Optimization

• We can define a cost for possible solutions
• Number of solutions is large (e.g., exponential)
• Efficient search is needed
  – Global methods: cleverly find best solution without considering all.
    • These can be hard to find.
  – Local methods: pick solution, then improve it with small changes, until it can’t be improved.
    • Can almost always be done, but doesn’t always find best answer.

Intelligent Scissors
(see also Snakes)

• Goes from edges to boundaries.
• Edge is strong change in intensity.
• Boundary is boundary of an object.
  – Smooth (more or less)
  – Closed.
  – ...
  – Decision can be based on entire boundary.
For Information

- See Text by Trucco and Verri, 5.4.
- Kass, Witkin and Terzopoulos, IJCV.

Sometimes edge detectors find the boundary pretty well.
Sometimes it’s not enough.

Improve Boundary Detection

- Integrate information over distance.
- Use shape cues
  - Smoothness
  - Closure
- Get User to Help.
Humans integrate contour information.

Strategy of Class

• What is a good path?
• Given endpoints, how do we find a good path?
• What if we don’t know the end points?
Note that like all vision this is *modeling* and *optimization*. 
We'll do something easier than finding the whole boundary. Finding the best path between two boundary points.

How do we decide how good a path is? Which of two paths is better?
Discrete Grid

- Contour should be near edge.
  - Strength of gradient.
- Contour should be smooth (good continuation).
  - Low curvature
  - Low change of direction of gradient.

To start: contour near edge

- For each step from one pixel to another, we measure edge strength.
- Find path with biggest total edge strength.
So How do we find the best Path? Computer Science at last.

A Curve is a path through the grid.
Cost depends on each step of the path.
We want to minimize cost.

Map problem to Graph

Weight represents cost of going from one pixel to another. Next term in sum.
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 orthogonal to the link, i.e., high gradient.
    – $c \approx -|\text{gradient orthogonal to link}|$

Defining the costs

• First, smooth the image to reduce noise.
• $c$ can be computed using a cross-correlation filter
  – assume it is centered at $p$

• Also typically scale $c$ by it’s length
  – set $c = (\text{max}|\text{filter response}|) \times \text{length}(c)$
    • where max = maximum $|\text{filter response}|$ over all pixels in the image (Seitz)
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 $c = (\max|\text{filter response}|) \times \text{length}(c)$
    - where $\max = \max|\text{filter response}|$ over all pixels in the image

(Seitz)

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
      - set cost($q$) = $\min(\text{cost}(p) + c_{pq}, \text{cost}(q))$

(Seitz)
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)
  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 $\infty$, 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 $p = r$

\[ \text{demo} \]
Smoothness

• Discrete Curvature: if you go from p(j-1) to p(j) to p(j+1) how much did direction change?
  – Be careful with discrete distances.
• Change of direction of gradient from p(j-1) to p(j)

Combine into a cost function

• Path: p(1), p(2), … p(n).

\[ \sum_{j} d(p(j), p(j+1)) \ast [g(p(j)) + \lambda f(p(j-1), p(j))] \]

Where
• \(d(p(j), p(j+1))\) is distance between consecutive grid points (ie, 1 or sqrt(2)).
• \(g(p(j))\) measures strength of gradient
• \(\lambda\) is some parameter
• \(f\) measures smoothness, curvature.
One Example cost function

\[ \sum_{j=1}^{n} d(p(j), p(j+1)) \ast [g(p(j)) + \lambda f(p(j-1), p(j))] \]

\[ g(p(j)) = \frac{1}{l_j + \rho} \quad l_j = \frac{\|\nabla I\|}{\max\|\nabla I\|} \quad \rho \text{ is some constant} \]

\( f \) is the angle between the gradient at \( p(j-1) \) and \( p(j) \).
Or it could more directly measure curvature of the curve.


Application: 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.
Results

(Seitz Class)

demo
Graph Cuts

• Min Cut is standard CS problem;
  – find minimum cost set of edges to separate source from sink.
  – User specifies some inside pixels and outside pixels for object.
  – Min cut gives closed contour with shortest path, that separates inside and outside.
  – Many variations on this to improve performance.

GrabCut: Interactive Foreground Extraction using Iterated Graph Cuts
Lessons

• Perceptual organization, middle level knowledge, needed for boundary detection.
• Fully automatic methods not good enough yet.
• Formulate desired solution then optimize it.