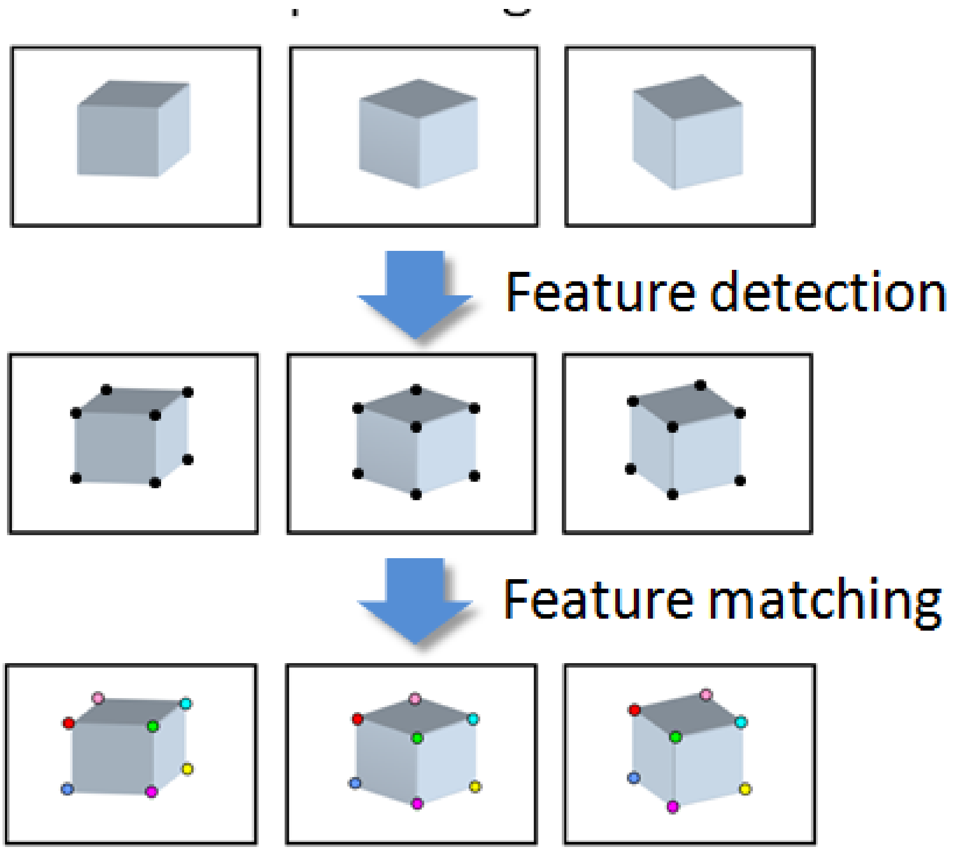
Camera Calibration

COS 429
Princeton University

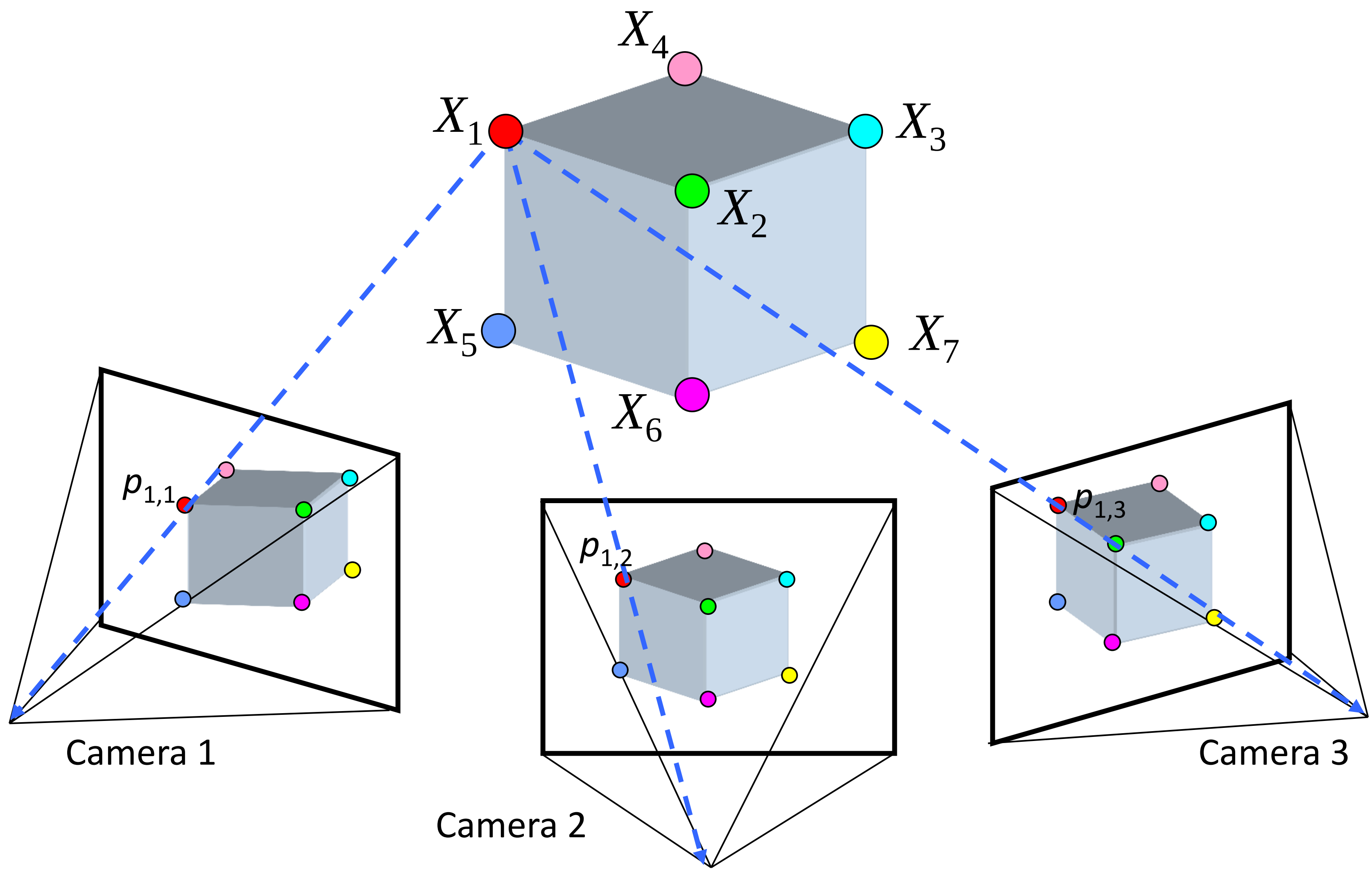
Point Correspondences

What can you figure out from point correspondences?



Noah Snavely

Point Correspondences

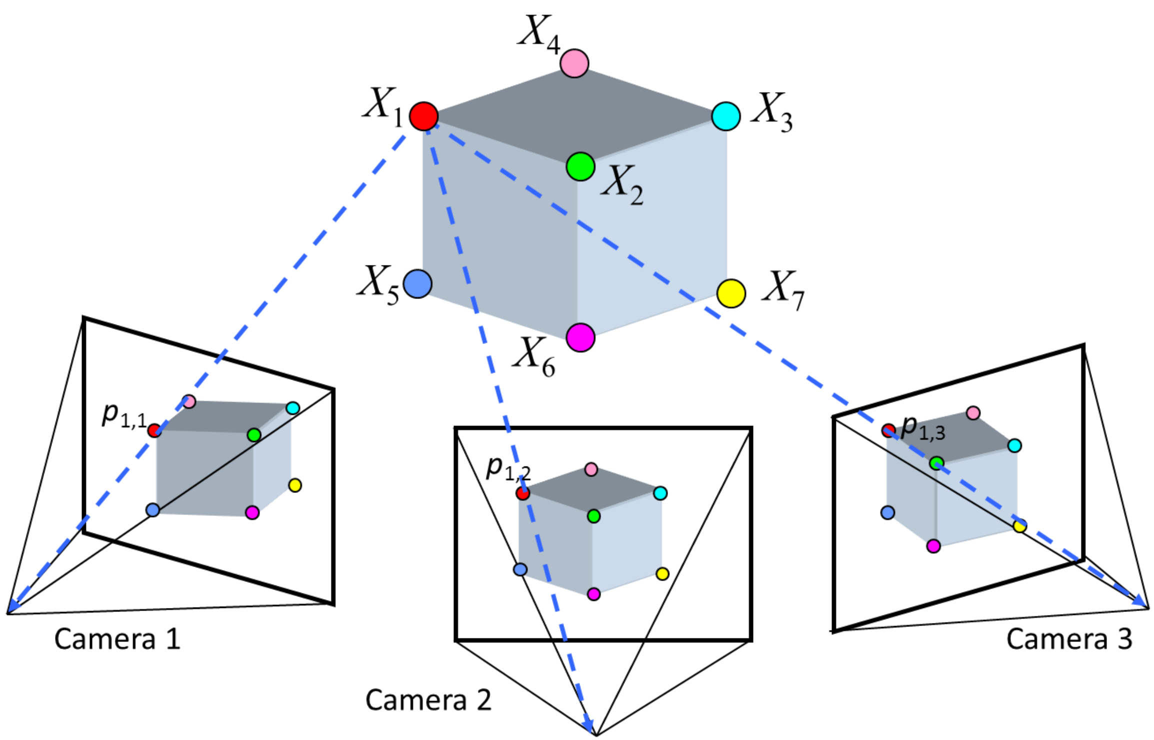


Noah Snavely

Camera calibration

Suppose we know 3D point positions and correspondences to pixels in every image

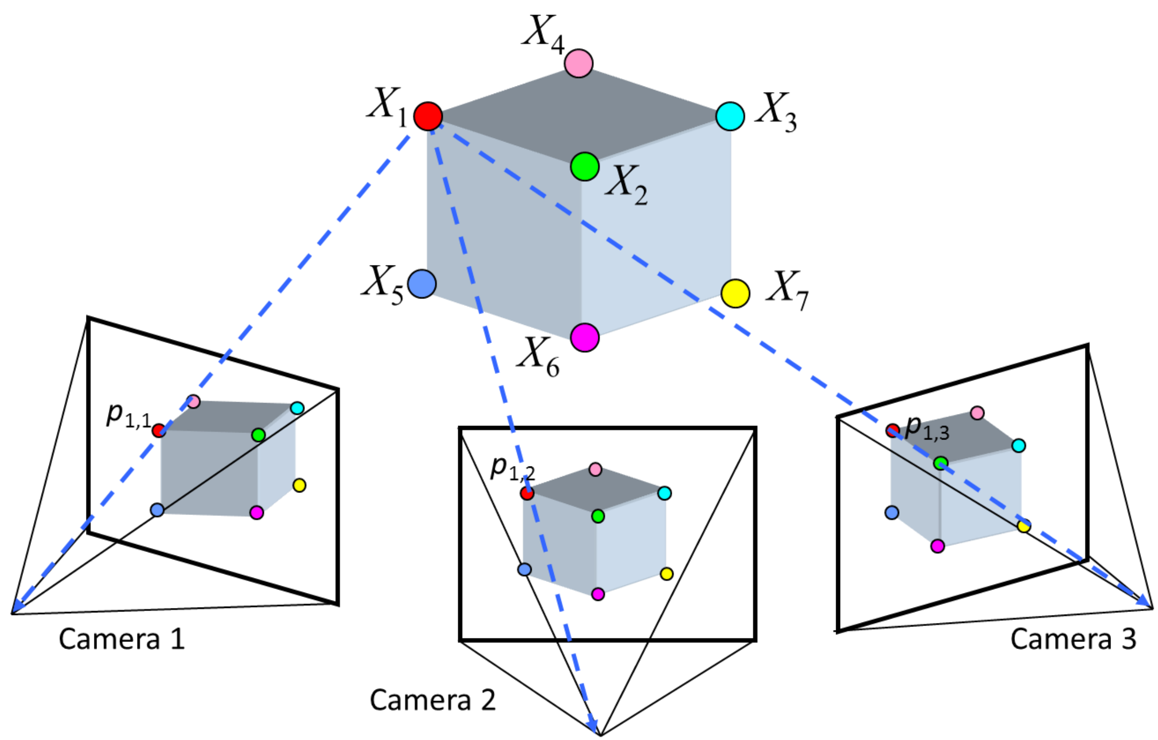
- Can we compute the camera parameters?



Noah Snavely

Camera triangulation

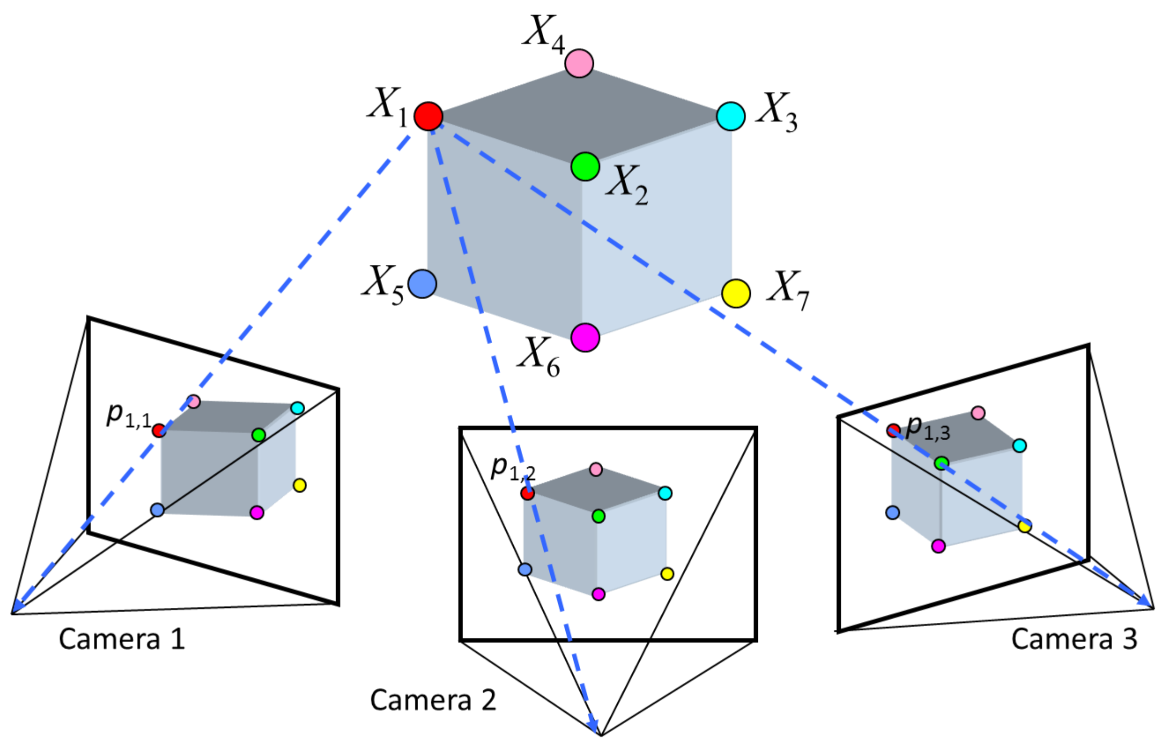
Suppose we know the camera parameters and correspondences between points and pixels in every image?



Noah Snavely

Camera calibration & triangulation

Need to understand how 3D points project into images



Noah Snavely

Outline

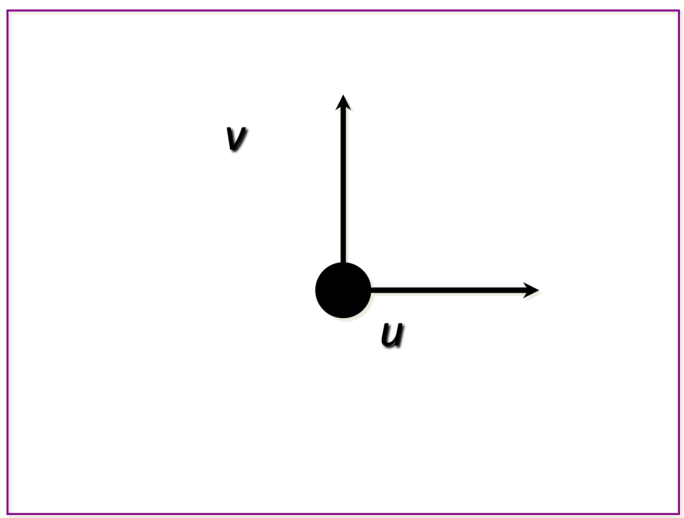
- Camera model
- Camera calibration
- Camera triangulation
- Structure from motion

Outline

- Camera model
- Camera calibration
- Camera triangulation
- Structure from motion

2D Image Basics

- Origin at center, for now
- Y axis is up
- Will often write (*u*, *v*) for image coordinates



3D Geometry Basics

- 3D points = column vectors

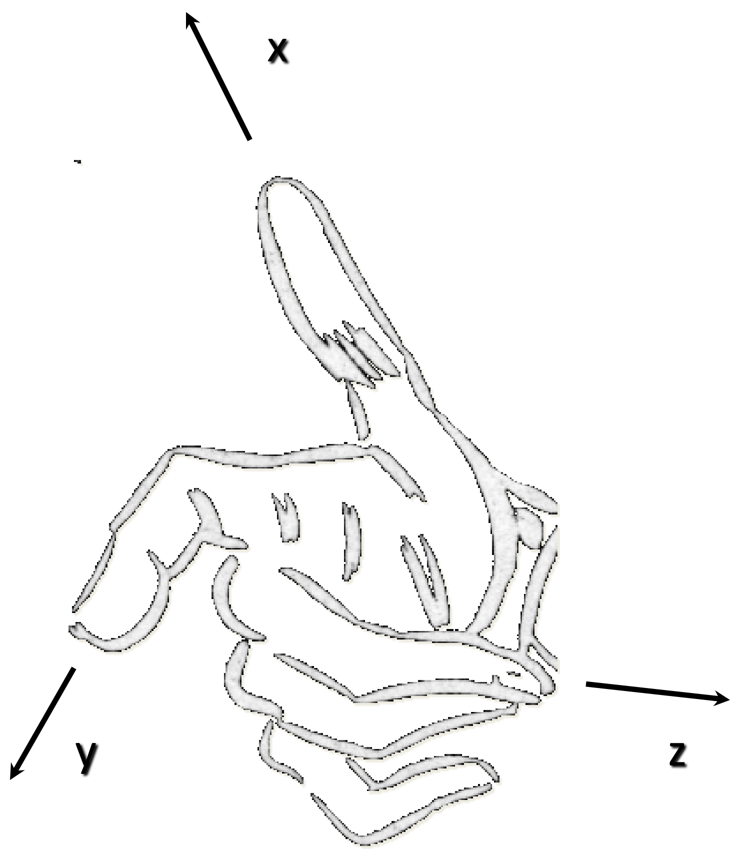
$$\vec{p} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

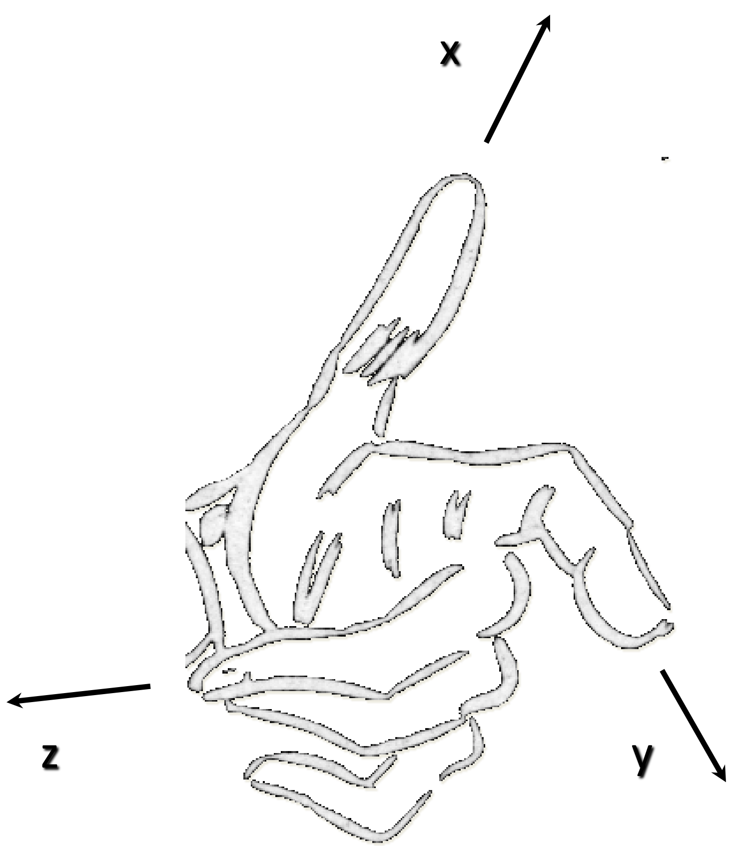
- Transformations = pre-multiplied matrices

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

3D Geometry Basics

- Right-handed vs. left-handed coordinate systems





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*:

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

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

Scale

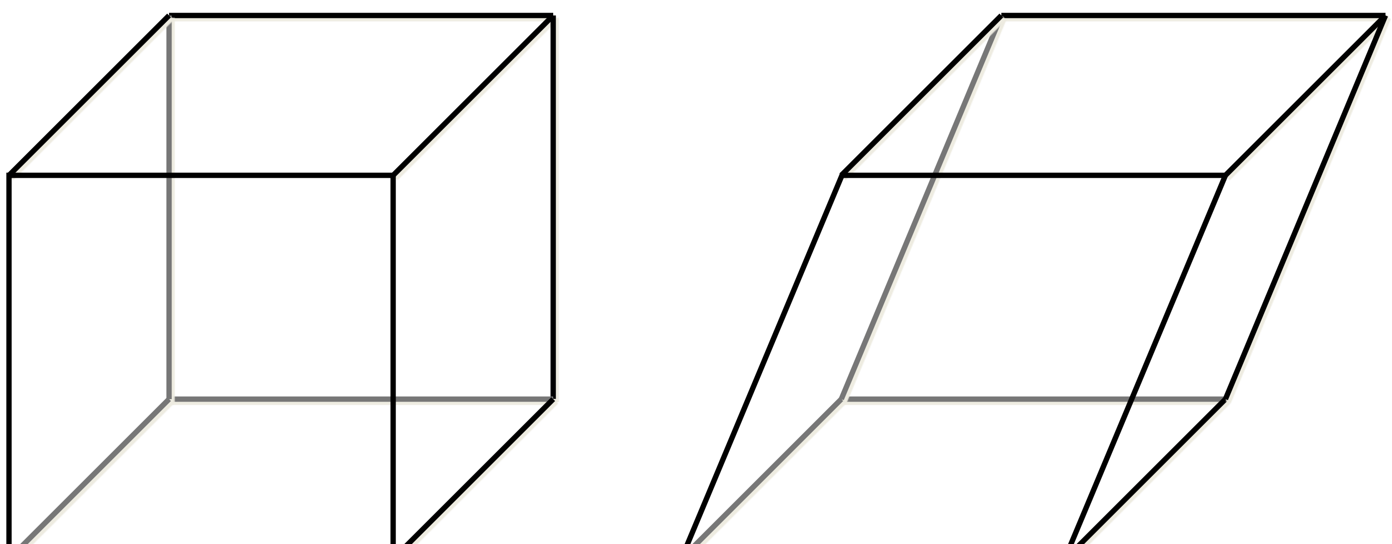
- Scale in *x*, *y*, *z*:

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

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 3D point:

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

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

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

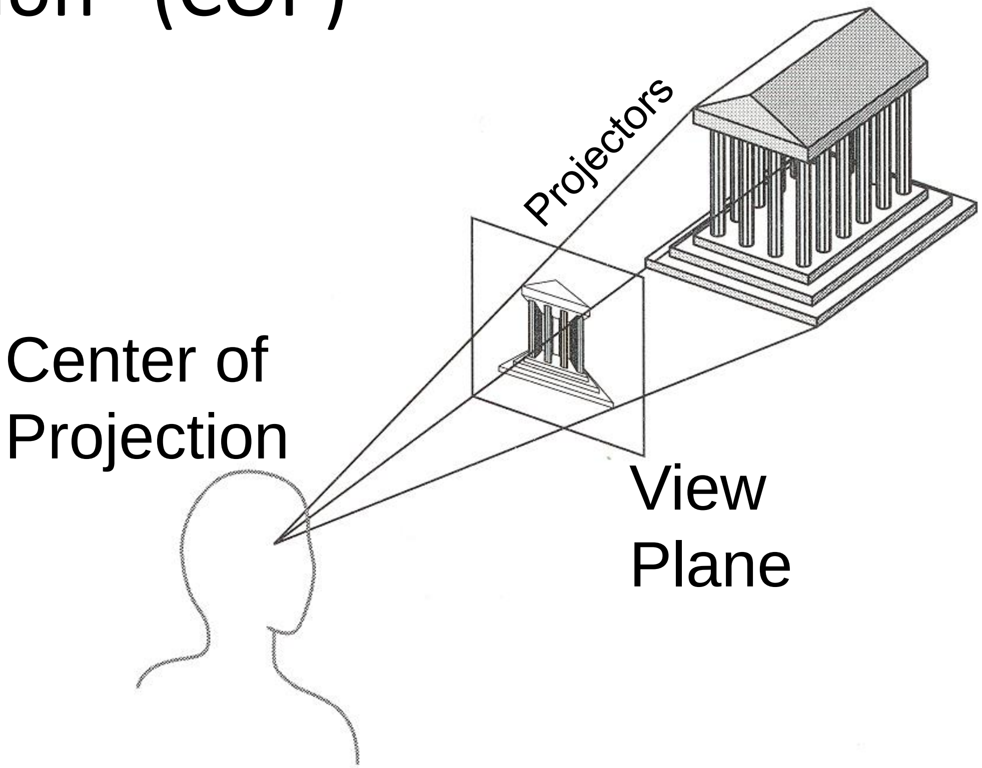
Translation in Homogeneous Coordinates

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

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

Perspective Projection

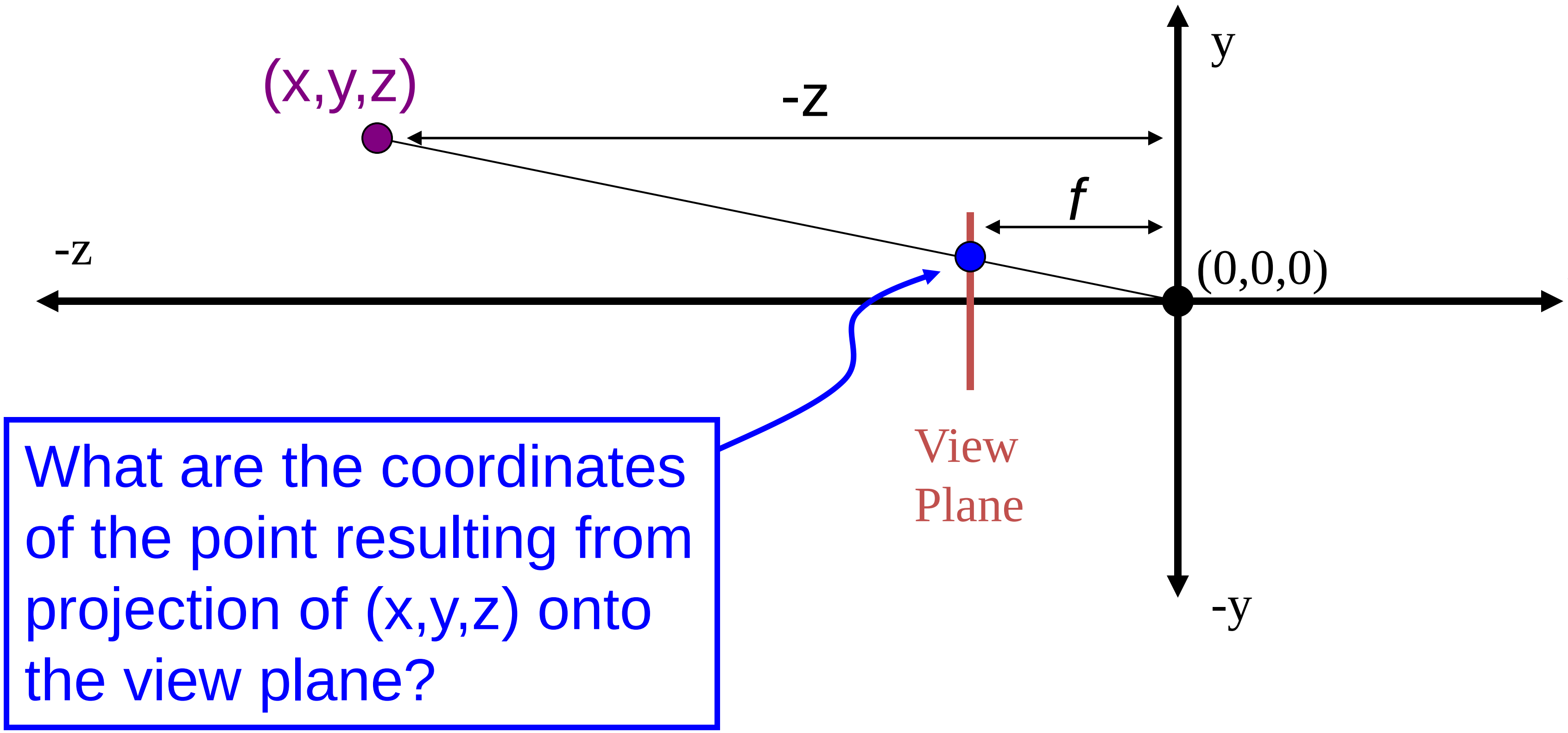
- Map points onto “view plane” along “projectors” emanating from “center of projection” (COP)



Angel Figure 5.9

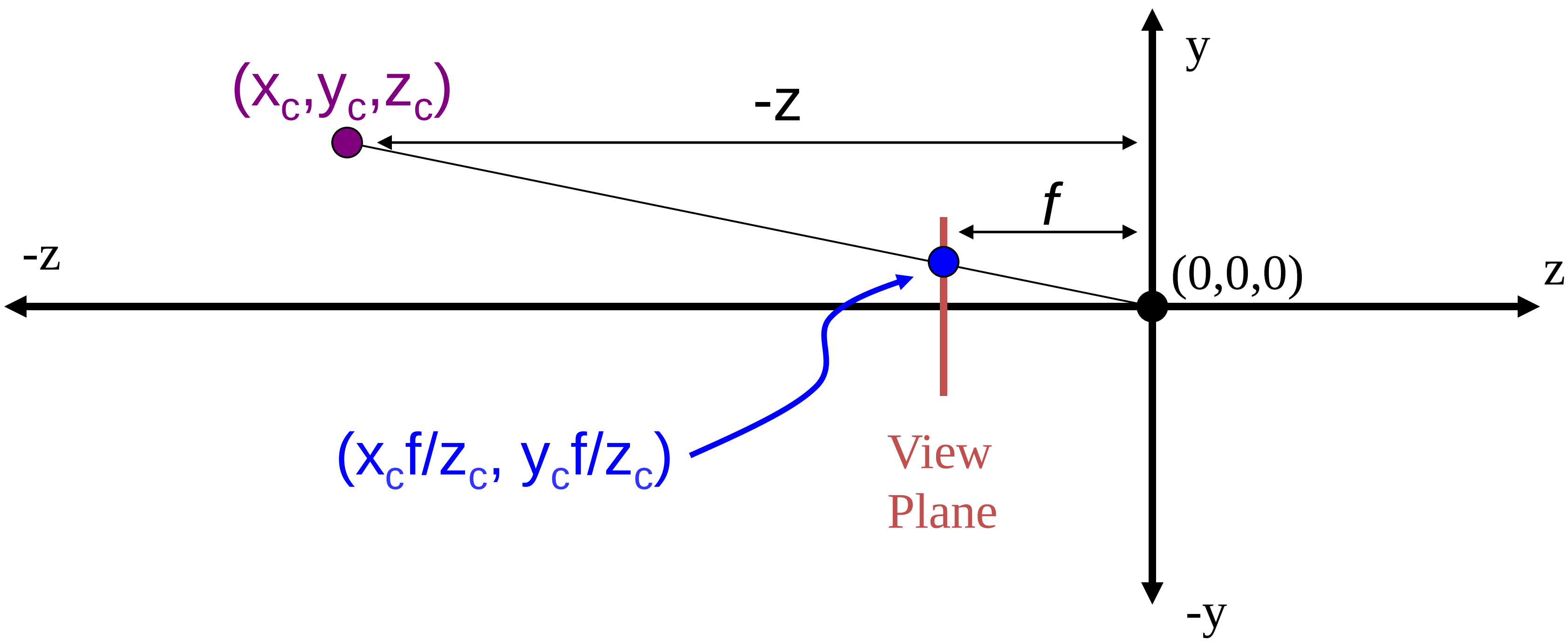
Perspective Projection

- Compute 2D coordinates from 3D coordinates with similar triangles



Perspective Projection

- Compute 2D coordinates from 3D coordinates with similar triangles



Perspective Projection Matrix

- 4x4 matrix representation:

$$x_s = x_c f / z_c \qquad x_s = x'/w' \qquad x' = x_c$$

$$y_s = y_c f / z_c \qquad y_s = y'/w' \qquad y' = y_c$$

$$z_s = f \qquad z_s = z'/w' \qquad z' = z_c$$

$$w_s = 1 \qquad w' = z_c / f$$

$$\begin{bmatrix} x_s \\ y_s \\ z_s \\ w_s \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/f & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

Perspective Projection Properties

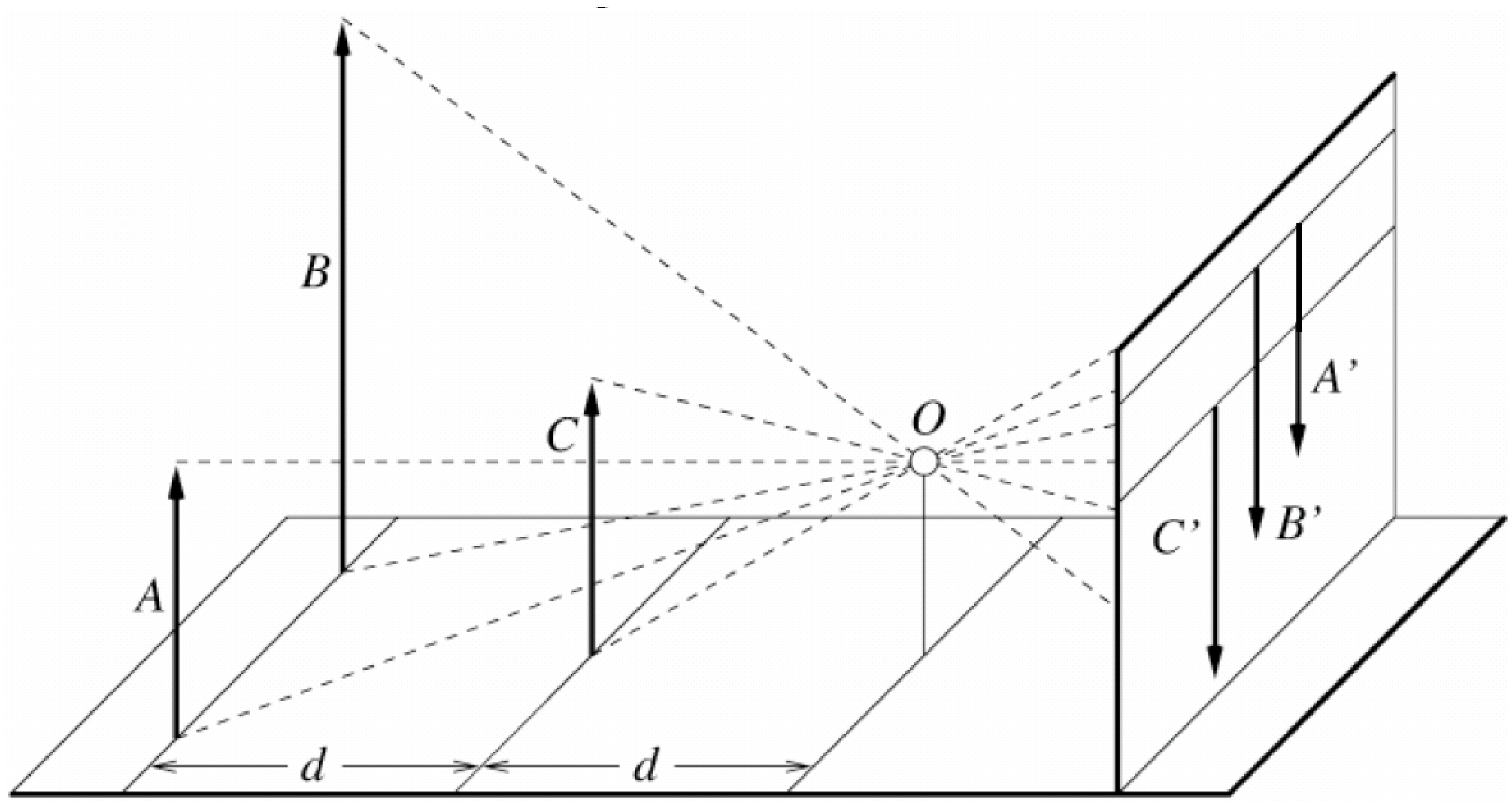
- Points → points
- Lines → lines (collinearity preserved)
- Distances and angles are **not** preserved
- Many points along same ray map to same point in image

- Degenerate cases:
  - Line through focal point projects to a point.
  - Plane through focal point projects to line
  - Plane perpendicular to image plane projects to part of the image.

Slide by Kristen Grauman

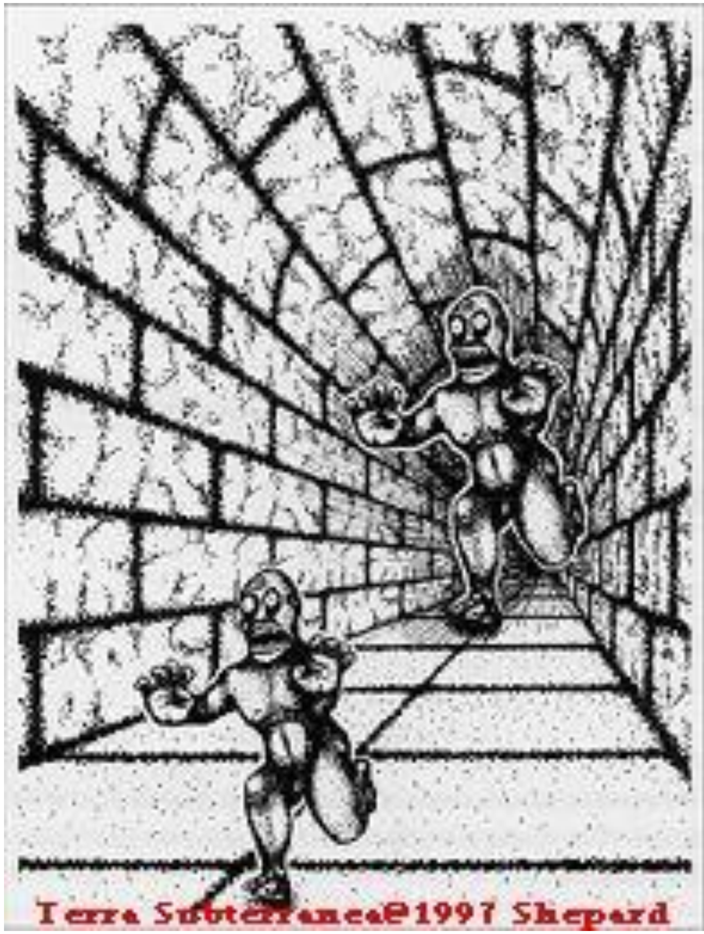
Perspective Effects

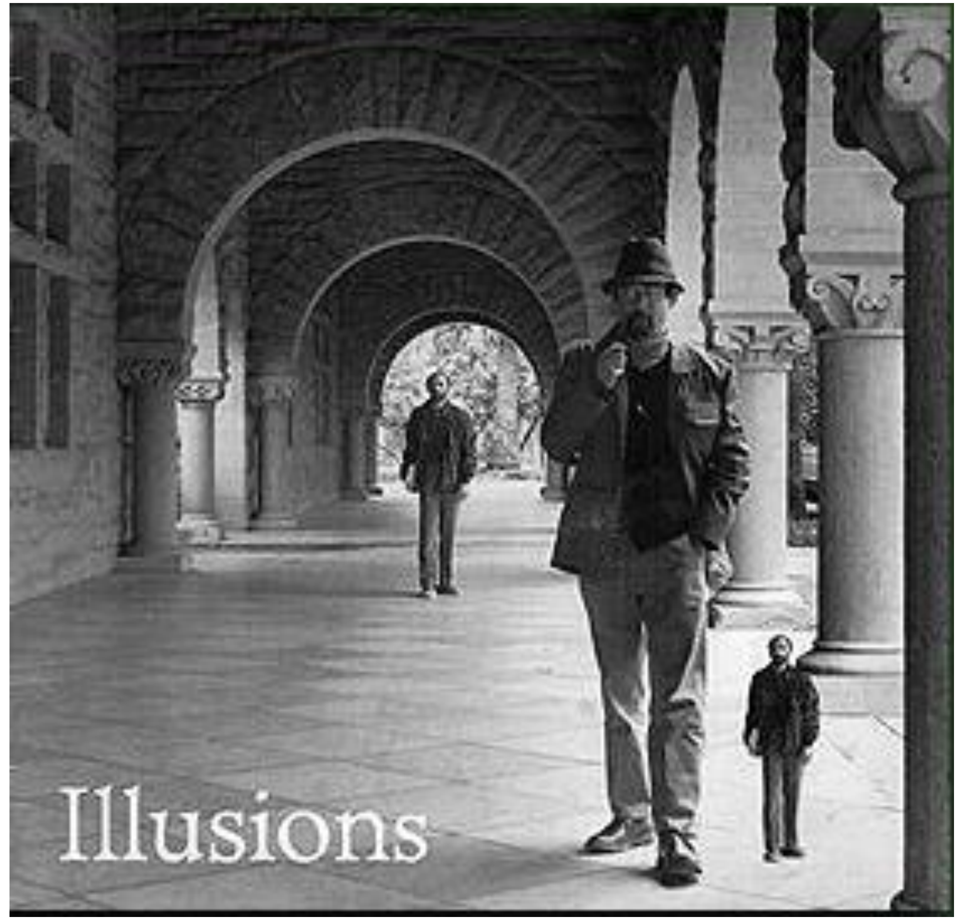
- Far away objects appear smaller



Forsyth and Ponce

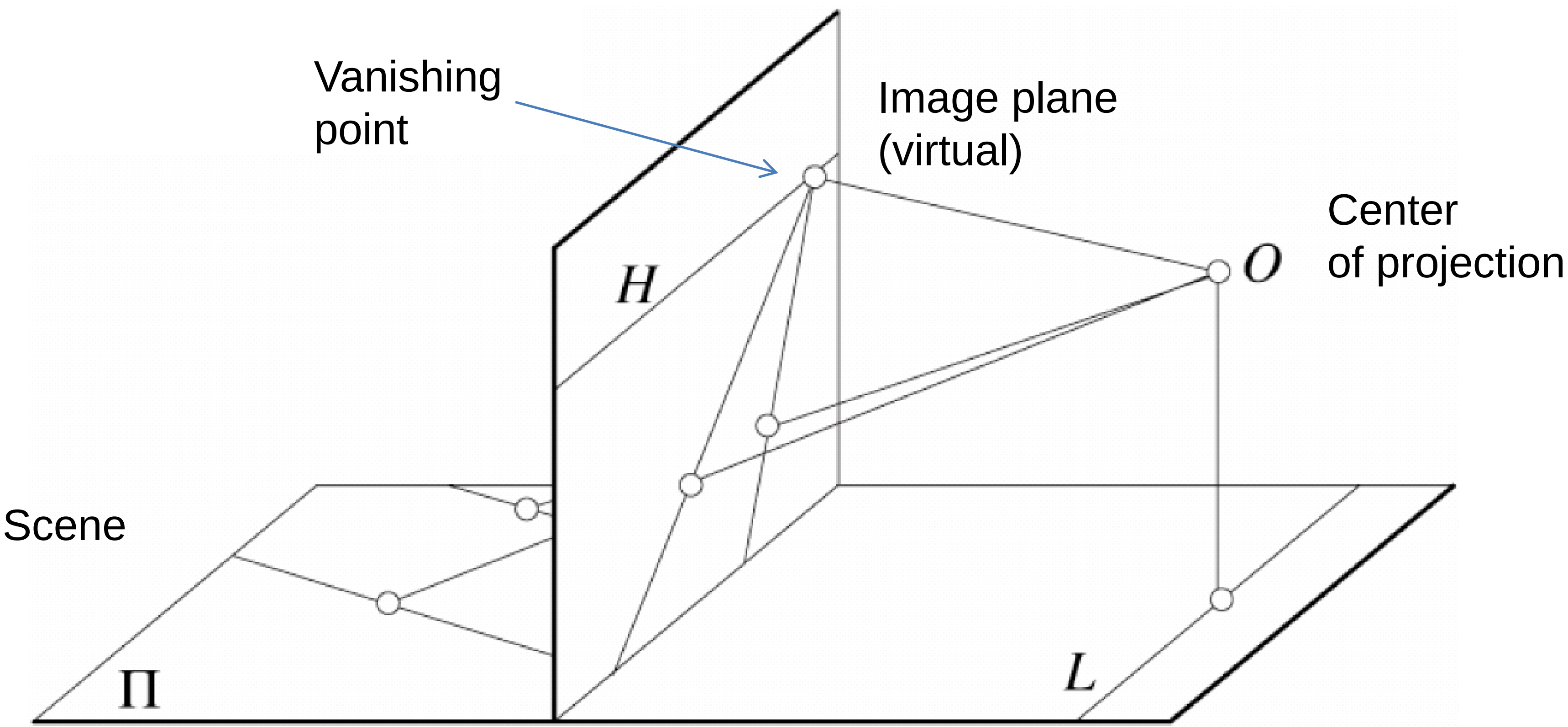
Perspective Effects





Perspective Effects

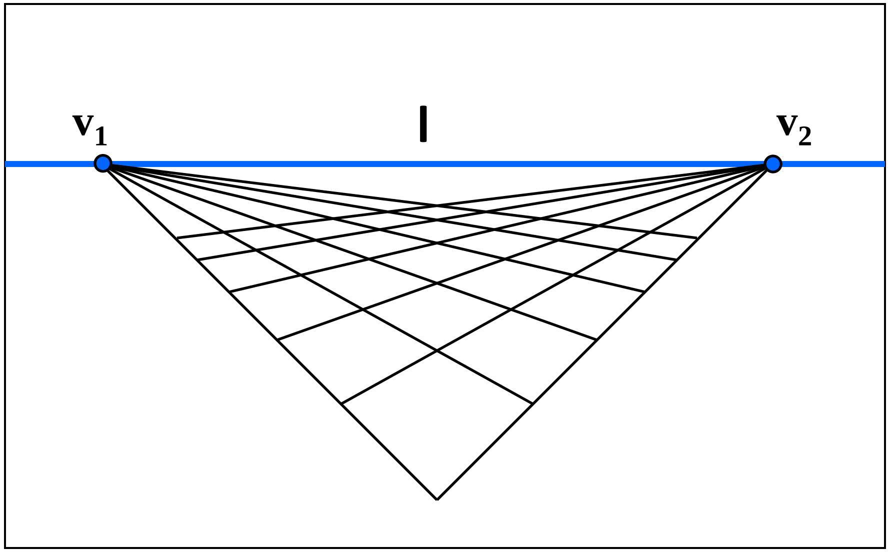
- Parallel lines in the scene intersect at a point after projection onto image plane



Slide by Kristen Grauman

Perspective Effects

- Vanishing Points
  - Any set of parallel lines on a plane define a vanishing point
  - The union of all of these vanishing points is the *horizon line*
    - also called *vanishing line*
  - Different planes (can) define different vanishing lines



A Camera Model

Scale to pixel size

Camera orientation

Camera location

Perspective projection

Translate to image center

3D point (homogeneous coords)

Then unwarp radial distortion

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

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

A Camera Model

Camera coordinates

Normalized device coordinates

Image coordinates
Pixel coordinates

World coordinates

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

A Camera Model

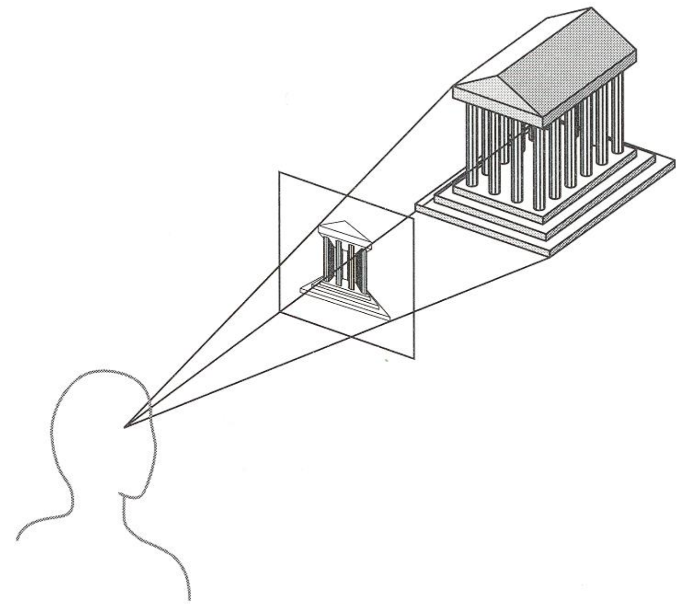
Intrinsics

Extrinsics

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

Camera parameters

- **Extrinsic:** how to map world coordinates to camera coordinates (translation and rotation)
- **Intrinsic:** how to map camera coordinates to image coordinates (projection, translation, scale)



Camera parameters

- **Extrinsic:** how to map world coordinates to camera coordinates (translation and rotation)

$$\mathbf{P}_c = \mathbf{R}(\mathbf{P}_w - \mathbf{T})$$

Camera coordinates

World coordinates

Camera parameters

- **Intrinsic:** how to map camera coordinates to image coordinates (projection, translation, scale)

$$x = -\left(f\frac{X}{Z} - o_x\right)s_x$$

$$y = -\left(f\frac{Y}{Z} - o_y\right)s_y$$

Coordinates of projected point in image coordinates

Coordinates of image point in camera coordinates

Coordinates of image center in pixel units

Effective size of a pixel (mm)

Camera parameters

- Substituting previous eqns describing intrinsic and extrinsic parameters, can relate *pixels coordinates* to *world points:*

$$-(x_{im} - o_x)s_x = f\,\frac{\mathbf{R}_1 \cdot (\mathbf{P}_w - \mathbf{T})}{\mathbf{R}_3 \cdot (\mathbf{P}_w - \mathbf{T})}$$

$$-(y_{im} - o_y)s_y = f\,\frac{\mathbf{R}_2 \cdot (\mathbf{P}_w - \mathbf{T})}{\mathbf{R}_3 \cdot (\mathbf{P}_w - \mathbf{T})}$$

$\mathbf{R}_i$ = Row i of rotation matrix

Projection matrix

- This can be rewritten as a matrix product using homogeneous coordinates:

$$\begin{bmatrix} wx_{im} \\ wy_{im} \\ w \end{bmatrix} = \mathbf{M}_{\mathrm{int}}\mathbf{M}_{ext}\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

where:

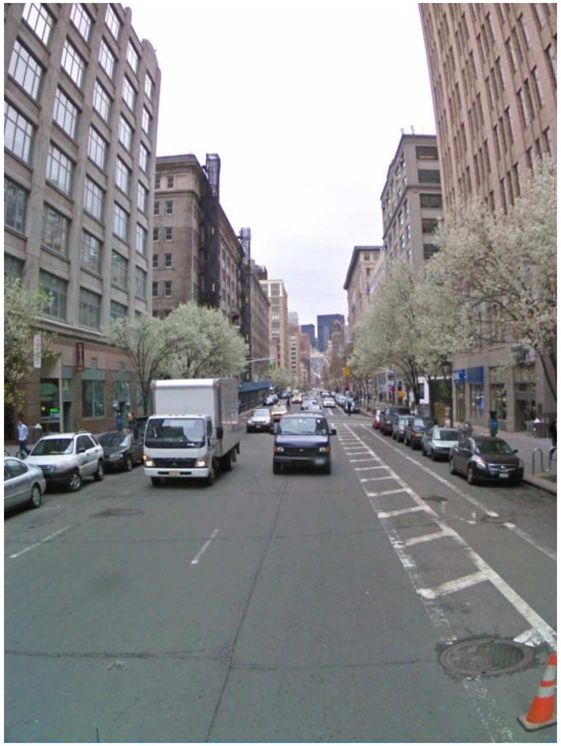
$$\mathbf{M}_{\mathrm{int}} = \begin{bmatrix} -f/s_x & 0 & o_x \\ 0 & -f/s_y & o_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{M}_{ext} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & -\mathbf{R}_1^{\mathrm{T}}\mathbf{T} \\ r_{21} & r_{22} & r_{23} & -\mathbf{R}_2^{\mathrm{T}}\mathbf{T} \\ r_{31} & r_{32} & r_{33} & -\mathbf{R}_3^{\mathrm{T}}\mathbf{T} \end{bmatrix}$$

Radial Distortion

- Finally, would like to include transformation to unwarp radial distortion:

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



Radial Distortion

- Radial distortion cannot be represented by matrix

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

- (c_u, c_v) is image center,

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

κ is first-order radial distortion coefficient

A Camera Model

Camera orientation

Scale to pixel size

Camera location

Translate to image center

Perspective projection

3D point (homogeneous coords)

Then unwarp radial distortion

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

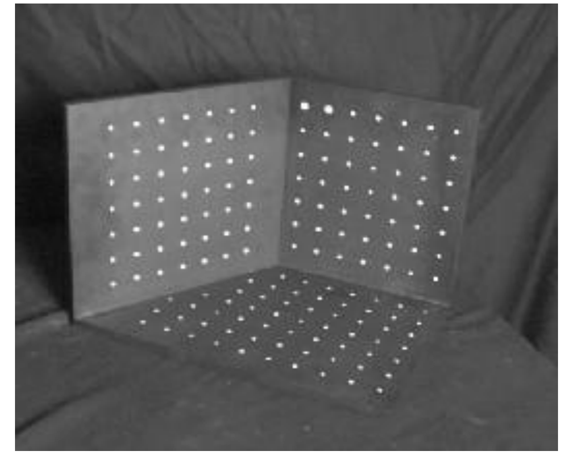
Then perform homogeneous divide, and get (u,v) coords

Outline

- Camera model
- Camera calibration
- Camera triangulation
- Structure from motion

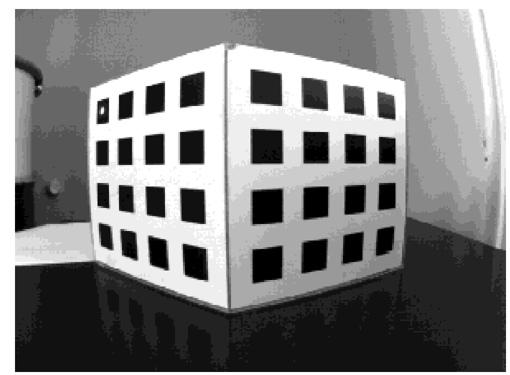
Camera Calibration

- Compute intrinsic and extrinsic parameters using observed camera data



Main idea

- Place “calibration object” with known geometry in the scene
- Get correspondences
- Solve for mapping from scene to image: estimate $\mathbf{M}=\mathbf{M}_{int}\mathbf{M}_{ext}$



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

- Input:
  - 3D $\leftrightarrow$ 2D correspondences
  - General perspective camera model (no radial distortion, for now)

$$\begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} \cong \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}$$

- Output:
  - Camera model parameters

Direct linear calibration

$$\begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} \cong \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}$$

$$u_i = \frac{m_{00}X_i + m_{01}Y_i + m_{02}Z_i + m_{03}}{m_{20}X_i + m_{21}Y_i + m_{22}Z_i + m_{23}}$$

$$v_i = \frac{m_{10}X_i + m_{11}Y_i + m_{12}Z_i + m_{13}}{m_{20}X_i + m_{21}Y_i + m_{22}Z_i + m_{23}}$$

$$u_i(m_{20}X_i + m_{21}Y_i + m_{22}Z_i + m_{23}) = m_{00}X_i + m_{01}Y_i + m_{02}Z_i + m_{03}$$
$$v_i(m_{20}X_i + m_{21}Y_i + m_{22}Z_i + m_{23}) = m_{10}X_i + m_{11}Y_i + m_{12}Z_i + m_{13}$$

$$\begin{bmatrix} X_i & Y_i & Z_i & 1 & 0 & 0 & 0 & 0 & -u_iX_i & -u_iY_i & -u_iZ_i & -u_i \\ 0 & 0 & 0 & 0 & X_i & Y_i & Z_i & 1 & -v_iX_i & -v_iY_i & -v_iZ_i & -v_i \end{bmatrix} \begin{bmatrix} m_{00} \\ m_{01} \\ m_{02} \\ m_{03} \\ m_{10} \\ m_{11} \\ m_{12} \\ m_{13} \\ m_{20} \\ m_{21} \\ m_{22} \\ m_{23} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Direct linear calibration

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -u_1X_1 & -u_1Y_1 & -u_1Z_1 & -u_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -v_1X_1 & -v_1Y_1 & -v_1Z_1 & -v_1 \\ & & & & & & \vdots & & & & & \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -u_nX_n & -u_nY_n & -u_nZ_n & -u_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -v_nX_n & -v_nY_n & -v_nZ_n & -v_n \end{bmatrix} \begin{bmatrix} m_{00} \\ m_{01} \\ m_{02} \\ m_{03} \\ m_{10} \\ m_{11} \\ m_{12} \\ m_{13} \\ m_{20} \\ m_{21} \\ m_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

Can solve for m_{ij} by linear least squares

Direct linear calibration

- Advantage:
  - Very simple to formulate and solve
- Disadvantages:
  - Doesn’t tell you the camera parameters
  - Doesn’t model radial distortion
  - Hard to impose constraints (e.g., known *f*)
  - Doesn’t minimize the right error function

Why?

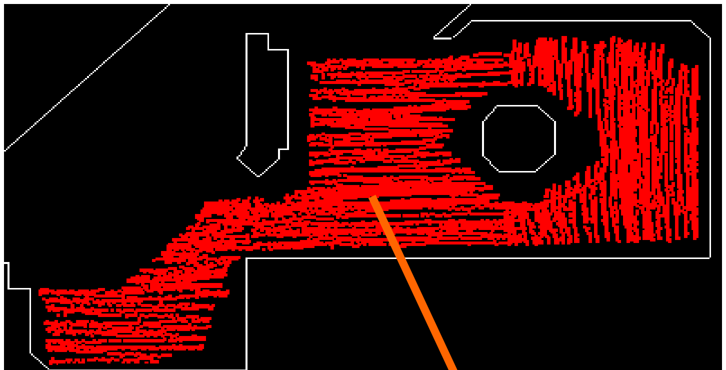
Nonlinear Camera Calibration

- Incorporating additional constraints into camera model
  - No shear, no scale (rigid-body motion)
  - Square pixels
  - Radial distortion
  - etc.
- These impose *nonlinear* constraints on camera parameters

Application: Sea of Images

Aliaga

Application: Sea of Images



Large-scale Dense Capture

Off-line:

- Camera calibration
- Feature correspondence

Real-time:

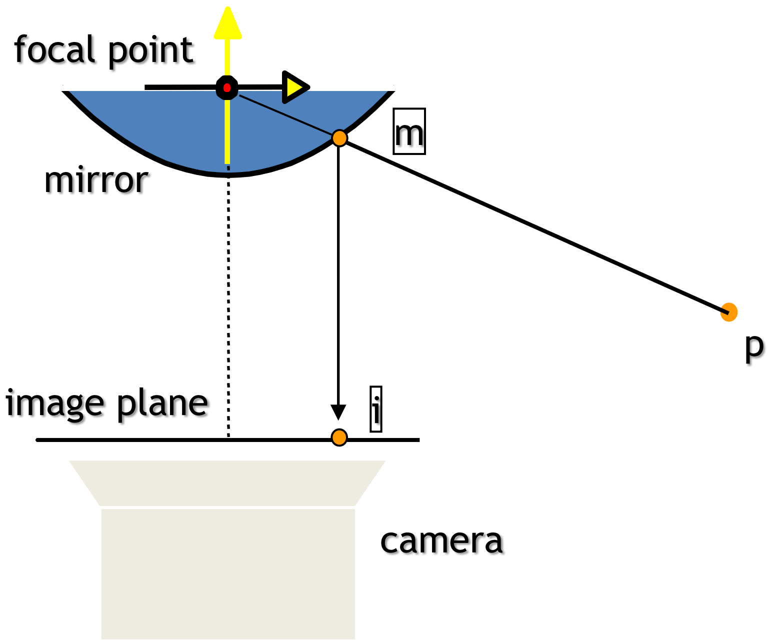
- Image warping
- Image composition

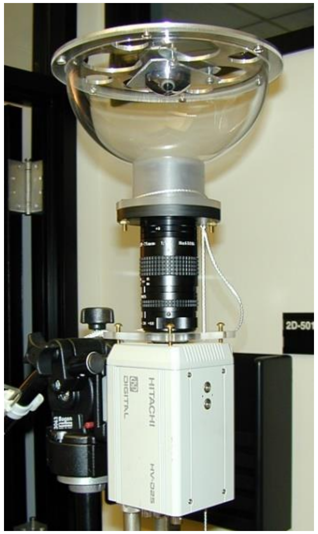
Interactive

Aliaga

Sea of Images Capture

Hemispherical FOV camera





Paraboloidal Catadioptric Camera
[Nayar97]

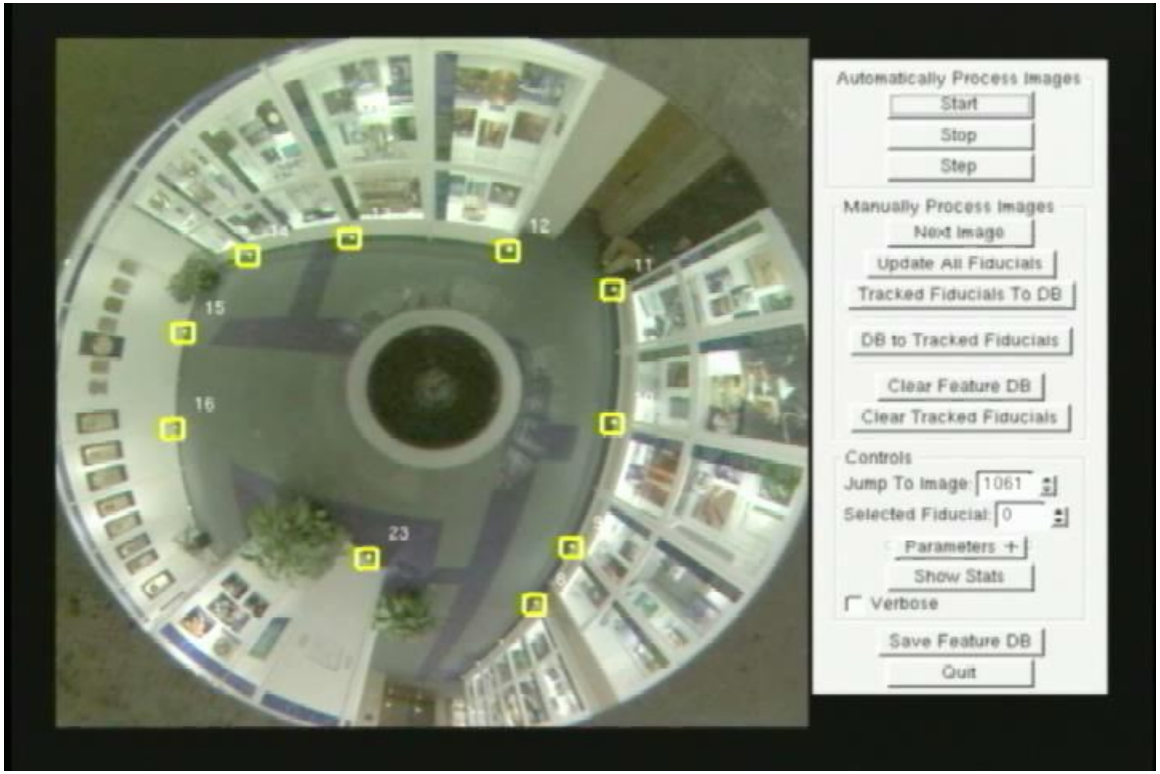
Sea of Images Capture

Capture lots of images with hemispherical FOV camera driven on cart

Aliaga

Sea of Images Calibration

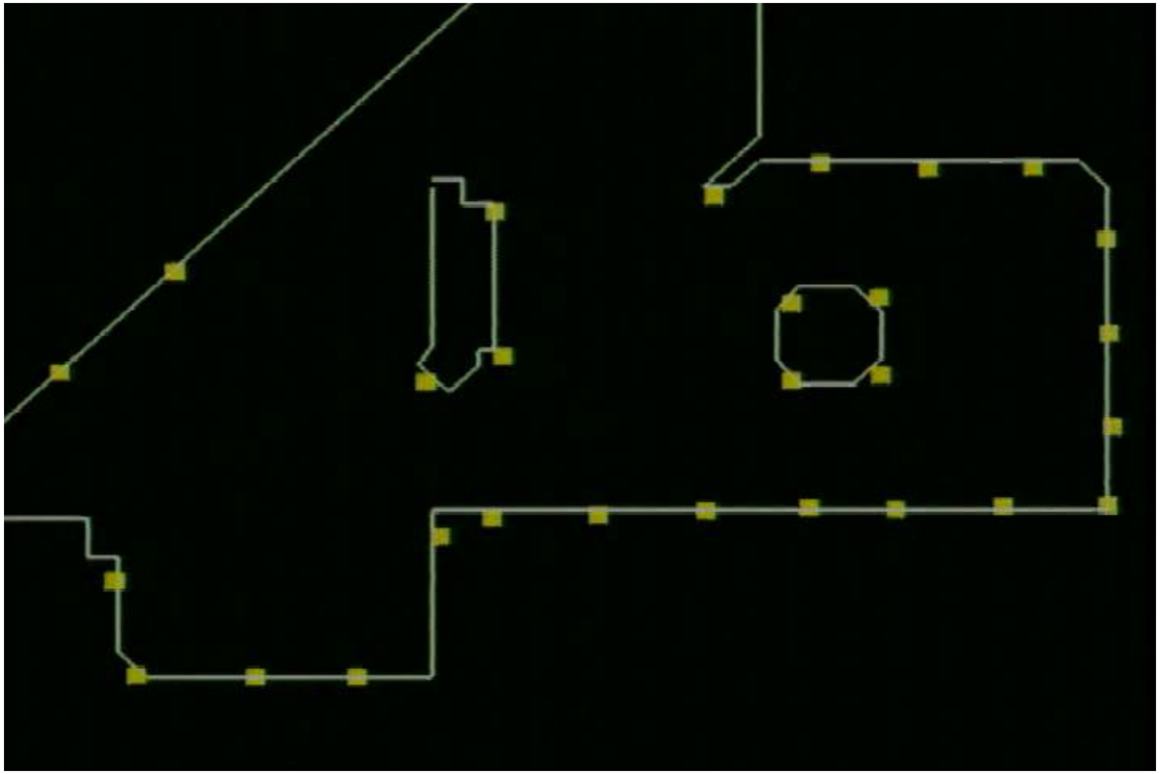
Calibrate camera based on positions of fiducials



Aliaga

Sea of Images Calibration

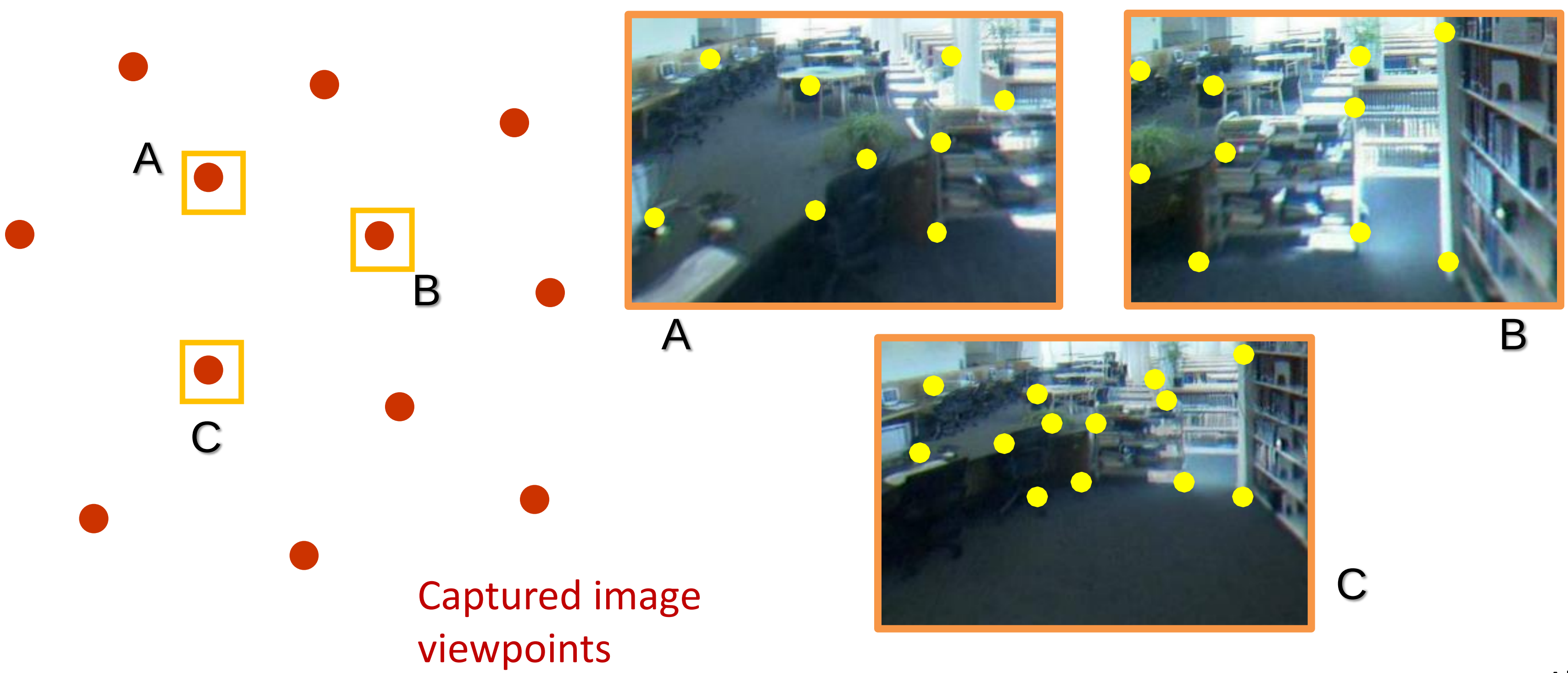
Result is a “sea of images” with known camera viewpoints spaced a few inches apart



Aliaga

Sea of Images Correspondence

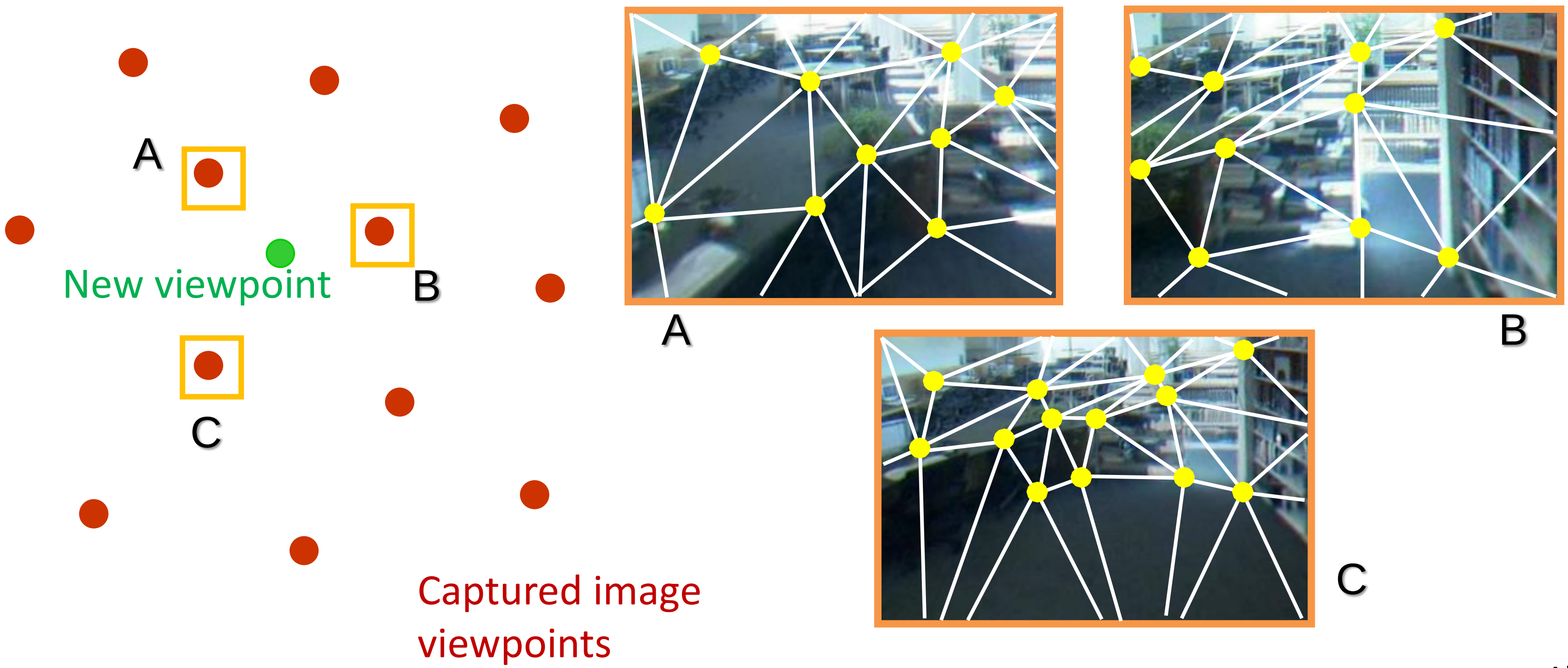
Search for feature correspondences in nearby images



Aliaga

Sea of Images Rendering

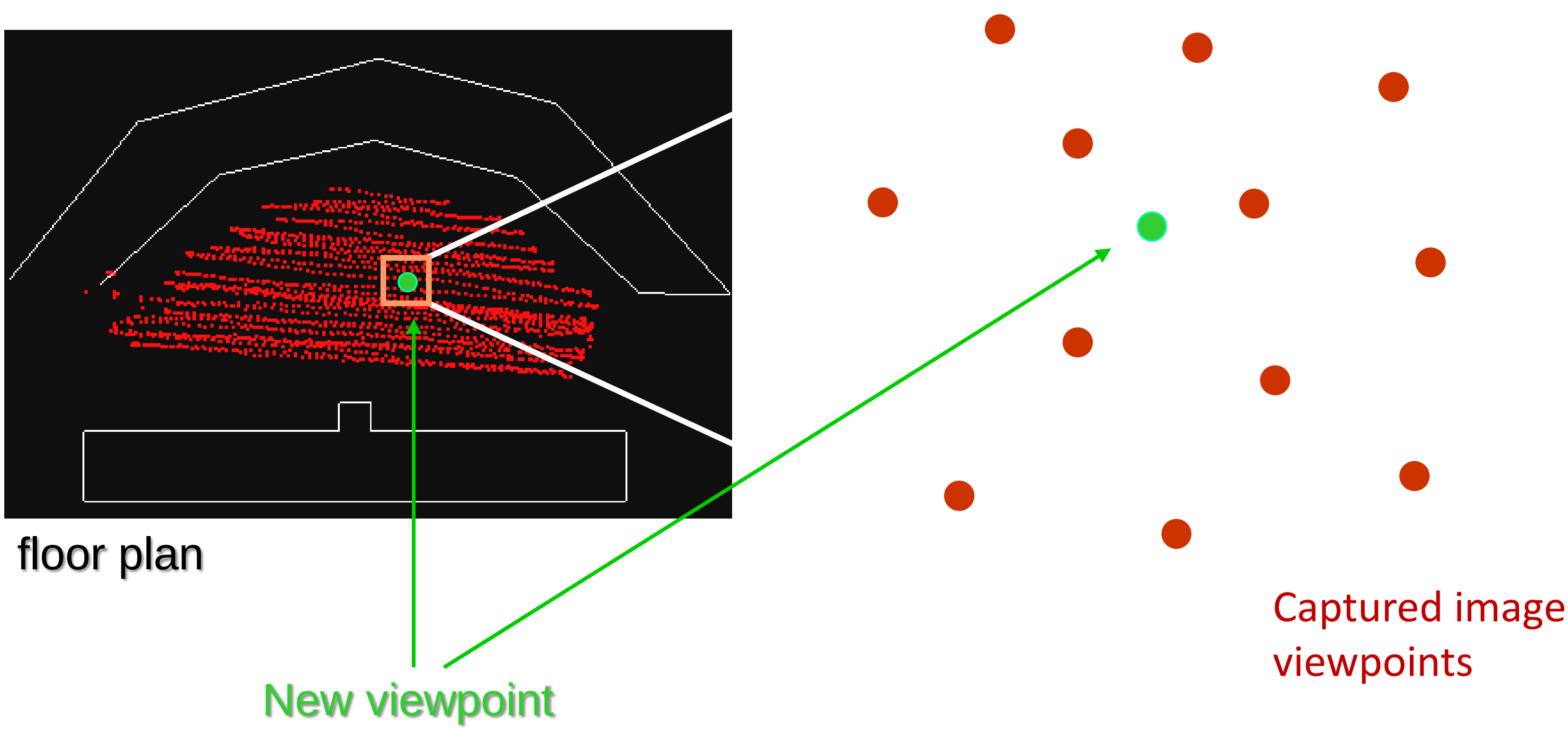
Create image from new viewpoint by warping and compositing three nearest images using triangles of feature correspondences



Aliaga

Sea of Images Rendering

Neat data management based on hierarchical encoding to enable interactive walkthrough



Aliaga

Sea of Images Results

- Bell Labs Museum
  - 900 square ft
  - 9832 images
  - 2.2 inch spacing
- Princeton Library
  - 120 square ft
  - 1947 images
  - 1.6 inches
- Personal Office
  - 30 square feet
  - 3475 images
  - 0.7 inches

Aliaga

Sea of Images Results

Aliaga

Sea of Images Results

Aliaga

Sea of Images Results

Aliaga

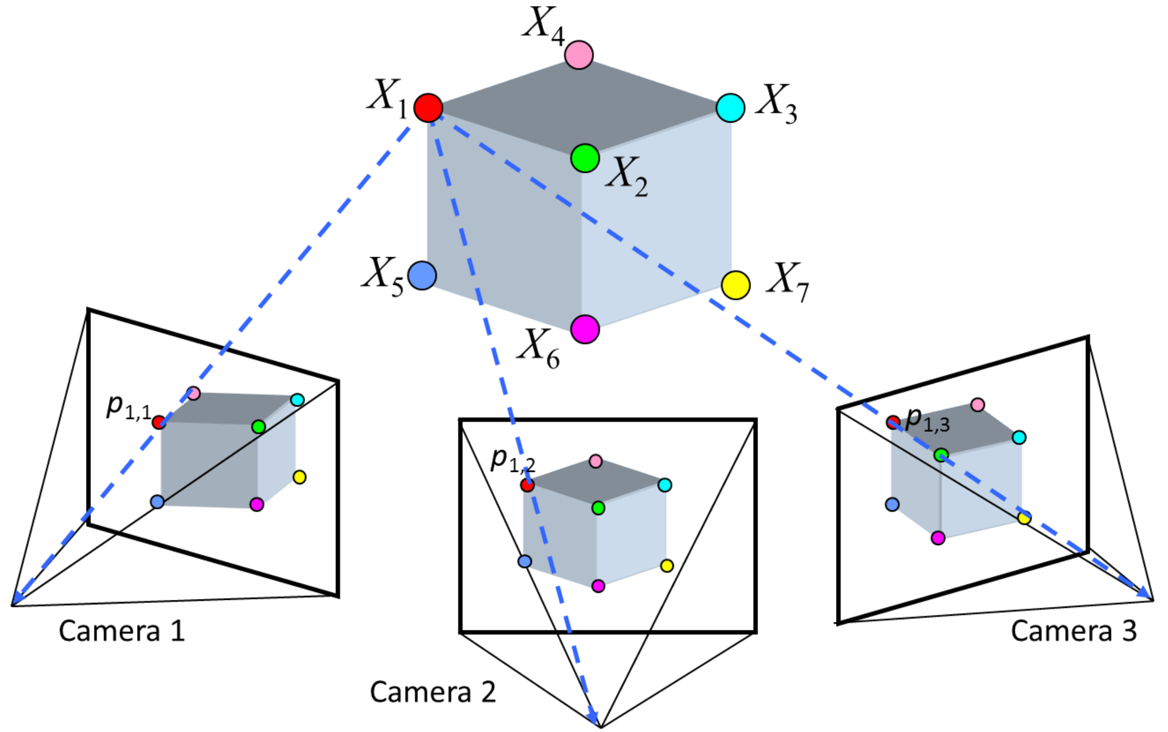
Outline

- Camera model
- Camera calibration
- Camera triangulation
- Structure from motion

Camera calibration summary

Suppose we know 3D point positions and correspondences to pixels in every image

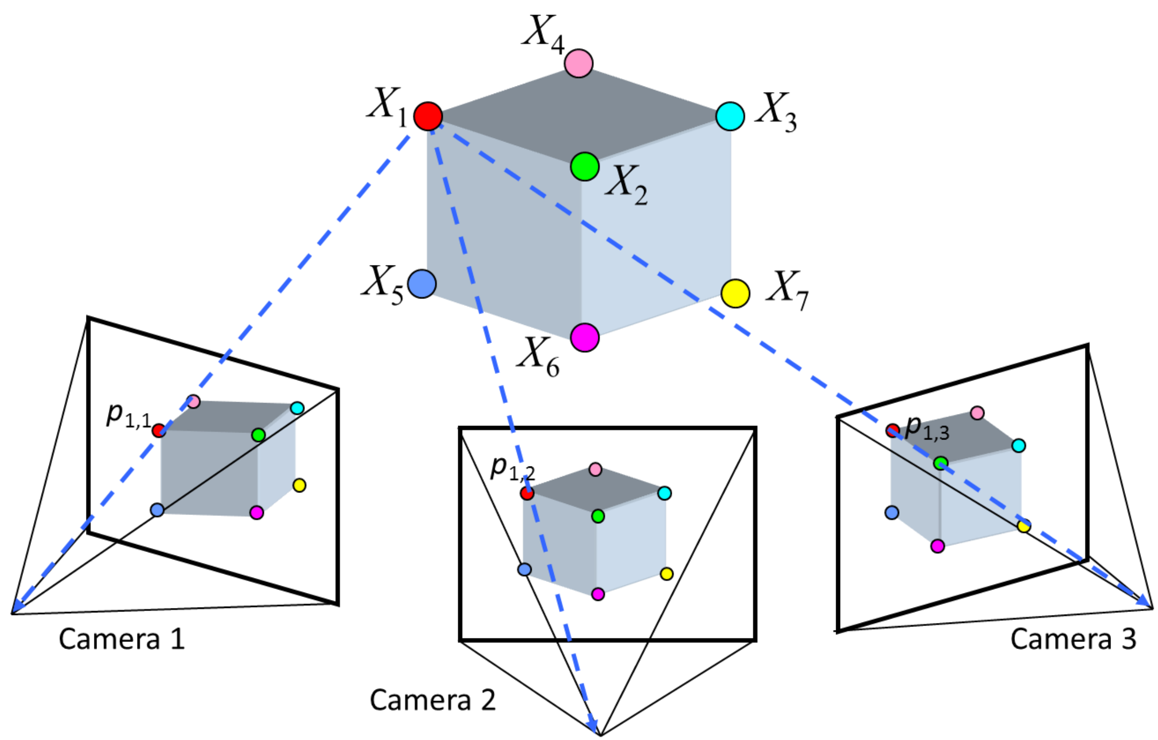
– We can compute the camera parameters



Noah Snavely

Camera triangulation

Suppose we know the camera parameters and correspondences between points and pixels in every image?

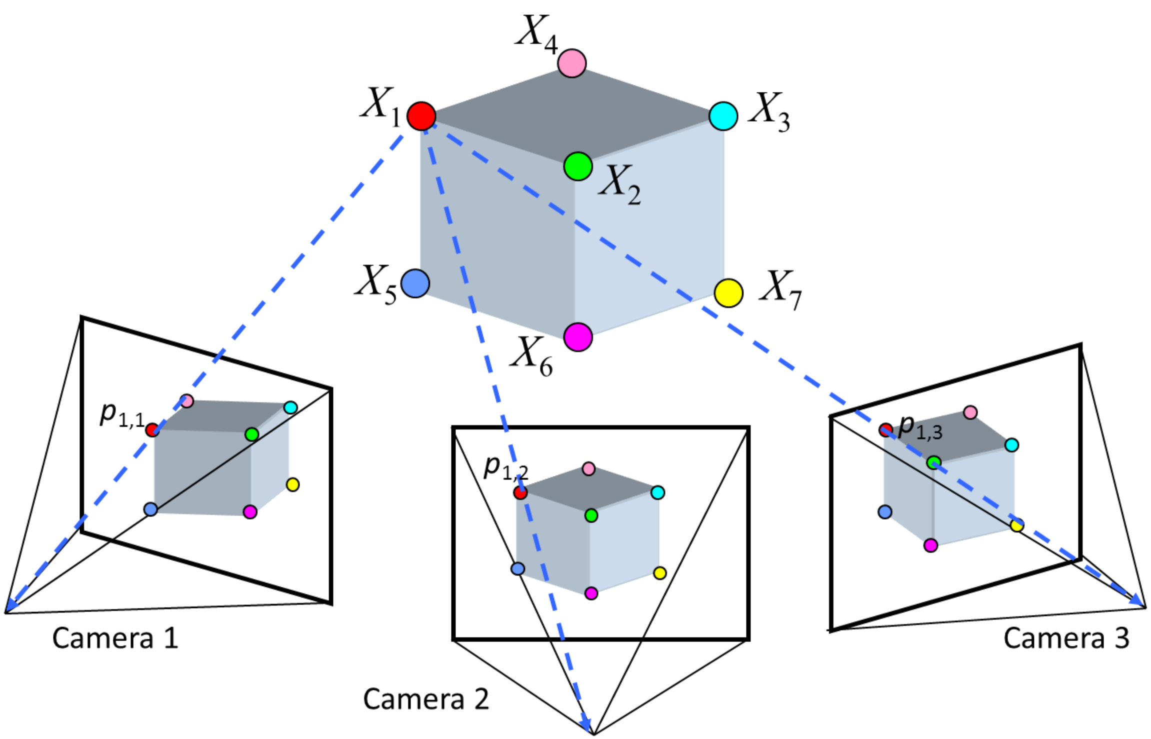


Noah Snavely

Camera calibration & triangulation

Suppose no camera parameters or point positions

- And have matches between these points and two images
- Can we compute them both together?



Noah Snavely

Outline

- Camera model
- Camera calibration
- Camera triangulation
- Structure from motion

Structure from motion

- Given many images
  a) figure out where they were all taken from?
  b) build a 3D model of the scene?

Noah Snavely

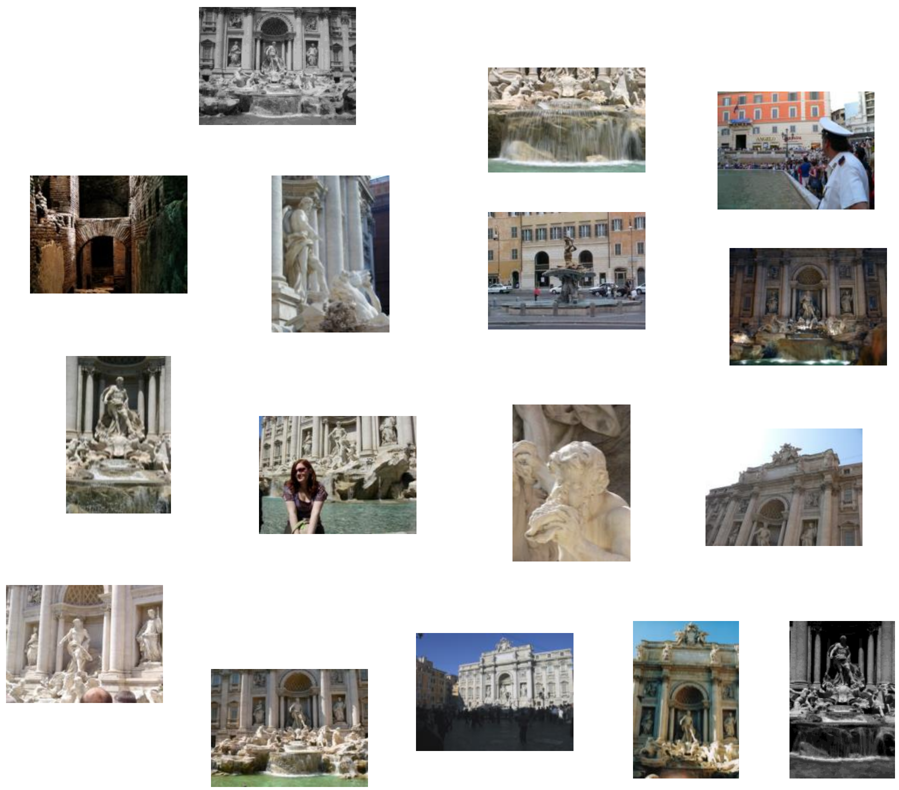
Structure from motion

- Feature detection
- Feature description
- Feature matching
- Feature correspondence
- Camera calibration and triangulation

Noah Snavely

Feature detection

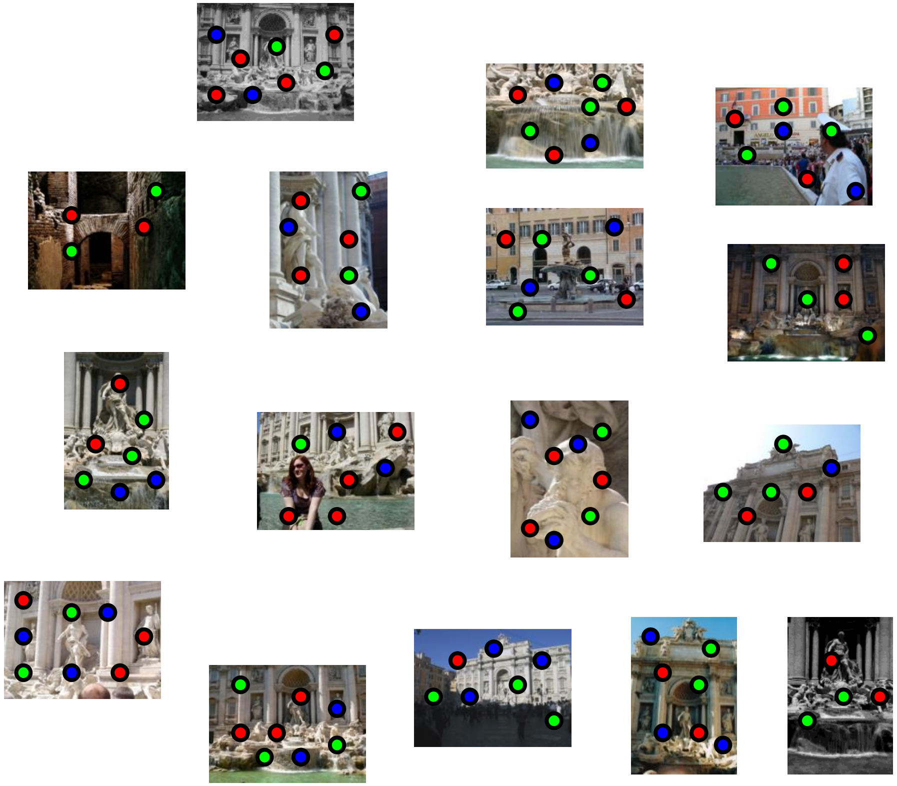
Detect features using SIFT [Lowe, IJCV 2004]



Noah Snavely

Feature detection

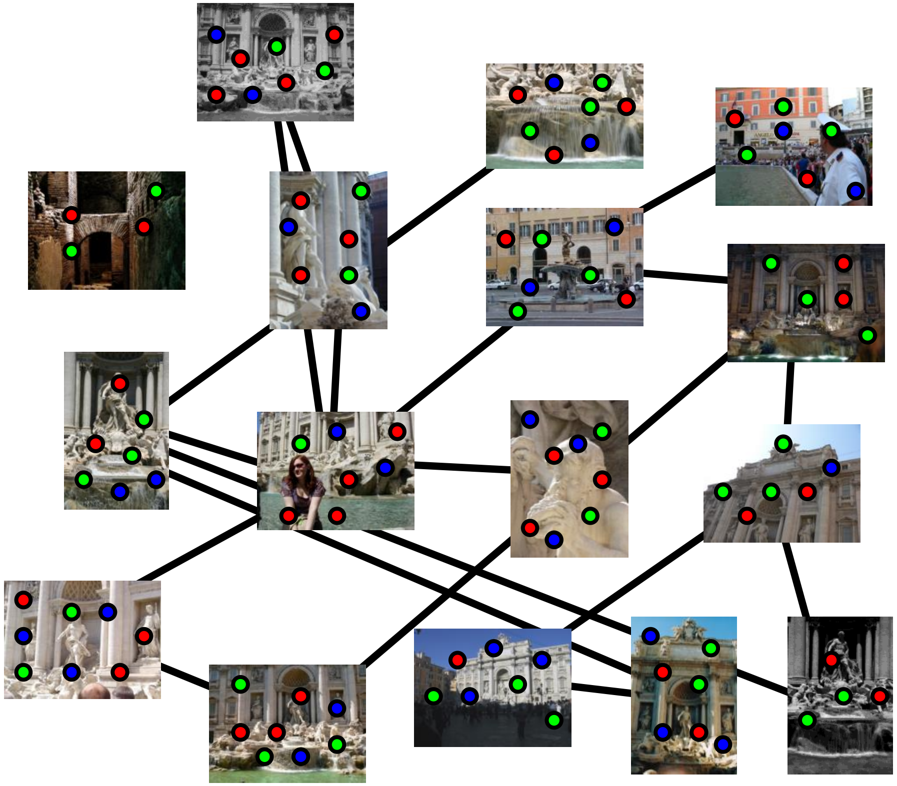
Detect features using SIFT [Lowe, IJCV 2004]



Noah Snavely

Feature matching

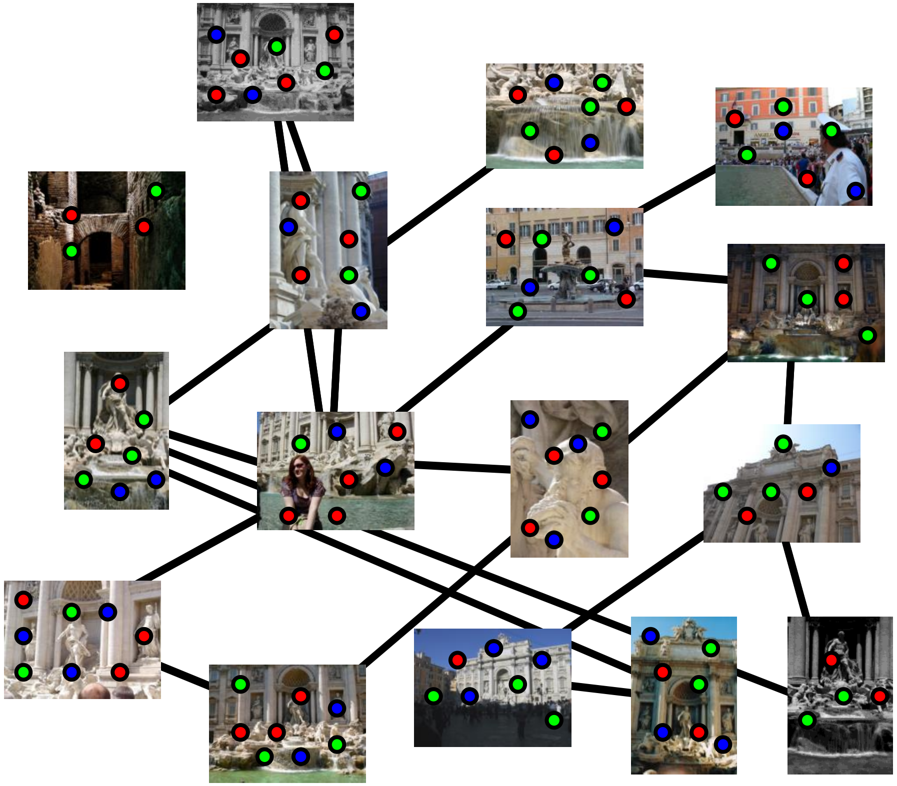
Match features between each pair of images



Noah Snavely

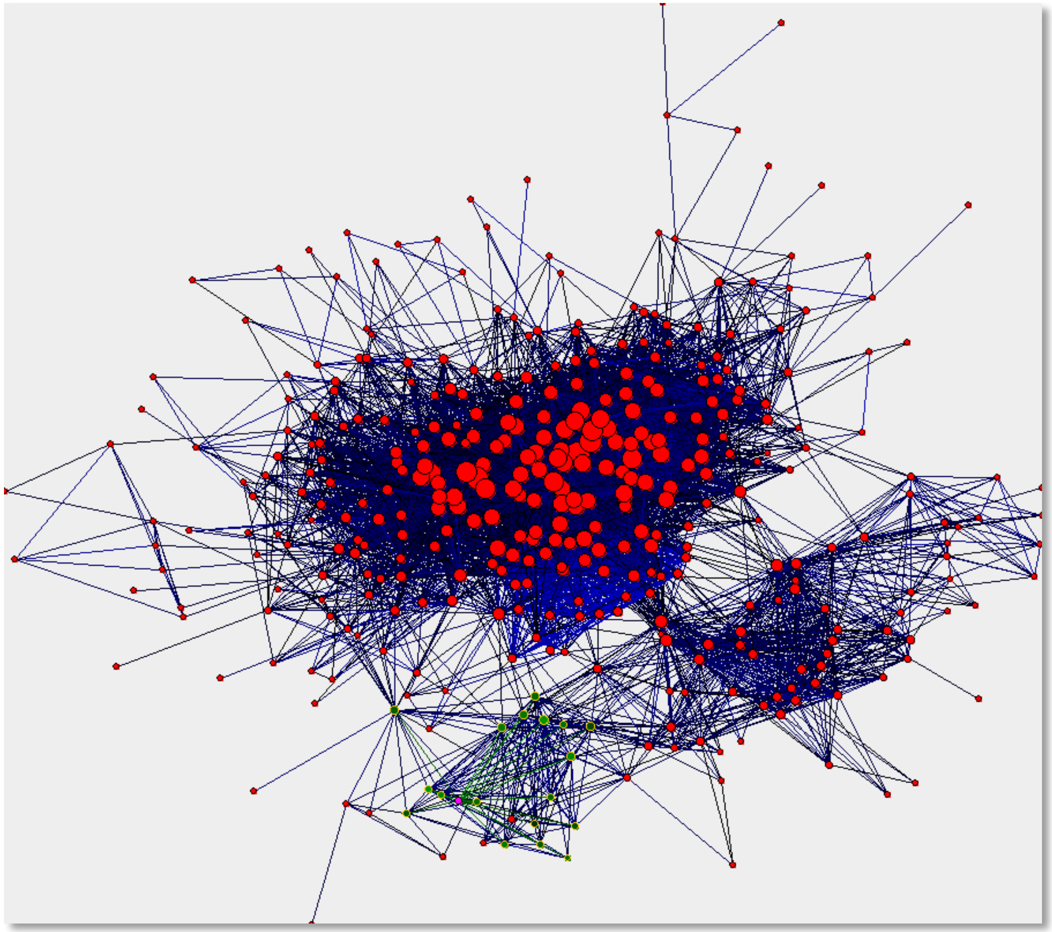
Feature matching

Refine matching using RANSAC to correspondences between each pair



Noah Snavely

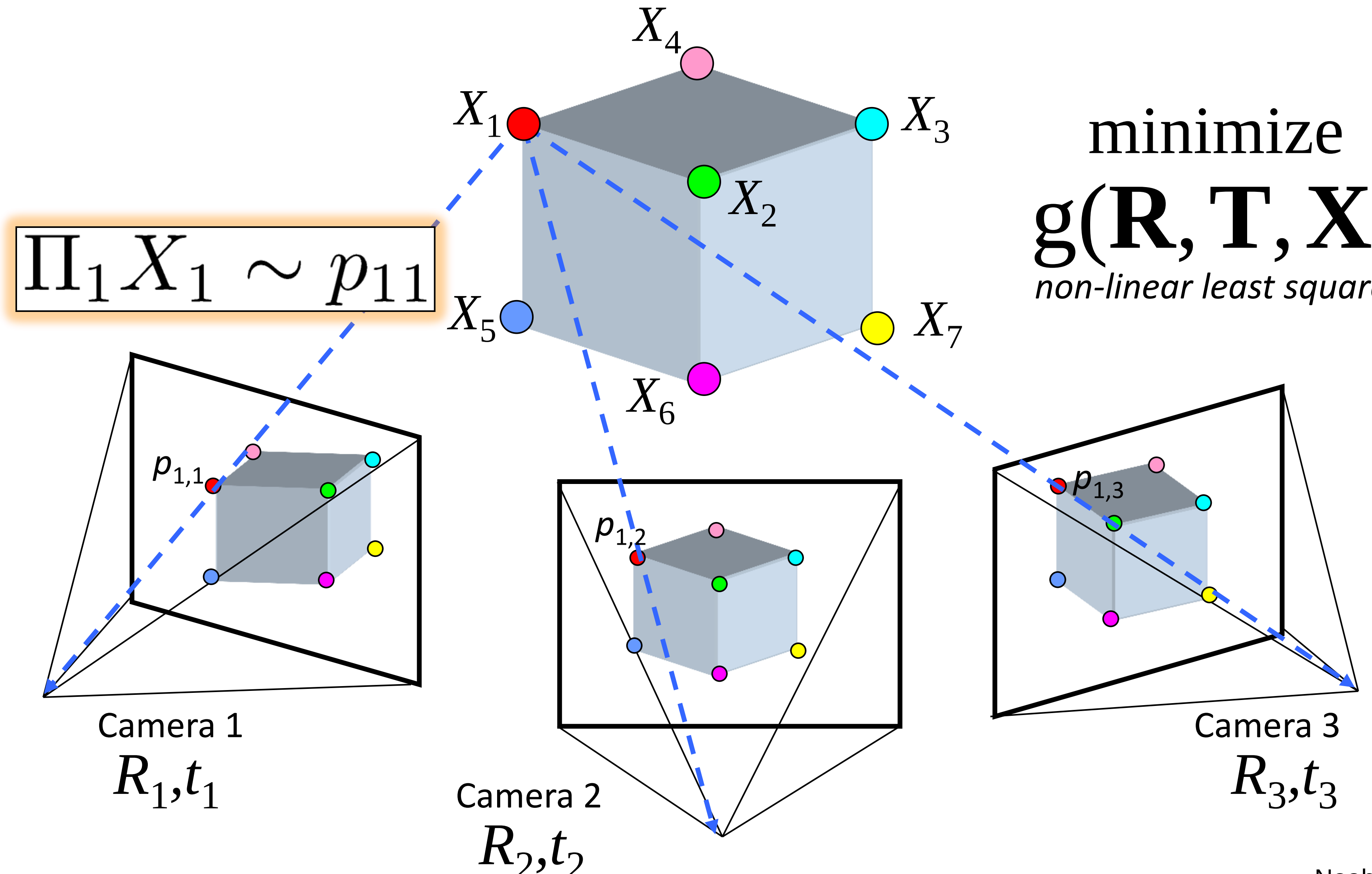
Image connectivity graph



(graph layout produced using the Graphviz toolkit: http://www.graphviz.org/)

Noah Snavely

Structure from motion



minimize

$$\mathrm{g}(\mathbf{R}, \mathbf{T}, \mathbf{X})$$

non-linear least squares

Noah Snavely

Structure from motion

- 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, e.g. Levenberg-Marquardt

Noah 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

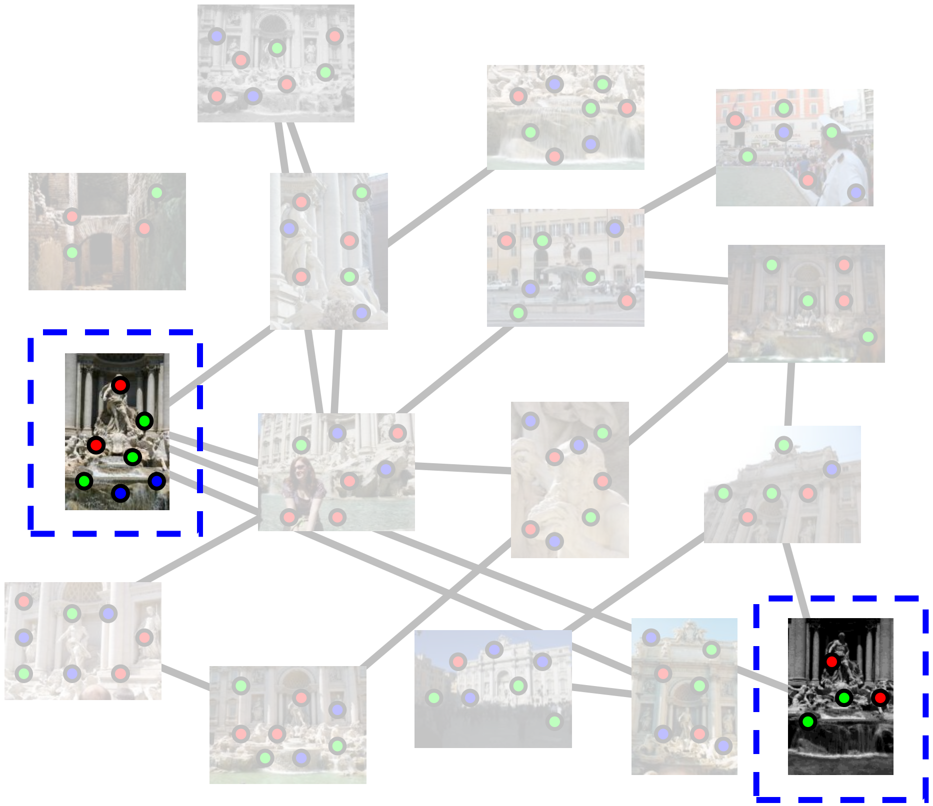
    = very large optimization problem

Noah Snavely

Structure from motion

Noah Snavely

Incremental structure from motion

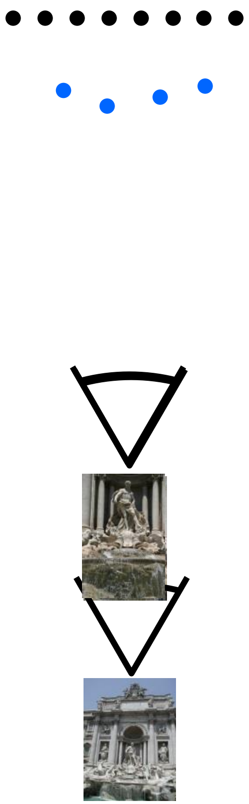


Noah Snavely

Incremental structure from motion

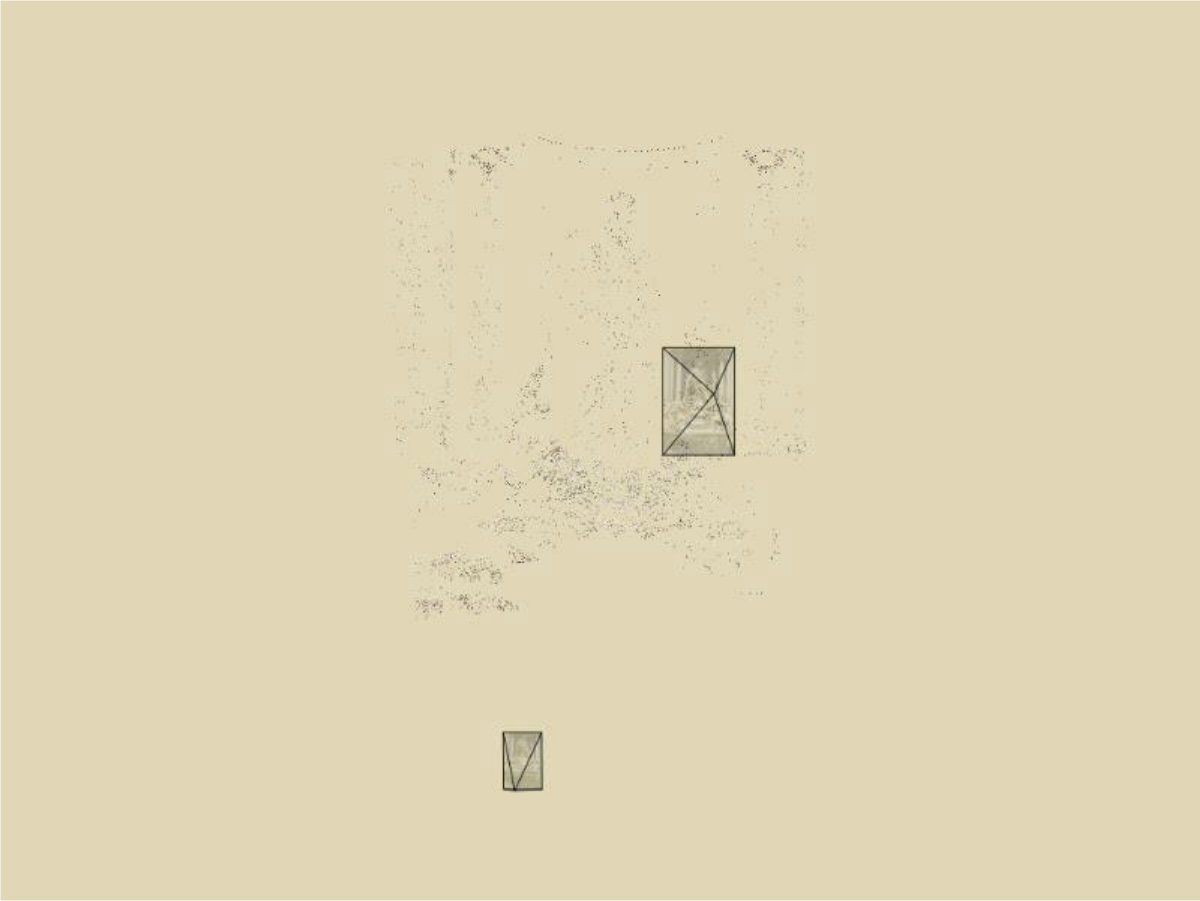
Noah Snavely

Incremental structure from motion



Noah Snavely

Incremental structure from motion



Noah Snavely

Incremental structure from motion

Noah Snavely

Structure from Motion Results

- For photo sets from the Internet, 20% to 75% of the photos were registered
- Most unregistered photos belonged to different connected components

> 1 week for 2600 photo

Noah Snavely

Structure from Motion Results

Noah Snavely

Photo Tourism:
Exploring Photo Collections in 3D

Noah Snavely

Steven M. Seitz

University of Washington

Richard Szeliski

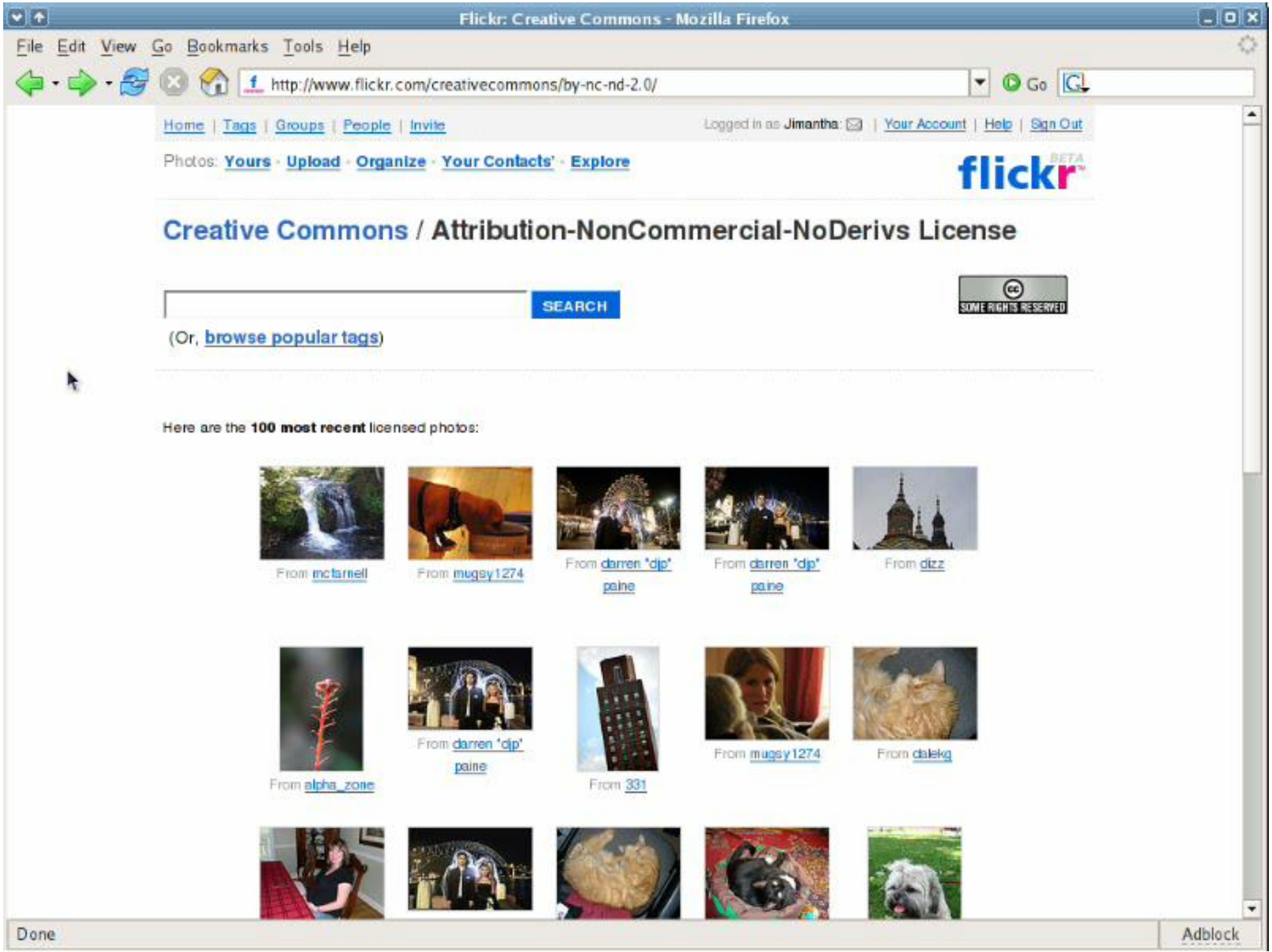
Microsoft Research

15,464

37,383

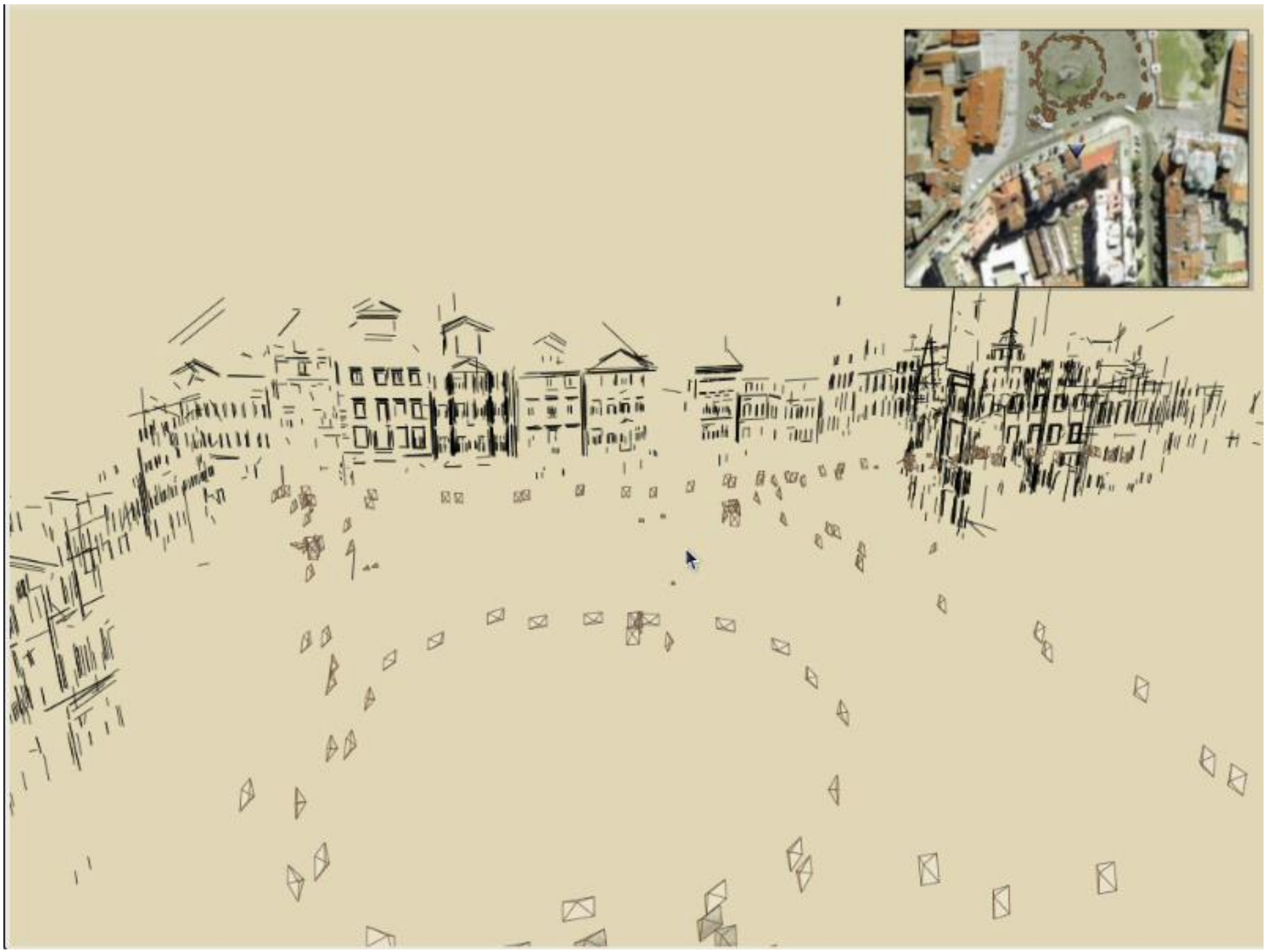
76,389

Noah Snavely



Noah Snavely

Example: Prague Old Town Square

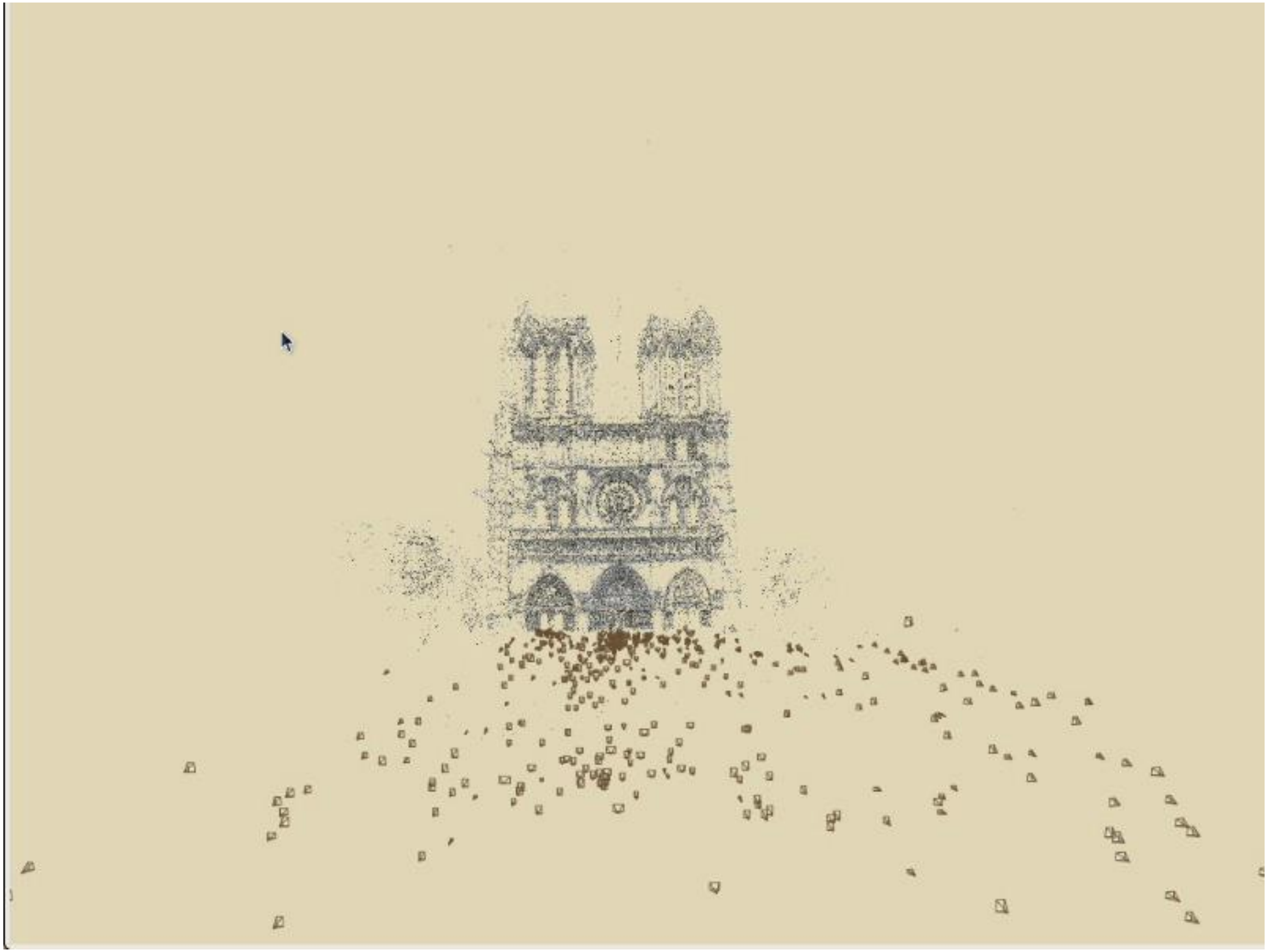


Noah Snavely

Rendering transitions

Noah Snavely

Annotations



Noah Snavely

Annotations

Noah Snavely

Summary

- Camera model
- Camera calibration
- Camera triangulation
- Structure from motion

Noah Snavely