Today’s Readings

- Intelligent Scissors, Morikawa et al, SIGGRAPH 1995

---

**Intelligent Scissors (demo)**

*Figure 2: Image demonstrating how the free-view cursor adapts and aligns to an object boundary as the free-view mouse (view center) moves. The path of the free point is shown in white. Lines represent segments from previous free point positions (across 1, 2, and 3) are shown in gray.*

---

**Extracting objects**

*How could this be done?*

- hard to do manually
- hard to do automatically ("image segmentation")
- easy to do semi-automatically

---

**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 3: Image demonstrating how the free-view cursor adapts and aligns to an object boundary as the free-view mouse (view center) moves. The path of the free point is shown in white. Lines represent segments from previous free point positions (across 1, 2, and 3) are shown in gray.*
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?

Let's look at this more closely

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

Want to hug image edges: how to define cost of a link?
- good (low-cost) links follow the intensity edge
  - want intensity to change rapidly $\perp$ to the link
- $c = \frac{1}{\sqrt{2}} |\text{intensity of } i - \text{intensity of } j|$
Defining the costs

$c$ can be computed using a cross-correlation filter
- Assume it is centered at $p$.

A couple more modifications
- Scale the filter response by length of link $c$. Why?
- Make $c$ positive:
  - Set $c = (\text{max} - |\text{filter response}|/\text{length})$
  - Where max = maximum $|\text{filter response}|/\text{length}$ over all pixels in the image.

Dijkstra's shortest path algorithm

Algorithm:
1. Init node costs to $\infty$, set $p$ = seed point, cost($p$) = 0
2. Expand $p$ as follows:
   - For each of $p$'s neighbors $q$ that are not expanded:
     - $\text{set cost}(q) = \text{min} (\text{cost}(p) + c_{pq}, \text{cost}(q))$
   - If $q$'s cost changed, make $q$ point back to $p$.
   - Put $q$ on the ACTIVE list (if not already there).
Algorithm

1. Init node costs to $\infty$, set $p = \text{seed point}, \text{cost}(p) = 0$
2. Expand $p$ as follows:
   - set $c_{pq} = \text{new cost}(p) + \text{cost}(p)$
   - if cost changed, make $p$ point back
   - put $q$ on the ACTIVE list (if not already there)
3. set $r = \text{node with minimum cost on the ACTIVE list}$
4. repeat Step 2 for $p = r$

Properties

1. Computes the minimum cost path from the seed to every node in the graph. This set of minimum paths is represented as a tree.
2. 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 a fraction of a second for a typical (640x480) image
3. Once this tree is computed once, we can extract the optimal path from any point to the seed in $O(N)$ time.
4. It runs in real time as the mouse moves.
5. What happens when the user specifies a new seed?