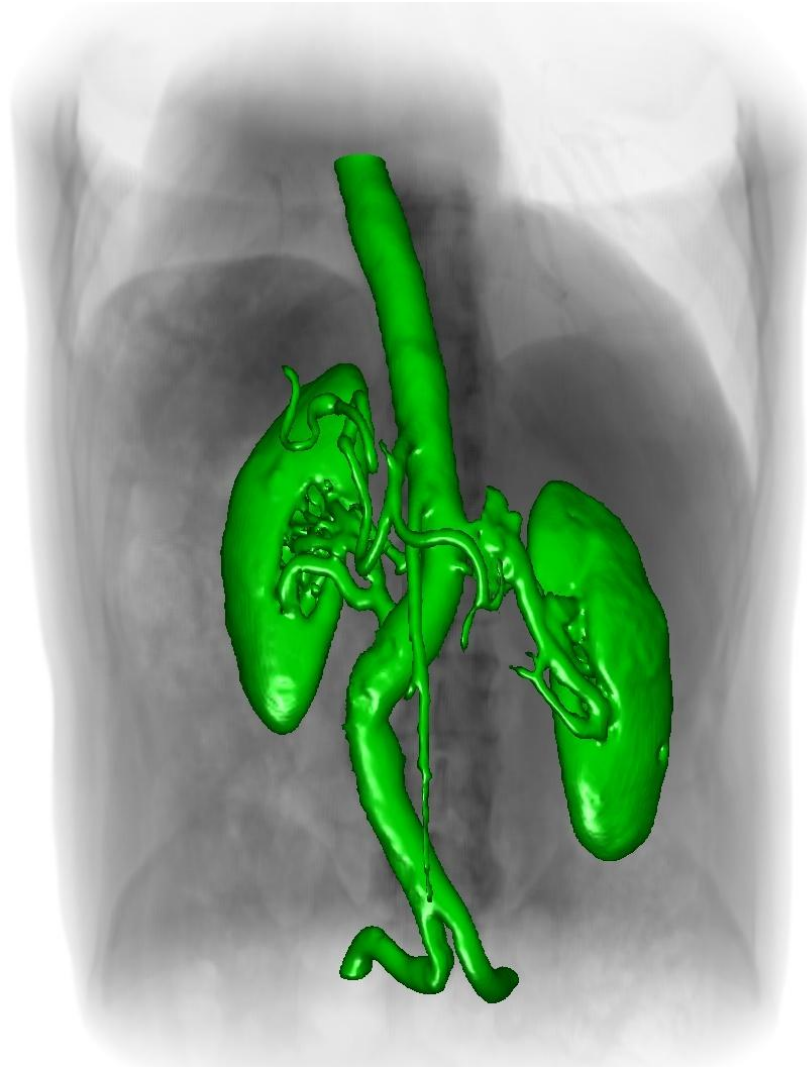


A Work-Efficient GPU Algorithm for Level Set Segmentation



Mike Roberts

Jeff Packer

Mario Costa Sousa

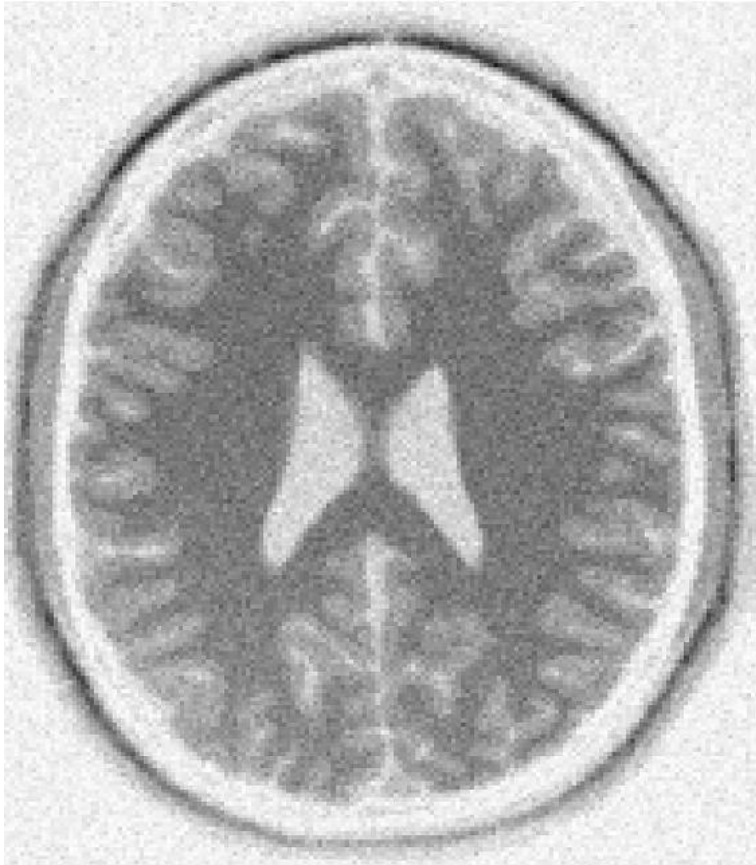
Joseph Ross Mitchell



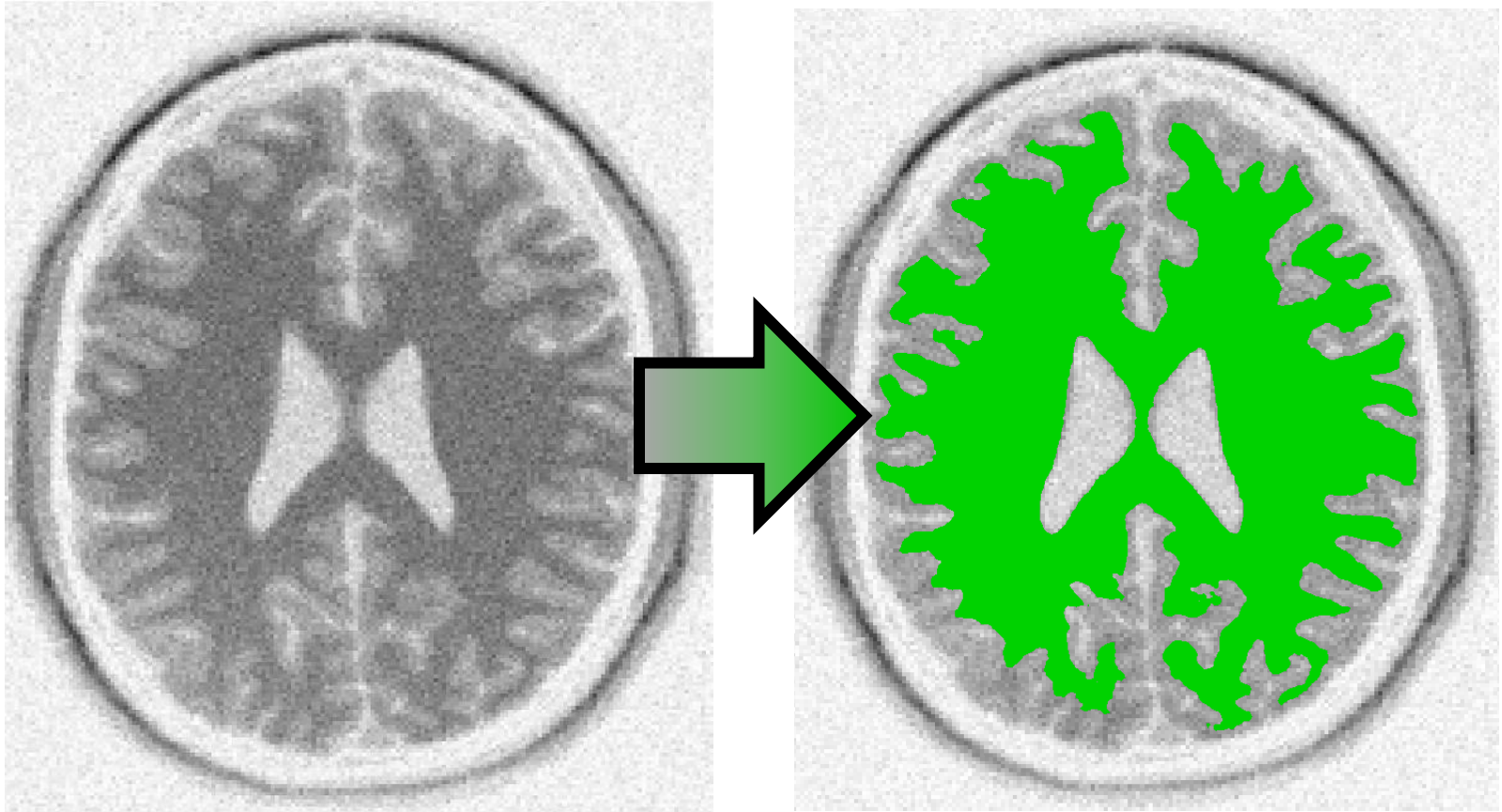
What do I mean by work-efficient?

If a parallel algorithm performs asymptotically equal *work* to the most efficient sequential algorithm, then the parallel algorithm is work-efficient.

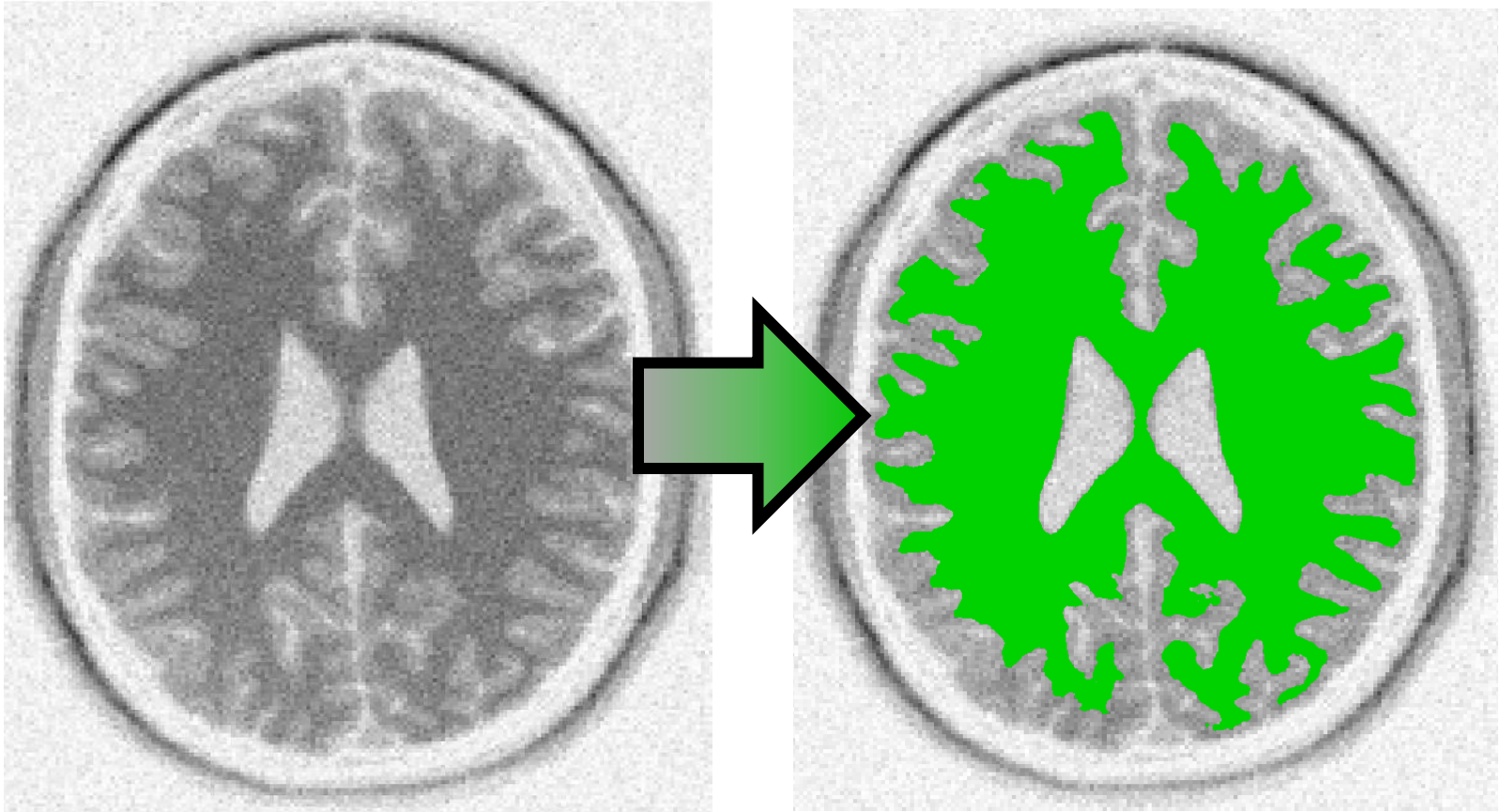
What do I mean by segmentation?



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?

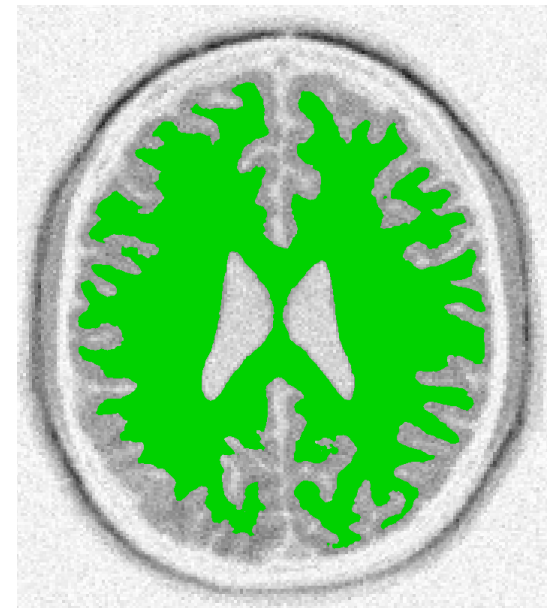
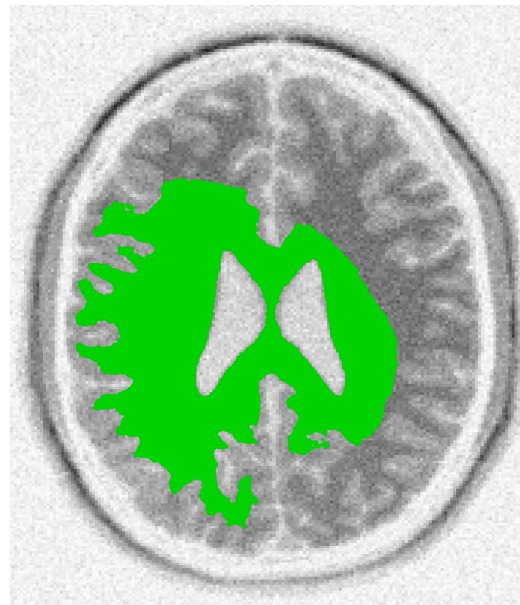
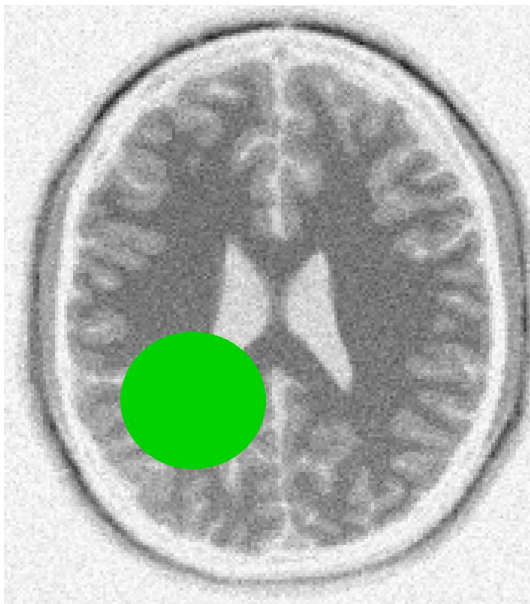
Good: Competitive accuracy compared to manual segmentations by experts (Cates et al. 2004)

Bad: Can be slow, even on the GPU

This limitation motivates our algorithm

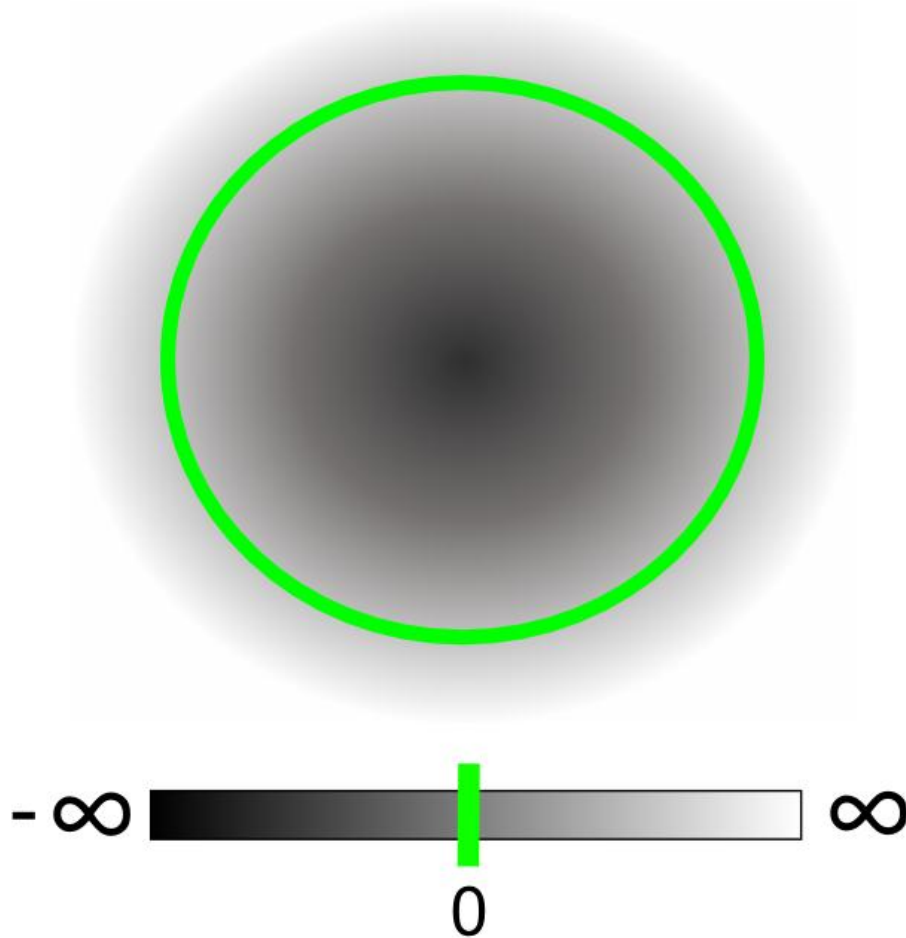
Segmentation with Level Sets

- Embed a seed surface in an image
- Iteratively deform the surface along normal according to local properties of the surface and the underlying image



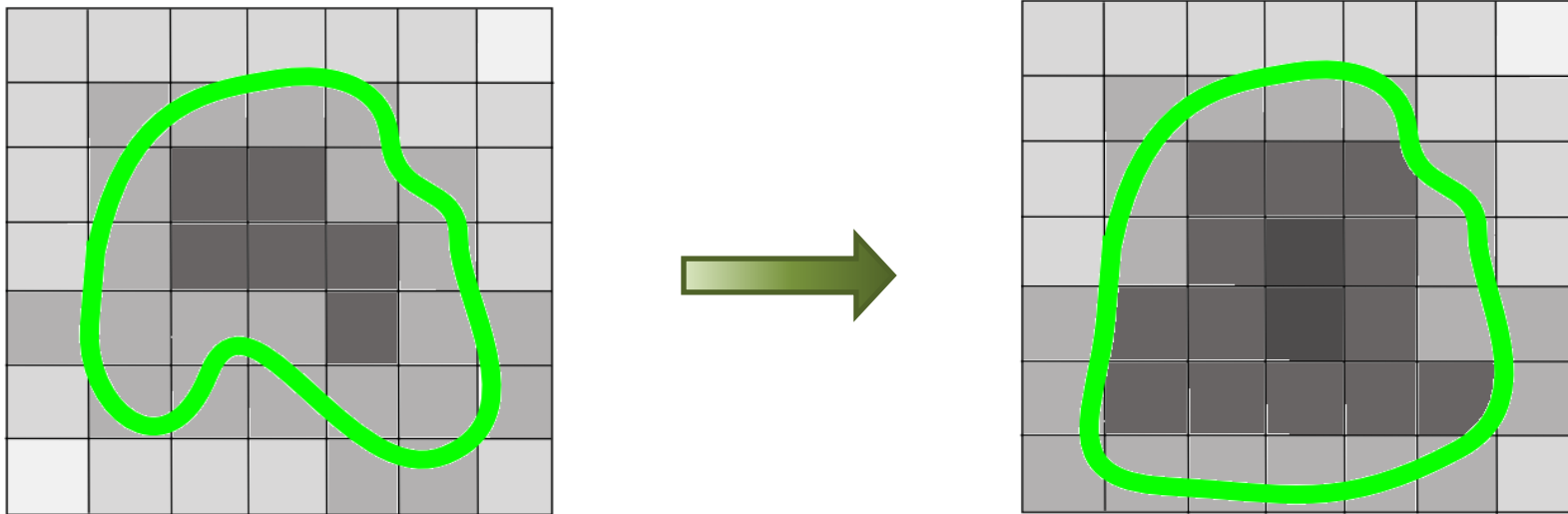
—————→ iterations

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

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)
 - Hierarchical 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

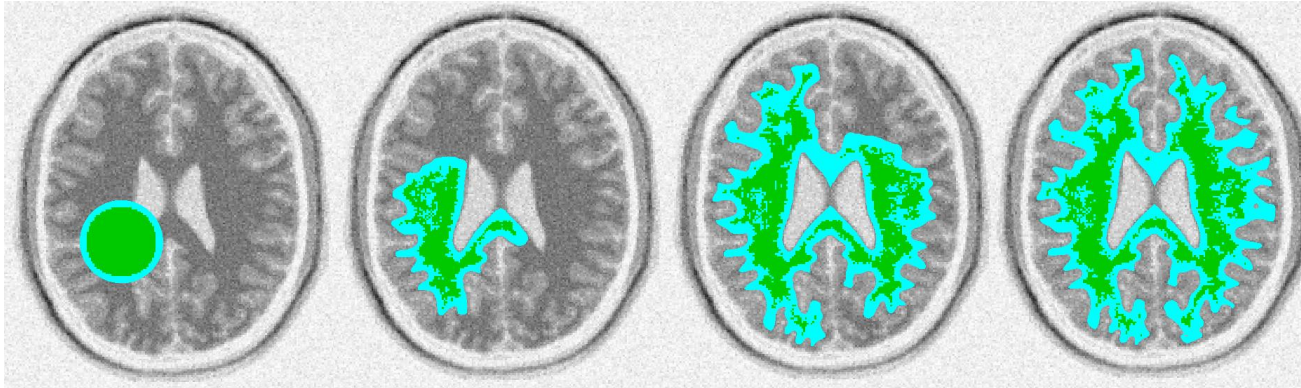
We only want to spend time updating the voxels
that are actually changing

Leveraging Temporal Coherence

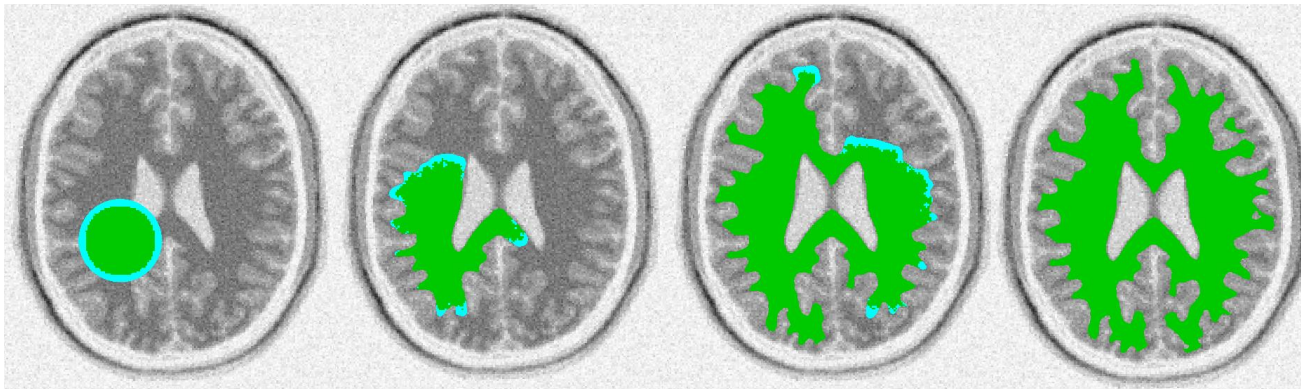
Necessary conditions for voxels to be in the active computational domain:

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?”



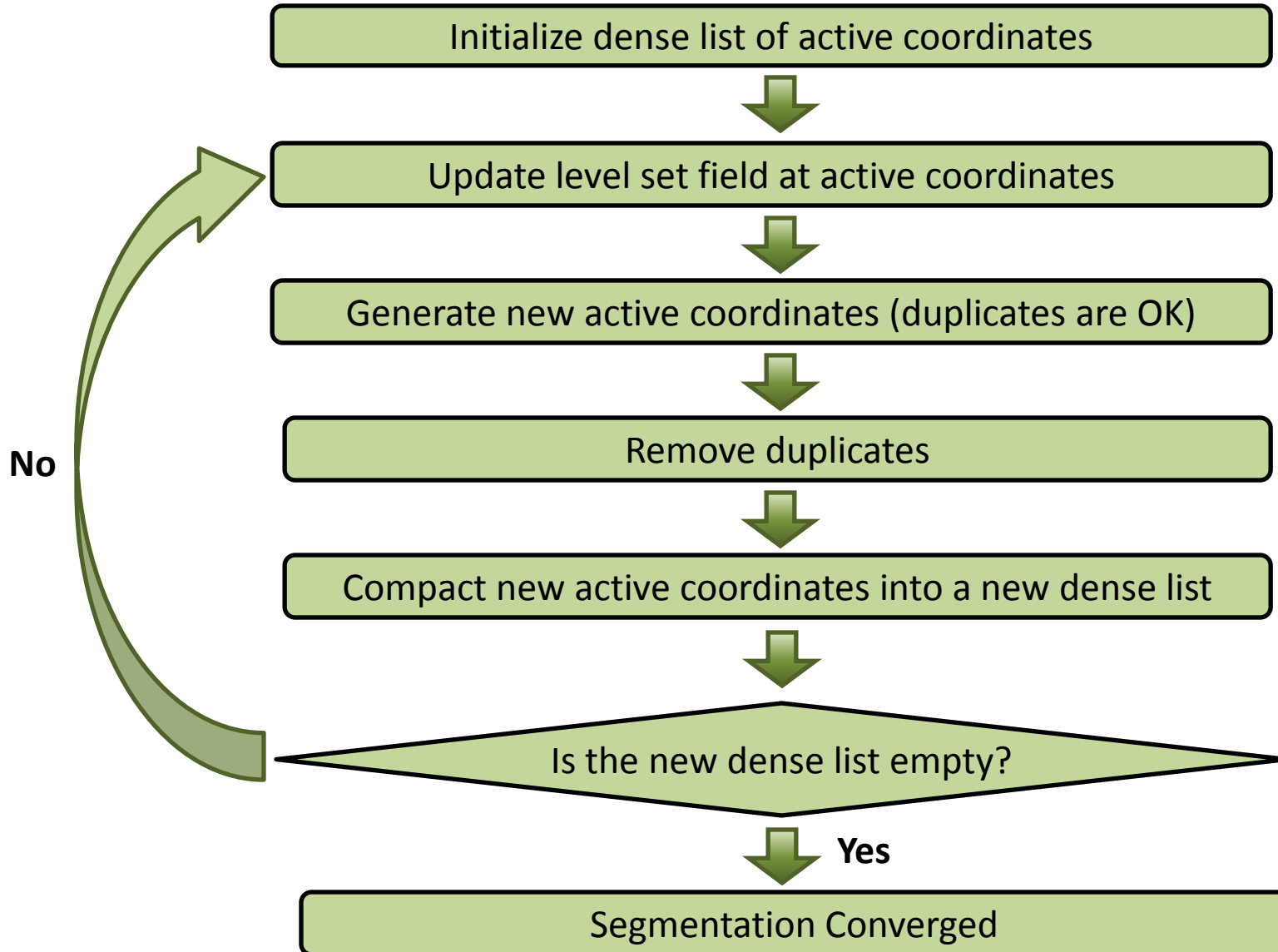
currently active computational domain



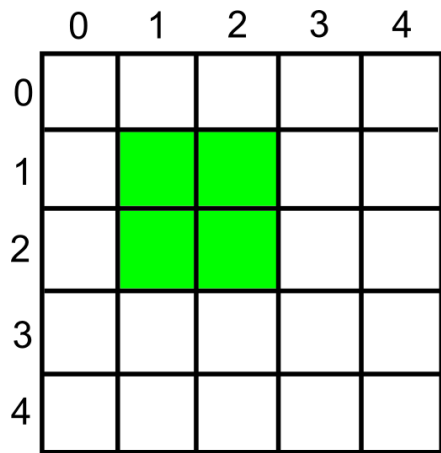
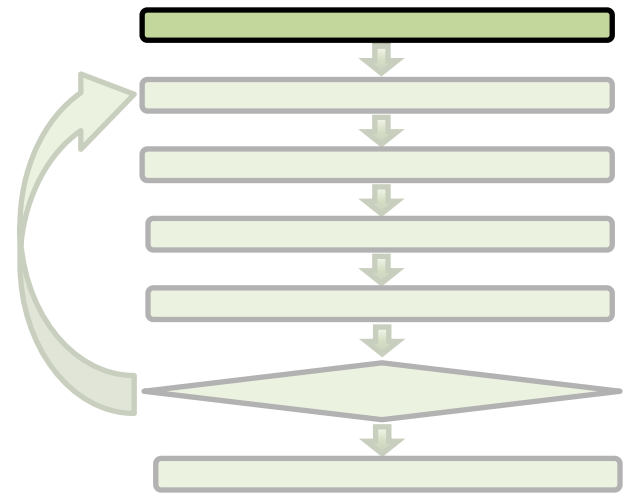
segmented region

Live Demo

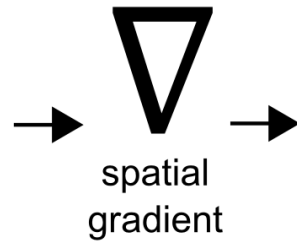
Our Work-Efficient GPU Pipeline



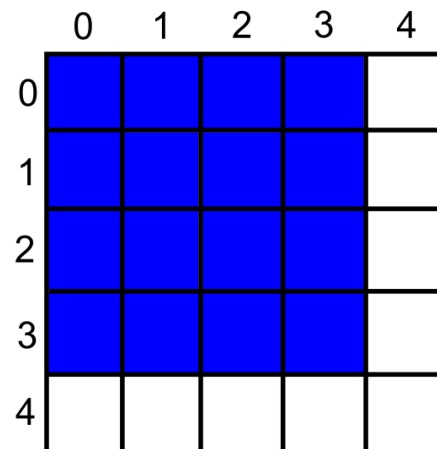
Initializing a scratchpad buffer with active coordinates



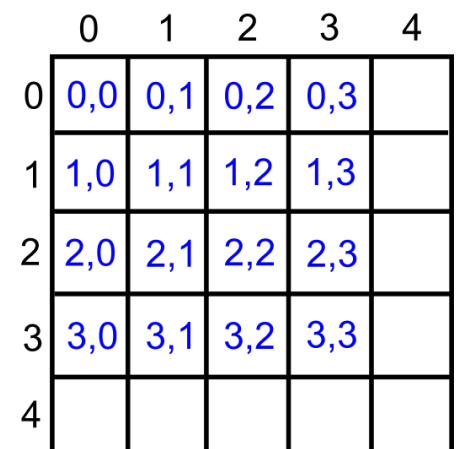
initial level set field



spatial
gradient

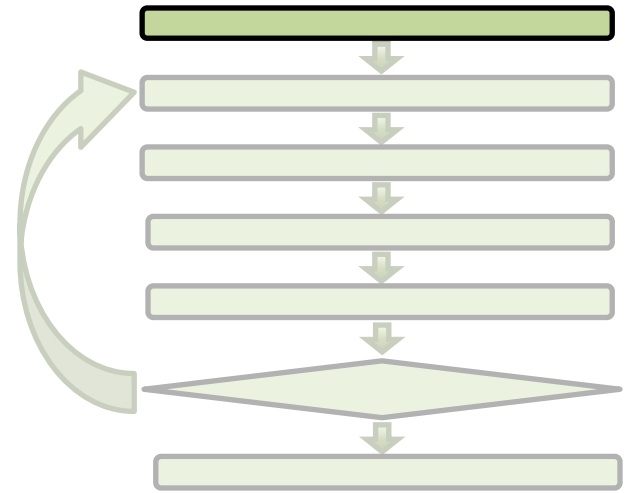


initial active coordinates



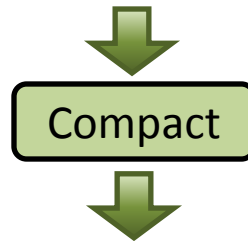
write active coordinates
to scratchpad buffer

Compacting the scratchpad buffer to produce a dense list



scratchpad 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									
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			

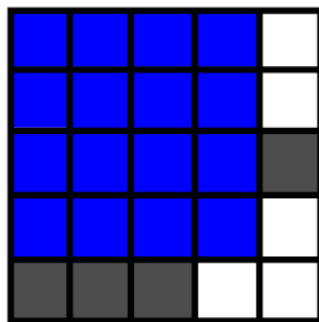
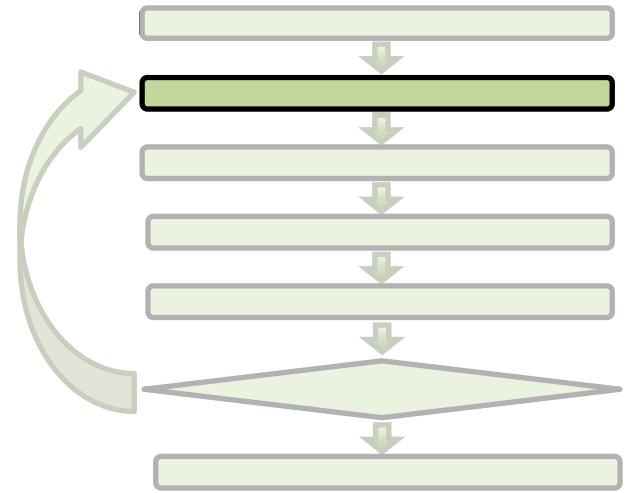


coordinate buffer

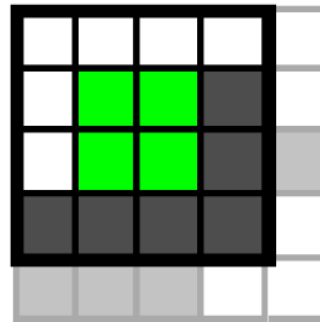
0,0	0,1	0,2	0,3	1,0	1,1	1,2	1,3	2,0	2,1	2,2	2,3	3,0	3,1	3,2	3,3													
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24				

For more details see Harris et al. 2007; Sengupta et al. 2007, 2008

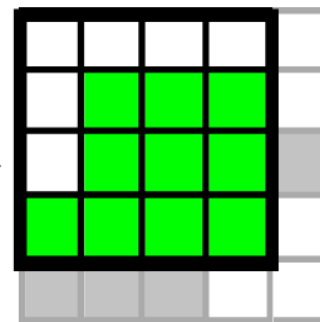
Updating the level set field at active coordinates



active coordinates

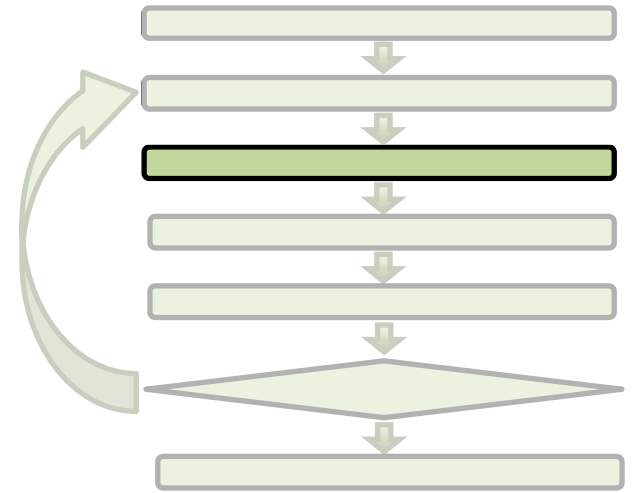


old level set field

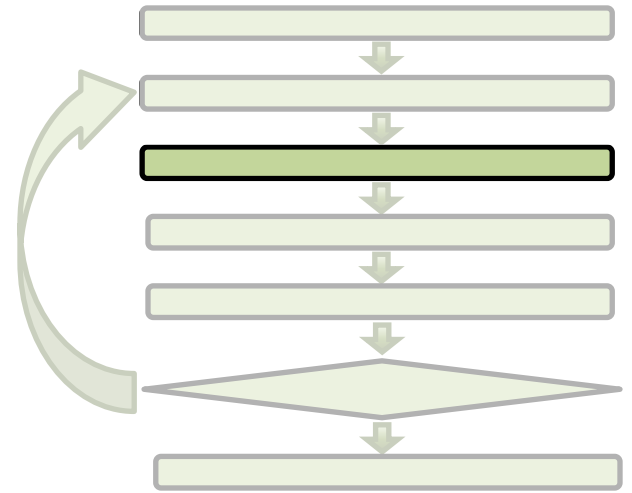


new level set field

Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)

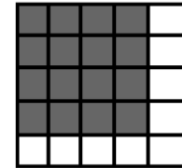


Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)



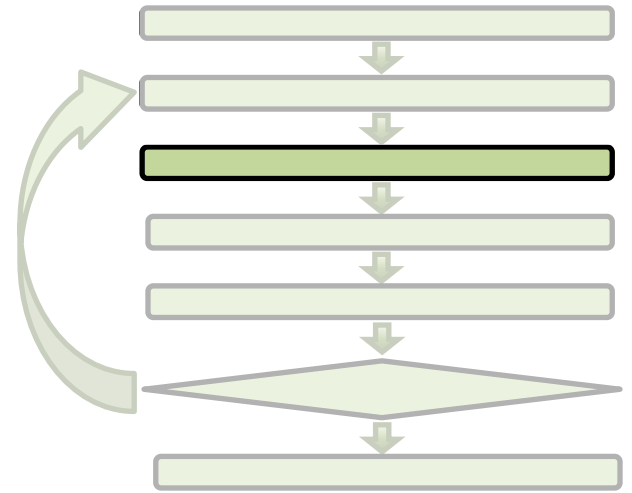
coordinate buffer

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
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----



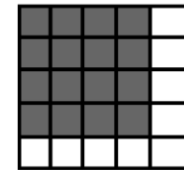
old active set

Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)

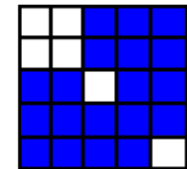


coordinate buffer

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
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

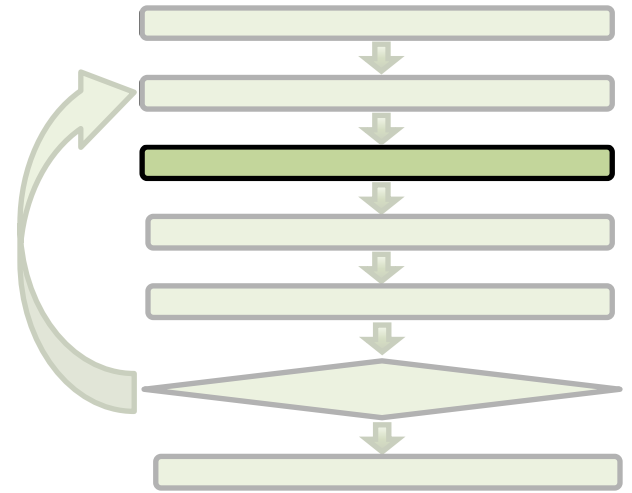


old active set



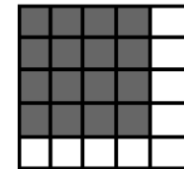
new active set

Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)



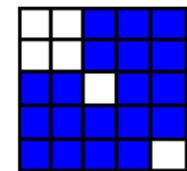
coordinate buffer

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
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----



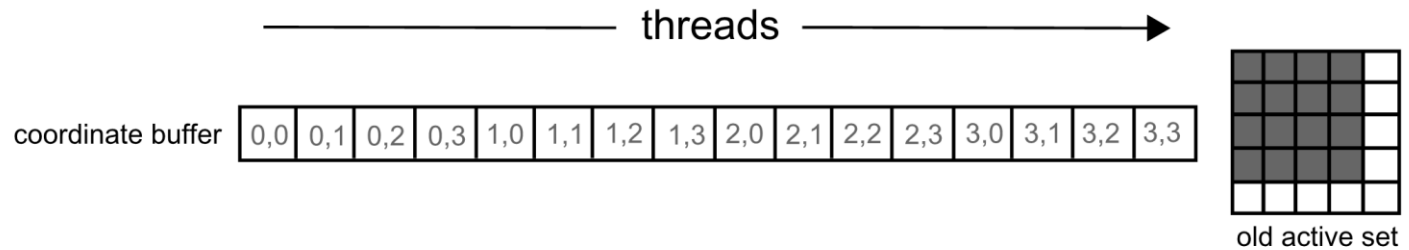
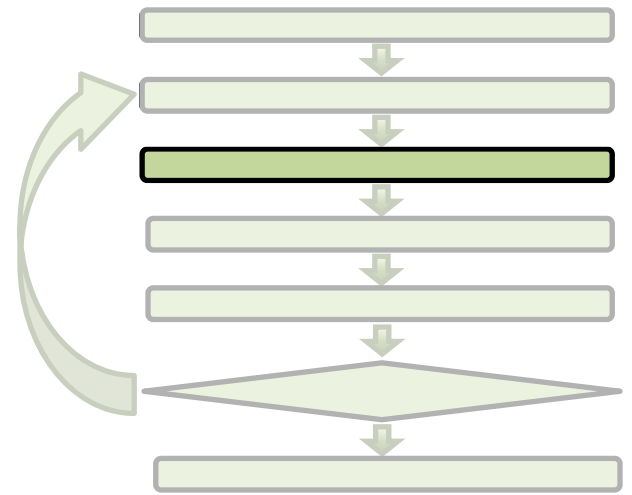
old active set

auxiliary buffers { B^\bullet
 B^{\rightarrow}
 B^{\uparrow}
 B^{\leftarrow}
 B^{\downarrow}

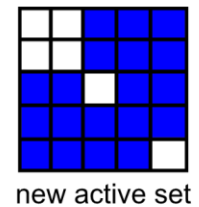


new active set

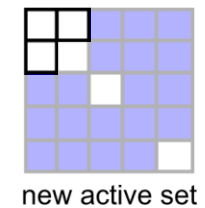
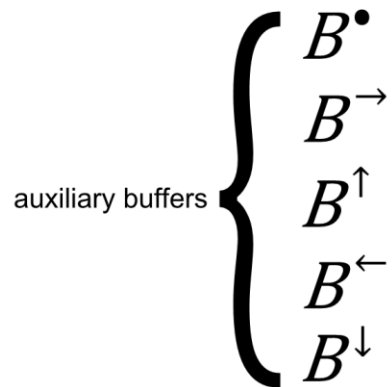
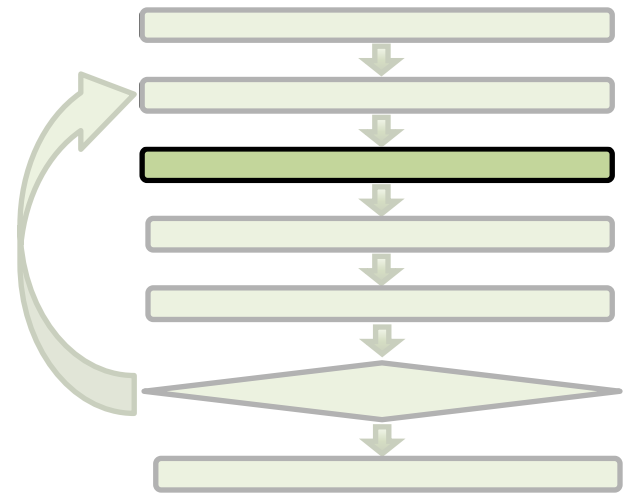
Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)



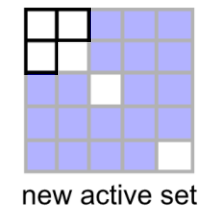
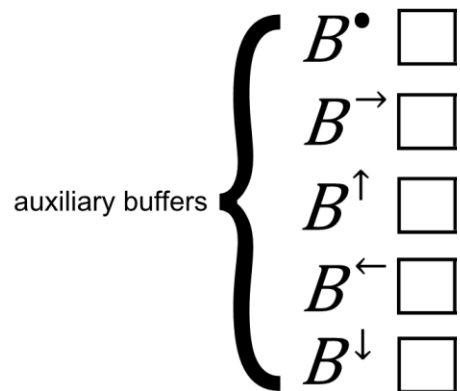
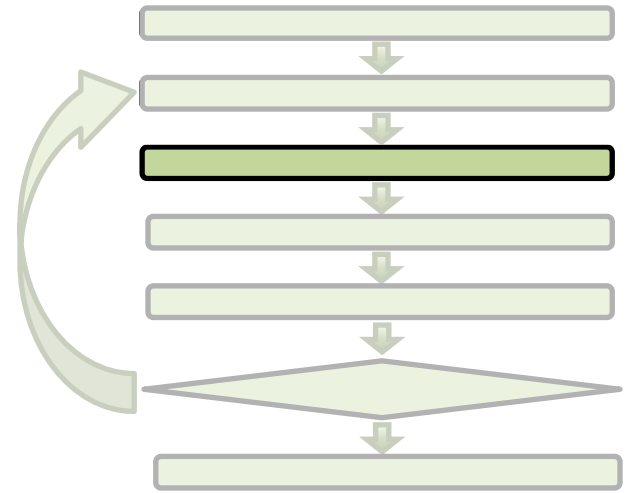
auxiliary buffers

$$\left\{ \begin{array}{l} B^\bullet \\ B^\rightarrow \\ B^\uparrow \\ B^\leftarrow \\ B^\downarrow \end{array} \right.$$


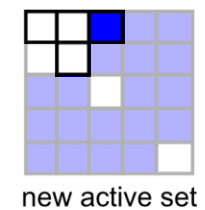
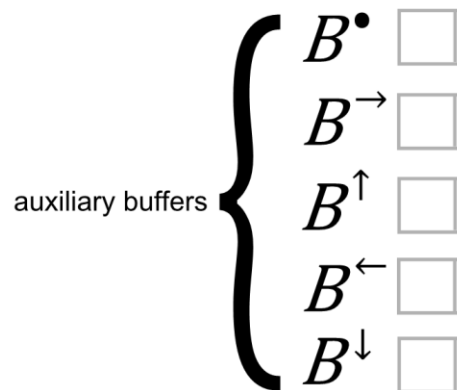
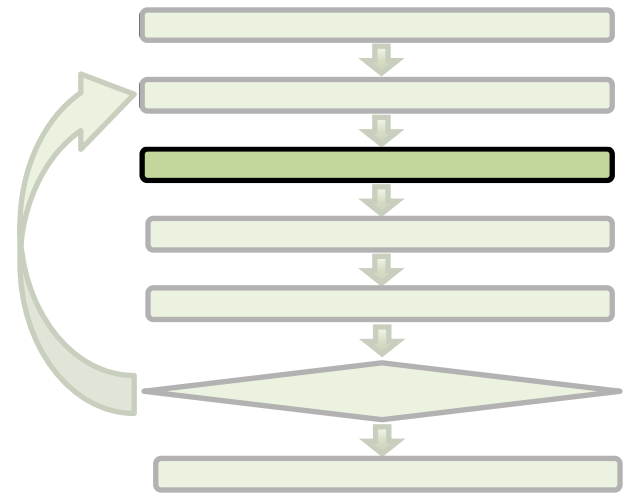
Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)



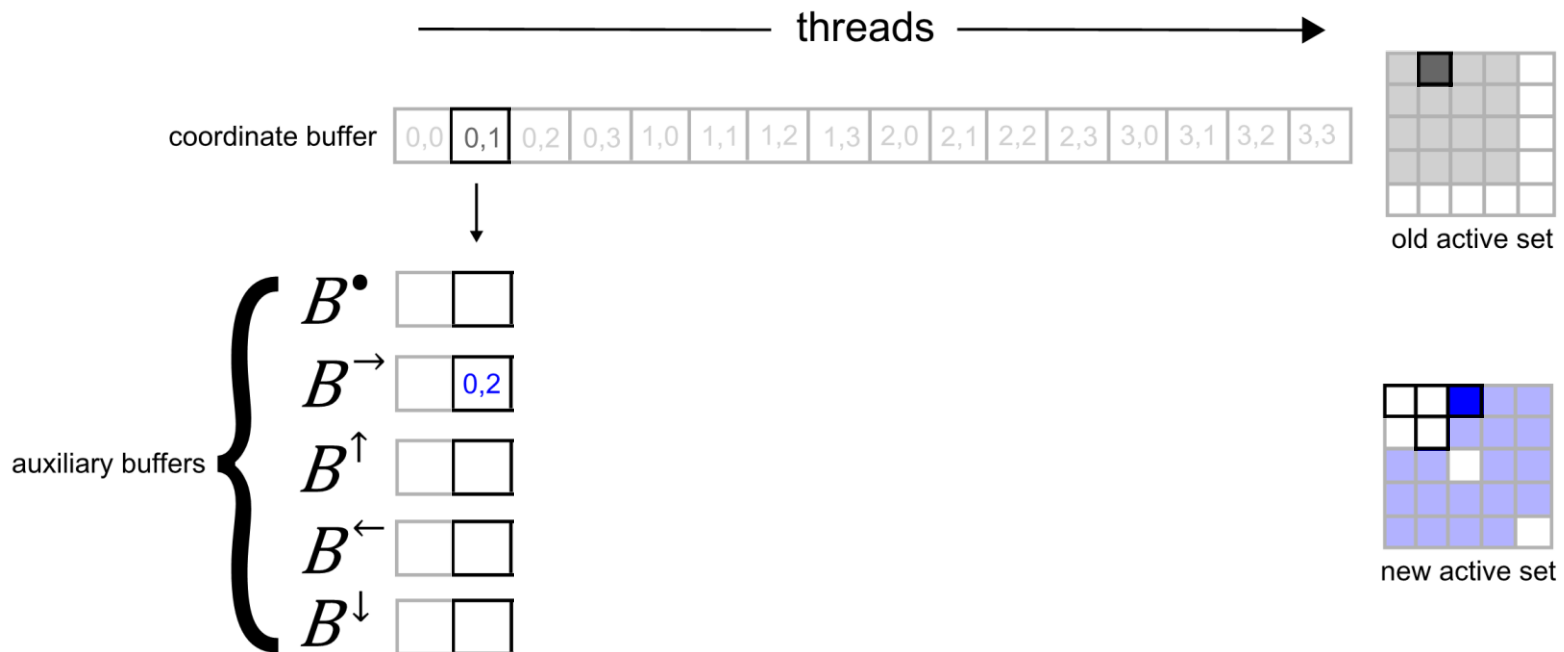
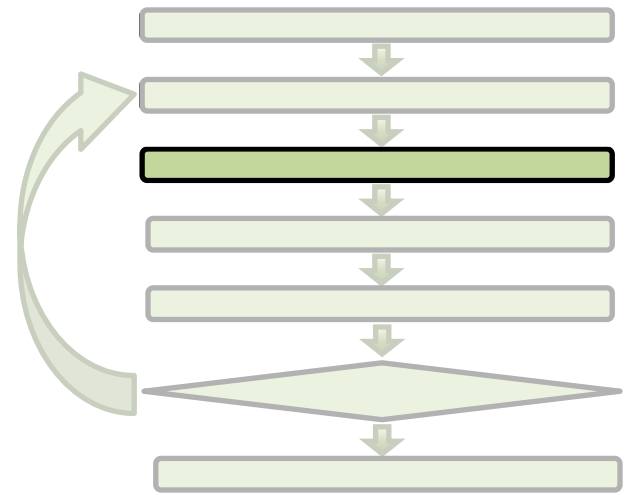
Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)



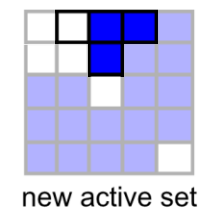
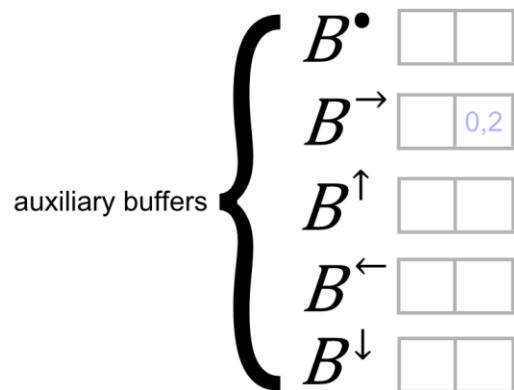
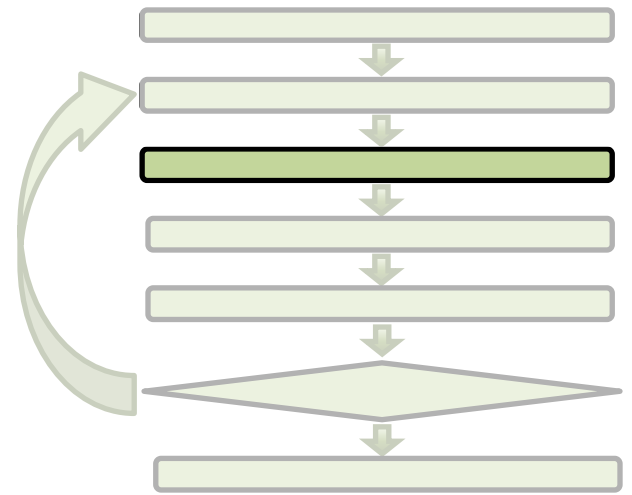
Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)



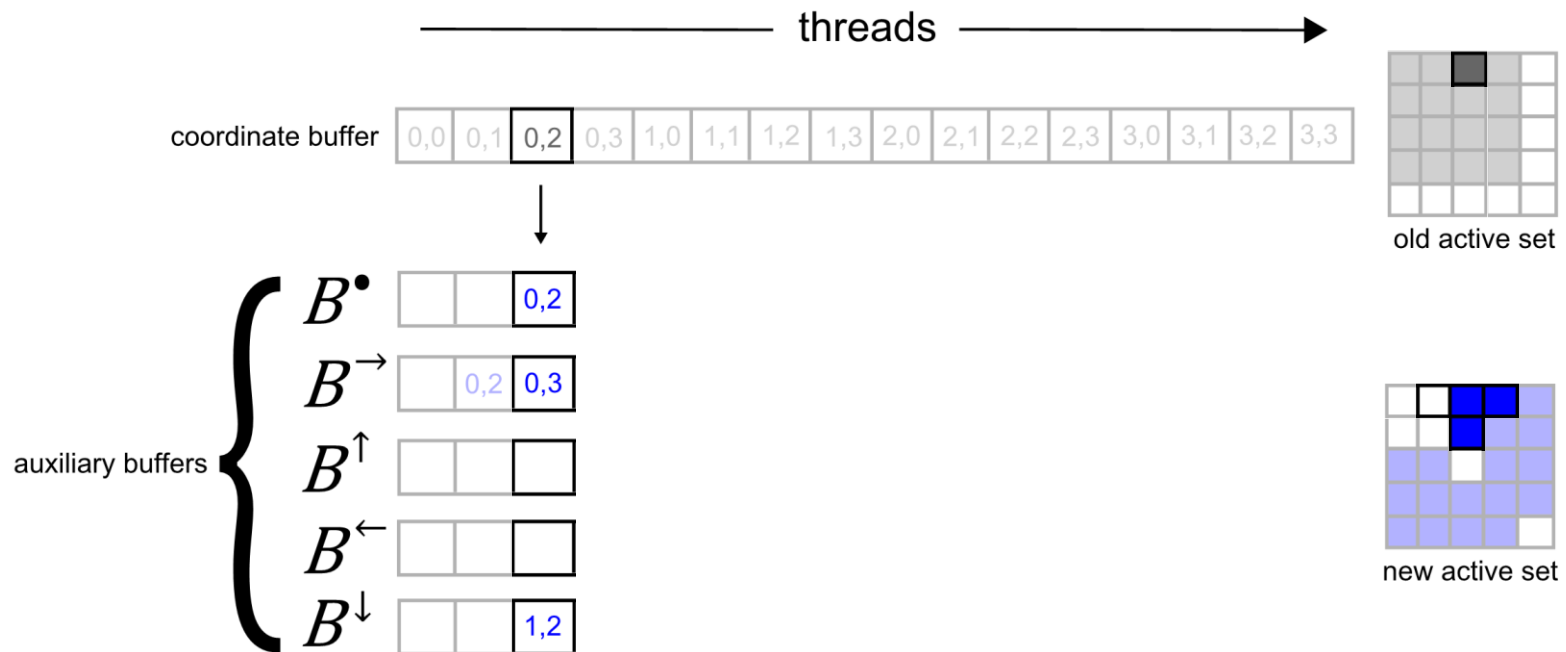
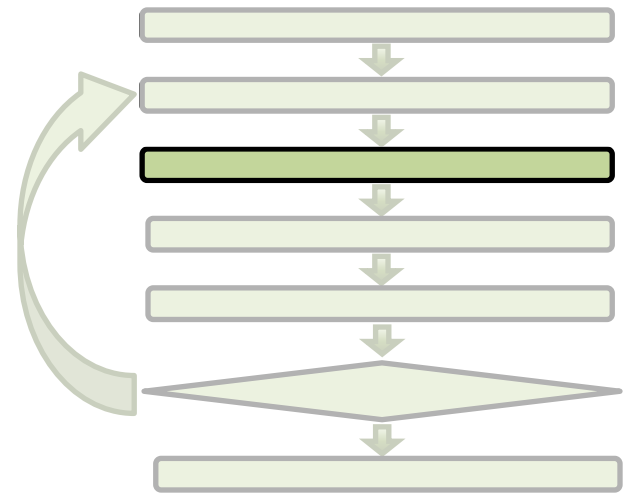
Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)



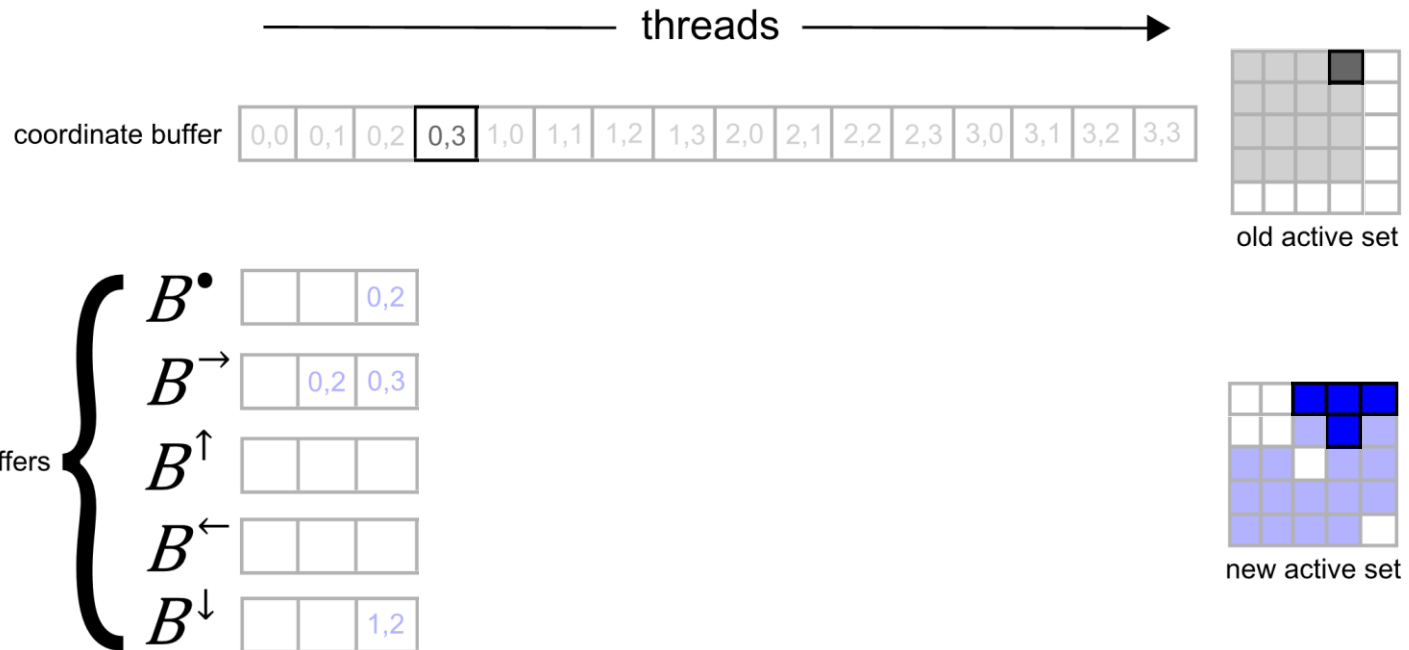
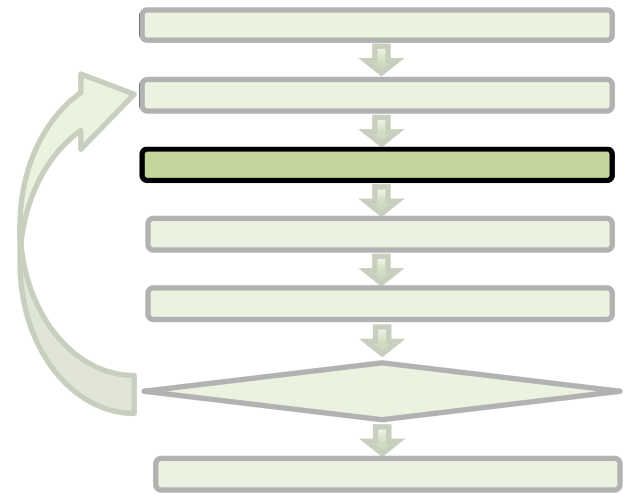
Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)



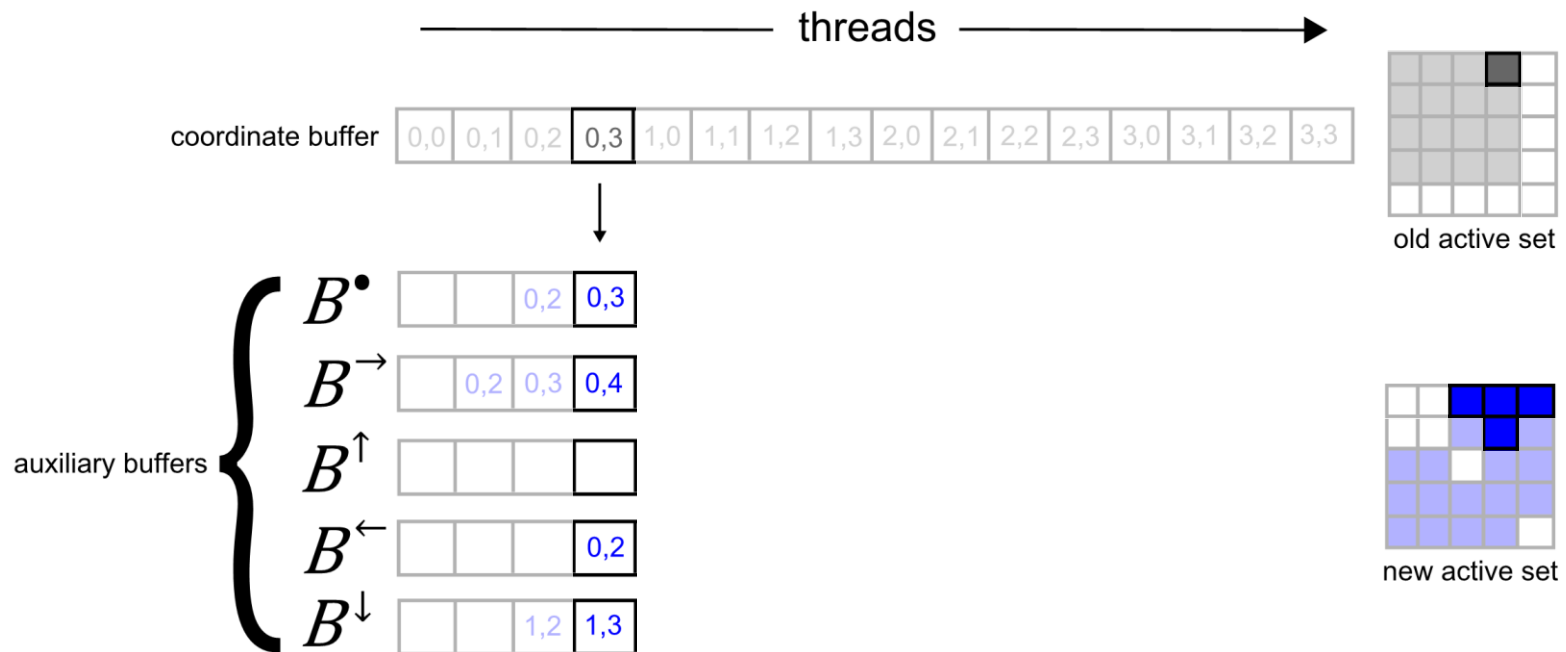
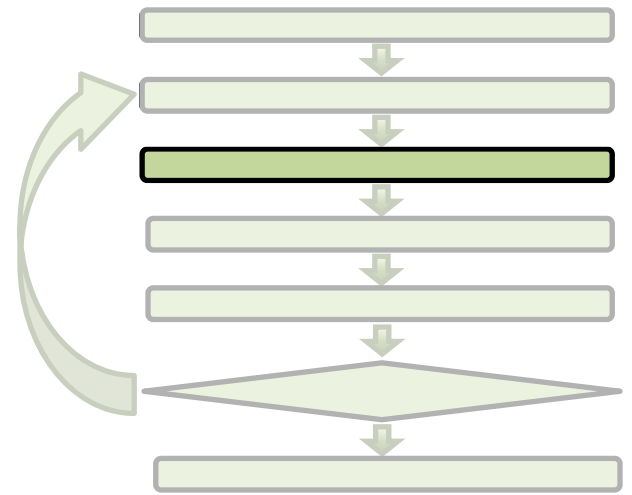
Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)



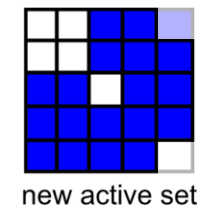
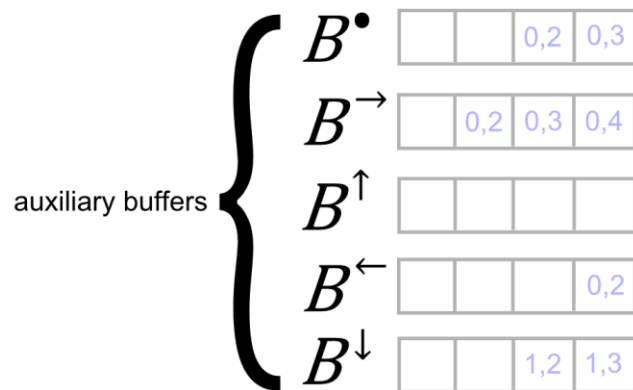
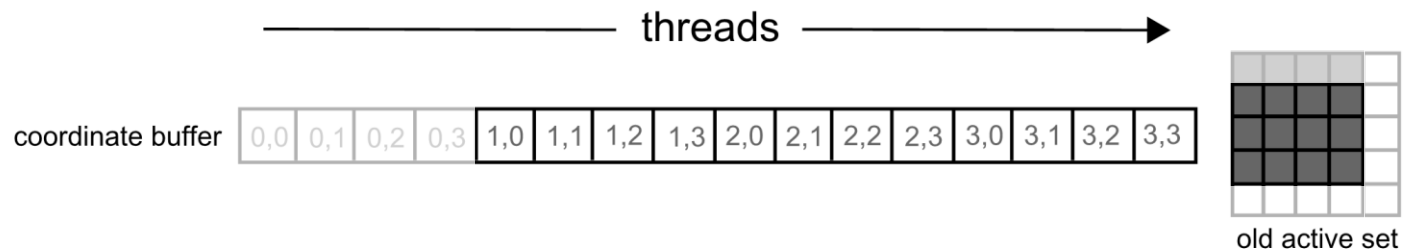
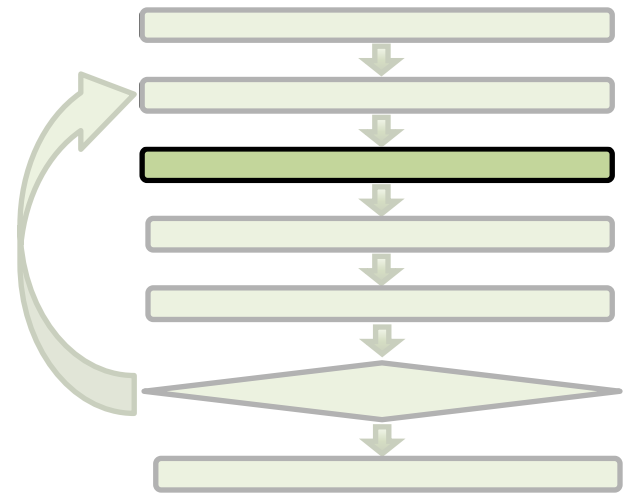
Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)



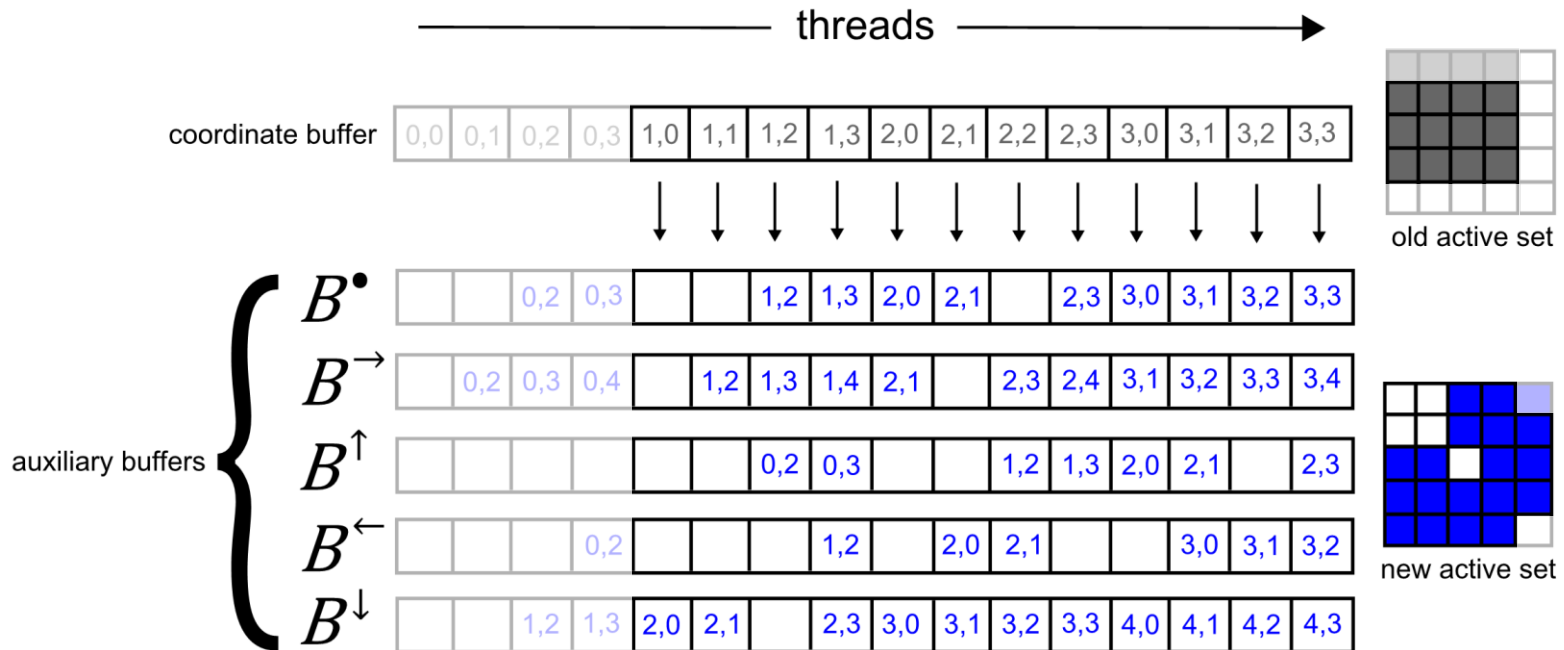
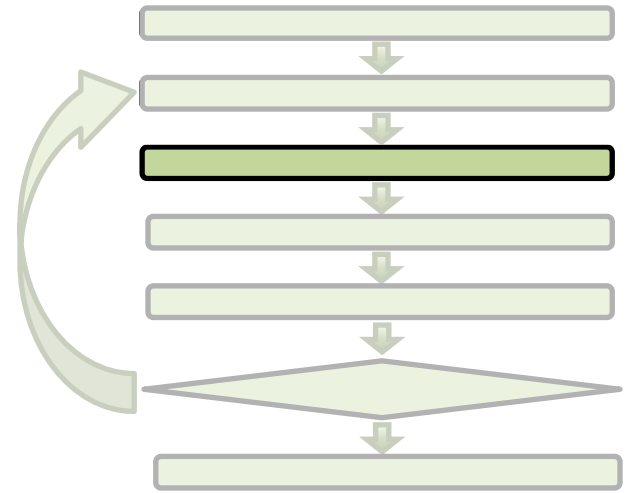
Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)



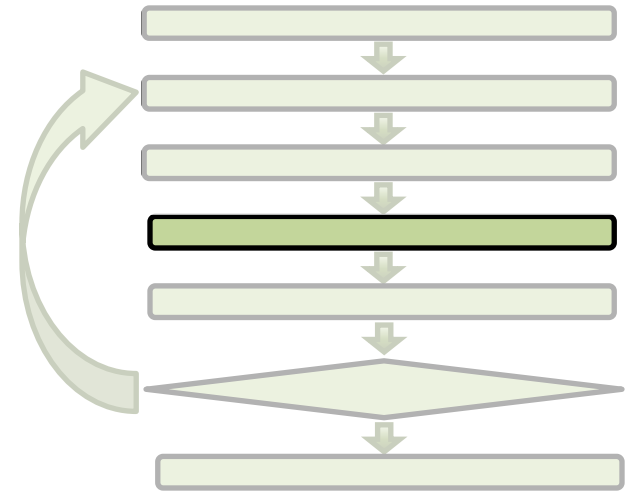
Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)



Generating new active coordinates into a series of auxiliary buffers (duplicates are OK)



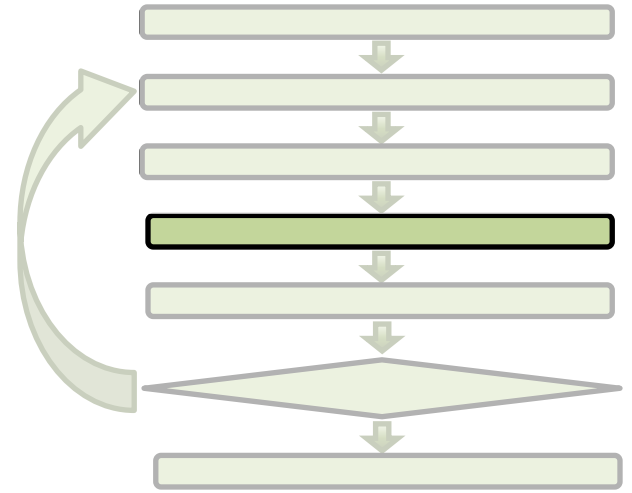
Removing duplicate active coordinates from
the auxiliary buffers



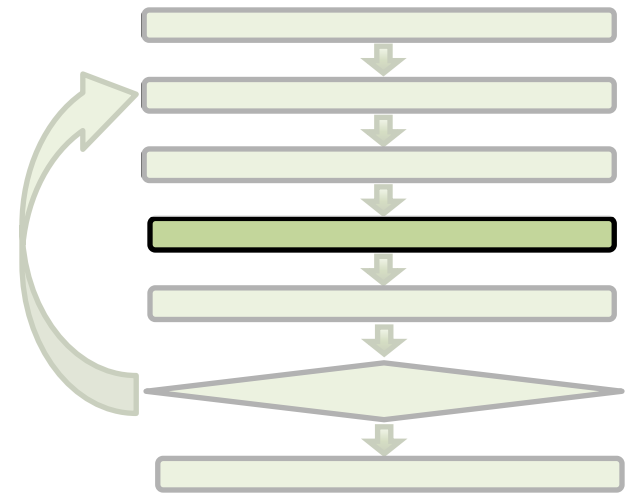
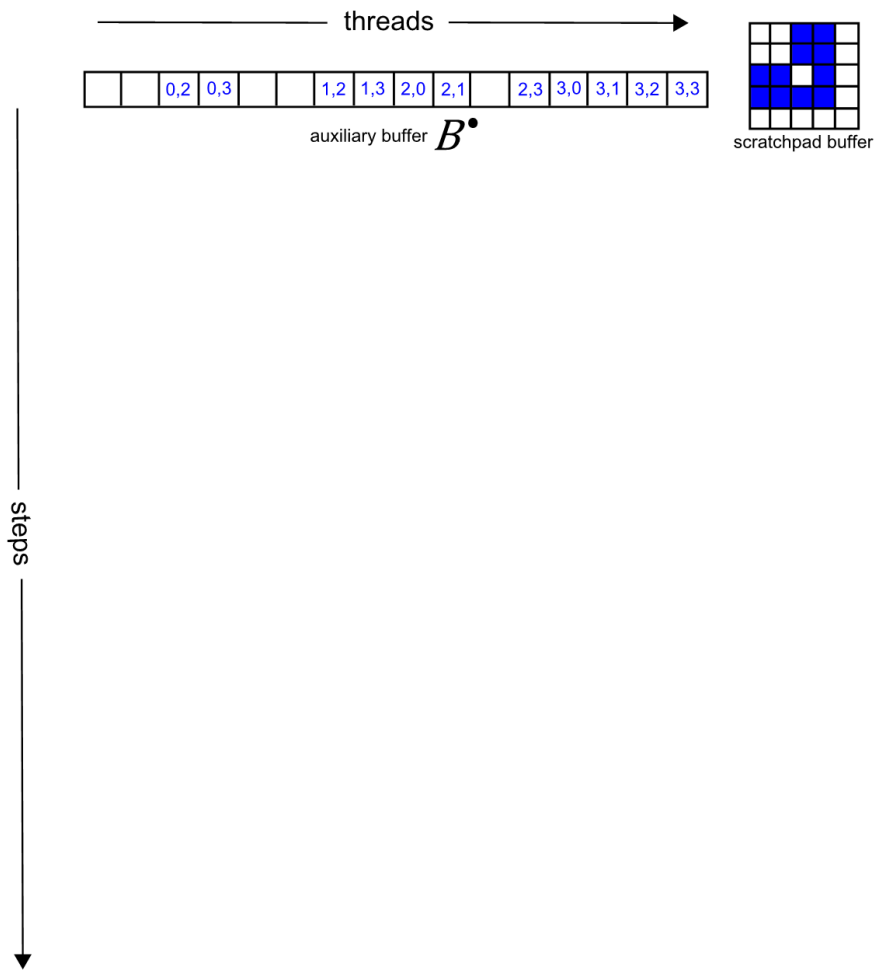
Removing duplicate active coordinates from the auxiliary buffers

threads →

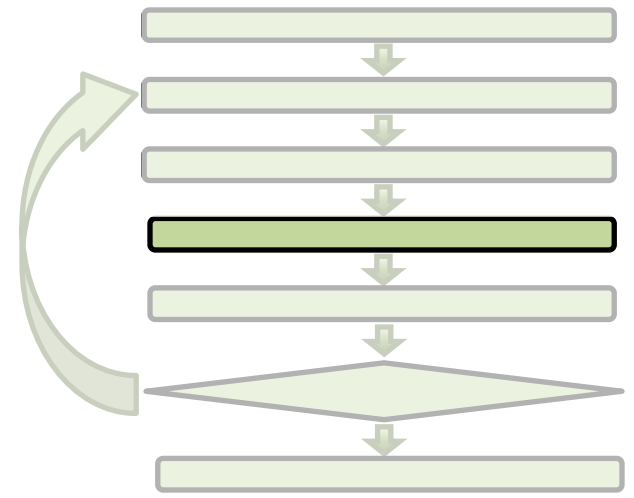
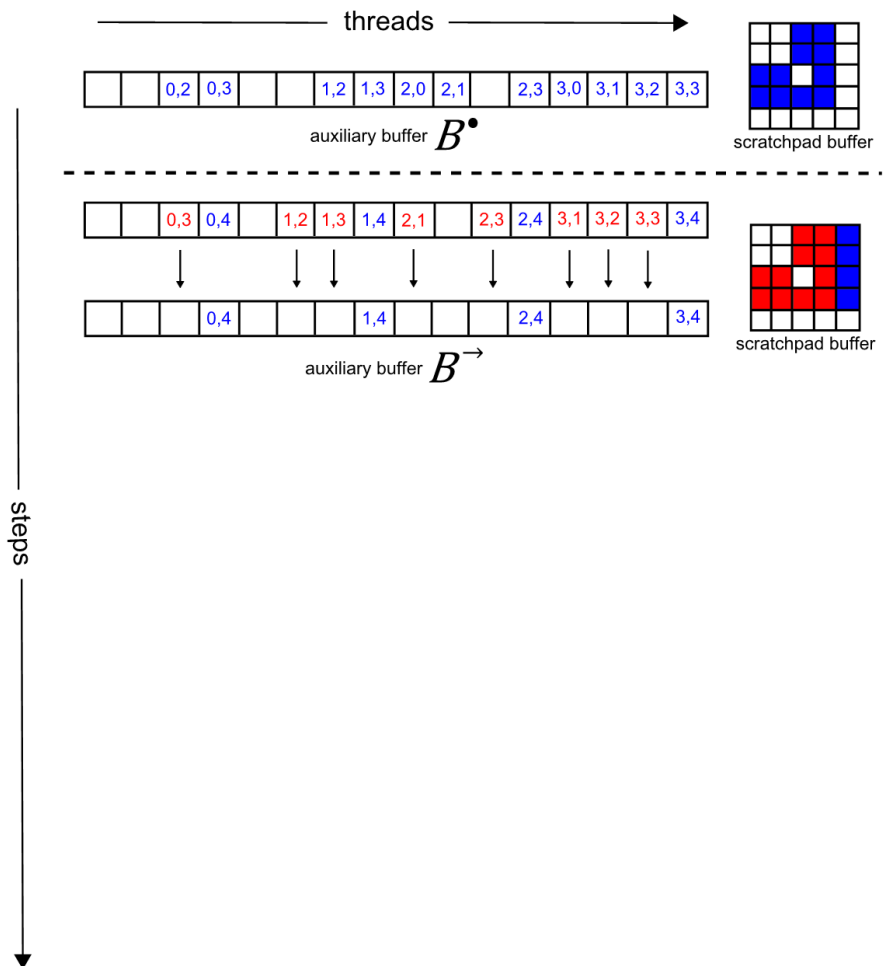
steps ↓



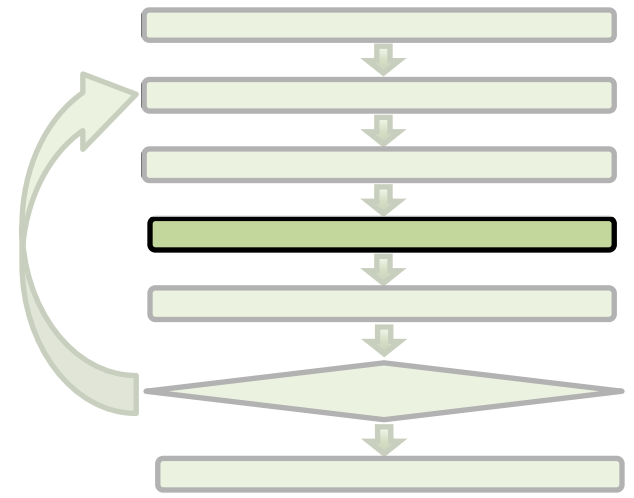
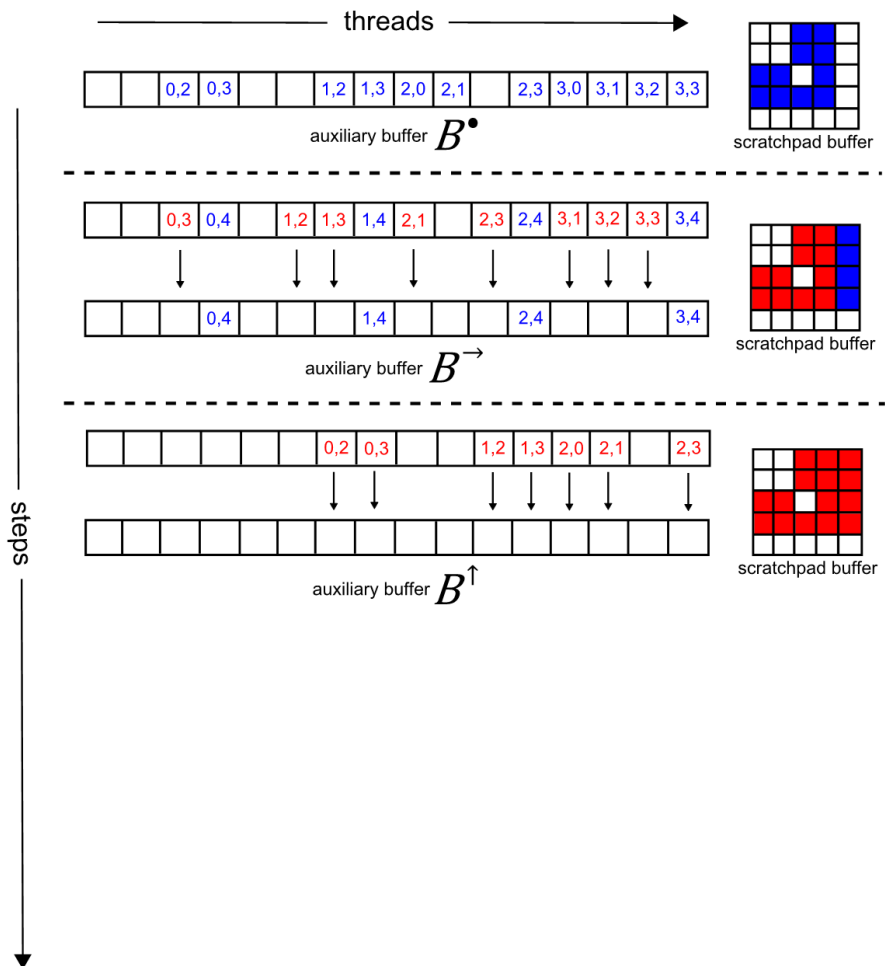
Removing duplicate active coordinates from the auxiliary buffers



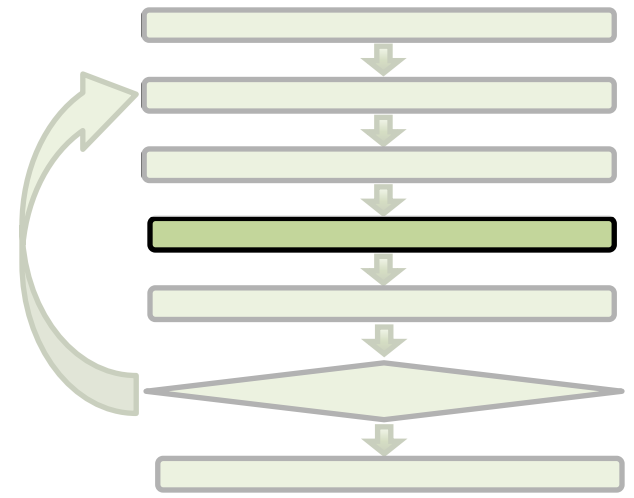
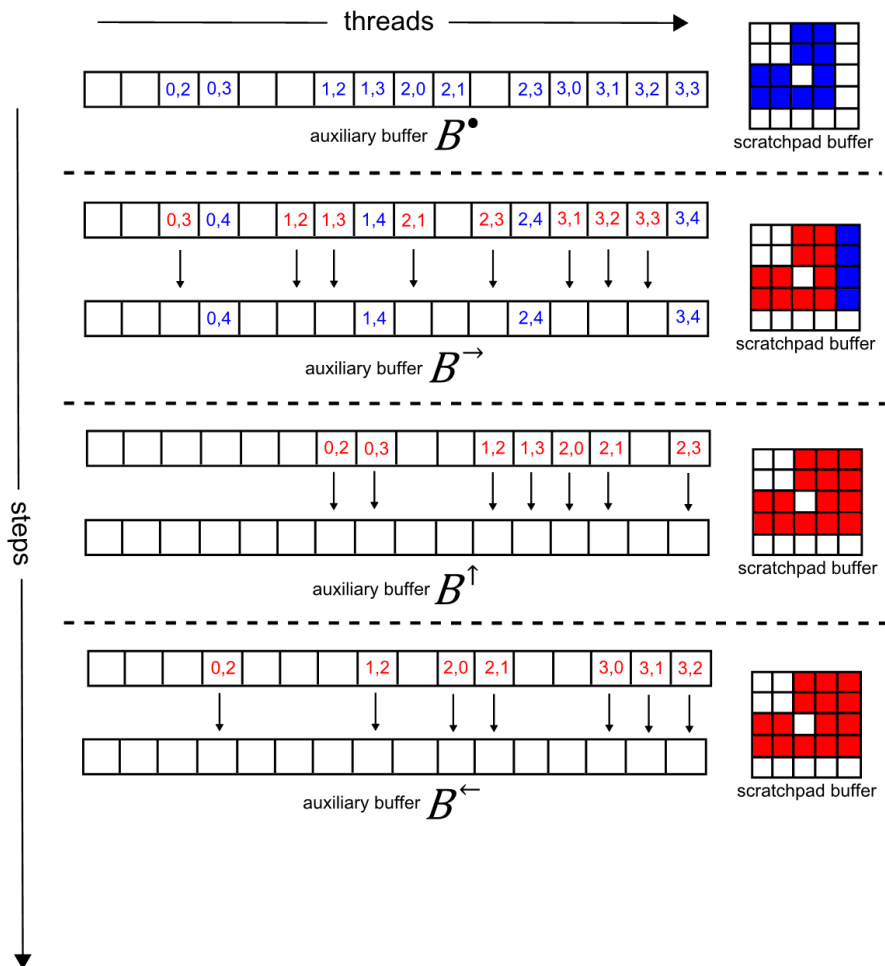
Removing duplicate active coordinates from the auxiliary buffers



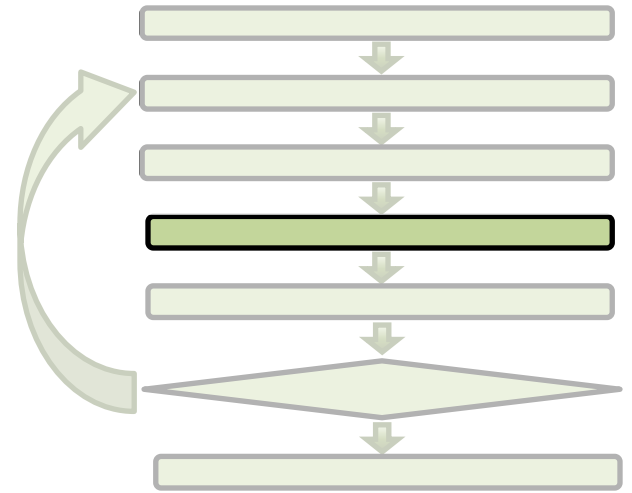
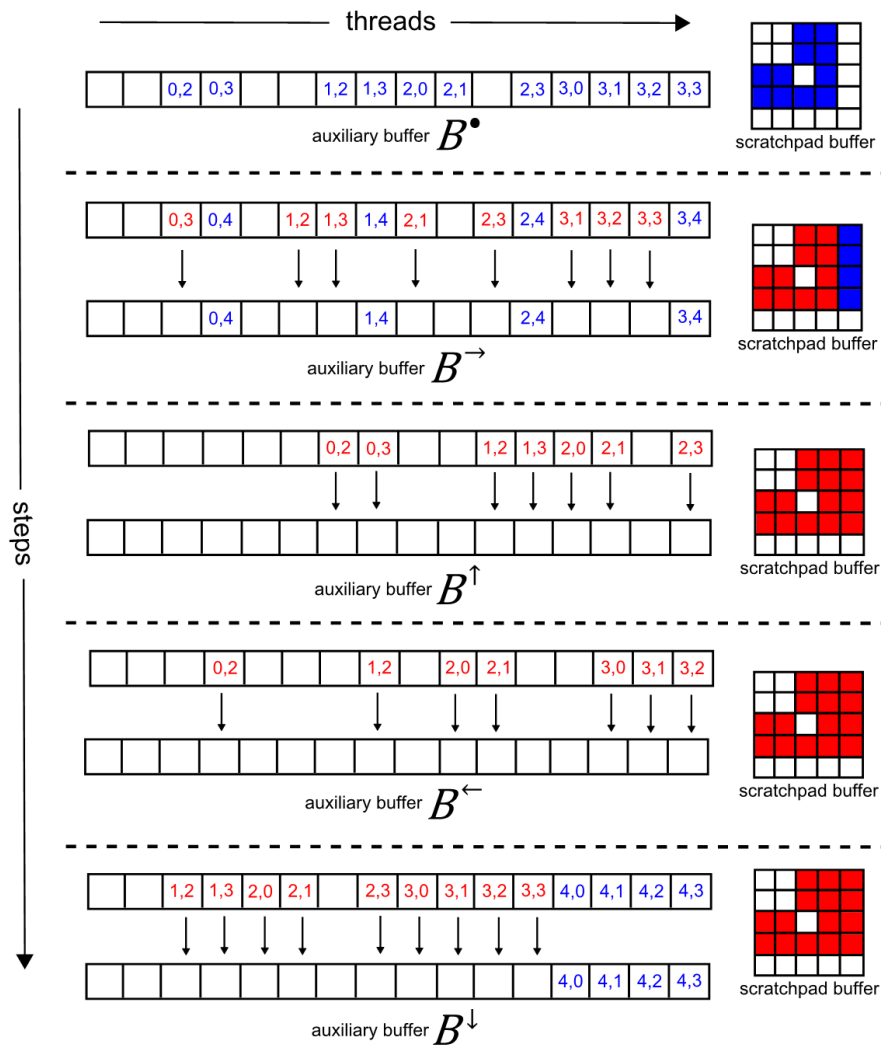
Removing duplicate active coordinates from the auxiliary buffers



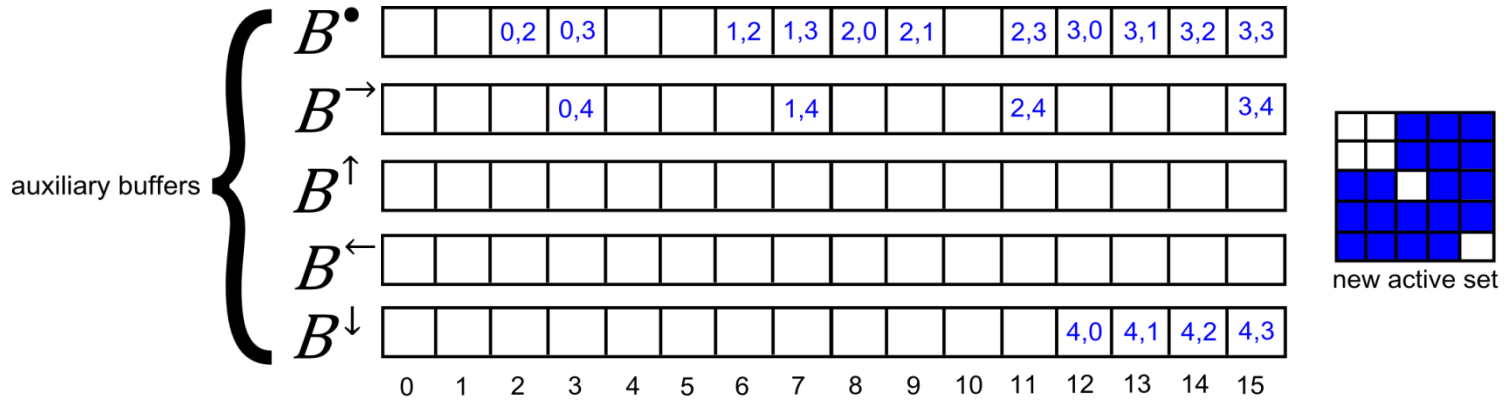
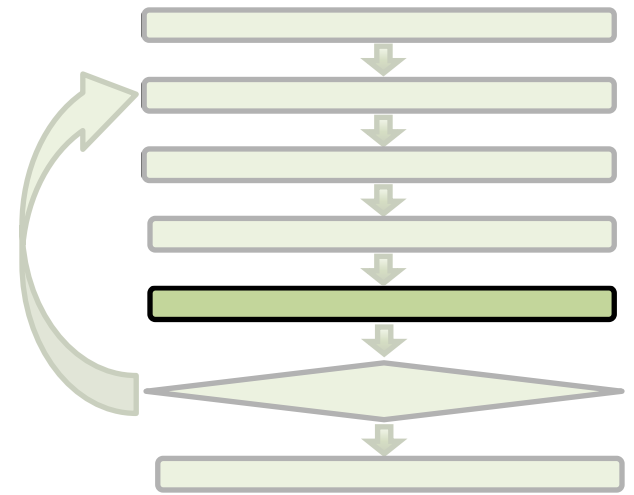
Removing duplicate active coordinates from the auxiliary buffers



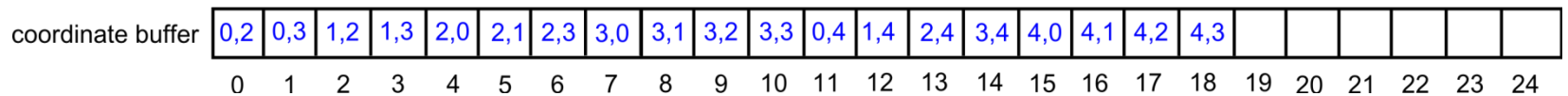
Removing duplicate active coordinates from the auxiliary buffers



Compacting the auxiliary buffers to produce a new dense list of active coordinates



Compact



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

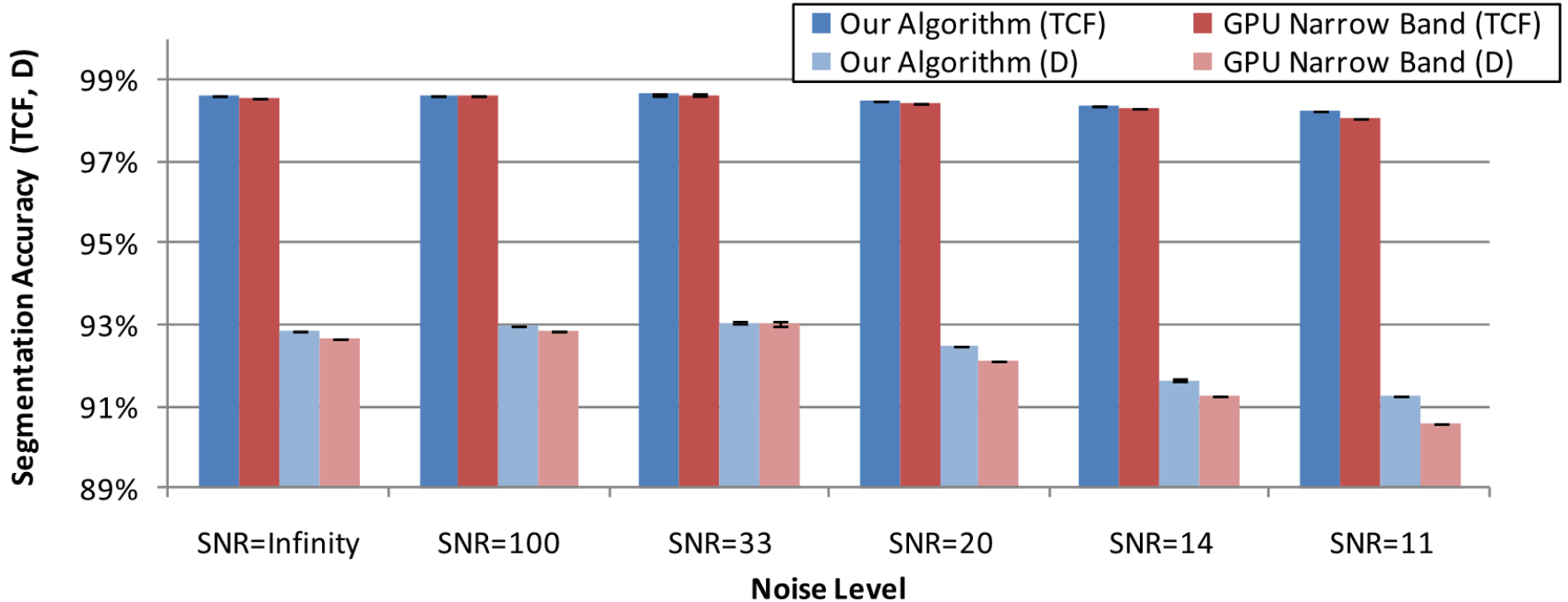
Good: Our algorithm is work-efficient and requires a logarithmic number of steps to update the level set field

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

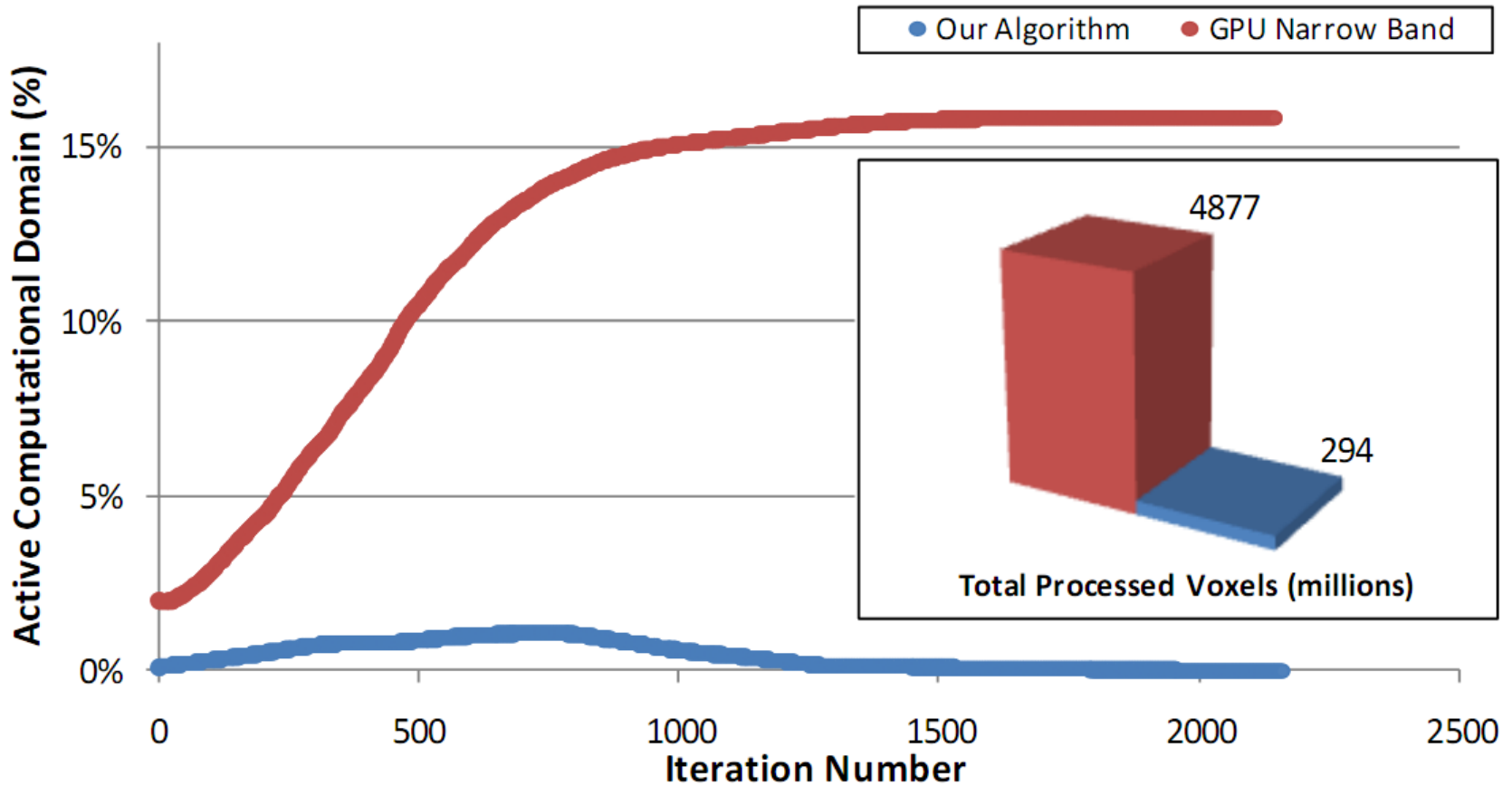


SNR = Signal-to-noise Ratio

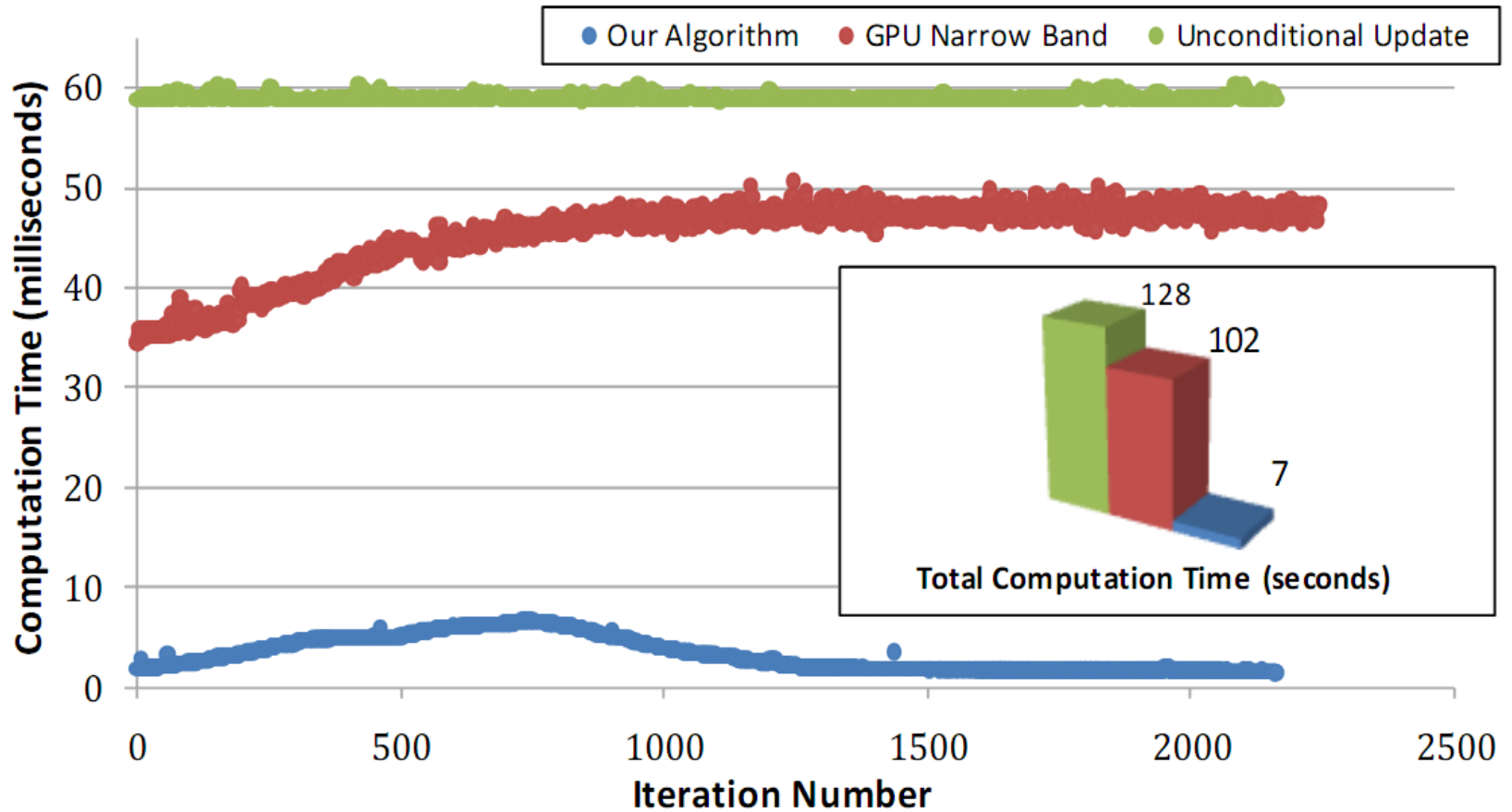
D = Dice Coefficient

TCF = Total Correct Fraction of Labeled Voxels

Computational Domain Size



Speed

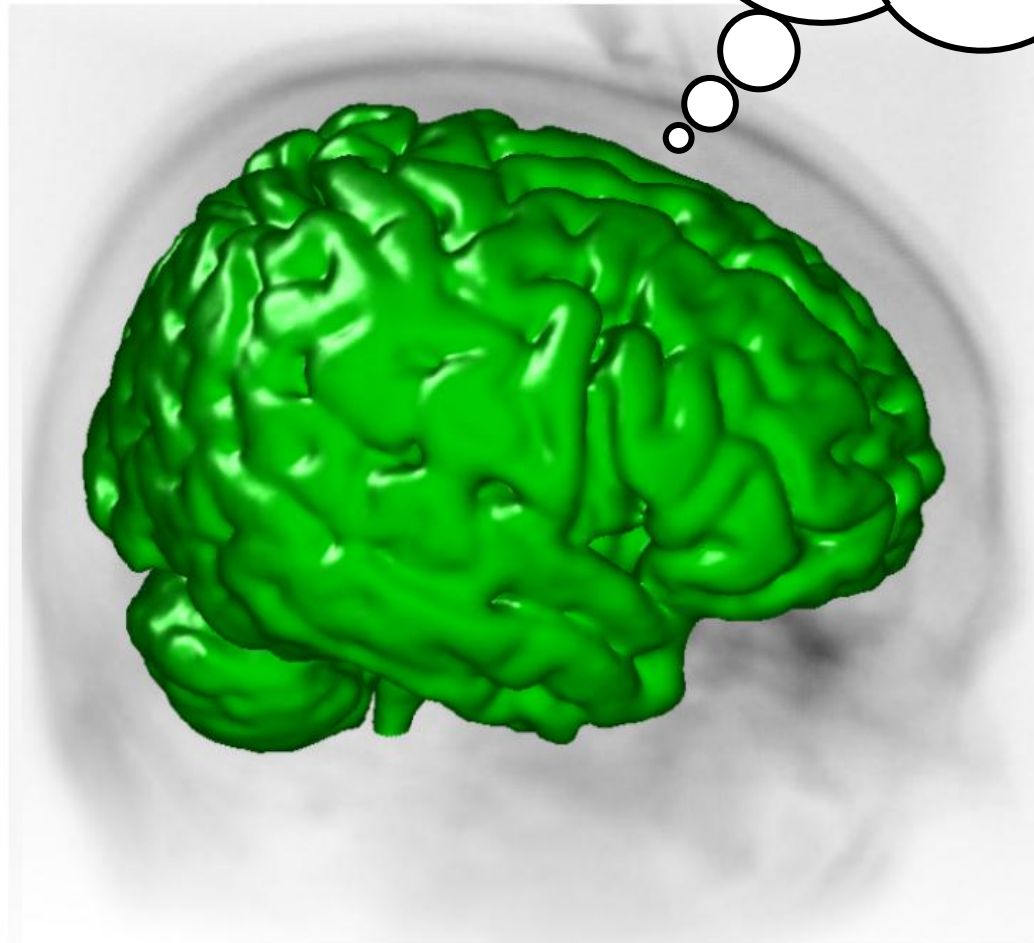


Limitations

- 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

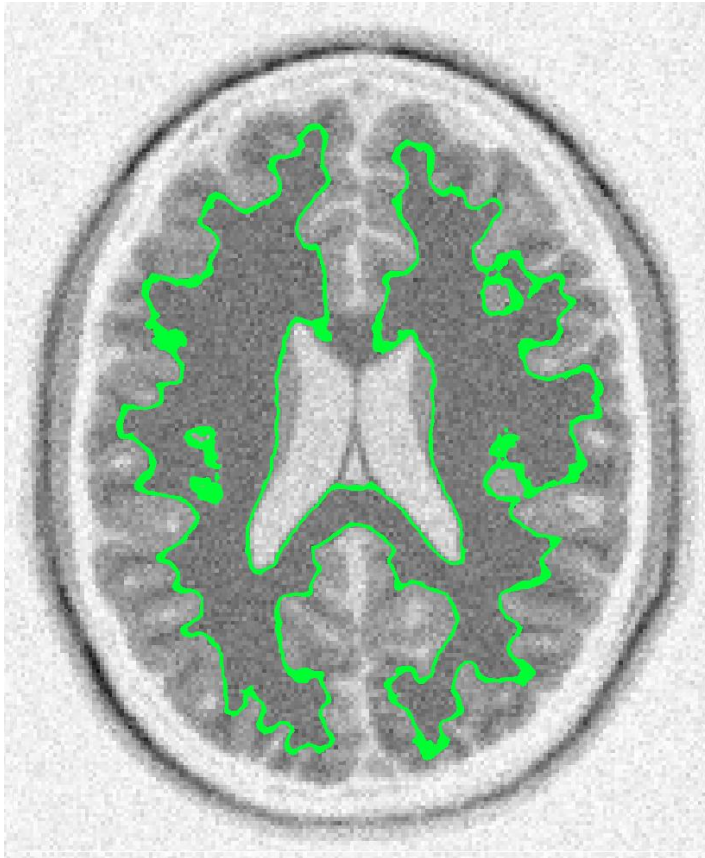
Speed function proposed by Lefohn et al. 2003, 2004

$$\alpha \left(\underbrace{\begin{array}{c} \text{grow} \\ \text{image} \\ \text{intensity} \\ \text{contract} \end{array}}_{\text{data term}} \right) + (1 - \alpha) \left(\underbrace{\begin{array}{c} \text{curvature} \end{array}}_{\text{curvature term}} \right)$$

α controls the smoothness of the segmentation

Speed Function

The curvature term enforces a smooth segmentation and prevents leaking

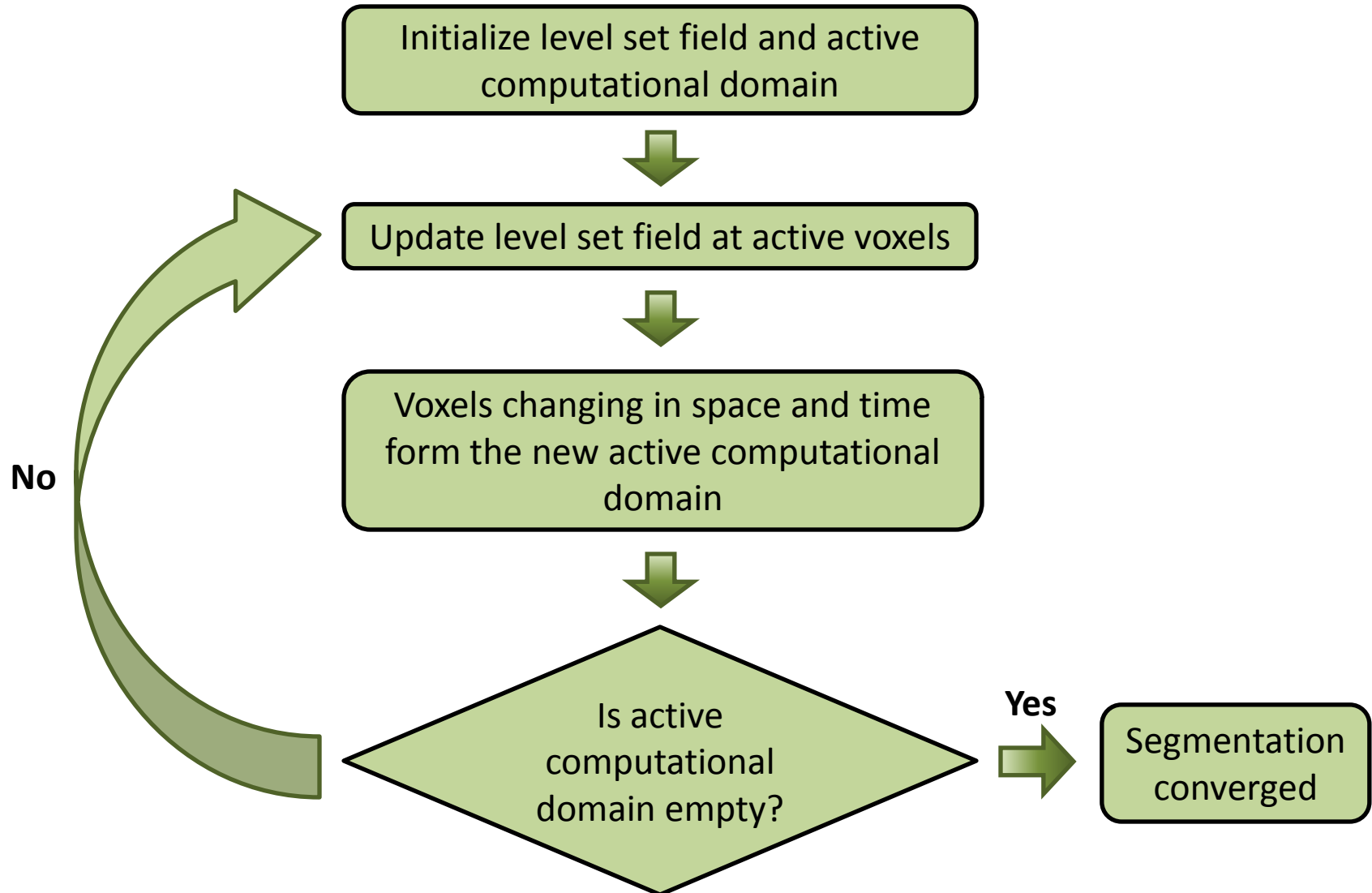


with curvature influence

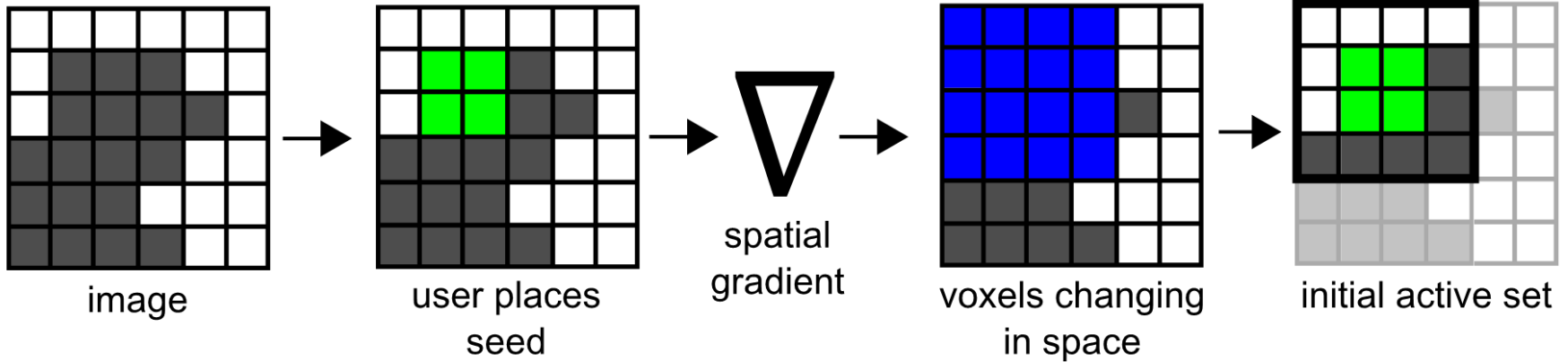
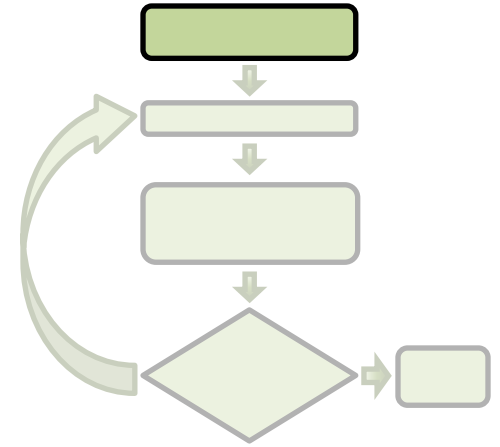


without curvature influence

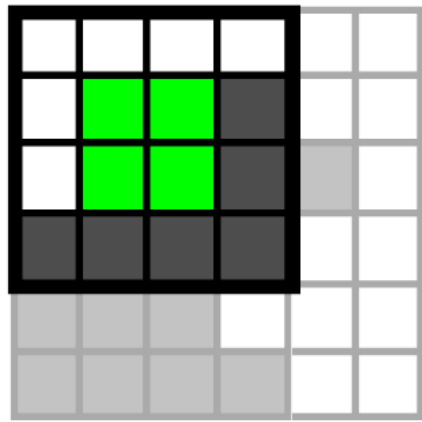
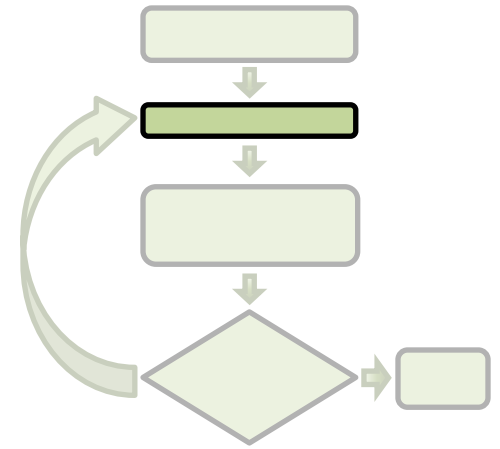
Temporally Coherent Algorithm



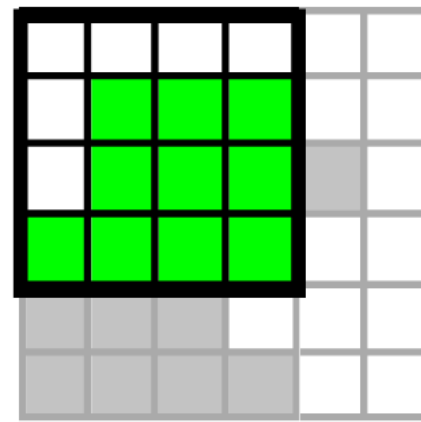
Initializing the level set field and the active computational domain



Updating the level set field at active voxels

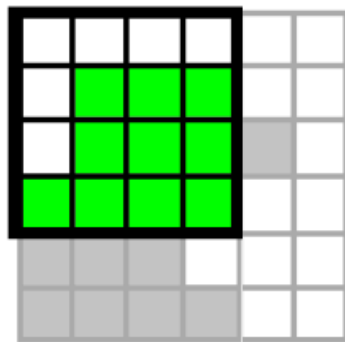
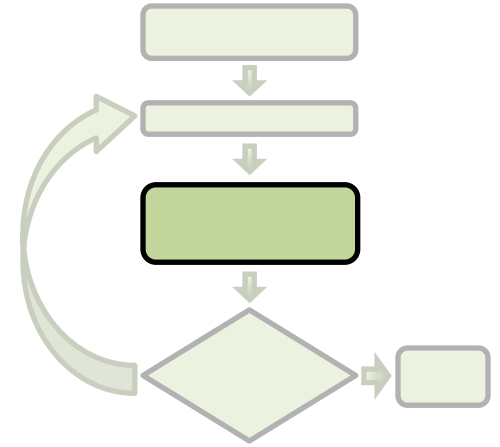


old level set field

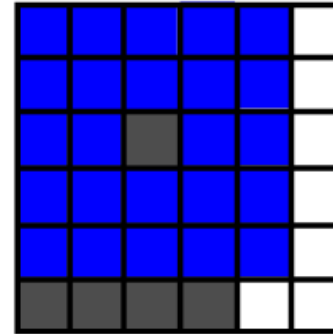


new level set field

Finding the voxels that are changing in *space*

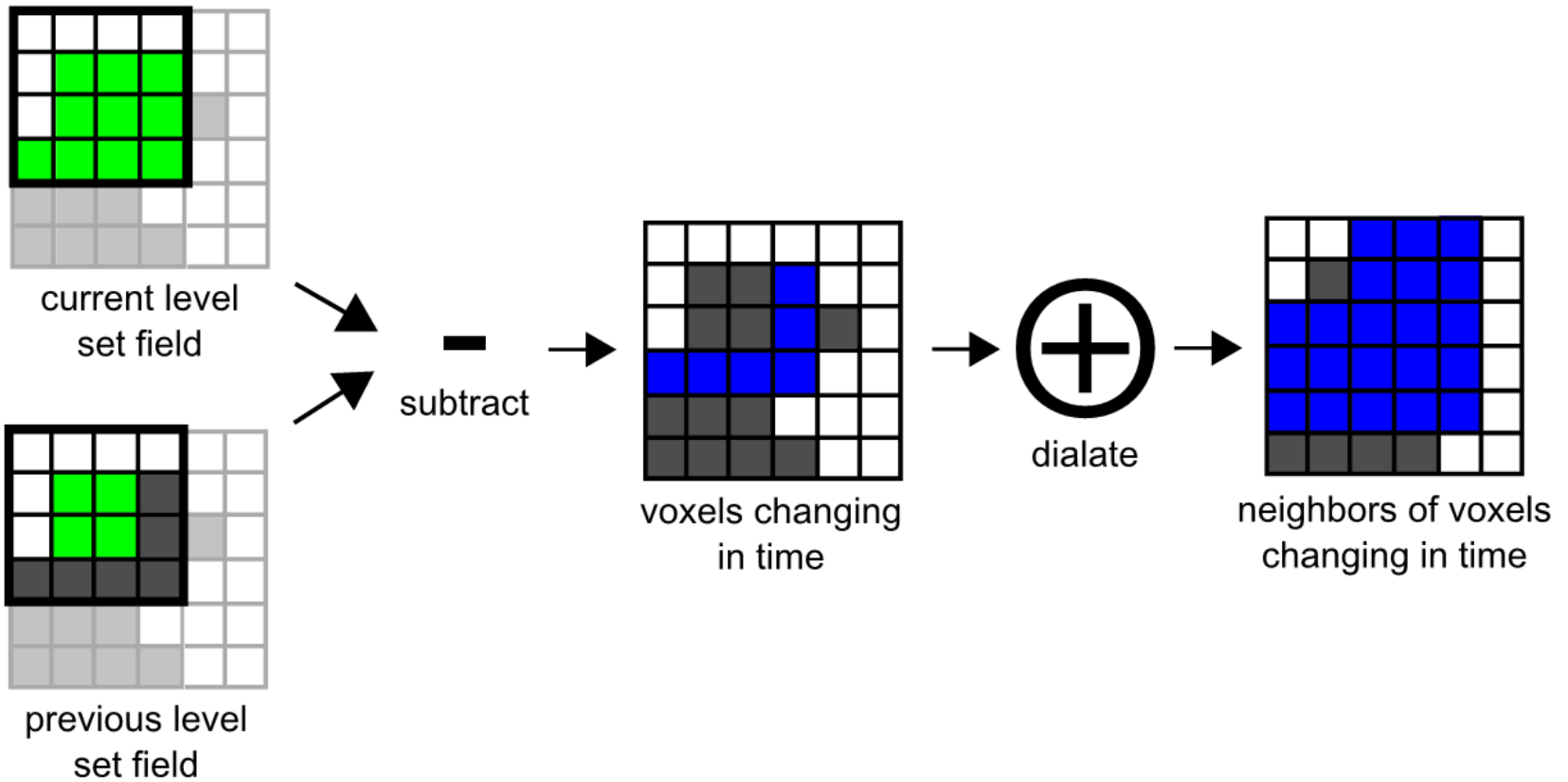
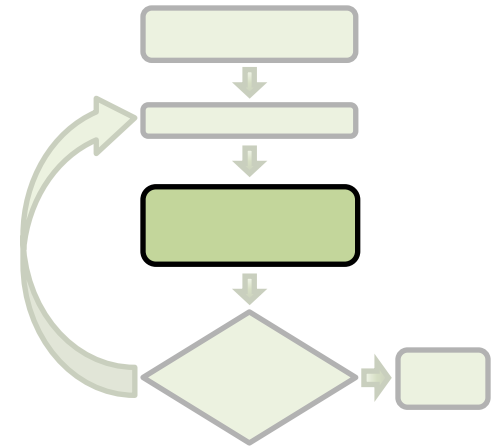


current level
set field

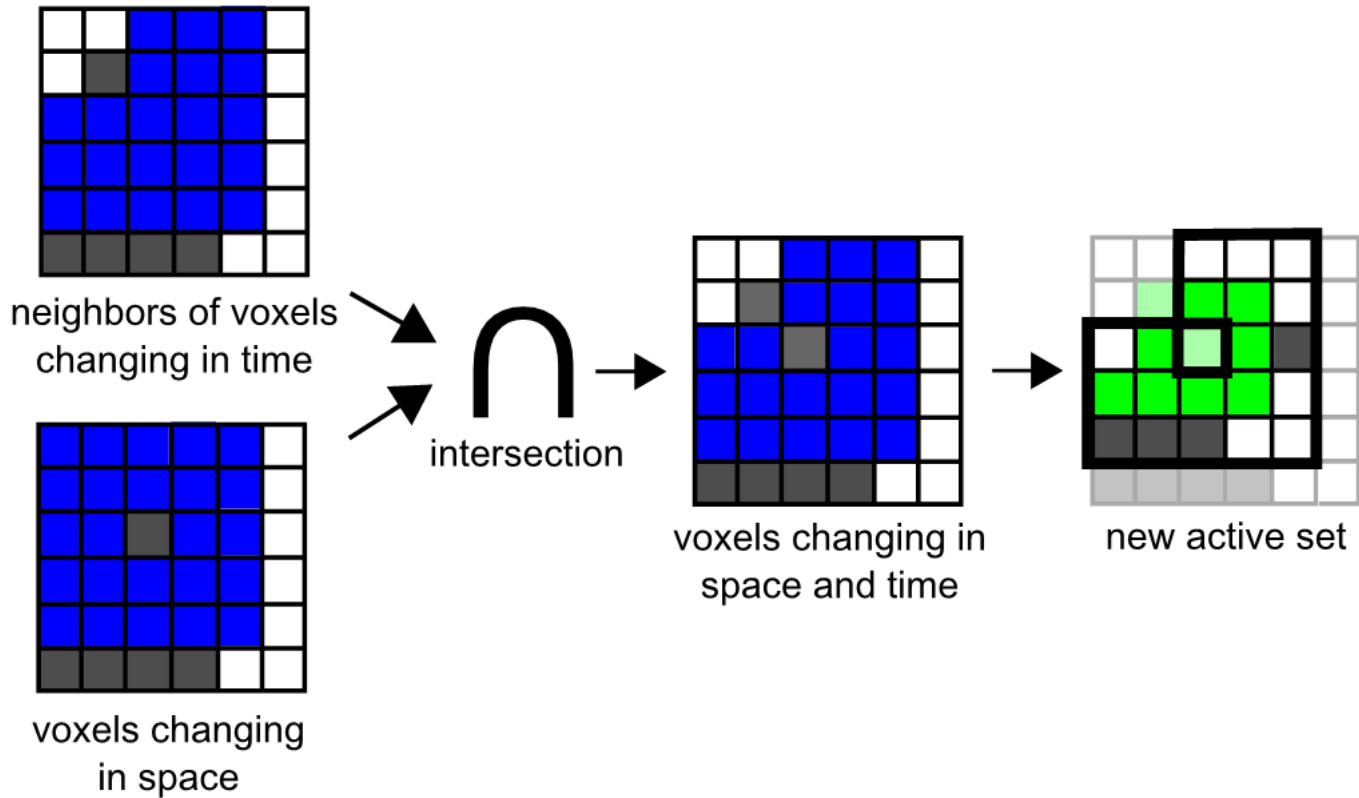
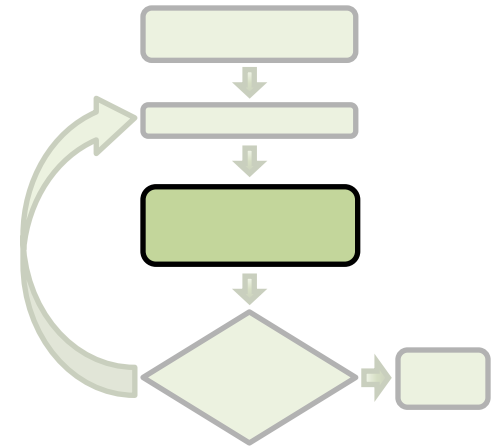


voxels changing
in space

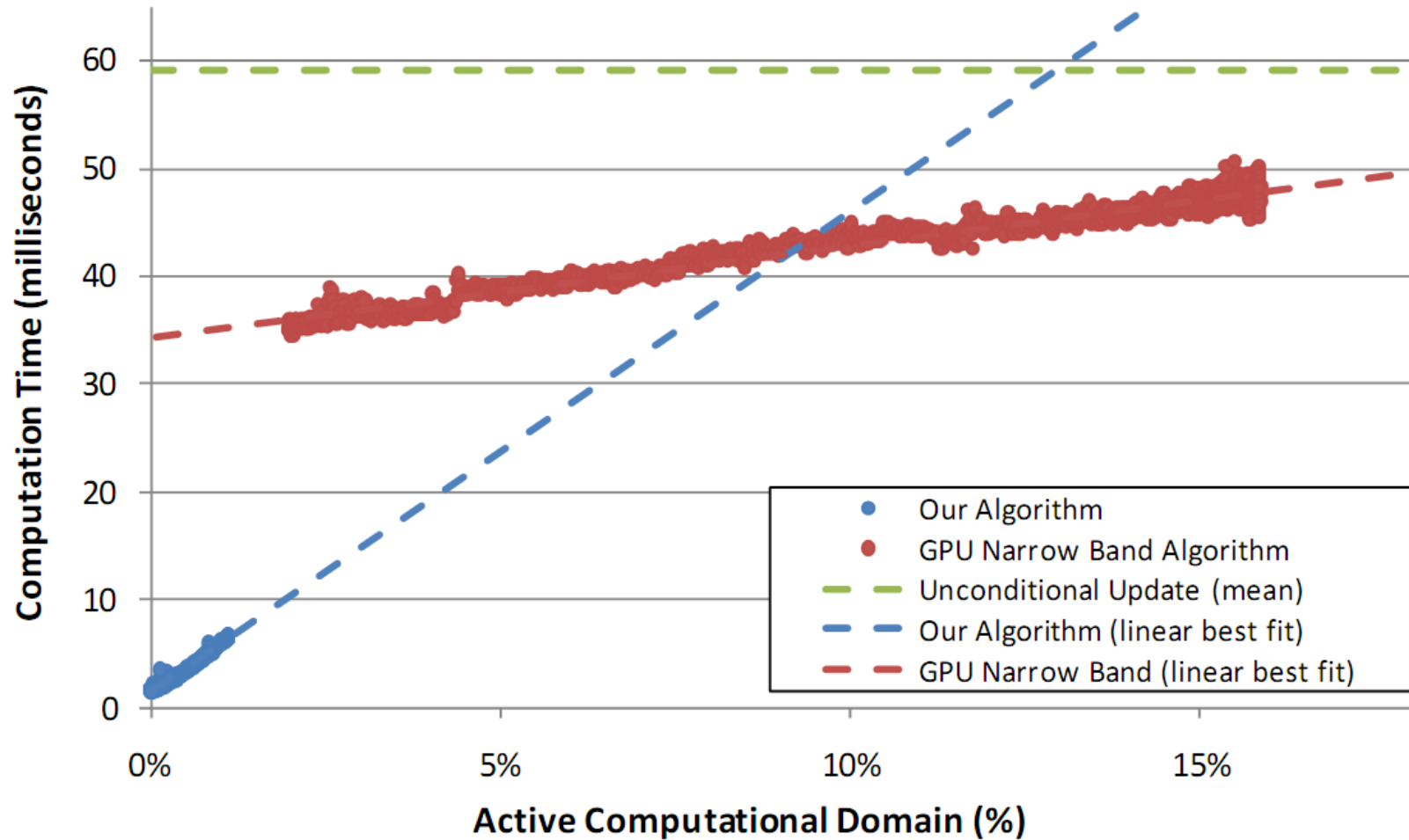
Finding the voxels that are changing in *time*



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



Speed vs. Computational Domain Size



Speed per Subroutine

