Announcements

- Project 1 is out today
  - help session at the end of class

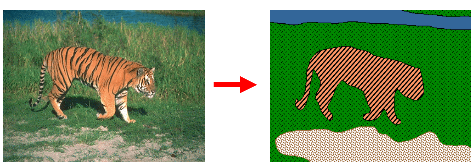
Segmentation

From Sandlot Science

Today's Readings

- Intelligent Scissors
  - http://www.cs.washington.edu/education/courses/490cv/02wi/readings/book-7-revised-a-indx.pdf

From images to objects



What Defines an Object?

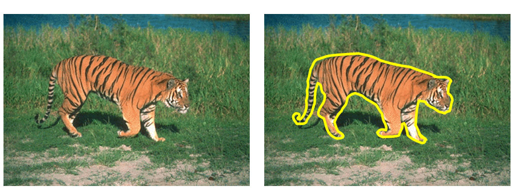
- Subjective problem, but has been well-studied
- Gestalt Laws seek to formalize this
  - proximity, similarity, continuation, closure, common fate
  - see notes by Steve Joordens, U. Toronto

Segmentation using edges

Edges ≠ Segments

- Spurious edges
- Missing edges

Extracting objects



How could this be done?

Image Segmentation

Many approaches proposed

- color cues
- region cues
- contour cues

We will consider a few of these

Today:

- Intelligent Scissors (contour-based)
  - E. N. Mortensen and W. A. Barrett, Intelligent Scissors for Image Composition, in ACM Computer Graphics (SIGGRAPH `95), pp. 191-198, 1995
- Normalized Cuts (region-based)
  - J. Shi and J. Malik, Normalized Cuts and Image Segmentation, IEEE Conf. Computer Vision and Pattern Recognition(CVPR), 1997
  - Discussed in Forsyth, chapter 16.5

Intelligent Scissors

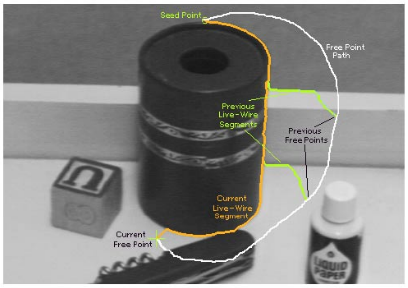


Figure 2: *Image demonstrating how the live-wire segment adapts and snaps to an object boundary as the free point moves (via cursor movement). The path of the free point is shown in white. Live-wire segments from previous free point positions (t_0, t_1, and t_2) are shown in green.*

Intelligent Scissors

Approach answers a basic question

- Q: how to find a path from seed to mouse that follows object boundary as closely as possible?
- A: define a path that stays as close as possible to edges

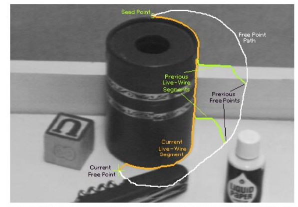


Figure 2: *Image demonstrating how the live-wire segment adapts and snaps to an object boundary as the free point moves (via cursor movement). The path of the free point is shown in white. Live-wire segments from previous free point positions (t_0, t_1, and t_2) are shown in green.*

Intelligent Scissors

Basic Idea

- Define edge score for each pixel
  - edge pixels have low cost
- Find lowest cost path from seed to mouse

11	13	12	9	5	8	3	1	2	4	10
14	11	7	4	2	5	8	4	6	3	8
11	6	3	5	7	9	12	11	10	7	4
7	4	6	11	13	18	17	14	8	5	2
6	2	7	10	15	15	21	19	8	3	5
8	3	4	7	9	13	14	15	9	5	6
11	5	2	8	3	4	5	7	2	5	9
12	4	2	1	5	6	3	2	4	8	12
10	9	7	5	9	8	5	3	7	8	15

mouse

seed

Questions

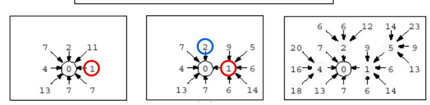
- How to define costs?
- How to find the path?

Path Search (basic idea)

Graph Search Algorithm

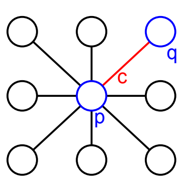
- Computes minimum cost path from seed to *all other pixels*

11	13	12	9	5	8	3	1	2	4	10
14	11	7	4	2	5	8	4	6	3	8
11	6	3	5	7	9	12	11	10	7	4
7	4	6	11	13	18	17	14	8	5	2
6	2	7	10	15	15	21	19	8	3	5
8	3	4	7	9	13	14	15	9	5	6
11	5	2	8	3	4	5	7	2	5	9
12	4	2	1	5	6	3	2	4	8	12
10	9	7	5	9	8	5	3	7	8	15



How does this really work?

Treat the image as a graph



Graph

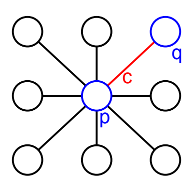
- node for every pixel **p**
- link between every adjacent pair of pixels, **p,q**
- cost **c** for each link

Note: each node has a cost

- this is a little different than the figure before where each pixel had a cost

Defining the costs

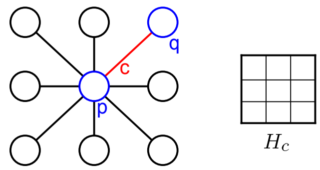
Treat the image as a graph



Want to hug image edges: how to define cost of a link?

- the link should follow the edge
- this means the gradient ∇ at p is nearly perpendicular to the link direction, d
  - option 1: c = $|\nabla \cdot d|$
  - option 2: c ≈ - |derivative of intensity across link| **for project 1**
  - other options (see reading)

Defining the costs



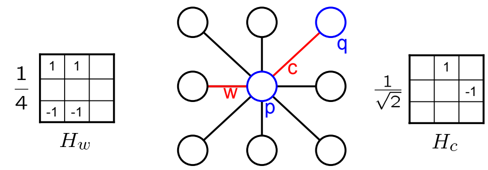
c can be computed using a cross-correlation filter

- assume it is centered at p

Also typically scale c by it's length

- set c = (max-|filter response|) * length(c)
  - where max = maximum |filter response| over all pixels in the image

Defining the costs



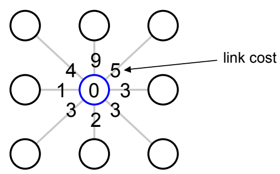
c can be computed using a cross-correlation filter

- assume it is centered at p

Also typically scale c by it's length

- set c = (max-|filter response|) * length(c)
  - where max = maximum |filter response| over all pixels in the image

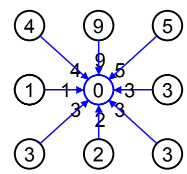
Dijkstra's shortest path algorithm



Algorithm

1. set p = seed point, cost(p) = ∞
2. expand p as follows:
   - for each of p's neighbors q that are not expanded
     - » set cost(q) = min(cost(p) + c_{pq}, cost(q))

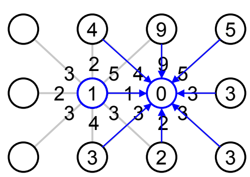
Dijkstra's shortest path algorithm



Algorithm

1. set p = seed point, cost(p) = ∞
2. expand p as follows:
   - for each of p's neighbors q that are not expanded
     - » set cost(q) = min(cost(p) + c_{pq}, cost(q))
       - » if q's cost changed, make q point back to p
     - » put q on the ACTIVE list (if not already there)

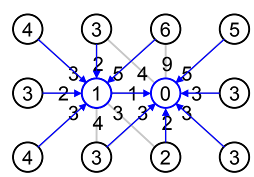
Dijkstra's shortest path algorithm



Algorithm

1. set p = seed point, cost(p) = ∞
2. expand p as follows:
   - for each of p's neighbors q that are not expanded
     - » set cost(q) = min(cost(p) + c_{pq}, cost(q))
       - » if q's cost changed, make q point back to p
     - » put q on the ACTIVE list (if not already there)
3. set r = node with minimum cost on the ACTIVE list
4. repeat Step 2 for p = r

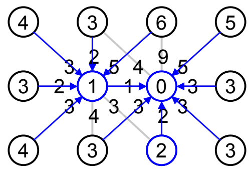
Dijkstra's shortest path algorithm



Algorithm

1. set p = seed point, cost(p) = ∞
2. expand p as follows:
   - for each of p's neighbors q that are not expanded
     - » set cost(q) = min(cost(p) + c_{pq}, cost(q))
       - » if q's cost changed, make q point back to p
     - » put q on the ACTIVE list (if not already there)
3. set r = node with minimum cost on the ACTIVE list
4. repeat Step 2 for p = r

Dijkstra's shortest path algorithm



Algorithm

1. set p = seed point, cost(p) = ∞
2. expand p as follows:
   for each of p's neighbors q that are not expanded
   » set cost(q) = min(cost(p) + c_{pq}, cost(q))
      » if q's cost changed, make q point back to p
   » put q on the ACTIVE list (if not already there)
3. set r = node with minimum cost on the ACTIVE list
4. repeat Step 2 for p = r

Dijkstra's shortest path algorithm

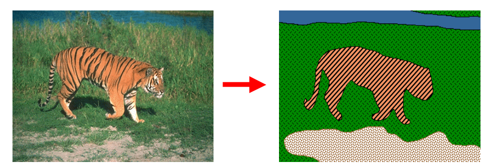
Properties

- It computes the minimum cost path from the seed to every node in the graph. This set of minimum paths is represented as a *tree*
- Running time, with N pixels:
  - $O(N^2)$ time if you use an active list
  - O(N log N) if you use an active priority queue (heap)
  - takes < second for a typical (640x480) image
- Once this tree is computed once, we can extract the optimal path from any point to the seed in O(N/2) time.
  - it runs in real time as the mouse moves
- What happens when the user specifies a new seed?

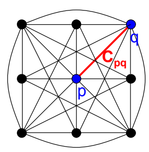
Results

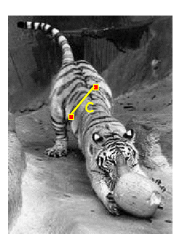
demo

How about doing this automatically?



Images as graphs

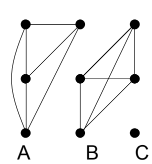


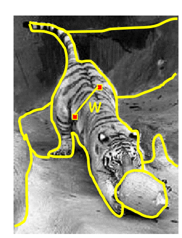


Fully-connected graph

- node for every pixel
- link between *every* pair of pixels, **p,q**
- cost c_{pq} for each link
  - c_{pq} measures *dissimilarity*
    » dissimilarity: difference in color and position
    » this is different than the costs for intelligent scissors

Segmentation by Graph Cuts

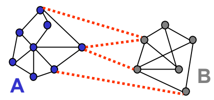




Break Graph into Segments

- Delete links that cross between segments
- Easiest to break links that have high cost
  - similar pixels should be in the same segments
  - dissimilar pixels should be in different segments

Cuts in a graph



Link Cut

- set of links whose removal makes a graph disconnected
- cost of a cut:

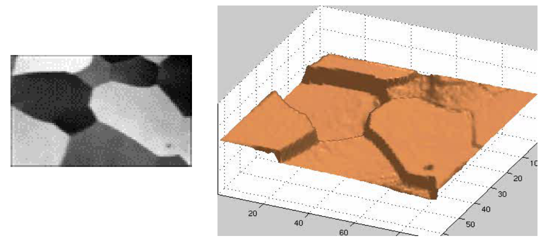
$$cut(A,B) = \sum_{p \in A, q \in B} c_{p,q}$$

Normalized Cut

- a cut penalizes large segments
- fix by normalizing for size of segments

$$Ncut(A,B) = cut(A,B)\left[\frac{1}{\sum_{p \in A} c_{p,q}} + \frac{1}{\sum_{q \in B} c_{p,q}}\right]$$

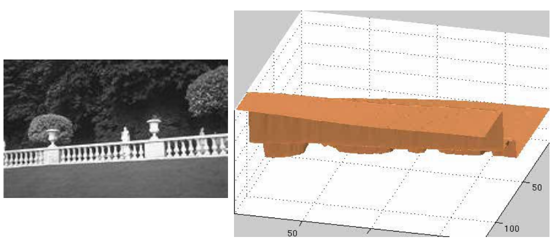
Interpretation as a Dynamical System



Treat the links as springs and shake the system

- elasticity proportional to cost
- vibration “modes” correspond to segments

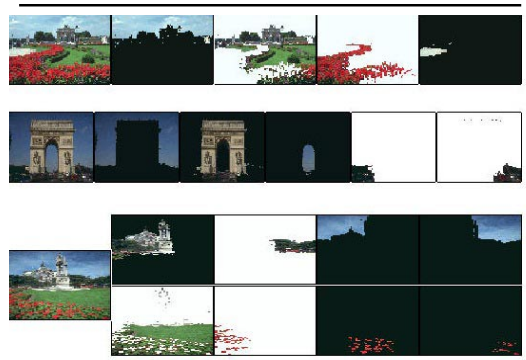
Interpretation as a Dynamical System



Treat the links as springs and shake the system

- elasticity proportional to cost
- vibration “modes” correspond to segments

Color Image Segmentation



Normalize Cut in Matrix Form

$\mathbf{W}$ is the cost matrix : $\mathbf{W}(i,j) = w_{i,j}$;

$\mathbf{D}$ is the sum of costs from node i : $\mathbf{D}(i,i) = \sum_j \mathbf{W}(i,j)$;

After lots of math, we get:

$$Ncut(A,B) = \frac{\mathbf{y}^T(\mathbf{D}-\mathbf{W})\mathbf{y}}{\mathbf{y}^T\mathbf{D}\mathbf{y}}, \quad \text{with } \mathbf{y}_i \in \{1,-b\}, \mathbf{y}^T\mathbf{D}\mathbf{1} = 0.$$

- Solution given by “generalized” eigenvalue problem:

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

- Solved by converting to standard eigenvalue problem:

$$\mathbf{D}^{-\frac{1}{2}}(\mathbf{D}-\mathbf{W})\mathbf{D}^{-\frac{1}{2}}\mathbf{z} = \lambda \mathbf{z}, \quad where\ \mathbf{z} = \mathbf{D}^{\frac{1}{2}}\mathbf{y}$$

- optimal solution corresponds to second smallest eigenvector
- for more details, see
  - J. Shi and J. Malik, Normalized Cuts and Image Segmentation, IEEE Conf. Computer Vision and Pattern Recognition(CVPR), 1997