Active Contours

Computer Vision

Some Slides are from Prof. Robert Collins, Penn State Univ.
Figure 5.1 Some popular image segmentation techniques: (a) active contours (Isard and Blake 1998) © 1998 Springer; (b) level sets (Cremers, Rousson, and Deriche 2007) © 2007 Springer; (c) graph-based merging (Felzenszwalb and Huttenlocher 2004b) © 2004 Springer; (d) mean shift (Comaniciu and Meer 2002) © 2002 IEEE; (e) texture and intervening contour-based normalized cuts (Malik, Belongie, Leung et al. 2001) © 2001 Springer; (f) binary MRF solved using graph cuts (Boykov and Funka-Lea 2006) © 2006 Springer.
Active Contours

- Raises level of image feature description from edges to boundaries.
- Edge is strong change in pixel intensity.
- Boundary is boundary of an object.
  - Smooth (more or less)
  - Closed
Sometimes edge detectors find the boundary pretty well.
Sometimes they don’t
Improved Boundary Detection

- Integrate information over distance.
- Use shape cues
  - Smoothness
  - Closure
- Get User to Help.
Importance of Continuity

Human perception is good at integrating contour continuity information into boundary detection process.
Active Contours

• They are also called
  – Snakes
  – Deformable Contours

• Think of a snake as an elastic band:
  – of arbitrary shape
  – sensitive to image gradient
  – that can “wiggle” in the image
  – represented as a sequential list of points
Main Idea:
1. Roughly initialize a contour by hand
2. Snake iteratively adjusts itself, attracted by high image gradients
3. Snake eventually “glues” itself to a high contrast boundary.
The Energy Functional

• Associate to each possible shape and location of the snake a value $E$.
  – Values should be s.t. the image contour to be detected has the *minimum* value.
  – $E$ is called the *energy* of the snake.

• Iteratively adjust points on the snake to achieve a smaller energy $E$
Energy Functional Design

- We need a function that given a snake state, associates to it an Energy value.
- The function should be designed so that the snake moves towards the contour that we are seeking!
Consider a contour parametrization $c = c(s)$ where $s$ is the “arc length”.

Each point $P_i$ on the contour has coordinates $(x_i(s), y_i(s))$. 
What moves the snake?

Forces applied to its points

External Forces:
  - boundaries in the image (gradients)
  - term associated with the DATA

Internal Forces:
  - continuity and curvature
  - terms associated with the CURVE

Diagram:

- Edge attraction
- Continuity
- Smoothness
Forces moving the snake (External)

- It needs to be attracted to contours:
  - Edge pixels “pull” the snake points.
  - The stronger the edge, the stronger the pull.
  - The force is proportional to $|\nabla I|$
Edgeness Term

Given a snake with $N$ points $p_1, p_2, \ldots, p_N$

Define the edgeness term of the Energy Functional:

$$E_g(p_i) = -\|\nabla I(p_i)\|$$

Magnitude of the gradient should be LARGE (which will make this term SMALL (very negative))
Forces preserving the snake (Internal)

- The snake should not break apart!
  - Points on the snake must stay close to each other
  - The farther the neighbor, the stronger the force to pull them back together
  - The force is proportional to the distance $|P_i - P_{i-1}|$
Continuity Term

Given a snake with $N$ points $p_1, p_2, \ldots, p_N$

Let $d$ be the average distance between points

Define the continuity term of the Energy Functional:

$$E_c(p_i) = (d - \| p_i - p_{i-1} \|)^2$$

Distance between points should be kept close to average
Forces preserving the snake (Internal)

The snake contour should be “smooth”
- Penalize high curvature.
- Force proportional to snake curvature

Snake contour
BAD
GOOD
Smoothness Term

Given a snake with N points $p_1, p_2, \ldots, p_N$
Curvature should be kept small

Define the smoothness term of the Energy Functional:

$$E_s(p_i) = \left\| p_{i-1} - 2p_i + p_{i+1} \right\|^2$$

Second derivative measures curvature
Snake Energy Functional

Given a snake with $N$ points $p_1, p_2, \ldots, p_N$

Define the following Energy Functional:

$$E = \sum_{i=1}^{N} a_i E_c(p_i) + b_i E_s(p_i) + c_i E_g(p_i)$$

Where:

- $E_c$ “Continuity”
- $E_s$ “Smoothness”
- $E_g$ “Edgeness”

$a_i, b_i, c_i$ are “weights” to control influence
Greedy Algorithm

Minimize energy one point at a time. For each point, consider a finite set of moves in a small window around it.

Compute the new energy for each candidate location. Move the point to the one with the minimum value.
Implementation Considerations

- To avoid numerical problems, the terms of the energy function should be normalized.
  - $E_c$ and $E_s$ are normalized by their maximum in the neighborhood.
  - $E_g$ is normalized as $|\nabla I - m| / (M - m)$
    - $M$ and $m$ are the max and min value of the gradient magnitude in the neighborhood.

That is, want all terms scaled from 0 to 1 so they are treated equally.
Implementation Considerations

Keeping high-curvature corners

• Before starting a new iteration:
  – Search for “corners”:
    • max curvature
    • large gradient
  – Corner points should not contribute to the energy (set $b_i = 0$)
Snake Algorithm

- **Input:**
  - gray scale image $I$
  - a chain of points $p_1, p_2, \ldots, p_N$
- $f$ is the fraction of points that must move to start a new iteration
- $U(p)$ is a neighborhood around $p$
- $d$ is the average distance between snake points (computed from the list of points).
Snake Algorithm

While the fraction of moved points > f

1. For i=1,2,...,N
   1. find a point in $U(p_i)$ s.t. the energy $E$ is minimum,
   2. move $p_i$ to this location
2. For i=1,2,...,N
   1. Estimate the curvature $k = |p_{i-1} - 2p_i + p_{i+1}|$
   2. Look for local max, and set $b_{\text{max}} = 0$
3. Update d

Where

$$E = \sum_{i=1}^{N} a_i E_c(p_i) + b_i E_s(p_i) + c_i E_g(p_i)$$
Issue with this Algorithm

This algorithm is not guaranteed to find the “best” curve, in the sense of lowest cost.

Why? Greedy algorithms do not explore the space of all curves
Non-Optimality

Typical snake behavior is far from optimal

Snake movement gets “hung up” on high contrast stone in wall.

There have been a *lot* of papers written about how to make snakes work robustly: different energy functions, different optimization methods…
A More Optimal Strategy

• Given a start and end point, use dynamic programming to determine “best” path from start to end location.
  – Need to determine what is a good path?
  – Need procedure to find best path

Algorithm we will discuss now is from E. N. Mortensen and W. A. Barrett, “Intelligent Scissors for Image Composition,” in ACM Computer Graphics (SIGGRAPH `95), pp. 191-198, 1995
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 these 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 (change in intensity across edge).

- Find path with biggest total edge strength.
So How do we find the best Path?

**Dynamic programming**

A Curve is a path through the grid.
Cost depends on each step of the path.
We want to minimize cost.
Incrementally determine best path, starting from end state.
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
  – \( c \approx - |\text{difference of intensity orthogonal to link}| \)
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 its length
  - set \( c = (\max - |\text{filter response}|) \times \text{length}(c) \)
    - where \( \max = \max |\text{filter response}| \) 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 its length
  - set $c = (\max - |\text{filter response}|) \times \text{length}(c)$
    - where $\max = \text{maximum } |\text{filter response}|$ over all pixels in the image
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 $\text{cost}(q) = \min(\text{cost}(p) + c_{pq}, \text{cost}(q))$
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)
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 $cost(q) = \min( cost(p) + c_{pq}, cost(q) )$
     » 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 $\infty$, 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 \( p = r \)
5. Stop when next point to expand is goal point. Read off shortest path.
Intelligent Scissors and Image Composition
Active Shape Model

Figure 8: a) Initial Position b) After 10 iterations c) At convergence of ASM search

Active Appearance Model