Image Segmentation

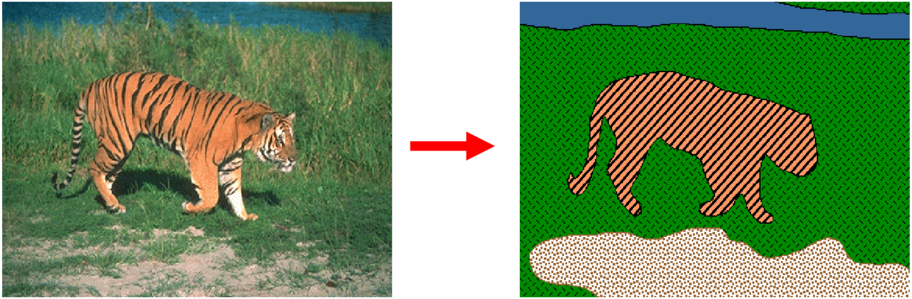
Philipp Krähenbühl

Stanford University

April 24, 2013

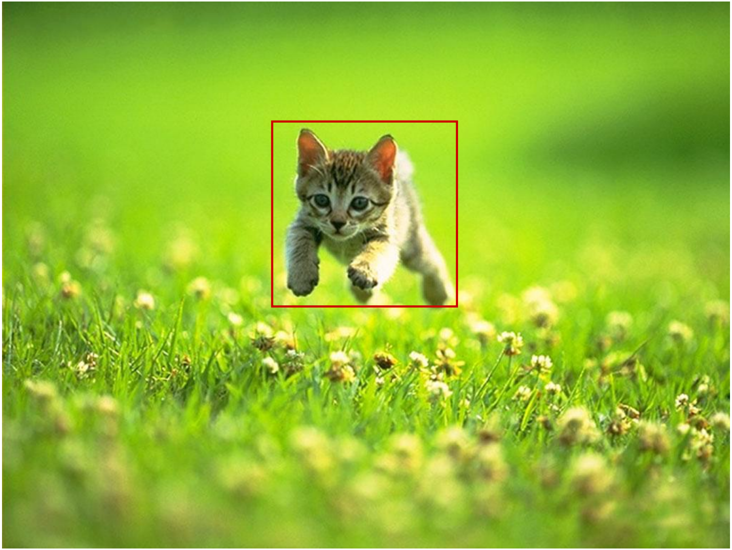
Image Segmentation

- Goal: identify groups of pixels that go together



Success Story

Success Story



Success Story

Gestalt Theory

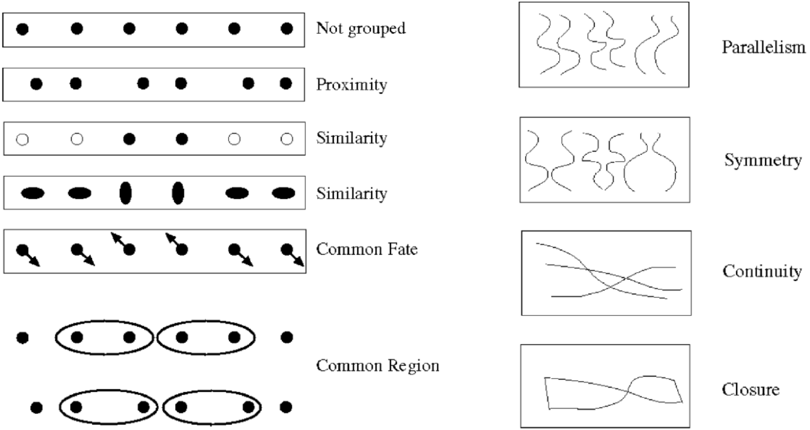
- Gestalt: whole or group
  - The whole is greater than the sum of its parts
  - Relationships between parts can yield new properties/features
- Psychologists identified series of factors that predispose set of elements to be grouped (by human visual system)

Max Wertheimer (1880-1943)

I stand at the window and see a house, trees, sky. Theoretically I might say there were 327 brightnesses and nuances of color. Do I have "327"? No. I have sky, house, and trees.

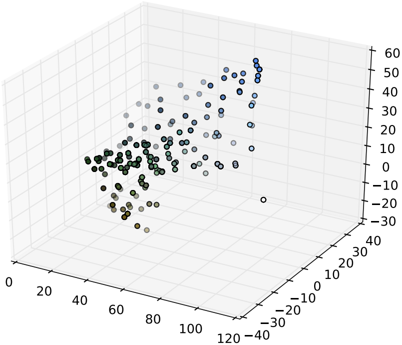
Untersuchungen zur Lehre von der Gestalt, Psychologische Forschung, Vol. 4, pp. 301-350, 1923
`http://psy.ed.asu.edu/~classics/Wertheimer/Forms/forms.htm`

Gestalt Theory



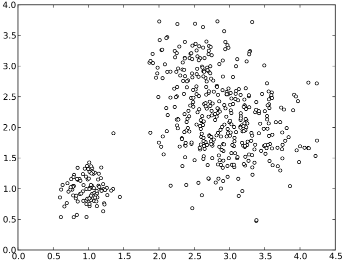
- These factors make intuitive sense, but are very difficult to translate into algorithms.

Segmentation as clustering



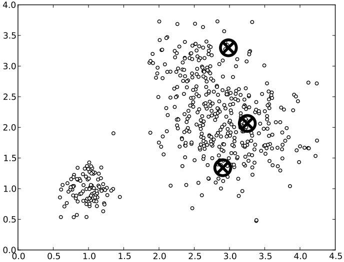
- Pixels are points in a high dimensional space
  - color: 3d
  - color+location:5d
- Cluster pixels into segment

K-Means clustering



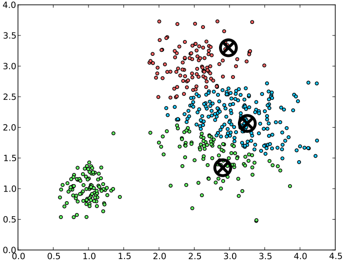
1. Randomly initialize K cluster centers, $c_1, \ldots, c_k$
2. Given cluster centers, determine points in each cluster
   - For each point p, find the closest c_i. Put p into cluster i.
3. Given points in each cluster, solve for c_i
   - Set c_i to be the mean of points in cluster i
4. If c_i have changed, repeat Step 2

K-Means clustering



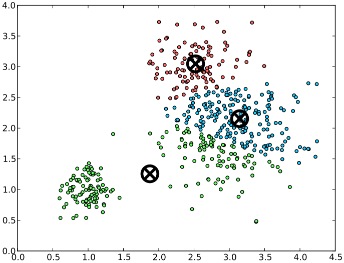
1. Randomly initialize K cluster centers, $c_1, \ldots, c_k$
2. Given cluster centers, determine points in each cluster
   - For each point p, find the closest c_i. Put p into cluster i.
3. Given points in each cluster, solve for c_i
   - Set c_i to be the mean of points in cluster i
4. If c_i have changed, repeat Step 2

K-Means clustering



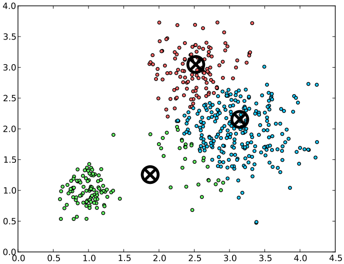
1. Randomly initialize K cluster centers, $c_1, \ldots, c_k$
2. Given cluster centers, determine points in each cluster
   - For each point p, find the closest c_i. Put p into cluster i.
3. Given points in each cluster, solve for c_i
   - Set c_i to be the mean of points in cluster i
4. If c_i have changed, repeat Step 2

K-Means clustering



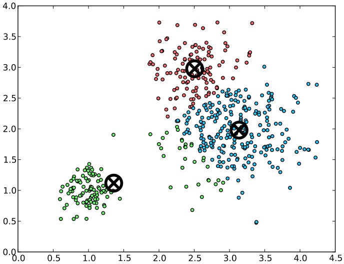
1. Randomly initialize K cluster centers, $c_1, \ldots, c_k$
2. Given cluster centers, determine points in each cluster
   - For each point p, find the closest c_i. Put p into cluster i.
3. Given points in each cluster, solve for c_i
   - Set c_i to be the mean of points in cluster i
4. If c_i have changed, repeat Step 2

K-Means clustering



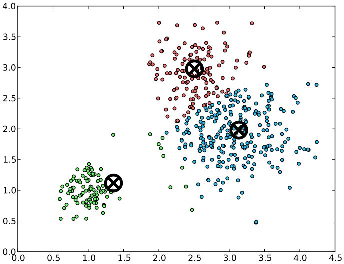
1. Randomly initialize K cluster centers, $c_1, \ldots, c_k$
2. Given cluster centers, determine points in each cluster
   - For each point p, find the closest c_i. Put p into cluster i.
3. Given points in each cluster, solve for c_i
   - Set c_i to be the mean of points in cluster i
4. If c_i have changed, repeat Step 2

K-Means clustering



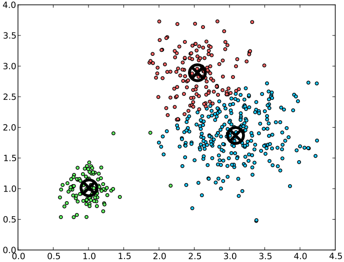
1. Randomly initialize K cluster centers, $c_1, \ldots, c_k$
2. Given cluster centers, determine points in each cluster
   - For each point p, find the closest c_i. Put p into cluster i.
3. Given points in each cluster, solve for c_i
   - Set c_i to be the mean of points in cluster i
4. If c_i have changed, repeat Step 2

K-Means clustering



1. Randomly initialize K cluster centers, $c_1, \ldots, c_k$
2. Given cluster centers, determine points in each cluster
   - For each point p, find the closest c_i. Put p into cluster i.
3. Given points in each cluster, solve for c_i
   - Set c_i to be the mean of points in cluster i
4. If c_i have changed, repeat Step 2

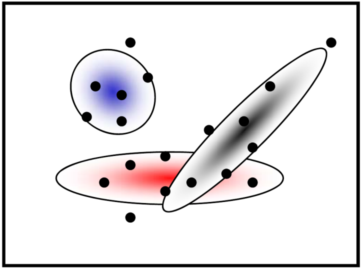
K-Means clustering



1. Randomly initialize K cluster centers, $c_1, \ldots, c_k$
2. Given cluster centers, determine points in each cluster
   - For each point p, find the closest c_i. Put p into cluster i.
3. Given points in each cluster, solve for c_i
   - Set c_i to be the mean of points in cluster i
4. If c_i have changed, repeat Step 2

K-Means clustering

Expectation Maximization (EM)



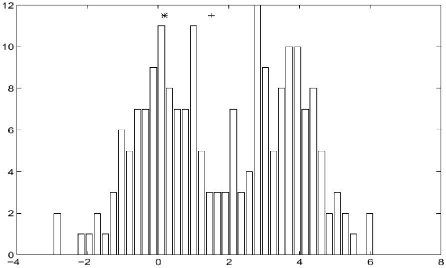
- Goal
  - Find blob parameters θ that maximize the likelihood function:

$$P(data|\theta) = \prod_x P(x|\theta)$$

- Approach:
  1. E-step: given current guess of blobs, compute ownership of each point
  2. M-step: given ownership probabilities, update blobs to maximize likelihood function
  3. Repeat until convergence

Expectation Maximization (EM)

Mean-Shift Algorithm



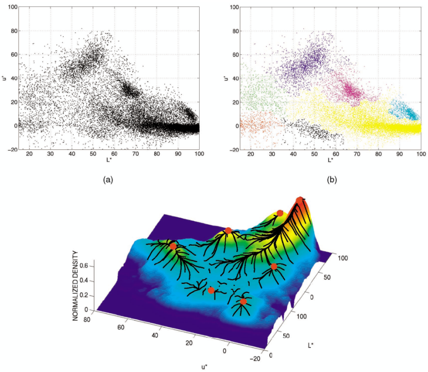
Iterative Mode search

1. Initialize random seed, and window W
2. Calculate center of gravity (the "mean") of W: $\sum_{x \in W} xH(x)$
3. Shift the search window to the mean
4. Repeat Step 2 until convergence

Mean-Shift Segmentation

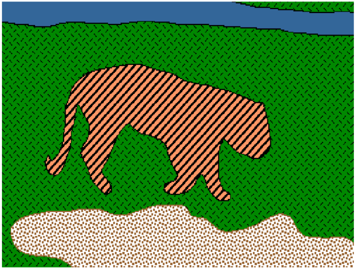
Iterative Mode search

- Find features (color, gradients, texture, etc)
- Initialize windows at individual pixel locations
- Perform mean shift for each window until convergence
- Merge windows that end up near the same "peak" or mode



Expectation Maximization (EM)

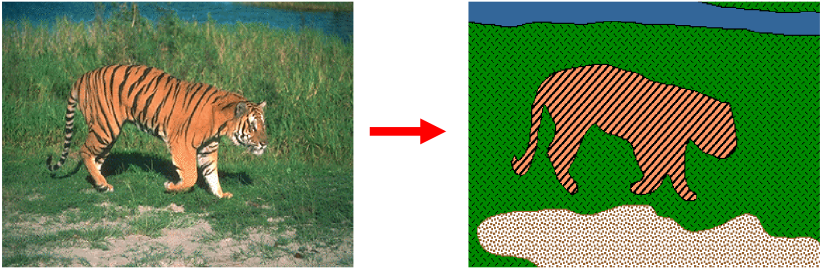
- Goal: identify groups of pixels that go together



- Up to now, we have focused on ways to group pixels into image segments based on their appearance...
  - Segmentation as clustering.
- We also want to enforce region constraints.
  - Spatial consistency
  - Smooth borders

Back to Image Segmentation

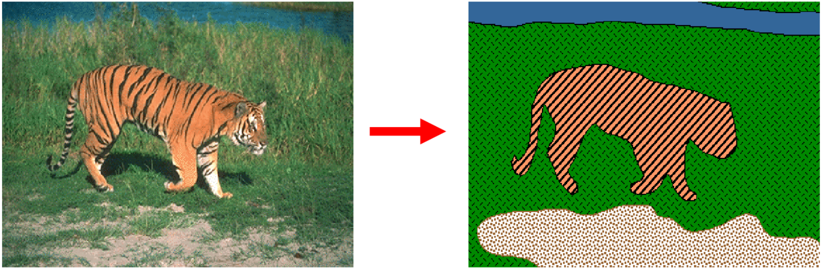
- Goal: identify groups of pixels that go together



- Up to now, we have focused on ways to group pixels into image segments based on their appearance...
  - Segmentation as clustering.
- We also want to enforce region constraints.
  - Spatial consistency
  - Smooth borders

Back to Image Segmentation

- Goal: identify groups of pixels that go together



- Up to now, we have focused on ways to group pixels into image segments based on their appearance...
  - Segmentation as clustering.
- We also want to enforce region constraints.
  - Spatial consistency
  - Smooth borders

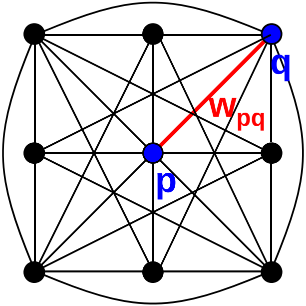
What we will learn today?

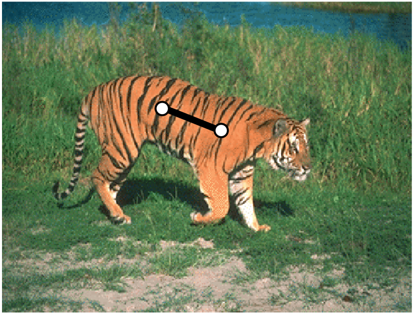
- Graph theoretic segmentation
  - Normalized Cuts
  - Using texture features
- Segmentation as Energy Minimization
  - Markov Random Fields (MRF) / Conditional Random Fields (CRF)
  - Graph cuts for image segmentation
  - Applications

What we will learn today?

- Graph theoretic segmentation
  - Normalized Cuts
  - Using texture features
- Segmentation as Energy Minimization
  - Markov Random Fields (MRF) / Conditional Random Fields (CRF)
  - Graph cuts for image segmentation
  - Applications

Images as Graphs

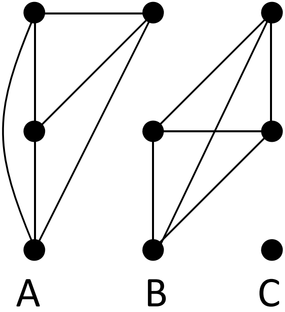


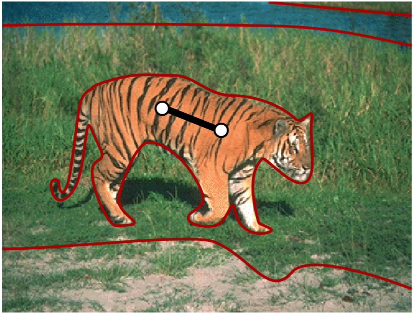


- (Fully-Connected) Graph
  - Node (vertex) for every pixel
  - Link between (every) pair of pixels, (p,q)
  - Affinity weight w_{pq} for each link (edge)
    - w_{pq} measures similarity
    - Inverse proportional to distance (difference in color and position)

Slide Credit: *Steve Seitz*

Segmentation by Graph Cuts



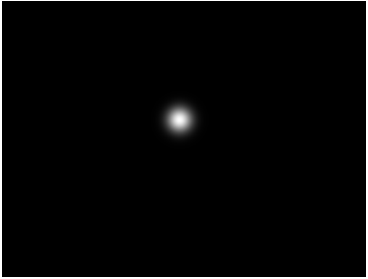


- Break Graph into Segments (cliques)
  - Delete links that cross between segments
  - Easiest to break links that low similarity (low affinity weight)
    - Similar pixels should be in the same segment
    - Dissimilar pixels should be if different segments

Slide Credit: *Steve Seitz*

Measuring Affinity

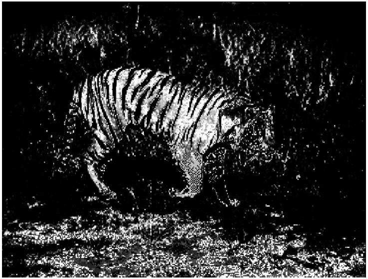
- Distance
$$\exp\left(-\frac{1}{2\sigma^2}\|x-y\|^2\right)$$
- Intensity
$$\exp\left(-\frac{1}{2\sigma^2}\|I(x)-I(y)\|^2\right)$$
- Color
$$\exp\left(-\frac{1}{2\sigma^2}\underbrace{dist(c(x),c(y))^2}_{\text{suitable color distance}}\right)$$
- Texture
$$\exp\left(-\frac{1}{2\sigma^2}\|\underbrace{f(x)-f(y)}_{\text{Filter output}}\|^2\right)$$



Source: *Forsyth & Ponce*

Measuring Affinity

- Distance
$$\exp\left(-\frac{1}{2\sigma^2}\|x-y\|^2\right)$$
- Intensity
$$\exp\left(-\frac{1}{2\sigma^2}\|I(x)-I(y)\|^2\right)$$
- Color
$$\exp\left(-\frac{1}{2\sigma^2}\underbrace{dist(c(x),c(y))^2}_{\text{suitable color distance}}\right)$$
- Texture
$$\exp\left(-\frac{1}{2\sigma^2}\|\underbrace{f(x)-f(y)}_{\text{Filter output}}\|^2\right)$$



Source: *Forsyth & Ponce*

Measuring Affinity

- Distance
$$\exp(-\frac{1}{2\sigma^2}\|x-y\|^2)$$
- Intensity
$$\exp(-\frac{1}{2\sigma^2}\|I(x)-I(y)\|^2)$$
- Color
$$\exp(-\frac{1}{2\sigma^2}\underbrace{dist(c(x),c(y))^2}_{\text{suitable color distance}})$$
- Texture
$$\exp(-\frac{1}{2\sigma^2}\|\underbrace{f(x)-f(y)}_{\text{Filter output}}\|^2)$$

Source: *Forsyth & Ponce*

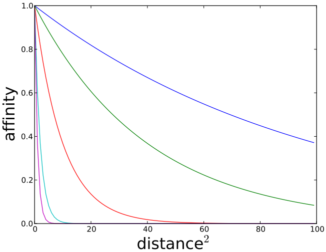
Measuring Affinity

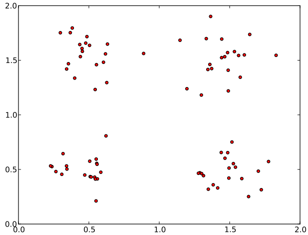
- Distance
$$\exp(-\frac{1}{2\sigma^2}\|x-y\|^2)$$
- Intensity
$$\exp(-\frac{1}{2\sigma^2}\|I(x)-I(y)\|^2)$$
- Color
$$\exp(-\frac{1}{2\sigma^2}\underbrace{dist(c(x),c(y))^2}_{\text{suitable color distance}})$$
- Texture
$$\exp(-\frac{1}{2\sigma^2}\|\underbrace{f(x)-f(y)}_{\text{Filter output}}\|^2)$$

Source: *Forsyth & Ponce*

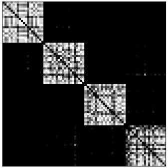
Scale Affects Affinity

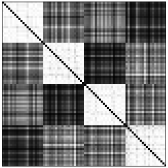
- Small σ: group only nearby points
- Large σ: group far-away points






small σ


medium σ


large σ

Slide Credit: *Svetlana Lazebnik*

Graph Cut: Using Eigenvalues

- Affinity matrix W
- Extract a single good cluster (v_n)
  - $v_n(i)$: probability of point i belonging to the cluster
  - Elements have high affinity with each other

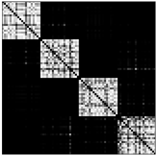
$$v_n^\top W v_n$$

  - Constraint $v_n^\top v_n = 1$
    - Prevents $v_n \rightarrow \infty$
- Constraint objective

$$v_n^\top W v_n - \lambda(1 - v_n^\top v_n)$$

- Reduces to Eigenvalue problem

$$v_n^\top W = \lambda v_n$$



Graph Cut: Using Eigenvalues

- Affinity matrix W
- Extract a single good cluster (v_n)
  - $v_n(i)$: probability of point i belonging to the cluster
  - Elements have high affinity with each other

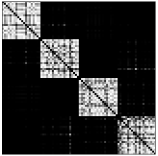
$$v_n^\top W v_n$$

  - Constraint $v_n^\top v_n = 1$
    - Prevents $v_n \to \infty$
- Constraint objective

$$v_n^\top W v_n - \lambda(1 - v_n^\top v_n)$$

- Reduces to Eigenvalue problem

$$v_n^\top W = \lambda v_n$$

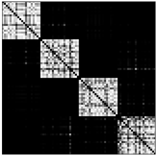

A

Graph Cut: Using Eigenvalues

- Affinity matrix W
- Extract a single good cluster (v_n)
  - $v_n(i)$: probability of point i belonging to the cluster
  - Elements have high affinity with each other

$$v_n^\top W v_n$$

  - Constraint $v_n^\top v_n = 1$
    - Prevents $v_n \to \infty$



A

- Constraint objective

$$v_n^\top W v_n - \lambda(1 - v_n^\top v_n)$$

- Reduces to Eigenvalue problem

$$v_n^\top W = \lambda v_n$$

Graph Cut: Using Eigenvalues

- Affinity matrix W
- Extract a single good cluster (v_n)
  - $v_n(i)$: probability of point i belonging to the cluster
  - Elements have high affinity with each other

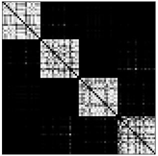
$$v_n^\top W v_n$$

  - Constraint $v_n^\top v_n = 1$
    - Prevents $v_n \to \infty$
- Constraint objective

$$v_n^\top W v_n - \lambda(1 - v_n^\top v_n)$$

- Reduces to Eigenvalue problem

$$v_n^\top W = \lambda v_n$$


A

Graph Cut: Using Eigenvalues

- Affinity matrix W
- Extract a single good cluster (v_n)
  - $v_n(i)$: probability of point i belonging to the cluster
  - Elements have high affinity with each other

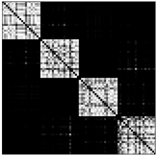
$$v_n^\top W v_n$$

  - Constraint $v_n^\top v_n = 1$
    - Prevents $v_n \to \infty$
- Constraint objective

$$v_n^\top W v_n - \lambda(1 - v_n^\top v_n)$$

- Reduces to Eigenvalue problem

$$v_n^\top W = \lambda v_n$$



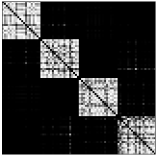
A

Graph Cut: Using Eigenvalues

- Affinity matrix W
- Extract a single good cluster (v_n)
  - $v_n(i)$: probability of point i belonging to the cluster
  - Elements have high affinity with each other

$$v_n^\top W v_n$$

  - Constraint $v_n^\top v_n = 1$
    - Prevents $v_n \to \infty$



A

- Constraint objective

$$v_n^\top W v_n - \lambda(1 - v_n^\top v_n)$$

- Reduces to Eigenvalue problem

$$v_n^\top W = \lambda v_n$$

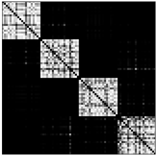
Graph Cut: Using Eigenvalues

- Affinity matrix W
- Extract a single good cluster (v_n)
  - $v_n(i)$: probability of point i belonging to the cluster
  - Elements have high affinity with each other

$$v_n^\top W v_n$$

  - Constraint $v_n^\top v_n = 1$
    - ★ Prevents $v_n \to \infty$
- Constraint objective

$$v_n^\top W v_n - \lambda(1 - v_n^\top v_n)$$



- Reduces to Eigenvalue problem

$$v_n^\top W = \lambda v_n$$

Graph Cut: Using Eigenvalues

- Affinity matrix W
- Extract a single good cluster (v_n)
  - $v_n(i)$: probability of point i belonging to the cluster
  - Elements have high affinity with each other

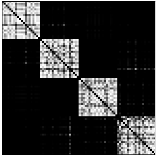
$$v_n^\top W v_n$$

  - Constraint $v_n^\top v_n = 1$
    - Prevents $v_n \to \infty$
- Constraint objective

$$v_n^\top W v_n - \lambda(1 - v_n^\top v_n)$$

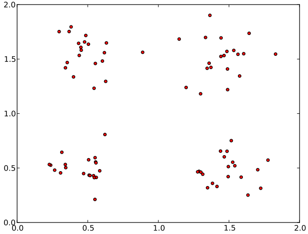
- Reduces to Eigenvalue problem

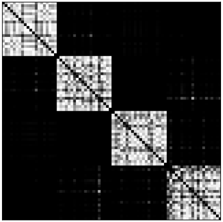
$$v_n^\top W = \lambda v_n$$



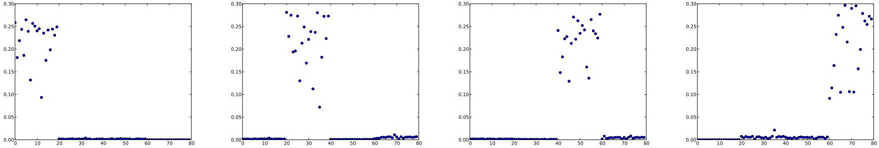
A

Graph Cut: Using Eigenvalues





4 largest eigenvalues



Clustering by Graph Eigenvectors

1. Construct an affinity matrix
2. Compute the eigenvalues and eigenvectors of the affinity matrix
3. Until there are sufficient clusters
   - Take the eigenvector corresponding to the largest unprocessed eigenvalue
   - zero all components corresponding to elements that have already been clustered
   - threshold the remaining components to determine which element belongs to this cluster,
     - choose a threshold by clustering the components, or using a threshold fixed in advance
   - If all elements have been accounted for, there are sufficient clusters; end

Clustering by Graph Eigenvectors

1. Construct an affinity matrix
2. Compute the eigenvalues and eigenvectors of the affinity matrix
3. Until there are sufficient clusters
    - Take the eigenvector corresponding to the largest unprocessed eigenvalue
    - zero all components corresponding to elements that have already been clustered
    - threshold the remaining components to determine which element belongs to this cluster,
        - choose a threshold by clustering the components, or using a threshold fixed in advance.
    - If all elements have been accounted for, there are sufficient clusters: end

Clustering by Graph Eigenvectors

1. Construct an affinity matrix
2. Compute the eigenvalues and eigenvectors of the affinity matrix
3. Until there are sufficient clusters
   - Take the eigenvector corresponding to the largest unprocessed eigenvalue
   - zero all components corresponding to elements that have already been clustered
   - threshold the remaining components to determine which element belongs to this cluster,
     - choose a threshold by clustering the components, or using a threshold fixed in advance.
   - If all elements have been accounted for, there are sufficient clusters: end

Clustering by Graph Eigenvectors

1. Construct an affinity matrix
2. Compute the eigenvalues and eigenvectors of the affinity matrix
3. Until there are sufficient clusters
   - Take the eigenvector corresponding to the largest unprocessed eigenvalue
   - zero all components corresponding to elements that have already been clustered
   - threshold the remaining components to determine which element belongs to this cluster,
     - choose a threshold by clustering the components, or using a threshold fixed in advance.
   - If all elements have been accounted for, there are sufficient clusters: end

Clustering by Graph Eigenvectors

1. Construct an affinity matrix
2. Compute the eigenvalues and eigenvectors of the affinity matrix
3. Until there are sufficient clusters
   - Take the eigenvector corresponding to the largest unprocessed eigenvalue
   - zero all components corresponding to elements that have already been clustered
   - threshold the remaining components to determine which element belongs to this cluster,
     - choose a threshold by clustering the components, or using a threshold fixed in advance.
   - If all elements have been accounted for, there are sufficient clusters: end

Clustering by Graph Eigenvectors

1. Construct an affinity matrix
2. Compute the eigenvalues and eigenvectors of the affinity matrix
3. Until there are sufficient clusters
   - Take the eigenvector corresponding to the largest unprocessed eigenvalue
   - zero all components corresponding to elements that have already been clustered
   - threshold the remaining components to determine which element belongs to this cluster,
     - choose a threshold by clustering the components, or using a threshold fixed in advance.
   - If all elements have been accounted for, there are sufficient clusters: end

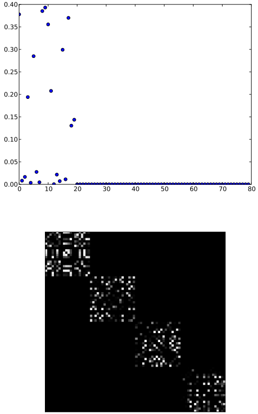
Clustering by Graph Eigenvectors

1. Construct an affinity matrix
2. Compute the eigenvalues and eigenvectors of the affinity matrix
3. Until there are sufficient clusters
   - Take the eigenvector corresponding to the largest unprocessed eigenvalue
   - zero all components corresponding to elements that have already been clustered
   - threshold the remaining components to determine which element belongs to this cluster,
     - choose a threshold by clustering the components, or using a threshold fixed in advance.
   - If all elements have been accounted for, there are sufficient clusters: end

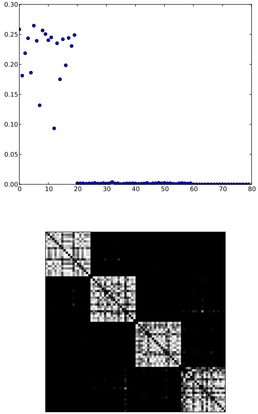
Clustering by Graph Eigenvectors

1. Construct an affinity matrix
2. Compute the eigenvalues and eigenvectors of the affinity matrix
3. Until there are sufficient clusters
   - Take the eigenvector corresponding to the largest unprocessed eigenvalue
   - zero all components corresponding to elements that have already been clustered
   - threshold the remaining components to determine which element belongs to this cluster,
     - choose a threshold by clustering the components, or using a threshold fixed in advance.
   - If all elements have been accounted for, there are sufficient clusters: end

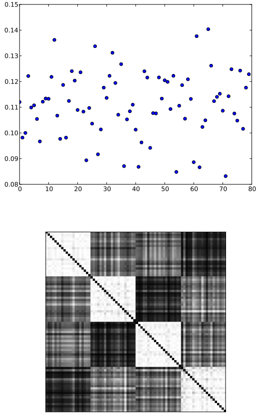
Effects of the scaling



small σ

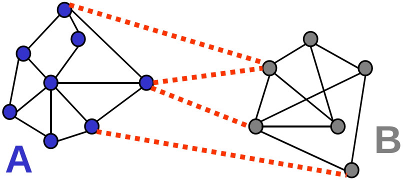


medium σ



large σ

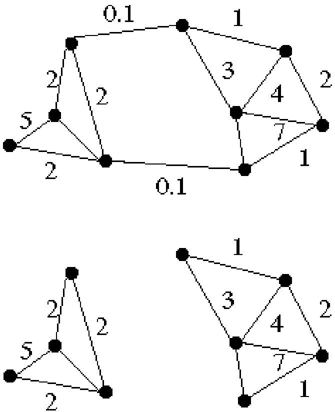
Graph Cut

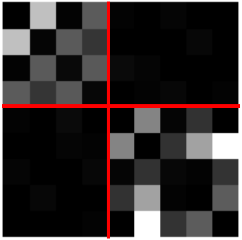


- Find set of edges whose removal makes graph disconnected
- Cost of a cut
  - Sum of weights of cut edges: $cut(A, B) = \sum_{p \in A, q \in B} w_{pq}$
- Graph cut gives us a segmentation
  - What is a "good" graph cut and how do we find one?

Slide Credit: *Steve Seitz*

Graph Cut





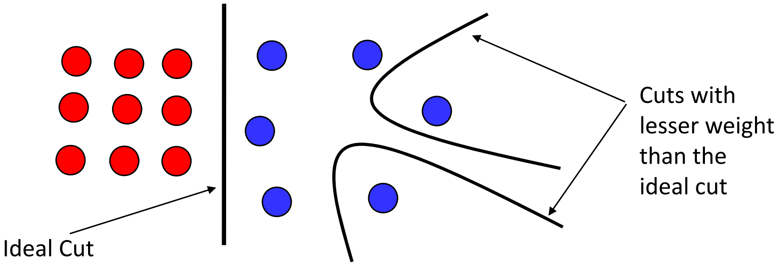
Here, the cut is nicely defined by the block-diagonal structure of the affinity matrix.

$\Rightarrow$ *How can this be generalized?*

Image Source: *Forsyth & Ponce*

Minimum Cut

- We can do segmentation by finding the minimum cut in a graph
  - a minimum cut of a graph is a cut whose cutset has the smallest affinity.
  - Efficient algorithms exist for doing this (max-flow)
- Drawback
  - Weight of cut proportional to number of edges in the cut
  - Minimum cut tends to cut off very small, isolated components



Slide Credit: *Khurram Hassan-Shafique*

Normalized Cut (NCut)

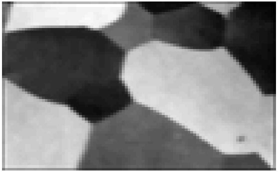
- A minimum cut penalizes large segments
- This can be fixed by normalizing for size of segments
- The normalized cut cost is:

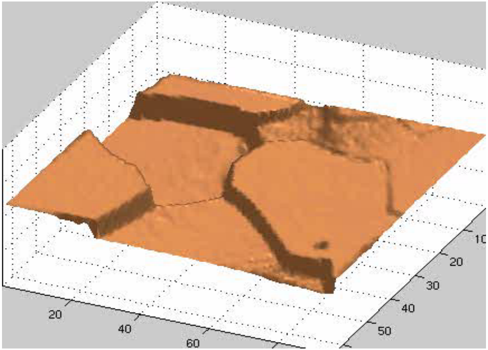
$$
\begin{aligned}
Ncut(A, B) &= \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)} \\
&= cut(A, B) \left[\frac{1}{\sum_{p \in A, q} w_{p,q}} + \frac{1}{\sum_{q \in B, p} w_{p,q}} \right]
\end{aligned}
$$

- $assoc(A, V)$ = sum of weights of all edges in V that touch A
- The exact solution is NP-hard but an approximation can be computed by solving a generalized eigenvalue problem.

J. Shi and J. Malik. Normalized cuts and image segmentation. PAMI 2000

Interpretation as a Dynamical System



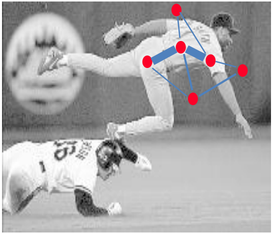


- Treat the links as springs and shake the system
  - Elasticity proportional to cost
  - Vibration "modes" correspond to segments
    - ★ Can compute these by solving a generalized eigenvector problem

Slide Credit: *Steve Seitz*

NCuts as a Generalized Eigenvalue Problem

- Definitions
  - W: the affinity matrix
  - D: diagonal matrix, $D_{ii} = \sum_j W_{ij}$
  - x: a vector in $\{-1, 1\}^N$,
- Rewriting the Normalized Cut in matrix form



$$
\begin{aligned}
Ncut(A, B) &= \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)} \\
&= \ldots
\end{aligned}
$$

Slide Credit: *Jitentra Malik*

Some more math...

We see again this is an unbiased measure, which reflects how tightly on average nodes within the group are connected to each other.

Another important property of this definition of association and disassociation of a partition is that they are naturally related:

$$
\begin{aligned}
Ncut(A,B) &= \frac{cut(A,B)}{asso(A,V)} + \frac{cut(A,B)}{asso(B,V)} \\
&= \frac{asso(A,V) - asso(A,A)}{asso(A,V)} \\
&\quad + \frac{asso(B,V) - asso(B,B)}{asso(B,V)} \\
&= 2 - \left(\frac{asso(A,A)}{asso(A,V)} + \frac{asso(B,B)}{asso(B,V)}\right) \\
&= 2 - Nasso(A,B)
\end{aligned}
$$

Hence the two partition criteria that we seek in our grouping algorithm, minimizing the disassociation between the groups and maximizing the association within the group, are in fact identical, and can be satisfied simultaneously. In our algorithm, we will use this *normalized cut* as the partition criterion.

Having defined the graph partition criterion that we want to optimize, we will show how such an optimal partition can be computed efficiently.

2.1 Computing the optimal partition

Given a partition of nodes of a graph, V, into two sets A and B, let $\mathbf{x}$ be an $N = |\mathbf{V}|$ dimensional indicator vector, $x_i = 1$ if node i is in A, and -1 otherwise. Let $\mathbf{d}(i) = \sum_j w(i,j)$, be the total connection from node i to all other nodes. With the definitions $\mathbf{x}$ and $\mathbf{d}$ we can rewrite $Ncut(A,B)$ as:

$$
\begin{aligned}
Ncut(A,B) &= \frac{cut(A,B)}{asso(A,V)} + \frac{cut(B,A)}{asso(B,V)} \\
&= \frac{\sum_{(\mathbf{x}_i>0,\mathbf{x}_j<0)} -w_{ij}\mathbf{x}_i\mathbf{x}_j}{\sum_{\mathbf{x}_i>0} \mathbf{d}_i} \\
&\quad + \frac{\sum_{(\mathbf{x}_i<0,\mathbf{x}_j>0)} -w_{ij}\mathbf{x}_i\mathbf{x}_j}{\sum_{\mathbf{x}_i<0} \mathbf{d}_i}
\end{aligned}
$$

Let $\mathbf{D}$ be an $N \times N$ diagonal matrix with $\mathbf{d}$ on its diagonal, $\mathbf{W}$ be an $N \times N$ symmetrical matrix with $\mathbf{W}(i,j) = w_{ij}$, $\mathbf{k} = \frac{\sum_{x_i>0} \mathbf{d}_i}{\sum_i \mathbf{d}_i}$, and $\mathbf{1}$ be an $N \times 1$ vector of all ones. Using the fact $\frac{\mathbf{1}+\mathbf{x}}{2}$ and $\frac{\mathbf{1}-\mathbf{x}}{2}$ are indicator vectors for $x_i > 0$ and $x_i < 0$ respectively, we can rewrite $4[Ncut(\mathbf{x})]$ as:

$$
\begin{aligned}
&= \frac{(\mathbf{1}+\mathbf{x})^T(\mathbf{D}-\mathbf{W})(\mathbf{1}+\mathbf{x})}{k\mathbf{1}^T\mathbf{D}\mathbf{1}} + \frac{(\mathbf{1}-\mathbf{x})^T(\mathbf{D}-\mathbf{W})(\mathbf{1}-\mathbf{x})}{(1-k)\mathbf{1}^T\mathbf{D}\mathbf{1}} \\
&= \frac{(\mathbf{x}^T(\mathbf{D}-\mathbf{W})\mathbf{x}+\mathbf{1}^T(\mathbf{D}-\mathbf{W})\mathbf{1})}{k(1-k)\mathbf{1}^T\mathbf{D}\mathbf{1}} + \frac{2(1-2k)\mathbf{1}^T(\mathbf{D}-\mathbf{W})\mathbf{x}}{k(1-k)\mathbf{1}^T\mathbf{D}\mathbf{1}}
\end{aligned}
$$

Let $\alpha(\mathbf{x}) = \mathbf{x}^T(\mathbf{D}-\mathbf{W})\mathbf{x}$, $\beta(\mathbf{x}) = \mathbf{1}^T(\mathbf{D}-\mathbf{W})\mathbf{x}$, $\gamma = \mathbf{1}^T(\mathbf{D}-\mathbf{W})\mathbf{1}$, and $M = \mathbf{1}^T\mathbf{D}\mathbf{1}$, we can then further expand the above equation as:

$$
\begin{aligned}
&= \frac{(\alpha(\mathbf{x})+\gamma) + 2(1-2\mathbf{k})\beta(\mathbf{x})}{\mathbf{k}(1-\mathbf{k})M} \\
&= \frac{(\alpha(\mathbf{x})+\gamma) + 2(1-2\mathbf{k})\beta(\mathbf{x})}{\mathbf{k}(1-\mathbf{k})M} - \frac{2(\alpha(\mathbf{x})+\gamma)}{M} \\
&\quad + \frac{2\alpha(\mathbf{x})}{M} + \frac{2\gamma}{M}
\end{aligned}
$$

dropping the last constant term, which in this case equals 0, we get

$$
\begin{aligned}
&= \frac{(1-2\mathbf{k}+2\mathbf{k}^2)(\alpha(\mathbf{x})+\gamma) + 2(1-2\mathbf{k})\beta(\mathbf{x})}{\mathbf{k}(1-\mathbf{k})M} + \frac{2\alpha(\mathbf{x})}{M} \\
&= \frac{\frac{(1-2k+2k^2)}{(1-k)^2}(\alpha(\mathbf{x})+\gamma) + \frac{2(1-2k)}{(1-k)^2}\beta(\mathbf{x})}{\frac{k}{1-k}M} + \frac{2\alpha(\mathbf{x})}{M}
\end{aligned}
$$

Letting $b = \frac{k}{1-k}$, and since $\gamma = 0$, it becomes,

$$
\begin{aligned}
&= \frac{(1+b^2)(\alpha(\mathbf{x})+\gamma) + 2(1-b^2)\beta(\mathbf{x})}{bM} + \frac{2b\alpha(\mathbf{x})}{bM} \\
&= \frac{(1+b^2)(\alpha(\mathbf{x})+\gamma)}{bM} + \frac{2(1-b^2)\beta(\mathbf{x})}{bM} + \frac{2b\alpha(\mathbf{x})}{bM} - \frac{2b\gamma}{bM} \\
&= \frac{(1+b^2)(\mathbf{x}^T(\mathbf{D}-\mathbf{W})\mathbf{x}+\mathbf{1}^T(\mathbf{D}-\mathbf{W})\mathbf{1})}{b\mathbf{1}^T\boldsymbol{D}\mathbf{1}} \\
&\quad + \frac{2(1-b^2)\mathbf{1}^T(\mathbf{D}-\mathbf{W})\mathbf{x}}{b\mathbf{1}^T\boldsymbol{D}\mathbf{1}} \\
&\quad + \frac{2b\mathbf{x}^T(\mathbf{D}-\mathbf{W})\mathbf{x}}{b\mathbf{1}^T\mathbf{D}\mathbf{1}} - \frac{2b\mathbf{1}^T(\mathbf{D}-\mathbf{W})\mathbf{1}}{b\mathbf{1}^T\boldsymbol{D}\mathbf{1}} \\
&= \frac{(\mathbf{1}+\mathbf{x})^T(\mathbf{D}-\mathbf{W})(\mathbf{1}+\mathbf{x})}{b\mathbf{1}^T\boldsymbol{D}\mathbf{1}} \\
&\quad + \frac{b^2(\mathbf{1}-\mathbf{x})^T(\mathbf{D}-\mathbf{W})(\mathbf{1}-\mathbf{x})}{b\mathbf{1}^T\boldsymbol{D}\mathbf{1}} \\
&\quad - \frac{2b(\mathbf{1}-\mathbf{x})^T(\mathbf{D}-\mathbf{W})(\mathbf{1}+\mathbf{x})}{b\mathbf{1}^T\boldsymbol{D}\mathbf{1}} \\
&= \frac{[(\mathbf{1}+\mathbf{x})-b(\mathbf{1}-\mathbf{x})]^T(\mathbf{D}-\mathbf{W})[(\mathbf{1}+\mathbf{x})-b(\mathbf{1}-\mathbf{x})]}{b\mathbf{1}^T\boldsymbol{D}\mathbf{1}}
\end{aligned}
$$

Setting $\mathbf{y} = (\mathbf{1}+\mathbf{x}) - b(\mathbf{1}-\mathbf{x})$, it is easy to see that

$$
\mathbf{y}^T\boldsymbol{D}\mathbf{1} = \sum_{x_i>0} \mathbf{d}_i - b\sum_{x_i<0} \mathbf{d}_i = 0 \qquad (4)
$$

since $b = \frac{k}{1-k} = \frac{\sum_{x_i>0} \mathbf{d}_i}{\sum_{x_i<0} \mathbf{d}_i}$, and

$$
\begin{aligned}
\mathbf{y}^T\boldsymbol{D}\mathbf{y} &= \sum_{x_i>0} \mathbf{d}_i + b^2\sum_{x_i<0} \mathbf{d}_i \\
&= b\sum_{x_i<0} \mathbf{d}_i + b^2\sum_{x_i<0} \mathbf{d}_i \\
&= b\left(\sum_{x_i<0} \mathbf{d}_i + b\sum_{x_i<0} \mathbf{d}_i\right) \\
&= b\mathbf{1}^T\boldsymbol{D}\mathbf{1}.
\end{aligned}
$$

Slide Credit: *Jitentra Malik*

NCuts as a Generalized Eigenvalue Problem

- After simplifications, we get

$$Ncut(A,B) = \frac{y^\top (D - W)y}{y^\top D y}$$

with $y_i \in \{-1, b\}$ and $y^\top D1 = 0$

Hard as a discrete problem

$\Downarrow$

Continuous approximation

- This is the Rayleigh Quotient
  - Solution given by the generalized eigenvalue problem

$$(D - W)y = \lambda D y$$

- Subtleties
  - Optimal solution is second smallest eigenvector
  - Gives continuous result—must convert into discrete values of y

Slide Credit: *Jitentra Malik*

NCuts as a Generalized Eigenvalue Problem

- After simplifications, we get

$$Ncut(A,B) = \frac{y^\top (D-W)y}{y^\top D y}$$

with $y_i \in \{-1, b\}$ and $y^\top D1 = 0$

- This is the Rayleigh Quotient
  - Solution given by the generalized eigenvalue problem

$$(D-W)y = \lambda D y$$

- Subtleties
  - Optimal solution is second smallest eigenvector
  - Gives continuous result—must convert into discrete values of y

Hard as a discrete problem

$\Downarrow$

Continuous approximation

Slide Credit: *Jitentra Malik*

NCuts as a Generalized Eigenvalue Problem

- After simplifications, we get

$$Ncut(A, B) = \frac{y^\top (D - W)y}{y^\top Dy}$$

with $y_i \in \{-1, b\}$ and $y^\top D1 = 0$

- This is the Rayleigh Quotient
  - Solution given by the generalized eigenvalue problem
$$(D - W)y = \lambda Dy$$
- Subtleties
  - Optimal solution is second smallest eigenvector
  - Gives continuous result—must convert into discrete values of y

Hard as a discrete problem

$\Downarrow$

Continuous approximation

Slide Credit: *Jitentra Malik*

NCuts Example

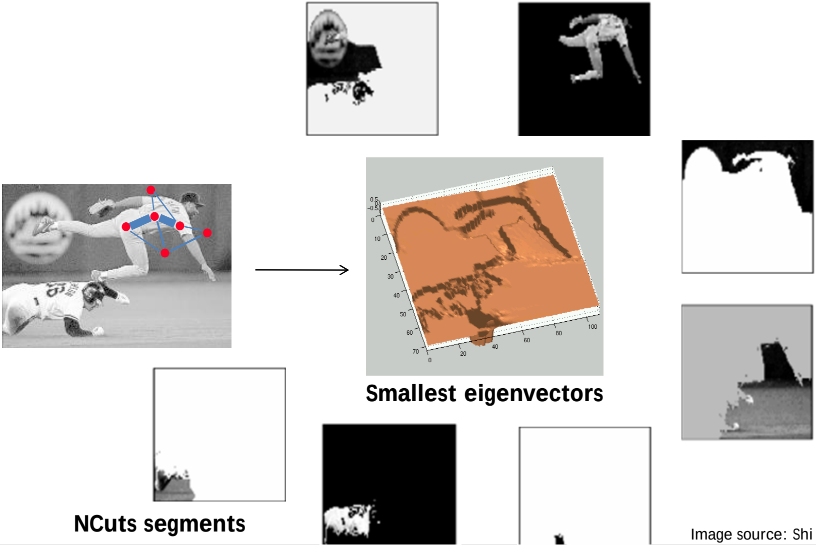
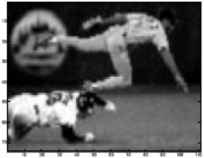


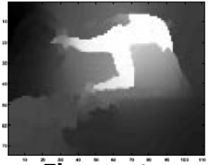
Image source: Shi & Malik

NCuts Example

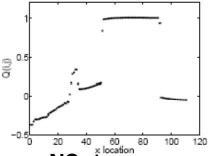
- Problem: eigenvectors take on continuous values
  - How to choose the splitting point to binarize the image?



Image



Eigenvector



NCut scores

- Possible procedures
  - Pick a constant value (0, or 0.5).
  - Pick the median value as splitting point.
  - Look for the splitting point that has the minimum NCut value:
    1. Choose n possible splitting points.
    2. Compute NCut value.
    3. Pick minimum.

NCuts: Overall Procedure

1. Construct a weighted graph $G = (V, E)$ from an image.
2. Connect each pair of pixels, and assign graph edge weights W_{ij} = Prob. that i and j belong to the same region.
3. Solve $(D - W)y = \lambda Dy$ for the smallest few eigenvectors. This yields a continuous solution.
4. Threshold eigenvectors to get a discrete cut
   - This is where the approximation is made (we're not solving NP).
5. Recursively subdivide if NCut value is below a pre-specified value.

NCuts Matlab code available at
`http://www.cis.upenn.edu/~jshi/software/`

NCuts: Overall Procedure

1. Construct a weighted graph $G = (V, E)$ from an image.
2. Connect each pair of pixels, and assign graph edge weights $W_{ij} =$ Prob. that i and j belong to the same region.
3. Solve $(D - W)y = \lambda Dy$ for the smallest few eigenvectors. This yields a continuous solution.
4. Threshold eigenvectors to get a discrete cut
   - This is where the approximation is made (we're not solving NP).
5. Recursively subdivide if NCut value is below a pre-specified value.

NCuts Matlab code available at
`http://www.cis.upenn.edu/~jshi/software/`

NCuts: Overall Procedure

1. Construct a weighted graph $G = (V, E)$ from an image.
2. Connect each pair of pixels, and assign graph edge weights $W_{ij} =$ Prob. that i and j belong to the same region.
3. Solve $(D - W)y = \lambda Dy$ for the smallest few eigenvectors. This yields a continuous solution.
4. Threshold eigenvectors to get a discrete cut
   - This is where the approximation is made (we're not solving NP).
5. Recursively subdivide if NCut value is below a pre-specified value.

NCuts Matlab code available at
`http://www.cis.upenn.edu/~jshi/software/`

NCuts: Overall Procedure

1. Construct a weighted graph $G = (V, E)$ from an image.
2. Connect each pair of pixels, and assign graph edge weights $W_{ij} =$ Prob. that i and j belong to the same region.
3. Solve $(D - W)y = \lambda Dy$ for the smallest few eigenvectors. This yields a continuous solution.
4. Threshold eigenvectors to get a discrete cut
   - This is where the approximation is made (we're not solving NP).
5. Recursively subdivide if NCut value is below a pre-specified value.

NCuts Matlab code available at
`http://www.cis.upenn.edu/~jshi/software/`

NCuts: Overall Procedure

1. Construct a weighted graph $G = (V, E)$ from an image.
2. Connect each pair of pixels, and assign graph edge weights $W_{ij} =$ Prob. that i and j belong to the same region.
3. Solve $(D - W)y = \lambda Dy$ for the smallest few eigenvectors. This yields a continuous solution.
4. Threshold eigenvectors to get a discrete cut
   - This is where the approximation is made (we're not solving NP).
5. Recursively subdivide if NCut value is below a pre-specified value.

NCuts Matlab code available at
`http://www.cis.upenn.edu/~jshi/software/`

NCuts: Overall Procedure

1. Construct a weighted graph $G = (V, E)$ from an image.
2. Connect each pair of pixels, and assign graph edge weights $W_{ij} =$ Prob. that i and j belong to the same region.
3. Solve $(D - W)y = \lambda Dy$ for the smallest few eigenvectors. This yields a continuous solution.
4. Threshold eigenvectors to get a discrete cut
   - This is where the approximation is made (we're not solving NP).
5. Recursively subdivide if NCut value is below a pre-specified value.

NCuts Matlab code available at
`http://www.cis.upenn.edu/~jshi/software/`

NCuts Results

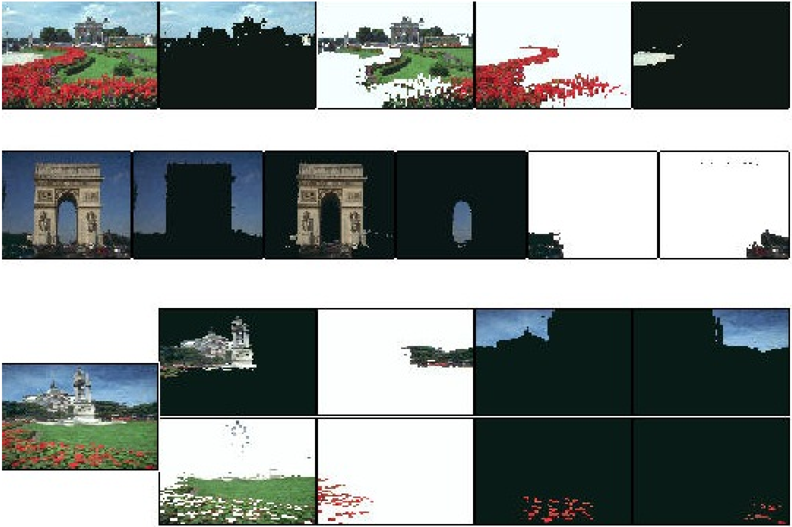
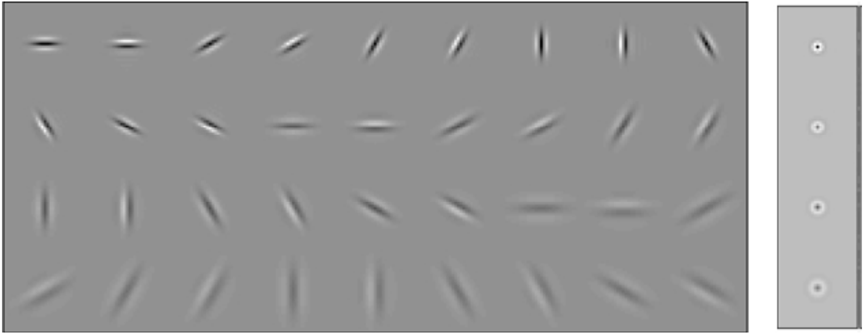


Image Source: Shi & Malik

Using Texture Features for Segmentation

- Texture descriptor is vector of filter bank outputs

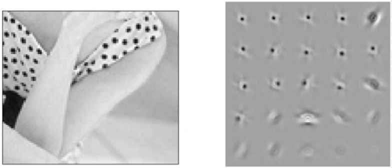
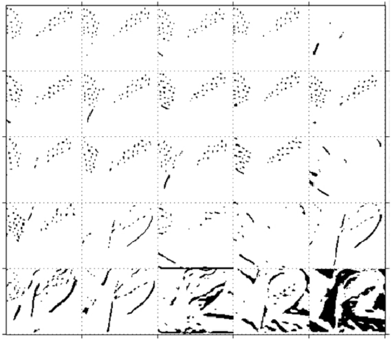


J. Malik, S. Belongie, T. Leung and J. Shi.

"Contour and Texture Analysis for Image Segmentation". IJCV 43(1),7-27,2001

Using Texture Features for Segmentation

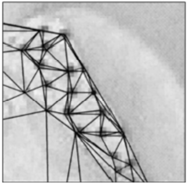
- Texture descriptor is vector of filter bank outputs.
- Textons are found by clustering.
  - Bag of words

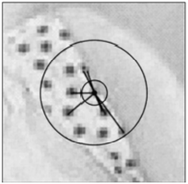

(c)


Slide Credit: *Svetlana Lazebnik*

Using Texture Features for Segmentation

- Texture descriptor is vector of filter bank outputs.
- Textons are found by clustering.
  - Bag of words
- Affinities are given by similarities of texton histograms over windows given by the "local scale" of the texture.





Slide Credit: *Svetlana Lazebnik*

Results with Color and Texture

Summary: Normalized Cuts

- Pros:
  - Generic framework, flexible to choice of function that computes weights ("affinities") between nodes
  - Does not require any model of the data distribution
- Cons:
  - Time and memory complexity can be high
    - Dense, highly connected graphs → many affinity computations
    - Solving eigenvalue problem for each cut
  - Preference for balanced partitions
    - If a region is uniform, NCuts will find the modes of vibration of the image dimensions

Slide Credit: *Kristen Grauman*

What we will learn today?

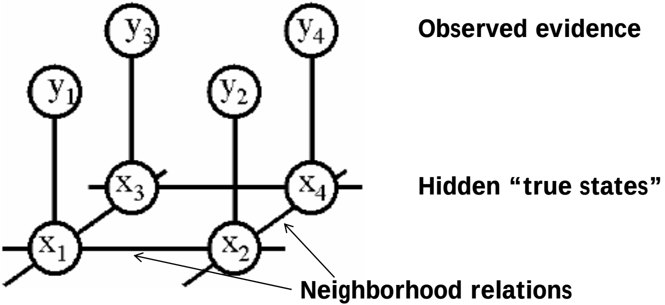
- Graph theoretic segmentation
  - Normalized Cuts
  - Using texture features
- Segmentation as Energy Minimization
  - Markov Random Fields (MRF) / Conditional Random Fields (CRF)
  - Graph cuts for image segmentation
  - Applications

What we will learn today?

- Graph theoretic segmentation
  - Normalized Cuts
  - Using texture features
- Segmentation as Energy Minimization
  - Markov Random Fields (MRF) / Conditional Random Fields (CRF)
  - Graph cuts for image segmentation
  - Applications

Markov Random Fields

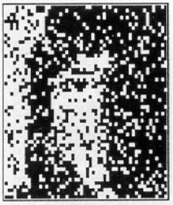
- Allow rich probabilistic models for images
- But built in a local, modular way
  - Learn/model local effects, get global effects out



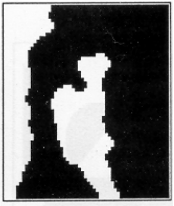
Slide Credit: *William Freeman*

MRF Nodes as Pixels

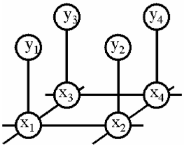
Original image



Degraded image

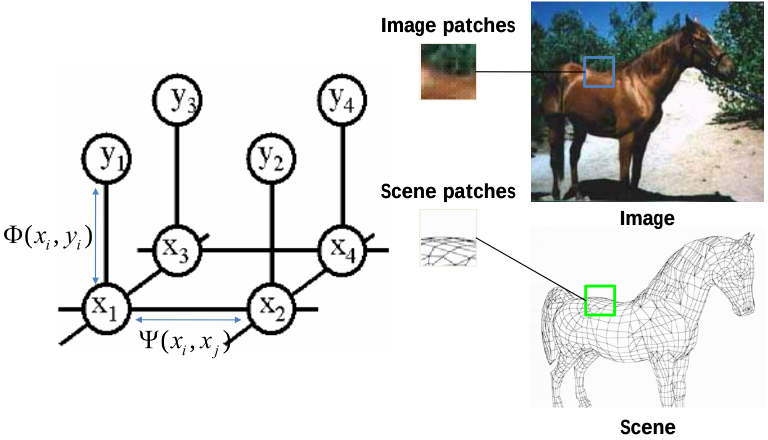


Reconstruction from MRF modeling pixel neighborhood statistics



Slide Credit: *Bastian Leibe*

MRF Nodes as Patches



Slide Credit: *William Freeman*

Network Joint Probability

$$P(x,y) = \prod_i \Phi(x_i, y_i) \prod_{i,j} \Psi(x_i, x_j)$$

Scene

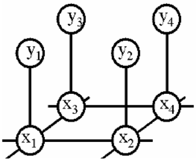
Image

Image-scene compatibility function

Local observations

Scene-scene compatibility function

Neighboring scene nodes



Slide Credit: *William Freeman*

Energy Formulation

- Joint probability

$$P(x, y) = \frac{1}{Z} \prod_i \Phi(x_i, y_i) \prod_{ij} \Psi(x_i, x_j)$$

- Taking the log turns this into an Energy optimization

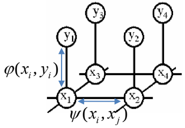
$$E(x, y) = \sum_i \varphi(x_i, y_i) + \sum_{ij} \psi(x_i, x_j)$$

- This is similar to free-energy problems in statistical mechanics (spin glass theory). We therefore draw the analogy and call E an energy function.
- φ and ψ are called potentials.

Energy Formulation

- Energy function

$$E(x,y) = \sum_i \underbrace{\varphi(x_i, y_i)}_{\text{unary term}} + \sum_{ij} \underbrace{\psi(x_i, x_j)}_{\text{pairwise term}}$$



- Unary potential φ
  - Encode local information about the given pixel/patch
  - How likely is a pixel/patch to belong to a certain class (e.g. foreground/background)?
- Pairwise potential ψ
  - Encode neighborhood information
  - How different is a pixel/patch's label from that of its neighbor? (e.g. based on intensity/color/texture difference, edges)

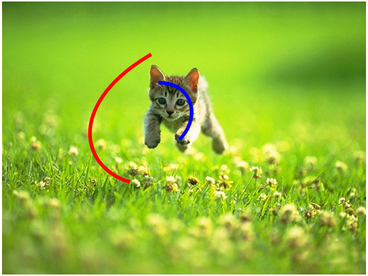
Slide Credit: *Bastian Leibe*

Segmentation using MRFs/CRFs

- Boykov and Jolly (2001)

$$E(x,y) = \sum_i \varphi(x_i, y_i) + \sum_{ij} \psi(x_i, x_j)$$

- Variables
  - x_i: Binary variable
    - foreground/background
  - y_i: Annotation
    - foreground/background/empty
- Unary term
  - $\varphi(x_i, y_i) = K[x_i \neq y_i]$
  - Pay a penalty for disregarding the annotation
- Pairwise term
  - $\psi(x_i, x_j) = [x_i \neq x_j] w_{ij}$
  - Encourage smooth annotations
  - w_{ij} affinity between pixels i and j



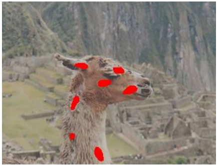
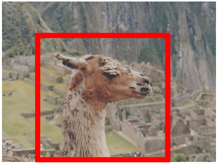
Efficient solutions

- Grid structured random fields
  - Efficient solution using Maxflow/Mincut
  - Optimal solution for binary labeling
  - Boykov & Kolmogorov, "An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision", PAMI 26(9): 1124-1137 (2004)
- Fully connected models
  - Efficient solution using convolution mean-field
  - Krähenbühl and Koltun, "Efficient Inference in Fully-Connected CRFs with Gaussian edge potentials", NIPS 2011

GrabCut: Interactive Foreground Extraction

Slides credit:
Carsten Rother

What GrabCut Does

	Magic Wand (Adobe, 2002)	**Intelligent Scissors** Mortensen and Barrett (1995)	**GrabCut**
User Input			
Result			
	Regions	Boundary	Regions & Boundary

GrabCut

- Energy function

$$E(\mathbf{x}, \mathbf{k}, \boldsymbol{\theta}|\mathbf{I}) = \sum_i \varphi(x_i, k_i, \boldsymbol{\theta}|z_i) + \sum_{ij} \psi(x_i, x_j|z_i, z_j)$$

- Variables
  - $x_i \in \{0, 1\}$: Foreground/background label
  - $k_i \in \{0, \ldots, K\}$: Gaussian mixture component
  - $\boldsymbol{\theta}$: Model parameters (GMM parameters)
  - $\mathbf{I} = \{z_1, \ldots, z_N\}$: RGB Image
- Unary term $\varphi(x_i, k_i, \boldsymbol{\theta}|z_i)$
  - Gaussian mixture model (log of a GMM)
- Pairwise term

$$\psi(x_i, x_j|z_i, z_j) = [x_i \neq x_j] \exp(-\beta\|z_i - z_j\|^2)$$

GrabCut

GrabCut

GrabCut - Unary term

- Gaussian Mixture Model

$$P(z_i|x_i,\boldsymbol{\theta}) = \sum_k \pi(x_i,k)p(z_k|k,\boldsymbol{\theta})$$

  - Hard to optimize ($\sum_k$)
- Tractable solution
  - Assign each variable x_i a single mixture component k_i

$$P(z_i|x_i,k_i,\boldsymbol{\theta}) = \pi(x_i,k_i)p(z_k|k_i,\boldsymbol{\theta})$$

  - Optimize over k_i
- Unary term

$$\begin{aligned}\varphi(x_i,k_i,\boldsymbol{\theta}|z_i) &= -\log \pi(x_i,k_i) - \log p(z_k|k_i,\boldsymbol{\theta})\\ &= -\log \pi(x_i,k_i) + \frac{1}{2}\log|\Sigma(k_i)|\\ &\quad + \frac{1}{2}(z_i-\mu(k_i))^\top \Sigma(k_i)^{-1}(z_i-\mu(k_i))\end{aligned}$$

GrabCut - Unary term

- Unary term

$$\varphi(x_i, k_i, \boldsymbol{\theta}|z_i) = -\log \pi(x_i, k_i) + \frac{1}{2}\log|\Sigma(k_i)| + \frac{1}{2}(z_i - \mu(k_i))^\top \Sigma(k_i)^{-1}(z_i - \mu(k_i))$$

- Model parameters

$$\boldsymbol{\theta} = \{ \underbrace{\pi(x_i, k_i)}_{\text{mixture weight}}, \underbrace{\mu(k_i), \Sigma(k_i)}_{\text{mean and variance}} \}$$

GrabCut - Iterative optimization

1. Initialize Mixture Models
2. Assign GMM components

$$k_i = \arg\min_k \varphi(x_i, k_i, \theta | z_i)$$

3. Learn GMM parameters

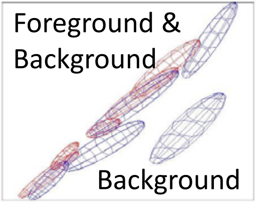
$$\theta = \arg\min \sum_i \varphi(x_i, k_i, \theta | z_i)$$

4. Estimate segmentation using mincut

$$\mathbf{x} = \arg\min E(\mathbf{x}, \mathbf{k}, \theta | \mathbf{I})$$

5. Repeat from 2 until convergence

Initialization



GrabCut - Iterative optimization

1. Initialize Mixture Models
2. Assign GMM components

$$k_i = \arg\min_k \varphi(x_i, k_i, \boldsymbol{\theta}|z_i)$$

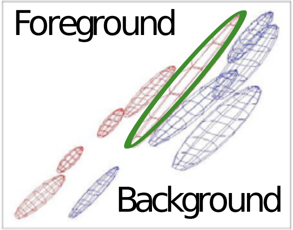
3. Learn GMM parameters

$$\boldsymbol{\theta} = \arg\min \sum_i \varphi(x_i, k_i, \boldsymbol{\theta}|z_i)$$

4. Estimate segmentation using mincut

$$\mathbf{x} = \arg\min E(\mathbf{x}, \mathbf{k}, \boldsymbol{\theta}|\mathbf{I})$$

5. Repeat from 2 until convergence



GrabCut - Iterative optimization

1. Initialize Mixture Models
2. Assign GMM components

$$k_i = \arg\min_k \varphi(x_i, k_i, \boldsymbol{\theta} | z_i)$$

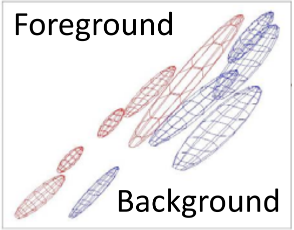
3. Learn GMM parameters

$$\boldsymbol{\theta} = \arg\min \sum_i \varphi(x_i, k_i, \boldsymbol{\theta} | z_i)$$

4. Estimate segmentation using mincut

$$\mathbf{x} = \arg\min E(\mathbf{x}, \mathbf{k}, \theta | \mathbf{I})$$

5. Repeat from 2 until convergence



GrabCut - Iterative optimization

1. Initialize Mixture Models
2. Assign GMM components

$$k_i = \arg\min_k \varphi(x_i, k_i, \boldsymbol{\theta}|z_i)$$

3. Learn GMM parameters

$$\boldsymbol{\theta} = \arg\min \sum_i \varphi(x_i, k_i, \boldsymbol{\theta}|z_i)$$

4. Estimate segmentation using mincut

$$\mathbf{x} = \arg\min E(\mathbf{x}, \mathbf{k}, \boldsymbol{\theta}|\mathbf{I})$$

5. Repeat from 2 until convergence

GrabCut - Iterative optimization

1. Initialize Mixture Models
2. Assign GMM components

$$k_i = \arg\min_k \varphi(x_i, k_i, \boldsymbol{\theta} | z_i)$$

3. Learn GMM parameters

$$\boldsymbol{\theta} = \arg\min \sum_i \varphi(x_i, k_i, \boldsymbol{\theta} | z_i)$$

4. Estimate segmentation using mincut

$$\mathbf{x} = \arg\min E(\mathbf{x}, \mathbf{k}, \boldsymbol{\theta} | \mathbf{I})$$

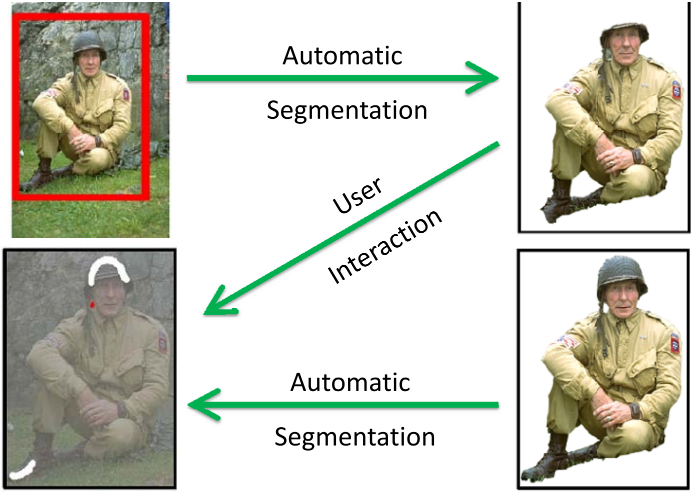
5. Repeat from 2 until convergence

GrabCut - Iterative optimization

Result

Energy after each Iteration

GrabCut - Further editing



GrabCut - More results

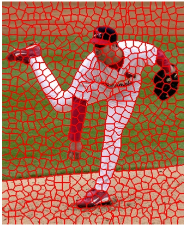
... GrabCut completes automatically

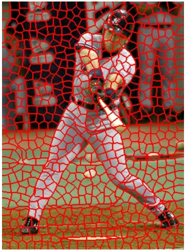
GrabCut - Live demo

- Included in MS Office 2010

Improving Efficiency of Segmentation

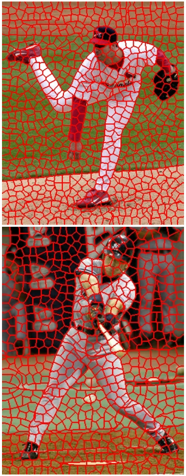
- Problem: Images contain many pixels
  - Even with efficient graph cuts, an MRF formulation has too many nodes for interactive results.
- Efficiency trick: Superpixels
  - Group together similar-looking pixels for efficiency of further processing.
  - Cheap, local oversegmentation
  - Important to ensure that superpixels do not cross boundaries
- Several different approaches possible
  - Superpixel code available here
  - http://www.cs.sfu.ca/~mori/research/superpixels/





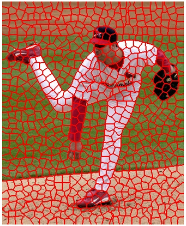
Improving Efficiency of Segmentation

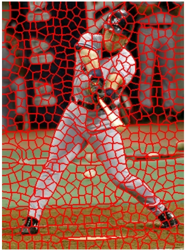
- Problem: Images contain many pixels
  - Even with efficient graph cuts, an MRF formulation has too many nodes for interactive results.
- Efficiency trick: Superpixels
  - Group together similar-looking pixels for efficiency of further processing.
  - Cheap, local oversegmentation
  - Important to ensure that superpixels do not cross boundaries
- Several different approaches possible
  - Superpixel code available here
  - `http://www.cs.sfu.ca/~mori/research/superpixels/`



Improving Efficiency of Segmentation

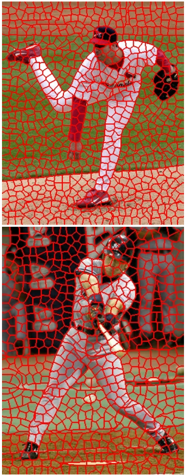
- Problem: Images contain many pixels
  - Even with efficient graph cuts, an MRF formulation has too many nodes for interactive results.
- Efficiency trick: Superpixels
  - Group together similar-looking pixels for efficiency of further processing.
  - Cheap, local oversegmentation
  - Important to ensure that superpixels do not cross boundaries
- Several different approaches possible
  - Superpixel code available here
  - http://www.cs.sfu.ca/~mori/research/superpixels/





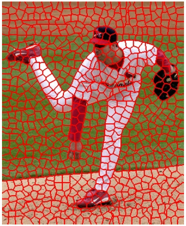
Improving Efficiency of Segmentation

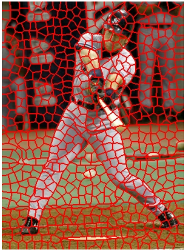
- Problem: Images contain many pixels
  - Even with efficient graph cuts, an MRF formulation has too many nodes for interactive results.
- Efficiency trick: Superpixels
  - Group together similar-looking pixels for efficiency of further processing.
  - Cheap, local oversegmentation
  - Important to ensure that superpixels do not cross boundaries
- Several different approaches possible
  - Superpixel code available here
  - http://www.cs.sfu.ca/~mori/research/superpixels/



Improving Efficiency of Segmentation

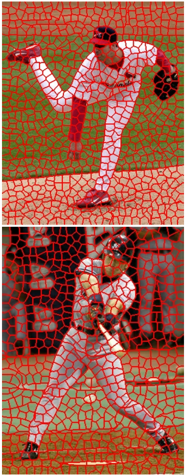
- Problem: Images contain many pixels
  - Even with efficient graph cuts, an MRF formulation has too many nodes for interactive results.
- Efficiency trick: Superpixels
  - Group together similar-looking pixels for efficiency of further processing.
  - Cheap, local oversegmentation
  - Important to ensure that superpixels do not cross boundaries
- Several different approaches possible
  - Superpixel code available here
  - http://www.cs.sfu.ca/~mori/research/superpixels/





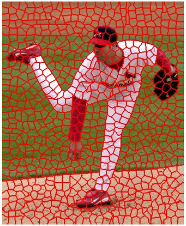
Improving Efficiency of Segmentation

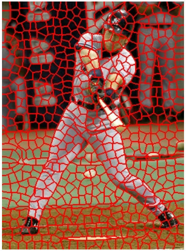
- Problem: Images contain many pixels
  - Even with efficient graph cuts, an MRF formulation has too many nodes for interactive results.
- Efficiency trick: Superpixels
  - Group together similar-looking pixels for efficiency of further processing.
  - Cheap, local oversegmentation
  - Important to ensure that superpixels do not cross boundaries
- Several different approaches possible
  - Superpixel code available here
  - http://www.cs.sfu.ca/~mori/research/superpixels/



Improving Efficiency of Segmentation

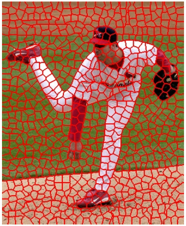
- Problem: Images contain many pixels
  - Even with efficient graph cuts, an MRF formulation has too many nodes for interactive results.
- Efficiency trick: Superpixels
  - Group together similar-looking pixels for efficiency of further processing.
  - Cheap, local oversegmentation
  - Important to ensure that superpixels do not cross boundaries
- Several different approaches possible
  - Superpixel code available here
  - `http://www.cs.sfu.ca/~mori/research/superpixels/`

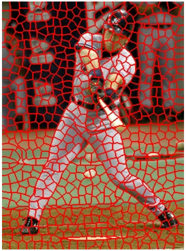




Improving Efficiency of Segmentation

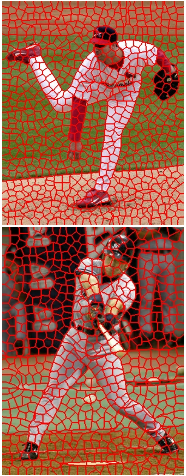
- Problem: Images contain many pixels
  - Even with efficient graph cuts, an MRF formulation has too many nodes for interactive results.
- Efficiency trick: Superpixels
  - Group together similar-looking pixels for efficiency of further processing.
  - Cheap, local oversegmentation
  - Important to ensure that superpixels do not cross boundaries
- Several different approaches possible
  - Superpixel code available here
  - http://www.cs.sfu.ca/~mori/research/superpixels/





Improving Efficiency of Segmentation

- Problem: Images contain many pixels
  - Even with efficient graph cuts, an MRF formulation has too many nodes for interactive results.
- Efficiency trick: Superpixels
  - Group together similar-looking pixels for efficiency of further processing.
  - Cheap, local oversegmentation
  - Important to ensure that superpixels do not cross boundaries
- Several different approaches possible
  - Superpixel code available here
  - `http://www.cs.sfu.ca/~mori/research/superpixels/`



Summary: Graph Cuts Segmentation

- Pros
  - Powerful technique, based on probabilistic model (MRF).
  - Applicable for a wide range of problems.
  - Very efficient algorithms available for vision problems.
  - Becoming a de-facto standard for many segmentation tasks.
- Cons/Issues
  - Graph cuts can only solve a limited class of models
    - Submodular energy functions
    - Can capture only part of the expressiveness of MRFs
  - Only approximate algorithms available for multi-label case

Summary: Graph Cuts Segmentation

- Pros
  - Powerful technique, based on probabilistic model (MRF).
  - Applicable for a wide range of problems.
  - Very efficient algorithms available for vision problems.
  - Becoming a de-facto standard for many segmentation tasks.
- Cons/Issues
  - Graph cuts can only solve a limited class of models
    - Submodular energy functions
    - Can capture only part of the expressiveness of MRFs
  - Only approximate algorithms available for multi-label case

Summary: Graph Cuts Segmentation

- Pros
  - Powerful technique, based on probabilistic model (MRF).
  - Applicable for a wide range of problems.
  - Very efficient algorithms available for vision problems.
  - Becoming a de-facto standard for many segmentation tasks.
- Cons/Issues
  - Graph cuts can only solve a limited class of models
    - Submodular energy functions
    - Can capture only part of the expressiveness of MRFs
  - Only approximate algorithms available for multi-label case

Summary: Graph Cuts Segmentation

- Pros
  - Powerful technique, based on probabilistic model (MRF).
  - Applicable for a wide range of problems.
  - Very efficient algorithms available for vision problems.
  - Becoming a de-facto standard for many segmentation tasks.
- Cons/Issues
  - Graph cuts can only solve a limited class of models
    - Submodular energy functions
    - Can capture only part of the expressiveness of MRFs
  - Only approximate algorithms available for multi-label case

Summary: Graph Cuts Segmentation

- Pros
  - Powerful technique, based on probabilistic model (MRF).
  - Applicable for a wide range of problems.
  - Very efficient algorithms available for vision problems.
  - Becoming a de-facto standard for many segmentation tasks.
- Cons/Issues
  - Graph cuts can only solve a limited class of models
    - Submodular energy functions
    - Can capture only part of the expressiveness of MRFs
  - Only approximate algorithms available for multi-label case

Summary: Graph Cuts Segmentation

- Pros
  - Powerful technique, based on probabilistic model (MRF).
  - Applicable for a wide range of problems.
  - Very efficient algorithms available for vision problems.
  - Becoming a de-facto standard for many segmentation tasks.
- Cons/Issues
  - Graph cuts can only solve a limited class of models
    - Submodular energy functions
    - Can capture only part of the expressiveness of MRFs
  - Only approximate algorithms available for multi-label case

Summary: Graph Cuts Segmentation

- Pros
  - Powerful technique, based on probabilistic model (MRF).
  - Applicable for a wide range of problems.
  - Very efficient algorithms available for vision problems.
  - Becoming a de-facto standard for many segmentation tasks.
- Cons/Issues
  - Graph cuts can only solve a limited class of models
    - Submodular energy functions
    - Can capture only part of the expressiveness of MRFs
  - Only approximate algorithms available for multi-label case

Summary: Graph Cuts Segmentation

- Pros
  - Powerful technique, based on probabilistic model (MRF).
  - Applicable for a wide range of problems.
  - Very efficient algorithms available for vision problems.
  - Becoming a de-facto standard for many segmentation tasks.
- Cons/Issues
  - Graph cuts can only solve a limited class of models
    - Submodular energy functions
    - Can capture only part of the expressiveness of MRFs
  - Only approximate algorithms available for multi-label case

Summary: Graph Cuts Segmentation

- Pros
  - Powerful technique, based on probabilistic model (MRF).
  - Applicable for a wide range of problems.
  - Very efficient algorithms available for vision problems.
  - Becoming a de-facto standard for many segmentation tasks.
- Cons/Issues
  - Graph cuts can only solve a limited class of models
    - Submodular energy functions
    - Can capture only part of the expressiveness of MRFs
  - Only approximate algorithms available for multi-label case

Summary: Graph Cuts Segmentation

- Pros
  - Powerful technique, based on probabilistic model (MRF).
  - Applicable for a wide range of problems.
  - Very efficient algorithms available for vision problems.
  - Becoming a de-facto standard for many segmentation tasks.
- Cons/Issues
  - Graph cuts can only solve a limited class of models
    - Submodular energy functions
    - Can capture only part of the expressiveness of MRFs
  - Only approximate algorithms available for multi-label case

What we will learn today?

- Graph theoretic segmentation
  - Normalized Cuts
  - Using texture features
- Segmentation as Energy Minimization
  - Markov Random Fields (MRF) / Conditional Random Fields (CRF)
  - Graph cuts for image segmentation
  - Applications