Grouping and Structure from Motion

Raquel Urtasun

TTI Chicago

March 12, 2013

Example of grouping techniques

- K-means style clustering, e.g., SLIC superpixels
- Normalized cuts
- Graph-based superpixels
- Mean-shift
- Watershed transform

Simple K-means

- Find three clusters in this data

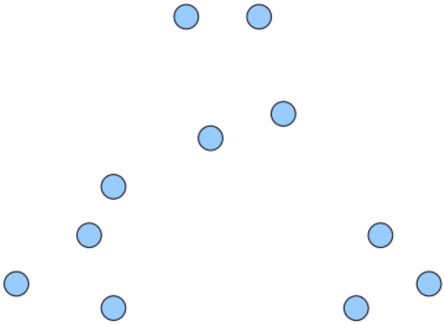


Figure: From M. Tappen

Simple K-means

- Find three clusters in this data

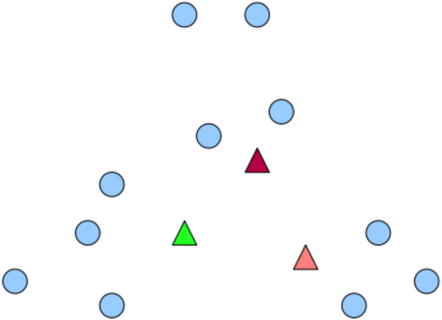


Figure: From M. Tappen

Simple K-means

- Find three clusters in this data

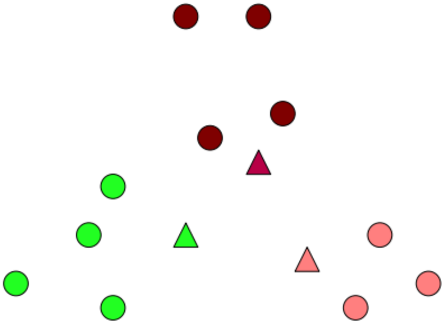


Figure: From M. Tappen

Simple K-means

- Find three clusters in this data

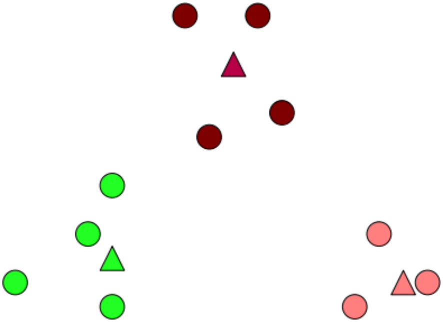


Figure: From M. Tappen

K-means style algorithms

- We would like to encode
  - Super-pixels have regular shape

K-means style algorithms

- We would like to encode
  - Super-pixels have regular shape
  - Pixels in super-pixels have similar appearance

K-means style algorithms

- We would like to encode
  - Super-pixels have regular shape
  - Pixels in super-pixels have similar appearance
- Let $\mathbf{S} = \{s_1, \cdots, s_m)$ be the set of superpixel assignments

K-means style algorithms

- We would like to encode
  - Super-pixels have regular shape
  - Pixels in super-pixels have similar appearance
- Let $\mathbf{S} = \{s_1, \cdots, s_m)$ be the set of superpixel assignments
- We define $\mu = \{\mu_1, \cdots, \mu_m\}$ as the mean location of each superpixel, and $\mathbf{c} = \{c_1, \cdots, c_m\}$ as the mean appearance descriptor.

K-means style algorithms

- We would like to encode
  - Super-pixels have regular shape
  - Pixels in super-pixels have similar appearance
- Let $\mathbf{S} = \{s_1, \cdots, s_m)$ be the set of superpixel assignments
- We define $\mu = \{\mu_1, \cdots, \mu_m\}$ as the mean location of each superpixel, and $\mathbf{c} = \{c_1, \cdots, c_m\}$ as the mean appearance descriptor.
- We can define the total energy of a pixel as

$$E(p) = E_{\mathrm{col}}(\mathbf{p}, c_{s_p}) + \lambda_{\mathrm{pos}} E_{\mathrm{pos}}(\mathbf{p}, \mu_{s_p})$$

K-means style algorithms

- We would like to encode
  - Super-pixels have regular shape
  - Pixels in super-pixels have similar appearance
- Let $\mathbf{S} = \{s_1, \cdots, s_m)$ be the set of superpixel assignments
- We define $\mu = \{\mu_1, \cdots, \mu_m\}$ as the mean location of each superpixel, and $\mathbf{c} = \{c_1, \cdots, c_m\}$ as the mean appearance descriptor.
- We can define the total energy of a pixel as

$$E(p) = E_{\mathrm{col}}(\mathbf{p}, c_{s_p}) + \lambda_{\mathrm{pos}} E_{\mathrm{pos}}(\mathbf{p}, \mu_{s_p})$$

- The problem becomes

$$\min_{\mathbf{S}, \mu, \mathbf{c}} \sum_{\mathbf{p}} E(\mathbf{p}, s_p, \mu_{s_p}, c_{s_p}).$$

K-means style algorithms

- We would like to encode
  - Super-pixels have regular shape
  - Pixels in super-pixels have similar appearance
- Let $\mathbf{S} = \{s_1, \cdots, s_m)$ be the set of superpixel assignments
- We define $\mu = \{\mu_1, \cdots, \mu_m\}$ as the mean location of each superpixel, and $\mathbf{c} = \{c_1, \cdots, c_m\}$ as the mean appearance descriptor.
- We can define the total energy of a pixel as

$$E(p) = E_{\text{col}}(\mathbf{p}, c_{s_p}) + \lambda_{\text{pos}} E_{\text{pos}}(\mathbf{p}, \mu_{s_p})$$

- The problem becomes

$$\min_{\mathbf{S},\mu,\mathbf{c}} \sum_{\mathbf{p}} E(\mathbf{p}, s_p, \mu_{s_p}, c_{s_p}).$$

K-means style algorithms

- We can define the total energy of a pixel as

$$E(p) = E_{\mathrm{col}}(\mathbf{p}, c_{s_p}) + \lambda_{\mathrm{pos}} E_{\mathrm{pos}}(\mathbf{p}, \mu_{s_p})$$

- The problem becomes

$$\min_{\mathbf{S},\mu,\mathbf{c}} \sum_{\mathbf{p}} E(\mathbf{p}, s_p, \mu_{s_p}, c_{s_p}).$$

- Simple iterative algorithm:
  - Solve for the assignments $\mathbf{S}$
  - Solve in parallel for the positions μ and appearances $\mathbf{c}$

K-means style algorithms

- We can define the total energy of a pixel as

$$E(p) = E_{\mathrm{col}}(\mathbf{p}, c_{s_p}) + \lambda_{\mathrm{pos}} E_{\mathrm{pos}}(\mathbf{p}, \mu_{s_p})$$

- The problem becomes

$$\min_{\mathbf{S},\mu,\mathbf{c}} \sum_{\mathbf{p}} E(\mathbf{p}, s_p, \mu_{s_p}, c_{s_p}).$$

- Simple iterative algorithm:
  - Solve for the assignments **S**
  - Solve in parallel for the positions μ and appearances **c**
- Is this easy to do?

K-means style algorithms

- We can define the total energy of a pixel as

$$E(p) = E_{\mathrm{col}}(\mathbf{p}, c_{s_p}) + \lambda_{\mathrm{pos}} E_{\mathrm{pos}}(\mathbf{p}, \mu_{s_p})$$

- The problem becomes

$$\min_{\mathbf{S}, \mu, \mathbf{c}} \sum_{\mathbf{p}} E(\mathbf{p}, s_p, \mu_{s_p}, c_{s_p}).$$

- Simple iterative algorithm:
  - Solve for the assignments **S**
  - Solve in parallel for the positions μ and appearances **c**
- Is this easy to do?

Results

[R. Achanta and A. Shaji and K. Smith and A. Lucchi and P. Fua and S. Susstrunk, PAMI12]

Joint Segmentation and Depth Estimation

- Let $\mathbf{S} = \{s_1, \cdots, s_m)$ be the set of superpixel assignments
- Let $\Theta = \{\theta_1, \cdots, \theta_m\}$ be the set of plane parameters

Joint Segmentation and Depth Estimation

- Let $\mathbf{S} = \{s_1, \cdots, s_m)$ be the set of superpixel assignments
- Let $\Theta = \{\theta_1, \cdots, \theta_m\}$ be the set of plane parameters
- We define $\mu = \{\mu_1, \cdots, \mu_m\}$ as the mean location of each superpixel, and $\mathbf{c} = \{c_1, \cdots, c_m\}$ as the mean appearance descriptor.

Joint Segmentation and Depth Estimation

- Let $\mathbf{S} = \{s_1, \cdots, s_m)$ be the set of superpixel assignments
- Let $\Theta = \{\theta_1, \cdots, \theta_m\}$ be the set of plane parameters
- We define $\mu = \{\mu_1, \cdots, \mu_m\}$ as the mean location of each superpixel, and $\mathbf{c} = \{c_1, \cdots, c_m\}$ as the mean appearance descriptor.
- We can define the total energy of a pixel as

$$E(p) = E_{\mathrm{col}}^{l,r}(\mathbf{p}, c_{s_p}, \theta_{s_p}) + \lambda_{\mathrm{pos}} E_{\mathrm{pos}}(\mathbf{p}, \mu_{s_p}) + \lambda_{\mathrm{disp}} E_{\mathrm{disp}}^{l,r}(\mathbf{p}, \theta_{s_p}),$$

Joint Segmentation and Depth Estimation

- Let $\mathbf{S} = \{s_1, \cdots, s_m)$ be the set of superpixel assignments
- Let $\Theta = \{\theta_1, \cdots, \theta_m\}$ be the set of plane parameters
- We define $\mu = \{\mu_1, \cdots, \mu_m\}$ as the mean location of each superpixel, and $\mathbf{c} = \{c_1, \cdots, c_m\}$ as the mean appearance descriptor.
- We can define the total energy of a pixel as

$$E(p) = E_{\mathrm{col}}^{l,r}(\mathbf{p}, c_{s_p}, \theta_{s_p}) + \lambda_{\mathrm{pos}} E_{\mathrm{pos}}(\mathbf{p}, \mu_{s_p}) + \lambda_{\mathrm{disp}} E_{\mathrm{disp}}^{l,r}(\mathbf{p}, \theta_{s_p}),$$

Joint Segmentation and Depth Estimation

- Let $\mathbf{S} = \{s_1, \cdots, s_m)$ be the set of superpixel assignments
- Let $\Theta = \{\theta_1, \cdots, \theta_m\}$ be the set of plane parameters
- We define $\mu = \{\mu_1, \cdots, \mu_m\}$ as the mean location of each superpixel, and $\mathbf{c} = \{c_1, \cdots, c_m\}$ as the mean appearance descriptor.
- We can define the total energy of a pixel as

$$E(p) = E_{\mathrm{col}}^{l,r}(\mathbf{p}, c_{s_p}, \theta_{s_p}) + \lambda_{\mathrm{pos}} E_{\mathrm{pos}}(\mathbf{p}, \mu_{s_p}) + \lambda_{\mathrm{disp}} E_{\mathrm{disp}}^{l,r}(\mathbf{p}, \theta_{s_p}),$$

- We can use:

$$E_{pos}(\mathbf{p}, \mu_{s_p}) = ||\mathbf{p} - \mu_{s_p}||_2^2 / g \qquad E_{col}(\mathbf{p}, c_{s_p} = (I_t(\mathbf{p}) - c_{s_p})^2$$

and

$$E_{disp}(\mathbf{p}, \theta_{s_p}) = \begin{cases} (d(\mathbf{p}, \theta_{s_p}) - \hat{d}(\mathbf{p}))^2 & \text{if } \mathbf{p} \in \mathcal{F} \\ \lambda & \text{otherwise} \end{cases}$$

Joint Segmentation and Depth Estimation

- Let $\mathbf{S} = \{s_1, \cdots, s_m)$ be the set of superpixel assignments
- Let $\Theta = \{\theta_1, \cdots, \theta_m\}$ be the set of plane parameters
- We define $\mu = \{\mu_1, \cdots, \mu_m\}$ as the mean location of each superpixel, and $\mathbf{c} = \{c_1, \cdots, c_m\}$ as the mean appearance descriptor.
- We can define the total energy of a pixel as

$$E(p) = E_{\rm col}^{l,r}(\mathbf{p}, c_{s_p}, \theta_{s_p}) + \lambda_{\rm pos} E_{\rm pos}(\mathbf{p}, \mu_{s_p}) + \lambda_{\rm disp} E_{\rm disp}^{l,r}(\mathbf{p}, \theta_{s_p}),$$

- We can use:

$$E_{pos}(\mathbf{p}, \mu_{s_p}) = ||\mathbf{p} - \mu_{s_p}||_2^2 / g \qquad E_{col}(\mathbf{p}, c_{s_p} = (I_t(\mathbf{p}) - c_{s_p})^2$$

and

$$E_{disp}(\mathbf{p}, \theta_{s_p}) = \begin{cases} (d(\mathbf{p}, \theta_{s_p}) - \hat{d}(\mathbf{p}))^2 & \text{if } \mathbf{p} \in \mathcal{F} \\ \lambda & \text{otherwise} \end{cases}$$

Joint Segmentation and Depth Estimation

- We can define the total energy of a pixel as

$$E(p) = E_{\mathrm{col}}^{l,r}(\mathbf{p}, c_{s_p}, \theta_{s_p}) + \lambda_{\mathrm{pos}} E_{\mathrm{pos}}(\mathbf{p}, \mu_{s_p}) + \lambda_{\mathrm{disp}} E_{\mathrm{disp}}^{l,r}(\mathbf{p}, \theta_{s_p}),$$

- The problem of joint unsupervised segmentation and flow estimation becomes

$$\min_{\Theta, \mathbf{S}, \mu, \mathbf{c}} \sum_{\mathbf{p}} E(\mathbf{p}, s_p, \theta_{s_p}, \mu_{s_p}, c_{s_p}).$$

Joint Segmentation and Depth Estimation

- We can define the total energy of a pixel as

$$E(p) = E_{\mathrm{col}}^{l,r}(\mathbf{p}, c_{s_p}, \theta_{s_p}) + \lambda_{\mathrm{pos}} E_{\mathrm{pos}}(\mathbf{p}, \mu_{s_p}) + \lambda_{\mathrm{disp}} E_{\mathrm{disp}}^{l,r}(\mathbf{p}, \theta_{s_p}),$$

- The problem of joint unsupervised segmentation and flow estimation becomes

$$\min_{\Theta, \mathbf{S}, \mu, \mathbf{c}} \sum_{\mathbf{p}} E(\mathbf{p}, s_p, \theta_{s_p}, \mu_{s_p}, c_{s_p}).$$

- Simple iterative algorithm
  - Solve for the assignments $\mathbf{S}$
  - Solve in parallel for the planes Θ, positions μ and appearances $\mathbf{c}$

- We can define the total energy of a pixel as

$$E(p) = E_{\mathrm{col}}^{l,r}(\mathbf{p}, c_{s_p}, \theta_{s_p}) + \lambda_{\mathrm{pos}} E_{\mathrm{pos}}(\mathbf{p}, \mu_{s_p}) + \lambda_{\mathrm{disp}} E_{\mathrm{disp}}^{l,r}(\mathbf{p}, \theta_{s_p}),$$

- The problem of joint unsupervised segmentation and flow estimation becomes

$$\min_{\Theta, \mathbf{S}, \mu, \mathbf{c}} \sum_{\mathbf{p}} E(\mathbf{p}, s_p, \theta_{s_p}, \mu_{s_p}, c_{s_p}).$$

- Simple iterative algorithm
  - Solve for the assignments **S**
  - Solve in parallel for the planes Θ, positions μ and appearances **c**
- How do we do this?

Joint Segmentation and Depth Estimation

- We can define the total energy of a pixel as

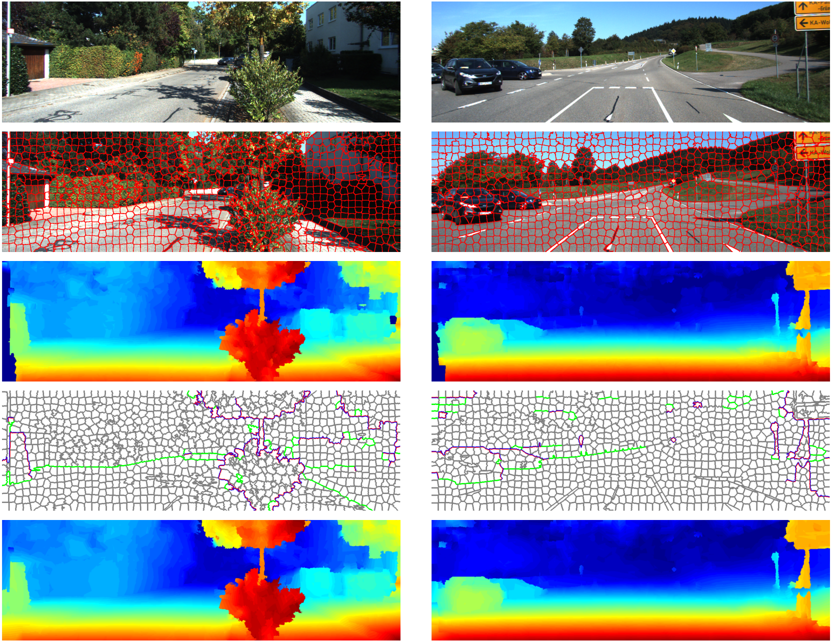
$$E(p) = E_{\mathrm{col}}^{l,r}(\mathbf{p}, c_{s_p}, \theta_{s_p}) + \lambda_{\mathrm{pos}} E_{\mathrm{pos}}(\mathbf{p}, \mu_{s_p}) + \lambda_{\mathrm{disp}} E_{\mathrm{disp}}^{l,r}(\mathbf{p}, \theta_{s_p}),$$

- The problem of joint unsupervised segmentation and flow estimation becomes

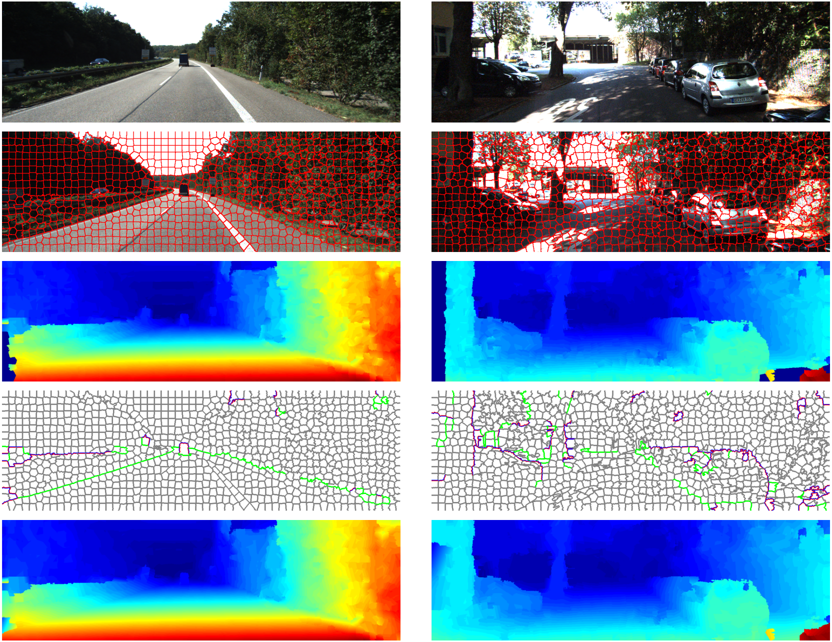
$$\min_{\Theta, \mathbf{S}, \mu, \mathbf{c}} \sum_{\mathbf{p}} E(\mathbf{p}, s_p, \theta_{s_p}, \mu_{s_p}, c_{s_p}).$$

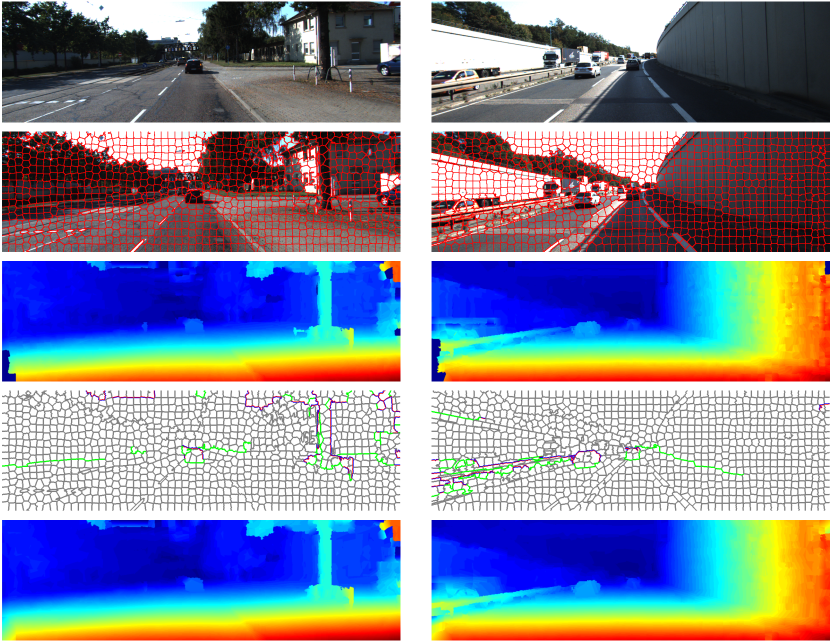
- Simple iterative algorithm
  - Solve for the assignments **S**
  - Solve in parallel for the planes Θ, positions μ and appearances **c**
- How do we do this?

[K. Yamaguchi, D. McAllester and R. Urtasun, CVPR13]



[K. Yamaguchi, D. McAllester and R. Urtasun, CVPR13]

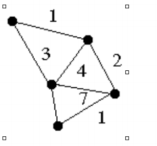




Example of grouping techniques

- K-means style clustering, e.g., SLIC superpixels
- Normalized cuts
- Graph-based superpixels
- Mean-shift
- Watershed transform

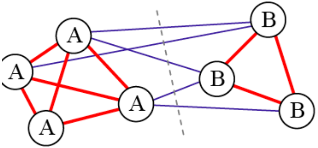
Segmentation as a mincut problem



$$\begin{bmatrix} 0 & 1 & 3 & \infty & \infty \\ 1 & 0 & 4 & \infty & 2 \\ 3 & 4 & 0 & 6 & 7 \\ \infty & \infty & 6 & 0 & 1 \\ \infty & 2 & 7 & 1 & 0 \end{bmatrix}$$

Weight Matrix: W

- Examines the **affinities** (similarities) between nearby pixels and tries to separate groups that are connected with weak affinities.



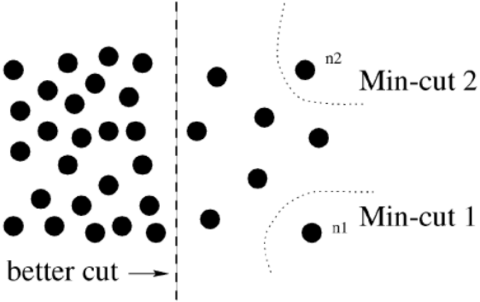
- The cut separate the nodes into two groups

Minimun Cuts

- The cut between two groups A and B is defined as the sum of all the weights being cut

$$cut(A,B) = \sum_{i \in A, j \in B} w_{i,j}$$

- Problem: Results in small cuts that isolates single pixels



- We need to normalize somehow

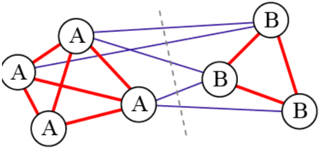
Normalized Cuts

[J. Shi and J. Malik, PAMI00]

- Better measure is the normalized cuts

$$N_{cut}(A,B) = \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)}$$

with $assoc(A,A) = \sum_{i \in A, j \in A} w_{ij}$ is the association term within a cluster and $Assoc(A,V) = assoc(A,A) + cut(A,B)$ is the sum of all the weights associated with nodes in A.



	A	B	sum
A	$assoc(A,A)$	$cut(A,B)$	$assoc(A,V)$
B	$cut(B,A)$	$assoc(B,B)$	$assoc(B,V)$
sum	$assoc(A,V)$	$assoc(B,v)$	

- We want minimize the disassociation between the groups and maximize the association within the groups

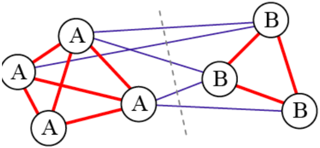
Normalized Cuts

[J. Shi and J. Malik, PAMI00]

- Better measure is the normalized cuts

$$N_{cut}(A,B) = \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)}$$

with $assoc(A,A) = \sum_{i \in A, j \in A} w_{ij}$ is the association term within a cluster and $Assoc(A,V) = assoc(A,A) + cut(A,B)$ is the sum of all the weights associated with nodes in A.



	A	B	sum
A	$assoc(A,A)$	$cut(A,B)$	$assoc(A,V)$
B	$cut(B,A)$	$assoc(B,B)$	$assoc(B,V)$
sum	$assoc(A,V)$	$assoc(B,v)$	

- We want minimize the disassociation between the groups and maximize the association within the groups

Normalized Cuts

- Computing the optimal normalized cut is NP-Complete.
- Instead, relax by computing a real value assignment

Normalized Cuts

- Computing the optimal normalized cut is NP-Complete.
- Instead, relax by computing a real value assignment
- Let $\mathbf{d} = \mathbf{W}1$ be the row sums of the symmetric matrix $\mathbf{W}$, and $\mathbf{D} = diag(\mathbf{d})$ be the corresponding diagonal matrix.

Normalized Cuts

- Computing the optimal normalized cut is NP-Complete.
- Instead, relax by computing a real value assignment
- Let $\mathbf{d} = \mathbf{W}1$ be the row sums of the symmetric matrix $\mathbf{W}$, and $\mathbf{D} = diag(\mathbf{d})$ be the corresponding diagonal matrix.
- Shi and Malik, compute the cut by solving

$$\min_{\mathbf{y}} \frac{\mathbf{y}^T(\mathbf{D} - \mathbf{W})\mathbf{y}}{\mathbf{y}^T\mathbf{D}\mathbf{y}}$$

relaxing $\mathbf{y}$ to be real-value

Normalized Cuts

- Computing the optimal normalized cut is NP-Complete.
- Instead, relax by computing a real value assignment
- Let $\mathbf{d} = \mathbf{W}1$ be the row sums of the symmetric matrix $\mathbf{W}$, and $\mathbf{D} = diag(\mathbf{d})$ be the corresponding diagonal matrix.
- Shi and Malik, compute the cut by solving

$$\min_{\mathbf{y}} \frac{\mathbf{y}^T(\mathbf{D} - \mathbf{W})\mathbf{y}}{\mathbf{y}^T\mathbf{D}\mathbf{y}}$$

  relaxing $\mathbf{y}$ to be real-value
- $\mathbf{D} - \mathbf{W}$ is the Laplacian

Normalized Cuts

- Computing the optimal normalized cut is NP-Complete.
- Instead, relax by computing a real value assignment
- Let $\mathbf{d} = \mathbf{W}1$ be the row sums of the symmetric matrix $\mathbf{W}$, and $\mathbf{D} = diag(\mathbf{d})$ be the corresponding diagonal matrix.
- Shi and Malik, compute the cut by solving

$$\min_{\mathbf{y}} \frac{\mathbf{y}^T(\mathbf{D} - \mathbf{W})\mathbf{y}}{\mathbf{y}^T\mathbf{D}\mathbf{y}}$$

  relaxing $\mathbf{y}$ to be real-value
- $\mathbf{D} - \mathbf{W}$ is the Laplacian

Solving for the cut

- Minimizing this **Rayleigh quotient** is equivalent to solving the generalized eigenvalue system

$$(\mathbf{D} - \mathbf{W})\mathbf{y} = \lambda \mathbf{D}\mathbf{y}$$

- This is a normal eigenvalue problem

$$(\mathbf{I} - \mathbf{N})\mathbf{z} = \lambda \mathbf{z}$$

with $\mathbf{N} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$ is the normalized affinity matrix, and $\mathbf{z} = \mathbf{D}^{1/2}\mathbf{y}$.

Solving for the cut

- Minimizing this **Rayleigh quotient** is equivalent to solving the generalized eigenvalue system
$$(\mathbf{D} - \mathbf{W})\mathbf{y} = \lambda \mathbf{D}\mathbf{y}$$
- This is a normal eigenvalue problem
$$(\mathbf{I} - \mathbf{N})\mathbf{z} = \lambda \mathbf{z}$$
with $\mathbf{N} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$ is the normalized affinity matrix, and $\mathbf{z} = \mathbf{D}^{1/2}\mathbf{y}$.
- This is an example of a spectral method for segmentation, solution is the second smallest eigenvector/eigenvalue

Solving for the cut

- Minimizing this **Rayleigh quotient** is equivalent to solving the generalized eigenvalue system
$$(\mathbf{D} - \mathbf{W})\mathbf{y} = \lambda \mathbf{D}\mathbf{y}$$
- This is a normal eigenvalue problem
$$(\mathbf{I} - \mathbf{N})\mathbf{z} = \lambda \mathbf{z}$$
with $\mathbf{N} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$ is the normalized affinity matrix, and $\mathbf{z} = \mathbf{D}^{1/2}\mathbf{y}$.
- This is an example of a spectral method for segmentation, solution is the second smallest eigenvector/eigenvalue
- This process can be applied in a hierarchical manner to have more clusters

Solving for the cut

- Minimizing this **Rayleigh quotient** is equivalent to solving the generalized eigenvalue system
$$(\mathbf{D} - \mathbf{W})\mathbf{y} = \lambda \mathbf{D}\mathbf{y}$$
- This is a normal eigenvalue problem
$$(\mathbf{I} - \mathbf{N})\mathbf{z} = \lambda \mathbf{z}$$
with $\mathbf{N} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$ is the normalized affinity matrix, and $\mathbf{z} = \mathbf{D}^{1/2}\mathbf{y}$.
- This is an example of a spectral method for segmentation, solution is the second smallest eigenvector/eigenvalue
- This process can be applied in a hierarchical manner to have more clusters
- Shi and Malik employ the following affinity
$$w_{i,j} = \exp\left(-\frac{||\mathbf{F}_i - \mathbf{F}_j||_2^2}{\sigma_f^2} - \frac{||p_i - p_j||_2^2}{\sigma_s^2}\right)$$
for pixels within a radius $||p_i - p_j||_2 < r$, and $\mathbf{F}$ is a feature vector with color, intensities, histograms, gradients, etc.

Solving for the cut

- Minimizing this **Rayleigh quotient** is equivalent to solving the generalized eigenvalue system
$$(\mathbf{D}-\mathbf{W})\mathbf{y} = \lambda \mathbf{D}\mathbf{y}$$
- This is a normal eigenvalue problem
$$(\mathbf{I}-\mathbf{N})\mathbf{z} = \lambda \mathbf{z}$$
with $\mathbf{N} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$ is the normalized affinity matrix, and $\mathbf{z} = \mathbf{D}^{1/2}\mathbf{y}$.
- This is an example of a spectral method for segmentation, solution is the second smallest eigenvector/eigenvalue
- This process can be applied in a hierarchical manner to have more clusters
- Shi and Malik employ the following affinity
$$w_{i,j} = \exp\left(-\frac{||\mathbf{F}_i - \mathbf{F}_j||_2^2}{\sigma_f^2} - \frac{||p_i - p_j||_2^2}{\sigma_s^2}\right)$$
for pixels within a radius $||p_i - p_j||_2 < r$, and $\mathbf{F}$ is a feature vector with color, intensities, histograms, gradients, etc.

Algorithm

[J. Shi and J. Malik, PAMI00]

1. Given an image or image sequence, set up a weighted graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ and set the weight on the edge connecting two nodes to be a measure of the similarity between the two nodes.
2. Solve $(\mathbf{D} - \mathbf{W})\boldsymbol{x} = \lambda \mathbf{D}\boldsymbol{x}$ for eigenvectors with the smallest eigenvalues.
3. Use the eigenvector with the second smallest eigenvalue to bipartition the graph.
4. Decide if the current partition should be subdivided and recursively repartition the segmented parts if necessary.

Examples

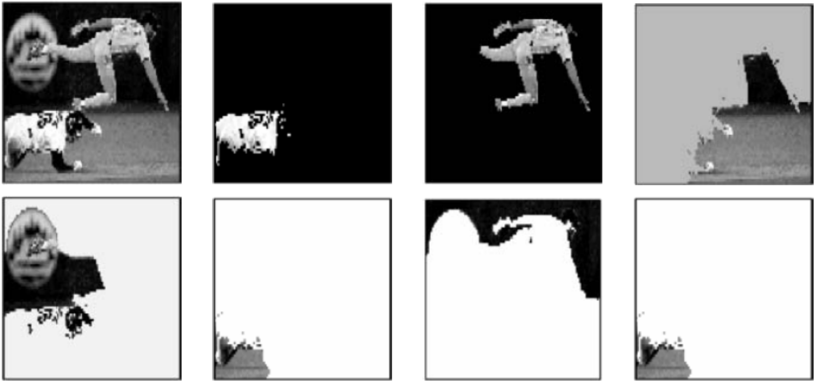


Figure: Shi and Malik N-Cuts

Example of grouping techniques

- K-means style clustering, e.g., SLIC superpixels
- Normalized cuts
- Graph-based superpixels
- Mean-shift
- Watershed transform

Graph-based Superpixels

- Construct a graph that has as many nodes as pixels
- Define edges between neighboring pixels, with whatever definition of neighboring

Graph-based Superpixels

- Construct a graph that has as many nodes as pixels
- Define edges between neighboring pixels, with whatever definition of neighboring
- Define the weight between neighbors to be the dissimilarity between them (a non-negative measure)

Graph-based Superpixels

- Construct a graph that has as many nodes as pixels
- Define edges between neighboring pixels, with whatever definition of neighboring
- Define the weight between neighbors to be the dissimilarity between them (a non-negative measure)
- Let $G(V, E)$ be the graph, we want to segment V into components $(C_1, \cdots, C_r)$

Graph-based Superpixels

- Construct a graph that has as many nodes as pixels
- Define edges between neighboring pixels, with whatever definition of neighboring
- Define the weight between neighbors to be the dissimilarity between them (a non-negative measure)
- Let $G(V, E)$ be the graph, we want to segment V into components $(C_1, \cdots, C_r)$
- Felzenswald and Hutterlocker defined a simple greedy algorithm which can be shown to have some interesting global properties

Graph-based Superpixels

- Construct a graph that has as many nodes as pixels
- Define edges between neighboring pixels, with whatever definition of neighboring
- Define the weight between neighbors to be the dissimilarity between them (a non-negative measure)
- Let $G(V, E)$ be the graph, we want to segment V into components $(C_1, \cdots, C_r)$
- Felzenswald and Hutterlocker defined a simple greedy algorithm which can be shown to have some interesting global properties

Algorithm

1. Sort E into $\pi = (o_1, \cdots, o_m)$ by non-decreasing weights
2. Start with segmentation S^0, where each vertex is its own component (i.e., as many superpixels as pixels)

Algorithm

1. Sort E into $\pi = (o_1, \cdots, o_m)$ by non-decreasing weights
2. Start with segmentation S^0, where each vertex is its own component (i.e., as many superpixels as pixels)
3. Repeat step 4 for $q = 1, \cdots, q = m$

Algorithm

1. Sort E into $\pi = (o_1, \cdots, o_m)$ by non-decreasing weights
2. Start with segmentation S^0, where each vertex is its own component (i.e., as many superpixels as pixels)
3. Repeat step 4 for $q = 1, \cdots, q = m$
4. Construct S^q given S^{q-1} as follows. Let $o_q = (v_i, v_j)$. If v_i and v_j are disjoint components of S^{q-1} and $w(o_q)$ is small compared to the internal difference of both components of S^{q-1}, then merge the two components. Otherwise do nothing $S^q = S^{q-1}$

Algorithm

1. Sort E into $\pi = (o_1, \cdots, o_m)$ by non-decreasing weights
2. Start with segmentation S^0, where each vertex is its own component (i.e., as many superpixels as pixels)
3. Repeat step 4 for $q = 1, \cdots, q = m$
4. Construct S^q given S^{q-1} as follows. Let $o_q = (v_i, v_j)$. If v_i and v_j are disjoint components of S^{q-1} and $w(o_q)$ is small compared to the internal difference of both components of S^{q-1}, then merge the two components. Otherwise do nothing $S^q = S^{q-1}$

The internal difference is defined as the largest weight in the minimum spanning tree of the component.

$$Int(C) = \max_{e \in MST(C,E)} w(e)$$

Algorithm

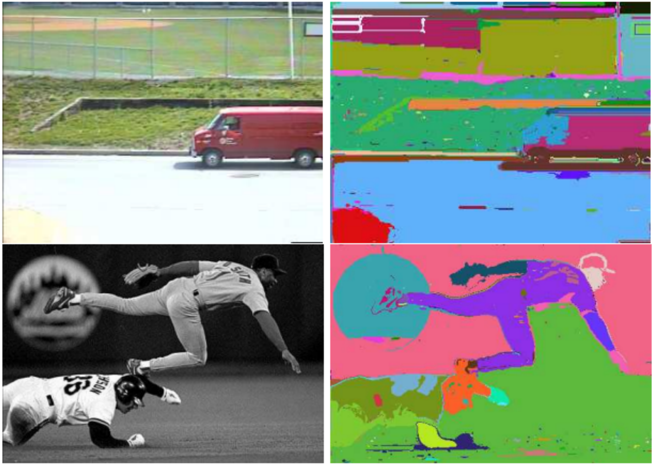
1. Sort E into $\pi = (o_1, \cdots, o_m)$ by non-decreasing weights
2. Start with segmentation S^0, where each vertex is its own component (i.e., as many superpixels as pixels)
3. Repeat step 4 for $q = 1, \cdots, q = m$
4. Construct S^q given S^{q-1} as follows. Let $o_q = (v_i, v_j)$. If v_i and v_j are disjoint components of S^{q-1} and $w(o_q)$ is small compared to the internal difference of both components of S^{q-1}, then merge the two components. Otherwise do nothing $S^q = S^{q-1}$

The internal difference is defined as the largest weight in the minimum spanning tree of the component.

$$Int(C) = \max_{e \in MST(C,E)} w(e)$$

Results

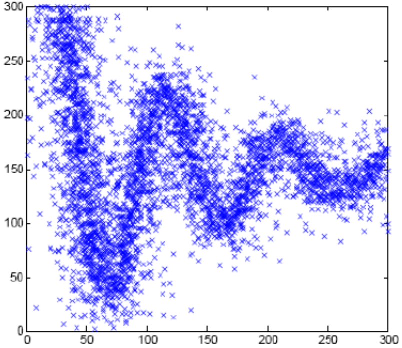
[P. Felzenszwald and D. Huttenlocher, IJCV04]



Example of grouping techniques

- K-means style clustering, e.g., SLIC superpixels
- Normalized cuts
- Graph-based superpixels
- Mean-shift
- Watershed transform

- We have a bunch of points drawn from some distribution
- What's the distribution that generated these points?



[Source: M. Tappen]

Parametric vs Non-Parametric

- We can fit a parametric distribution, e.g., mixture of Gaussians
- KDE idea: Use the data to define the distribution

Parametric vs Non-Parametric

- We can fit a parametric distribution, e.g., mixture of Gaussians
- KDE idea: Use the data to define the distribution
  - If I were to draw more samples from the same probability distribution, then those points would probably be close to the points that I have already drawn

Parametric vs Non-Parametric

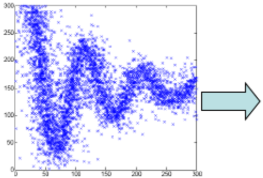
- We can fit a parametric distribution, e.g., mixture of Gaussians
- KDE idea: Use the data to define the distribution
  - If I were to draw more samples from the same probability distribution, then those points would probably be close to the points that I have already drawn
  - Build distribution by putting a little mass of probability around each data-point

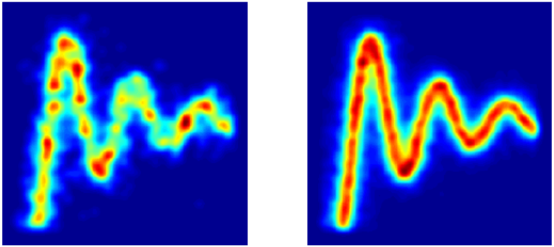
[Source: M. Tappen]

Parametric vs Non-Parametric

- We can fit a parametric distribution, e.g., mixture of Gaussians
- KDE idea: Use the data to define the distribution
  - If I were to draw more samples from the same probability distribution, then those points would probably be close to the points that I have already drawn
  - Build distribution by putting a little mass of probability around each data-point

[Source: M. Tappen]





(a) 2000 Samples (b) 20000 Samples

Figure 2-2: Kernel density estimates of the density function shown in Figure 2-1(a). Figure (a) shows the estimate found with a relatively small number of samples. It is uneven and does not approximate the true density well. (b) With more samples, the estimate of the density improves significantly.

[Source: M. Tappen]

KDE

- We approximate the density by

$$\hat{f}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} K_H(\mathbf{x} - \mathbf{x}_i)$$

with $\mathbf{x}_i$ the points, and $K_H(\mathbf{x} - \mathbf{x}_i)$ the kernel

- Gaussian kernel is typically used

KDE

- We approximate the density by

$$\hat{f}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} K_H(\mathbf{x} - \mathbf{x}_i)$$

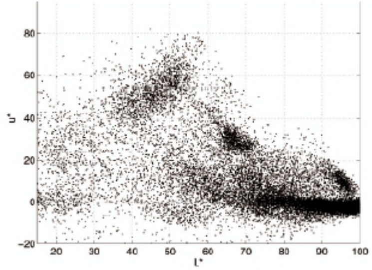
  with $\mathbf{x}_i$ the points, and $K_H(\mathbf{x} - \mathbf{x}_i)$ the kernel
- Gaussian kernel is typically used
- Alternative way to think about this, put 1 wherever you have a sample and convolve with a Gaussian

KDE

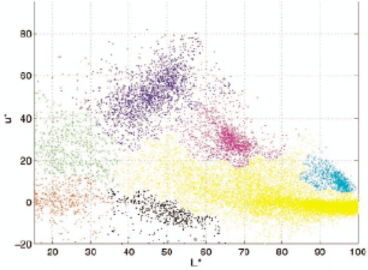
- We approximate the density by

$$\hat{f}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} K_H(\mathbf{x} - \mathbf{x}_i)$$

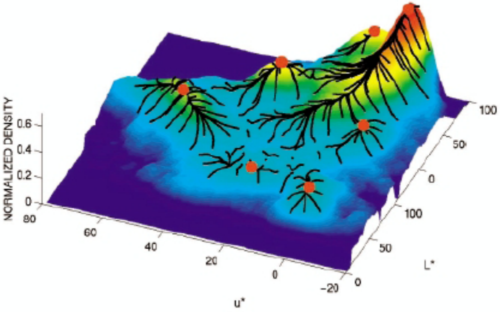
  with $\mathbf{x}_i$ the points, and $K_H(\mathbf{x} - \mathbf{x}_i)$ the kernel
- Gaussian kernel is typically used
- Alternative way to think about this, put 1 wherever you have a sample and convolve with a Gaussian



(a)



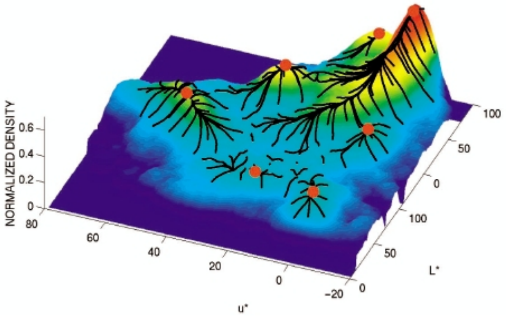
(b)



(c)

What is mean-shift

- The density will have peaks (also called modes)
- If we started at point and did gradient-ascent, we would end up at one of the modes
- Cluster based on which mode each point belongs to



[Source: M. Tappen]

No need for gradient ascent

- A set of iterative steps can be taken that will monotonically converge to a mode
- No worries about step sizes

No need for gradient ascent

- A set of iterative steps can be taken that will monotonically converge to a mode
- No worries about step sizes
- This is an adaptive gradient ascent, for each iteration

$$\mathbf{y}_{j+1} = \frac{\sum_{i=1}^{n} \mathbf{x}_i g(||\frac{\mathbf{y}_j - \mathbf{x}_i}{h}||_2^2)}{\sum_{i=1}^{n} g(||\frac{\mathbf{y}_j - \mathbf{x}_i}{h}||_2^2)}$$

with $g = \frac{d}{du}k(u)$, and $k(\mathbf{x}) = C\sum_{i=1}^{n} k(||\frac{\mathbf{y}_j - \mathbf{x}_i}{h}||_2^2)$

No need for gradient ascent

- A set of iterative steps can be taken that will monotonically converge to a mode
- No worries about step sizes
- This is an adaptive gradient ascent, for each iteration

$$\mathbf{y}_{j+1} = \frac{\sum_{i=1}^{n} \mathbf{x}_i g(||\frac{\mathbf{y}_j - \mathbf{x}_i}{h}||_2^2)}{\sum_{i=1}^{n} g(||\frac{\mathbf{y}_j - \mathbf{x}_i}{h}||_2^2)}$$

with $g = \frac{d}{du}k(u)$, and $k(\mathbf{x}) = C\sum_{i=1}^{n} k(||\frac{\mathbf{y}_j - \mathbf{x}_i}{h}||_2^2)$

- Why is this the update?

No need for gradient ascent

- A set of iterative steps can be taken that will monotonically converge to a mode
- No worries about step sizes
- This is an adaptive gradient ascent, for each iteration

$$\mathbf{y}_{j+1} = \frac{\sum_{i=1}^{n} \mathbf{x}_i g(||\frac{\mathbf{y}_j - \mathbf{x}_i}{h}||_2^2)}{\sum_{i=1}^{n} g(||\frac{\mathbf{y}_j - \mathbf{x}_i}{h}||_2^2)}$$

  with $g = \frac{d}{du}k(u)$, and $k(\mathbf{x}) = C\sum_{i=1}^{n} k(||\frac{\mathbf{y}_j - \mathbf{x}_i}{h}||_2^2)$
- Why is this the update?
- This procedure gives you one mode, how to get all?

No need for gradient ascent

- A set of iterative steps can be taken that will monotonically converge to a mode
- No worries about step sizes
- This is an adaptive gradient ascent, for each iteration

$$\mathbf{y}_{j+1} = \frac{\sum_{i=1}^{n} \mathbf{x}_i g(||\frac{\mathbf{y}_j - \mathbf{x}_i}{h}||_2^2)}{\sum_{i=1}^{n} g(||\frac{\mathbf{y}_j - \mathbf{x}_i}{h}||_2^2)}$$

  with $g = \frac{d}{du}k(u)$, and $k(\mathbf{x}) = C\sum_{i=1}^{n} k(||\frac{\mathbf{y}_j - \mathbf{x}_i}{h}||_2^2)$
- Why is this the update?
- This procedure gives you one mode, how to get all?
- Start from each point, and record the clusters

No need for gradient ascent

- A set of iterative steps can be taken that will monotonically converge to a mode
- No worries about step sizes
- This is an adaptive gradient ascent, for each iteration

$$\mathbf{y}_{j+1} = \frac{\sum_{i=1}^{n} \mathbf{x}_i g(||\frac{\mathbf{y}_j - \mathbf{x}_i}{h}||_2^2)}{\sum_{i=1}^{n} g(||\frac{\mathbf{y}_j - \mathbf{x}_i}{h}||_2^2)}$$

  with $g = \frac{d}{du} k(u)$, and $k(\mathbf{x}) = C \sum_{i=1}^{n} k(||\frac{\mathbf{y}_j - \mathbf{x}_i}{h}||_2^2)$

- Why is this the update?
- This procedure gives you one mode, how to get all?
- Start from each point, and record the clusters
- For segmentation use whatever feature representation you want for $\mathbf{x}_i$

[Source: M. Tappen]

No need for gradient ascent

- A set of iterative steps can be taken that will monotonically converge to a mode
- No worries about step sizes
- This is an adaptive gradient ascent, for each iteration

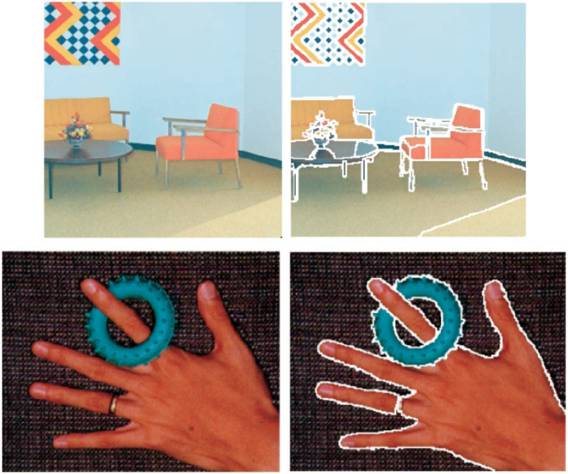
$$\mathbf{y}_{j+1} = \frac{\sum_{i=1}^{n} \mathbf{x}_i g(||\frac{\mathbf{y}_j - \mathbf{x}_i}{h}||_2^2)}{\sum_{i=1}^{n} g(||\frac{\mathbf{y}_j - \mathbf{x}_i}{h}||_2^2)}$$

  with $g = \frac{d}{du}k(u)$, and $k(\mathbf{x}) = C\sum_{i=1}^{n} k(||\frac{\mathbf{y}_j - \mathbf{x}_i}{h}||_2^2)$

- Why is this the update?
- This procedure gives you one mode, how to get all?
- Start from each point, and record the clusters
- For segmentation use whatever feature representation you want for $\mathbf{x}_i$

[Source: M. Tappen]

Results



[Source: M. Tappen]

Let's look at Structure from Motion

Structure from motion

- We saw in class how 2D and 3D point sets could be aligned
- We also saw how this alignment could be used to estimate camera pose and internal parameters

Structure from motion

- We saw in class how 2D and 3D point sets could be aligned
- We also saw how this alignment could be used to estimate camera pose and internal parameters
- We now look at the converse problem

Structure from motion

- We saw in class how 2D and 3D point sets could be aligned
- We also saw how this alignment could be used to estimate camera pose and internal parameters
- We now look at the converse problem
- Estimating the location of 3D points from multiple images given a sparse set of correspondences between image features.

Structure from motion

- We saw in class how 2D and 3D point sets could be aligned
- We also saw how this alignment could be used to estimate camera pose and internal parameters
- We now look at the converse problem
- Estimating the location of 3D points from multiple images given a sparse set of correspondences between image features.
- This process typically involves simultaneous estimation of 3D geometry (structure) and camera pose (motion)

Structure from motion

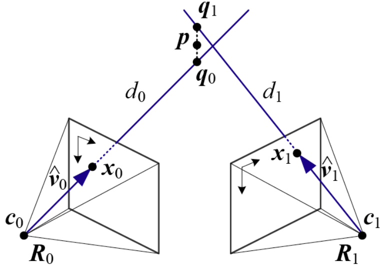
- We saw in class how 2D and 3D point sets could be aligned
- We also saw how this alignment could be used to estimate camera pose and internal parameters
- We now look at the converse problem
- Estimating the location of 3D points from multiple images given a sparse set of correspondences between image features.
- This process typically involves simultaneous estimation of 3D geometry (structure) and camera pose (motion)
- This problem is called **structure from motion**

Structure from motion

- We saw in class how 2D and 3D point sets could be aligned
- We also saw how this alignment could be used to estimate camera pose and internal parameters
- We now look at the converse problem
- Estimating the location of 3D points from multiple images given a sparse set of correspondences between image features.
- This process typically involves simultaneous estimation of 3D geometry (structure) and camera pose (motion)
- This problem is called **structure from motion**

Triangulation

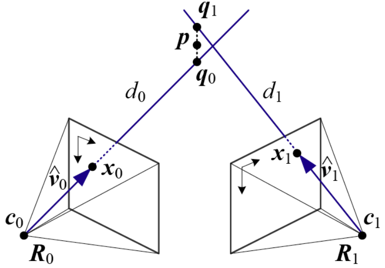
- It is the problem of determining a point's 3D position from a set of corresponding image locations and known camera positions
- This problem is the converse of the pose estimation problem



- Simplest solution: find 3D point $\mathbf{p}$ that lies closest to all the 3D rays corresponding to the 2D feature locations $\{\mathbf{x}_j\}$ observed by cameras $\mathbf{P}_j = \mathbf{K}_j[\mathbf{R}_j|\mathbf{t}_j]$, with $\mathbf{t}_j = -\mathbf{R}_j\mathbf{c}_j$.

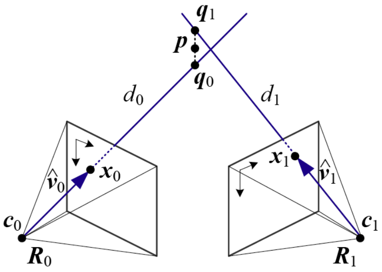
Triangulation

- It is the problem of determining a point's 3D position from a set of corresponding image locations and known camera positions
- This problem is the converse of the pose estimation problem



- Simplest solution: find 3D point $\mathbf{p}$ that lies closest to all the 3D rays corresponding to the 2D feature locations $\{\mathbf{x}_j\}$ observed by cameras $\mathbf{P}_j = \mathbf{K}_j[\mathbf{R}_j|\mathbf{t}_j]$, with $\mathbf{t}_j = -\mathbf{R}_j\mathbf{c}_j$.

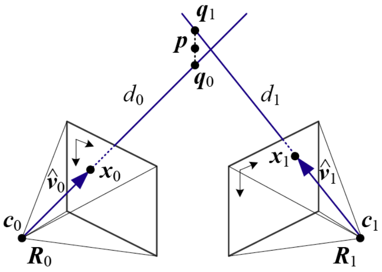
Triangulation



- The rays originate at $\mathbf{c}_j$ in a direction $\hat{\mathbf{v}}_j = \mathcal{N}(\mathbf{R}_j^{-1} K_j^{-1} \mathbf{x}_j)$
- The nearest point to $\mathbf{p}$ is the point $\mathbf{q}_j = \mathbf{c}_j + d_j \hat{\mathbf{v}}_j$ such that

$$\min_{d_j} ||\mathbf{c}_j + d_j \hat{\mathbf{v}}_j - \mathbf{p}||_2^2$$

Triangulation



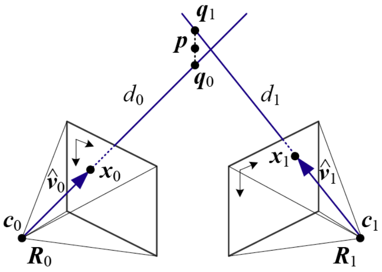
- The rays originate at $\mathbf{c}_j$ in a direction $\hat{\mathbf{v}}_j = \mathcal{N}(\mathbf{R}_j^{-1} K_j^{-1} \mathbf{x}_j)$
- The nearest point to $\mathbf{p}$ is the point $\mathbf{q}_j = \mathbf{c}_j + d_j \hat{\mathbf{v}}_j$ such that

$$\min_{d_j} ||\mathbf{c}_j + d_j \hat{\mathbf{v}}_j - \mathbf{p}||_2^2$$

- This has a minimimun at $d_j = \hat{\mathbf{v}}_j \cdot (\mathbf{p} - \mathbf{c}_j)$, thus

$$\mathbf{q}_j = \mathbf{c}_j + (\hat{\mathbf{v}}_j \hat{\mathbf{v}}_j^T)(\mathbf{p} - \mathbf{c}_j) = \mathbf{c}_j + (\mathbf{p} - \mathbf{c}_j)_{||}$$

Triangulation



- The rays originate at $\mathbf{c}_j$ in a direction $\hat{\mathbf{v}}_j = \mathcal{N}(\mathbf{R}_j^{-1} K_j^{-1} \mathbf{x}_j)$
- The nearest point to $\mathbf{p}$ is the point $\mathbf{q}_j = \mathbf{c}_j + d_j \hat{\mathbf{v}}_j$ such that

$$\min_{d_j} ||\mathbf{c}_j + d_j \hat{\mathbf{v}}_j - \mathbf{p}||_2^2$$

- This has a minimimun at $d_j = \hat{\mathbf{v}}_j \cdot (\mathbf{p} - \mathbf{c}_j)$, thus

$$\mathbf{q}_j = \mathbf{c}_j + (\hat{\mathbf{v}}_j \hat{\mathbf{v}}_j^T)(\mathbf{p} - \mathbf{c}_j) = \mathbf{c}_j + (\mathbf{p} - \mathbf{c}_j)_{||}$$

Triangulation

- The nearest point to $\mathbf{p}$ is the point $\mathbf{q}_j$ such that

$$\min_{d_j} ||\mathbf{c}_j + d_j \hat{\mathbf{v}}_j - \mathbf{p}||_2^2$$

- This has a minimimun at $d_j = \hat{\mathbf{v}}_j \cdot (\mathbf{p} - \mathbf{c}_j)$, thus

$$\mathbf{q}_j = \mathbf{c}_j + (\hat{\mathbf{v}}_j \hat{\mathbf{v}}_j^T)(\mathbf{p} - \mathbf{c}_j) = \mathbf{c}_j + (\mathbf{p} - \mathbf{c}_j)_{||}$$

- The square distance is then

$$r_j^2 = ||(\mathbf{I} - \hat{\mathbf{v}}_j \hat{\mathbf{v}}_j^T)(\mathbf{p} - \mathbf{c}_j)||_2^2 = ||(\mathbf{p} - \mathbf{c}_j)_\perp||_2^2$$

Triangulation

- The nearest point to $\mathbf{p}$ is the point $\mathbf{q}_j$ such that

$$\min_{d_j} ||\mathbf{c}_j + d_j \hat{\mathbf{v}}_j - \mathbf{p}||_2^2$$

- This has a minimimun at $d_j = \hat{\mathbf{v}}_j \cdot (\mathbf{p} - \mathbf{c}_j)$, thus

$$\mathbf{q}_j = \mathbf{c}_j + (\hat{\mathbf{v}}_j \hat{\mathbf{v}}_j^T)(\mathbf{p} - \mathbf{c}_j) = \mathbf{c}_j + (\mathbf{p} - \mathbf{c}_j)_{||}$$

- The square distance is then

$$r_j^2 = ||(\mathbf{I} - \hat{\mathbf{v}}_j \hat{\mathbf{v}}_j^T)(\mathbf{p} - \mathbf{c}_j)||_2^2 = ||(\mathbf{p} - \mathbf{c}_j)_\perp||_2^2$$

- The optimal value of $\mathbf{p}$ can be computed by least-squares

$$\mathbf{p} = \frac{\sum_j (\mathbf{I} - \hat{\mathbf{v}}_j \hat{\mathbf{v}}_j^T)\mathbf{c}_j}{\sum_j (\mathbf{I} - \hat{\mathbf{v}}_j \hat{\mathbf{v}}_j^T)}$$

Triangulation

- The nearest point to $\mathbf{p}$ is the point $\mathbf{q}_j$ such that

$$\min_{d_j} ||\mathbf{c}_j + d_j \hat{\mathbf{v}}_j - \mathbf{p}||_2^2$$

- This has a minimimun at $d_j = \hat{\mathbf{v}}_j \cdot (\mathbf{p} - \mathbf{c}_j)$, thus

$$\mathbf{q}_j = \mathbf{c}_j + (\hat{\mathbf{v}}_j \hat{\mathbf{v}}_j^T)(\mathbf{p} - \mathbf{c}_j) = \mathbf{c}_j + (\mathbf{p} - \mathbf{c}_j)_{||}$$

- The square distance is then

$$r_j^2 = ||(\mathbf{I} - \hat{\mathbf{v}}_j \hat{\mathbf{v}}_j^T)(\mathbf{p} - \mathbf{c}_j)||_2^2 = ||(\mathbf{p} - \mathbf{c}_j)_\perp||_2^2$$

- The optimal value of $\mathbf{p}$ can be computed by least-squares

$$\mathbf{p} = \frac{\sum_j (\mathbf{I} - \hat{\mathbf{v}}_j \hat{\mathbf{v}}_j^T)\mathbf{c}_j}{\sum_j (\mathbf{I} - \hat{\mathbf{v}}_j \hat{\mathbf{v}}_j^T)}$$

Alternative formulation

- Can produce significantly better estimates if some cameras are closer to the 3D point than others is to minimize the residual in the measurement equations

$$
\begin{aligned}
x_j &= \frac{p^j_{00}X + p^j_{01}Y + p^j_{02}Z + p^j_{03}W}{p^j_{20}X + p^j_{21}Y + p^j_{22}Z + p^j_{23}W} \\
y_j &= \frac{p^j_{10}X + p^j_{11}Y + p^j_{12}Z + p^j_{13}W}{p^j_{20}X + p^j_{21}Y + p^j_{22}Z + p^j_{23}W}
\end{aligned}
$$

  where (x_j, y_j) are the measured 2D feature locations, and $\{p^j_{00}, \cdots, p^j_{23}\}$ are the known entries in the camera matrix $\mathbf{P}$, and $\mathbf{p} = (X, Y, Z, W)$ in homogeneous coordinates

- Why is this better?

- How do we solve this now?

Two frame structure from motion

Figure: Images from N. Snavely

- Simultaneous recovery of 3D structure and pose from image correspondences
- Why is this not the stereo problem?

Two frame structure from motion

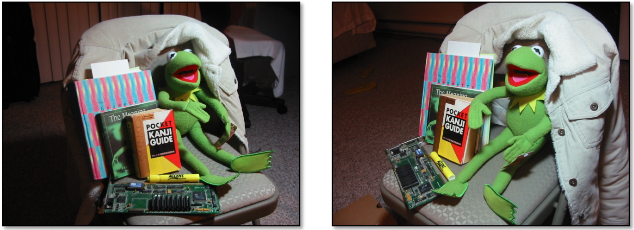


Figure: Images from N. Snavely

- Simultaneous recovery of 3D structure and pose from image correspondences
- Why is this not the stereo problem?
- What can we do?

Two frame structure from motion

Figure: Images from N. Snavely

- Simultaneous recovery of 3D structure and pose from image correspondences
- Why is this not the stereo problem?
- What can we do?
- We can estimate the fundamental matrix given enough correspondences!

$$\mathbf{x}'^T \mathbf{F} \mathbf{x} = 0$$

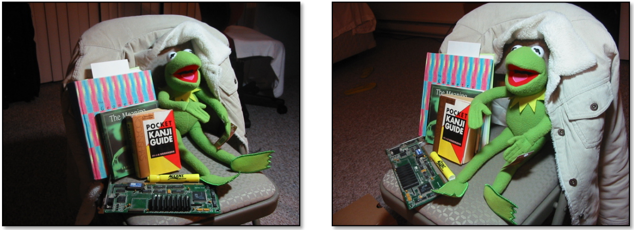


Figure: Images from N. Snavely

- Simultaneous recovery of 3D structure and pose from image correspondences
- Why is this not the stereo problem?
- What can we do?
- We can estimate the fundamental matrix given enough correspondences!

$$\mathbf{x}'^T \mathbf{F} \mathbf{x} = 0$$

8 Point Algorithm

- We can write a system of equations

$$\begin{bmatrix} u_1 u_1' & v_1 u_1' & u_1' & u_1 v_1' & v_1 v_1' & v_1' & u_1 & v_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ u_n u_n' & v_n u_n' & u_n' & u_n v_n' & v_n v_n' & v_n' & u_n & v_n & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ \vdots \\ f_{33} \end{bmatrix} = 0$$

- How to solve this?

8 Point Algorithm

- We can write a system of equations

$$\begin{bmatrix} u_1 u_1' & v_1 u_1' & u_1' & u_1 v_1' & v_1 v_1' & v_1' & u_1 & v_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ u_n u_n' & v_n u_n' & u_n' & u_n v_n' & v_n v_n' & v_n' & u_n & v_n & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ \vdots \\ f_{33} \end{bmatrix} = 0$$

- How to solve this?
- Shouldn't **F** have rank 2?

$$\min_{\mathbf{F}'} ||\mathbf{F} - \mathbf{F}'||_2^2$$

8 Point Algorithm

- We can write a system of equations

$$\begin{bmatrix} u_1 u_1' & v_1 u_1' & u_1' & u_1 v_1' & v_1 v_1' & v_1' & u_1 & v_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ u_n u_n' & v_n u_n' & u_n' & u_n v_n' & v_n v_n' & v_n' & u_n & v_n & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ \vdots \\ f_{33} \end{bmatrix} = 0$$

- How to solve this?
- Shouldn't **F** have rank 2?

$$\min_{\mathbf{F}'} ||\mathbf{F} - \mathbf{F}'||_2^2$$

- Solve by SVD (take 2 biggest singular values / singular vectors)

8 Point Algorithm

- We can write a system of equations

$$\begin{bmatrix} u_1 u'_1 & v_1 u'_1 & u'_1 & u_1 v'_1 & v_1 v'_1 & v'_1 & u_1 & v_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ u_n u'_n & v_n u'_n & u'_n & u_n v'_n & v_n v'_n & v'_n & u_n & v_n & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ \vdots \\ f_{33} \end{bmatrix} = 0$$

- How to solve this?
- Shouldn't **F** have rank 2?

$$\min_{\mathbf{F}'} ||\mathbf{F} - \mathbf{F}'||_2^2$$

- Solve by SVD (take 2 biggest singular values / singular vectors)
- What happens in the presence of noise?

8 Point Algorithm

- We can write a system of equations

$$\begin{bmatrix} u_1 u_1' & v_1 u_1' & u_1' & u_1 v_1' & v_1 v_1' & v_1' & u_1 & v_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ u_n u_n' & v_n u_n' & u_n' & u_n v_n' & v_n v_n' & v_n' & u_n & v_n & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ \vdots \\ f_{33} \end{bmatrix} = 0$$

- How to solve this?
- Shouldn't **F** have rank 2?

$$\min_{\mathbf{F}'} ||\mathbf{F} - \mathbf{F}'||_2^2$$

- Solve by SVD (take 2 biggest singular values / singular vectors)
- What happens in the presence of noise?
- Normalize the points to have mean 0 and unit variance (Hartley 99)

8 Point Algorithm

- We can write a system of equations

$$\begin{bmatrix} u_1 u_1' & v_1 u_1' & u_1' & u_1 v_1' & v_1 v_1' & v_1' & u_1 & v_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ u_n u_n' & v_n u_n' & u_n' & u_n v_n' & v_n v_n' & v_n' & u_n & v_n & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ \vdots \\ f_{33} \end{bmatrix} = 0$$

- How to solve this?
- Shouldn't **F** have rank 2?

$$\min_{\mathbf{F}'} ||\mathbf{F} - \mathbf{F}'||_2^2$$

- Solve by SVD (take 2 biggest singular values / singular vectors)
- What happens in the presence of noise?
- Normalize the points to have mean 0 and unit variance (Hartley 99)
- This works better in practice

8 Point Algorithm

- We can write a system of equations

$$\begin{bmatrix} u_1 u_1' & v_1 u_1' & u_1' & u_1 v_1' & v_1 v_1' & v_1' & u_1 & v_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ u_n u_n' & v_n u_n' & u_n' & u_n v_n' & v_n v_n' & v_n' & u_n & v_n & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ \vdots \\ f_{33} \end{bmatrix} = 0$$

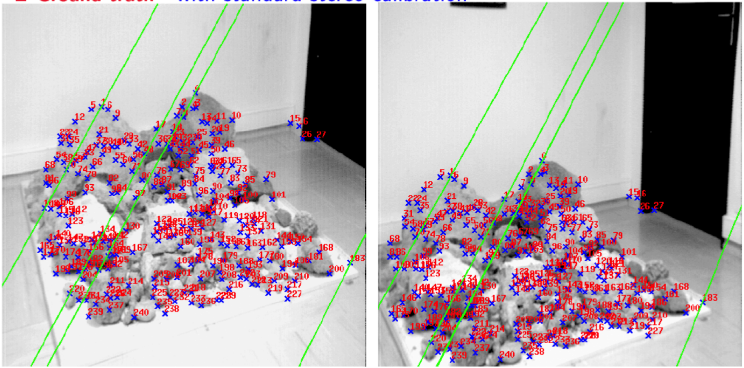
- How to solve this?
- Shouldn't **F** have rank 2?

$$\min_{\mathbf{F}'} ||\mathbf{F} - \mathbf{F}'||_2^2$$

- Solve by SVD (take 2 biggest singular values / singular vectors)
- What happens in the presence of noise?
- Normalize the points to have mean 0 and unit variance (Hartley 99)
- This works better in practice

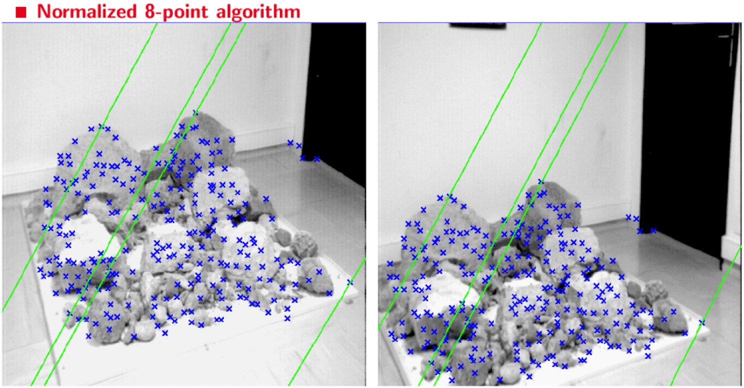
Results

■ **Ground truth** with standard stereo calibration



[Source: N. Snavely]

Results



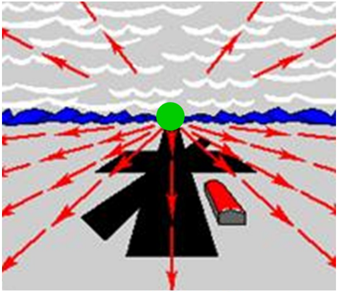
[Source: N. Snavely]

Now what?

- Given the fundamental matrix, we can calibrate the cameras
- Given this calibration we can triangulate
- This is a chicken and egg problem so we can re-iterate this process.

Pure Translational Motion (known rotation)

- If we know the rotation, we can pre-rotate all the points in the second image to match the viewing direction of the first.
- The resulting set of 3D points move towards (or away from) the **focus of expansion (FOE)**
- See exercise about this ...

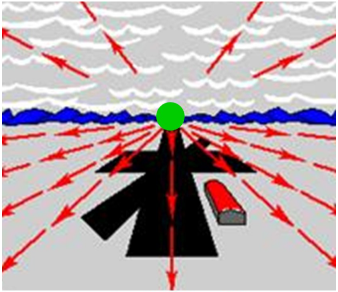


adapted from Gibson 1978

- What if its a purely rotational motion?

Pure Translational Motion (known rotation)

- If we know the rotation, we can pre-rotate all the points in the second image to match the viewing direction of the first.
- The resulting set of 3D points move towards (or away from) the **focus of expansion (FOE)**
- See exercise about this ...

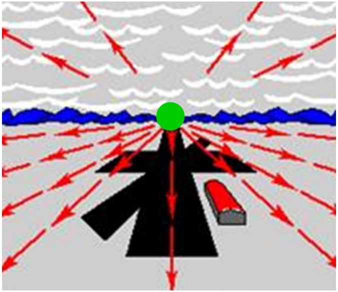


adapted from Gibson 1978

- What if its a purely rotational motion?
- What happens in the general case if we know **K**?

Pure Translational Motion (known rotation)

- If we know the rotation, we can pre-rotate all the points in the second image to match the viewing direction of the first.
- The resulting set of 3D points move towards (or away from) the **focus of expansion (FOE)**
- See exercise about this ...



adapted from Gibson 1978

- What if its a purely rotational motion?
- What happens in the general case if we know **K**?

More views...

- The geometry of three views is described by a 3 x 3 x 3 tensor called the **trifocal tensor**
- The geometry of four views is described by a 3 x 3 x 3 x 3 tensor called the **quadrifocal tensor**
- After this it starts to get complicated ...
- We will not see this in class

Structure from motion

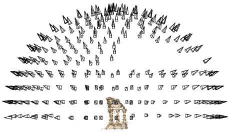
Given many images, how can we

- figure out where they were all taken from?
- build a 3D model of the scene?

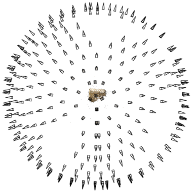
[Source: N. Snavely]

Structure from Motion

- Input: images with points in correspondence $p_{i,j} = (u_{i,j}, v_{i,j})$
- Output:
  - **structure**: 3D location $\mathbf{x}_i$ for each point $\mathbf{p}_i$
  - **motion**: camera parameters $\mathbf{R}_j$, $\mathbf{t}_j$ possibly $\mathbf{K}_j$
- Objective function: minimize reprojection error



Reconstruction (side)



(top)

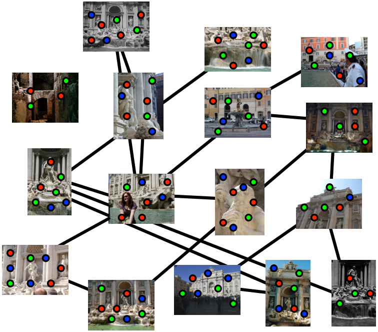
[Source: N. Snavely]

Reconstructions from Video

[Source: N. Snavely]

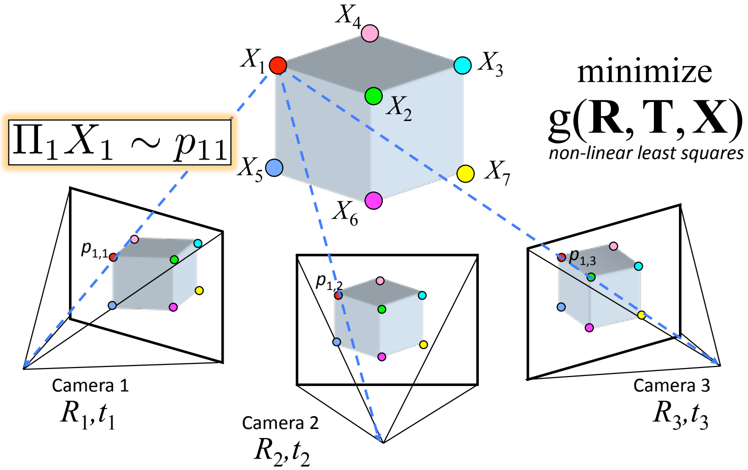
How do we get correspondences?

- Feature detection and matching
- We can construct a graph of matches
- Use RANSAC to estimate fundamental matrices between each pair



[Source: N. Snavely]

Structure from Motion Problem



[Source: N. Snavely]

Problem Size

- What are the variables?
- How many variables per camera?
- How many variables per point?
- E.g., Trevi Fountain collection, 466 input photos, $> 100,000$ 3D points

[Source: N. Snavely]

Bundle Adjustment

- Minimize sum of squared reprojection errors:

$$g(\mathbf{X}, \mathbf{R}, \mathbf{T}) = \sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} || \mathbf{P}(\mathbf{x}_i, \mathbf{R}_j, \mathbf{t}_j) - \begin{bmatrix} u_{i,j} \\ v_{i,j} \end{bmatrix} ||_2^2$$

  with w_{ij} indicator whether they are visible or not, $\mathbf{P}(\mathbf{x}_i, \mathbf{R}_j, \mathbf{t}_j)$ the predicted image location, and $(u_{i,j}, v_{i,j})$ the observed image location

- Minimizing this is called bundle adjustment

Bundle Adjustment

- Minimize sum of squared reprojection errors:

$$g(\mathbf{X}, \mathbf{R}, \mathbf{T}) = \sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} ||\mathbf{P}(\mathbf{x}_i, \mathbf{R}_j, \mathbf{t}_j) - \begin{bmatrix} u_{i,j} \\ v_{i,j} \end{bmatrix} ||_2^2$$

  with w_{ij} indicator whether they are visible or not, $\mathbf{P}(\mathbf{x}_i, \mathbf{R}_j, \mathbf{t}_j)$ the predicted image location, and $(u_{i,j}, v_{i,j})$ the observed image location

- Minimizing this is called bundle adjustment
- We saw some other versions in class

Bundle Adjustment

- Minimize sum of squared reprojection errors:

$$g(\mathbf{X}, \mathbf{R}, \mathbf{T}) = \sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} || \mathbf{P}(\mathbf{x}_i, \mathbf{R}_j, \mathbf{t}_j) - \begin{bmatrix} u_{i,j} \\ v_{i,j} \end{bmatrix} ||_2^2$$

  with w_{ij} indicator whether they are visible or not, $\mathbf{P}(\mathbf{x}_i, \mathbf{R}_j, \mathbf{t}_j)$ the predicted image location, and $(u_{i,j}, v_{i,j})$ the observed image location

- Minimizing this is called bundle adjustment
- We saw some other versions in class
- Optimized using non-linear least squares, e.g. Levenberg-Marquardt

Bundle Adjustment

- Minimize sum of squared reprojection errors:

$$g(\mathbf{X}, \mathbf{R}, \mathbf{T}) = \sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} ||\mathbf{P}(\mathbf{x}_i, \mathbf{R}_j, \mathbf{t}_j) - \begin{bmatrix} u_{i,j} \\ v_{i,j} \end{bmatrix} ||_2^2$$

  with w_{ij} indicator whether they are visible or not, $\mathbf{P}(\mathbf{x}_i, \mathbf{R}_j, \mathbf{t}_j)$ the predicted image location, and $(u_{i,j}, v_{i,j})$ the observed image location

- Minimizing this is called bundle adjustment
- We saw some other versions in class
- Optimized using non-linear least squares, e.g. Levenberg-Marquardt
- Initialization is very important

Bundle Adjustment

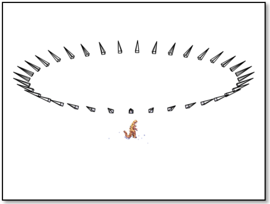
- Minimize sum of squared reprojection errors:

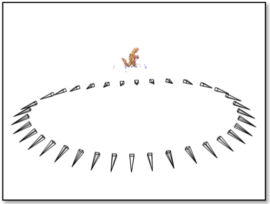
$$g(\mathbf{X}, \mathbf{R}, \mathbf{T}) = \sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} || \mathbf{P}(\mathbf{x}_i, \mathbf{R}_j, \mathbf{t}_j) - \begin{bmatrix} u_{i,j} \\ v_{i,j} \end{bmatrix} ||_2^2$$

  with w_{ij} indicator whether they are visible or not, $\mathbf{P}(\mathbf{x}_i, \mathbf{R}_j, \mathbf{t}_j)$ the predicted image location, and $(u_{i,j}, v_{i,j})$ the observed image location
- Minimizing this is called bundle adjustment
- We saw some other versions in class
- Optimized using non-linear least squares, e.g. Levenberg-Marquardt
- Initialization is very important

Failure cases

- Necker reversal

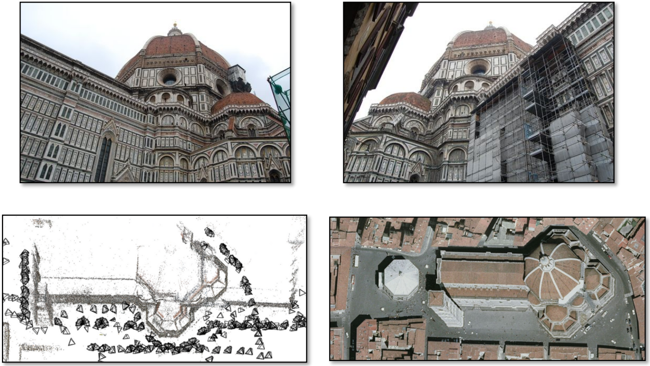




[Source: N. Snavely]

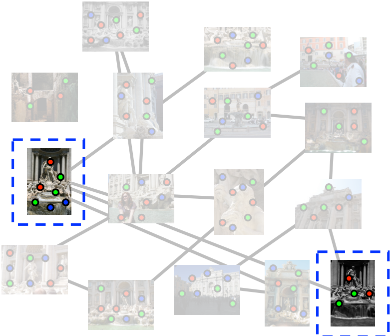
Failure cases

- Repetitive Patterns



[Source: N. Snavely]

Incremental Structure From Motion



[Source: N. Snavely]

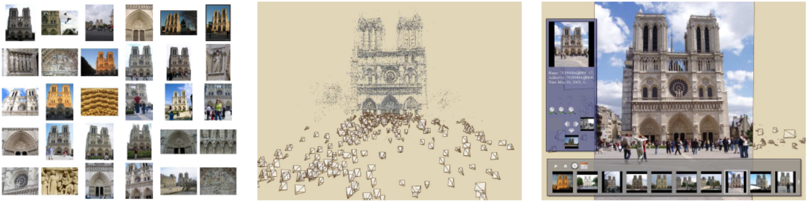
Incremental Structure From Motion

- What's the problem with this approach?

More Results

[Source: N. Snavely]

Applications: 3D Reconstruction from Photo Collections



(a) (b) (c)

Figure 1: Our system takes unstructured collections of photographs such as those from online image searches (a) and reconstructs 3D points and viewpoints (b) to enable novel ways of browsing the photos (c).

[N. Snavely et al. Siggraph 2006]