Polymerization Strategy for the Compression, Segmentation, and Modeling of Volumetric Data

Bruce H. McCormick Brad Busse Purna Doddapaneni
Zeki Melek John Keyser
Department of Computer Science, Texas A&M University

Abstract

We present a data structure for the representation of volumetric data. The data structure is designed to allow for easy compression, storage, segmentation, and reconstruction of volumetric data. We call our data structure the L-block, abstracting many of the properties of Lego® blocks, and refer to the process of creating and manipulating L-blocks as the polymerization strategy.

The concept of an enhanced volume data set (EVDS) is introduced, where the data set is enhanced by explicitly introducing Boolean labeling of edges between adjacent voxels of the volume data. This enhancement, by "polymerizing" adjacent connected voxels into connected components, facilitates real-time data compression and segmentation of embedded objects within the volume data set. These connected components are packaged in the new container type, the L-block, with the intention of efficiently packaging the connected components with a minimum of adjacent unconnected voxels.

We present the L-block data structure in detail. We describe methods for compressing volume data using the L-block structure, intersecting and merging L-blocks, and segmenting data. While the L-block data structure is general, it was developed to represent scanned brain microstructure at a neuronal level of detail. We highlight the performance of our implementation of the polymerization strategy on a set of sampled neuronal data.

1. Introduction

1.1. Motivation

Volumetric representations are needed to model the objects found in volumetric data sets. Sources of such data sets include medical imaging procedures (e.g. MRI) and, more generally, various three-dimensional scanning processes on real-world data.

The work presented in this paper has been particularly motivated by our attempts to scan and reconstruct brain tissue at a neuronal level of detail. The data sets acquired in this work tend to have several distinguishing features. Among them are:

- The full volume data set can be extremely large. Raw data set sizes can reach into the 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.
- The neurons to be modeled have very long, thin branching structures.
- Data will be acquired at a high rate, and one would like to have a quick way of compressing and storing it in a geometrically meaningful way that facilitates future reconstruction.

We have found current volumetric representation techniques to be deficient in addressing at least one of these features. Due to the potential data size, methods that keep the entire volume in memory at once are unrealistic. Several methods (such as the well-known 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 can work well for compression, but fail to give any meaningful insight into the geometric structure of the objects to be modeled.

1.2. Main Results

We introduce a new data structure designed specifically to address the data set features listed above. There are two key components of our data structure. First, we introduce the concept of an *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 (and coverings with L-blocks) are designed to efficiently package the connected components of the EVDS, with a minimum of

adjacent unconnected voxels. We refer to the process of constructing L-blocks from volumetric data as the *polymerization algorithm*. We have implemented the L-block structure described, and present the results of its application to some sample neuron data.

2. Representations

In this section we describe the representation of the L-block data structure. We begin by describing the concept of an enhanced volume data set. Next we discuss the L-block data structure itself. Finally, we discuss the L-block structure in relation to other volumetric models.

2.1. Domain

We assume that we are given a uniform n-dimensional grid. Every vertex of this grid is assigned a value. The nature of this value may vary. Three possible examples include:

- An integer gray-scale value. This might occur when the data has been obtained from some scanning process, such as from MRI.
- A binary value. This could indicate whether the vertex is or is not in some object. Thresholding or similar techniques might have been used to convert grayscale values to binary values, for example.
- A vector of values. This might arise from multi-spectral scanned data such as a color camera with three channels (RGB).

For simplicity, we will usually refer to a 3-dimensional data set, commonly called a *volume data set*. For 3-D data sets, the voxels form the vertices of the grid. Although we often confine our description to 3D volumes, the concepts are equally applicable to other dimensional data sets.

2.2. Enhanced volume data sets

We create an *enhanced volume data set* (EVDS) as described below. The goals of the enhancement are to:

- Allow data compression in real time, in such a manner as to facilitate subsequent segmentation of the volume data set;
- 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;
- Separate segmentation clearly from both geometric modeling and visualization of the identified objects in the volume data set.
- Exhibit the statistical basis for the enhancement of the volume data set.

Given a volume data set, we define an 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 [5].

Edge labeling is used to provide independent information about whether two vertices sharing a common active edge belong to the same underlying object. Boolean labeling {0, 1}, as derived typically from a decision function, is of course a crude estimate of this co-habitation in the same object.

It is important to note that the decision function used to assign the Boolean values is in effect the segmentation process, and is of primary importance in determining how faithful a particular segmentation or reconstruction is. This function can be arbitrarily complex. Choosing such a function is outside the scope of this paper; we assume such a function is available, and seek to provide the data structure support needed to work with the result.

In three dimensions, the data volume is typically created by serially scanning successive *sections* perpendicular to (say) the vertical Z-axis. Here voxels at the same (X, Y) position in two successive registered images can be conveniently labeled as likely drawn from the same underlying physical object by marking their common vertical edge as active. However, as the specimen could have been sectioned perpendicular to the X- or Y- axes, we extend the same edge labeling scheme to all three directions.

Any enhanced volume data set (in three dimensions) can have many representations. Most useful for our purposes is an assignment at each vertex of an association {<*voxel value*> <*edge labels*>}, where the Boolean vector <*edge labels*> indicates 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.

Note that for a regular grid, there may be a choice in the number of edges emanating from any one vertex. We only assume that each vertex has a fixed set of emanating edges. The number of edges emanating from any one vertex is referred to as the connectivity level. For example, consider a regular 3D grid of vertices. If no edges are stored at any vertex (i.e. the data set is not enhanced), we have 0-connectivity. 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 give 13-connectivity. Note that for dimension *k*, the connectivity

level will often be *k*, as well, and the connectivity level may be at most $(3^k-1)/2$.

A vertex in an EVDS with 3-connectivity can be thought of as having "links" that extend to the neighboring vertices along the three coordinate axes (see Figure 1). Thus it behaves somewhat like a Lego® block, with connections possible along 3 axes.

Figure 1. Vertices in a 3-connected and 9-connected EVDS. The bars show potential links to neighboring vertices.

2.3. L-blocks, Coverings, and Partitions

Given an EVDS, "whitespace" is defined as vertices that do not satisfy the threshold test. We use L-blocks, L-block coverings, and L-block partitions to represent the data that is not whitespace.

2.3.1. L-blocks

An *L-block* is defined as a *k*-dimensional iso-rectangular block of enhanced vertex information. The block must be entirely contained within the uniform *k*-dimensional grid of the EVDS. An $(l_1, l_2, \ldots, l_k)$ L-block refers to a block of l_1 vertices in the first dimension, l_2 in the second, etc. Each L-block is defined by its <*header*> information followed by its <*vertex array*>. The <*header*>={<*position*>< *template*>}, is given by: (1) the *position*, e.g. *(x,y,z)*, of its least vertex, as indexed within the parent *k*-dimensional uniform grid, and (2) its *template* $(l_1\ l_2\ \ldots\ l_k)$. Its <*vertex array*> contains the enhanced vertex information (voxel value(s) and edge labels). In summary the {<*position*> <*template*> <*vertex array*>} characterizes a L-block.

Given a *k*-dimensional $(l_1, l_2, \ldots, l_k)$ L-block, the number of bits required to store the header is 2*D*, where $D = \sum_{i=1}^{k} D_i$, and D_i is the number of bits needed to store the size of the EVDS (not just the L-block) in each of the *k* directions. For example, a 1024^3 data set would give L-blocks with $D_i = 10$, $D = 30$, and each header requiring at least 60 bits. The vertex array of the L-block requires $(b_r + j)\prod_{i=1}^{k} l_i$ bits, where b_r is the number of bits required to store an individual sample and *j* is the connectivity level. For binary data, b_r would be 1, for grayscale data, b_r is often 8, and for color images, b_r is often 24.

The L-block as a whole can be visualized as a block of vertices, with extensions that demonstrate connectivity. An example is seen in Figure 2.

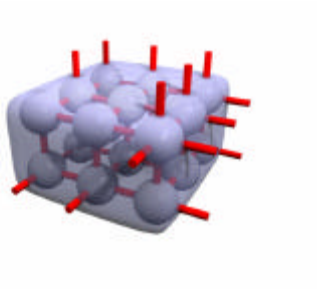

Figure 2. A (3,3,2) L-block. Cylinders represent active edges emanating from the L-block.

2.3.2. Coverings and Partitions

A *covering* of a volume, *V*, by a set, *A*, of L-blocks is defined as $C(A,V) = \bigcup_{a \in A} L_a$, such that any vertex in *V* is in the vertex array of some L_a. The L-blocks, L_a, may overlap and need not be adjacent. Note that the volume *V* may be of arbitrary shape and size (it need not be rectangular). A covering $C(A,V)$ can be given a hierarchical decomposition in terms of other coverings,

$C(A,V) = \bigcup_{i=1}^{m} C(A_i, V_i)$, where

$\{A_i \subseteq A \mid i = 1,\ldots,m\}$ and $\bigcup_{i=1}^{m} V_i \subseteq V$.

A *partition P(A,V)* is a restricted form of covering such that the L-blocks themselves do not overlap. Similar to coverings, an L-block partition can be given a hierarchical decomposition in terms of other L-block partitions.

A hierarchical L-block covering (and thus an L-block partition) can be defined by a <*header*> followed by a <*sub-block list*>. The header is identical to that for an L-block, and thus also requires *2D* bits to store. The <*sub-block list*> is composed of a list of pointers to other L-block coverings. The number of bits needed for the sub-block list, then, is b_v+Nb_m, where b_v is the number of bits needed to store the maximum number of pointers (note that b_v is never more than *D*), *N* is the number of pointers used, and b_m is the number of bits used for a pointer (usually operating system or compiler dependent). Of course, each covering or L-block that is referenced has its own storage cost. The elements of a covering can be stored in any order, and can be transmitted without concern for their order of arrival at the receiving site.

An example of a hierarchical L-block covering can be seen in Figure 3.

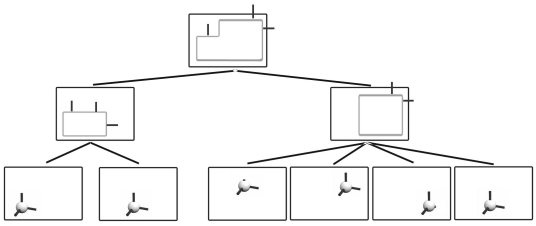

Figure 3. An L-block covering (top of the tree) is formed from the union of two other L-block coverings (middle row). Those L-block coverings are formed from unions of (1,1,1) L-blocks (on the bottom row).

2.3.3. Notation

Hereafter, we will use the following abbreviations. An L-block, consisting of a header and enhanced vertex array will be referred to as an LB, with the template optionally given immediately beforehand. For example, a single 3-D voxel could be described by a (1,1,1)LB. A covering of L-blocks will be referred to as an LBC, and a partitioning of L-blocks as an LBP. It will be assumed that all LBCs and LBPs are expressed hierarchically.

2.4. Applying LBCs to volume data

Our intention is to identify objects of interest from within a volumetric data set. To this end the *polymerization strategy*, described in section 3.2 below, views active edges within the EVDS as hardening (polymerizing) into a structure not unlike a jungle gym and the objects of interest lifted (segmented) out of the 3D block-structured grid.

Several definitions and lemmas are introduced here to help formalize the polymerization process.

2.4.1. Volume and regions of interest

A *volume of interest,* VOI_A, is a covering $C(\mathrm{A},V)$ of a connected component G, where $G \subseteq C(\mathrm{A},V)$. A *volume segmentation strategy* then consists of identifying volumes of interest, $\{VOI_1, VOI_2, ..., VOI_n\}$, that collectively provide a covering of all connected components within a given volume, **V**.

For the special case where **V** is a planar slice of the EVDS, a *region of interest,* ROI_A, is a covering $C(\mathrm{A},V)$ of a connected component G, where $G \subseteq C(\mathrm{A},V)$. An *image segmentation strategy* then consists of identifying regions of interest, $\{ROI_1, ROI_2, ..., ROI_n\}$, that collectively provide a covering of all connected components within the volume (image) **V**. ROIs from successive images can then be threaded together by their active edges to form VOIs.

2.4.2. Graph of a covering

Two distinct L-blocks, L_a and L_b, are *joined* if their vertices share at least one active edge. Adjacent or overlapping LBs may or may not be joined. We define the graph $\mathbf{G}$ of a covering $C(A,V)$, as follows:

1) *Vertices of the graph* $\mathbf{G}$: Each L-block, $L_a, a \in A$, is assigned a vertex in $\mathbf{G}$.
2) *Edges of the graph* $\mathbf{G}$: An edge $e_{ab} \in \mathbf{G}$ links two L-blocks iff the L-blocks are joined. In general, the edge is undirected.

Lemma. Every connected component within a covering $C(A,V)$ resides in a connected component of the graph of the covering. The converse is not in general true.

2.4.3. Cost of a covering or partition

Useful segmentations, by coverings or partitions, of the active vertices in a volume **V** must be efficient: avoiding overly large blocks (covering excessive white space) or requiring large numbers of small blocks. The quality of the covering (partition) can be controlled in part by assigning a cost to a covering (partition). For a covering $C(A,V)$, we assign a cost $\$(C(A,V)) = k + \sum_{a \in \mathrm{A}} \$(L_a)$, and $\$(L_a) = l + N_a m$, where k is a cost associated with the covering (irrespective of the L-blocks), l is a cost associated with a single L-block, m is a cost associated with a single vertex, and N_a is the number of vertices in L_a. An identical cost formula will be used for partitions.

Although many such cost functions are possible, one natural one is based on the memory requirements of the structure, as outlined in the bit costs described in section 2.3. In this case, the parameters k and l represent the space needed to store the header information, while m represents the space needed to store the voxel value and edge information at each vertex.

The objective, then, is to find a minimal cost covering for the given volume. Notice that for $l = 0$, any covering of all active vertices by L-blocks with (1 1 1) templates is of minimal cost, provided the (1 1 1) template is permitted in the covering. For $l > m$,

minimal coverings will trade off using fewer blocks with covering more white space. As l increases, a minimal cost covering will, in general, use fewer blocks of larger size. Counter to intuition, for a given covering $C(A,V)$ and cost parameter $0 < l$, a partition $P(A',V)$ of lower cost might not exist.

3. Operations

3.1. Binary L-block operations

There are many possible operations that can be defined on L-blocks. We present two binary L-block operations that are quite useful for reconstructing solids from scanned data. Merging is commonly used when processing data. Intersection is important in that it can be used to examine only the portion of a data structure within a restricted region of space.

3.1.1. Merge/Union

The merging of two LBs will form another LB. For given input LBs L_a and L_b, assume the header information in dimension *i* is given by position p_i and template value t_i. Then, the new header will have position $p_{i,ab} = \min(p_{i,a}, p_{i,b})$, and template index $t_{i,ab} = \max(p_{i,a} + t_{i,a} - p_{i,ab}, p_{i,b} + t_{i,b} - p_{i,ab})$. It is necessary to assign values to all the vertices within the merged L-block. For those corresponding to vertices from the input LBs, this is straightforward. For the other vertices, no information is known, and we assign "empty" values with no active edges to each of these vertices.

Considering LBCs, the merging of two LBCs is a straightforward process. Either a new LBC is created, with pointers to the input LBCs as sub-blocks, or the pointers of the two input LBCs are merged into a single list. In either case, the header information is adjusted as for merging two LBs. LBCs are closed under union.

3.1.2. Intersection

The intersection of two LBs, L_a and L_b, $L_{ab} = L_a \bigcap L_b$, is either an LB or the empty set $\varnothing$. Forming the header and vertex array is straightforward. For LBPs this representation is unique: Given partitions *P*(A,*V*) and *P*(B,*V*), with $(a, m \in \mathrm{A} \mid b, n \in \mathrm{B})$, then $L_{ab} = L_{mn} \neq \varnothing$ implies $a = m, b = n$.

Lemma: The class of coverings with respect to a volume V is closed under intersection.

The intersection of two LBCs, $C(\Delta,V) = C(A,V) \bigcap C(B,V)$, is again an LBC. The resulting LBC is formed from a collection of LBs: $\{L_{ab} = L_a \bigcap L_b \mid a \in A, b \in B, ab \in \Delta\}$. In general the covering set Δ will have members in neither A nor B.

Lemma. The class of partitions with respect to a volume V is closed under intersection.

The intersection of two LBPs, $P(\Delta,V) = P(A,V) \bigcap P(B,V)$, is again formed as a collection of LBs in the same way as for coverings. The result is a partition since the L_{ab} are disjoint:

$$L_{ab} \bigcap L_{mn} = (L_a \bigcap L_b) \bigcap (L_m \bigcap L_n)$$
$$= (L_a \bigcap L_m) \bigcap (L_b \bigcap L_n)$$
$$= d_{am} L_a \bigcap d_{bn} L_b = d_{am} d_{bn} L_{ab}$$

That is, the intersection of the two partitions can be expressed as a collection of disjoint L-blocks.

3.2. The polymerization strategy

The *polymerization strategy* refers to the use of an enhanced data set stored in 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.

Connected components in the extended volume data set are of particular interest, as these are the substrata upon which objects in the volume data set are modeled. Focusing on its connected components, and efficiently packaging these within LBCs, can significantly compress an EVDS. Given an EVDS, polymerization therefore lets us compress the data, retaining only what is needed.

Isolated vertices, those having no active edges, occur regularly in scanned volume data, often due to "noise." Such vertices can be ignored, or at worst, packaged in small LBs for separate consideration, should their voxel value exceed some threshold. In this case the remaining vertices outside the coverings can be treated as "white space", and ignored in subsequent image processing. The content of the EVDS, exclusive of its "white space", is then captured in the L-block covering or partition.

At other times only the boundaries of objects warrant consideration. Here "black space" LBs, whose every

associated edge is active, can be separately noted, and suppressed.

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 need 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 (ROIs)* in the Z-plane image are then packaged in (m n 1) L-blocks and added to the evolving compressed representation of the EVDS. Black space blocks can be deleted at this time or their processing deferred. 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. Comparison with other data structures

4.1. LBCs as a geometric superstructure

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, e.g. [3] for discussions of these). 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 [7].

4.2. Comparison with other methods

There are a large number of approaches to the storage of volumetric data. Here, we will briefly summarize several of these other volumetric data structures, and highlight the relative advantages/disadvantages of our data structure. The results are summarized in Table 1.

4.2.1. Alternative Data Structures

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[14]. We briefly summarize the key alternative approaches:

- *Grid-sampled data*. This is the standard input format for sampled data – values are kept at every point in the entire data set. Since this clearly provides no compression, it will not be considered for comparison.
- *Spatial-occupancy enumeration*. [3] In this very simple structure, the individual voxels of interest are listed and stored individually.
- *Octree/quadtree*. [4][9][10][11] The well-known octree uses a hierarchical spatial-occupancy approach. Blocks of data are recursively broken into 8 suboctants, each of which are either completely filled, completely empty, or partially filled, in which case they are further subdivided.
- *BSP-tree*. [3] This approach recursively divides space by an arbitrary plane at each level, usually dividing the remaining points equally. The leaves of this tree are convex regions bounded by the planes in ancestor nodes, and are classifed as either inside or outside. Though created for a continuous domain, binary space partition trees can be easily adapted to a grid domain by limiting the binary planes at each step to an axis-aligned approach. BSP trees can also describe infinite volumes, which is not needed in our domain.
- *kD-trees*. [8] A kD-tree is similar to a cross between an octree and a BSP-tree. Space is subdivided recursively, but at each level, the (axis aligned) direction and exact position of the plane can be chosen. Because of their similarity to BSP-trees, we will treat them together.
- *AABB-trees*. [13] Axis-aligned bounding box trees, more commonly used in collision detection, are perhaps the most similar to LBCs. These trees consist of a hierarchical collection of iso-rectangular boxes, each bounding the boxes of the child nodes. The key difference in our approach is that the enhanced data allows us to easily build LBCs incrementally and to maintain connectivity between nodes without having to go through a parent node.

4.2.2. Comparison Criteria

While our data structure is general, and could be used in a number of volumetric applications, we are particularly interested in it as a data type for initial processing of scanned neuron data. Several distinguishing characteristics of this data were listed in Section 1.1, and the criteria we describe below are derived from those considerations.

Obviously, a key concern is the amount of storage required by the structure. Note that efficiency will vary depending on whether the voxel values are binary (“in” or “out”) or more complex (e.g. grayscale values).

A second concern is that the method be able to handle neuron shapes well. Neurons tend to be very long and very thin, and have a complex branching structure; methods that work well for big, blocky shapes (e.g. octrees) typically represent long thin structures poorly.

In addition, we would like the method to let us easily determine the shape of the modeled object from the data structure (e.g. by adapting size along axes). In addition, we would like the method to allow us to easily capture the complex branching structure of the neurons.

Data Structure	Storage (binary)	Storage (values)	Suited for neuron shape	Determine shape from structure	Incremental Construction
Spatial enumeration	Poor	Good	Fair	Poor	Excellent
Quadtree/octree	Excellent	Excellent	Poor	Fair	Poor
BSP/kD tree	Excellent	Excellent	Fair	Good	Poor
AABB tree	Good	Very Good	Good	Very Good	Fair
EVDS/LBCs	Good	Very Good	Good	Excellent	Very Good

Table 1. A comparison of several volumetric data structures on important criteria.

Finally, we want the method to be well suited to incremental construction, where we cannot keep the entire input dataset in memory at once. This makes spatial subdivision methods poorly suited to the problem.

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 a volume data set obtained from a set of 14 serial scanned images from Golgi-stained mouse brain tissue imaged through a light microscope with incoherent illumination. Registration of the images along the vertical axis was done by hand. Regions near the borders of the images are unusable (i.e. one image might contain data at positions unavailable in the images above and below). 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. We often refer to individual images as "sections." The details of this data set have been described in [2]. A sample is shown in Figure 4.

We plan to use our L-block structure to process data obtained from the Brain Tissue Scanner (BTS) [6]. The BTS is a unique instrument developed at Texas A&M that uses a diamond knife to concurrently cut tissue and scan the tissue at the knife edge. We anticipate the BTS data for Golgi stain will be similar to the sample data. Other stains may have different properties.

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.

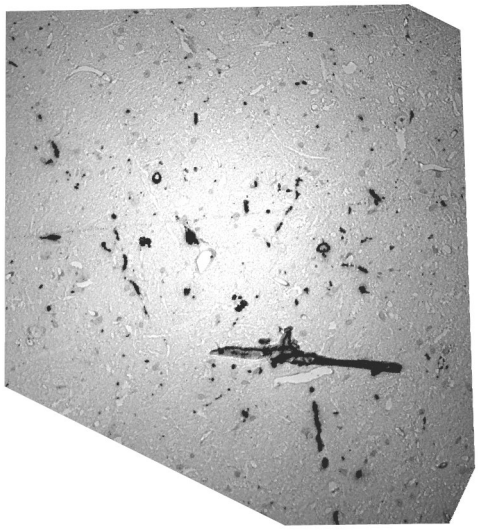

Figure 4. A section in our dataset. The large black region is a stain smear.

5.2. Compression of Data

The memory needed to hold useful amounts of uncompressed neural data is exceedingly large. For example, the raw BTS data for an entire mouse brain requires approximately 29 terabytes. 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 the thresholding stage are considered "white space," and it is assumed that they can be ignored thereafter. The EVDS is partitioned into 2 by 2 by 2 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 will depend on the stain, the threshold used, and the density of the data. For our sample Golgi-stained data set the initial data requires approximately 112 MB of storage space. With realistic threshold levels, we form 47,258 LBs requiring about 4.4 MB to store, yielding a compression factor of approximately 25.

We then provide additional compression by combining LBs where appropriate. Merging L-blocks has the advantage of eliminating the overhead of the header information. While combining two (2,2,2) LBs into, say a (4,2,2) LB is straightforward, combining larger LBs with smaller ones may be more problematic. Because 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 strictly 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. Figure 5 shows a close-up image of merged LBs (drawn as wireframe boxes) from the sample neuron data set. Figure 6 shows the merged LBs for the entire sample data set, and Figure 7 shows the merged LBs for a portion of the data set, overlaid with the valid data. For the entire data set, our merging strategy reduces the total number of LBs fourfold to 12,841, requiring less than 3.7 MB of storage.

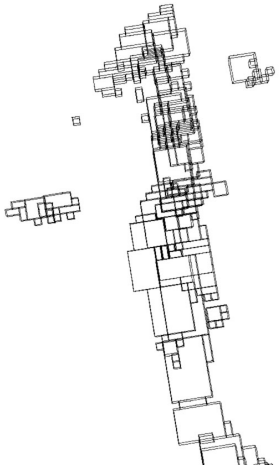

Figure 5. Merged L-blocks.

Noise reduction can also be used to reduce data storage. LBs that have no active edges emanating can be eliminated. Such LBs are unlikely to be a part of a neuronal structure, and are most likely due to noise in the input data. For our sample data set, noise reduction reduces the total number of LBs to 7525, requiring only 2.3 MB to store. Together, noise reduction and LB merging provide a factor of 2x compression in our sample data, giving an overall compression of approximately 50x.

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. Due to the amount of data, it is not practical to store many sectional images in memory at once. Merging LBs requires only storage of the (possibly already combined) LBs that cover portions of the immediately preceding section – typically there will be only a few such LBs, and in any case, the number is bounded by the size of the section. Although this process biases LB merging in the Z direction, it is necessary due to time and memory constraints. Finally, noise reduction can be applied to only those LBs currently in memory.

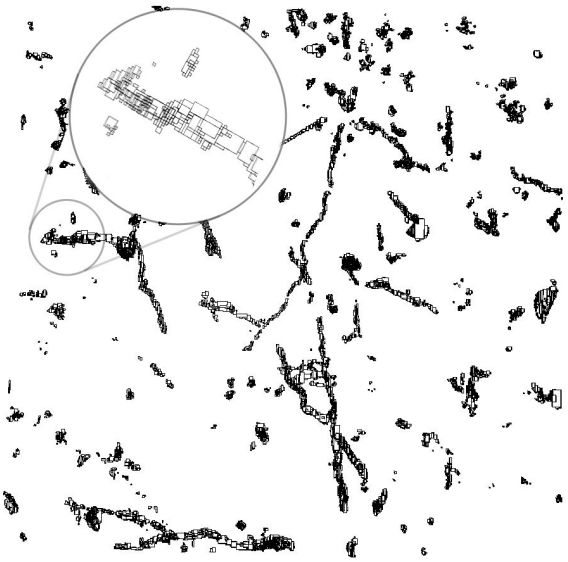

Figure 6. The merged L-blocks formed for the entire data set. The portion used to form Figure 5 is highlighted.

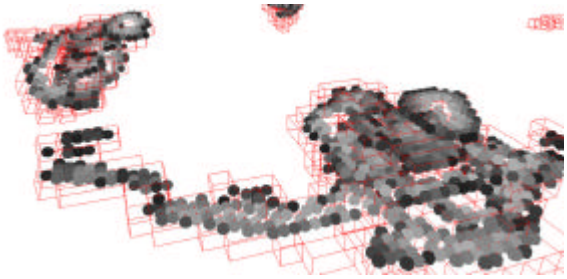

Figure 7. A portion of the reconstructed data showing the valid data within the L-blocks.

5.3. Data segmentation

Taking advantage of the fact that neural data (Golgi stained) is both sparse and clustered, our data is further combined into clusters, each expressed as an LBC. Clusters are defined as groups of interconnecting L-blocks. If two L-blocks border on each other and at least one of the voxels composing that border has an active link to a voxel in the other L-block, both L-blocks are

considered to be in the same cluster. 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. Notice that it is possible for two different LBCs to have L-blocks that overlap in space, but relevant data in one L-block will be empty space in the other, so no harm is done. Also, if LBs are clustered before merging, the space of LBs to be examined for potential merging is reduced, thus speeding up the algorithm.

Figure 8a shows an example of the connectivity between the LBs of Figure 5. 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.

Figure 8. a) The connectivity graph for Figure 5. LBs in the same connected component are grouped in an LBC. b) The major threads from a.

Noise reduction features are also implemented at the cluster level to conserve space. Two types of noise are targeted in our implementation. The first type is clusters that are too small to be a valid neural structure by themselves and too isolated to be a fragment of a larger structure. The second type is stain smears – medium to large swaths of data that exist entirely on the XY plane (arising from the staining agent smearing as the tissue was being cut). Figure 4 includes a stain smear. The rules used to identify these noise clusters are ad hoc, and though very effective on the sample data sets, would need to be altered for other data sets.

Note that this segmentation can be performed locally. That is, LBCs can be formed based on just a limited amount of data in memory at one time. Again, this is necessary due to the potentially large amount of data.

For the sample data set, the number of clusters formed from the data after initial thresholding is 5610. Section 5.2 describes the exact number of LBs and the total memory requirements. After noise reduction is used to eliminate some clusters entirely, the total number of clusters is reduced to only 1654.

5.4. Neuron Reconstruction

We have implemented a method for extracting a neuron model 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 [1].

We begin by dilating the L-blocks within each cluster (i.e. expanding the apparent size of the L-blocks by adjusting the header information). After doing this, LBs begin to 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. This can be visualized as in Figure 8a. Of course, this also joins some LBCs that should be kept separate. We must therefore reduce the complexity of the connectivity graph in order to identify the major threads along which the neuron lies.

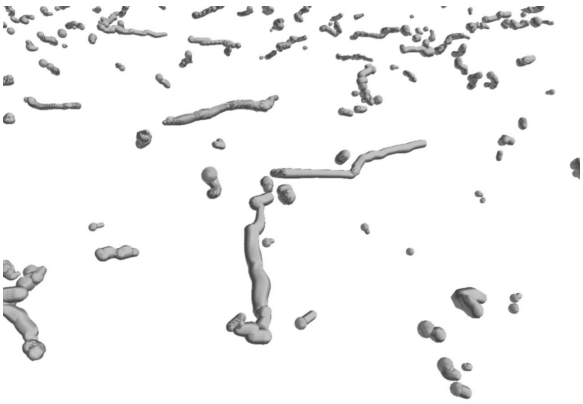

Figure 9. A view of the reconstructed neurons.

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 is done by first temporarily removing "fine-scale" detail, which can be identified based in part on the LB sizes. Removing this detail simplifies the connectivity graph considerably, after which we apply graph algorithms to simplify the graph further. In the end, 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 L-block representation. A picture of the thread axis so obtained is shown in Figure 8b. A 3D reconstructed view can be seen in Figure 9. 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 have presented a data structure for the representation of volumetric solids. We have described the basic operations on this structure, including the polymerization strategy for segmenting data from a volumetric data set. We have outlined how this strategy can be used for the compression, storage, segmentation, and reconstruction of volume data. The L-block data structure and polymerization strategy have been implemented, and we have demonstrated that it can be used effectively in the reconstruction of neuron data.

6.2. Features of Our Data Structure

To conclude, we point out a number of features of our data structure that give it an advantage over other structures, in certain circumstances.

- The structure can be used for data compression and segmentation in an incremental manner. That is, only a limited amount of the data set needs to be in memory at any one time, making it useful for application in a real-time scanning environment.
- The compressed data has clear geometric meaning, as opposed to compression methods such as standard image compression (e.g. JPEG/MPEG), where most geometric interpretation is lost.
- The data structure is well suited for describing long, thin objects, as well as branching structures. Many other methods (e.g. octrees) are geared more toward compact, fatter objects.
- The structure is well suited for sparse, clustered data, for which it provides very good data.
- The data structure is very general, capable of mimicking the operation of other data structures. Thus it is flexible and can be applied easily to a wide variety of situations.

6.3. Future Work

Among the directions for further work are:

- A number of operations on L-blocks and their coverings could be defined. Though many operations are straightforward, some (such as reordering the hierarchy of a LBC) are quite challenging. In particular, determining a satisfying cost covering is challenging.
- It would be interesting to determine what benefits might be gained from expanding the data structure, e.g. by allowing multiple edge labels, or non-axis aligned (e.g. sheared) collections of voxel data.
- 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.

Acknowledgement

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

References

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

[2] Burton, B.P., B.H. McCormick, R. Torp, and J.H. Fallon. Three-dimensional reconstruction of neuronal forests. *Neurocomputing*, 38-40:1643-1650, 2001.

[3] Foley, J. D., A. van Dam, S. K. Feiner, J. F. Hughes. *Computer Graphics Principles and Practice, Second Edition in C*. Addison-Wesley, Boston, pp. 548-557, 1996.

[4] Jackins, C. and S. L. Tanimoto. Oct-trees and their use in representing 3-d objects. *Computer Graphics and Image Processing*, 14, pp. 249-270, 1980.

[5] Kong, T.Y. and A. Rosenfeld. Digital topology: introduction and survey. *Computer Vision, Graphics, and Image Processing*, 48, pp. 357-393, 1989.

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

[7] McCormick, B.H., B. Busse, Z. Melek and J. Keyser, *Polymerization Strategy for the Compression, Segmentation, and Modeling of Volumetric Data*. Technical Report 2002-12-1, Department of Computer Science, Texas A&M University, College Station, TX, December 2002. Available from http://research.cs.tamu.edu/bnl

[8] Overmars, M. H. and J. Van Leeuwen. Dynamic multi-dimensional data structures based on quad- and k-d trees. *Acta Inform.*, 17, pp. 267-235, 1982.

[9] Samet, H. The quadtree and related hierarchical data structures. *ACM Computing Survey*, 16, 1984.

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

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

[12] Thompson, J.F., B.K. Soni, and N.P. Weatherill, *Handbook of Grid Generation*, CRC Press, 1999, pp. 1-4.

[13] van den Bergen, G. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*, 2:4, pp. 1-13, 1997.

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