Announcements

- Add through registration system
- Project 1 is out today
  - help session at the end of class

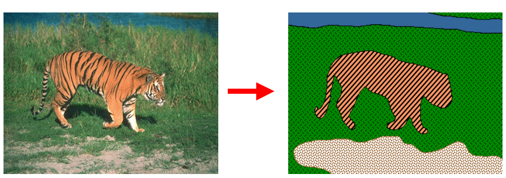
Image Segmentation

From Sandlot Science

Today's Readings

- Intelligent Scissors

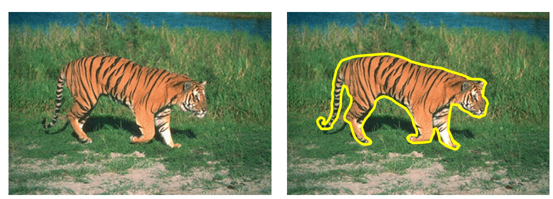
From images to objects



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

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

Intelligent Scissors

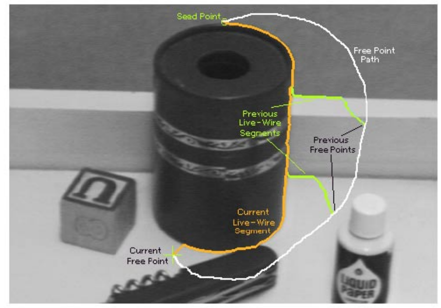


Figure 2: *Image demonstrating how the live-wire segment adapts and snaps to an object boundary as the free point moves (via cursor movement). The path of the free point is shown in white. Live-wire segments from previous free point positions (t_0, t_1, and t_2) are shown in green.*

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

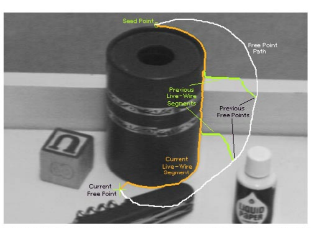
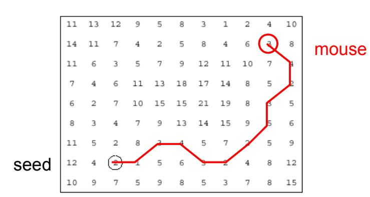


Figure 2: *Image demonstrating how the live-wire segment adapts and snaps to an object boundary as the free point moves (via cursor movement). The path of the free point is shown in white. Live-wire segments from previous free point positions (t_0, t_1, and t_2) are shown in green.*

Intelligent Scissors

Basic Idea

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



Questions

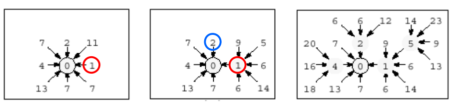
- How to define costs?
- How to find the path?

Path Search (basic idea)

Graph Search Algorithm

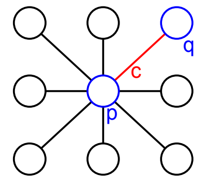
- Computes minimum cost path from seed to *all other pixels*

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

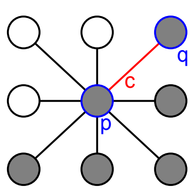
- node for every pixel **p**
- link between every adjacent pair of pixels, **p**,**q**
- cost **c** for each link

Note: each *link* has a cost

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

Defining the costs

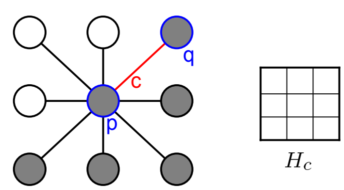
Treat the image as a graph



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

- the link should follow the intensity edge
  - want intensity to change rapidly ⊥ to the link
- c ≈ - |difference of intensity ⊥ to link|

Defining the costs



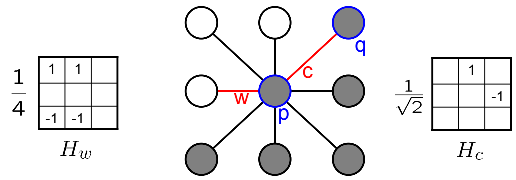
c can be computed using a cross-correlation filter

- 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



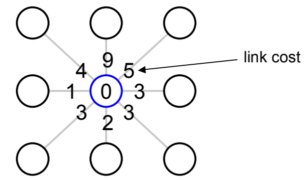
c can be computed using a cross-correlation filter

- 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



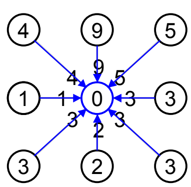
Algorithm

1. init node costs to ∞, set p = seed point, cost(p) = 0
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. init node costs to ∞, set p = seed point, cost(p) = 0
2. expand p as follows:

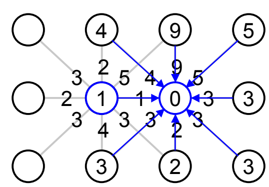
   for each of p's neighbors q that are not expanded

   » 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. init node costs to ∞, set p = seed point, cost(p) = 0
2. expand p as follows:

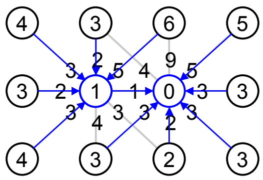
   for each of p's neighbors q that are not expanded

   » 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

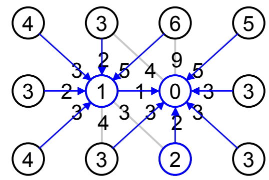
Dijkstra's shortest path algorithm



Algorithm

1. init node costs to ∞, set p = seed point, cost(p) = 0
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

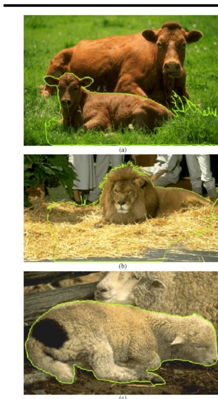
1. init node costs to ∞, set p = seed point, cost(p) = 0
2. expand p as follows:
   for each of p's neighbors q that are not expanded
   - » set cost(q) = min(cost(p) + c_{pq}, cost(q))
     - » if q's cost changed, make q point back to p
   - » put q on the ACTIVE list (if not already there)
3. set 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