A Work-Efficient GPU Algorithm for Level Set Segmentation

Mike Roberts
Jeff Packer
Mario Costa Sousa
Joseph Ross Mitchell
What do I mean by work-efficient?

If a parallel algorithm performs asymptotically equal work to the most efficient sequential algorithm, then the parallel algorithm is work-efficient.
What do I mean by segmentation?
What do I mean by segmentation?
What do I mean by segmentation?

Goal: Fast, interactive, and accurate segmentations even when the data is noisy and heterogeneous
Why Level Sets?

**Good:** Competitive accuracy compared to manual segmentations by experts (Cates et al. 2004)
Why Level Sets?

**Good:** Competitive accuracy compared to manual segmentations by experts (Cates et al. 2004)

**Bad:** Can be slow, even on the GPU
Why Level Sets?

**Good:** Competitive accuracy compared to manual segmentations by experts (Cates et al. 2004)

**Bad:** Can be slow, even on the GPU

*This limitation motivates our algorithm*
Segmentation with Level Sets

- Embed a seed surface in an image
- Iteratively deform the surface along normal according to local properties of the surface and the underlying image
Represent the level set surface as the zero isosurface of an implicit field.
Segmentation with Level Sets

- Deformation occurs by updating fixed elements in the implicit field
- Surface splitting and merging events are handled implicitly
- Requires many small iterations for surface to converge on a region of interest
Previous Work

• **CPU**
  - Narrow Band (Adalsteinson and Sethian 1995)
  - Sparse Field (Whitaker 1998, Peng et al. 1999)
  - Sparse Block Grid (Bridson 2003)
  - Dynamic Tubular Grid (Nielson and Museth 2006)
  - Heirarchical Run-Length-Encoded (Houston et al. 2006)
  - Above algorithms:
    - leverage spatial coherence by only processing elements near level set surface
    - require at least linear time to update the level set field

• **GPU**
  - GPU Narrow Band (Lefohn et al. 2003, 2004; Jeong et al. 2009)
  - Requires a linear number of steps to update the level set field
  - Saves memory by only storing a sparse representation of the level set field
Our Approach

*Leverage spatial and temporal coherence in the level set simulation to reduce GPU work*
Our Approach

*Leverage spatial and temporal coherence in the level set simulation to reduce GPU work*

**Contributions:**

1. Novel algorithm that limits computation by examining the temporal and spatial derivatives of the level set field
2. Work-efficient mapping to many-core GPU hardware that updates the level set field in a logarithmic number of steps
Leveraging Temporal Coherence

We only want to spend time updating the voxels that are actually changing.
Leveraging Temporal Coherence

Necessary conditions for voxels to be in the active computational domain:

1. Are we close to the surface border? (Lefohn et al. 2003)
2. Is the field neighborhood changing over time?
Leveraging Temporal Coherence

“Are we close to the surface border?”

“Are we close to the surface border?” AND “Is the field neighborhood changing over time?”

- Currently active computational domain
- Segmented region
Our Work-Efficient GPU Pipeline

1. Initialize dense list of active coordinates
2. Update level set field at active coordinates
3. Generate new active coordinates (duplicates are OK)
4. Remove duplicates
5. Compact new active coordinates into a new dense list
6. Is the new dense list empty?
   - No: Repeat from step 1
   - Yes: Segmentation Converged
Initializing a scratchpad buffer with active coordinates

- **Initial level set field**
- **Spatial gradient**
- **Initial active coordinates**
- **Write active coordinates to scratchpad buffer**
Compacting the scratchpad buffer to produce a dense list

For more details see Harris et al. 2007; Sengupta et al. 2007, 2008
Updating the level set field at active coordinates

active coordinates → old level set field → new level set field
Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)
Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)
Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)
Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)
Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)
Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)
Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)
Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)
Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)
Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)
Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)
Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)
Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)
Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)
Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)
Removing duplicate active coordinates from the auxiliary buffers
Removing duplicate active coordinates from the auxiliary buffers
Removing duplicate active coordinates from the auxiliary buffers
Removing duplicate active coordinates from the auxiliary buffers
Removing duplicate active coordinates from the auxiliary buffers
Removing duplicate active coordinates from the auxiliary buffers
Removing duplicate active coordinates from the auxiliary buffers
Compacting the auxiliary buffers to produce a new dense list of active coordinates.

$a^{\bullet}$  

$B$  

$B'^{\uparrow}$  

$B'^{\downarrow}$  

$B'$  

$B'$  

Compact  

coordinate buffer: 

0,2 0,3 1,2 1,3 2,0 2,1 2,3 3,0 3,1 3,2 3,3 4,0 4,1 4,2 4,3 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Algorithmic Complexity

• Compact (Harris et al. 2007; Sengupta et al. 2007, 2008)
  – $O(\log n)$ steps
  – $O(n)$ work

• Rest of our algorithm
  – $O(1)$ steps
  – $O(n)$ work

**Good:** Our algorithm is work-efficient and requires a logarithmic number of steps to update the level set field

**Bad:** Our algorithm requires memory proportional to the size of the level set field
Experimental Methodology

- 256x256x256 human head MRI (ground truth from expert)
- Segmented white and grey matter
- Variety of noise levels
- 10 repeated segmentations per noise level
- Nvidia GTX 280
- Measured computational domain size, speed, accuracy
- Repeated using our algorithm and the GPU narrow band algorithm (Lefohn et al. 2004)
Accuracy

SNR = Signal-to-noise Ratio
D  = Dice Coefficient
TCF = Total Correct Fraction of Labeled Voxels
Computational Domain Size

![Graph showing active computational domain size over iteration number. The graph compares 'Our Algorithm' and 'GPU Narrow Band' with 'Active Computational Domain (%)' on the y-axis and 'Iteration Number' on the x-axis. The graph includes a bar chart showing total processed voxels (millions) with values 4877 and 294.]
Speed

The graph shows the computation time in milliseconds for different algorithms across iteration numbers. The x-axis represents the iteration number, and the y-axis represents the computation time in milliseconds.

Legend:
- Blue dots: Our Algorithm
- Red dots: GPU Narrow Band
- Green dots: Unconditional Update

Inset bar chart shows:
- Total Computation Time (seconds):
  - Our Algorithm: 7 seconds
  - GPU Narrow Band: 102 seconds
  - Unconditional Update: 128 seconds
Limitations

• Requires a large amount of GPU memory
  – About 500 MB for a 256x256x256 data set

• Scaling to high order neighborhoods increases memory requirements
  – Need extra auxiliary buffers
  – Increases redundant work per thread
Future Work

• Reduce the memory requirements
  – Implement sparse representation of the level set field and other buffers (i.e. hierarchical run-length-encoded level sets) on the GPU

• Applicable to other level set problems in computer graphics?
  – Fluid simulation, surface reconstruction, image restoration, etc

• Are there other applications for the duplicate removal algorithm?
Questions?
Bonus Slides
Speed Function

Speed function proposed by Lefohn et al. 2003, 2004

\[ \alpha \left( \text{data term} \right) + (1 - \alpha) \left( \text{curvature term} \right) \]

\( \alpha \) controls the smoothness of the segmentation
The curvature term enforces a smooth segmentation and prevents leaking
Temporally Coherent Algorithm

1. Initialize level set field and active computational domain
2. Update level set field at active voxels
3. Voxels changing in space and time form the new active computational domain
4. Is active computational domain empty?
   - Yes: Segmentation converged
   - No: Go back to step 2
Initializing the level set field and the active computational domain
Updating the level set field at active voxels
Finding the voxels that are changing in space

- Current level set field
- Spatial gradient
- Voxels changing in space
Finding the voxels that are changing in time

current level set field

previous level set field

subtract

voxels changing in time

dilate

neighbors of voxels changing in time
Finding the voxels that are changing in
space and time

neighbors of voxels changing in time

voxels changing in space

intersection

voxels changing in space and time

new active set
Speed vs. Computational Domain Size

![Graph showing computation time vs. active computational domain size. The graph compares different algorithms, including Our Algorithm, GPU Narrow Band Algorithm, Unconditional Update (mean), Our Algorithm (linear best fit), and GPU Narrow Band (linear best fit).]
Speed per Subroutine

![Graph showing the computation time (milliseconds) for different iterations and subroutines. The graph includes a pie chart indicating the total computation time for each subroutine category.]