A Work-Efficient GPU Algorithm for Level Set Segmentation

Mike Roberts

Jeff Packer

Mario Costa Sousa

Joseph Ross Mitchell

imaging informatics .ca

UNIVERSITY OF CALGARY

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

iterations

Segmentation with Level Sets

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

Speed Function

Speed function proposed by Lefohn et al. 2003, 2004

α

$+(1-\alpha)$

data term

curvature term

α controls the smoothness of the segmentation

Speed Function

The curvature term enforces a smooth segmentation and prevents leaking

with curvature influence

without curvature influence

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

Live Demo

Temporally Coherent Algorithm

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*

Finding the voxels that are changing in *time*

Finding the voxels that are changing in *space and time*

Our Work-Efficient GPU Pipeline

Initializing a scratchpad buffer with active coordinates

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

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)

coordinate buffer

0,0	0,1	0,2	0,3	1,0	1,1	1,2	1,3	2,0	2,1	2,2	2,3	3,0	3,1	3,2	3,3

old active set

auxiliary buffers

$$\begin{cases} B^{\bullet} \\ B^{\rightarrow} \\ B^{\uparrow} \\ B^{\leftarrow} \\ B^{\downarrow} \end{cases}$$

new active set

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

Asymptotic Complexity

- Compact (Harris et al. 2007; Sengupta et al. 2007, 2008)
  - $\boldsymbol{O(\log n)}$ steps
  - $\boldsymbol{O(n)}$ work

- Rest of our algorithm
  - $\boldsymbol{O(1)}$ steps
  - $\boldsymbol{O(n)}$ 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

Speed

Our Algorithm
GPU Narrow Band
Unconditional Update
Computation Time (milliseconds)
0
10
20
30
40
50
60
0
500
1000
1500
2000
2500
Iteration Number
128
102
7
Total Computation Time (seconds)

Speed *vs.* Computational Domain Size

Speed per Subroutine

Results

Results

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?