Graph Abstraction for Simplified Proofreading of Slice-based Volume Segmentation

Ronell B. Sicat KAUST
Markus Hadwiger KAUST
Niloy J. Mitra University College London

Abstract

Volume segmentation is an integral data analysis tool in experimental science. For example, in neuroscience, analysis of 3D volumes of neural structures from electron microscopy data is a key analysis step. Despite advances in computational methods, experts still prefer to manually ***proofread*** *and correct the automatic segmentation outputs. Such corrections are often annotated at the level of data slices in order to minimize distortion artifacts and effectively handle the massive data volumes. In absence of crucial global context in 3D, such a workflow remains tedious, time consuming, and error prone. In this paper, we present a simple graph-based abstraction for segmentation volumes leading to an interactive proofreading tool making the process simpler, faster, and intuitive. Starting from an initial volume segmentation, we first construct a graph abstraction and then use it to identify potential problematic regions for the user to investigate and correct spurious segmentations, if identified. We also use the graph to suggest automatic corrections, thus drastically simplifying the proofreading effort. We implemented the proofreading tool as an Avizo© plugin and evaluated the method on complex real-world use cases.*

1. Introduction

Segmenting volumetric data is at the heart of many data analysis tasks in science. Given the importance of reliable segmentation in such analysis tasks, practitioners still prefer to manually investigate and correct the output of automatic segmentation algorithms in a subsequent *proofreading* stage. For example, the extraction of 3D geometric representations of neurons from electron microscopy (EM) images is an important task in neuroscience, especially in connectomics [LD11] where the goal is to map out all neural connections of the mammalian brain. A typical pipeline [CVS10] involves registration, segmentation, linkage, proofreading, and annotation. Despite advances in automatic registration and segmentation techniques, the segmentation outputs are typically manually proofread by experts. Even for moderate volumes of data, e.g., 10^8-10^9 voxels, the process is extremely tedious and can take several hours.

Proofreading involves two main steps: (i) *searching* for slices that contain segmentation errors, and (ii) *correcting* the segmentation errors, typically by splitting or merging segments. With increasing access to complex volumetric data, the proofreading step is becoming a real bottleneck, where proofreading can at times take longer than just manually segmenting an entire neuron [PLZM11]. The process is cumbersome as most proofreading tools only allow users to operate at the level of individual slices, without any global 3D context. This makes searching for segmentation errors tedious. There is only little support to facilitate such proofreading efforts [JST10]. Peng et al. [PLZM11] provide intuitive 3D tracing and proofreading tools, but the 3D segmentation volumes can become complex to navigate, especially in complex datasets with many intertwined structures.

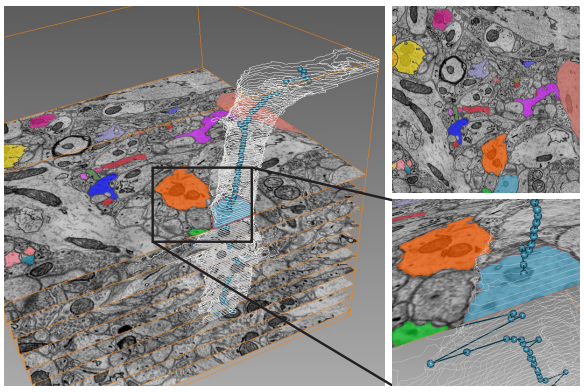

Figure 1: *(Left) Given a stack of volume slices with segmented regions (e.g., top-right shows a single slice with segments highlighted), we construct a graph abstraction by establishing edges between segments from neighboring slices. The graph gives a very low-cost approximation to the skeletons of the connected segment volumes.*

In this short paper, we propose a simple 3D graph-based segmentation abstraction to make proofreading easier, faster, and more intuitive. Given an initial slice-level volume segmentation, we first construct a graph, where each node denotes a segment area and each edge connects two corresponding nodes across neighboring slices. Here, we focus on volumetric data of neural structures and hence a connected component in the graph represents a single neural structure in the data (see Figure 1). In addition to storing a boundary contour at each node, we compute a measure of segmentation consistency to indicate potential artifacts.

We use the graph abstractions to detect potentially inconsistent segmentations, propose automatic corrections, and highlight uncertain regions to allow the user to further refine the segments. We demonstrate our framework on a representative dataset [KFB10] obtained using electron microscopy of a mouse cortex (datasize: $1024 \times 1024 \times 150$).

Related works. Kaynig et al. [KFB10] formulate the linkage problem in their 3D neuron reconstruction pipeline as partitioning an edge-weighted graph into connected components to group segmented regions belonging to the same neural structure. They also construct a graph where nodes in one slice are connected by weighted edges to all nodes in the adjacent slices. Edge weights denote if the nodes belong to the same neural structures. Unlike their approach, instead of discarding the graph after the initial reconstruction, we demonstrate that a modified version simplifies the proofreading of the segmentation results. In the context of man-made objects, Mehra et al. [MZL*09] use curve networks to abstract objects, while Li et al. [LLZM10] propose arterial snakes to simultaneously abstract the topology and geometry of the curve-like 3D objects.

Earlier, Sherbondy et al. [SHN03] used hardware assistance to interactively view the current 3D segment volumes to assist interactive segmentation. In contrast, we present a lightweight abstraction suitable to assist manipulation of very large datasets while still providing necessary searching and correction support for assisted proofreading.

2. Graph Construction

Given a stack of segmented images $I_1, I_2, \ldots$, we first construct a graph abstraction. We go through each slice and compute the boundary contour of each of its segmented regions using iso-contour extraction. For each such closed segmented region R_i in the i-th slice, its center of mass $c_{R_i} \in \mathbb{R}^3$ becomes a node in the embedded graph. We connect pairs of nodes from adjacent slices if the projections of the respective segments overlap, i.e., $R_i \cap \Pi(R_{i+1}) \neq \emptyset$, where $\Pi(R_{i+1})$ denotes projection of a segment from the $(i+1)$-th slice to the i-th slice. Figure 1 shows an example abstraction graph. Although more advanced versions of graph construction can be used [KFB10], we found such a simpler construction to be sufficient given the high slice density of the input data.

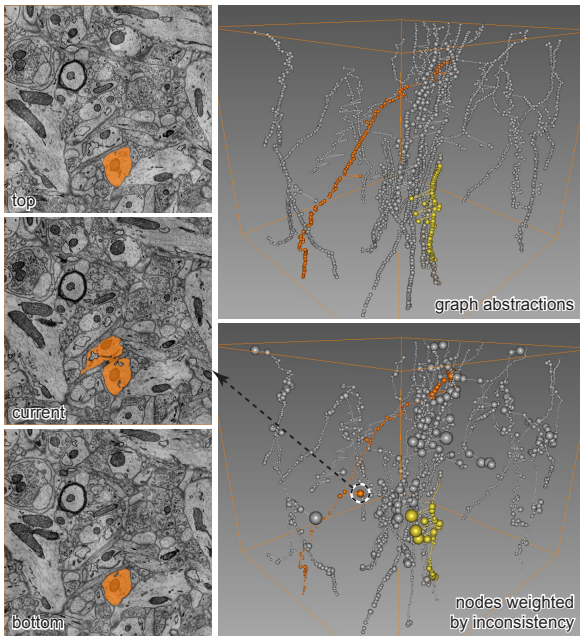

Figure 2: *A raw graph abstraction (top-right) does not immediately highlight potential erroneous regions. We weigh the nodes based on their local inconsistencies (bigger nodes denoting higher inconsistency) to quickly identify regions to focus on. (Bottom-right) Here, we highlight two threads. The user then selects a node and we show the relevant segmentation slices — the subfigures (left) show three neighboring slices according to the highlighted orange node.*

Once we compute the graph from all the segmented regions in the input data, we use it to steer visualization of the segmented neural structures (see Figure 2 top-right). We call each connected component of the graph a *thread*, denoting abstraction for the respective segmentation volume. The graph provides a preview of the size (based on the lengths of the threads), orientation (based on the traces of the threads), and density (based on the density of the threads) of the detected neural structures in the data.

Besides providing a quick data preview and an exploration handle to navigate through the segmented input slices (see supplementary video), the graph also provides important clues regarding the quality and consistency of the input segmentation. We make two observations: (i) smooth threads denote that the segments evolve continuously across the slices; and (ii) sharp changes or jagged parts of threads indicate sharp segmentation changes across neighboring slices (see Figure 2). Typically, such jagged features indicate wrong segmentations, which can easily be corrected using information from neighboring slices. Based on these observations, we introduce proofreading tools, as described next.

3. Proofreading Tasks

Error visualization and user guidance. Input slices that have been segmented individually can often contain inconsistencies. To support the proofreading process, we use our graph abstractions to quickly locate and investigate such segmentation inconsistencies, which appear as sharp node transitions in the threads of the graph (see Figure 2). Empirically, we observed that such sharp node transitions are either due to missing regions or holes; or incorrectly included regions or extensions (see Figure 3). Building on this observation, we compute an inconsistency weight for each node to capture the smoothness of the node transitions and the overlap between segments from adjacent slices. Specifically, E_{R_i} for the node corresponding to the region R in slice i is given by $E_{R_i} := \alpha D_{R_i} + (1-\alpha) O_{R_i}$, where D_{R_i} is the error weight measured by node distances, O_{R_i} is the error weight measured by region overlaps, and $\alpha \in [0,1]$ specifies relative weights ($\alpha = 0.5$ as default). For O_{R_i}, for each adjacent node of R_i, say R_{i-1}, we first compute the sum of the pixels in the two regions $P_{R_i} + P_{R_{i-1}}$ and the number of overlapping pixels $2 \times \#(R_i \cap R_{i-1})$ (the multiplication by two counts the common pixels from both regions). The error weight contributed by the overlap with one adjacent region is then measured as the ratio of the number of non-overlapping pixels from both regions to their total number of pixels, i.e., $\frac{P_{R_i}+P_{R_{i-1}}-2\times\#(R_i\cap R_{i-1})}{P_{R_i}+P_{R_{i-1}}}$. We then take the average of these weights for all adjacent regions. Specifically, for R_i with adjacent regions R_{i-1} and R_{i+1},

$$D_{R_i} = \min\left(1, \frac{\|c_{R_i} - c_{R_{i-1}}\| + \|c_{R_i} - c_{R_{i+1}}\|}{2\beta}\right),$$

where the c_{R_i} are the centers of mass (see above), β is a scale factor (we use 10% of the image diagonal), and

$$O_{R_i} = \frac{\frac{P_{R_i}+P_{R_{i-1}}-2\times\#(R_i\cap R_{i-1})}{P_{R_i}+P_{R_{i-1}}} + \frac{P_{R_i}+P_{R_{i+1}}-2\times\#(R_i\cap R_{i+1})}{P_{R_i}+P_{R_{i+1}}}}{2}.$$

We then use the inconsistency score of each node to scale the sphere that represents the node in the graph visualization (see Figure 2). Insets of the segmented regions of the current node with high inconsistency (as indicated by the large node sizes in the graph visualization) and its neighboring slices are also displayed. Such a graph provides a quick preview of the segmentation data to facilitate proofreading and allows the users to focus on potentially erroneous regions by visual inspection or by providing the user with a list of nodes sorted by their inconsistency scores.

Correcting erroneous segmentations. Subsequently, the user selects a node to apply corrections. Whenever the user selects such a node, we show the slicing plane with the segmented region overlayed on top of the original EM image. We also highlight the segmented regions of the current node and their neighboring nodes and display them on three separate panels to help the user identify the problematic areas.

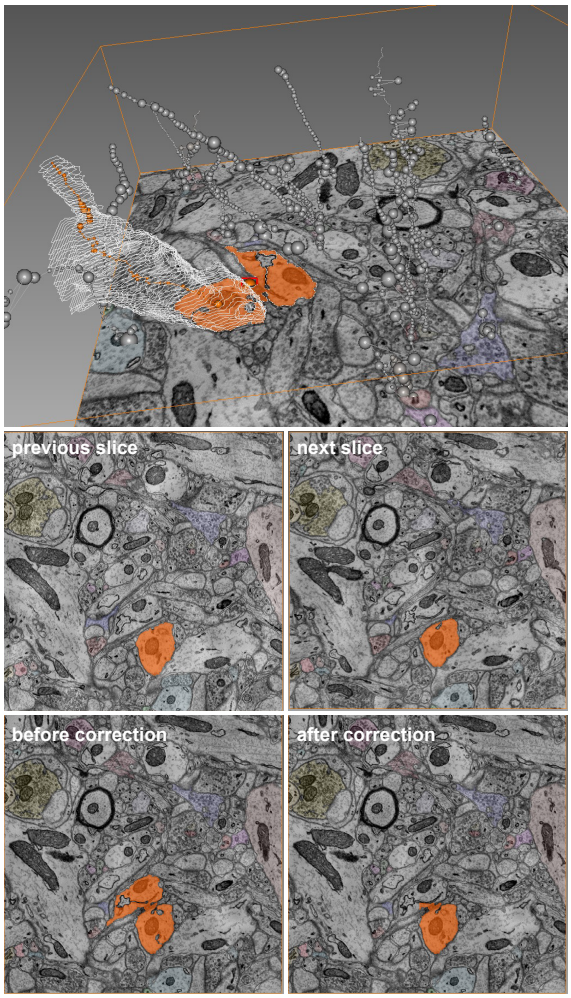

Figure 3: *(Top) Jagged corners of graph threads help to quickly identify potentially erroneous regions. We use segmentation information from the neighboring slices (middle row) to automatically propose corrections to the segmentations (bottom row).*

The user can then manually correct the erroneous segmentation using Avizo's built-in Segmentation Editor while focusing on the region of interest automatically set to the target region. Furthermore, the user can use a brush tool to remove or add pixels in the segmented region.

In order to simplify the correction procedure, we also provide an automatic proofreading function, which tries to correct a node's segmented region using our previous assumptions on the data (smooth shape transitions among adjacent slices). The process runs in two steps: (i) the average bounding box of the regions above and below the target region are used to clip out extensions or incorrectly included pixels. While our implementation uses rectangular bounding boxes for simplicity, complex polygons (e.g., convex hulls) could be used for better results; (ii) holes in the target region are

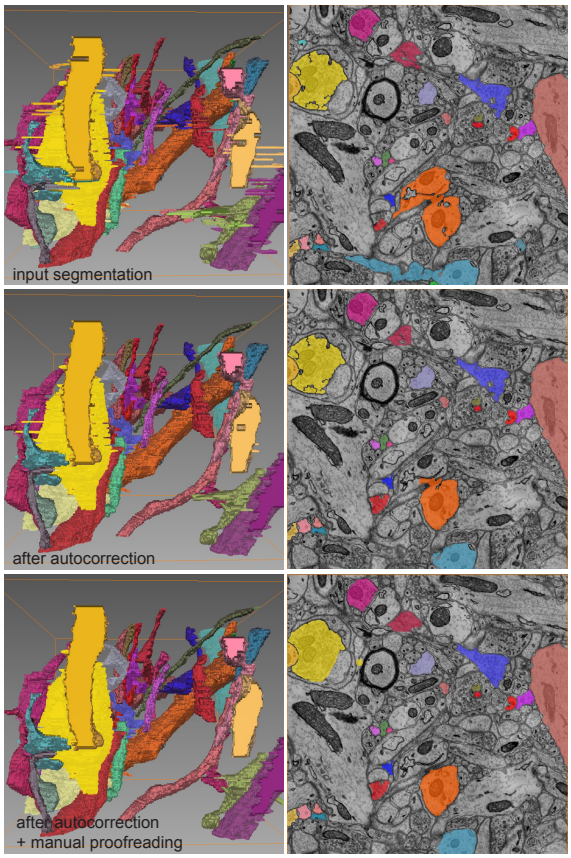

Figure 4: *Starting from input segmentations (top), we first apply autocorrection (middle), and then the user manually refines the segments using the proposed graph-based exploration tool to produce a final volume segmentation (bottom). The proofreading took less than 5mins. with our tool in contrast to 1-2 hours in existing workflows (c.f., [PLZM11]).*

filled by processing the pixels in the new region bounding box and including a pixel in the target region (filling a missing pixel) if the corresponding pixel exists in the neighboring regions, i.e., the current pixel $p_{R_i} \in (R_{i-1} \cap R_{i+1})$. While this simple automatic proofreading function does not guarantee perfect corrections, it drastically reduces the manual effort requiring the user to refine only regions that are still ambiguous (see Figures 3 and 4).

Automatic proofreading. The user can apply the automatic correction described above on all the nodes in the graph, thus decreasing the amount of manual proofreading effort needed. Specifically, the user can go through each connected component of each neural structure and apply the automatic proofreading function on each node in order of decreasing inconsistency, if the node inconsistency score is above a user specified threshold (0.15 in our tests). Alternatively, the user can focus on a single neural structure by applying this step only on the respective connected component.

The process can be repeated several times since a single run may still leave some large errors, with the user always having the option to intervene with manual corrections. Note that the automatic proofreading process is different from simply smoothing the segmentation volumes (e.g., Laplacian smoothing), which can lead to spurious segments that drastically affect the already-correct neighboring segments.

We tested our framework on a dataset of moderate complexity ($1024 \times 1024 \times 150$). The graph construction took about 8 minutes for 44 segments (total of 2892 nodes), using an unoptimized plugin implementation in the Avizo framework. Subsequent exploration, interaction, and correction ran at near-interactive rates (see supplementary video).

4. Conclusion

We have introduced a simple graph as an intuitive abstraction for 3D volumetric segmentation data that can be used to assist proofreading of slice-based segmentations, and presented initial results obtained using our tool on a real brain tissue dataset. We have shown how the graph can be used to visualize neural structures, help users to focus on regions with possible errors, apply proofreading operations on segmented regions, and reduce the overall proofreading effort needed for the dataset. In the future, we will focus on robust construction of the graph abstractions and investigate further graph-assisted exploration and correction tools.

5. Acknowledgements

We thank Verena Kaynig for her help and the dataset, and our collaborators at the Harvard Center for Brain Science.

References

[CVS10] CHKLOVSKII D. B., VITALADEVUNI S., SCHEFFER L. K.: Semi-automated reconstruction of neural circuits using electron microscopy. *Neurobiology 20*, 5 (2010), 667–75. 1

[JST10] JAIN V., SEUNG H. S., TURAGA S. C.: Machines that learn to segment images: a crucial technology for connectomics. *Current opinion in neurobiology 20*, 5 (2010), 653–66. 1

[KFB10] KAYNIG V., FUCHS T. J., BUHMANN J. M.: Geometrical consistent 3d tracing of neuronal processes in sstem data. In *Proc. MICCAI* (2010), pp. 209–216. 2

[LD11] LICHTMAN J. W., DENK W.: The big and the small: Challenges of imaging the brain's circuits. *Science 334*, 6056 (2011), 618–623. 1

[LLZM10] LI G., LIU L., ZHENG H., MITRA N. J.: Analysis, reconstruction and manipulation using arterial snakes. *ACM Trans. Graph. 29*, 6 (2010), 152:1–152:10. 2

[MZL*09] MEHRA R., ZHOU Q., LONG J., SHEFFER A., GOOCH A., MITRA N. J.: Abstraction of man-made shapes. *ACM Transactions on Graphics 28*, 5 (2009), #137, 1–10. 2

[PLZM11] PENG H., LONG F., ZHAO T., MYERS E.: Proof-editing is the bottleneck of 3d neuron reconstruction: The problems and solutions. *Neuroinformatics*, 9 (2011), 103–5. 1, 4

[SHN03] SHERBONDY A., HOUSTON M., NAPEL S.: Fast Volume Segmentation With Simultaneous Visualization Using Programmable Graphics Hardware. *IEEE Visualization* (2003). 2