Today’s Readings

- *Intelligent Scissors*, Mortensen et al. SIGGRAPH 1995

Extracting objects

- How can this be done?
  - hard to do manually
    - By selecting each pixel on the boundary
  - hard to do automatically (“image segmentation”)
  - pretty easy to do *semi-automatically*
Image Scissors (with demo!)

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 basic question
  - Q: how to find a path from seed to mouse that follows an object boundary as closely as possible?
  - A: define a path that stays as close as possible to edges
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 intensity edges
  - want intensity to change rapidly \perp to the link
- \( c \propto \frac{1}{\sqrt{2}} \left| \text{intensity of } r - \text{intensity of } s \right| \)

\[ H_c \]

- \( c \) can (almost) be computed using a cross-correlation filter
  - assume it is centered at \( p \)
Defining the costs

\[ \frac{1}{4} \begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix} \]

\( H_w \)

\[ \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} \]

\( H_c \)

c can (almost) 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 \( \text{max} = \text{maximum} |\text{filter response}|*\text{length} \) over all pixels in the image

Dijkstra’s shortest path algorithm

Algorithm

1. init node costs to \( \infty \), set \( p = \text{seed point} \), 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 $\infty$, set $p =$ seed point, $\text{cost}(p) = 0$
2. expand $p$ as follows:
   for each of $p$’s neighbors $q$ that are not expanded
   set $\text{cost}(q) = \min(\text{cost}(p) + c_{pq}, \text{cost}(q))$
   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 $p =$ seed point, $\text{cost}(p) = 0$
2. expand $p$ as follows:
   for each of $p$’s neighbors $q$ that are not expanded
   set $\text{cost}(q) = \min(\text{cost}(p) + c_{pq}, \text{cost}(q))$
   if $q$'s cost changed, make $q$ point back to $p$
   put $q$ on the ACTIVE list (if not already there)
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 fraction of a 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) time.
    • it runs in real time as the mouse moves
  – What happens when the user specifies a new seed?
Example Results

Kuan-chuan Peng

Le Zhang