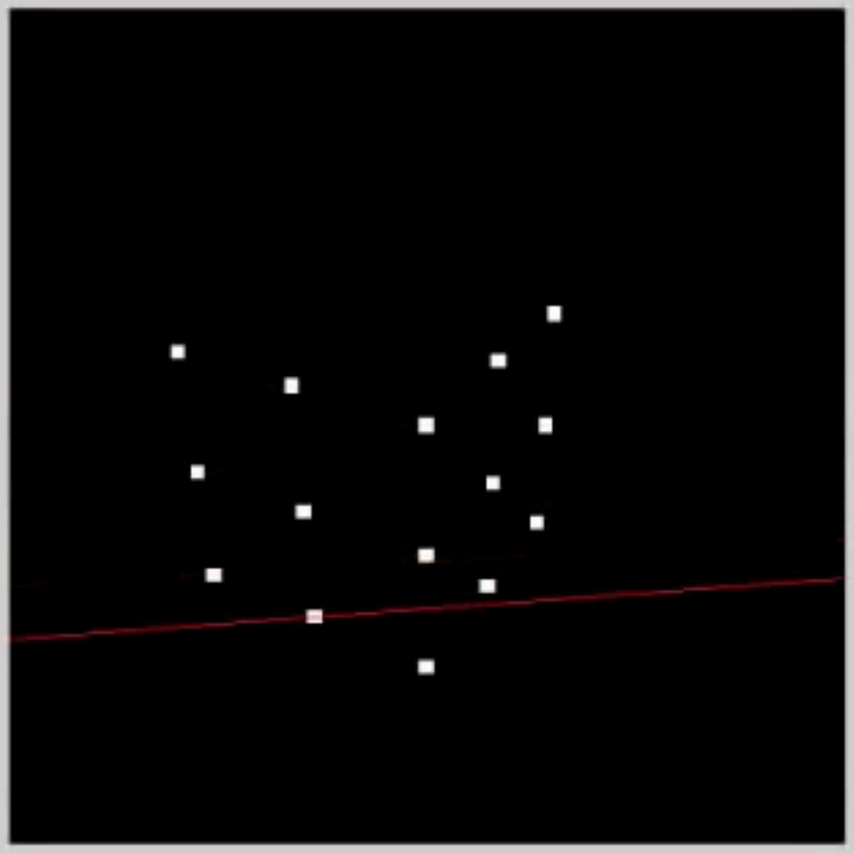
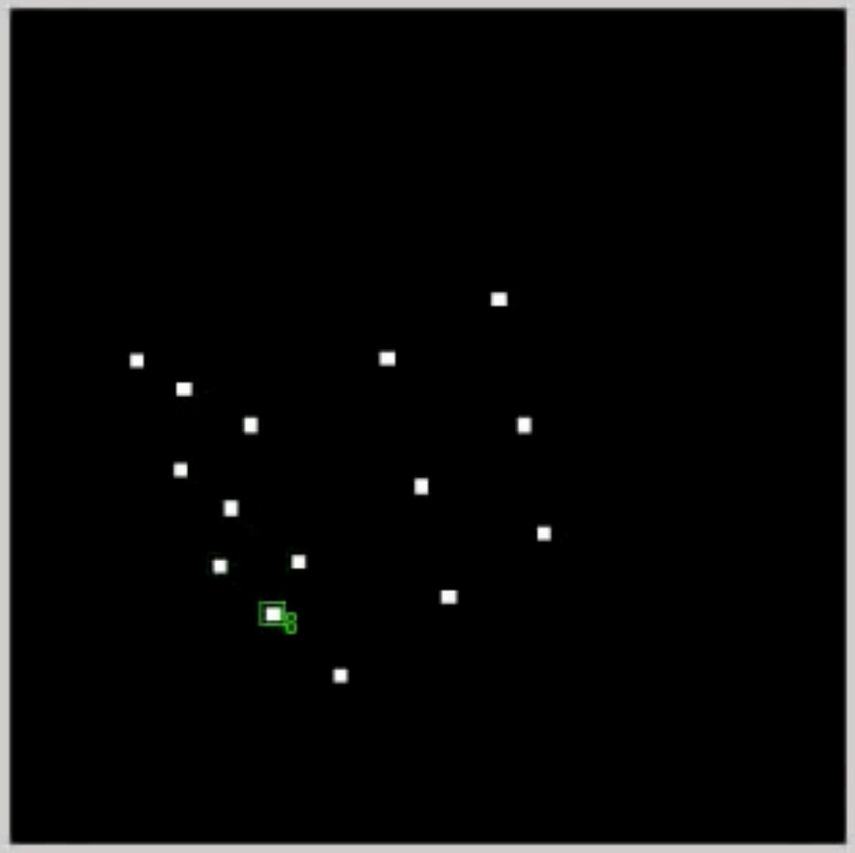


Figure 1

File Edit View Insert Tools Desktop Window Help





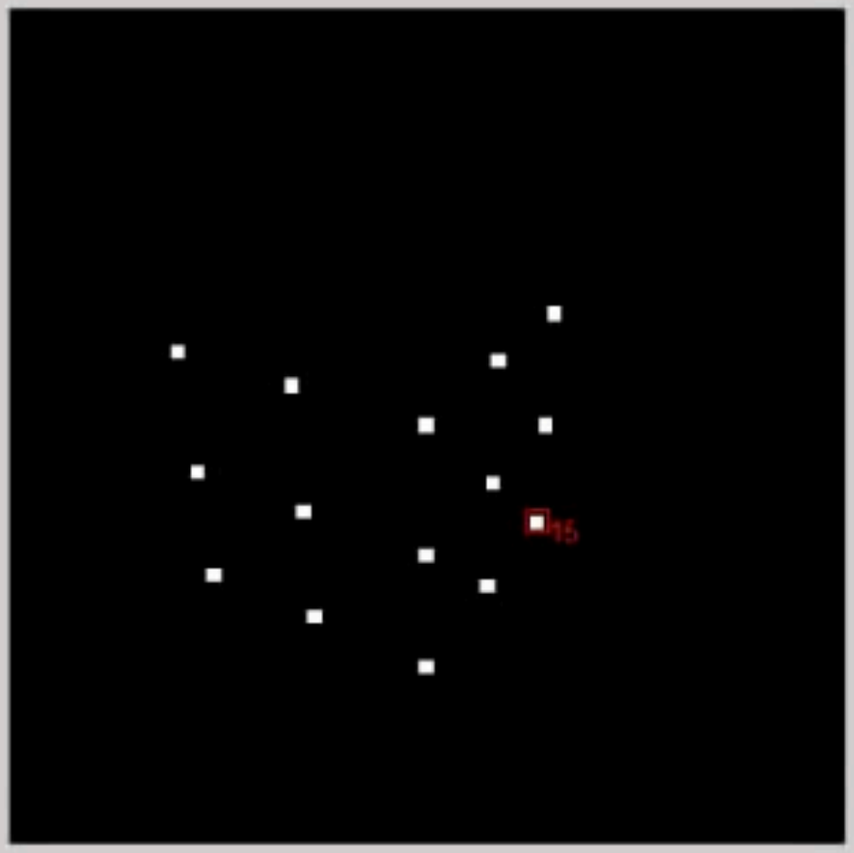
>> drawepipolar

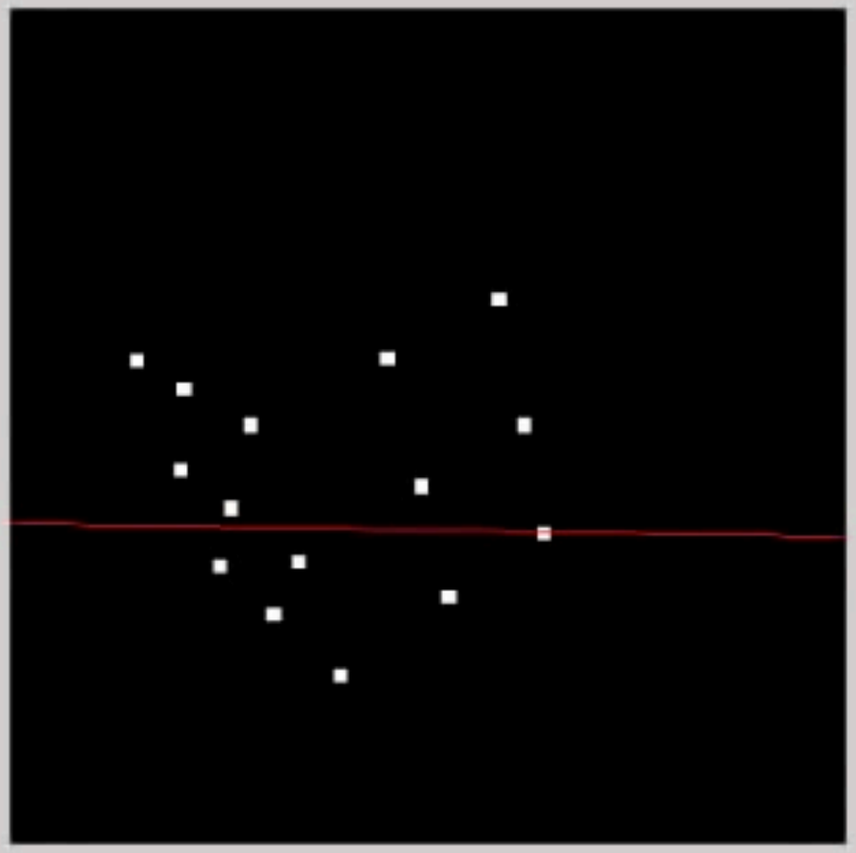
Paused: Press any key

OVR

Figure 1

File Edit View Insert Tools Desktop Window Help





```
>> drawepipolar
>>
```

Recovering Motion Parameters (advanced)

- Once you have essential matrix **E**, you can recover the relative motion between cameras
- Recall that the essential matrix is made up of the translation and rotation matrices; ie., $\mathbf{E} = [\mathbf{t}]_x \mathbf{R}$
- We can extract the translation and rotation by taking SVD of **E** again, $\mathbf{E} = \mathbf{U}\,\mathbf{D}\,\mathbf{V}^T$
- Then form the following combinations:
  - **t** is either **u3** or –**u3**, where **u3** is the third (last) column of **U**
  - **R** is either $\mathbf{U}\,\mathbf{W}\,\mathbf{V}^T$ or $\mathbf{U}\,\mathbf{W}^T\,\mathbf{V}^T$
- where

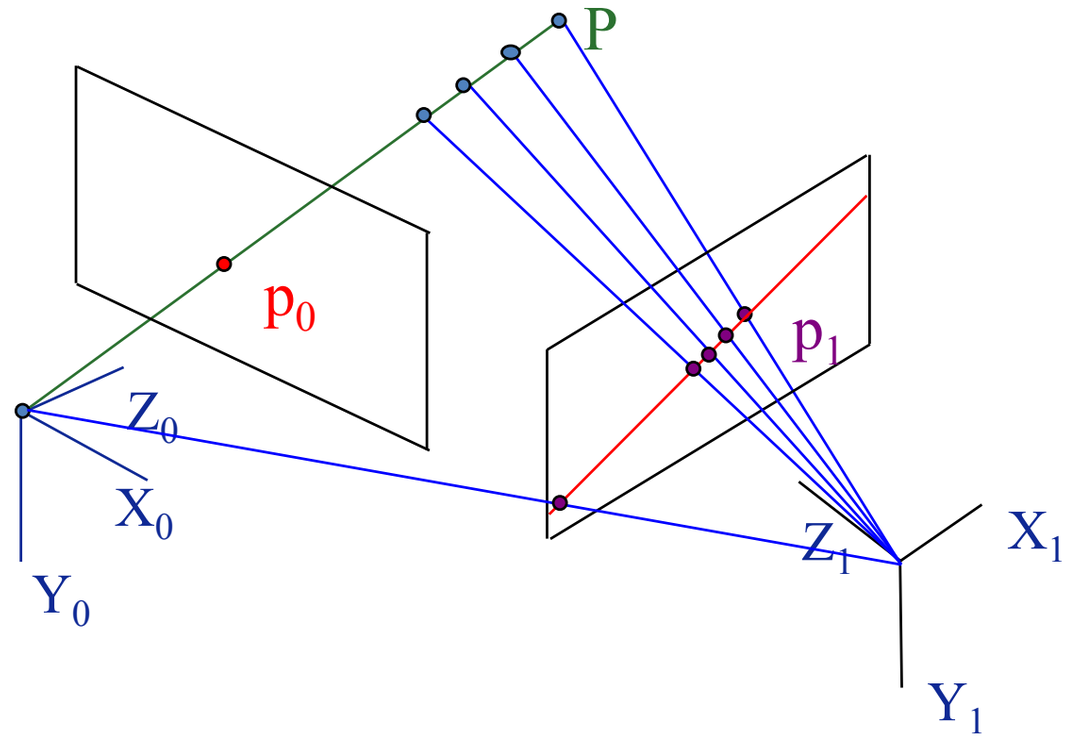
$$\mathbf{W} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Fundamental Matrix

Recall the Essential Matrix

- Is the matrix **E**, that relates the image of a point in one camera to its image in the other camera, given a translation and rotation

$$\mathbf{p}_0^T \, \mathbf{E} \, \mathbf{p}_1 = 0$$



- where

$$\mathbf{E} = [\mathbf{t}]_x \, \mathbf{R}$$

- and
  - $\mathbf{p}_0$, $\mathbf{p}_1$ are corresponding points (normalized image coordinates)

Fundamental Matrix

- To work with the essential matrix we have to know the intrinsic camera parameter matrix **K**
  - We use $\mathbf{p}_0$, $\mathbf{p}_1$ which are normalized image coordinates (i.e., $x = X/Z$, $y = Y/Z$)
  - We find normalized image coords using $\mathbf{p} = \mathbf{K}^{-1}\,\mathbf{u}$, where **u** are the un-normalized image coords

- If we don't know the intrinsic parameter matrix
  - all we have is the un-normalized image points
  - we can still relate the views
  - We use the fundamental matrix **F**

Fundamental Matrix

- We have

$$\mathbf{p}_0^T \mathbf{E} \mathbf{p}_1 = 0$$

- Let

$$\mathbf{p}_1 = \mathbf{K}^{-1}\mathbf{u}_1$$

$$\mathbf{p}_0^T = \left(\mathbf{K}^{-1}\mathbf{u}_0\right)^T = \mathbf{u}_0^T \mathbf{K}^{-T}$$

- Then

$$\mathbf{u}_0^T \mathbf{K}^{-T} \mathbf{E} \mathbf{K}^{-1} \mathbf{u}_1 = 0$$

- or

$$\mathbf{u}_0^T \mathbf{F} \mathbf{u}_1 = 0$$

- where F is the fundamental matrix $\mathbf{F} = \mathbf{K}^{-T} \mathbf{E} \mathbf{K}^{-1}$

- Note
  - **F** is defined in terms of pixel coordinates
  - You can still reconstruct the epipolar lines using **F**

Fundamental Matrix

Known vs. unknown intrinsic camera matrix K
Normalized (p) vs. un-normalized (u) image coordinates

- We have

$$\mathbf{p}_0^T \mathbf{E} \mathbf{p}_1 = 0$$

- Let

$$\mathbf{p}_1 = \mathbf{K}^{-1}\mathbf{u}_1$$

$$\mathbf{p}_0^T = \left(\mathbf{K}^{-1}\mathbf{u}_0\right)^T = \mathbf{u}_0^T \mathbf{K}^{-T}$$

- Then

$$\mathbf{u}_0^T \mathbf{K}^{-T} \mathbf{E} \mathbf{K}^{-1} \mathbf{u}_1 = 0$$

- or

$$\mathbf{u}_0^T \mathbf{F} \mathbf{u}_1 = 0$$

- where F is the fundamental matrix

$$\mathbf{F} = \mathbf{K}^{-T} \mathbf{E} \mathbf{K}^{-1}$$

- Recall perspective projection is: $x = f\,X/Z + cx$, $y = f\,Y/Z + cy$
  - If f=1 and (cx,cy) = (0,0), then this is the image projection of the point
  - As we will see later, it is often useful to consider the projection of a point as if it were taken by a camera with f=1 and (cx,cy) = (0,0)
  - These are called "normalized image coordinates"

$$\mathbf{K} = \begin{pmatrix} f/s_x & 0 & c_x \\ 0 & f/s_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \quad \text{or} \quad \mathbf{K} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

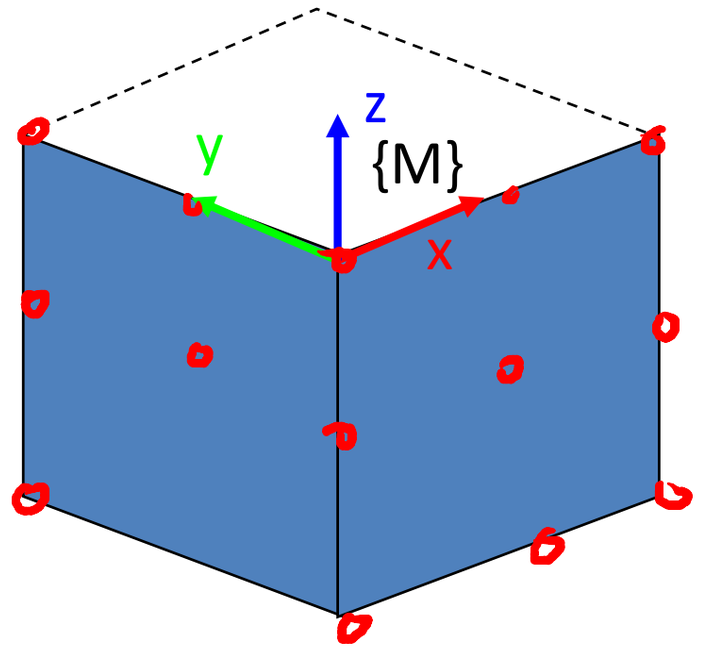
- Note
  - **F** is defined in terms of pixel coordinates
  - You can still reconstruct the epipolar lines using **F**

Example – Create a Scene

- Create some points on the face of cube
- Render image from two views

- Let pose of cube with respect to camera 1 be

  ax=120°, ay=0°, az=60°, tx=3, ty=0, tz=0



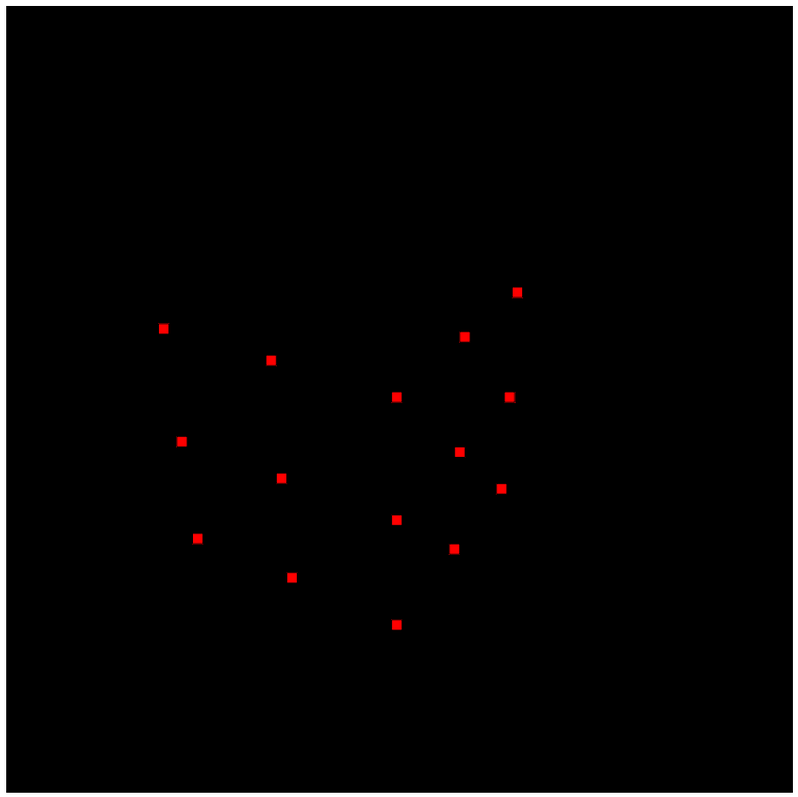
- Let pose of camera 2 with respect to camera 1 be

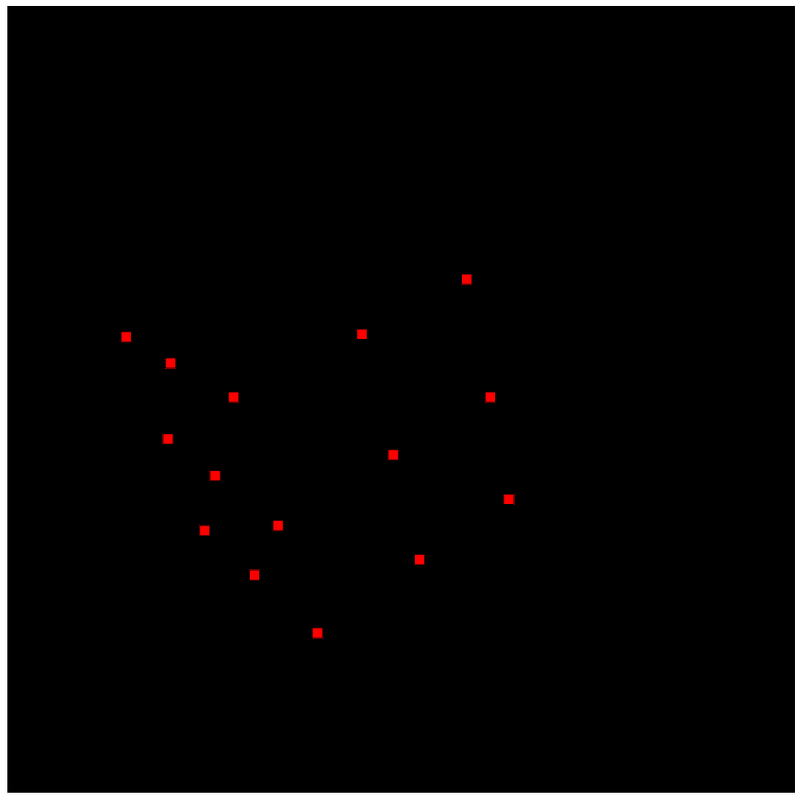
  ax=0°, ay=-25°, az=0°, tx=3, ty=0, tz=1

- Assume XYZ fixed angles

Orientation

- The Matlab code to create these points is the same as used earlier in the lecture on the essential matrix





```
% These are the points in image 1
u1 = [
   61.4195  102.1798  150.0000   68.3768  106.2098  150.0000   74.3208  109.6134  150.0000  176.0870  196.1538  174.0000  192.8571  172.2222  190.0000;
  124.4290  136.1955  150.0000  167.2490  181.1490  197.2377  203.8325  219.1146  236.6025  127.4080  110.0296  170.7846  150.0000  207.7350  184.6410;
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000 ];

% These are the corresponding points in image 2
u2 = [
   45.5272   63.4568   86.9447   61.6620   80.2653  104.1468   75.7981   94.7507  118.6606  135.5451  176.3357  147.5739  184.5258  157.8633  191.6139;
  126.5989  136.7293  150.0000  165.9997  180.2731  198.5963  200.5196  217.7991  239.5982  125.7710  105.4355  172.3407  150.0000  212.1766  188.5687;
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000 ];
```

Orientation

Ground truth for F

- Calculating the fundamental matrix using the known R, t, K:

```
E = [ 0 -t(3) t(2); t(3) 0 -t(1); -t(2) t(1) 0] * R_c2_c1;

% Fundamental matrix
F = inv(K)'*E*inv(K);
disp('True F:'); disp(F);
```

- Results

True F:

0	-0.0000	0.0017
-0.0000	0	-0.0099
0.0006	0.0117	-0.2696

Orientation

Solving for F

- We solve for F using the same methods as we used to solve for E
  - Except the corresponding points are in un-normalized coordinates
- We have

$$\mathbf{u}_0^T \, \mathbf{F} \, \mathbf{u}_1 = 0$$

$$\begin{pmatrix} x_0 & y_0 & 1 \end{pmatrix} \begin{pmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} = 0$$

- Write as **A x = 0**, where **x** = (F11, F12, F13, … , F33)

$$\begin{pmatrix} x_0 x_1 & x_0 y_1 & x_0 & y_0 x_1 & y_0 y_1 & y_0 & x_1 & y_1 & 1 \end{pmatrix} \begin{pmatrix} F_{11} \\ F_{12} \\ F_{13} \\ \vdots \\ F_{33} \end{pmatrix} = 0$$

Solving for F

- We have **A x = 0**
  - This is a system of homogeneous equations
  - We solve using singular value decomposition

- As we did earlier, we will do:
  - Preconditioning: We will first translate and scale the data points so they are centered at the origin and the average distance to the origin is √2
  - Postconditioning: The values of **F** are not independent. There are only five independent parameters. **F** must have rank=2 … we will enforce this

Orientation

Complete Code for computing Fundamental Matrix (1)

```
% Calculate the essential matrix.

clear all
close all

K = [ 300  0  150;   % intrinsic camera parameters
      0  300  150;
      0  0   1];

% These are the points in image 1
u1 = [
   61.4195  102.1798  150.0000   68.3768  106.2098  150.0000   74.3208  109.6134  150.0000  176.0870  196.1538  174.0000  192.8571  172.2222  190.0000;
  124.4290  136.1955  150.0000  167.2490  181.1490  197.2377  203.8325  219.1146  236.6025  127.4080  110.0296  170.7846  150.0000  207.7350  184.6410;
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000 ];

% These are the corresponding points in image 2
u2 = [
   45.5272   63.4568   86.9447   61.6620   80.2653  104.1468   75.7981   94.7507  118.6606  135.5451  176.3357  147.5739  184.5258  157.8633  191.6139;
  126.5989  136.7293  150.0000  165.9997  180.2731  198.5963  200.5196  217.7991  239.5982  125.7710  105.4355  172.3407  150.0000  212.1766  188.5687;
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000 ];

I1 = imread('I1.tif');
I2 = imread('I2.tif');
```

Read images and corresponding points

```
% Display points on the images for visualization
imshow(I1, []);
for i=1:length(u1)
    x = round(u1(1,i));     y = round(u1(2,i));
    rectangle('Position', [x-4 y-4 8 8], 'EdgeColor', 'r');
    text(x+4, y+4, sprintf('%d', i), 'Color', 'r');
end
figure, imshow(I2, []);
for i=1:length(u2)
    x = round(u2(1,i));     y = round(u2(2,i));
    rectangle('Position', [x-4 y-4 8 8], 'EdgeColor', 'r');
    text(x+4, y+4, sprintf('%d', i), 'Color', 'r');
end
```

Display images and points

-

Orientation

Complete Code for computing Fundamental Matrix (2)

```
% Get unnormalized image points
p1 = u1;
p2 = u2;
```

Don't normalize points

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Scale and translate image points so that the centroid of
% the points is at the origin, and the average distance of the points to the
% origin is equal to sqrt(2).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xn = p1(1:2,:);             % xn is a 2xN matrix
N = size(xn,2);
t = (1/N) * sum(xn,2);      % this is the (x,y) centroid of the points
xnc = xn - t*ones(1,N);     % center the points; xnc is a 2xN matrix
dc = sqrt(sum(xnc.^2));     % dist of each new point to 0,0; dc is 1xN vector
davg = (1/N)*sum(dc);       % average distance to the origin
s = sqrt(2)/davg;           % the scale factor, so that avg dist is sqrt(2)
T1 = [s*eye(2), -s*t ; 0 0 1];
p1s = T1 * p1;

xn = p2(1:2,:);             % xn is a 2xN matrix
N = size(xn,2);
t = (1/N) * sum(xn,2);      % this is the (x,y) centroid of the points
xnc = xn - t*ones(1,N);     % center the points; xnc is a 2xN matrix
dc = sqrt(sum(xnc.^2));     % dist of each new point to 0,0; dc is 1xN vector
davg = (1/N)*sum(dc);       % average distance to the origin
s = sqrt(2)/davg;           % the scale factor, so that avg dist is sqrt(2)
T2 = [s*eye(2), -s*t ; 0 0 1];
p2s = T2 * p2;
```

Scale and translate points

```
% Compute fundamental matrix F from point correspondences.
% We know that p1s' F p2s = 0, where p1s,p2s are the scaled image coords.
% We write out the equations in the unknowns F(i,j)
%   A x = 0
A = [p1s(1,:)'.*p2s(1,:)'   p1s(1,:)'.*p2s(2,:)'  p1s(1,:)' ...
        p1s(2,:)'.*p2s(1,:)'   p1s(2,:)'.*p2s(2,:)'  p1s(2,:)' ...
        p2s(1,:)'             p2s(2,:)'  ones(length(p1s),1)];

% The solution to Ax=0 is the singular vector of A corresponding to the
% smallest singular value; that is, the last column of V in A=UDV'
[U,D,V] = svd(A);
x = V(:,size(V,2));                  % get last column of V

% Put unknowns into a 3x3 matrix.  Transpose because Matlab's "reshape"
% uses the order F11 F21 F31 F12 ...
Fscale = reshape(x,3,3)';
```

Compute F

```
% Force rank=2
[U,D,V] = svd(Fscale);
Fscale = U*diag([D(1,1) D(2,2) 0])*V';
```

Force F to have rank 2

```
% Undo scaling
F = T1' * Fscale * T2;
```

Undo scaling and translation

```
disp('Calculated fundamental matrix:');
disp(F);

save F
```

Orientation

Results

- Run program “fundamental.m”
  - This inputs the corresponding points, and calculates the fundamental matrix
- Verify that calculated fundamental matrix equals the “true” fundamental matrix (to within a scale factor)

True F:

0	-0.0000	0.0017
-0.0000	0	-0.0099
0.0006	0.0117	-0.2696

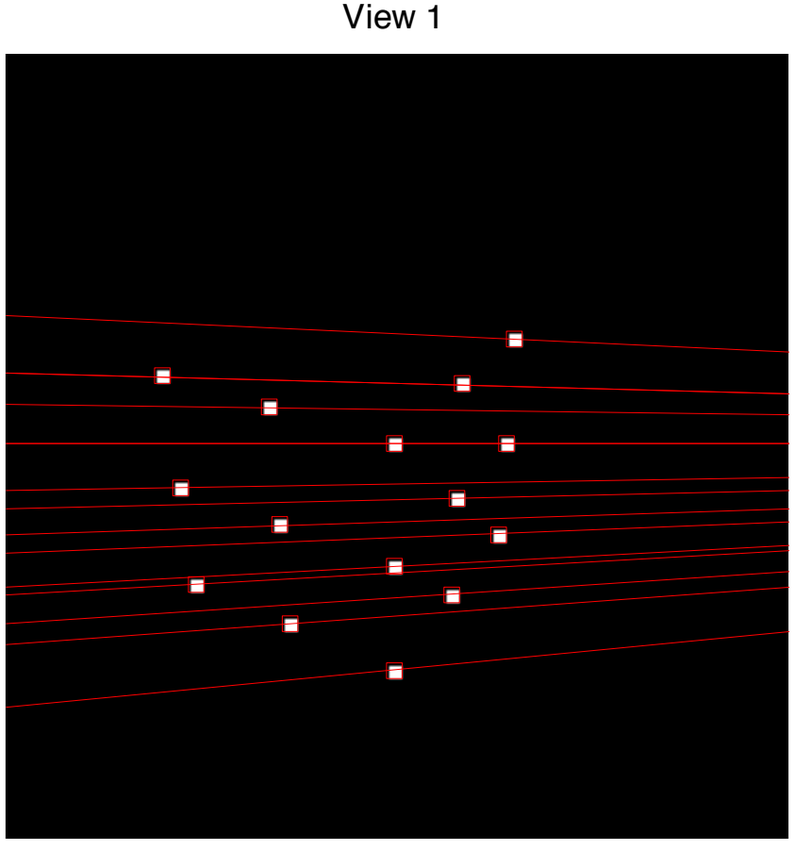
Calculated fundamental matrix:

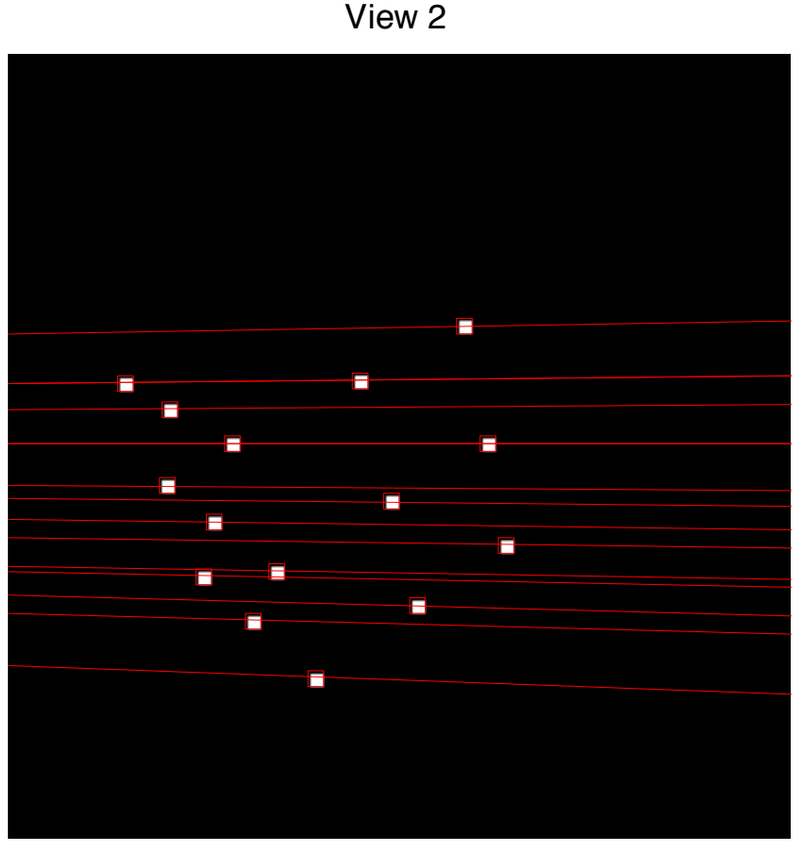
-0.0000	-0.0000	0.0029
-0.0000	0.0000	-0.0172
0.0011	0.0203	-0.4703

Orientation

Epipolar Lines

- Run program “drawepipolarFund.m”
  - This inputs a pair of images, a set of corresponding points, and a fundamental matrix
  - It draws epipolar lines in the images





Orientation

Content removed

- As the DL-based CV part was extended this year some of the other parts had to be reduced:
  - **Optical flow**
    - DL-based methods exists
  - **Tracking, Kalman filters and particle filters**
    - DL-based methods starting to emerge, e.g. LSTM-KF