3D Geometry and Camera Calibration

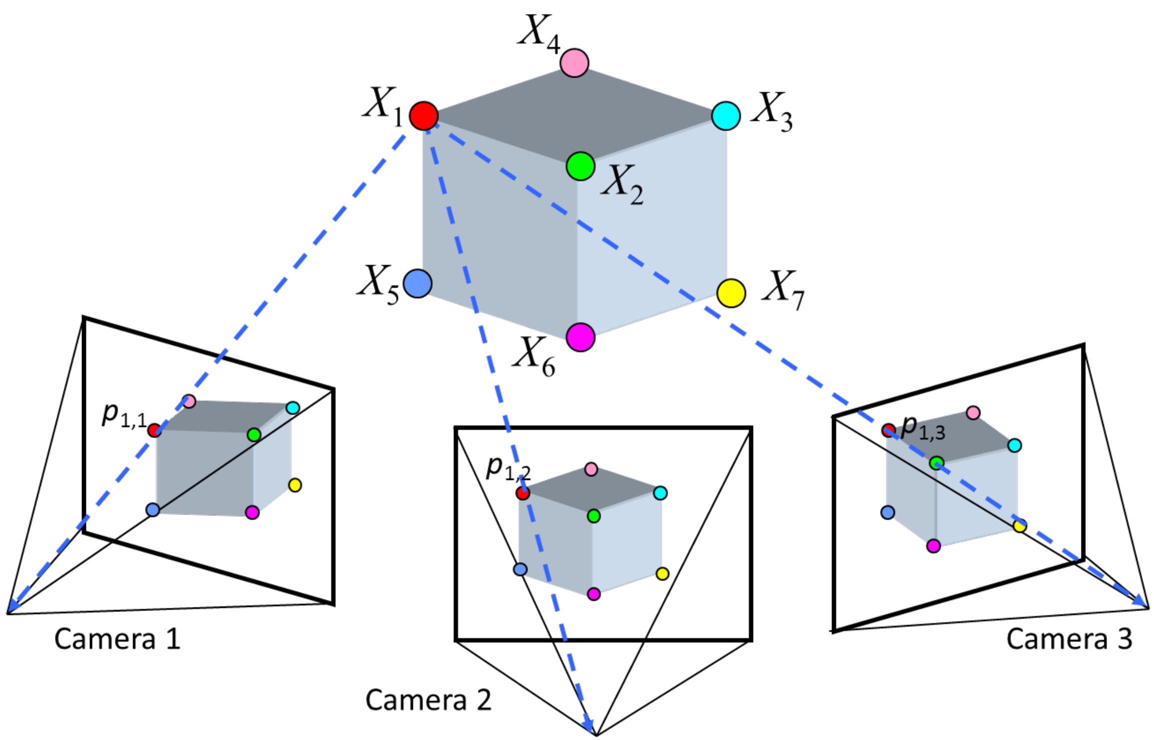
COS 429: Computer Vision

PRINCETON UNIVERSITY

Acknowledgments: T. Funkhouser,, N. Snavely

Point Correspondences

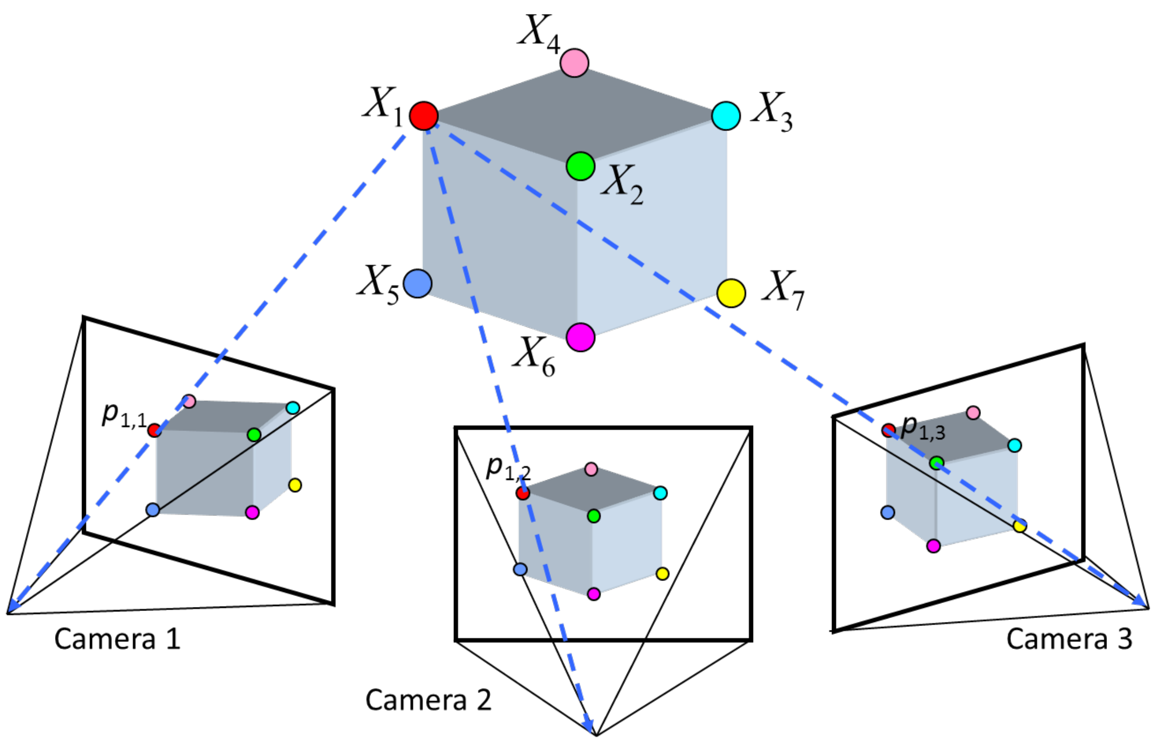
- What can we figure out given correspondences?



[Snavely]

Triangulation

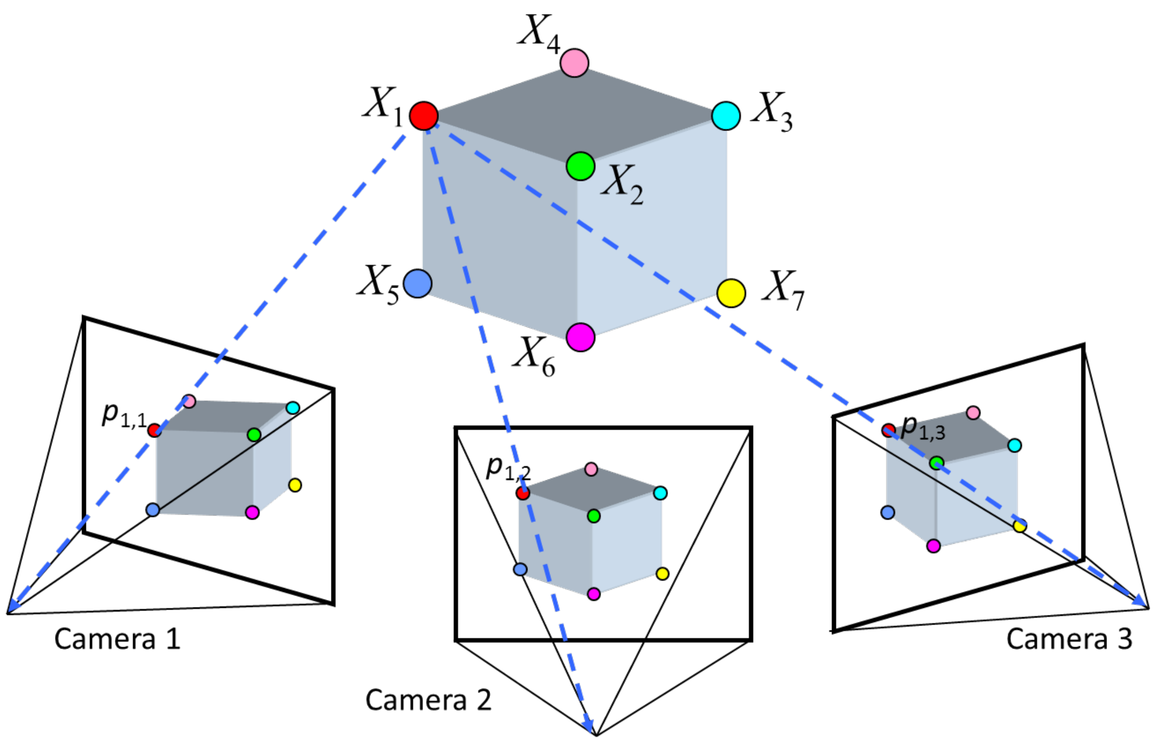
- If we know camera parameters and correspondences between points in different images…
  - How do we figure out 3D point positions?



[Snavely]

Camera Calibration

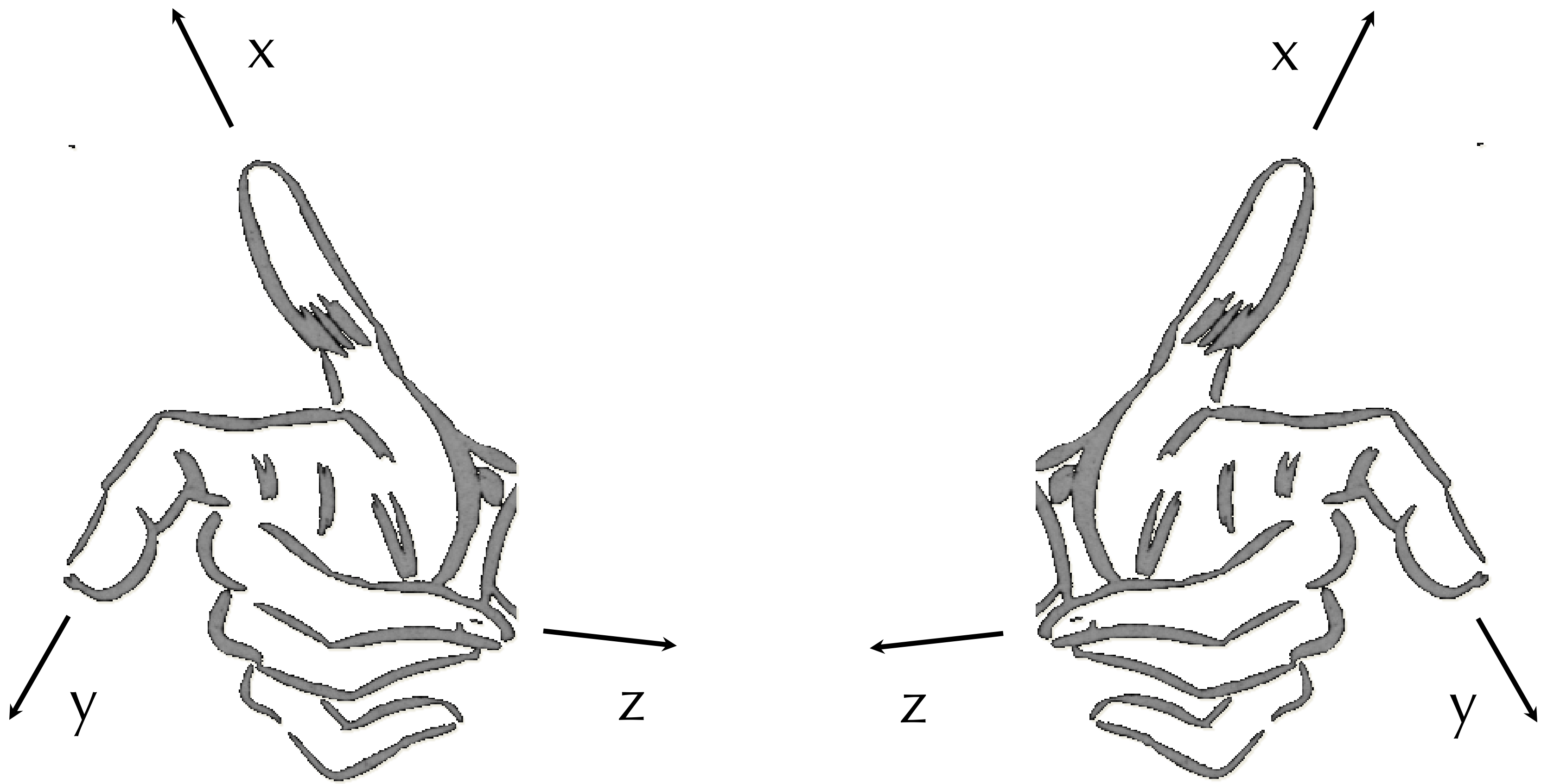
- If we know 3D point positions and correspondences between points and pixels…
  - How do we compute the camera parameters?



[Snavely]

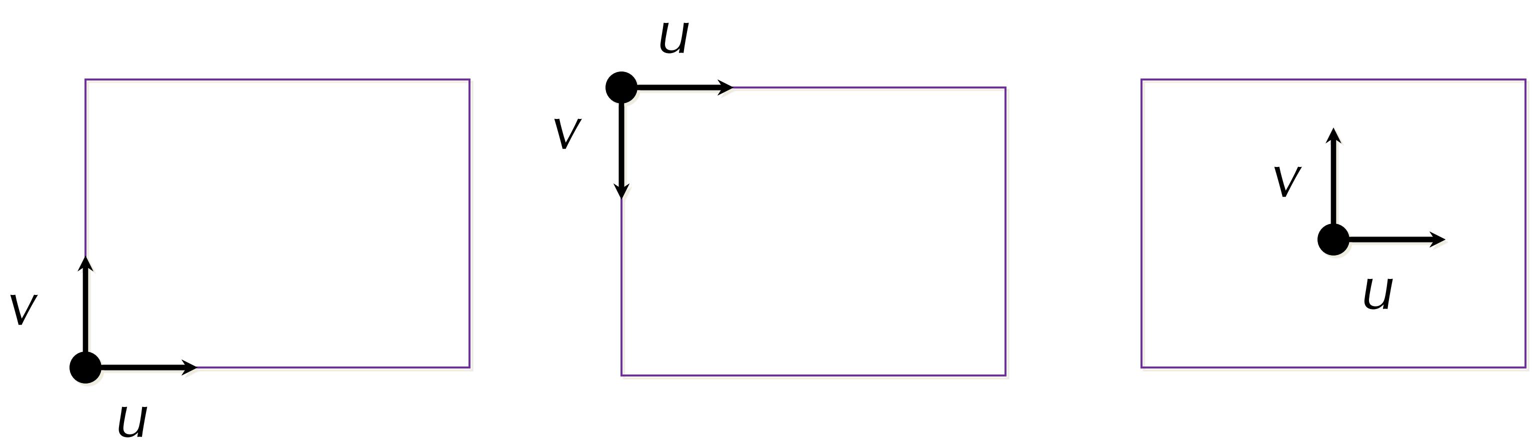
3D Coordinate Systems

- Right-handed vs. left-handed



2D Coordinate Systems

- y axis up vs. y axis down
- Origin at center vs. corner
- Will often write (u, v) for image coordinates



3D Geometry Basics

- 3D points = column vectors

$$\vec{p} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

- Transformations = pre-multiplied matrices

$$\mathbf{T}\vec{p} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Rotation

- Rotation about the *z* axis

$$\mathbf{R}_z = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- Rotation about *x*, *y* axes similar (cyclically permute *x*, *y*, *z*)

Arbitrary Rotation

- Any rotation is a composition of rotations about x, y, and z
- Composition of transformations = matrix multiplication (watch the order!)
- Result: orthonormal matrix
  - Each row, column has unit length
  - Dot product of rows or columns = 0
  - Inverse of matrix = transpose

Arbitrary Rotation

- Rotate around *x*, *y*, then *z*:

$$\mathbf{R} = \begin{pmatrix} \cos\theta_y \cos\theta_z & -\cos\theta_x \sin\theta_z + \sin\theta_x \sin\theta_y \cos\theta_z & \sin\theta_x \sin\theta_z + \cos\theta_x \sin\theta_y \cos\theta_z \\ \cos\theta_y \sin\theta_z & \cos\theta_x \cos\theta_z + \sin\theta_x \cos\theta_y \sin\theta_z & -\sin\theta_x \cos\theta_z + \cos\theta_x \sin\theta_y \sin\theta_z \\ -\sin\theta_y & \sin\theta_x \cos\theta_y & \cos\theta_x \cos\theta_y \end{pmatrix}$$

- Don't do this! It's probably buggy!
  Compute simple matrices and multiply them…

Scale

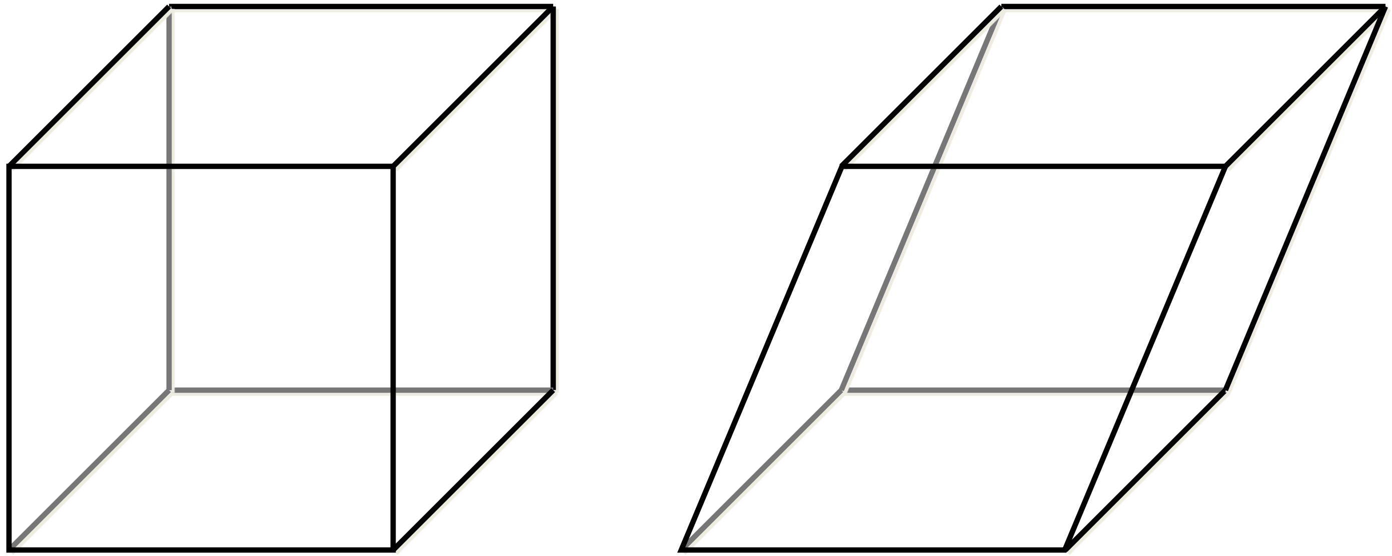
- Scale in x, y, z:

$$\mathbf{S} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{pmatrix}$$

Shear

- Shear parallel to xy plane:

$$\boldsymbol{\sigma}_{xy} = \begin{pmatrix} 1 & 0 & \sigma_x \\ 0 & 1 & \sigma_y \\ 0 & 0 & 1 \end{pmatrix}$$



Translation

- Can translation be represented by multiplying by a 3×3 matrix?
- No.
- Proof:

$$\forall \mathbf{A} : \quad \mathbf{A}\vec{0} = \vec{0}$$

Homogeneous Coordinates

- Add a fourth dimension to each point:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \to \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

- To get "real" (3D) coordinates, divide by *w*:

$$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \to \begin{pmatrix} x/w \\ y/w \\ z/w \end{pmatrix}$$

Translation in Homogeneous Coordinates

$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} x + t_x w \\ y + t_y w \\ z + t_z w \\ w \end{pmatrix}$$

- After divide by w, this is just a translation by (t_x, t_y, t_z)

Perspective Projection

- What does 4th row of matrix do?

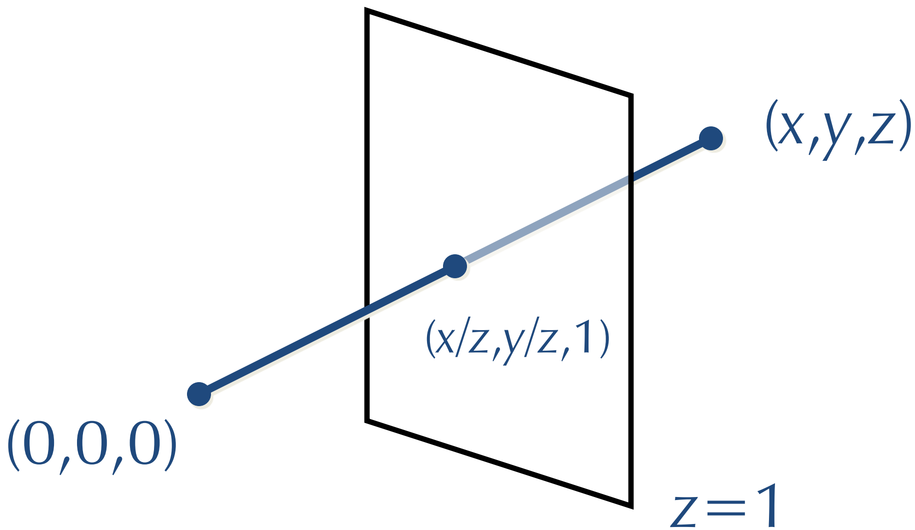
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ z \end{pmatrix}$$

- After divide,

$$\begin{pmatrix} x \\ y \\ z \\ z \end{pmatrix} \rightarrow \begin{pmatrix} x/z \\ y/z \\ 1 \end{pmatrix}$$

Perspective Projection

- This is projection onto the $z=1$ plane



- Add scaling, flipping, etc. $\Rightarrow$ pinhole camera model

Putting It All Together: A Camera Model

Scale to pixel size, flip y

Camera orientation

Camera location

Translate to image center

Perspective projection

3D point (homogeneous coords)

$$\mathbf{T}_{img}\mathbf{S}_{pix}\mathbf{P}_{cam}\mathbf{R}_{cam}\mathbf{T}_{cam}\vec{x}$$

Then perform homogeneous divide, and get (u,v) coords

Putting It All Together: A Camera Model

Intrinsics

Extrinsics

$$\mathbf{T}_{img}\mathbf{S}_{pix}\mathbf{P}_{cam}\mathbf{R}_{cam}\mathbf{T}_{cam}\vec{x}$$

Putting It All Together: A Camera Model

Eye coordinates

Camera coordinates
Normalized device coordinates

Image coordinates
Pixel coordinates

World coordinates

$$\mathbf{T}_{img}\mathbf{S}_{pix}\mathbf{P}_{cam}\mathbf{R}_{cam}\mathbf{T}_{cam}\vec{x}$$

More General Camera Model

- Multiply all these matrices together
- Don't care about "z" after transformation

$$\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ \bullet & \bullet & \bullet & \bullet \\ i & j & k & l \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \xrightarrow[\text{divide}]{\text{homogeneous}} \begin{pmatrix} \dfrac{ax+by+cz+d}{ix+jy+kz+l} \\ \dfrac{ex+fy+gz+h}{ix+jy+kz+l} \\ \bullet \end{pmatrix}$$

- Scale ambiguity → 11 free parameters
  - 6 extrinsic, 5 intrinsic

Radial Distortion

- Radial distortion is nonlinear: cannot be represented by matrix

$$u_{img} \rightarrow c_u + u^*_{img}\left(1+\kappa({u^*_{img}}^2 + {v^*_{img}}^2)\right)$$
$$v_{img} \rightarrow c_v + v^*_{img}\left(1+\kappa({u^*_{img}}^2 + {v^*_{img}}^2)\right)$$

- (c_u, c_v) is image center,

  $u^*_{img} = u_{img} - c_u$, $v^*_{img} = v_{img} - c_v$,

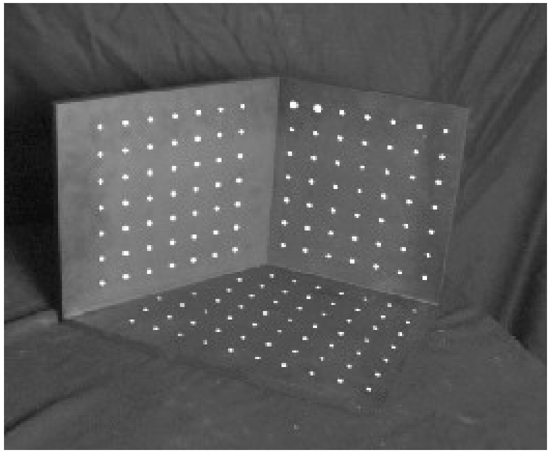
  κ is first-order radial distortion coefficient

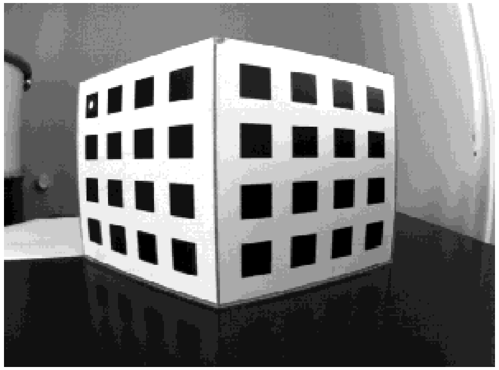
Camera Calibration

- Determining values for camera parameters
- Necessary for any algorithm that requires 3D $\leftrightarrow$ 2D mapping
- Method used depends on:
  - What data is available
  - Intrinsics only vs. extrinsics only vs. both
  - Form of camera model

Camera Calibration

- General idea: place "calibration object" with known geometry in the scene
- Get correspondences
- Solve for mapping from scene to image





The Opti-CAL Calibration Target Image

Camera Calibration

Chromaglyphs
Courtesy of Bruce Culbertson, HP Labs
http://www.hpl.hp.com/personal/Bruce_Culbertson/ibr98/chromagl.htm

Camera Calibration – Example 1

- Given:
  - 3D ↔ 2D correspondences
  - General perspective camera model (11-parameter, no radial distortion)
- Write equations:

$$\frac{ax_1 + by_1 + cz_1 + d}{ix_1 + jy_1 + kz_1 + l} = u_1$$

$$\frac{ex_1 + fy_1 + gz_1 + h}{ix_1 + jy_1 + kz_1 + l} = v_1$$

$$\vdots$$

Camera Calibration – Example 1

$$\begin{pmatrix} x_1 & y_1 & z_1 & 1 & 0 & 0 & 0 & 0 & -u_1x_1 & -u_1y_1 & -u_1z_1 & -u_1 \\ 0 & 0 & 0 & 0 & x_1 & y_1 & z_1 & 1 & -v_1x_1 & -v_1y_1 & -v_1z_1 & -v_1 \\ x_2 & y_2 & z_2 & 1 & 0 & 0 & 0 & 0 & -u_2x_2 & -u_2y_2 & -u_2z_2 & -u_2 \\ 0 & 0 & 0 & 0 & x_2 & y_2 & z_2 & 1 & -v_2x_2 & -v_2y_2 & -v_2z_2 & -v_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ \vdots \\ l \end{pmatrix} = \vec{\mathbf{0}}$$

- Linear equation
- Overconstrained (more equations than unknowns)
- Underconstrained (rank deficient matrix – any multiple of a solution, including 0, is also a solution)

Camera Calibration – Example 1

- Standard linear least squares methods for $Ax=0$ will give the solution $x=0$
- Instead, look for a solution with $|x| = 1$
- That is, minimize $|Ax|^2$ subject to $|x|^2=1$

Camera Calibration – Example 1

- Minimize $|Ax|^2$ subject to $|x|^2=1$
- $|Ax|^2 = (Ax)^T(Ax) = (x^TA^T)(Ax) = x^T(A^TA)x$
- Expand x in terms of eigenvectors of A^TA:

$$x = \mu_1 e_1 + \mu_2 e_2 + \ldots$$
$$x^T(A^TA)x = \lambda_1 \mu_1^2 + \lambda_2 \mu_2^2 + \ldots$$
$$|x|^2 = \mu_1^2 + \mu_2^2 + \ldots$$

Camera Calibration – Example 1

- To minimize

$$\lambda_1\mu_1{}^2+\lambda_2\mu_2{}^2+\ldots$$

  subject to

$$\mu_1{}^2+\mu_2{}^2+\ldots = 1$$

  set $\mu_{min} = 1$ and all other $\mu_i = 0$

- Thus, least squares solution is eigenvector of A^TA corresponding to minimum (nonzero) eigenvalue

Camera Calibration – Example 2

- Incorporating radial distortion
- Option 1:
  - Find distortion first (straight lines in calibration target)
  - Warp image to eliminate distortion
  - Run (simpler) perspective calibration
- Option 2: nonlinear least squares
  - Usually gradient descent or Levenberg-Marquardt
  - Common implementations available (e.g. Matlab optimization toolbox)

Camera Calibration – Example 3

- Incorporating additional constraints into camera model
  - No shear
  - Square pixels
  - Camera projection center = image center
  - etc.
- These impose *nonlinear* constraints on camera parameters

Camera Calibration – Example 3

- Option 1: solve for general perspective model, then find closest solution that satisfies constraints

- Option 2: constrained nonlinear least squares

Camera Calibration – Example 4

- What if 3D points are not known?
- Structure from motion problem!
- As we saw, can often be solved since
  # of knowns > # of unknowns

Structure from Motion (SfM)

- Minimize sum of squared reprojection errors:

$$g(\mathbf{X}, \mathbf{R}, \mathbf{T}) = \sum_{i=1}^{m} \sum_{j=1}^{n} \underbrace{w_{ij}}_{\substack{\textit{indicator variable}: \\ \text{is point } i \text{ visible in image } j\text{ ?}}} \cdot \left\| \underbrace{\mathbf{P}(\mathbf{x}_i, \mathbf{R}_j, \mathbf{t}_j)}_{\substack{\textit{predicted} \\ \text{image location}}} - \underbrace{\begin{bmatrix} u_{i,j} \\ v_{i,j} \end{bmatrix}}_{\substack{\textit{observed} \\ \text{image location}}} \right\|^2$$

- Minimizing this function is called bundle adjustment
  - Optimized using non-linear least squares

[Snavely]

Structure from Motion

[Snavely]

Problem Size

- What are the variables?
- How many variables per camera?
- How many variables per point?

- Trevi Fountain collection

466 input photos

+ > 100,000 3D points

= very large optimization problem

[Snavely]

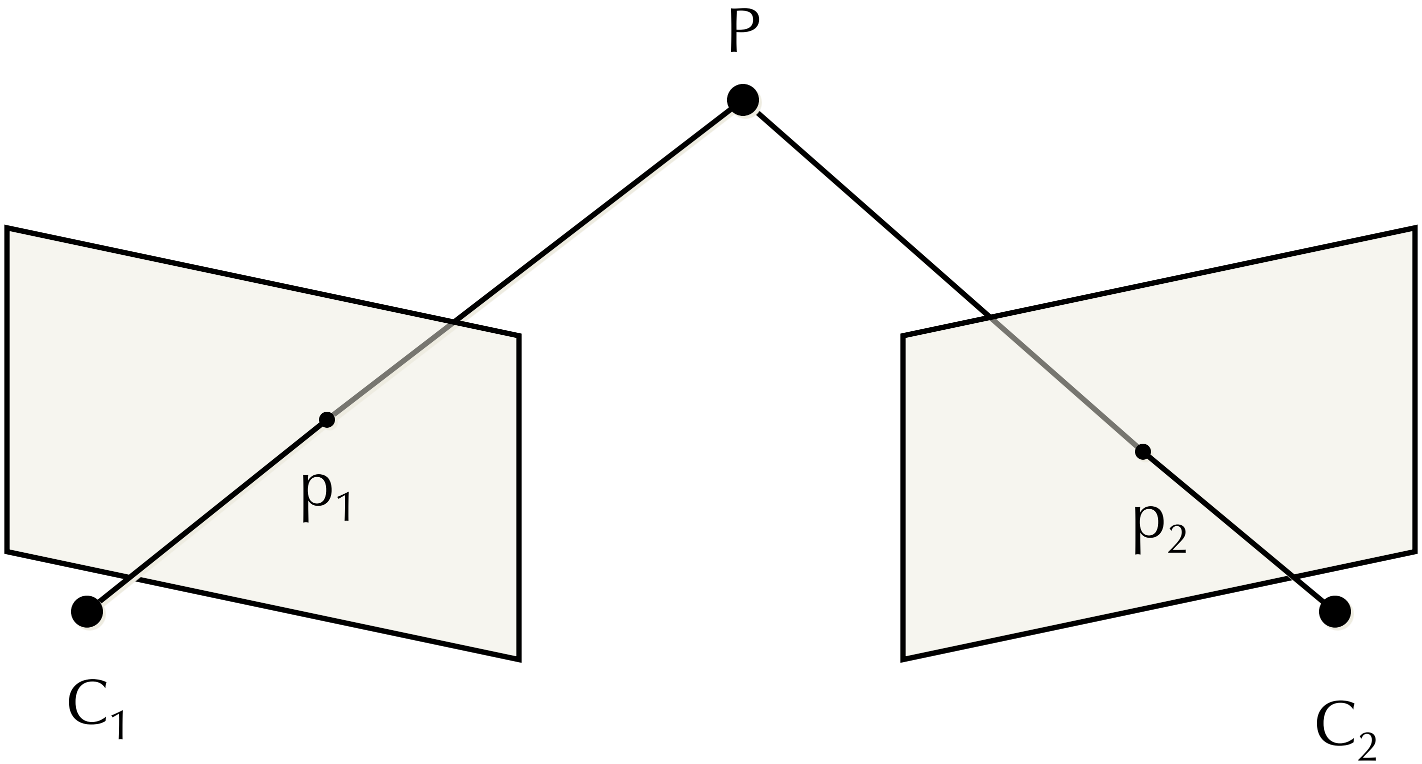
Structure from Motion

[Snavely]

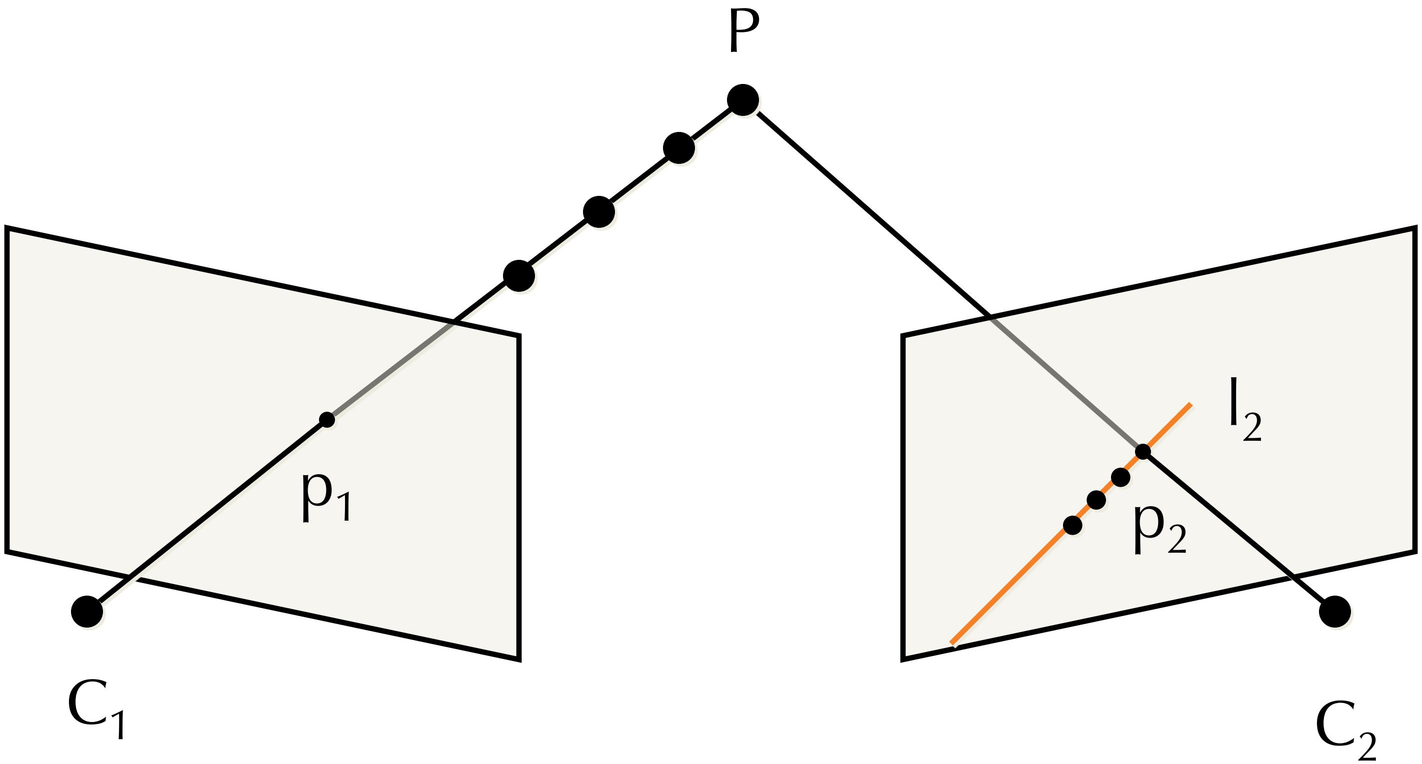
Multi-Camera Geometry

- Epipolar geometry – relationship between observed positions of points in multiple cameras
- Assume:
  - 2 cameras
  - Known intrinsics and extrinsics

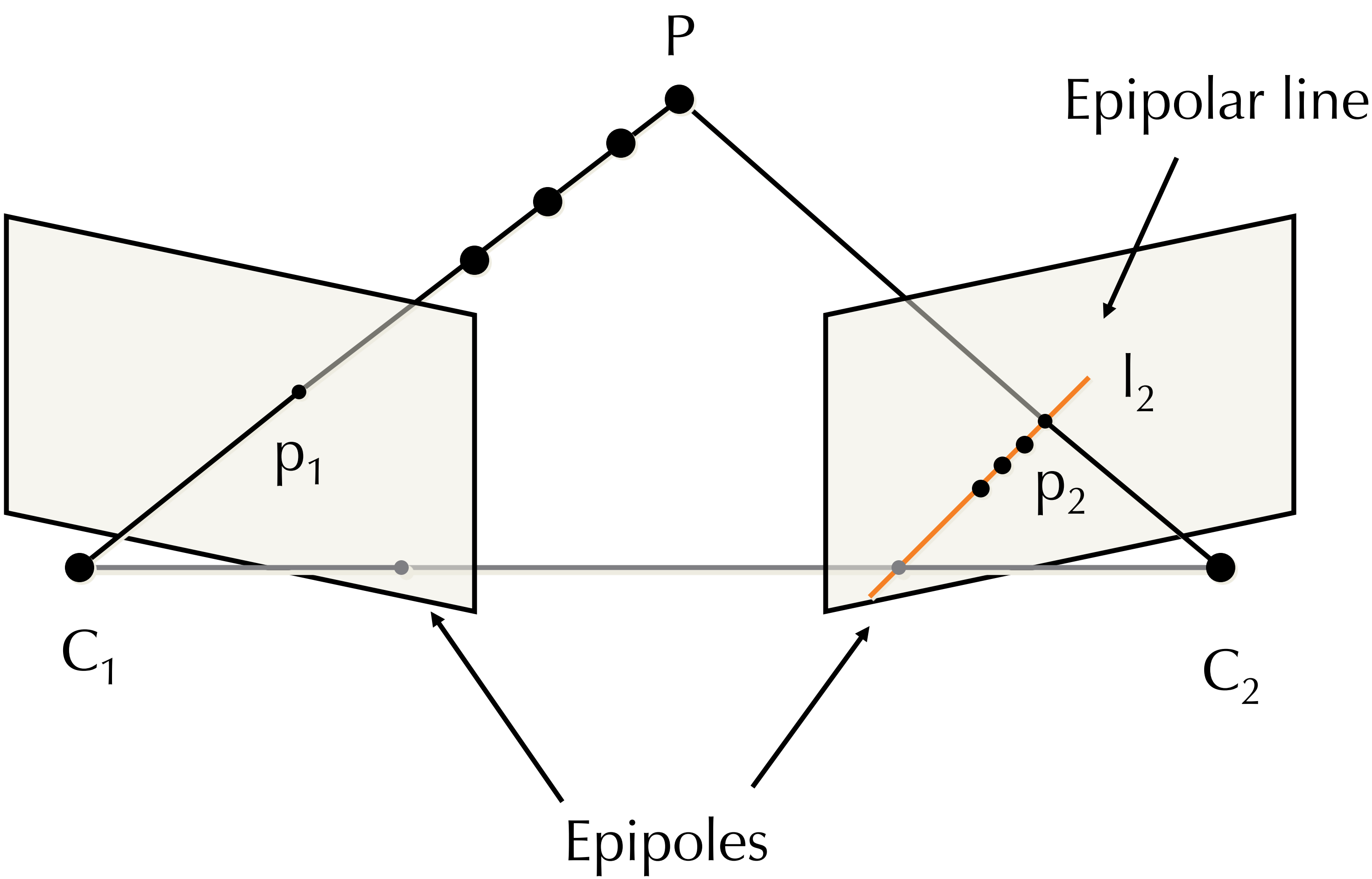
Epipolar Geometry



Epipolar Geometry

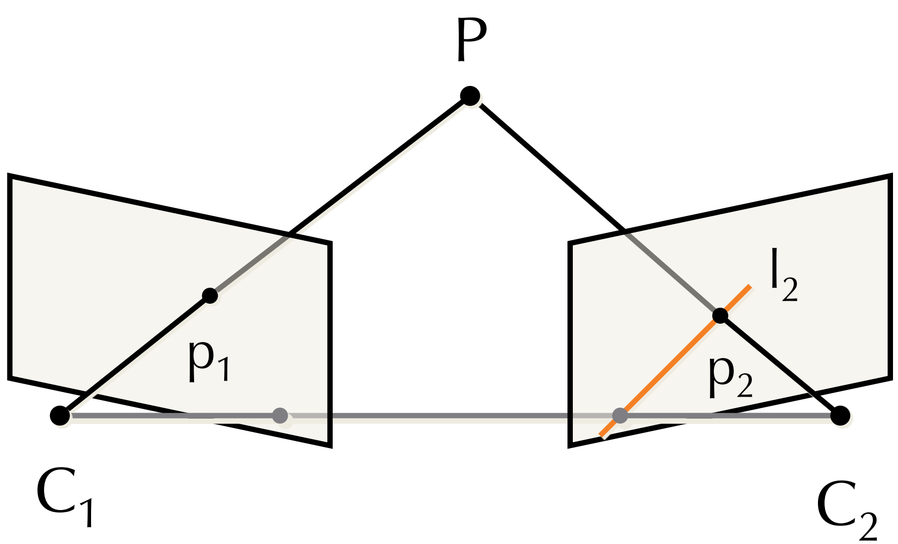


Epipolar Geometry



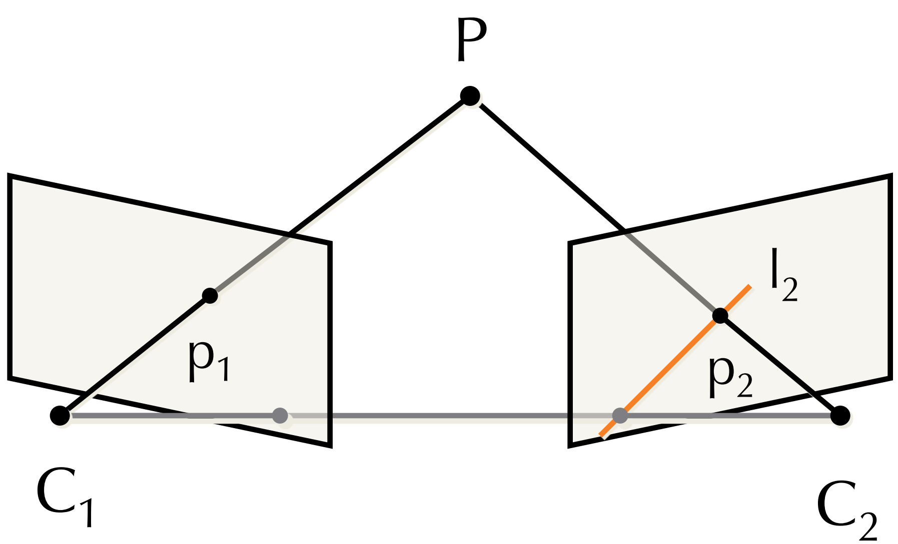
Epipolar Geometry

- Goal: derive equation for l_2
- Observation: P, C_1, C_2 determine a plane



Epipolar Geometry

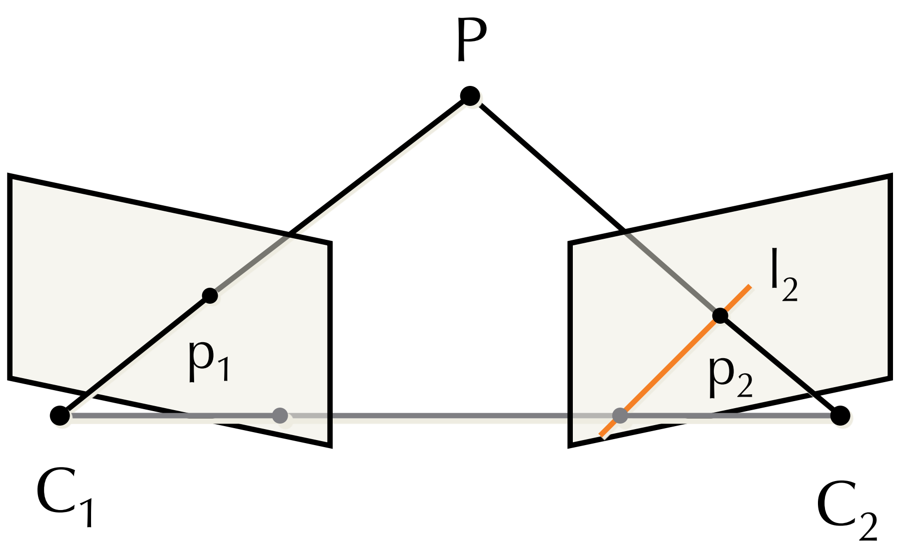
- Work in coordinate frame of C_1
- Normal of plane is $T \times Rp_2$, where T is relative translation, R is relative rotation



Epipolar Geometry

- p_1 is perpendicular to this normal:

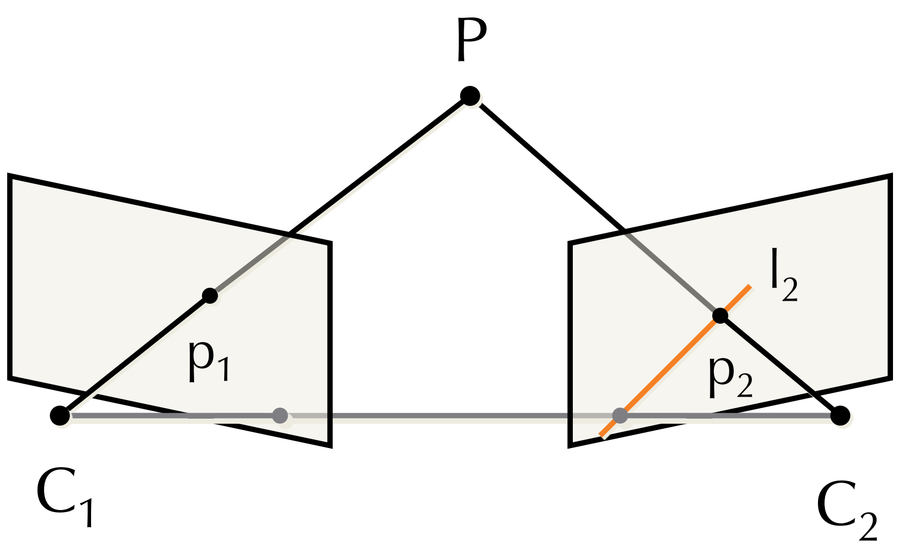
$$p_1 \bullet (T \times Rp_2) = 0$$



Epipolar Geometry

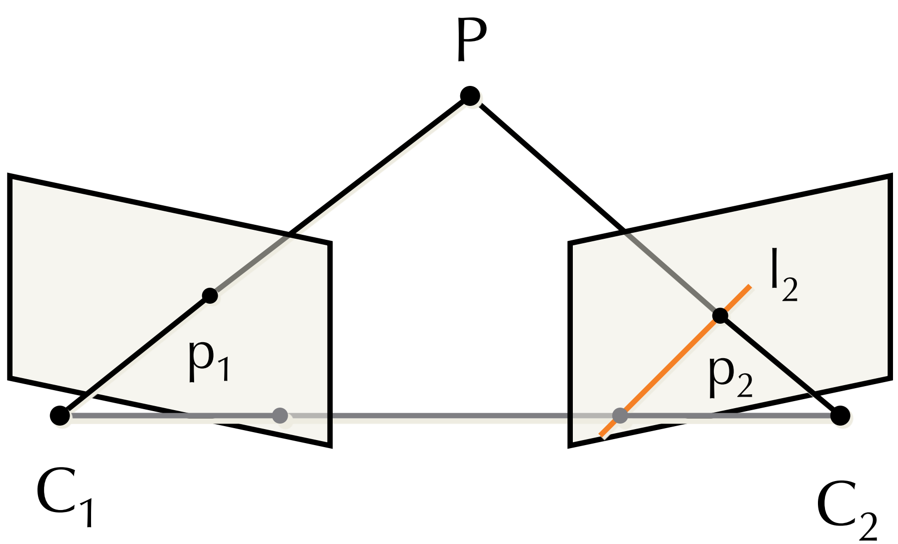
- Write cross product as matrix multiplication

$$\vec{T} \times x = \mathbf{T}^{\times} x, \qquad \mathbf{T}^{\times} = \begin{pmatrix} 0 & -T_z & T_y \\ T_z & 0 & -T_x \\ -T_y & T_x & 0 \end{pmatrix}$$



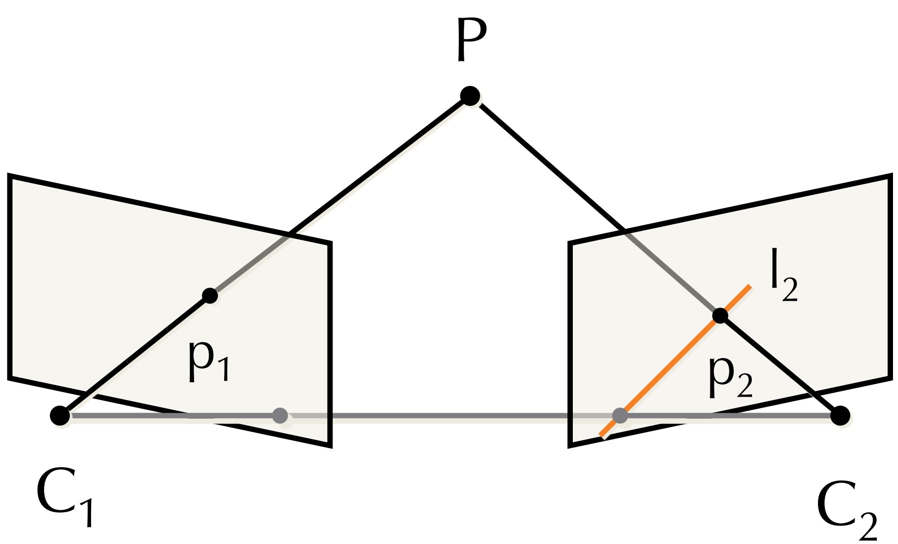
Epipolar Geometry

- $p_1 \bullet T^{\times} R\, p_2 = 0 \quad \Rightarrow \quad p_1{}^{T} E\, p_2 = 0$
- E is the essential matrix



Essential Matrix

- E depends only on camera geometry
- Given E, can derive equation for line l_2



Fundamental Matrix

- Can define fundamental matrix F analogously, operating on pixel coordinates instead of camera coordinates

$$u_1^T F u_2 = 0$$

- Advantage: can sometimes estimate F without knowing camera calibration
  - Given a few good correspondences, can get epipolar lines and estimate more correspondences, all without calibrating cameras