ECE5554 – Computer Vision
Lecture 8a – Graph-Based Segmentation

Creed Jones, PhD

Course update

- HW4 is due <u>on Wednesday</u>
  - August 3 at 11:59 PM!
- Quiz 4 is tomorrow
  - Covers lectures 7 and 8
- SPOT surveys on this course will open soon
  - open from August 6 through August 12
  - participation is completely anonymous and completely voluntary
  - I would appreciate your responses – especially comments that I can act on!
- Lecture 10 on Monday, August 8 will be asynchronous
  - No synchronous class session; I will be traveling
  - There will be three pre-recorded lectures, watch at your convenience
  - I will look for questions in Piazza

Final Exam will be Thursday, August 11, 8 PM to 11 PM Eastern time (NOTE UPDATE!)

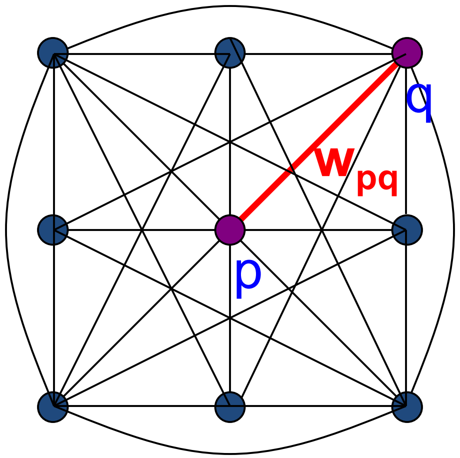
- The exam will be a collection of questions similar to the quiz questions, plus a few additional questions (may be a short calculation, a question requiring a few sentences in response, etc.)
- There will be a two-hour time limit (once you start) but I am designing the exam to require one hour or less

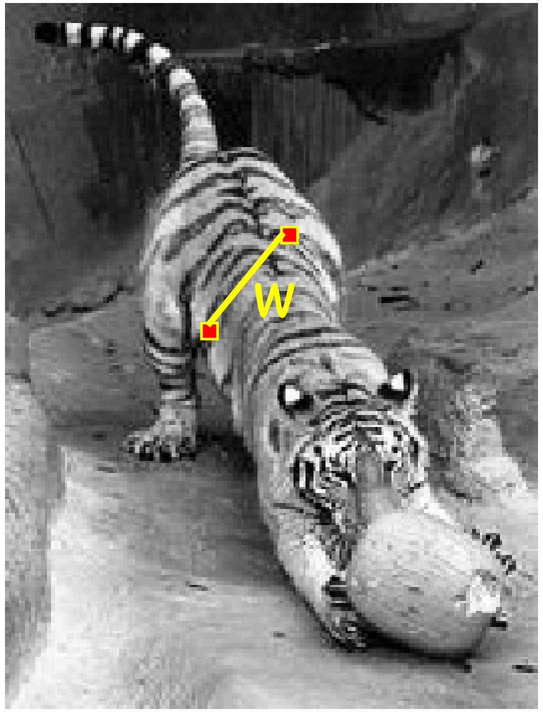
Today's Objectives

Graph-based Segmentation

- Images as graphs
- Mincuts
- Normalized cuts
  - Python example

Images as graphs





- *Fully-connected* graph
  - node (vertex) for every pixel
  - link between <u>*every*</u> pair of pixels, **p**,**q**
  - affinity weight $\mathbf{w_{pq}}$ for each link (edge)
    - $\mathbf{w_{pq}}$ measures *similarity*
      - similarity is *inversely proportional* to difference (in color and position…)

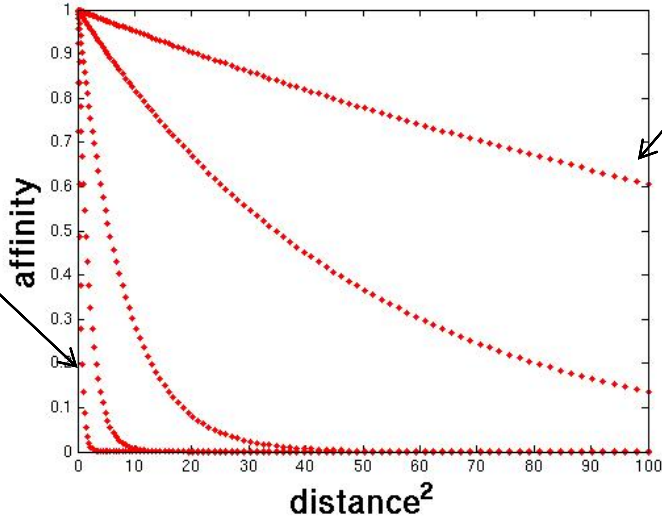
Source: Steve Seitz

Measuring affinity

- One possibility:

$$aff(x,y) = \exp\left\{-\left(\frac{1}{2\sigma_d^2}\right)\left(\|x-y\|^2\right)\right\}$$

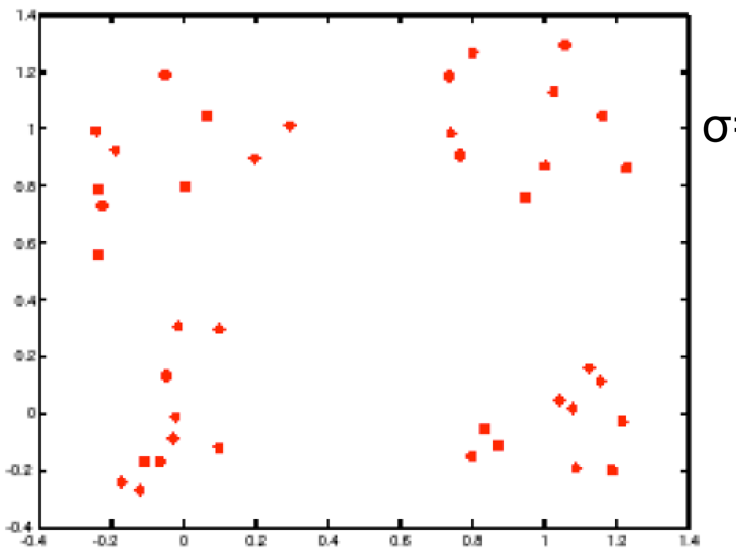
Small sigma: group only nearby points



Large sigma: group distant points

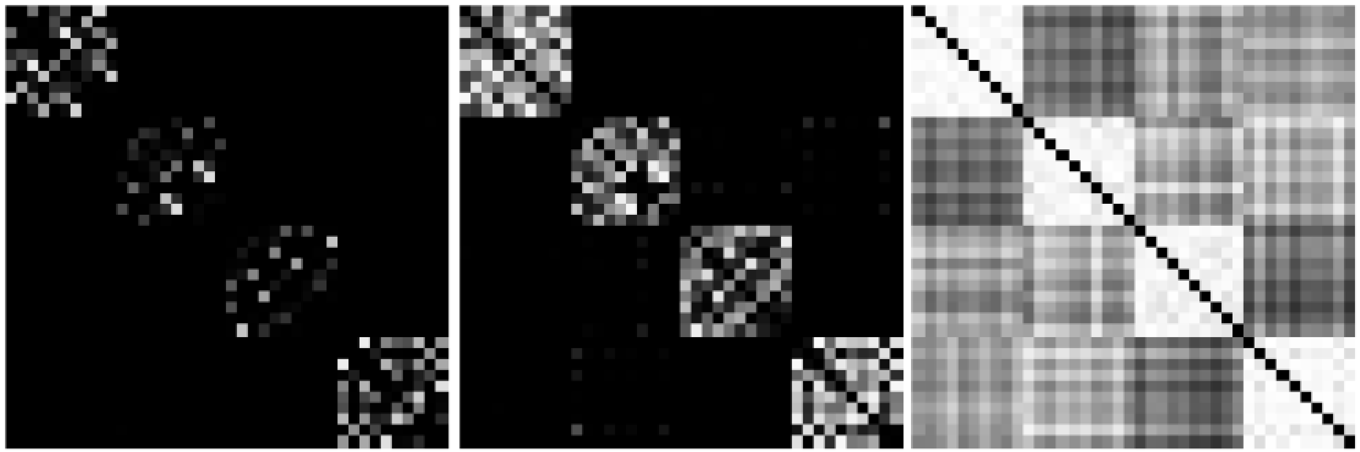
Measuring affinity

Data points



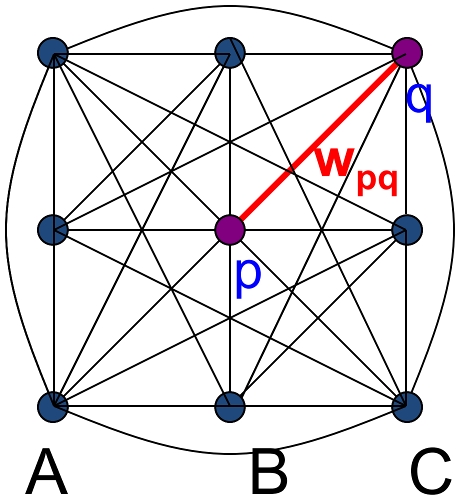
σ=.2

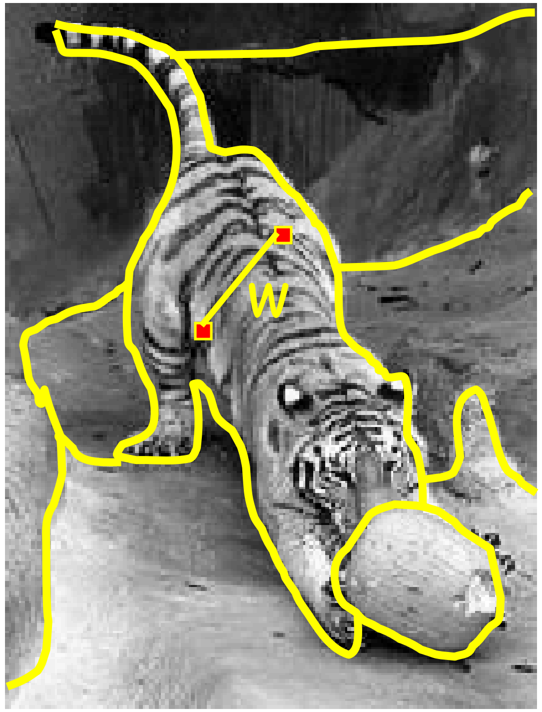
Affinity matrices



σ=.1 σ=.2 σ=1

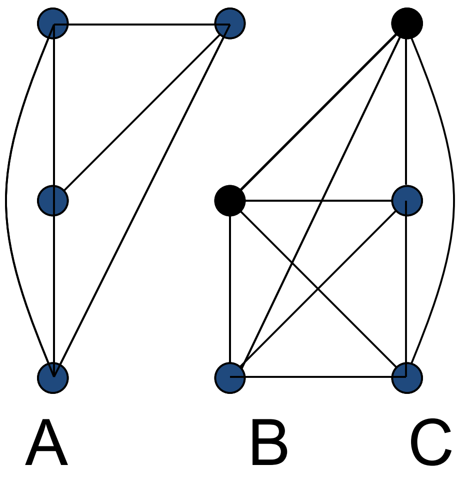
Segmentation by graph cuts

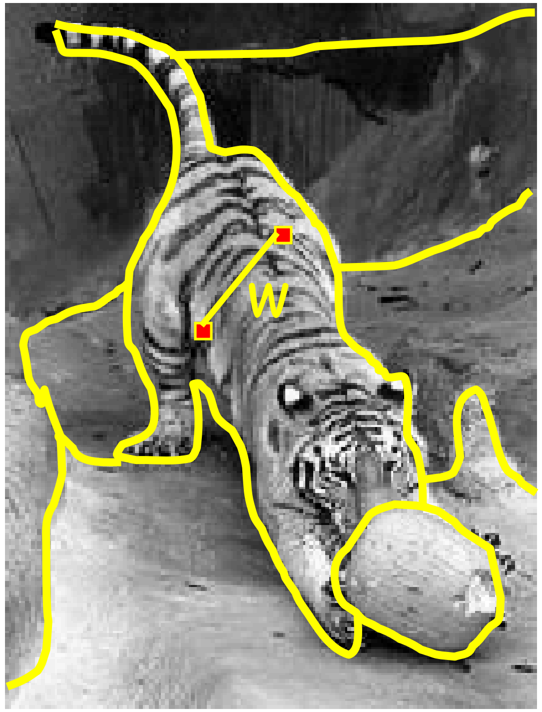




- Break Graph into Segments
  - Want to delete links that cross **between** segments

Segmentation by graph cuts

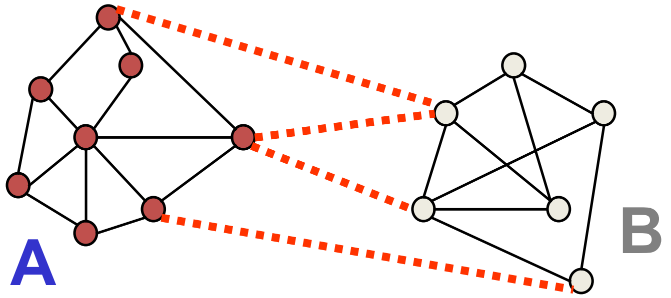




- Break Graph into Segments
  - Want to delete links that cross **between** segments
  - Easiest to break links that have low similarity (low weight)
    - similar pixels should be in the same segments
    - dissimilar pixels should be in different segments

Source: Steve Seitz

Cuts in a graph: minimum cut



- Link cut (= “cut set”)
  - set of links whose removal makes a graph disconnected
  - cost of a cut:

$$cut(A,B) = \sum_{p \in A, q \in B} w_{p,q}$$

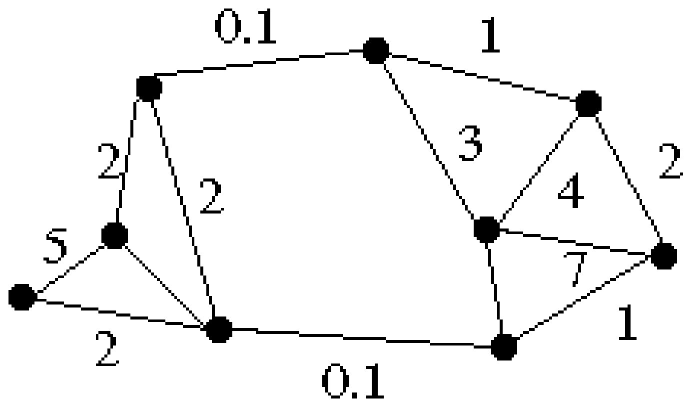
Find minimum cut

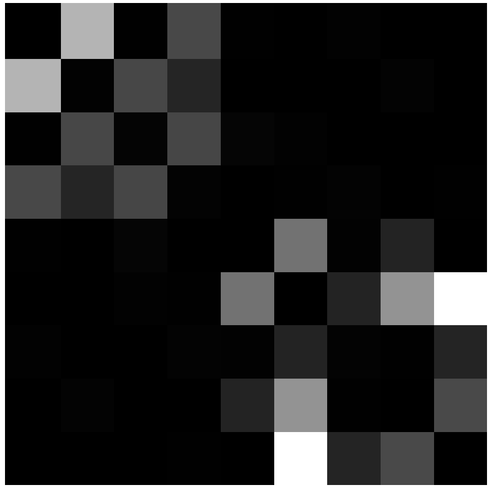
- gives you a segmentation
- fast algorithms exist for doing this

Minimum cut

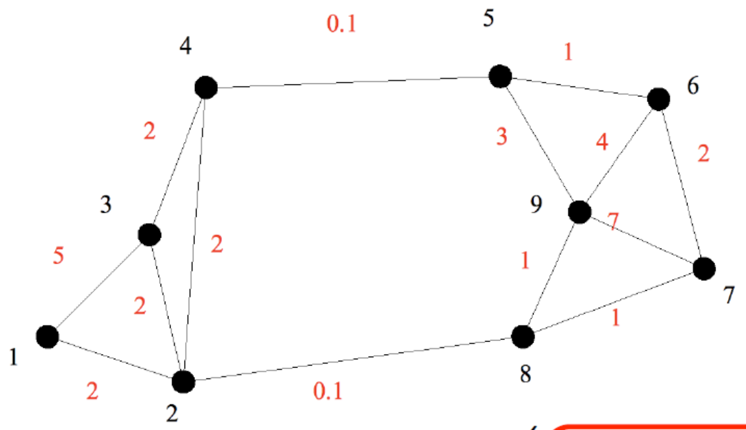
- We can do segmentation by finding the *minimum cut* in a graph

Minimum cut example





Affinity Matrix



$$M = \begin{pmatrix} 0 & 2 & 5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 2 & 2 & 0 & 0 & 0 & 0.1 & 0 \\ 5 & 2 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 2 & 0 & 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.1 & 0 & 1 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 1 & 0 & 2 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 1 & 7 \\ 0 & 0.1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 3 & 4 & 7 & 1 & 0 \end{pmatrix}$$

Minimum cut

- Problem with minimum cut:

  Weight of cut is roughly proportional to number of edges in the cut; tends to produce small, isolated components.

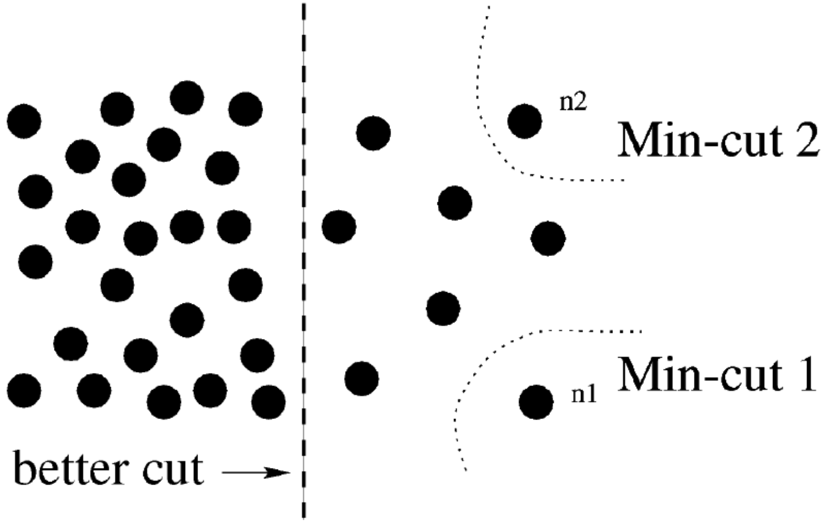
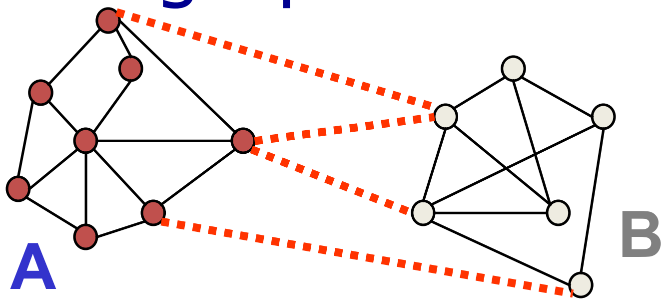


Fig. 1. A case where minimum cut gives a bad partition.

[Shi & Malik, 2000 PAMI]

Cuts in a graph: normalized cut



Normalized cut

- fix bias of Min Cut by **normalizing** for size of segments:

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

assoc(A,V) = sum of weights of all edges that touch A

- Ncut value small when we get two clusters with many edges with high weights, and few edges of low weight between them
- Approximate solution for minimizing the Ncut value: generalized eigenvalue problem

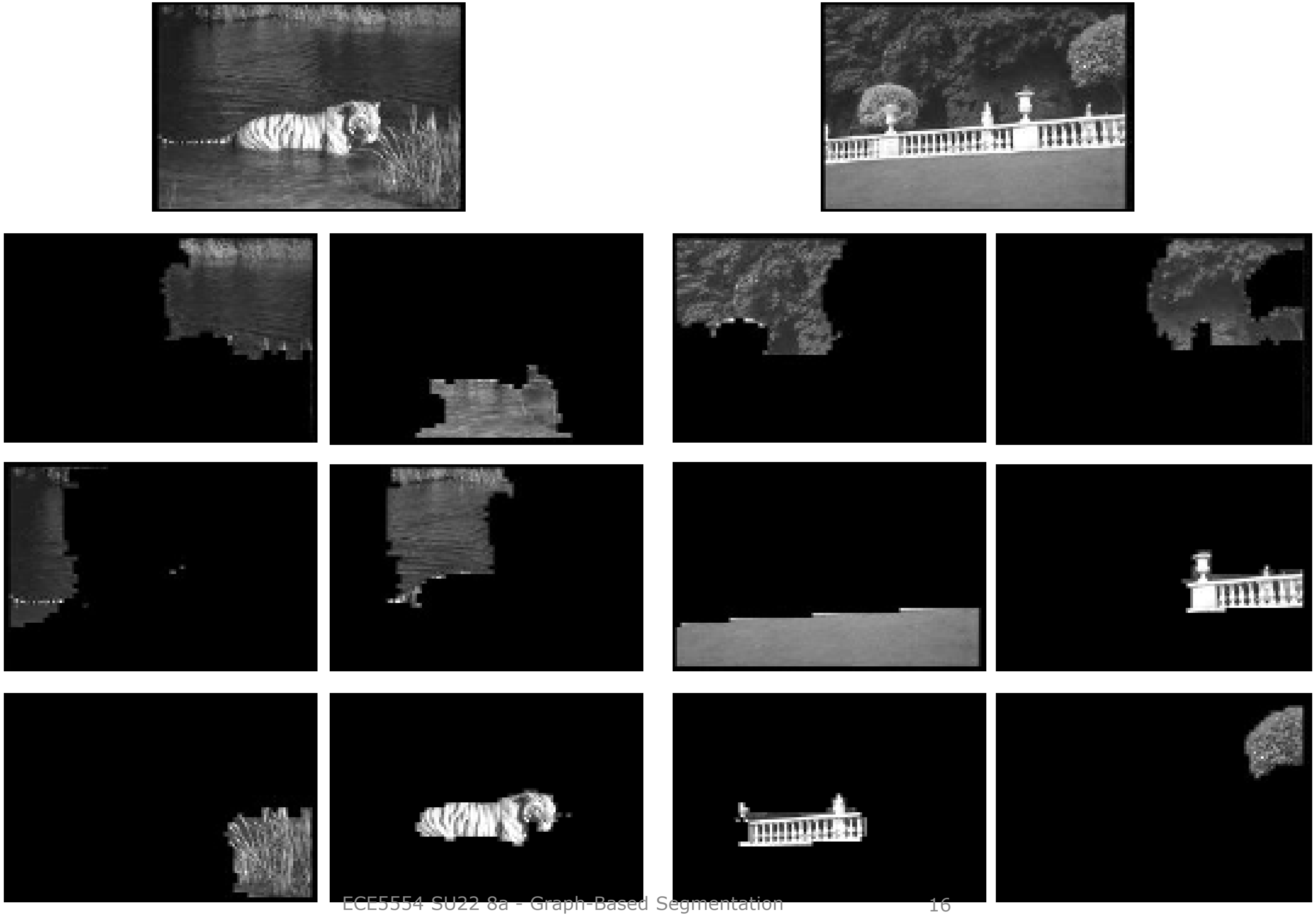
Shi and Malik, [Normalized Cuts and Image Segmentation](), CVPR, 1997

Source: Steve Seitz

Normalized cut: Algorithm

- Let **W** be the affinity matrix of the graph (n x n for n pixels)
- Let **D** be the diagonal matrix with entries $\mathbf{D}(i, i) = \Sigma_j \mathbf{W}(i, j)$
- Solve *generalized eigenvalue problem* $(\mathbf{D} - \mathbf{W})\mathbf{y} = \lambda \mathbf{D}\mathbf{y}$ for the eigenvector with the second smallest eigenvalue
  - The ith entry of **y** can be viewed as a “soft” indicator of the component membership of the ith pixel
  - Use 0 or median value of the entries of **y** to split the graph into two components
  - To find more than two components:
    - Recursively bipartition the graph
    - Run k-means clustering on values of several eigenvectors

Example results: Normalized Cut algorithm



Example results: Normalized Cut algorithm

Results: Berkeley Segmentation Engine

https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/resources.html

Normalized cuts: pros and cons

Pros:

- Generic framework, flexible to choice of function that computes weights ("affinities") between nodes
- Does not require model of the data distribution

Cons:

- Time complexity can be high
  - Dense, highly connected graphs → many affinity computations
  - Solving eigenvalue problem
- Preference for balanced partitions

Kristen Grauman

Efficient graph-based segmentation




- Runs in time nearly linear in the number of edges
- Easy to control coarseness of segmentations
- Results can be unstable

Felzenszwalb and Huttenlocher, [Efficient Graph-Based Image Segmentation](), IJCV 2004

Efficient Graph-Based Image Segmentation

Pedro F. Felzenszwalb
Artificial Intelligence Lab, Massachusetts Institute of Technology
pff@ai.mit.edu

Daniel P. Huttenlocher
Computer Science Department, Cornell University
dph@cs.cornell.edu

Abstract

This paper addresses the problem of segmenting an image into regions. We define a predicate for measuring the evidence for a boundary between two regions using a graph-based representation of the image. We then develop an efficient segmentation algorithm based on this predicate, and show that although this algorithm makes greedy decisions it produces segmentations that satisfy global properties. We apply the algorithm to image segmentation using two different kinds of local neighborhoods in constructing the graph, and illustrate the results with both real and synthetic images. The algorithm runs in time nearly linear in the number of graph edges and is also fast in practice. An important characteristic of the method is its ability to preserve detail in low-variability image regions while ignoring detail in high-variability regions.

Keywords: image segmentation, clustering, perceptual organization, graph algorithm.

The Felzenszwalb method is implemented in OpenCV

```
import numpy as np
import cv2
from cv2.ximgproc import segmentation

# Load an image and perform graph-based segmentation on color alone
img = cv2.imread('C:\\Data\\spock.jpg')
(ROWS, COLS, PLANES) = img.shape
print("Image shape is" + str(img.shape))
segmentor = segmentation.createGraphSegmentation(sigma=0.75, k=300, min_size=50)
result = np.uint8(segmentor.processImage(img))
pcolor = cv2.applyColorMap(cv2.equalizeHist(result), cv2.COLORMAP_RAINBOW)
combined = np.concatenate((img, pcolor), 1)
cv2.imwrite('C:\\Data\\ColorSeg\\GRAPH.png', combined)
```

Parameters for the Graph Segmentor

```
segmentor = segmentation.createGraphSegmentation(sigma=0.75, k=300, min_size=50)
```

- Sigma:
  - Width of Gaussian smoothing kernel applied prior to segmentation
  - Common value is 0.5
- K:
  - boundary determination threshold factor ($thr = \frac{k}{size(component)}$)
  - Common value is 300
- min_size:
  - smallest component retained
  - Common value is 100

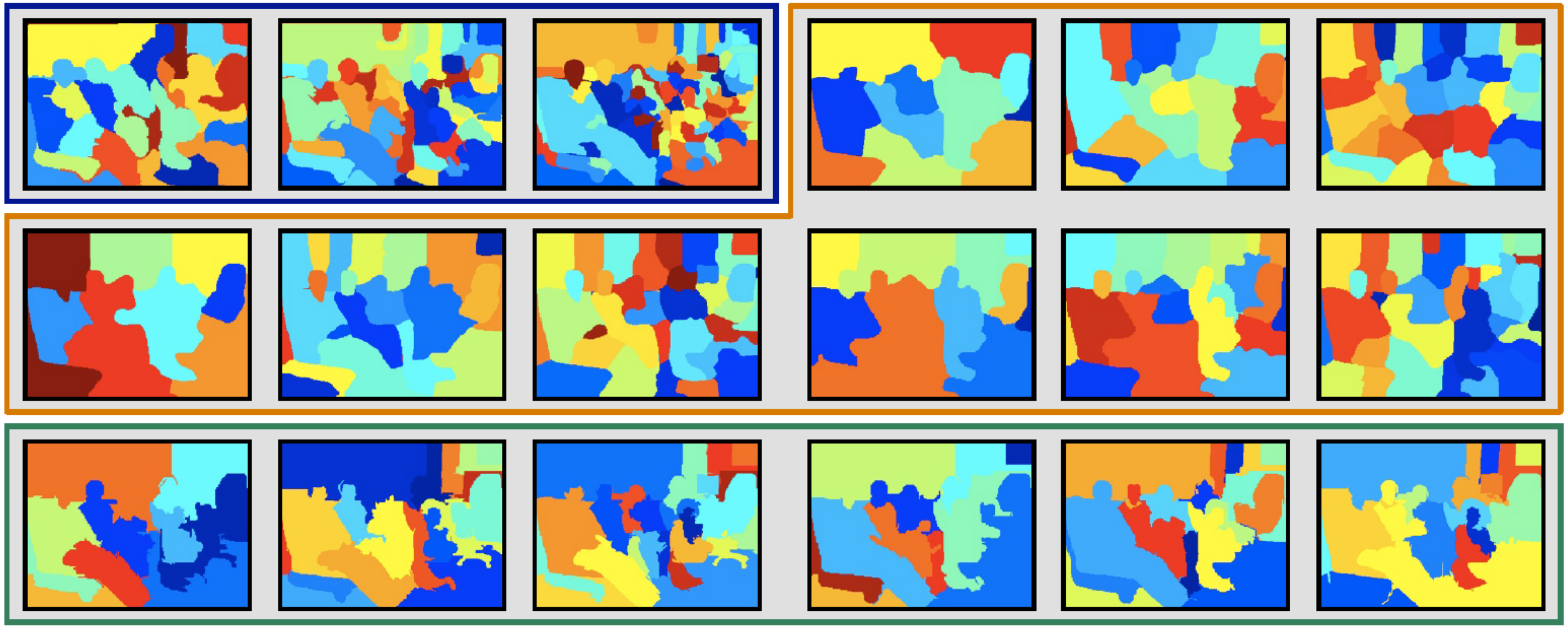
The Felzenszwalb method is implemented in OpenCV

```
import numpy as np
import cv2
from cv2.ximgproc import segmentation

# Load an image and perform graph-based segmentation on color alone
img = cv2.imread('C:\\Data\\spock.jpg')
(ROWS, COLS, PLANES) = img.shape
print("Image shape is" + str(img.shape))
segmentor = segmentation.createGraphSegmentation(sigma=0.75, k=300, min_size=50)
result = np.uint8(segmentor.processImage(img))
pcolor = cv2.applyColorMap(cv2.equalizeHist(result), cv2.COLORMAP_RAINBOW)
combined = np.concatenate((img, pcolor), 1)
cv2.imwrite('C:\\Data\\ColorSeg\\GRAPH.png', combined)
```

The Felzenszwalb method is implemented in OpenCV

```
import numpy as np
import cv2
from cv2.ximgproc import segmentation

# Load an image and perform graph-based segmentat
img = cv2.imread('C:\\Data\\spock.jpg')
(ROWS, COLS, PLANES) = img.shape
print("Image shape is" + str(img.shape))
segmentor = segmentation.createGraphSegmentation(
result = np.uint8(segmentor.processImage(img))
pcolor = cv2.applyColorMap(cv2.equalizeHist(resul
combined = np.concatenate((img, pcolor), 1)
cv2.imwrite('C:\\Data\\ColorSeg\\GRAPH.png', comb
```

Comparison: 3 Methods



- Mean Shift [Comaniciu and Meer, PAMI'02]
- Normalized cuts with boundary estimates [Shi and Malik, PAMI'00; Fowlkes *et al.*, CVPR'03]
- Graph-based segmentation [Felzenszwalb and Huttenlocher, IJCV'04]

Today's Objectives

Graph-based Segmentation

- Images as graphs
- Mincuts
- Normalized cuts
  - Python example