A Work-Efficient GPU Algorithm for Level Set Segmentation
Supporting Document

Mike Roberts* Mario Costa Sousa* Joseph Ross Mitchell*

University of Calgary

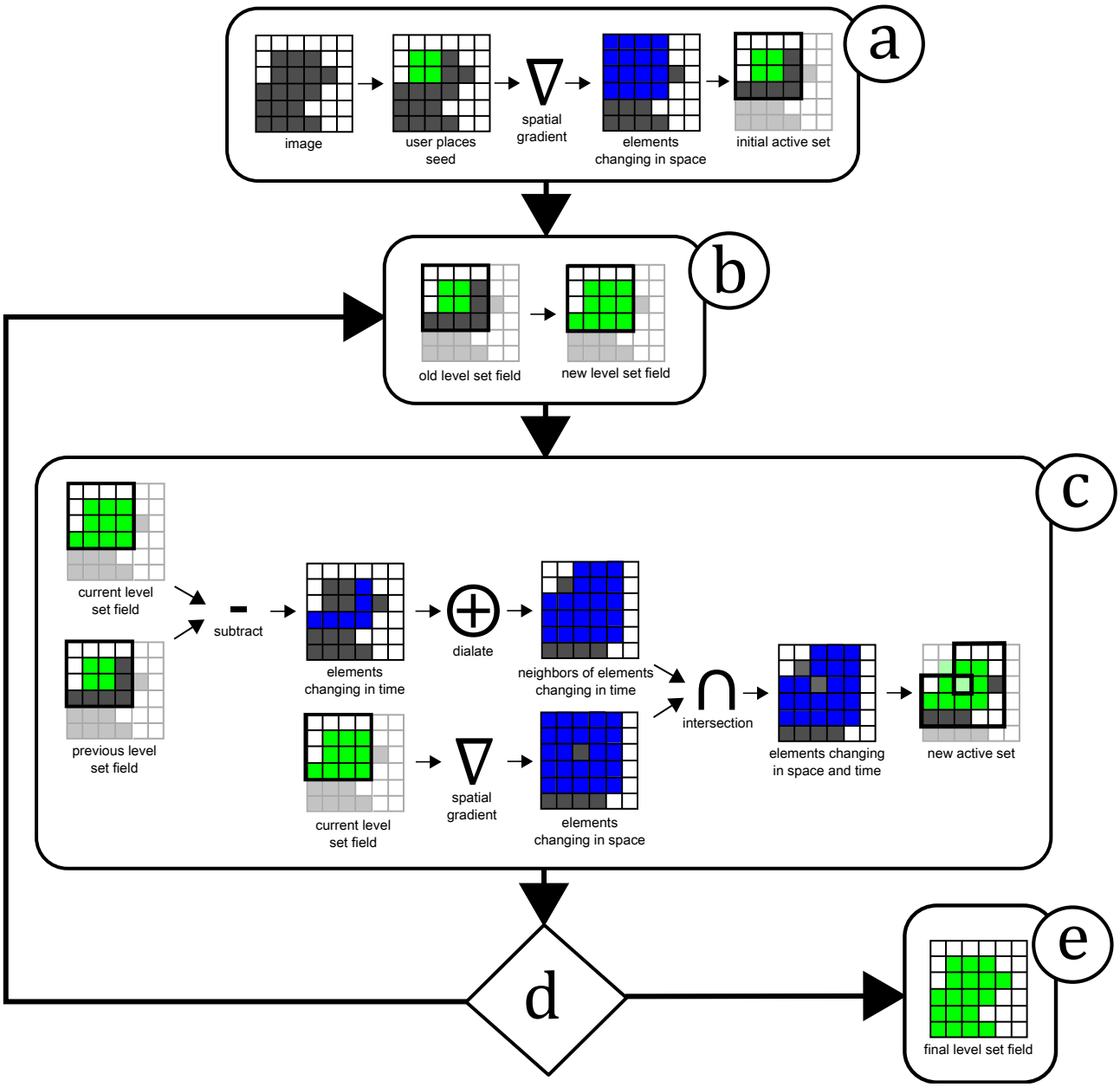


Figure 1: *Our algorithm for tracking the active computational domain. Image data is shown in grey, the currently segmented area in the level set field is shown in green, and intermediate results for computing the active computational domain are shown in blue. The active computational domain is outlined in black, and inactive elements are shown as partially transparent. The user places a seed to initialize the level set field and the initial active set is computed according to the spatial derivatives of the level set field (a). During each iteration the level set field is updated at all active elements (b). The new active computational domain is computed according to the temporal and spatial derivatives of the level set field (c). If the new active computational domain is empty (d) then our segmentation has globally converged (e). Otherwise we go to (b).*

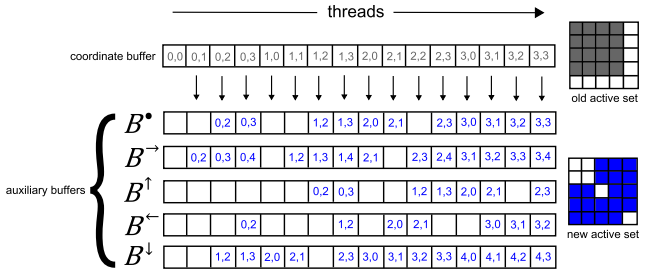


Figure 2: *Generating new active coordinates. There may be duplicate coordinates in the auxiliary buffers when taken collectively. All duplicate coordinates must subsequently be removed (see Figure 3).*

```
1:  for h ← 0 to m in parallel do
2:      v ← V_h
3:      g ← false
4:      for all coordinates n ∈ η(v) do
5:          if φ^read_n ≠ φ^write_n and φ^read_n ≠ φ^read_v then
6:              g ← true
7:              e ← n − v
8:              B^e_h ← n
9:      if g then
10:         B^(0,0,0)_h ← v
```

Listing 1: *Generating new active coordinates. The temporal and spatial derivatives of the level set field are tested on line 5. m is the current size of the active computational domain.*

1 Algorithmic Details

1.1 Assumptions

We define the set $\eta(\mathbf{x})$ to be the set of all elements in the local neighborhood of $\mathbf{x}$. We define m to be the current size of the active computational domain. We assume the level set field ϕ is 3D and the elements in ϕ are 6-connected. Therefore it is guaranteed that for all elements $\mathbf{x} \in \text{Domain}(\phi)$, $\eta(\mathbf{x})$ contains at most seven elements: the 6-connected neighbors of $\mathbf{x}$ and $\mathbf{x}$ itself. We define the set $E = \{(0,0,0), (\pm1,0,0), (0,\pm1,0), (0,0,\pm1)\}$ as the set of offset vectors from a voxel to its 6-connected neighbors.

1.2 Data Structures and Notation

Our algorithm requires three 3D buffers: ϕ^{write} and ϕ^{read} to store the current and previous level set field respectively; and U to use as a scratchpad. Our algorithm also requires eight 1D buffers: V to store the current dense list of active coordinates; and $B^{(0,0,0)}$, $B^{(\pm1,0,0)}$, $B^{(0,\pm1,0)}$, and $B^{(0,0,\pm1)}$ to use as auxiliary buffers when generating new active coordinates. The size of each buffer is equal to $n = |\text{Domain}(\phi)|$. All buffers are initially filled with null values.

We use a subscript notation to refer to individual buffer elements (e.g. V_i refers to the i^{th} element of V; $V_{j\ldots k}$ refers to the range of elements in V from V_j to V_k; and $U_{\mathbf{x}}$ refers to the element of U with the 3D coordinates $\mathbf{x}$).

1.3 Generating New Active Coordinates

We generate new active coordinates as shown in Figure 2 and Listing 1.

*email: {mlrobert,smcosta,rmitch}@ucalgary.ca

1.4 Removing Duplicate Active Coordinates

We make the observation that although there may be duplicate coordinates in the seven auxiliary buffers taken collectively, it is guaranteed that there are no duplicate coordinates in each of the seven auxiliary buffers taken individually. This is because there are no duplicate coordinates in $V_{0\ldots m}$ and for all offset vectors $\mathbf{e} \in E$, either $B^{\mathbf{e}}_i = V_i + \mathbf{e}$ or $B^{\mathbf{e}}_i = \text{null}$ for all array indices i where $0 \leq i \leq m$.

Based on the guarantee in the previous paragraph, we are able to remove all duplicate coordinates in seven passes without requiring any additional sorting or synchronization primitives. We describe this process in Listing 2 and Figure 3.

1.5 Compacting the New Active Coordinates

We compact the seven auxiliary buffers in parallel to produce a new dense list of active coordinates and store the result in V as shown in Listing 2. Since we only ever write to the first m elements of each auxiliary buffer, we only need to compact $7m$ elements in total, rather than compacting the maximum allocated size of each buffer.

After compacting the seven auxiliary buffers, we check if any new active coordinates were compacted into V. If so, we clear $B^{\mathbf{e}}_{0\ldots m}$ for all offset vectors $\mathbf{e} \in E$, update m to be the number of new active coordinates that were compacted into V, and go to our next iteration. Otherwise our algorithm has globally converged on the segmented region contained in ϕ^{read}.

1.6 Compacting the New Active Coordinates

We compact the seven auxiliary buffers in parallel to produce a new dense list of active coordinates and store the result in V as shown in Listing 2. Since we only ever write to the first m elements of each auxiliary buffer, we only need to compact $7m$ elements in total, rather than compacting the maximum allocated size of each buffer.

After compacting the seven auxiliary buffers, we check if any new active coordinates were compacted into V. If so, we clear $B^{\mathbf{e}}_{0\ldots m}$ for all offset vectors $\mathbf{e} \in E$, update m to be the number of new active coordinates that were compacted into V, and go to our next iteration. Otherwise our algorithm has globally converged on the segmented region contained in ϕ^{read}.

2 Evaluation

We compare the accuracy of our algorithm and the GPU narrow band algorithm in Figure 4. We observed that our algorithm was slightly more accurate than the GPU narrow band algorithm with less than 0.2% variability in all experiments. We speculate that this slight accuracy improvement is due to different floating point precision semantics in CUDA and GLSL. The accuracies we observed are comparable to those reported by Lefohn et al.

In Figure 5 we qualitatively compare the results of using only the spatial derivatives of the level set field with the results of using both the spatial and temporal derivatives of the level set field to compute the active computational domain.

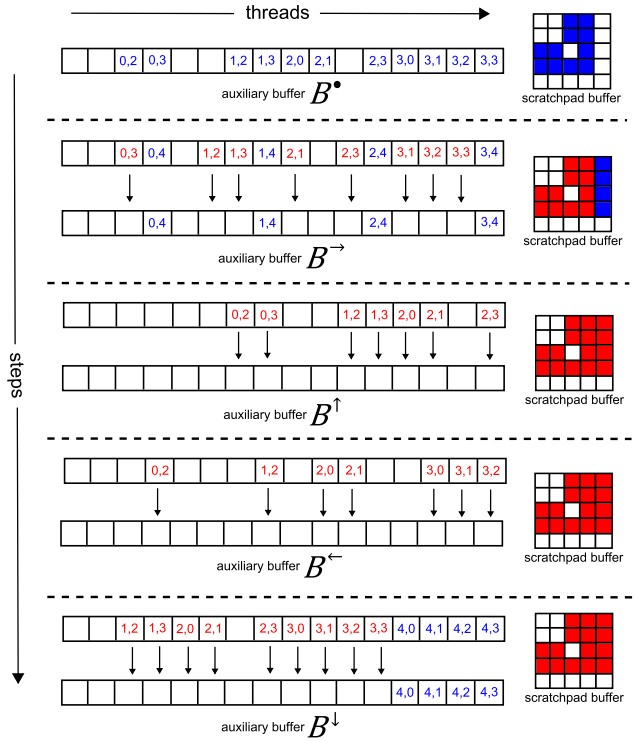


Figure 3: *Removing duplicate coordinates from the auxiliary buffers in parallel without sorting. Coordinates that have not been previously tagged in the scratchpad buffer are shown in blue. Coordinates that have been previously tagged in the scratchpad buffer are shown in red, and are removed from their containing auxiliary buffer. This process is free of race conditions because each step examines one auxiliary buffer and there are no duplicate coordinates within each auxiliary buffer.*

1: $E' \leftarrow E - \{(0,0,0),(0,0,1)\}$
2: **for** $h \leftarrow 0$ to m in parallel **do**
3: $\quad \mathbf{b} \leftarrow B_h^{(0,0,0)}$
4: $\quad$ **if** $\mathbf{b} \neq$ null **then**
5: $\quad\quad U_{\mathbf{b}} \leftarrow$ tagged
6: **for all** offset vectors $\mathbf{e} \in E'$ **do**
7: $\quad$ **for** $h \leftarrow 0$ to m in parallel **do**
8: $\quad\quad \mathbf{b} \leftarrow B_h^{\mathbf{e}}$
9: $\quad\quad$ **if** $\mathbf{b} \neq$ null **then**
10: $\quad\quad\quad$ **if** $U_{\mathbf{b}} =$ tagged **then**
11: $\quad\quad\quad\quad B_h^{\mathbf{e}} \leftarrow$ null
12: $\quad\quad\quad$ **else**
13: $\quad\quad\quad\quad U_{\mathbf{b}} \leftarrow$ tagged
14: **for** $h \leftarrow 0$ to m in parallel **do**
15: $\quad \mathbf{b} \leftarrow B_h^{(0,0,1)}$
16: $\quad$ **if** $\mathbf{b} \neq$ null **and** $U_{\mathbf{b}} =$ tagged **then**
17: $\quad\quad B_h^{(0,0,1)} \leftarrow$ null
18: $V \leftarrow$ **compact** $\left(B_{0\ldots m}^{(0,0,0)}, B_{0\ldots m}^{(\pm 1,0,0)}, B_{0\ldots m}^{(0,\pm 1,0)}, B_{0\ldots m}^{(0,0,\pm 1)}\right)$

Listing 2: *Generating a new dense list of unique active coordinates without sorting the auxiliary buffers. m is the current size of the active computational domain.*

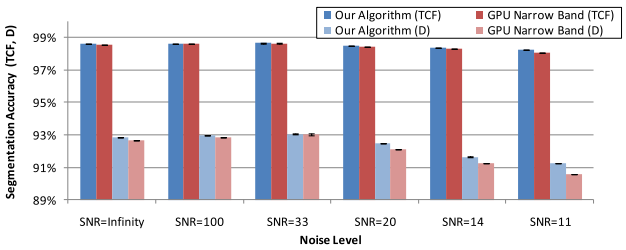
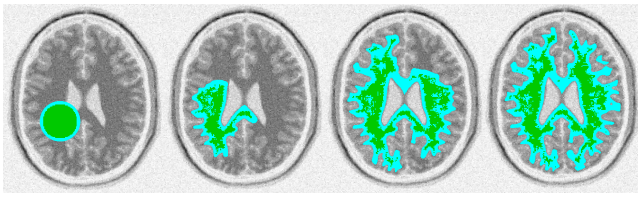
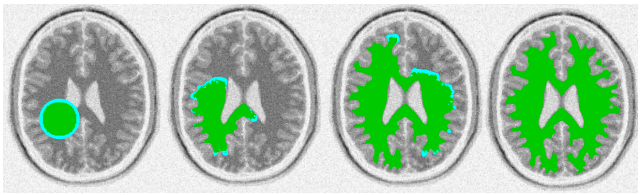


Figure 4: *Accuracy of our algorithm and the GPU narrow band algorithm while performing a set of repeated (N=10) white matter segmentations in a 256^3 head MRI with varying signal-to-noise-ratio (SNR) values. For each segmentation we used a randomly selected seed point and we measured the Dice Coefficient (D) and Total Correct Fraction (TCF). These accuracy measurements are comparable to those reported by Lefohn et al. We speculate the slight difference in accuracy between our algorithm and the GPU narrow band algorithm is due to different floating-point precision semantics in CUDA and OpenGL.*



(a)



(b)

Figure 5: *The progression of the active computational domain (shown in blue) while segmenting the white matter in a 256^3 head MRI. In (a) the active computational domain is determined according to the spatial derivative of the level set field. In (b) the active computational domain is determined according to the spatial and temporal derivatives of the level set field. In (b) regions that have locally converged are immediately marked as inactive, resulting in a much smaller active computational domain, and the size of the active computational domain drops to zero when the segmentation has globally converged.*