

## Particle Growing Method in Medical Image Segmentation

Youngmin Kim  
Hyunyoung Song

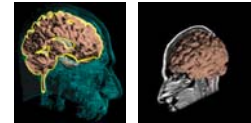
University of Maryland  
CMSC828V Spring 2005



## Medical Image Processing



- Data
  - Sampled representation (Image)
  - Acquired from medical instrumentation as CT or MRI scanners
- Registration
  - Aligns or develops correspondences between data
  - CT scan may be aligned with MRI scan to combine
- Segmentation
  - Identifies and classifies the sampled data

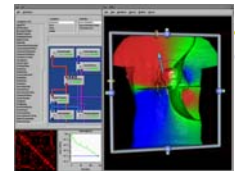


## Segmentation Issue



- Several Free Parameters
  - Control the quality of the results
- Needs for the tools that
  - Evaluates the algorithm very fast
  - Visualizes the results very fast when we change the parameters

## Previous Works

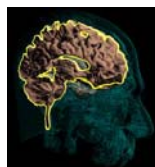


- ITK(Insight Toolkit)
  - NLM, NIH open source project
  - Leading-edge segmentation and registration algorithm
- GLUT
  - GLUT-based C++ user interface library which provides controls such as buttons, checkboxes, radio buttons, and spinners to OpenGL applications
- VTK- 3D visualization toolkit (Kitware)
- Utah
  - SCI-RUN – Problem Solving Environment for simulation, modeling, and visualization of scientific problems
  - Bio-PSE – superset of SCIRun, adding capabilities for investigating bioelectric field problems

## Latest Papers in Segmentation



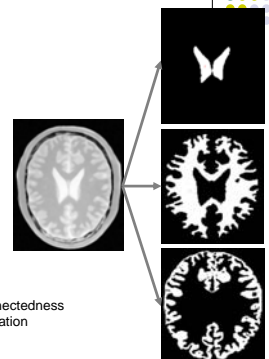
- Siggraph papers
  - Ex:Thermo-key-Image Segmentation using Thermal Methods, 2003
- IEEE Visualization 2004
  - Medical Visualization
  - Implicit Surfaces, Level Sets
    - Ex:Interactive Deformation and Visualization of Level Set Surfaces Using Graphics Hardware



## ITK Applications for Segmentation



- Region Growing
  - Connected Threshold
  - Otsu Segmentation
  - Neighborhood Connected
  - Confidence Connected
  - Isolated Connected
  - Confidence Connected in Vector Images
- Level Sets
  - Fast Marching Segmentation
  - Shape Detection Segmentation
  - Geodesic Active Contours Segmentation
  - Threshold Level Set Segmentation
  - Canny-Edge Level Set Segmentation
  - Laplacian Level Set Segmentation
- Hybrid
  - Fuzzy Connectedness and Confidence Connectedness
  - Fuzzy Connectedness and Voronoi Classification



## 1. Region Growing

- Basic segmentation filter in ITK
- Starting from boundary preserving smoothing
  - Removes noise
- Seed Selection
- Region Growing based on two thresholds
  - Upper threshold
  - Lower threshold
- Fast, but we should provide two thresholds

## Result of Region Growing

Structure	Seed Index	Lower	Upper
White matter	(60,116)	150	180
Ventricle	(81,112)	210	250
Gray matter	(107,69)	180	210

0.064 Sec  
0.058 Sec

## 2. Level Set Methods

- Level Set Function:  $\psi(\mathbf{X}, t)$
- Generic level-set equation
 
$$\frac{d}{dt}\psi = -\alpha \mathbf{A}(\mathbf{x}) \cdot \nabla \psi - \beta P(\mathbf{x}) |\nabla \psi| + \gamma Z(\mathbf{x}) \kappa |\nabla \psi|$$

Advection   Propagation   Spatial modifier for curvature K
- Track the evolution of contours and surfaces
  - By Computing the update to the solution  $\psi$  of the PDE
- Can omit one or more terms depending on the algorithm
- Typical Way in Practice
  - Contour is initialized by a user
  - Evolve until it fits the form of the segment in the image

## 2-(1) Fast Marching Method

- The simplest level set approach
- Usually used as the initial step for other level set methods
- Propagates a contour from a set of user-selected seed
- Maintains the internal pixel(or voxel) lists
- Contour advances with a speed image
  - Computed from the intensity of input image gradient magnitude
- Very fast in all the level methods, but sometimes it cannot detect the complete segmentation

## 2-(1) Fast Marching Preprocess

- Noise Minimization
  - itk::CurvatureAnisotropicDiffusionImageFilter
- Gradient of the Image
  - itk::GradientMagnitudeRecursiveGaussianImageFilter

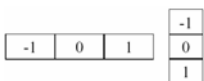


Fig1 : Calculate Gradient Magnitude

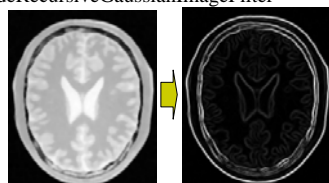


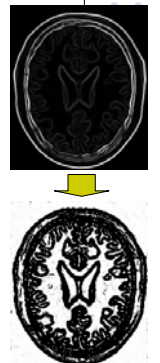
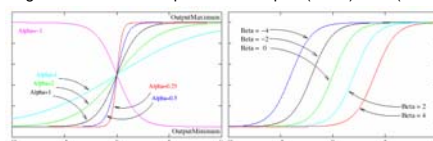
Fig2 : Original to gradient magnitude image

## 2-(1) Fast Marching Preprocess

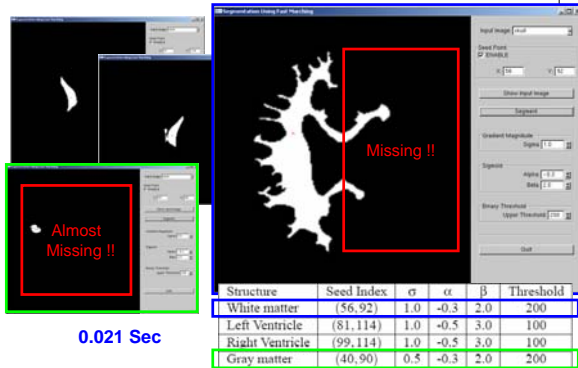
- Speed Image
  - itk::SigmoidImageFilter

$$I = (Max - Min) \frac{1}{1 + e^{-\frac{(I - \beta)}{\alpha}}} + Min$$

Fig: Effects of various parameters:alpha (width) beta (center)



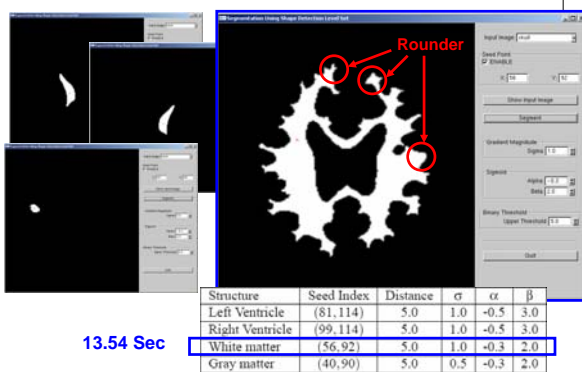
## Result of Fast Marching



## 2-(2) Shape Detection Level Set Method

- Consider **curvature term** in general level-set PDE
- Use “Fast Marching Method” as a helper in the determination of an initial level set
- Propagates a contour with a speed
- Segmented shapes are **rounder** than FastMarching due to the curvature term
- Segment better, but much more slower (about 500~600 times slower) than the fast marching method

## Result of Shape Detection Level Set



## 3. Hybrid Methods

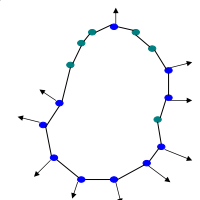
- Integrates boundary-based and region-based segmentation
  - For example:
    - Fuzzy connectedness: region-based
    - Voronoi diagram classification: boundary-based
- ITK provides two methods
  - (1) Fuzzy Connectedness and Confidence Connectedness
  - (2) Fuzzy Connectedness and Voronoi Classification
- Usually needs two steps
  - Use the first segmentation method as a prior to roughly segment or to estimate
    - Confidence Connectedness in (1)
    - Fuzzy connectedness in (2)
  - Then, use the second segmentation methods
    - Fuzzy Connectedness in (1)
    - Voronoi Diagram in (2)

## Problems in previous methods

- Region Growing
  - Needs user intervention to select thresholds
- Level Set
  - Usually slow for correct segmentation
- Hybrid
  - Needs preprocessor that produces a rough segmentation
  - Therefore, slow
- Visualization in all three methods
  - How we can extract polygons from each method for visualization is another matter

## 4. Particle Growing - Algorithm

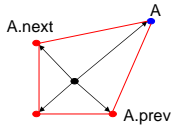
- Initialization
  - Speed Image from ITK filters
  - Generate 4 control points from seed point
- Progress
  - Pick a control pt from `TrialQueue`
  - If(`speedVal > minSpeedValue`)
    - march toward the average normal direction
    - `ProgressAmt*ProgressVector`
    - add into `TrialQueue`
  - else
    - add into `BoundaryNodeQueue`
- Control Point Insertion
  - If(`Distance btw control pts > DistanceThr`)
    - add one control pts in the middle



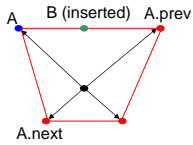
Segmented Region



**Init**  
Expand the seed to 4 ctrl pts



**Progress**  
A (98.24, 123.12)  
Speed value = 0.9834 > minSpeedValue  
Normal = (0.76, -0.64)  
Progress = 0.9834 \* (0.76, -0.64) \* ProgressAmt



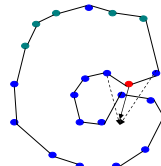
**Control Point Insertion**  
Dist(A, A.prev) > DistanceThr  
B = (A + A.prev) / 2



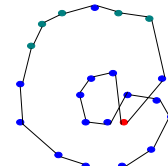
## 4. Particle Growing - Problem

- Intersection
  - Cross Intersection

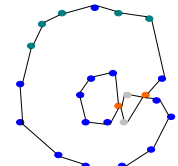
Green: Boundary Node  
Blue: Trial Node  
Red: Update Node  
Orange: Intersection Node  
Gray: Deletion Node



Before Intersection



After Intersection

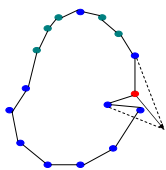


Resolve Intersection

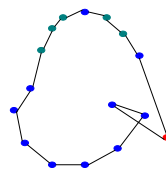
## 4. Particle Growing - Problem

- Intersection
  - Self Intersection

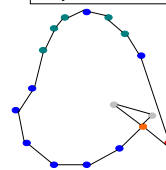
Green: Boundary Node  
Blue: Trial Node  
Red: Update Node  
Orange: Intersection Node  
Gray: Deletion Node



Before Intersection



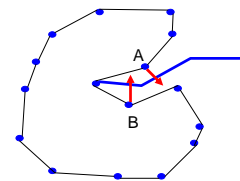
After Intersection



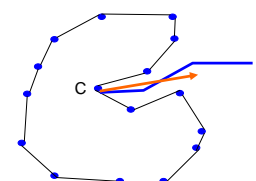
Resolve Intersection

## Resolve Intersection

- Distance Field to Prevent intersection – trial
  - Sample distance value along normal direction of progress
  - If distance value doesn't increase or decrease monotonously then intersection possibility after the progress



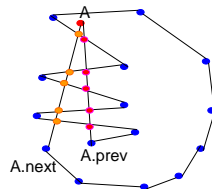
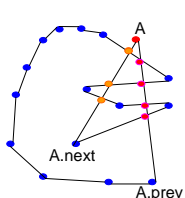
Good! Prevents A and B from potential intersection



Bad! Prevents C from marching toward although intersection doesn't occur along the path

## Heuristics to detect and remove intersection

- If (A progress to inside of segmented area) first intersection nodes to second intersection nodes; delete A;
- If (Intersection Nodes are Odd or Even) arrange links accordingly
- Delete dangling nodes, Insert intersection nodes



• A : Outside the segmented area  
• A-A.next intersection node # ODD  
• A-A.prev intersection node # EVEN

• A : Inside the segmented area  
• A-A.next intersection node # ODD  
• A-A.prev intersection node # ODD

## Performance

- Originally  $O(N^2)$  complexity for segmentation
  - Iterate the entire loop to detect intersection (per progress)
- QuadTree
  - Simple spatial partitioning
  - Intersection detected locally
  - $O(N \log N)$  complexity when there is no intersection
  - 3-5 times faster than original version
- Dealing with intersection
  - It takes  $O(N)$  time to deal with intersection in the current implementation
  - It makes the entire complexity  $O(N^2)$
  - We have found a way to process it  $O(1)$ , but it needs some more data structure (Future Work)

## Result Comparisons



Execution Time	Left Ventricle	Right Ventricle	White Matter	Gray Matter	Quality Description
Region Growing	0.031	0.031	0.064	0.058	-
Fast Marching	0.024	0.023	0.021 for half	-	White matter Malfunction
Shape Detection	1.611	1.574	13.541	-	-
Particle Growing	0.005	0.004	0.088	-	-

System: Intel Pentium 4 CPU 1.5 GHz, 2GB RAM

## Evaluation of Particle Growing



- Quality of Segmentation
  - Hard to evaluate the quality
  - Dependent on Speed Input Image
  - Dependent on the other parameters
    - Min-Speed, Distance Threshold, Progress Amount
- Computation Time
  - Very fast in segmenting convex region
  - Slow in very complex region
    - Currently, 2 times slower than Fast Marching
    - Self-intersections happen a lot
    - Not inherent in the algorithm, but due to the implementation

## Conclusion



- Present a new segmentation method using particle growing
- Use speed input image for propagating particles
- Very fast for segmenting convex regions
- Very easy to visualize the segmented parts
  - Draw the lines in 2D
  - Draw triangles in 3D (Future work)

## Future Works



- Fix some bugs in processing intersections
  - The program breaks sometimes
- Apply efficient data structure to help process self-intersection
  - Makes the entire algorithm work in  $O(\log M)$  time
- 2D to 3D extension
- Parameter adjustment and test variety of 2D and 3D images