Structure from Motion

Computer Vision
CS 543 / ECE 549
University of Illinois

Derek Hoiem

Many slides adapted from Lana Lazebnik, Silvio Saverese, Steve Seitz, Martial Hebert
This class: structure from motion

• Recap of epipolar geometry
  – Depth from two views

• Projective structure from motion

• Affine structure from motion
Recap: Epipoles

- Point $x$ in left image corresponds to **epipolar line** $l'$ in right image.
- Epipolar line passes through the epipole (the intersection of the cameras’ baseline with the image plane).
Recap: Fundamental Matrix

• Fundamental matrix maps from a point in one image to a line in the other
  \[ l' = Fx \quad l = F^\top x' \]

• If \( x \) and \( x' \) correspond to the same 3d point \( X \):
  \[ x'^\top Fx = 0 \]
Recap: Automatic Estimation of F

Assume we have matched points $x \leftrightarrow x'$ with outliers

8-Point Algorithm for Recovering F

- Correspondence Relation
  $$x'^T F x = 0$$

1. Normalize image coordinates
   $$\tilde{x} = T x \quad \tilde{x}' = T' x'$$

2. RANSAC with 8 points
   - Randomly sample 8 points
   - Compute $F$ via least squares
   - Enforce $\det(\tilde{F}) = 0$ by SVD
   - Repeat and choose $F$ with most inliers

3. De-normalize: $F = T'^T \tilde{F} T$
Recap

• We can get projection matrices $P$ and $P'$ up to a projective ambiguity (see HZ p. 255-256)

$$P = \begin{bmatrix} I & 0 \end{bmatrix} \quad P' = \begin{bmatrix} [e'] \times F & e' \end{bmatrix} \quad e'^T F = 0$$

See HZ p. 255-256

• Code:

```matlab
function P = vgg_P_from_F(F)
[U, S, V] = svd(F);
e = U(:, 3);
P = [-vgg_contreps(e)*F e];
```
Triangulation: Linear Solution

• Generally, rays $C \rightarrow x$ and $C' \rightarrow x'$ will not exactly intersect

• Can solve via SVD, finding a least squares solution to a system of equations

\[
\begin{bmatrix}
v \\ u \\ u' \\ v'
\end{bmatrix}
\begin{bmatrix}
p_{3}^{T} & -p_{1}^{T} \\ p_{3}^{T} & -p_{2}^{T} \\ p_{3}^{T} & -p_{1}^{T} \\ p_{3}^{T} & -p_{2}^{T}
\end{bmatrix}
\rightarrow
\begin{bmatrix}
X \\ P
\end{bmatrix}
\]

Further reading: HZ p. 312-313
Triangulation: Linear Solution

Given $P, P', x, x'$

1. Precondition points and projection matrices
2. Create matrix $A$
3. $[U, S, V] = \text{svd}(A)$
4. $X = V(:, \text{end})$

Pros and Cons

- Works for any number of corresponding images
- Not projectively invariant

$$\begin{align*}
x &= w \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \\
x' &= w' \begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix}
\end{align*}$$

$$\begin{bmatrix}
\begin{bmatrix}
p_1^T \\
p_2^T \\
p_3^T
\end{bmatrix} \\
\begin{bmatrix}
p_1'\!^T \\
p_2'\!^T \\
p_3'\!^T
\end{bmatrix}
\end{bmatrix}$$

$$\begin{bmatrix}
up_3^T - p_1^T \\
v p_3^T - p_2^T \\
u'p_3'\!^T - p_1'\!^T \\
v'p_3'\!^T - p_2'\!^T
\end{bmatrix}$$

Code: [http://www.robots.ox.ac.uk/~vgg/hzbook/code/vgg_multiview/vgg_X_from_xP_lin.m](http://www.robots.ox.ac.uk/~vgg/hzbook/code/vgg_multiview/vgg_X_from_xP_lin.m)
Triangulation: Non-linear Solution

- Minimize projected error while satisfying

\[
\hat{x}'^T F \hat{x} = 0
\]

\[
\text{cost}(X) = \text{dist}(x, \hat{x})^2 + \text{dist}(x', \hat{x}')^2
\]
Triangulation: Non-linear Solution

- Minimize projected error while satisfying
  \[ \hat{x}'^T F \hat{x} = 0 \]
  \[\text{cost}(X) = \text{dist}(x, \hat{x})^2 + \text{dist}(x', \hat{x}')^2\]

- Solution is a 6-degree polynomial of \( t \), minimizing
  \[ d(x, l(t))^2 + d(x', l'(t))^2 \]

Further reading: HZ p. 318
Projective structure from motion

- Given: $m$ images of $n$ fixed 3D points
  - $x_{ij} = P_i X_j$, $i = 1, \ldots, m$, $j = 1, \ldots, n$
- Problem: estimate $m$ projection matrices $P_i$ and $n$ 3D points $X_j$ from the $mn$ corresponding 2D points $x_{ij}$

Slides from Lana Lazebnik
Projective structure from motion

• Given: \( m \) images of \( n \) fixed 3D points
  • \( x_{ij} = P_i X_j, \quad i = 1, \ldots, m, \quad j = 1, \ldots, n \)

• Problem: estimate \( m \) projection matrices \( P_i \)
  and \( n \) 3D points \( X_j \) from the \( mn \) corresponding points \( x_{ij} \)

• With no calibration info, cameras and points can only be recovered up to a 4x4 projective transformation \( Q \):
  • \( X \rightarrow QX, \ P \rightarrow PQ^{-1} \)

• We can solve for structure and motion when
  • \( 2mn \geq 11m + 3n - 15 \)

• For two cameras, at least 7 points are needed
Sequential structure from motion

• Initialize motion from two images using fundamental matrix

• Initialize structure by triangulation

• For each additional view:
  – Determine projection matrix of new camera using all the known 3D points that are visible in its image – *calibration*
Sequential structure from motion

• Initialize motion from two images using fundamental matrix

• Initialize structure by triangulation

• For each additional view:
  – Determine projection matrix of new camera using all the known 3D points that are visible in its image – calibration
  – Refine and extend structure: compute new 3D points, re-optimize existing points that are also seen by this camera – triangulation
Sequential structure from motion

• Initialize motion from two images using fundamental matrix

• Initialize structure by triangulation

• For each additional view:
  – Determine projection matrix of new camera using all the known 3D points that are visible in its image – calibration
  – Refine and extend structure: compute new 3D points, re-optimize existing points that are also seen by this camera – triangulation

• Refine structure and motion: bundle adjustment
Bundle adjustment

- Non-linear method for refining structure and motion
- Minimizing reprojection error

\[ E(P, X) = \sum_{i=1}^{m} \sum_{j=1}^{n} D(x_{ij}, P_i X_j)^2 \]
Auto-calibration

• Auto-calibration: determining intrinsic camera parameters directly from uncalibrated images

• For example, we can use the constraint that a moving camera has a fixed intrinsic matrix
  – Compute initial projective reconstruction and find 3D projective transformation matrix $Q$ such that all camera matrices are in the form $P_i = K [R_i | t_i]$

• Can use constraints on the form of the calibration matrix, such as zero skew
Summary so far

• From two images, we can:
  – Recover fundamental matrix F
  – Recover canonical cameras P and P’ from F
  – Estimate 3D positions (if K is known) that correspond to each pixel

• For a moving camera, we can:
  – Initialize by computing F, P, X for two images
  – Sequentially add new images, computing new P, refining X, and adding points
  – Auto-calibrate assuming fixed calibration matrix to upgrade to similarity transform
Photo synth


http://photosynth.net/
3D from multiple images

Building Rome in a Day: Agarwal et al. 2009
Structure from motion under orthographic projection

3D Reconstruction of a Rotating Ping-Pong Ball

• Reasonable choice when
  • Change in depth of points in scene is much smaller than distance to camera
  • Cameras do not move towards or away from the scene

Orthographic projection for rotated/translated camera

\[
\begin{pmatrix}
  u \\
  v
\end{pmatrix} =
\begin{bmatrix}
  1 & 0 & 0 \\
  0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
  X \\
  Y \\
  Z
\end{bmatrix}
\quad
\begin{pmatrix}
  u_{fp} \\
  v_{fp}
\end{pmatrix} =
\begin{bmatrix}
  1 & 0 & 0 \\
  0 & 1 & 0
\end{bmatrix}
\begin{pmatrix}
  R_f' \\
  X_p \\
  Y_p \\
  Z_p
\end{pmatrix} + t_f
\]

\[
R_f =
\begin{bmatrix}
  1 & 0 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1
\end{bmatrix}
R_f'
\quad
\begin{pmatrix}
  u_{fp} \\
  v_{fp}
\end{pmatrix} =
R_f
\begin{bmatrix}
  X_p \\
  Y_p \\
  Z_p
\end{bmatrix} + t_f
\]
Affine structure from motion

- Affine projection is a linear mapping + translation in inhomogeneous coordinates

\[ \mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} = \mathbf{AX} + \mathbf{t} \]

1. We are given corresponding 2D points \((\mathbf{x})\) in several frames
2. We want to estimate the 3D points \((\mathbf{X})\) and the affine parameters of each camera \((\mathbf{A})\)
Step 1: Simplify by getting rid of \( t \): shift to centroid of points for each camera

\[
x_i = A_i X + t_i \quad \hat{x}_{ij} = x_{ij} - \frac{1}{n} \sum_{k=1}^{n} x_{ik}
\]

\[
x_{ij} - \frac{1}{n} \sum_{k=1}^{n} x_{ik} = A_i X_j + t_i - \frac{1}{n} \sum_{k=1}^{n} (A_k X_k + t_k) = A_i \left( X_j - \frac{1}{n} \sum_{k=1}^{n} X_k \right) = A_i \hat{X}_j
\]

\[
\hat{x}_{ij} = A_i \hat{X}_j
\]

2d normalized point (observed)  \quad \text{3d normalized point}  \quad \text{Linear (affine) mapping}
Suppose we know 3D points and affine camera parameters ... then, we can compute the observed 2d positions of each point

\[
\begin{bmatrix}
A_1 \\
A_2 \\
\vdots \\
A_m
\end{bmatrix}
\begin{bmatrix}
X_1 & X_2 & \cdots & X_n
\end{bmatrix}
\]

Camera Parameters (2mx3) 

3D Points (3xn)
What if we instead observe corresponding 2d image points?

Can we recover the camera parameters and 3d points?

$$\mathbf{D} = \begin{bmatrix}
\hat{x}_{11} & \hat{x}_{12} & \cdots & \hat{x}_{1n} \\
\hat{x}_{21} & \hat{x}_{22} & \cdots & \hat{x}_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
\hat{x}_{m1} & \hat{x}_{m2} & \cdots & \hat{x}_{mn}
\end{bmatrix} \Rightarrow \begin{bmatrix}
\mathbf{A}_1 \\
\mathbf{A}_2 \\
\vdots \\
\mathbf{A}_m
\end{bmatrix} \begin{bmatrix}
\mathbf{X}_1 \\
\mathbf{X}_2 \\
\vdots \\
\mathbf{X}_n
\end{bmatrix}$$

What rank is the matrix of 2D points?
Factorizing the measurement matrix

Measurements = Motion \times Shape

D = AX

Source: M. Hebert
Factorizing the measurement matrix

- Singular value decomposition of $D$:

\[
\begin{align*}
D & = U W V^T \\
\end{align*}
\]
Factorizing the measurement matrix

- Singular value decomposition of $D$:

$$D = U W V^T$$

To reduce to rank 3, we just need to set all the singular values to 0 except for the first 3.
Factorizing the measurement matrix

- Obtaining a factorization from SVD:

\[ \mathbf{D} = \mathbf{U}_3 \times 3 \mathbf{W}_3 \times \mathbf{V}^T_3 \]
Factorizing the measurement matrix

- Obtaining a factorization from SVD:

\[ D = U_3 \times W_3^{1/2} \times V_3^T \]

Possible decomposition:

\[ M = U_3 \begin{pmatrix} W_3^{1/2} \end{pmatrix} \quad S = W_3^{1/2} V_3^T \]
Affine ambiguity

- The decomposition is not unique. We get the same $D$ by using any $3\times3$ matrix $C$ and applying the transformations $A \rightarrow AC$, $X \rightarrow C^{-1}X$
- That is because we have only an affine transformation and we have not enforced any Euclidean constraints (like forcing the image axes to be perpendicular, for example)

Source: M. Hebert
Eliminating the affine ambiguity

- Orthographic: image axes are perpendicular and of unit length

\[ \mathbf{a}_1 \cdot \mathbf{a}_2 = 0 \]
\[ |\mathbf{a}_1|^2 = |\mathbf{a}_2|^2 = 1 \]

Source: M. Hebert
Solve for orthographic constraints

Three equations for each image $i$

\[
\begin{align*}
\tilde{a}_{i1}^T CC^T \tilde{a}_{i1} &= 1 \\
\tilde{a}_{i2}^T CC^T \tilde{a}_{i2} &= 1 \\
\tilde{a}_{i1}^T CC^T \tilde{a}_{i2} &= 0
\end{align*}
\]

- Solve for $L = CC^T$
- Recover $C$ from $L$ by Cholesky decomposition: $L = CC^T$
- Update $A$ and $X$: $A = \tilde{A}C$, $X = C^{-1}\tilde{X}$
Algorithm summary

- Given: \( m \) images and \( n \) tracked features \( x_{ij} \)
- For each image \( i \), center the feature coordinates
- Construct a \( 2m \times n \) measurement matrix \( D \):
  - Column \( j \) contains the projection of point \( j \) in all views
  - Row \( i \) contains one coordinate of the projections of all the \( n \) points in image \( i \)
- Factorize \( D \):
  - Compute SVD: \( D = U W V^T \)
  - Create \( U_3 \) by taking the first 3 columns of \( U \)
  - Create \( V_3 \) by taking the first 3 columns of \( V \)
  - Create \( W_3 \) by taking the upper left \( 3 \times 3 \) block of \( W \)
- Create the motion (affine) and shape (3D) matrices:
  \[ A = U_3 W_3^{\frac{1}{2}} \] \[ X = W_3^{\frac{1}{2}} V_3^T \]
- Eliminate affine ambiguity

Source: M. Hebert
Dealing with missing data

- So far, we have assumed that all points are visible in all views
- In reality, the measurement matrix typically looks something like this:

One solution:
- solve using a dense submatrix of visible points
- Iteratively add new cameras
Further reading

• Short explanation of Affine SfM: class notes from Lischinksi and Gruber

• Clear explanation of epipolar geometry and projective SfM
Review of Affine SfM from Interest Points

1. Detect interest points (e.g., Harris)

\[ \mu(\sigma_I, \sigma_D) = g(\sigma_I)^* \begin{bmatrix} I_x^2(\sigma_D) & I_x I_y(\sigma_D) \\ I_x I_y(\sigma_D) & I_y^2(\sigma_D) \end{bmatrix} \]

1. Image derivatives

\[ \det M = \lambda_1 \lambda_2 \]
\[ \text{trace } M = \lambda_1 + \lambda_2 \]

2. Square of derivatives

3. Gaussian filter \( g(\sigma_I) \)

4. Cornerness function – both eigenvalues are strong

\[ \text{har} = \det[\mu(\sigma_I, \sigma_D)] - \alpha[\text{trace}(\mu(\sigma_I, \sigma_D))^2] = g(I_x^2) g(I_y^2) - [g(I_x I_y)]^2 - \alpha[g(I_x^2) + g(I_y^2)]^2 \]

5. Non-maxima suppression
Review of Affine SfM from Interest Points

2. Correspondence via Lucas-Kanade tracking

a) Initialize \((x', y') = (x, y)\)

b) Compute \((u, v)\) by

\[
\begin{bmatrix}
\sum I_x I_x & \sum I_x I_y \\
\sum I_x I_y & \sum I_y I_y
\end{bmatrix}
\begin{bmatrix}
u \\
v
\end{bmatrix} = -
\begin{bmatrix}
\sum I_x I_t \\
\sum I_y I_t
\end{bmatrix}
\]

2nd moment matrix for feature patch in first image

\(I_t = I(x', y', t+1) - I(x, y, t)\)

Original \((x,y)\) position

d) Recalculate \(I_t\)

e) Repeat steps 2-4 until small change

- Use interpolation for subpixel values

c) Shift window by \((u, v)\):

\[x' = x' + u; \quad y' = y' + v;\]
Review of Affine SfM from Interest Points

3. Get Affine camera matrix and 3D points using Tomasi-Kanade factorization

\[ D = U \times W \times V^T \]

Solve for orthographic constraints
Tips for HW 3

• Problem 1: vanishing points
  – Use lots of lines to estimate vanishing points
  – For estimation of VP from lots of lines, see single-view geometry chapter, or use robust estimator of a central intersection point
  – For obtaining intrinsic camera matrix, numerical solver (e.g., `fsolve` in matlab) may be helpful

• Problem 3: epipolar geometry
  – Use reprojection distance for inlier check (make sure to compute line to point distance correctly)

• Problem 4: structure from motion
  – Use Matlab’s `chol` and `svd`
  – If you weren’t able to get tracking to work from HW2 can use provided points
Distance of point to epipolar line

\[ l = Fx = [a \ b \ c] \]

\[ x' = [u \ v \ 1] \]

\[ d(l, x') = \frac{|au + bv + c|}{\sqrt{a^2 + b^2}} \]
Next class

• Clustering and using clustered interest points for matching images in a large database