ACM Symposium on Solid Modeling and Applications (2004)
G. Elber, N. Patrikalakis, P. Brunet (Editors)

Compression, Segmentation, and Modeling of Filamentary Volumetric Data

Bruce H. McCormick, Brad Busse, Purna Doddapaneni, Zeki Melek, and John Keyser

Department of Computer Science, 3112 Texas A&M University
College Station, TX, 77843-3112 USA

Abstract:
We present a data structure for the representation of filamentary volumetric data, called the L-block. While the L-block can be used to represent arbitrary volume data sets, it is particularly geared towards representing long, thin, branching structures that prior volumetric representations have difficulty dealing with efficiently. The data structure is designed to allow for easy compression, storage, segmentation, and reconstruction of volumetric data such as scanned neuronal data. By "polymerizing" adjacent connected voxels into connected components, L-block construction facilitates real-time data compression and segmentation, as well as subsequent geometric modeling and visualization of embedded objects within the volume data set. We describe its application in the context of reconstruction of brain microstructure at a neuronal level of detail.

Categories and Subject Descriptors (according to ACM CCS): I.3.5[Computer Graphics]: Curve, Surface, Solid, and Curve Generation, I.4.10[Image Processing and Computer Vision]:Image Representation – Volumetric, J.3[Computer Applications]:Biology and Genetics

1. Introduction

1.1 Motivation

Recent advances in biomedical data capture have highlighted inadequacies in existing volumetric modeling approaches. Many of the existing modeling techniques are not suited for the nature of the data being captured or the analysis that must be performed on the data. As the collection of this data becomes more common, the need for appropriate tools to model and analyze the data becomes more important.

The work presented in this paper describes a volumetric data structure that is suitable for modeling within this newer environment. Our work is particularly motivated by our attempts to scan and reconstruct stained brain tissue at a neuronal level of detail. New data collection techniques allow this data to be collected at rates far exceeding those previously possible. Furthermore, existing modeling techniques have difficulty representing the shape of the objects being reconstructed in a fashion that supports analysis.

To give greater context to our problem, we briefly describe data collection and the distinguishing features of our data sets and analysis needs.

1.2 Data Collection

The Knife Edge Scanning Microscope (KESM) is a unique instrument developed at Texas A&M for the collection of volumetric data from brain tissue embedded in plastic [McC02]. The KESM uses knife-edge scanning, where a diamond knife acts as both a cutting tool and an optical element (directing the light), and tissue is scanned while being cut. A line-scan camera is used to record at the optical limit a linear array of pixels seen through a microscope, which focuses on the area just behind the cutting edge of the knife. As successive sections of tissue are taken deeper in the specimen, a volumetric data set is formed from the "stacks" of images. The imaging can be in either grayscale or color, depending on the camera.

When fully operational, an entire mouse brain scanned by the KESM would yield approximately 30 terabytes of raw volumetric data at the limit of optical resolution. . A small portion of one section imaged from test scans of KESM is shown in **Figure 1**.

Other 3D microscopes currently under development have the potential to produce terabytes of data; our work could easily be applied to these data sets.

1.3 Distinguishing Features of Our Data

Among the features that distinguish our data from more common biological volumetric data sets are:

- The full volume data set can be extremely large. Raw data set sizes can reach into terabytes.
- The data of interest within the volume data sets (i.e., the stained neuronal tissue) tends to be sparse, taking up only a modest portion of the overall volume.
- Segments of the neurons to be modeled have a very long but thin (as opposed to blobby) structure.
- The neurons have a high degree of branching.

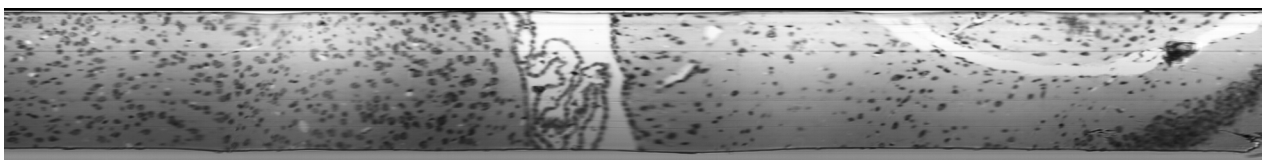

Figure 1. A portion of a section from the KESM. A Nissl stain was used, so only cell bodies are stained. The section shown is only 500 pixels by 5500 pixels, representing a tiny fraction of the overall brain.

1.4 Requirements of Analysis

Analysis of the neuronal data also varies from traditional volumetric analysis requirements. Due to the large volumetric data size, it is not feasible to maintain the entire volumetric region in main memory at once. Only a few sections of raw volume data can be kept in memory at any point in time. The real-world data tends to be noisy, with tiny, false "specks" appearing throughout the volume. We need to be able to identify and delete these easily. Another problem is the very high data acquisition rate. We need to be able to compress and store data rapidly. For the same reason, we would like to be able to exploit parallelism in processing the data.

Random access to the individual voxels from the entire volume is *not* a necessary feature. Due to the amount of data and its acquisition rate, we are interested in accessing only voxels "connected" to some group of voxels.

We need to be able to easily "thread" data – find out how data is connected to other regions, as well as capture gross shape properties of the data. Thus, we need a representation scheme that is geometrically meaningful. And finally, like most methods, we need interactive display routines to assist visualization.

2. Previous Work

A number of data structures can be used to describe volumetric solids. A current and detailed summary of the most important of these methods is given by Winter [Win02].

The simplest form of describing volumetric solids is spatial-occupancy enumeration, where the individual voxels of interest are listed and stored individually in a fixed, regular, rectilinear 3D array. To address the excessive overhead and cumbersome operations on such structures, the octree, uses a hierarchical spatial-occupancy approach [JT80] [Sam84] [SW88] [SW288]. It provides a consistent way to recursively subdivide the modeling space until all voxel elements are homogeneous. The octree suffers from a high pre-computation cost and cannot be incrementally constructed, which is necessary for the acquisition of neuronal data.

Other spatial subdivision approaches include the BSP-tree [FKN80]. This approach recursively divides space by an arbitrary plane at each level, usually dividing the remaining points equally. Being a spatial subdivision approach, the BSP-tree is forced to define planes based on the decomposition of space, rather than the structure of the object itself. Similar to BSP-trees, kD-trees [OV82] also subdivide space recursively, but at each level the (axis aligned) direction and exact position of the plane can be chosen. They suffer from the same shortcomings as BSP trees.

Axis-aligned bounding box trees (AABB-trees) [van97], more commonly used in collision detection, are used by our data structure. These trees consist of a hierarchical collection of iso-rectangular boxes, each bounding the boxes of child nodes. The key difference in our approach is that the enhanced data allows us to easily build boxes incrementally and to maintain connectivity between nodes without having to go through a parent node.

We have found these and other volumetric representation techniques to be deficient in addressing at least one of the features we care about. Due to the potential data size, methods that keep the entire volume in memory at once are unrealistic. Several methods (such as the octree) are poorly suited for modeling long, thin structures. Medial-axis methods, while good for representing neurons, tend to process too slowly and can require too much data to be stored in memory. Pure image and video compression techniques work well for compression, but fail to give any meaningful insight into the geometric structure of the objects to be modeled.

3. Representation and Operations

3.1 Enhanced volume data sets

Allowing data compression in real time in a manner that facilitates subsequent segmentation of the volume data set is one of our main concerns. We also need to provide data compression and segmentation strategies that exploit the efficiencies of examining successive serial images, yet are independent of the axis chosen for serial sectioning. Separation of segmentation from both geometric modeling and visualization of the identified objects in the volume data set is required.

We define an *enhanced volume data set* (EVDS) as follows: in addition to the value assigned to every vertex (voxel) of the grid, selected edges between vertices of the grid are given a Boolean label of 1 for *active* edges and 0 for *inactive* edges. This enhancement alone can aid in topological analysis of the relevant data [KR89]. Edge labeling is used to provide independent information about whether two vertices sharing a common active edge belong to the same underlying object. Far more advanced methods for describing digital topology exist [EHV*03]; we are capturing only basic connectivity between components at a local level.

It is important to note that the decision function used to assign the Boolean values is essentially the segmentation process, and is of primary importance in determining how faithful a particular segmentation or reconstruction is. We have developed and experimented with a variety of decision functions, generally based on statistical analysis of large-scale scanned regions, and analysis of prior reconstruction by manual editing.

Any enhanced volume data set (in three dimensions) can have many representations. Most useful for our purposes is an assignment at each vertex, in addition to the voxel value, of a Boolean vector indicating the activity level of the edges emanating from the vertex. Vertices at the boundary of the grid may lack some edges; we treat these as inactive.

The number of edges emanating from any one vertex is referred to as the connectivity level. Placing edges in the axis directions (i.e. (i,j,k) is connected to $(i+1,j,k)$, $(i,j+1,k)$, and $(i,j,k+1)$) gives 3-connectivity. Imagining an axis-aligned cube around the vertex, 3-connectivity would give connections across each face. Connections across the edges as well would yield 9-connectivity, while including the corners in addition would yield 13-connectivity. A vertex in an EVDS with 3-connectivity can be thought of as having "links" that extend to neighboring vertices along the three axes. Thus it behaves somewhat like a Lego® block, with connections possible along 3 axes.

Given an EVDS, "whitespace" is defined as vertices that do not satisfy some threshold test. Often this threshold test is equivalent to determining whether any of the edges leading to or from the vertex are active.

3.2 L-blocks

An *L-block* is defined as a 3-dimensional iso-rectangular block of enhanced vertex information. An (l_1, l_2, l_3) L-block refers to a block of l_1 vertices in the x-direction, etc. Each L-block includes both a header and a vertex block. The header defines both the *position*, e.g. (x,y,z), of its least vertex and its *template* (l_1, l_2, l_3). The vertex block contains the enhanced vertex information (voxel

value(s) and edge labels) for all voxels in the isorectangular block defined by the header.

The L-block as a whole can be visualized as a block of vertices, with extensions that demonstrate connectivity (see **Figure 2**).

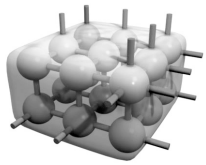

Figure 2. A (3,3,2) L-block. Cylinders represent active edges emanating from the L-block.

3.3 Notation

Hereafter, we will use the following abbreviations. An L-block will be referred to as an LB, with the template optionally given immediately beforehand. For example, a single voxel could be described by a (1,1,1)LB. LBs sharing an active edge are assumed to be *connected*. We store connected LBs in a hierarchical data structure, similar to an AABB-tree, that we refer to as an L-block covering, which we abbreviate LBC.

3.4 The polymerization strategy

The *polymerization strategy* refers to the construction of an enhanced data set stored as an LBC to encompass an object of interest within a given volume. This strategy will be successful to the extent that the data-dependent edge-labeling function captures the connectivity of the underlying physical objects in the scanned block. In practice, we usually use a conservative labeling function initially, allowing us to quickly segment and compress a superset of the critical data. Later, more sophisticated (and slower) techniques can be applied to these initial LBCs in order to adjust the edge labeling.

Focusing on connected components in the extended volume data set and efficiently packaging these within LBCs can significantly compress an EVDS. Isolated vertices outside the connected components may also be retained as individual LBs. The remaining vertices outside these coverings are treated as "white space", and ignored in subsequent image processing. Given an EVDS then, polymerization lets us retain only the data we care about, allowing for significant compression.

Volume data generated by serial sectioning and scanning of a three-dimensional specimen can be compressed in real time by incrementally generating the EVDS. As each consecutive image is scanned, only its immediate predecessor needs to be retained in memory, while the current image data is enhanced and incrementally added to the evolving EVDS. For example, let consecutive serial sections be scanned in the XY plane at depths of Z and Z+1 respectively. The Z+1-plane image data is used to enhance the Z-plane image data. Regions-of-interest in the Z-plane image are then packaged in (m n 1) L-blocks and added to the evolving compressed representation of the EVDS. This is a key advantage of the LBC approach in that it allows us to process, compress, and (coarsely) segment data on the fly based on only a local set of data.

4. Operations with LBCs

4.1 LBCs as Geometric Superstructures

L-block coverings can be viewed as a geometric superstructure, encapsulating several other common volumetric representations. These include grid-sampled data, enumerated voxels, octrees, BSP-trees, kD-trees, and AABB-trees (see section 2). LBCs can be used to describe these structures, with the same algorithmic benefits, but possibly at an increased storage cost. We refer the reader elsewhere for the details of this encapsulation [MBM*02].

4.2 KESM Specific Operations

The most important motivation behind the LBCs is the huge data flow in the KESM pipeline. The amount of data we have does not allow us to generate data structures that keep all the data in memory. LBs can be generated by processing only two consecutive layers in memory, and filtering and thresholding on the fly, while the next layer is scanned. Other data structures are also suitable for on the fly generation using only partial data, but their implementations are rather complex, whereas the LB generation is straightforward and extremely simple to implement. For more details see section 5.1.

Filtering and thresholding in image space can detect 2D noise but there is still 3D noise left in the volumetric data. This 3D noise arises from scanning limitations as well as staining artifacts. The LB structure allows us to easily detect 3D noise by looking for small isolated blocks. Since LBC is a local packaging structure, noise will be wrapped in small isolated packages. For more information, refer to section 5.2.

Since LBs consist of axis-aligned boxes, finding neighboring LBs within LBCs is fast. However, since two neighboring LBs might not be connected, the EVDSs must be checked to determine whether a connection actually exists.

Threading is described in section 5.3. The LB data structure allows us to work with blocks of data for initial thread generation, rather than with the pure volumetric data, making our algorithms run faster. The same is true for estimating thread thickness. While other data structures might have faster random access, their algorithm complexity can become a limiting factor.

4.3 General Operations

A pure LB data structure is not as convenient for random access as other commonly used volumetric data structures. However, our hierarchical LBCs can give better performance, giving the same access times as other hierarchicies. Still, random access may be slower than for a spatial subdivision approach. On the other hand, in the KESM pipeline, random access is not one of the required operations.

Isosurface calculation can be easily used for visualization.. Here we can use LBCs to our advantage, generating isosurface pieces per LBC, which is largely parallelizable. In our current implementation, we have used isosplats [CHJ03] for interactive isosurface generation and display. The surface continuity to the neighboring LBs is guaranteed by using the connectivity information already generated in the EVDS. The hierarchical LBC also helps performance if raytracing is used for visualization.

For any two features, a very fast and fairly tight lower bound on distance can be obtained from the minimum distance between their encapsulating LBCs. To find features within a given distance we can use the dilation process. Dilated LBs will overlap with

those near by, and this then becomes an AABB collision detection problem, which is well studied.

4.3.1. Case Study: Comparison between LBC and Octrees

As a case study we compare the header overhead of LBCs vs octrees for two data sets. Our first set consists of 90 sections of 250x230 resolution, a total of ~5Mbytes of raw grayscale data. This portion of the data includes several long thin neuron processes. Filtering and noise detection leaves us with 572,304 data points. LB storage stores some additional white space totaling 20,272 locations, into a total of 28,930 LBs. If stored as octree, this data requires 123,046 tree nodes plus overhead.

What is the overhead of the LB storage? Since the data is only 250x230x90, we can use 6 bytes per LB. We also store some whitespace, totaling 20,272 bytes which must be considered as part of the overhead. Finally, each LB must be pointed to from the LBC. This makes the total cost for using LBCs (excluding the cost of storing the grayscale information itself) 251,712 bytes.

For octrees, the header cost is one bit for marking leaf nodes, and a 2 byte pointer for each node. The total cost for of the octree for the same data is 261,473 bytes – more than for the LBCs.

Another data set shows different characteristics. This data set only contains cell bodies, which tend to be short and blobby, and consists of 100 sections of 500x500 resolution, making the total volumetric data storage 25 Mbytes. Filtering and noise detection leaves us with 156,831 bytes, stored in 10,684 LBs with additional 188,359 cells of whitespace included. Note that the whitespace included is a lot larger than in the previous case, because the cell bodies form chunks of data irregularly shaped. The header cost is 284,515 bytes. The octree version has 83,273 tree nodes, and the total cost is 176,996 bytes.

One should note that the second set of data is the worst case behavior for the LB data structure, and the best case for the octree. Even then, the additional cost is not that large. The first data set is typical of the volumetric data we will deal with, though on a smaller scale. Thus, using the LB data structure we get on the fly generation, noise detection, and simpler threading, with only minor or even less overhead than using octree storage.

5. Application

We describe here the results of the polymerization strategy applied to a sample database of scanned neuronal data – demonstrating the utility of our approach for the compression, storage, segmentation, and reconstruction of volume data.

5.1 Forming the EVDS

We use our L-block structure to process the volume data sets obtained from sets of serial scanned sections of stained mouse brain tissue. These sections were scanned using the Knife Edge Scanning Microscope (KESM) [McC02] and are of the size 4096 x 50,000. Note that due to limitations on the amount of scanned data available now and initial limitations on the microscope setup, we deal with much smaller data sets in the examples. However, this method is being developed to handle the larger data sets that will be generated in the near future.

To test our polymerization strategy, we have considered two specific sets of data, one consisting of mostly threads and the other mostly of cell bodies and considered only partial data. For the first case we have 90 sections, each of 250x230 resolution. And for the second we have 100 sections, each of 500x500 resolution. Each voxel of the data set represents a volume of 0.37 μ m by 0.37 μ m by 0.5 μ m. The Z direction is taken to be perpendicular to the image plane. The details of this data set have been described in [BMT*01]. A sample of the input sections is shown in **Figure 3**.

For the examples presented here, we use very simple functions to determine valid vertices and edge labels. We consider vertices significant if they pass a simple thresholding test (e.g. have grayscale values above a certain level and below another level). Edges are labeled active iff both of the adjacent vertices are significant. While future reconstruction efforts will likely involve more complex labeling functions, these suffice for making an EVDS for initial testing purposes.

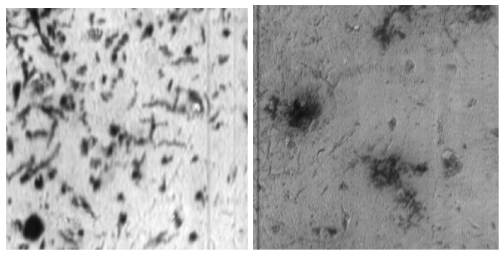

Figure 3. Two sections in our dataset.

5.2 Compression of Data

The memory needed to hold useful amounts of uncompressed neuronal data is exceedingly large. For example, the raw KESM data for an entire mouse brain requires approximately 30 terabytes and is a continuous data stream. For this reason, space saving features of the L-block structure and compression of the initial data are very important.

The majority of data compression takes place during the thresholding stage. Voxels that do not pass a thresholding stage are considered "white space". The EVDS is partitioned into 2x2x2 cells. If any of the voxels in a cell is valid (i.e. passes the threshold test), that cell is stored as a (2, 2, 2) LB. The compression achieved depends on the stain, the threshold used, and the density of the data. For the two data sets (250x230x90, 500x500x100) considered from the Golgi-stained data set, the uncompressed data requires approximately 5 MB and 25 MB of storage space respectively. With realistic threshold levels, we form 65,611 LBs and 50,218 LBs, requiring about 1.1 MB and 0.8 MB to store, yielding compression factors of approximately 4 (20%) and 31(3.2%) respectively.

We employ two types of noise reduction strategies. The first consists of removing large smear noise that exists on single layers. The second method of noise removal eliminates small disconnected LBs. **Table 1** gives the detailed statistics obtained during the entire process of reconstruction. Combining LBs minimizes the overhead of the header information. While

Data Set	2x2x2 LBs		Noise Removal I		Noise Removal II		Compression		Cleaning	
	LBs	LBCs	LBs	LBCs	LBs	LBCs	LBs	LBCs	LBs	LBCs
250x230x90	65611	2353	64737	2131	60784	1150	28938	1150	28930	1148
500x500x100	50218	4450	43506	3327	40887	2500	10684	2490	10648	2484

Table 1. L-block construction statistics

combining two (2, 2, 2)LBs is straightforward, combining larger LBs with smaller ones may be more problematic. Since LBs store all data in an iso-rectangular volume, expanding an LB might require storing "white" space along with relevant data. To determine whether or not it is appropriate to create such LBs, we use a cost function based on the relative storage requirements for the merged and unmerged LBs. We consider merging LBs in each of the positive X, Y, and Z axis directions. The L-blocks are extended if the space that would be saved by eliminating L-block overhead is greater than the space lost by storing empty data. For the entire data set, our merging strategy reduces the total number of LBs twofold (28,938) and fourfold (10,684), requiring less than 0.4 MB and 0.18 MB of storage respectively.

Note that these strategies are well suited for processing 3D microscopic data where data arrives one "section" at a time and each section must be processed in real time. Merging LBs requires only storage of the (possibly already combined) LBs that cover portions of the immediately preceding section – typically there will be relatively few such LBs, and in any case, the number is bounded by the size of the section. Finally, noise reduction can be applied to only those LBs currently in memory.

In the first data set, since we concentrate on the threads, the occupancy of the threads is greater and the amount of white space is less. For the second data set (where we classify primarily cell bodies), since there are few cell bodies in the region we get very good compression rates. From **Table 1**, we can see that the acquired compression rate due to noise removal techniques is not much. The reason for this is that, before starting with the first stage of reconstruction, some image-based filtering is done to get rid of some scanning artifacts. Some of the filtering techniques include contrast enhancement, applying a mean filter and a median filter, and multiplying in the X, Y and Z directions.

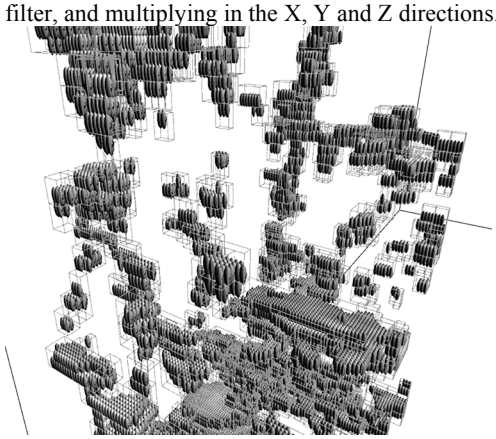

Figure 4. Filtered data stored as LBCs

5.3 Data segmentation

Taking advantage of the fact that neuronal data is both sparse and clustered, our data is combined into clusters, defined as groups of interconnecting LBs. If two LBs border on each other and at least one of the voxels on the border has an active link to a voxel in the other LB, both LBs are considered to be in the same cluster, and these are joined in an LBC. Since the voxels themselves are used to determine cluster boundaries, this scheme effectively segments the data, i.e. it does not group two pieces of data that should have been separate. If LBs are clustered before merging (see section 5.2), the space of LBs to be examined for potential merging is reduced, thus speeding up the algorithm. Once the LBCs are formed they can be processed in parallel.

Figure 6c shows an example of the connectivity between the LBs from a close-up region from **Figure 4**. The lines in the figure indicate that the LBs centered at each endpoint are joined by an active edge, and thus are grouped together in a cluster.

5.4 Neuron Reconstruction

We have implemented a method for extracting neuron models based on the segmented LBC. This is presented primarily to show the utility of the LB/LBC data structure, and not as an ideal neuron reconstruction algorithm. Note that our goal is actual neuron modeling, not simply visualization [ASK94]. We begin by dilating the LBs within each cluster, and LBs overlap (in extent) "nearby" LBs. We join these overlapping LBCs, effectively filling in gaps that could be missing due to noise. We form an expanded connectivity graph, linking the overlapping dilated LBs (**Figure 6c**). Of course, this also joins some LBCs that should be kept separate, therefore the complexity of the connectivity graph is reduced in order to identify the major threads along which the neuron lies. This complexity reduction involves identifying short cycles that do not change the global topology. The LB structure allows us to work with packages, rather than the data contained within, decreasing the memory requirements of the algorithms.

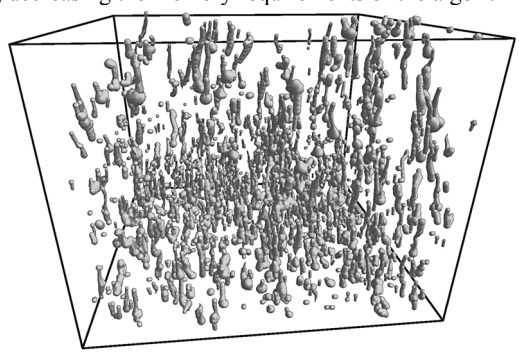

Figure 5. A view of the reconstructed neurons

The expanded connectivity graph is then simplified into a tree format (i.e. a hierarchical LBC) that captures the major dendritic threads passing through the sample section set (this tree is used for storage and does not necessarily describe the neuron structure itself). "Fine-scale" detail is removed temporarily, which can be identified based in part on the LB sizes and their connectivity with the rest of the graph. This simplifies the connectivity graph considerably. Graph algorithms are applied to simplify the graph further. We note that these graph simplifications are rather complex; their justification is beyond the scope of this paper. A "thread axis", possibly including branching, is constructed around which the hierarchical LBC can be created. Given the hierarchical LBC, a medial axis approximation can be obtained. Using radius estimation, the medial axis representation can be iteratively refined to match the LB representation (**Figure 6e**). A 3D reconstructed view can be seen in **Figure 5**. Note that because the reconstruction region is so small, we have only portions of each neuron, and thus we do not capture much of the branching structure typical of neurons.

6. Conclusion

6.1 Summary

We introduce a new data structure optimized for the representation of filamentary volumetric solids. There are two key

components of our data structure. First, we describe the *enhanced volume data set (EVDS)*, where the data set is enhanced by explicitly introducing Boolean labeling of edges between adjacent voxels of the volume data. Next, we introduce a new container type, the *L-block*. L-blocks are designed to efficiently package the connected components of the EVDS, with a minimum of adjacent unconnected voxels. We have implemented the L-block structure described, and present a comparison with other data structures and the results of its application to some sample neuron data. The structure is well suited for sparse, clustered data, for which it provides very good compression. The data structure is very general, and flexible, and is capable of mimicking the operation of other data structures.

6.2 Further Work

Currently, we are working on expanding on the LBs/LBCs in the following ways:

- The polymerization strategy on volume data is highly dependent on the edge labeling strategy. Developing an effective edge-labeling strategy is thus important for the polymerization strategy to be effective.
- The threading approach used here is still somewhat ad hoc. We are exploring alternative and more formal ways of generating threads from an LBC's graph.
- We are seeking ways of getting improved visualizations of LBCs and their contained data, particularly over large regions.

Acknowledgement

This work was supported by Texas Higher Education Coordinating Board ATP grant 000512-0146-2001 and NSF grant CCR-0220047.

References

[ASK94] Avila R. S., Sobierajski L. M., Kaufmann A. E., Visualizing Nerve Cells. *IEEE Computer Graphics and Applications*, (Sept. 1994), pp. 11-13.

[BMT*01] Burton B.P., McCormick B.H., Torp R., Fallon J.H., Three-Dimensional Reconstruction of Neuronal Forests. *Neurocomputing*, (2001), 38-40:1643-1650.

[CHJ03] Co C.S., Hamann B., and Joy K.I, Iso-splatting: A Point Based Alternative to Isosurface Visualization, Pacific Graphics 2003 Proceedings, 2003, pp 325-334.

[EHV*03] Edelsbrunner, H., Harer, J., Natarajan, V., Pascucci, V., Morse Complexes for Piecewise Linear 3-Manifolds. Proceedings of ACM Symposium on Compuatational Geometry, (2003), pp. 361-370.

[FKN80] Fuchs, H., Kedem, Z., Naylor, B., On Visible Surface Generation by A Priori Tree Structures, *Proceedings of SIGGRAPH*, pp. 124-133, 1980.

[JT80] Jackins C., Tanimoto S. L., Oct-trees and Their Use in Representing 3-D Objects. *Computer Graphics and Image Processing 14*, (1980), pp. 249-270.

[KR89] Kong T.Y., Rosenfeld A., Digital Topology: Introduction and Survey. *Computer Vision, Graphics, and Image Processing 48*, (1989), pp. 357-393.

[McC02] McCormick B.H., *Development of the Brain Tissue Scanner*. Technical Report, Department of Computer Science, Texas A&M University, College Station, TX, 18 (Mar. 2002). Available from http://research.cs.tamu.edu/bnl

[MBM*02] McCormick B.H., Busse B., Melek Z., Keyser J., *Polymerization Strategy for the Compression, Segmentation, and Modeling of Volumetric Data*. Technical Report 2002-12-1, Department of Computer Science, Texas A&M University, (Dec. 2002). Availabe at: http://research.cs.tamu.edu/bnl

[OV82] Overmars M. H., Van Leeuwen J., Dynamic Multi-Dimensional Data structures Based on Quad- and k-D Trees. *Acta Inform. 17*, (1982), pp. 267-235.

[Sam84] Samet H., The Quadtree and Related Hierarchical Data Structures. *ACM Computing Survey 16*, (1984).

[SW88] Samet H., Webber R. E., Hierarchical Data Structures and Algorithms for Computer Graphics, Part 1: Fundamentals. *IEEE Computer Graphics and Applications 5*, (1988), pp. 48-68.

[SW288] Samet H., Webber R. E., Hierarchical Data Structures and Algorithms for Computer Graphics, Part 2: Applications. *IEEE Computer Graphics and Applications 7*, (1988), pp. 59-75.

[van97] van den Bergen G., Efficient Collision Detection of Complex Deformable Models Using AABB Trees. *Journal of Graphics Tools,* 2:4, (1997), pp. 1-13.

[Win02] Winter A.S., *Volume Graphics, Field Based Modelling and Rendering*. Ph.D. Thesis, Department of Computer Science, University of Wales, Swansea, (Dec. 2002).

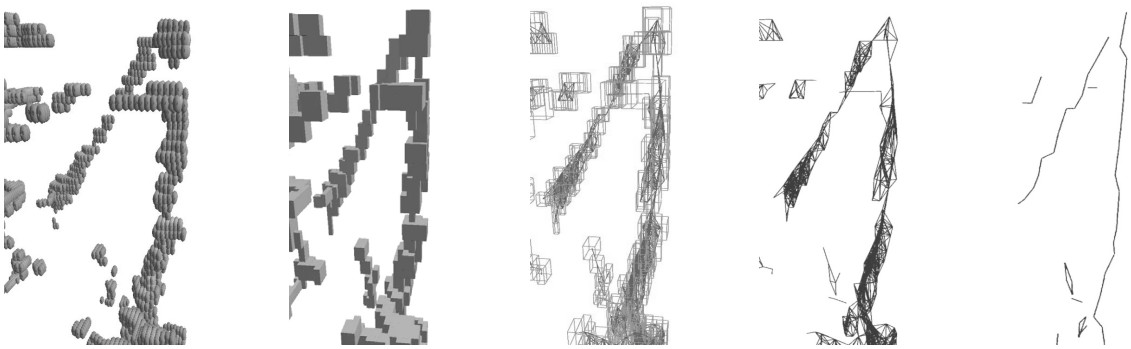

Figure 6. a) Filtered data b) L-blocks displayed as boxes c,d) connectivity graph e) estimated threads