A Quick Review

- The movement of boundary points on an active contour can be governed by a partial differential equation (PDE)
- PDE’s operate on discrete “time steps”
  - One time step per iteration
- Snake points move normal to the curve
  - The normal direction is recalculated for each iteration.
- Snake points move a distance determined by their speed.
Typical Speed Function

- Speed is usually a combination (product or sum) of internal and external terms:
  \[ s(x,y) = s_I(x,y) \times s_E(x,y) \]

- Internal (shape) speed:
  - e.g., \( s_I(x,y) = 1 - k(x,y) \)
  - where \( k(x,y) \) measures the snake's curvature at \((x,y)\)

- External (image) speed:
  - e.g., \( s_E(x,y) = (1 + \Delta(x,y))^{-1} \)
  - where \( \Delta(x,y) \) measures the image's edginess at \((x,y)\)

- Note that \( s(x,y) \) above is always positive.
  - Such a formulation would allow a contour to grow but not to shrink.

Active Contours using PDEs: Typical Problems

- Curvature measurements are very sensitive to noise
  - They use 2nd derivatives

- They don’t allow an object to split
  - This can be a problem when tracking an object through multiple slices or multiple time frames.
  - A common problem with branching vasculature or dividing cells

- How do you keep a curve from crossing itself?
  - One solution: only allow the curve to grow
Level Sets

- A philosophical/mathematical framework:
  - Represent a curve (or surface, etc.) as an isophote in a “special” image, denoted $\psi$, variously called the:
    - Merit function
    - Embedding
    - Level-set function
  - Manipulate the curve indirectly by manipulating the level-set function.

Active Contours using PDEs on Level Sets

- The PDE active-contour framework can be augmented to use a level-set representation.
- This use of an implicit, higher-dimensional representation addresses the active-contour problems mentioned 2 slides back.
Level Sets: An Example from the ITK Software Guide

Level Sets and the Distance Transform (DT)

- DT is applied to a binary or segmented image
- Typically applied to the contour’s initialization
- Outside the initial contour, we typically negate the DT
- Records at each pixel the distance from that pixel to the nearest boundary.
- The 0-level set of the initialization’s DT is the original boundary
Level-Set Segmentation: Typical Procedure

- Create an initial contour
  - Many level-set segmentation algorithms require the initialization to be inside the desired contour

- Initialize $\psi$:
  \[
  \psi(x,y) = \begin{cases} 
  -DT(x,y) & \text{if } (x,y) \text{ is outside the contour} \\
  DT(x,y) & \text{if } (x,y) \text{ is inside the contour}
  \end{cases}
  \]

- Use a PDE to incrementally update the segmentation (by updating $\psi$)
  - Level Set Eq: $d\psi/dt = \text{velocity} \times \text{gradient}_\text{mag}(\psi)$
  - Stop at the right time
    - This can be tricky; more later.

Measuring curvature and surface normals

- One of the advantages of level sets is that they can afford good measurements of curvature
- Because the curve is represented implicitly as the 0-level set, it can be fit to $\psi$ with sub-pixel resolution
- Surface normals are collinear with the gradient of $\psi$. (why?)
- See Snyder 8.5 for details on computing curvature ($\kappa$).
Allowing objects to split or merge

- Suppose we want to segment vasculature from CT with contrast
- Many segmentation algorithms only run in 2D
  - So we need to slice the data
  - But we don’t want to initialize each slice by hand

Solution:
- Initialize 1 slice by hand
- Segment that slice
- Use the result as the initialization for neighboring slices

But vasculature branches
- One vessel on this slice might branch into 2 vessels on the next slice
- Segmentation methods that represent a boundary as a single, closed curve will break here.
Allowing objects to split or merge

- Level Sets represent a curve implicitly
- Nothing inherently prevents the 0-level set of $\psi$ from representing multiple, distinct objects.
- Most level-set segmentation algorithms naturally handle splitting or merging
  - PDEs are applied and calculated locally

Active Surfaces

- Level Sets can represent surfaces too!
- $\psi$ now fills a volume
- The surface is still implicitly defined as the zero level set.
- The PDE updates “every” point in the volume
  - (To speed up computation, on each iteration we can update only pixels that are close to the 0 level set)
- Being able to split and merge 3D surfaces over time can be very helpful!
ITK’s Traditional PDE Formulation

\[ \frac{d}{dt} \psi = -\alpha A(x) \cdot \nabla \psi - \beta P(x) |\nabla \psi| + \gamma Z(x) \kappa |\nabla \psi| \]

- **A** is an advection term
  - Draws the 0-level set toward image edginess
- **P** is a propagation (expansion or speed) term
  - The 0-level set moves slowly in areas of edginess in the original image
- **Z** is a spatial modifier term for the mean curvature \( \kappa \)
- **\( \alpha, \beta, \) and \( \gamma \)** are weighting constants
- Many algorithms don’t use all 3 terms

A Very Simple Example
(ITK Software Guide 4.3.1)

- Initialize inside the object
- Propagation:
  - Slow down near edges
  - Is always positive (growth only)
- Stop at the “right” time
  - Perform enough iterations (time steps) for the curve to grow close to the boundaries
  - Do not allow enough time for the curve to grow past the boundaries
  - This method is very fast!
A More Complex Example (ITK Software Guide 4.3.3)

- Geodesic Active Contours Segmentation
- Uses an advection term, $A$
  - Draws the curve toward edginess in the input image
  - Things no longer “blow up” if we run too long
- Now, we can simply stop when things converge (sufficiently small change from one time step to the next).
  - Still, it’s a good idea to program a maximum number of allowed time steps, in case things don’t converge.

Some General Thoughts about Level Sets

- Remember, Level Sets are nothing more than a way of representing a curve (or surface, hypersurface, etc.)
- Level- Sets do have some advantages (e.g., splitting/merging)
- But, Level- Sets otherwise work no better than any other method.
  - Look at the many examples in the ITK software guide; their results often leave a little or a lot to be desired
Level Set References

- Snyder, 8.5.2
- *Insight into Images*, ch. 8
- *ITK Software Guide*, book 2, 4.3
- “The” book:
  - Also see: [http://math.berkeley.edu/~sethian/2006/level_set.html](http://math.berkeley.edu/~sethian/2006/level_set.html)
- All of the above reference several scientific papers.

Snyder ch. 11: Parametric Transforms

- Goal: Detect geometric features in an image
- Method: Exchange the role of variables and parameters
- References: Snyder 11 & ITK Software Guide book 2, 4.4
Geometric Features?

- For now, think of geometric features as shapes that can be graphed from an equation.
- Line: \( y = mx + b \)
- Circle: \( R^2 = (x-x_{\text{center}})^2 + (y-y_{\text{center}})^2 \)

(variables are shown in bold purple, parameters are in black)

Why Detect Geometric Features?

- Guide segmentation methods
  - Automated initialization!
- Prepare data for registration methods
- Recognize anatomical structures

From the ITK Software Guide v 2.4, by Luis Ibáñez, et al., p. 596
How do we do this again?

- Actually, each edge pixel “votes”
- If we are looking for lines, each edge pixel votes for every possible line through itself:

Example: 3 collinear edge pixels:

How to Find All Possible Shapes for each Edge Pixel

- Exchange the role of variables and parameters:
- Example for a line: \( y = mx + b \)
  (variables are shown in **bold purple**)
- Each edge pixel in the image:
  - Has its own \((x, y)\) coordinates
  - Establishes its own equation of \((m, b)\)
How to Implement Voting

- With an accumulator
  - Think of it as an image in parameter space
  - Its axes are the new variables (which were formally parameters)
  - But, writing to a pixel increments (rather than overwriting) that pixel’s value.
- Graph each edge pixel’s equation on the accumulator (in parameter space)
- Maxima in the accumulator are located at the parameters that fit the shape to the image.

Example 1: Finding Lines

- If we use $y = mx + b$
- Then each edge pixel results in a line in parameter space:
  $b = -mx + y$
Example 1: Finding Lines

- A closer look at the accumulator after processing 2 and then 3 edge pixels
- The votes from each edge pixel are graphed as a line in parameter space
- Each accumulator cell is incremented each time an edge pixel votes for it
  - I.e., each time a line in parameter space passes through it

Example 2: Finding Lines… A Better Way

- What’s wrong with the previous example?
  - Consider vertical lines: \( m = \infty \)
  - My computer doesn’t like infinite-width accumulator images. Does yours?
  - For parametric transforms, we need a different line equation, one with a bounded parameter space.
Example 2: Finding Lines… A Better Way

- A better line equation for parameter voting:
  \[ \rho = x \cos \theta + y \sin \theta \]
- \( \rho \leq \) the input image diagonal size
  - But, to make math easy, \( \rho \) can be - too.
- \( \theta \) is bounded within \([0, 2\pi]\)

See Machine Vision Fig. 11.5 for example of final accumulator for 2 noisy lines

Computational Complexity

- This can be really slow
  - Each edge pixel yields a lot of computation
  - The parameter space can be huge
- Speed things up:
  - Only consider parameter combinations that make sense...
  - Each edge pixel has an apx. direction attached to its gradient, after all.
Example 3: Finding Circles

- Equation: \( R^2 = (x-x_{\text{center}})^2 + (y-y_{\text{center}})^2 \)
- Must vote for 3 parameters if \( R \) is not known!

Example 4: General Shapes

- What if our shape is weird, but we can draw it?
  - Being able to draw it implies we know how big it will be
- See Snyder 11.4 for details
- Main idea:
  - For each boundary point, record its coordinates in a local reference frame (e.g., at the shape’s center-of-gravity).
  - Itemize the list of boundary points (on our drawing) by the direction of their gradient