SEGMENTATION STRATEGIES FOR POLYMERIZED VOLUME DATA SETS

A Thesis

by

VENKATA PURNA DODDAPANENI

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2004

Major Subject: Computer Science

SEGMENTATION STRATEGIES FOR POLYMERIZED VOLUME DATA SETS

A Thesis

by

VENKATA PURNA DODDAPANENI

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

Bruce H. McCormick (Chair of Committee)	John Keyser (Member)
Lihong Wang (Member)	Valerie E. Taylor (Head of Department)

December 2004

Major Subject: Computer Science

ABSTRACT

Segmentation Strategies for Polymerized Volume Data Sets.
(December 2004)
Venkata Purna Doddapaneni, B.Tech., Indian School of Mines (I.I.T.)
Chair of Advisory Committee: Dr. Bruce H. McCormick

A new technique, called *the polymerization algorithm*, is described for the hierarchical segmentation of polymerized volume data sets (PVDS) using the L-block data structure. The L-block data structure is defined as a 3-dimensional iso-rectangular block of enhanced vertex information. Segmentation of the PVDS is attained by intersecting and merging L-block coverings of the enhanced volumetric data. The data structure allows for easy compression, storage, segmentation, and reconstruction of volumetric data obtained from scanning a mammalian brain at sub-micron resolution, using three-dimensional light microscopy (knife-edge scanning microscopy (KESM), confocal microscopy (CFM), and multi-photon microscopy (MPM)). A hybrid technique using the polymerization algorithm and an existing vector-based tracing algorithm is developed. Both the polymerized and the hybrid algorithm have been tested and their analyzed results are presented.

To my parents and my brother, for their guidance, support and love.

ACKNOWLEDGMENTS

I owe an immense debt of gratitude to my supervisor, Dr. McCormick, for his invaluable advice and careful guidance throughout the course of this thesis. I would like to thank Dr. Keyser for his support and patience for this work.

Preparation of this thesis was supported in part by grant Texas Higher Education Coordinating Board ATP grant 000512-0146-2001.

My appreciation goes to my other committee member, Dr.Wang for his ideas in improving my work. I would also like to thank Dr. Gutierrez and Dr. Choe for sharing their ideas during my work.

I owe a great deal to my fellow graduate students, Zeki Melek for the vizualization part, Brad Busse for helping with both the polymerization code and neuron tracing code. I would also like to thank my office mates, David Mayerich for Golgi and Nissl stained data and Prathyusha Aragonda for helping me with my work.

Lastly, to Anshul Kaushik and Shruti Rajagopalan, thank you for your patience, support, favors, and all the other things that make it so worthwhile to know you.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF FIGURES

Page

Page

Page

Page

Page

Page

LIST OF TABLES

Page

CHAPTER I

INTRODUCTION

Many techniques exist for generating biomedical volumetric data sets. Popular among these are magnetic resonance imaging (MRI), computerized tomography (CT), and more recently, three-dimensional light microscopy (using knife-edge scanning microscopy (KESM), confocal microscopy (CFM), and multi-photon microscopy (MPM)). Once the data sets have been obtained, segmentation of the data efficiently and reconstruction of the imaged tissue in three dimensions pose a daunting task. Methods that are appropriate for gross anatomy (including MRI and CT) do not scale well when applied, for example, to the microstructure of brain tissue, as obtained by three-dimensional light microscopy. Each source of data sets has its own characteristics and the three-dimensional reconstruction and data representations used must honor these characteristics. This chapter describes the characteristics of brain microstructure and the problems associated with representing such data using existing techniques. We overcome these problems in part with our new L-block data structure defined [1], [2], over polymerized volume data sets [3] as described below. Reconstruction of such neuronal data sets are also presented. The later part of the thesis explains a known method of neuron tracing using "vector tracing" [4], [5] and developes a hybrid technique that uses both the L-blocks approach and vector tracing to give improved results.

This thesis follows the style and format of *IEEE Transactions on Information Technology in Biomedicine.*

1. Goals of the Thesis

This thesis aims to provide *segmentation strategies* for polymerized volume data sets (PVDS) [3]. Polymerization adds edge labeling between neighboring voxels in the volume data sets defined over a regular three-dimensional grid system. Segmentation strategies are in general complex systems, not governed by a single algorithm. The development of such strategies consists therefore in both the selection of appropriate techniques among those available in the literature, as well as the development of novel ones as appropriate.

The goals of this thesis are:

a. Design segmentation strategies that provide good real-time data compression during data acquisition and can be efficiently extended to provide the basis for 3D modeling of brain microstructure.

b. Test these segmentation strategies using volume data sets built from graphical primitives

c. Test these segmentation strategies using submicron three-dimensional light microscopy data from the mouse brain.

2. Background and Rationale for the Polymerization Approach

The neuronal volumetric data sets whose segmentation is studied within this thesis have been edge-labeled (or "polymerized") to give a rough indication of whether the edge termini (voxels) are part of the same object in the underlying volume data set. These volumetric data sets are huge (in the order of terabytes), noisy, sparse, filamentary, and have long thin branching structures. Since the raw data is obtained from continuous serial sectioning of embedded mouse brain tissue, the storage and compression of the data must be synchronized with the sectioning process.

Existing representations fail to address some features of this kind of volumetric data sets. Most representations fail to address real-time data, where we have to

incrementally build the condensed data set. Some well known techniques and their drawbacks are described below. A detailed summary of most of these methods is explained by Winter [6].

Grid-sampled data: [7] This is the simplest representation consisting of the raw data set. This representation is the standard input format and no compression is involved.

Spatial-occupancy enumeration: [8]-[10] This is the simplest of the space-partitioning models. In this method each object in the volumetric space is divided and decomposed into regular cubic cells. This methods suffers from the excessive overhead required to store the entire data set in its memory for its construction.

Octree/quadtree: [11]-[14] An octree is the well known hierarchical spatial-occupancy enumeration scheme in which cells in a 3D volume data are regularly subdivided into regular octants. Depending on the position of the cell relative to the object, it is marked

- "full" if it lies totally inside the object
- "empty" if it lies totally outside the object
- "partially full" if it lies partially inside the object

Though the octree has the advantage of consuming less space for data representation, it still suffers from the precomputation cost. Like the earlier data structures, the octree cannot be used for incremental compression of the data. The quadtree approach is similar, except that it is applied for 2D image slices. When the data is sparce, the cost of storing this data structure can be more than the data itself.

BSP-tree: [9] Binary Search Partition tree introduced by Schumuker et al., is another type of volumetric representation. In this representation, each node of the tree subdivides the space into two parts (e.g., in 3D, by a plane). The leaves of this tree are convex regions bounded by planes in ancestor nodes, and are classified either inside or outside. The drawback of this approach is that it depends on the decomposition of the space rather than the structure of the object itself. Like the octree approach, this technique is not advisable when the data set is sparse.

kD-tree: [15] This is a hybrid between an octree and BSP-tree. It is a balanced binary-tree structure in which at each level the volume under the root node is sub-divided into cells, each containing a sub-volume. At each level the axis of sub-division is chosen such that each of the sub-divided part has the same number of points as the other. Similar to earlier approaches, it too suffers from the same drawbacks.

AABB-tree: [9], [16] An axis-aligned bounding box trees is a hierarchical collection of iso-rectangular boxes. Each parent node bounds its children. These data structures are often used for collision detection. Unlike the earlier spatial partition approaches, AABB-trees can overlap and share neighbours. Our data structure uses this approach, with the exception that polymerized data allows us to easily build L-block coverings incrementally and to maintain connectivity between nodes without having to go through a parent node.

As mentioned above most of the above representations are either slow, computationally require too much data to be held concurrently in memory, or ill-suited to model long, filamentary structures. 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. These concerns are addressed by the polymerization approach, which constructs L-block containers from volumetric data. The L-block container is a new type of data container, apart from the other containers like strings, vectors, arrays, etc. The polymerized algorithm is described in detail in the sections that follow.

The initial part of this work was done by Brad Busse. The L-block code for 3-connectivity was developed by him.

3. Overview of the Thesis

The organization of this thesis is as follows:

Chapter II defines and describes the containers for polymerized volume data sets: the L-block data structure, L-block coverings/partitions, and some operations on L-

blocks that are required during the process of reconstruction. Cost functions are defined for evaluating the quality of coverings/partitions of the volumetric data. The storage overhead required when converting uncompressed data from serially-scanned sections into L-block data structures is also described in this section.

Chapter III gives an overview and explanation for all the steps involved in the three-dimensional reconstruction pipeline for volumetric data, from generation of the data through visualization of the reconstructed data. Described are some of the preprocessing techniques tuned to specific sources of raw data (with some examples and results) that are performed before starting the reconstruction process. Also addressed are some common errors in the raw data and ways to suppress them for successful reconstruction.

Chapter IV describes and analyzes the results obtained on synthetic data sets by application of the strategies described in Chapters II & III. These synthetic data sets are built from graphical primitives, and serve as a benchmark to test the functionality and efficiency of the polymerization algorithm.

Chapter V describes and analyzes the results obtained on three-dimensional light microscopic data by application of the strategies described in Chapters II & III. The microscopic data includes both Nissl-stained and Golgi-stained mouse-brain tissue. These data sets were obtained from scanning the embedded tissue using the knife-edge scanning microscope (KESM) [17]-[19]. The segmentation results on these data sets are used for both evaluating the performance of the algorithm and also for comparison with the results on the data sets generated by graphical primitives of Chapter IV.

Chapter VI begins the second part of the thesis, which extends the polymerization algorithm [1], [2] to embrace vector-based tracing [4], [5]. The rationale for this extension is described in this section, along with an explanation of the vector-based tracing algorithm. Literature for vector-based tracing is reviewed.

Chapter VII describes certain matched filters used for edge detection, which form the basis for vector-based tracing. It also analyses the application of these filters on volume data obtained from Nissl- and Golgi-stained tissues, and on graphical primitives

(placed at different orientations). Discrete matched filters are also generated from modeling graphical primitives viewed in the image plane, first in geometric optics and then in diffraction-limited optics. These filters are then compared with the matched filters used in the paper by Al-Kofahi et al. [4], [5].

Chapter VIII contains a brief overview (theory) of the vector tracing algorithm as explained in the paper by Al-Kofahi et al. [4], [5]. Equations used for calculating points along the trajectory in the process of neuron tracing are stated and explained.

Chapter IX contains the criterion for selection of seed points for the initiation of vector-based tracing, for detection of branch points during the tracing process, and for stopping the vector-tracing process.

Chapter X parallels Chapter IV, but now with the addition of vector-tracing. The chapter describes and analyzes the results obtained by application of the strategies described in Chapters VI - IX on synthetic volumetric data sets. Based on the analysis of results for synthetic data, certain parameter optimization techniques for successful reconstruction of the volumetric data are discussed.

Chapter XI parallels Chapter V, but now with the addition of vector-tracing. The chapter describes and analyzes the results obtained by application of methods described in Chapters VI - IX on three-dimensional light microscope data, the same data used in Chapter V. We evaluate the improved segmentation provided by the addition of vector tracing.

Chapter XII summarizes the results of the thesis. It lists some areas of improvement and additions to be considered for future study.

CHAPTER II

L-BLOCK COVERING OF POLYMERIZED VOLUME DATA SETS (PVDS)

1. Polymerized Volume Data Sets

Polymerized volume data sets (*PVDS*) [3] are volumetric data sets with enhanced vertex information. Given a uniform n-dimensional grid, a PVDS is generated by storing extra information (edge-labeling) along with the grayscale value assigned to each vertex of the grid. Edges associated with each vertex (voxel) of the grid are given a Boolean label. *Active edges* are assigned the value 1 and the remaining edges, which are labeled *inactive*, are assigned the value 0. Given two vertices sharing a common edge, we can estimate from the edge-labeling information whether the two vertices are part of a common underlying object in the data set. We also define *isolated vertices*, as the vertices that have no active edges. These isolated vertices are generally "noise" in the volumetric data, often caused by staining artifacts. Depending on a threshold test of their gray scale values, they are either ignored or packed into small L-blocks [1][2]. The other vertices of the packaged L-block can be treated as "white-space". The voxels in the white space are ignored in subsequent three-dimensional image processing.

2. Data Structure

i. The L-Block

An *L-block* is defined as a k-dimensional axis-aligned iso-rectangular block of enhanced vertices. Each vertex of the L-block is a voxel with its gray scale value and associated connectivity with its neighboring voxels. Each L-block is defined with a set {<header><vertex array>}.

The <header> = {<position><template>}, is defined by:

- Position of its least vertex as indexed within the parent k-dimensional uniform grid. For a 3-dimensional uniform grid, this index would be (i, j, k) for the X, Y and Z dimensions respectively.
- Template, (l_1, l_2, ….l_k), where a (l_1, l_2, ….l_k) L-block implies a block of l_1 vertices in the first dimension, l_2 in the second, etc. For a 3-dimensional uniform grid, the template of the L-block is denoted by a triplet (l_1, l_2, l_3) where l_1 is the number of vertices that the L-block extends in the X-direction, l_2 vertices in the Y-direction, and l_3 vertices in the Z-direction. Figure 1 shows an example of a (3,3,2) L-block, where the red cylinders represent active edges emanating from the L-block. Note that not all vertices have active edges (cylinders) emanating from them; i.e., not all the vertices have active connections with their neighboring vertices.

The <vertex array> contains enhanced vertex information. Along with storing the intensity value assigned to each voxel of the volumetric data, we also store the "edge labels" mentioned earlier. Hence vertex array is defined by,

<vertex array> = {<grayscale value> <edge labels>}.

For a given vertex, the width of the <edge labels> depends on the number of edges emanating from the vertex, i.e. the connectivity of the vertex. For a three-dimensional grid, and assuming 6-adjacency, <edge labels> for the vertex (i,j,k) can be defined as (1,1,1), if there are outgoing active edges between (i,j,k) and (i+1,j,k), (i,j+1,k) and (i,j,k+1) respectively.

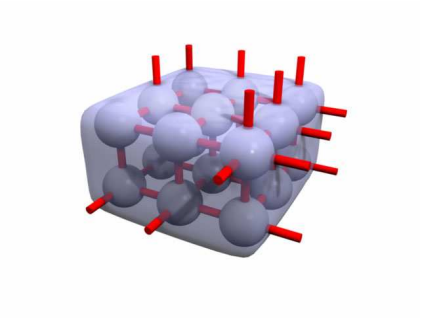


Figure 1. A (3,3,2) L-block. Cylinders represent active edges emanating from the L-block.

ii. L-Block Coverings and Partitions

Grouping L-blocks together, we define a hierarchical covering called a *L-block covering*. This hierarchical covering is based on the position and connectivity information of its constituent L-blocks. For a given set A of L-blocks, we define a covering of a volume, V, as $C(A,V)=\bigcup_{\alpha\in A} L_\alpha$, such that any active vertex in V is in the vertex array of some L-block, L_α. The leaf nodes for this hierarchical covering are (1,1,1) L-blocks. Figure 2 shows an example of a hierarchical covering.

L-block coverings may be combined. We can write a covering $C(A,V)$ in terms of other coverings,

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

A *partition P(A,V)*=$\Sigma_{\alpha\in A}\, L_\alpha$, by L-Blocks $\{ L_\alpha \mid \alpha \in A \}$, is a restricted form of covering P(A,V) = C(A,V), where A labels a partition in V. In this partition 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.

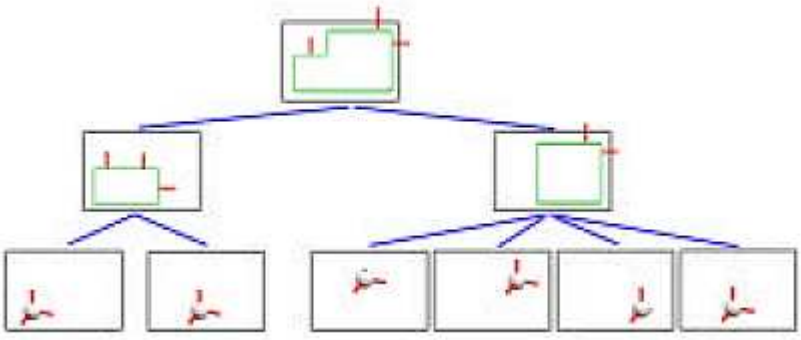


Figure 2. An L-block covering (top of the tree) is formed from the union of two other L-block coverings (middle row). Those L-block coverings are formed from unions of (1,1,1) L-blocks (on the bottom row).

iii. Cost of Coverings and Partitions

An efficient covering (or partition) should not contain excessive white space nor a large number of small blocks. We control the quality of a covering or partition by assigning a cost function to covering/partition, defined by

$$\$(C(A,V)) = \kappa + \Sigma_{\alpha \in A} \$(L_\alpha),$$

$$\text{with} \quad \$(L_\alpha) = \lambda + N_\alpha \mu.$$

Here κ is the cost of header information associated with the covering (irrespective of its constituent L-blocks); λ is a cost associated with the header for each L-block; μ is a cost for storing the grayscale value and edge information at each vertex; and N_α is the number of vertices in L_α.

Given this cost function, we try to minimize the cost of the covering for the given volume of data. This cost function serves as a measure of the trade-off when merging smaller L-blocks into larger ones, to form an efficient cover. As λ increases, coverings using fewer blocks of larger size are favored. So for $\lambda > \mu$, minimal coverings use fewer blocks while covering more white space. As λ decreases, minimal cost coverings use smaller blocks and cover less white space. When $\lambda = 0$, the covering of all active vertices by L-blocks with (1,1,1) template is of minimal cost, provided the (1,1,1) template is permitted.

iv. Storage Requirements

The total storage cost of storing a given k-dimensional L-block can be calculated in terms of the bits required by the set {<header><vertex array>}. For the <header>, along with the storage cost of the <position>, we also need to store the dimensions of the L-block. Let D_i be the number of bits required to store the size of PVDS in direction i, (0< i <k). Then the total number of bits, including all k-directions would be $D = \sum_{i=1}^{k} D_i$. So the number of bits required to store the header would be at least 2D.

For computing the <vertex array> we need to store the intensity value and the connectivity level for each voxel in the given data set. Let b_r (b_r is 1 for binary data, 8 for grayscale data, and 24 for color images) be the number of bits required to store the intensity value for each voxel and j be the local-connectivity level (adjacency set size = 2*j). For each sample voxel we need to store (b_r + j) number of bits, so for the entire data set we require (l_1 * l_2 *….* l_k) samples. Hence the total number of bits required by <vertex array> is $(b_r + j)\prod_{i=1}^{k} l_i$.

Similarly we can also calculate the storage requirement for an L-block covering. From the definition we know that a covering is made up of smaller coverings. Therefore the covering has to have a list (<sub-block list>) composed of pointers to all the coverings under it and also a <header>. The storage of the header would be the same as above, at least 2D bits. For the <sub-block list> we need bits (b_p) to store each pointer, total number of pointers (N) and bits (b_m) to store maximum number of pointers. So each <sub-block> list requires b_m+Nb_p bits.

v. Operations on L-Blocks

In order to reconstruct scanned tissue data, we define merge and intersection operations on the L-blocks. These two operations are the most relevant operations required for this work. Since we perform the reconstruction in an incremental fashion, starting from 2x2x2 L-blocks and subsequently building on these, merging is an important operation. Intersection is another important operation, used when we want to selectively examine that portion of the data in a given region of interest.

Merge/Union

The merge operation is mainly used for forming larger L-blocks or L-block coverings from smaller set of L-blocks or L-block coverings respectively. The result of merging two L-blocks will form another L-block. Suppose we are given two L-blocks L_α and L_β. Assuming that the header information in dimension i is given by position P_i

and template value t_i , the new position of the merged L-block will be the minimum of the two. This can be written as $P_{i,\alpha\beta} = \min(P_{i,\alpha}, P_{i,\beta})$ and the new template will be $t_{i,\alpha\beta} = \max(P_{i,\alpha} + t_{i,\alpha} - P_{i,\alpha\beta}, P_{i,\beta} + t_{i,\beta} - P_{i,\alpha\beta})$.

Once the merge operation is done, we have to take care of the values for each voxel in the resulting L-block. For the voxels which were not present in either L_α or in L_β, we assign them "empty" values and for the rest of the pixels retain their earlier values before merging. One drawback of assigning “empty” values for non-present voxels is, we might “accidently” reset the values of the voxels that might belong to a valid L-block. Figure 3, shows a snapshot of reconstructed Golgi-stained tissue data. Figure 4 shows a closer view of the reconstructed data, in which we can see the merged L-blocks. The colors on the blocks indicate different sizes of L-blocks.

We can define a merge operation for two L-block coverings in a similar fashion. The resulted L-block covering will contain the pointers to the input blocks as its sub-blocks or will contains a single list formed by merging both the pointers.

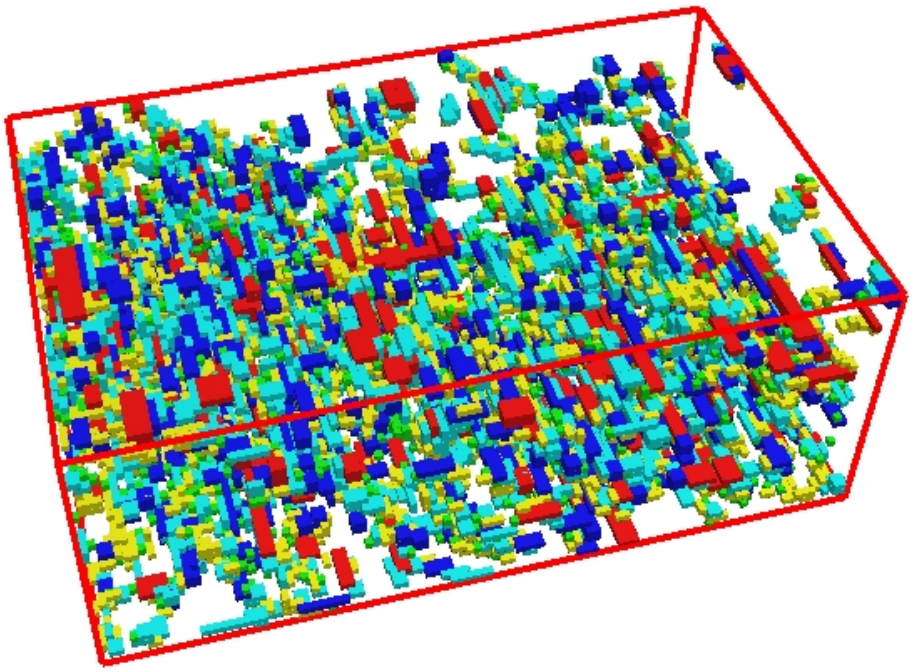


Figure 3. Snapshot of reconstructed Golgi-stained tissue data.

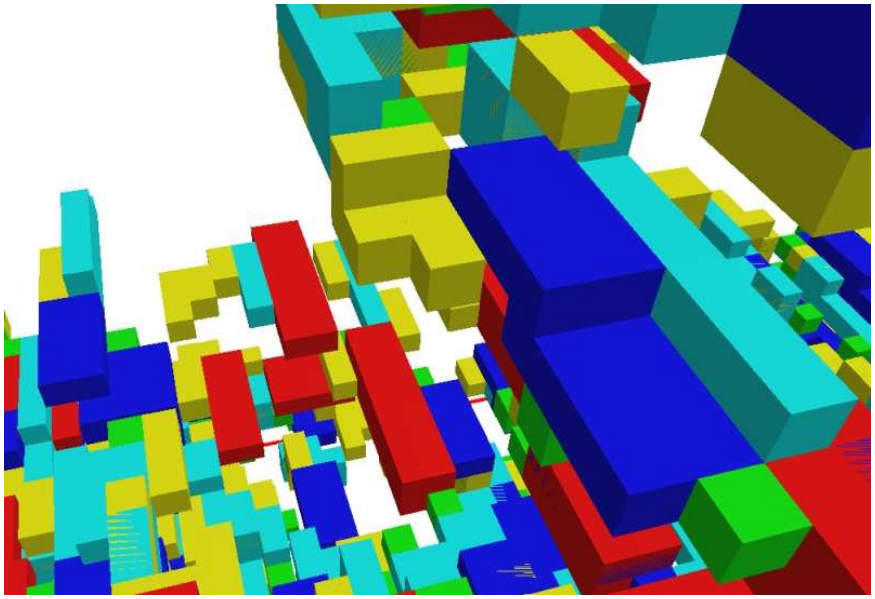


Figure 4. Closer view of the reconstructed Golgi-stained tissue data, showing the merged L-blocks. Different colors for the L-blocks are representative of their varying sizes.

Intersection

Intersection operation is mainly used when we want to see if two L-blocks have anything in common or want to examine the portion of data in a region of interest. For given L-blocks, L_α, L_β, we define the intersection as $L_{\alpha\beta} = L_\alpha \cap L_\beta$, which might be equal to Φ (an empty set). For L-block partitions this representation is unique: Let A and B be partitions with ($\alpha, \mu \in A \mid \beta, \nu \in$ B), then $L_{\alpha\beta} = L_{\mu\nu}$, implies $\alpha = \mu$, $\beta = \nu$.

3. Previous Work

The design and development of the knife egde scanning microscope was done under the direction of Dr. McCormick [17]-[19]. This instrument is the main source of the input date used for this research. The initial work of three-dimensional

reconstruction was started by Burton[20]. His work included reconstruction from smaller sets of synthetic data.

Development of the polymerization algorithm was started with the technical report by McCormick et al., [2]. This work included both the development of polymerized volume data sets and reconstruction of the initial scanned Golgi-stained tissue data sets. The raw data in this work was reconstructed using the polymerization code for three-connectivity developed by Brad Busse.

Further developments of the polymerization algorithm and the extension of the polymerization model to larger adjacency sets will be found in the papers by McCormick et al., [1], [21]. This work included the reconstructions of synthetic data, Golgi-stained tissue data and Nissl-stained tissue data. The three-dimensional reconstruction used 8- and 12- adjacency along with 6-adjacency.

All through this work, the visualization code used to interactively view reconstructed data and the reconstructed data was developed by Zeki Melek.

Raw input data (Golgi- and Nissl-Stained tissue data), used for this research was generated by David Mayerich using the Knife Edge Scanning Microscope [22]. The work regarding the specifications for storing volumetric data was supplied by Wonryull Koh[23].

CHAPTER III

RECONSTRUCTION PIPELINE

The entire reconstruction pipeline, from acquiring the raw data through visualizing the reconstructed data, is described in the following sections.

1. Data Acquisition

The raw data for this research is obtained from serial scanning of the embedded mouse brain tissue using the Knife Edge Scanning Microscope (KESM), a unique instrument invented and designed by Dr. Bruce McCormick at Texas A&M University [17]-[19]. The KESM is used for concurrent sectioning and scanning of a tissue embedded in a plastic block. It overcomes some of the drawbacks of optical sectioning associated with earlier scanning techniques like the Confocal Microscope (CFM). Unlike the Multi-Photon Microscope (MPM), the KESM can image non-fluourescent stained tissue. Once fully functional, the KESM will scan a mouse brain in 95 hours, generating 27 terabytes of data (mouse brain + clear plastic). Detailed description of the functionality of the instrument is given by David Mayerich [22]. Two sets of volumetric data were generated by the KESM: The first from scanning Golgi-stained tissue, where the entire neuron stains, and the second from Nissl-stained tissue, where only the cell bodies and nuclei stain. Apart from the data obtained from the KESM, we also tried our reconstruction pipeline on CFM data and also on synthetic data.

2. Background Noise Filtering

The raw data obtained is noisy due to the artifacts in tissue staining techniques and scanning procedures. The types of noise in the data can be reduced contrast, varying contrast (both within a given sectional image and also between subsequent

images), smears, and small specs of stain displaced in the sectioning process. Before we start processing these slices, we must make them relatively clear of noise. There are many noise removal techniques, but no one universal technique works for all kinds of noise. The techniques that worked for us are described below:

i. Mexican Hat Filter

This kernel is centered at each pixel of the image. The image obtained after application of the filter is the result of replacing the center pixel grayscale value (pixel in consideration) by the average of itself and its neighborhood, where the neighborhood grayscale values are multiplied by a set of integer weights. An example of a 7x7 Mexican Filter is given in Figure 6. The kernel in Figure 6Figure 6(a) is an ideal 7x7 Mexican Filter, while the one in Figure 6(b) is an approximation of the same. The amount of the neighborhood considered depends on the dimension of the kernel used. The weights shown in the Figure 6 are multiplied by pixels grayscale values surrounding the central pixel and the sum obtained is to be normalized by dividing by the sum of the positive weights (in this case 52). As you can see in Figure 6(a), this filter has a highly positive center region and a negative "surround". Due to this property, in the region of the image that is uniform in gray level, applying this kernel reduces the gray level to zero. A discontinuity in the neighborhood contributes a non-zero value. This value might be positive or negative depending on where the central point lies with respect to the discontinuity. The negative annulus serves as a natural background subtractor. We used this filter for image stacks obtained from CFM, to remove background noise (due to the light scattering from the tissue below the layer) present in the images. Figure 5 shows the effect of applying the Mexican Hat filters of dimensions 3, 5 and 7 on the original images (Golgi-stained image and CFM image). From the figure it can be seen that, as the dimension of the filter is increased, the image gets blurrier and also gets rid of the background noise. This process aided us in setting up proper threshold, to extract the required data.

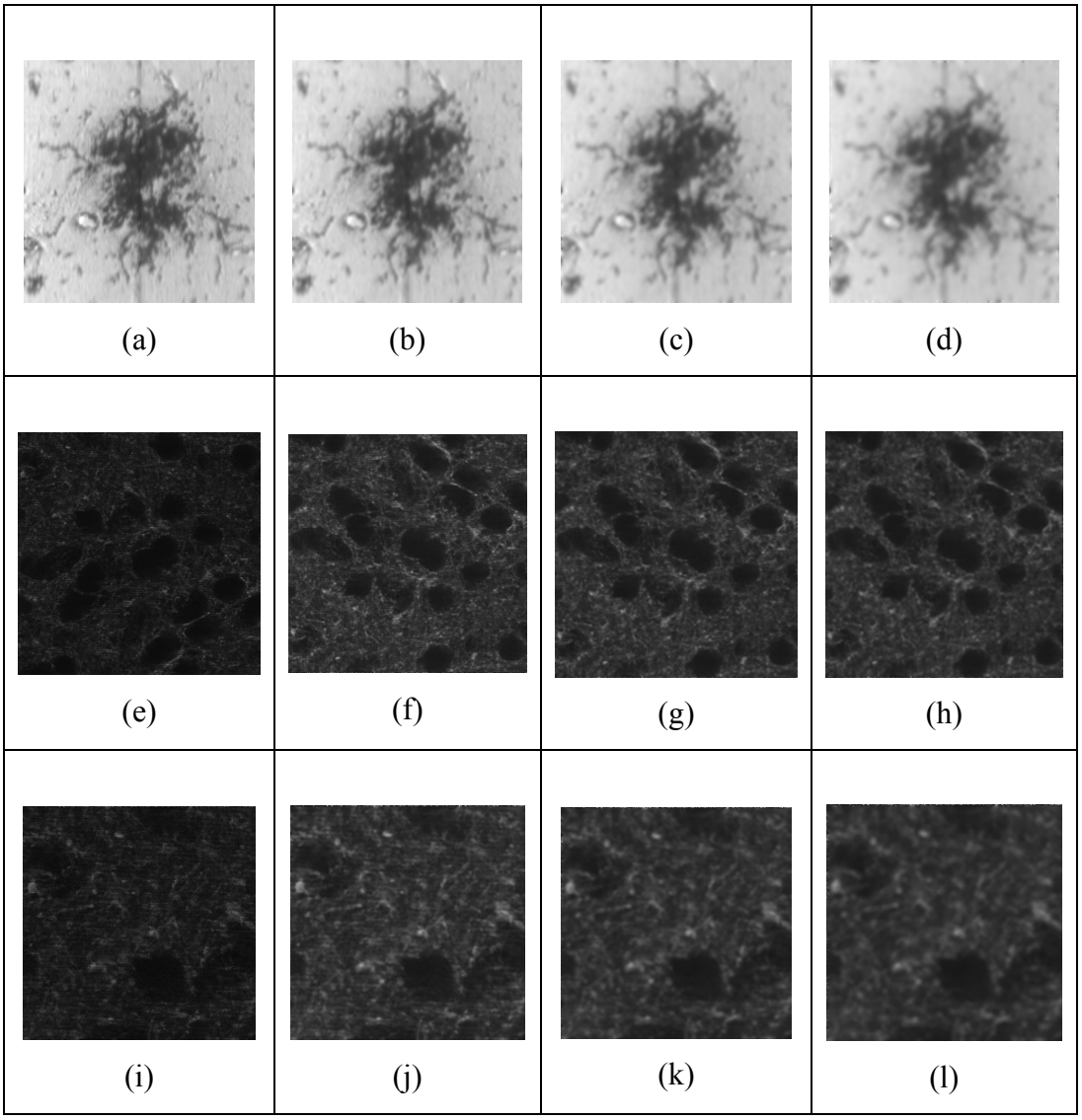


Figure 5. (a) 10X image of a slice of Golgi-stained tissue data; (b), (c) and (d) result of applying Mexican Hat filter of dimensions 3, 5, and 7 respectively on (a); (e) image from CFM; (f), (g) and (h) result of applying Mexican Hat filter of dimensions 3, 5, and 7 respectively on (e); (i) image from CFM viewed at a closer distance; (j), (k) and (l) result of applying Mexican Hat filter of dimensions 3, 5, and 7 respectively on (i).

ii. Multiplying Images

This is a multiplicative operation on the images, taken two at a time. The pixels in the output image are the product of pixels in the first image multiplied by the corresponding pixels in the second image. This technique fills in missing information

between sections and also enhances connectivity information. For example, assume that a thin fiber runs through more than 3 sections (images) and that due to some scanning artifact, the corresponding pixel values were not scanned in image 2, but were obtained in the other images. When we multiply the images 1 & 2 (or 2 & 3) i.e., multiply the corresponding grayscale values in both the images and then normalizing the value thus obtained, the resultant image will have that pixels marked. The result of multiplying two images can be seen in Figure 7. In the figure it can be seen that, even if the data were missing in one of the images, it can be filled in by the other. This method helps us recovering data that had been lost due to scanning artifacts.

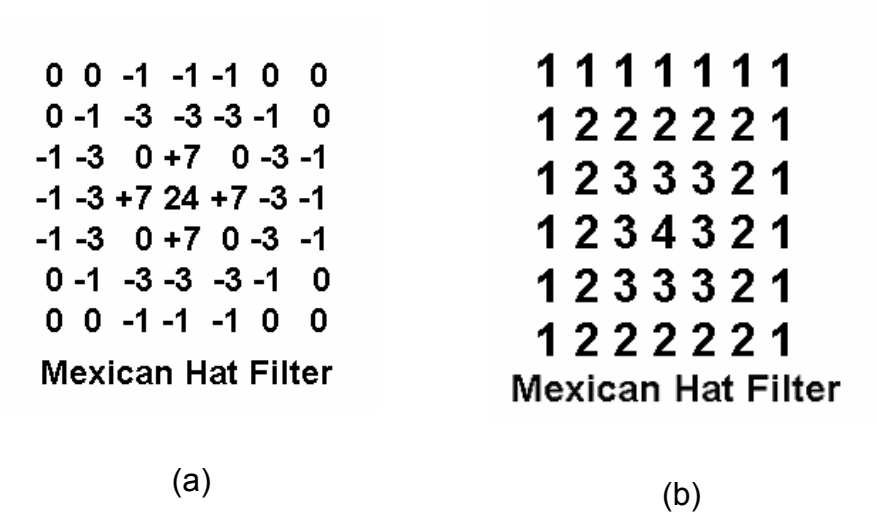


Figure 6. (a) 7x7 Mexican Hat filter; (b) 7X7 Mexican Hat filter (approximation).

iii. Median Filter

It is a filter, which gives better results than a mean filter in preserving useful data in the image. Instead of simply replacing the pixel value with the *mean* of neighboring pixel values, it replaces it with the *median* of those values. In Median Filtering first the size of the neighborhood is decided and then the center pixel is replaced by the median of the values present in the neighborhood (after initially sorting

them). This process helps remove unwanted noise in the form of small spikelike isolated values and also is a good representation of the surroundings. Since it takes good care of the surroundings, median filtering preserves edges in the image. In the presence of noise this technique blurs the images. We used this filter on the images obtained from scanning Golgi-stained tissue, basically for two purposes: First, to remove the unwanted small specks of noise present in the images, and secondly to identify and pick out darker areas in the image for reconstruction of cellbodies and thick fibers. Figure 8 shows the result of applying Median Filters of dimensions 3x3, 5x5, 7x7, 9x9 and 11x11 respectively. As the size of the kernel increases, the image gets blurrier and retains only the cell bodies, getting rid of the background.

iv. Image Homogenization

Homogenization can enhance details in the underexposed parts of an image. This procedure has been implemented so that it compensates image-inhomogeneity to a certain degree by means of equalizing the image's local mean and local variance in a running window of the target object size. We homogenized some images, as the images that were obtained had intensity variations due to nicks in the diamond knife-edge and knife-chatter. Also the illumination by the light was not uniform, resulting in low intensity and high intensity regions. Since the position of the knife-edge relative to the camera is fixed, this noise or pattern is uniform in all the images. So homogenizing these images by sampling lines of images along both the axes and then subtracting the mean from each of them, helps in cleaning the noise. An example of homogenizing an image is shown in Figure 9. Homogenization not only helps with setting up a proper threshold, but also enhances the information in the data set. As can be seen in Figure 9, the cell bodies that could not be identified earlier due to the intensity of light were brought out by homogenization.

v. Contrast Enhancement

Contrast enhancement can help enhance image data obtained from scanning Golgi-stained brain tissue. It brings out the fine detailed fibers that might be lost due to

the higher contrast of the cell-bodies. Figure 10 shows both the Golgi-stained image and the result of applying contrast enhancement on it. Figure 11 shows contrast enhancement, but applied on Nissl-stained tissue. In both the cases, apart from getting rid of the background noise, the filter picked up fine details that exist in the images.

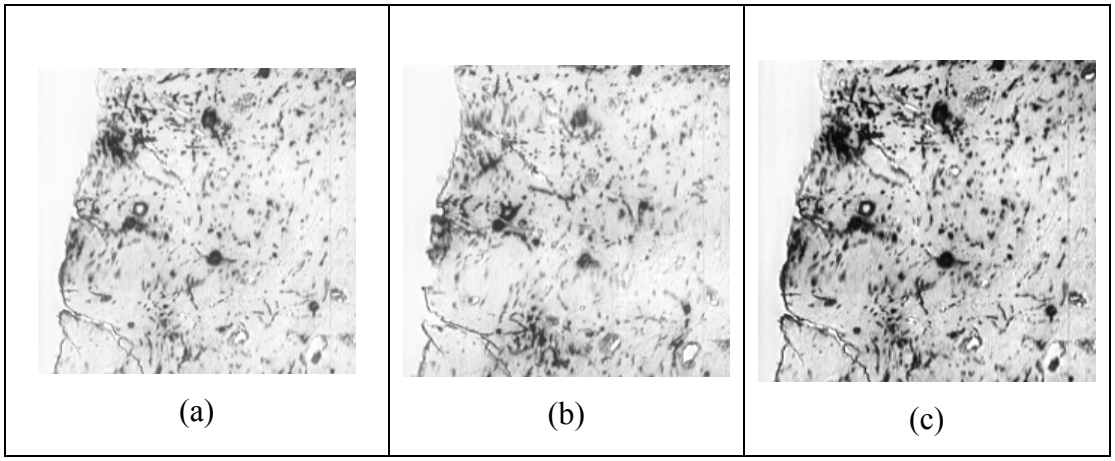


Figure 7. (a), (b) 10X images of successive slices of Golgi-stained tissue data; (c) result of multiplying images (a) and (b).

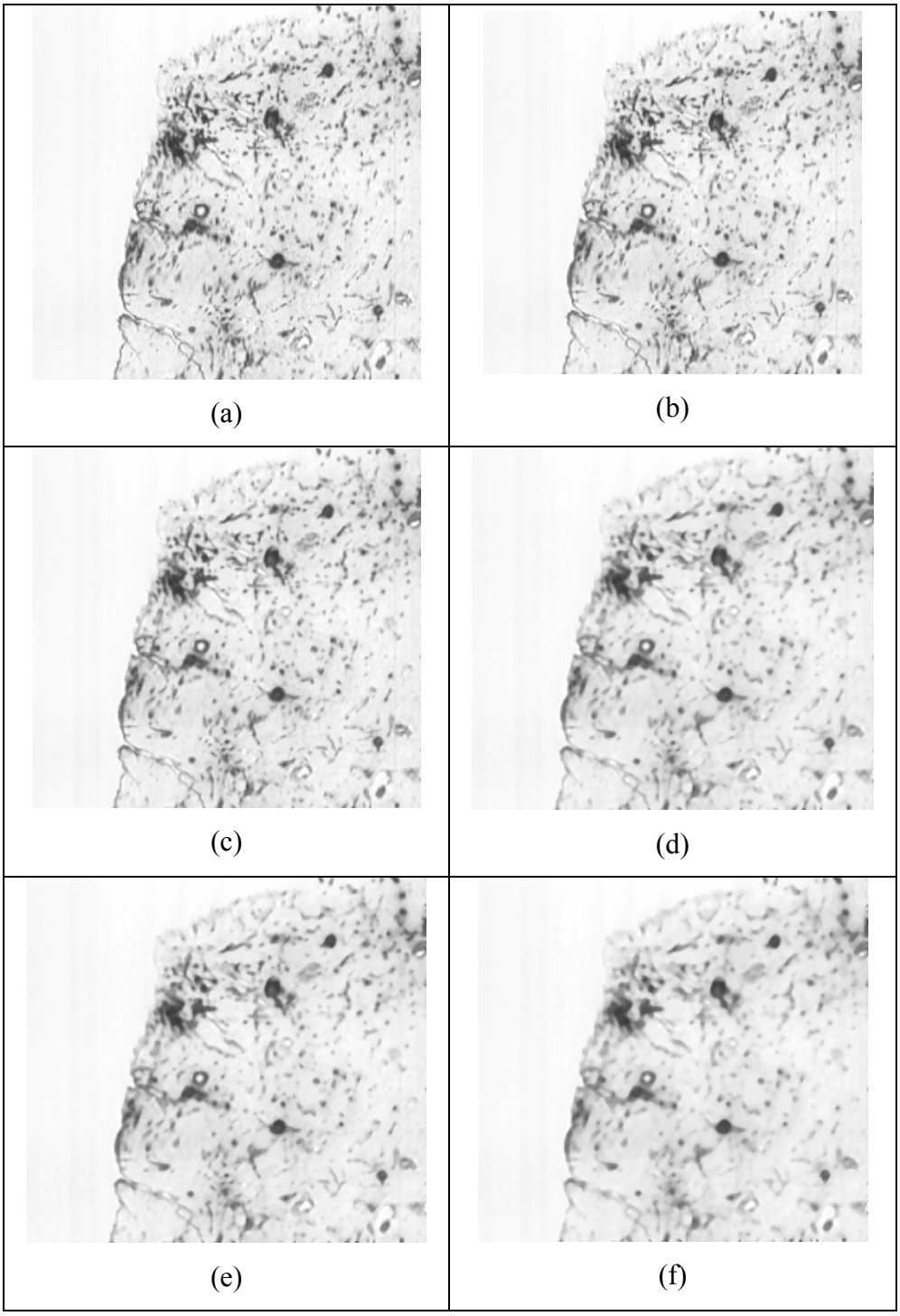


Figure 8. (a) 10X image of a slice of Golgi-stained tissue data; (b), (c), (d), (e) and (f) result of applying Median filter of dimensions 3x3, 5x5, 7x7, 9x9, and 11x11 respectively on image (a).

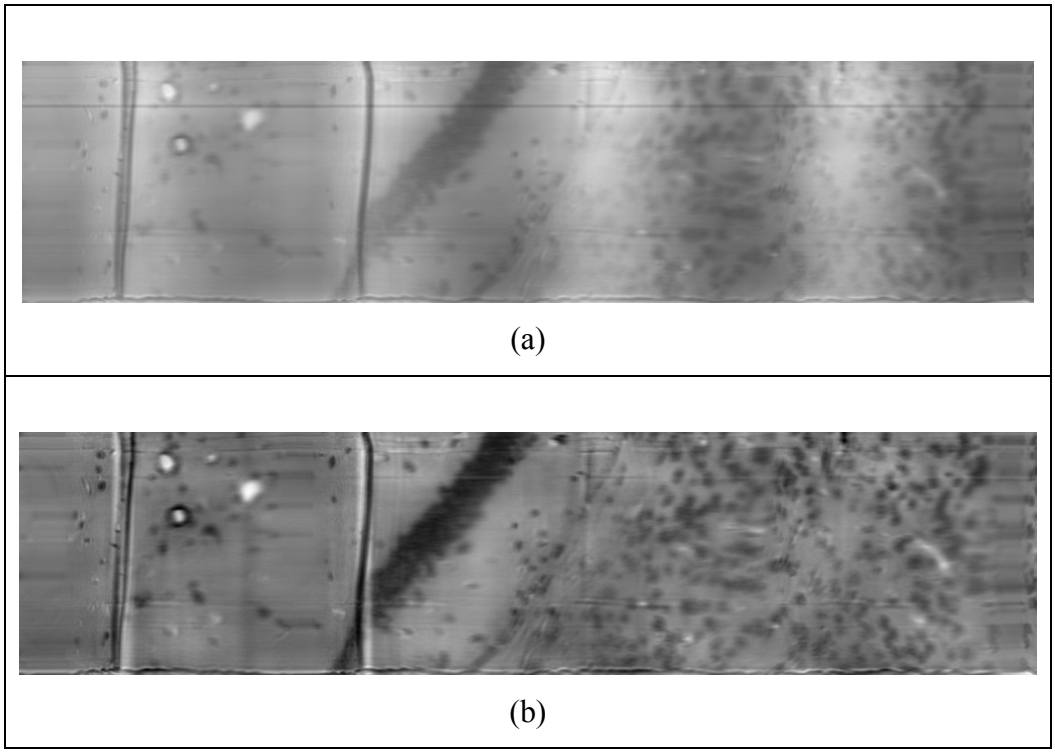


Figure 9. (a) 10X image of a slice of Nissl-stained tissue data; (b) result of applying image homogenization on image (a).

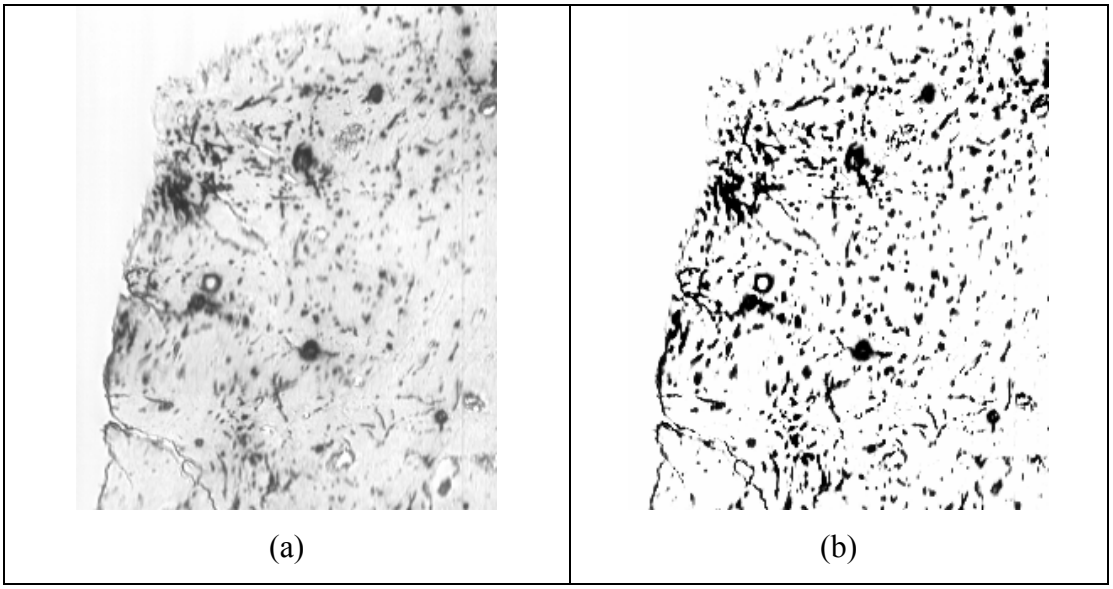


Figure 10. (a) 10X image of a slice of Golgi-stained tissue data; (b) result of applying Contrast Enhancement on image (a).

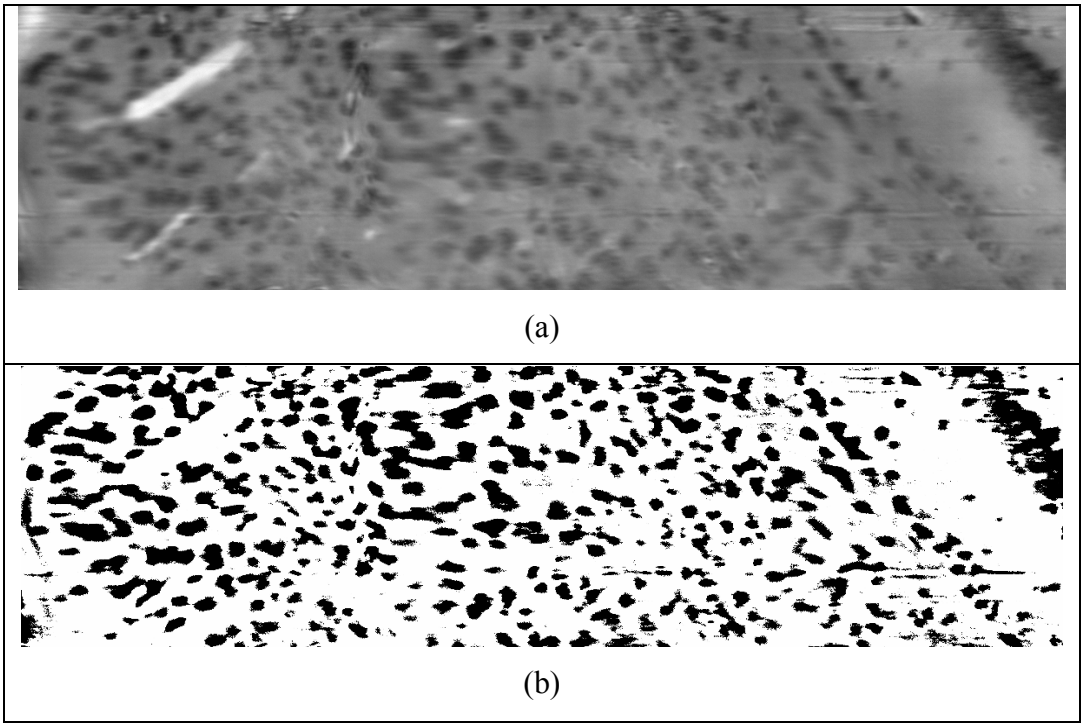


Figure 11. (a) 10X image of a slice of Nissl-stained tissue data; (b) result of applying Contrast Enhancement on image (a).

3. Polymerized Volume Data Sets

Once we get the filtered volume data set, we construct a Polymerized Volume Data Set (PVDS) by assigning Boolean values for selected edges between vertices of the three- dimensional volumetric grid [McCormick et al., [3]. Depending on the chosen local-connectivity we label the edges between the considered vertex and its adjacent vertices either active (a value of 1) or inactive (a value of 0) using adjacency-dependent discriminant functions [24]. For example a vertex in PVDS can be thought of as having links that extend to the neighboring vertices along the three coordinate axes, as shown in Figure 12. These discriminant functions are obtained by supervised-learning on a sample volume data set. The edge labeling technique provides independent information about whether two vertices sharing a common edge belong to the same underlying object or not. Labeling also aids in converting connected voxels into connected

components, thus facilitating segmentation of embedded objects in the volume data set and also for topological analysis of the relevant data [25], [26]. The PVDS associates each vertex with {<gray scale value> <edge labels>}, where <gray scale value> is its initial gray-scale value and <edge labels> is the Boolean vector containing the activity level of the edges emanating from the particular vertex. The number of emanating edges depends on the adjacency set being considered (6-, 8-, and 12- adjacency typically) and this adjacency is fixed for the entire volume data set. For example in Figure 12, considering 6-adjacency and that the position of the pixel(in red), is (i, j, k) and the pixels (in green) at (i+1, j, k), (i, j+1, k) and (i, j, k+1) are active. After forming the PVDS all the active edges are marked active (thick blue cylinders) and the rest as inactive (thin green cylinders). If we consider 12-adjacency then we have to consider connections across edges.

4. Data Compression and L-Block Formation

As mentioned earlier there is a continuous flow of raw data, which necessitates faster compression techniques and also less memory overhead. To overcome this, at a given time we maintain two consecutive sectional images in memory and discard both of them after the processing described below. As seen in Figure 13, we store two images, the most recently scanned image (Image 1) and the current scanned image (Image2) in memory. Noise reduction techniques are applied on the considered two images, and the edges emanating from each pixel in the resultant images are labeled. Data in the volume data set that do not satisfy the threshold test are considered as “whitespace”. Our data structure is used to represent data that is not “whitespace”. Using the labeled edges from the two image slices, we form 2x2x2 L-blocks [1], [2] by considering alternating pixels both in the X and Y directions. For each of these voxels an L-block is formed, if the edges in the X, Y and Z directions to that pixel are labeled active. After the processing of all the slices, we will be left with a list of 2x2x2 L-blocks on which further processing is performed.

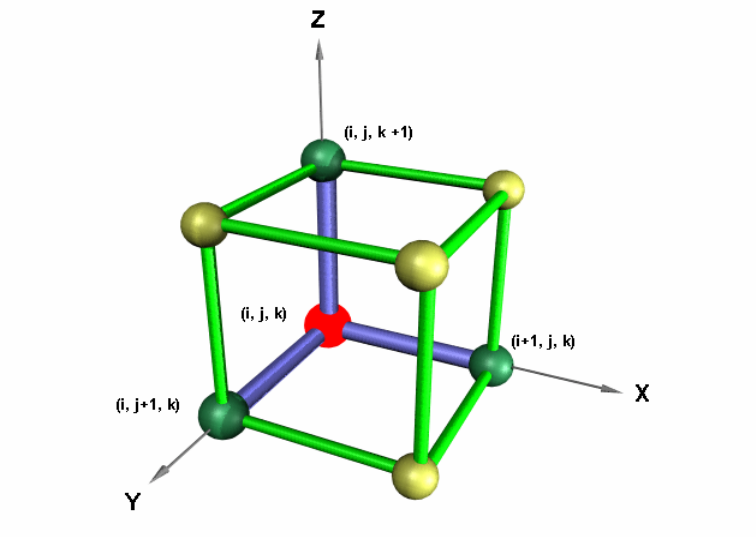


Figure 12. A 2x2x2 L-block with active edges in X, Y, and Z directions; edges in blue represent active edges and edges in green are inactive edges.

5. Cluster Formation

We generate clusters from the list of 2x2x2 L-blocks, by using the connectivity information of each voxel in the L-block. A cluster here is defined as an interconnected group of L-blocks.

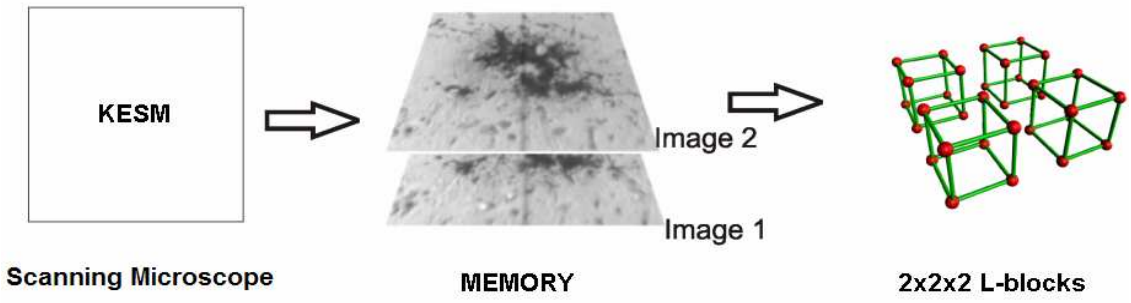


Figure 13. Sectional images from the KESM are stored in memory, two at a time; the processed data is stored as a list of 2x2x2 L-blocks.

6. Noise Removal

The compression achieved and the quality of the data depends on the staining technique used, the selection of discriminant functions, and the density of data present. To further improve the quality of the data, so as to reduce the data storage and to improve quality of reconstruction, we employ two types of noise reduction techniques.

Noise Removal – I: This type of noise removes smears or stains that exist in single layers. These smears are the sectioning artifacts resulting from disturbances in the KESM stage while sectioning the tissue. We can remove these by looking and testing another cluster on a different layer at a specific distance.

Noise Removal – II: This type of noise removes small flecks in the data. This noise that might have been generated during the scanning procedures. We can remove this type of noise by removing all clusters that have less than a specified number of L-blocks and that do not have another cluster within a specific distance away. Clusters that are small in size and at a specific distance away from the cluster in consideration, are also deleted in this process.

7. Combining L-Block Clusters

Clusters generated so far consist of only 2x2x2 L-blocks. In order to facilitate computation time and to reduce the number of L-blocks to be considered, we combine smaller L-blocks into bigger ones, depending on the connectivity information and the spatial separation of the L-blocks. The process of combining smaller blocks can add on some empty space, helping forming larger L-blocks and also reduce storage space needed for storing the positions and dimensions of smaller L-blocks. A cost funtion is used as a decision function in combining smaller L-blocks into larger ones. This cost function ensures that we do not add on unnecessary empty space when combing the blocks. For example, where we have the L-blocks forming a diagonal thread-like

structure, we might form a block, which has this thread as its diagonal. The cost function used for processing is based on the two factors, the amount of empty space that is being added and the memory overhead that is required when forming a bigger block. In the process of combining we start with a single L-block and parse through the X, Y and Z directions of the L-block looking for adjacent blocks that can be combined with this block into a bigger block. This process is stopped as we reach the end of the initial L-block list.

8. Thread Generation

During the entire process, from scanning the tissue to the formation of bigger blocks, there are cases where we might loose some valid data. This loss of valid data causes gaps in the data thus obtained. We try to fill these gaps in the segmented L-block data during the process of thread generation. This is done by expanding/dilating the L-blocks within each cluster till they overlap with the adjacent blocks and then combine them by effectively filling in the missing data. In this process we form an expanded connectivity graph, by linking the overlapping dilated L-blocks. To capture the major dendrite threads, this expanded connectivity graph is then simplified into a tree format by deleting all the finescale details (momentarily and added back in later on). Graph algorithms are employed to further simplify this graph and a hierarchical L-block structure is created around this graph.

9. Visualization

Visualization of the reconstructed data is performed by generation of an isosurface. Here we model, not just visualize, the neurons [27]. Taking the advantage of L-blocks, we generate an isosurface for each of the L-blocks separately, which is largely parallelizable. The hierarchical nature of the L-block data structure allows raytracing to be used for visualization. For interactive isosurface generation and display we have used isosplats [9]. The surface continuity to the neighboring L-blocks is guaranteed by using the connectivity information already generated in the PVDS.

CHAPTER IV

L-BLOCK COVERINGS OF VOLUME DATA SETS BUILT FROM GRAPHICAL PRIMITIVES

The polymerization algorithm [1], [2] is tested here on synthetic data sets, before applying it on biological volumetric data. This strategy gives us two advantages: knowledge of the expected results, and secondly, control over the complexity of the input data. In this chapter the performance of the algorithm will be evaluated, varying the complexity of the input data set.

Synthetic data sets used for testing purposes have been divided into two categories: *filamentary data sets*, and *blob data sets*. Similar data sets are encountered in the scanned mouse brain tissue. For each data set we have provided its visualization generated by commercial software, Amira [28]. This software package constructs an iso-surface generation for a given data set.

In generating the synthetic data we differentiate between imaging in *geometric optics* Vs imaging in *diffraction-limited optics*. Imaging in geometric optics gives us images of the graphical primitives, if appropriately large, e.g., as seen by the naked eye. Imaging in diffraction-limited optics takes into consideration the diffraction of light during the scanning process as it passes through the microscope objective. This diffraction introduces certain blur into the scanned images. To generate this effect we use a two-dimensional point-spread function applied to each two-dimensional image in the synthetic data set (see Chapter VII). This strategy is appropriate for knife-edge scanning microscopy, while confocal microscopy and multi-photon microscopy require using a three-dimensional point-spread function.

1. Filamentary Data

Synthetic filamentary data have been designed that resembles scanned Golgi-stained mouse brain tissue data. Golgi stains both cell bodies and neuronal arbors (axons and dendrites). These thin arbors, in turn give rise to filamentary data. The simulated data sets include fine fibers, parallel fibers, fiber bundles, and neuropil-like mats of fine fibers. Each is described in turn below.

i. Fine Fibers

Fine fibers in the synthetic data stem from thin arbors: dendrites (that become thinner and fade as they move away from their initiation point at the cell-body) and axons. Two types of data sets are used for testing fine fibers: the first shown in Figure 14, consists of fibers such as seen in the initial segments of a dendritic tree, with considerable diameter (their diameter here is around 6 pixels). The second, shown in Figure 15, consists of fine fibers finer than in the first case, as seen in axons with diameters of a couple pixels. Axonal images may also have gaps within the fibers.

Figure 14 shows the reconstructed synthetic data set both using Amira software and the polymerization algorithm. To increase the complexity of the data, each fiber is given a curvature. This aids in testing the algorithm for fibers in mouse brain tissue, where none of the fibers are straight cylindrical structures.

Figure 15 shows the reconstructed results of the second set of synthetic data, consisting of fine fibers with gaps along them. These gaps were induced into these fine fibers to reflect scanning and staining artifacts. From these figures, it can be seen that the polymerization algorithm, by bridging more gaps, works slightly better than the Amira reconstruction. Later as described in Chapter VI we will get yet better results using a hybrid approach. Figure 15(g) shows that the thread generation breaks, as there are insufficient L-blocks to trace these threads completely. Using higher connectivity, such as 6- or 12- adjacency on these images, does not help the reconstruction, since the gaps between fibers are large.

ii. Parallel Fibres

Figure 16 shows an image of parallel fibers, taken from Reference [29]. This image provides the rationale for testing the polymerization algorithm on parallel fibers in a synthetic data set generated to resemble those of Figure 16. Figure 17 shows the reconstructed synthetic data set both using Amira software and the polymerization algorithm. To test the performance of polymerized algorithm these structures were made relatively long and dense.

Figure 18 shows the performance of the polymerization algorithm when there is a gradual decrease of distance between the parallel fibers. All images are viewed in geometric optics.

Invariance of the polymerization algorithm under rotation has been tested by running the algorithm on synthetic data sets with parallel fibers placed at different orientations. Figure 19 plots Number of L-blocks Vs Volume of L-block (in log base 2), for fibers placed at different orientations. The plots, except for the 75° case, follow a similar pattern.

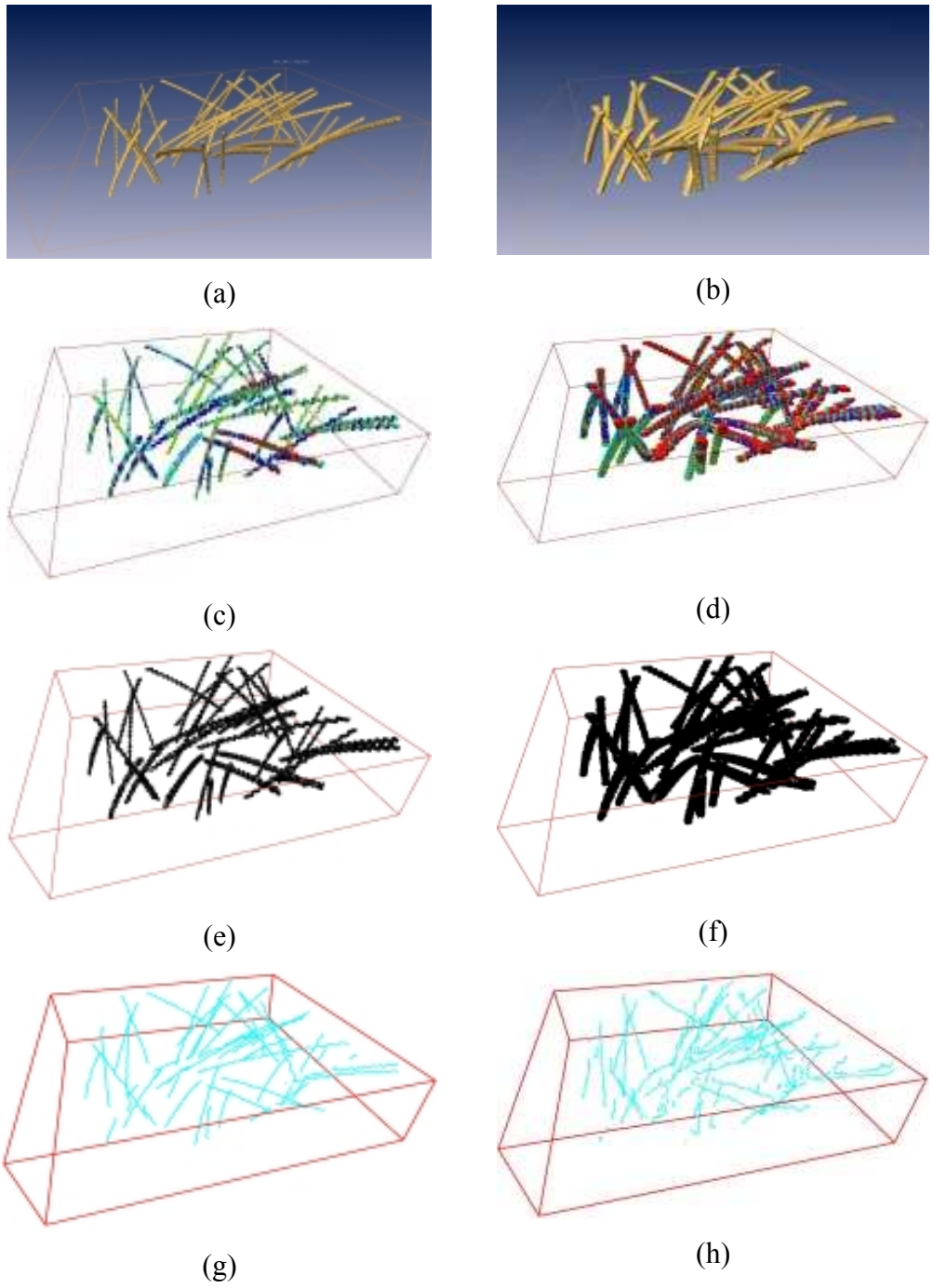


Figure 14. (a) Amira reconstruction of fine fibers; (c) colored L-blocks where the colors represent different sizes; (e) contents of reconstructed L-blocks; (g) the result of thread generation after polymerization. Similarly (b), (d), (f), and (h) show reconstructions of the same fibers, as imaged in diffraction-limited optics.

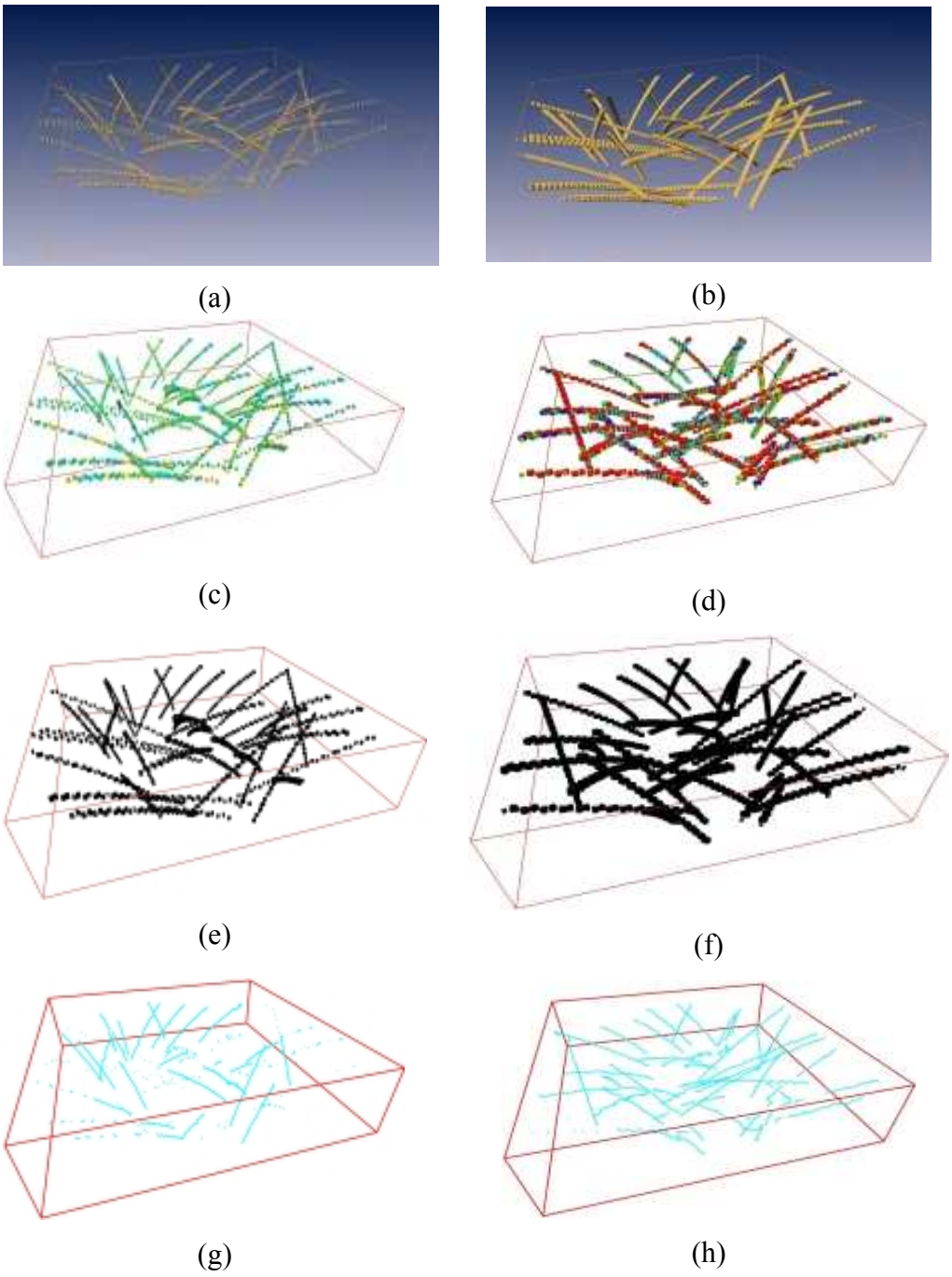


Figure 15. (a) Amira reconstruction of fine fibers with gaps; (c) colored L-blocks where the colors represent different sizes; (e) contents of reconstructed L-blocks; and (g) result of thread generation after polymerization. Similarly (b), (d), (f) and (h) show reconstructions of the same fibers, as imaged in diffraction-limited optics.

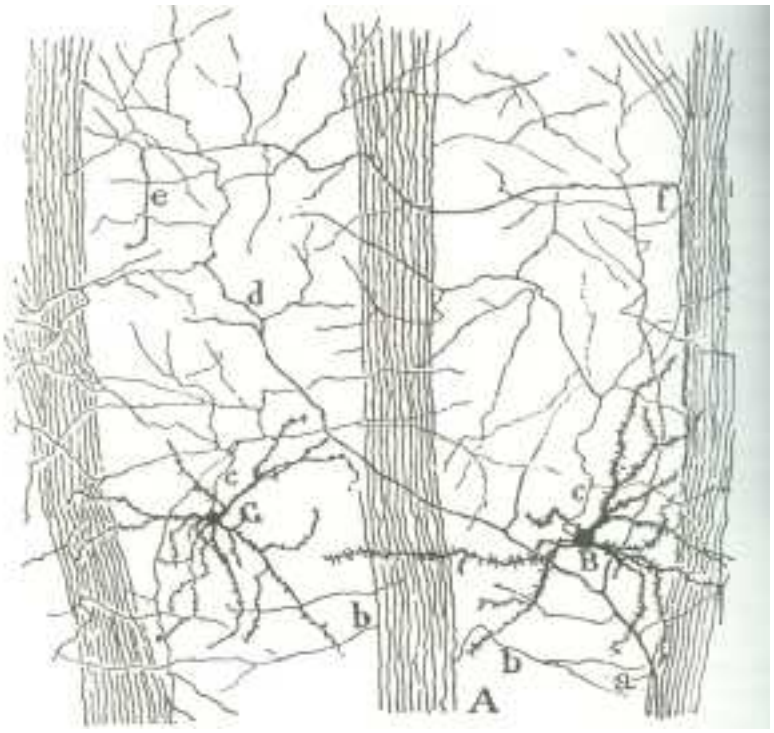


Figure 16. Part of a sagittal preparation through the corpus striatum of a several-day-old rabbit. (Source: Ref. [29]; reproduced with permission of the publisher)

iii. Fibers with Branching

Figure 20 shows the images drawn by Santiago Ramón y Cayal showing the sometimes bundled nature of neurites. Both these pictures, drawn from Golgi-stained tissue, were taken from Reference [29]. Figure 21 shows results of reconstructing synthetic data. Synthetic data was generated to resemble the branching fibers in Figure 20(a) and to test the performance of the polymerization algorithm. This data set aids us in testing how effectively the algorithm adapts to random nature of fibers for different orientations and for dense fibers (large number of fibers within a given volume). Figure 21 shows the reconstruction of the data set using both Amira software and the polymerization algorithm. The results show that the algorithm performs well when it encountered dense and branching fibers in an ideal and clean data set.

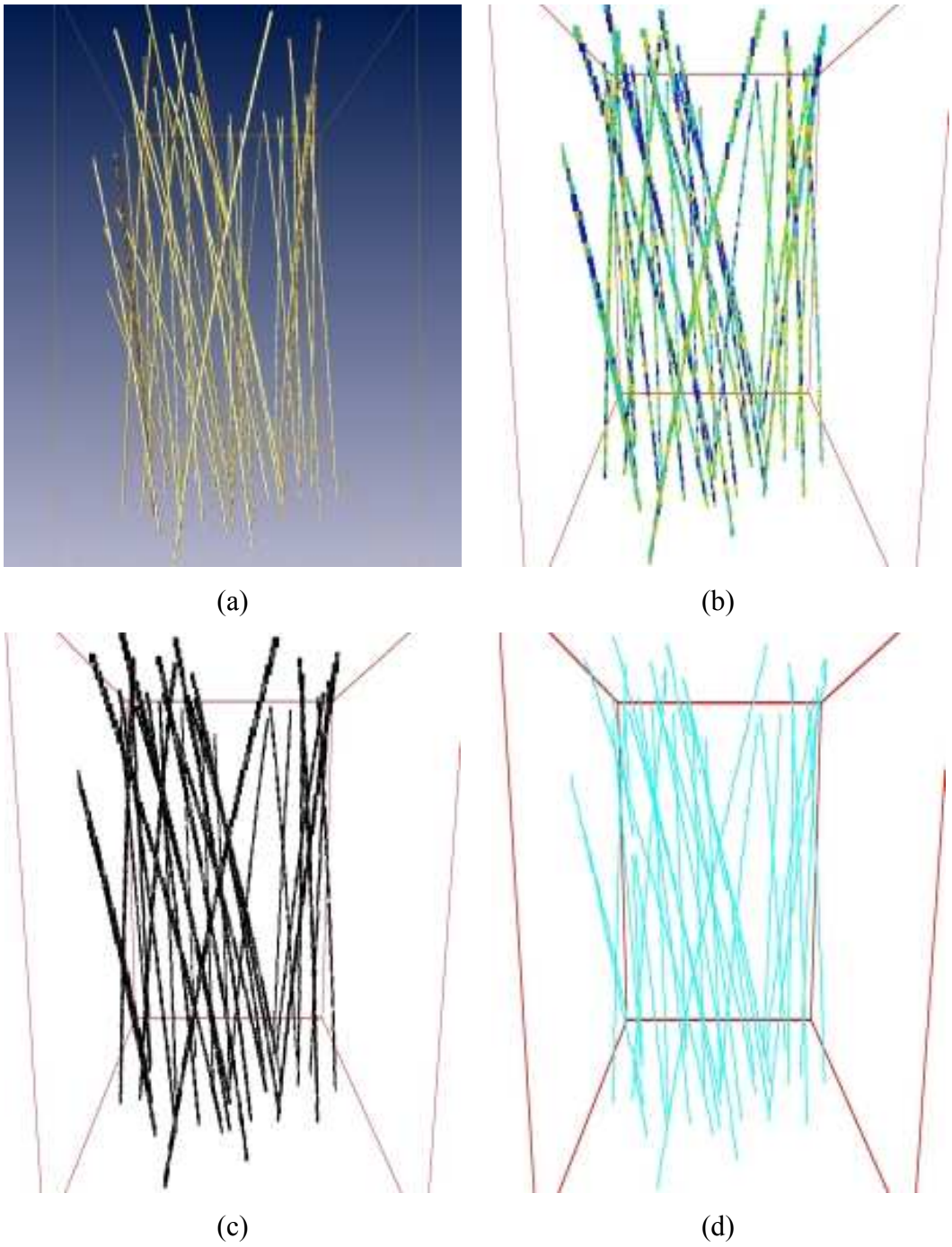


Figure 17. (a) Amira reconstruction of dense parallel fibers; (b) different colored L-blocks representing different sizes; (c) contents of reconstructed L-blocks; (d) result of thread generation after polymerization.

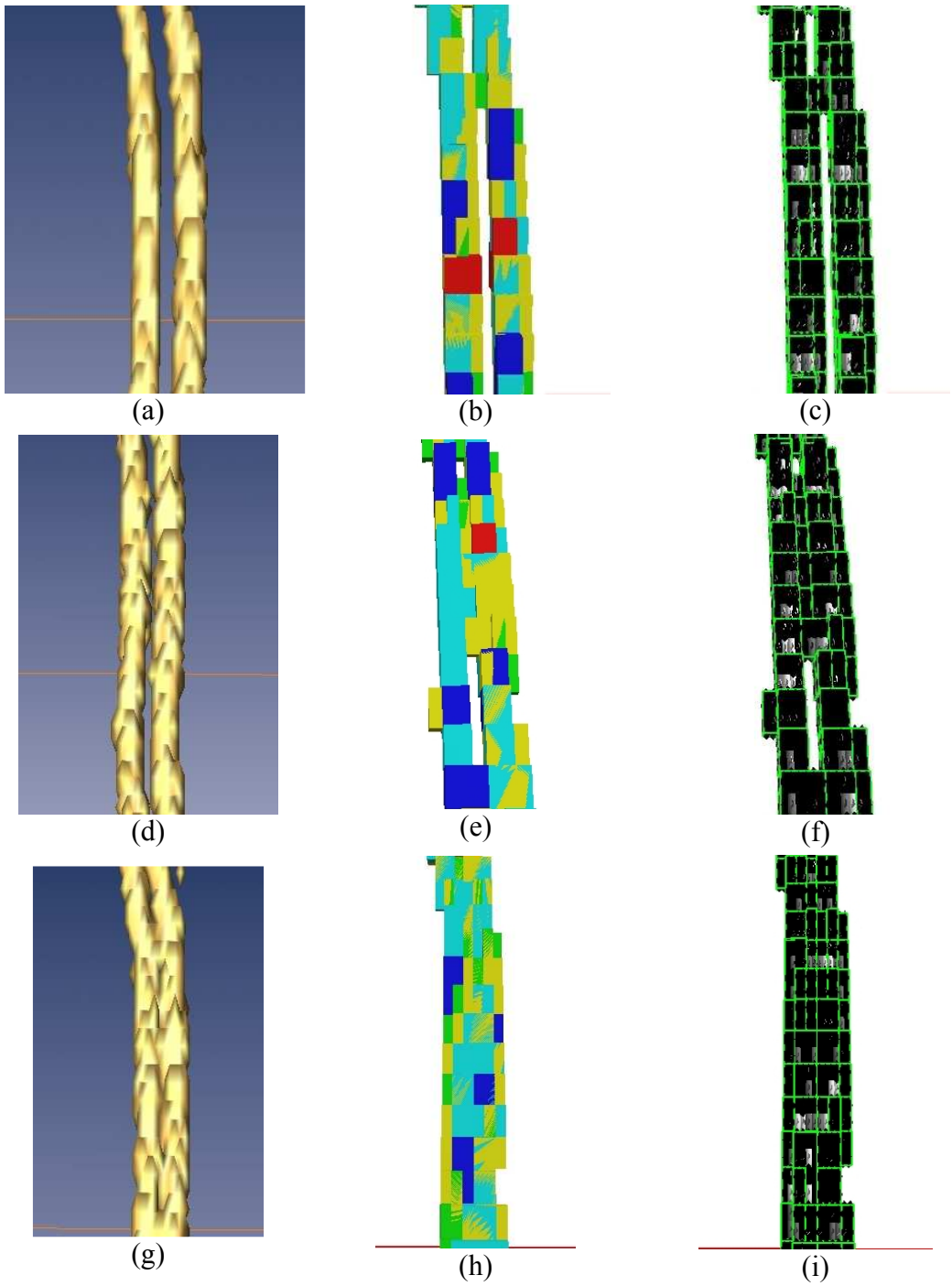


Figure 18. The effect of a gradual decrease of spacing between fibers: (a), (d), and (g) are Amira reconstruction images; (b), (e), and (h) are different colored L-blocks representing different sizes; (c), (f) and (i) are the corresponding L-blocks with contents.

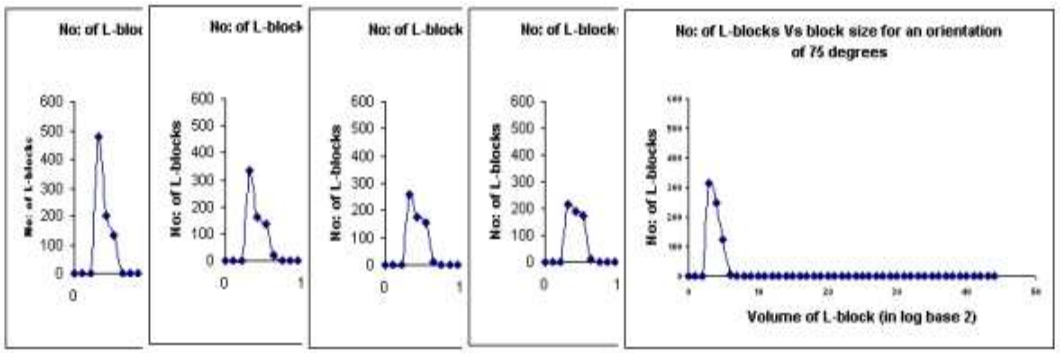


Figure 19. Plots of number of L-blocks vs volume of L-block (in log base 2) for parallel fibers placed at 5°, 30°, 45°, 60°, and 75° orientations respectively.

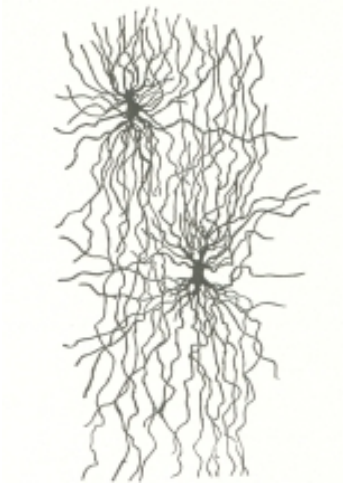


Figure 20(a)

Figure 20(b)

Figure 20. (a) An image of glial cells in white matter of the adult human brain; and (b) an image of collaterals of fibers in the commissural fascicle of the neonatal rat. (Source: Ref. [29]; reproduced with permission of the publisher)

iv. Neuropil-like Mats of Fine Fibers

Figure 22 shows a sketch with dense neuropil mat structures, from Ref. [30]. This sketch was drawn from Golgi-stained tissue and shows the complex nature of tissue data sets. These fibers are fine, dense, and branch heavily in a given volume,

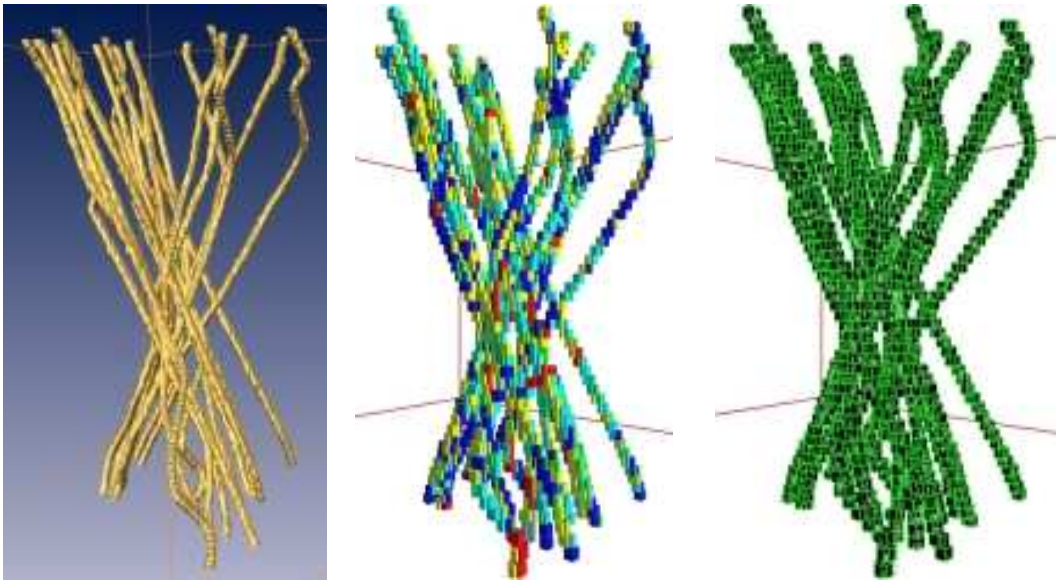
representing neuropil mats. Synthetic data, seen in Figure 23, was generated to resemble this Golgi data, at point B in Figure 22. This data set aids in testing how effectively the algorithm adapts to random nature of fibers going in all directions. Figure 23 shows the reconstruction of this synthetic data set using both Amira software and the polymerization algorithm.

2. Blob Data

Blob data resembles scanned Nissl-stained mouse brain tissue data. Nissl stains only cell bodies present in the tissue and does not stain neuronal arbors. These cell bodies can take many shapes. For purpose of this thesis, the polymerization algorithm was tested on synthetic data containing spheres, ellipsoids, and cylinders. In interpreting the results below, we note that the polymerization algorithm is primarily a data representation and not a data visualization tool.

i. Spheres

Figure 24 shows synthetic data consisting of spheres, along with reconstructions by Amira software and the polymerized algorithm. The results show that reconstruction of these data sets using the polymerized algorithm is not perfect. This can be attributed to the surface curvature of the object and the nature of the input data set. Since an L-block is a three-dimensional rectangular structure, it cannot cover curved surfaces accurately. From Figure 24(a), it can be seen that the input data is not perfectly spherical, resulting in errors in reconstructed data. Figure 24(b) shows that polymerization algorithm assigns one big L-block to the data in the center of the sphere (since the center can be reasonably approximated by a cube), and adds curvature to the cube by using smaller L-blocks.



(a) (b) (c)

Figure 21 (a) Amira reconstruction of branching fibers; (b) different colored L-blocks representing different sizes; (c) contents of reconstructed L-blocks along with the L-block wireframe.

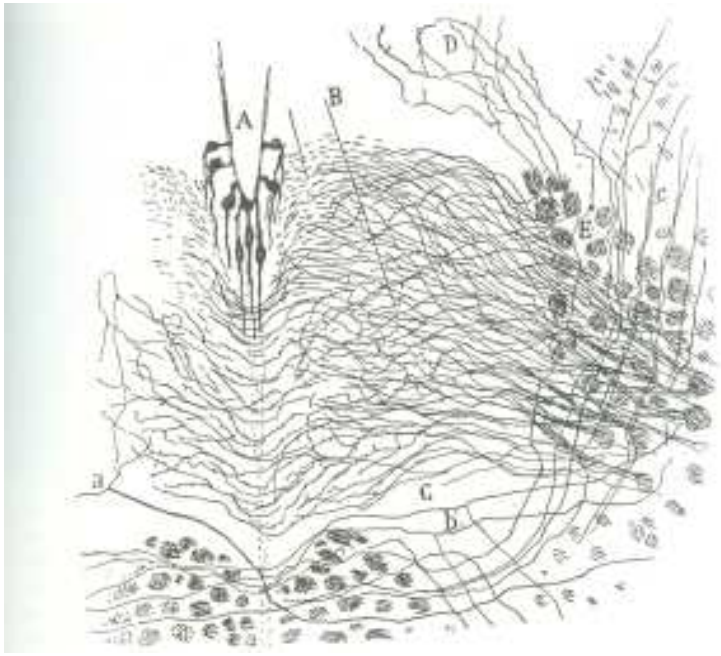
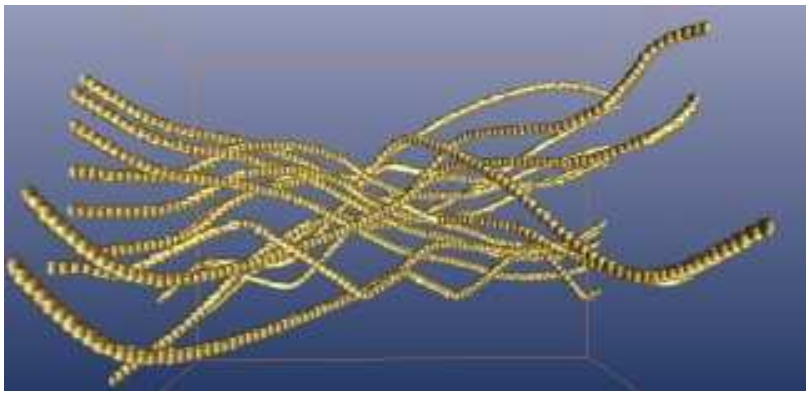
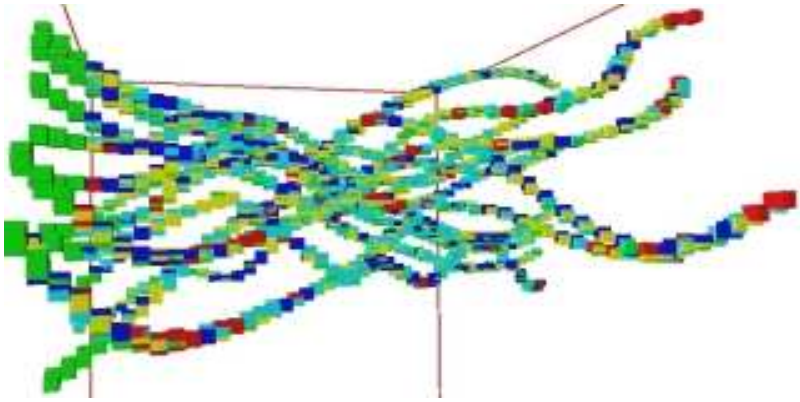


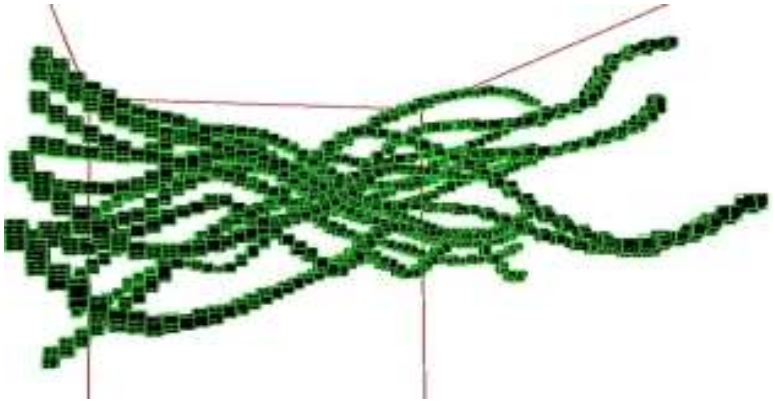
Figure 22 The hypoglossal nucleus in a near-term rabbit fetus, generated by the Golgi method. (Source: Ref. [30]; reproduced with permission of the publisher)



(a)



(b)



(c)

Figure 23. (a) Amira reconstruction of neuropil-like mats of fine fibers; (b) different colored L-blocks representing different sizes; (c) contents of reconstructed L-blocks along with the L-block wireframe.

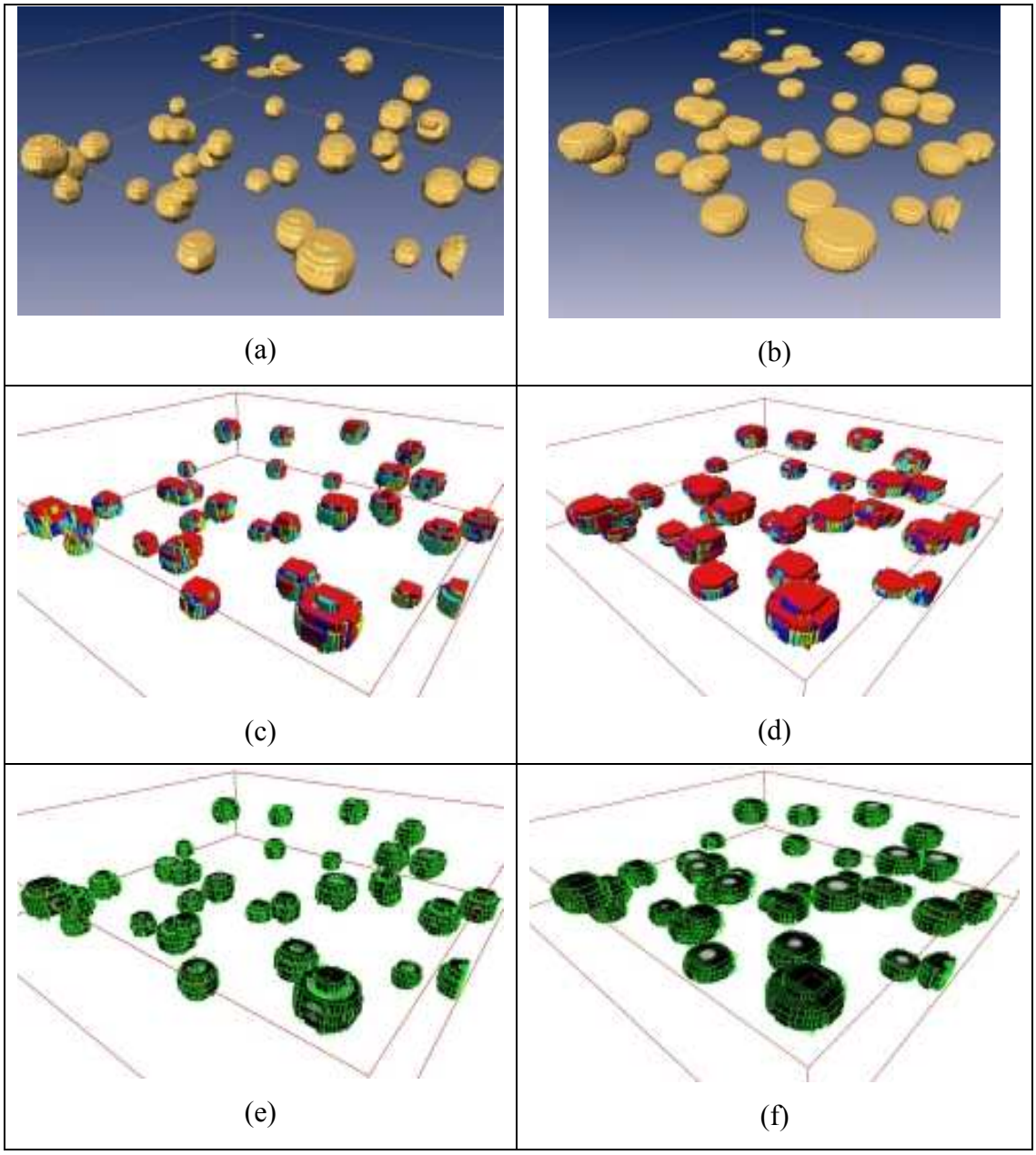


Figure 24. (a) Amira reconstruction of the data set with spherical objects; (c) L-blocks, colored according to their sizes; and (e) contents of the L-blocks along with their wire frame. Similarly (b), (d), and (f) show reconstructions for the same graphical primitives viewed in diffraction-limited optics.

ii. Ellipsoids

Figure 25 shows the reconstruction of synthetic data consisting of ellipsoids using both Amira software and the polymerization algorithm. The results show that reconstruction of these data sets using the polymerized algorithm is almost right, except for the bottom surface of the ellipsoids. This can be attributed to size of data present at the tip of the ellipsoids being less than that could be covered by an L-block. Figure 25(b) shows that the polymerization algorithm assigns one big L-block to the data in the center of the ellipsoid (since the center can be reasonably approximated for a three-dimensional rectangular block), and adds curvature of the ellipsoid by using smaller L-blocks.

The scale invariance of the polymerization algorithm with respect to size has been tested by running the algorithm on synthetic data sets with different sizes of ellipsoids. Figure 26 shows the plots (Number of L-blocks Vs Volume of L-block (in log base 2), obtained, for ellipsoids of three different sizes. All the three plots follow a similar pattern, showing that polymerization algorithm scales the L-blocks appropriately.

iii. Cylinders

Figure 27 shows synthetic data consisting of cylinders and the corresponding reconstructions using both Amira software and the polymerization algorithm. The results show that reconstruction of these data sets using the polymerized algorithm gives better results, though not accurate. This can be attributed to the curvature of the cylinders in the data sets. In these results, it can be seen that polymerization algorithm assigns one big L-block to the data in the center of the cylinder (since the center can be reasonably approximated for a three-dimensional rectangular block), and adds the curvature of the cylinder to the sides of this big L-block by using smaller L-blocks.

Orientations invariance of the polymerization algorithm has been tested by running the algorithm on synthetic data sets with cylindrical structures placed at different orientations. Figure 28 shows the plots (Number of L-blocks Vs Volume of L-block (in log base 2), obtained, for cylinders placed at 15°, 30°, 45°, 60°, and 75°

orientations. All the plots follow a similar pattern, showing that polymerization algorithm works well for structures oriented in different orientations.

The scale invariance of the polymerization algorithm to size has been tested by running the algorithm on synthetic data sets with different sizes of cylinders. Figure 29 shows the plots (Number of L-blocks Vs Volume of L-block (in log base 2), obtained, for cylinders of four different sizes. All the four plots follow a similar pattern, showing that polymerization algorithm scales the L-blocks according to the data sets.

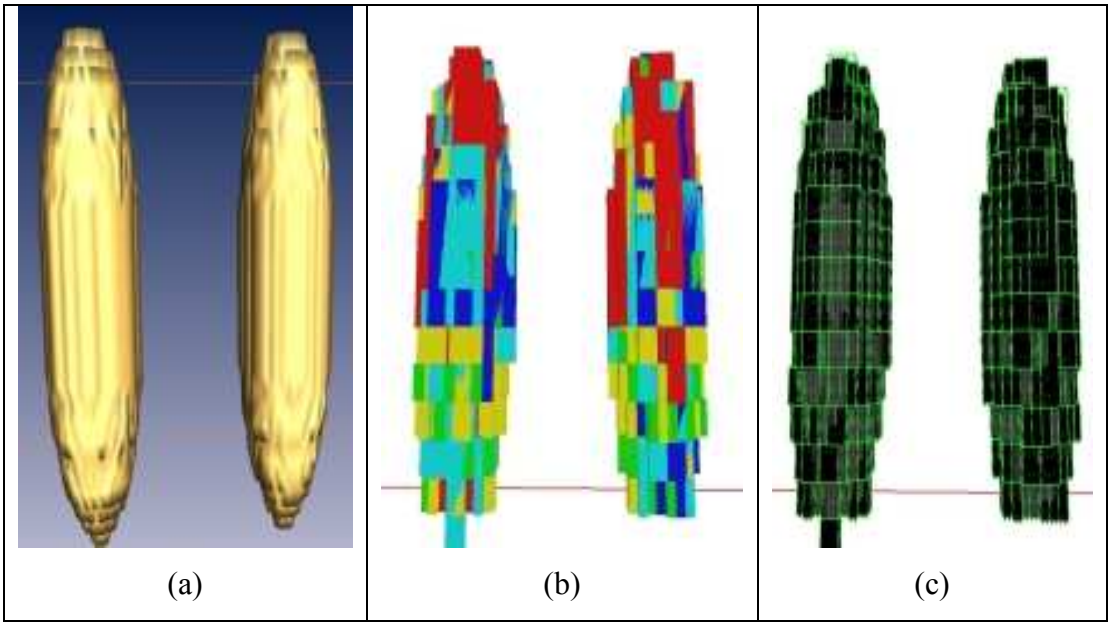


Figure 25. (a) Amira reconstruction of the data set with ellipsoids; (b) L-blocks , colored according to their sizes; (c) the L-blocks along with their contents.

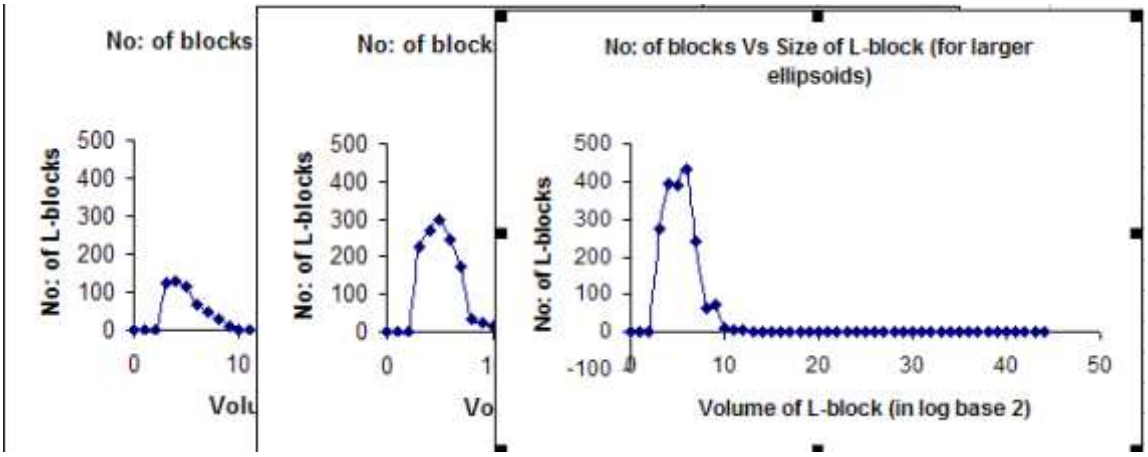


Figure 26. Plots of number of L-blocks vs volume of L-block (in log base 2) for three different data sets consisting of ellipsoids with maximum cross-sectional diameter of 12, 22, and 42 pixels respectively.

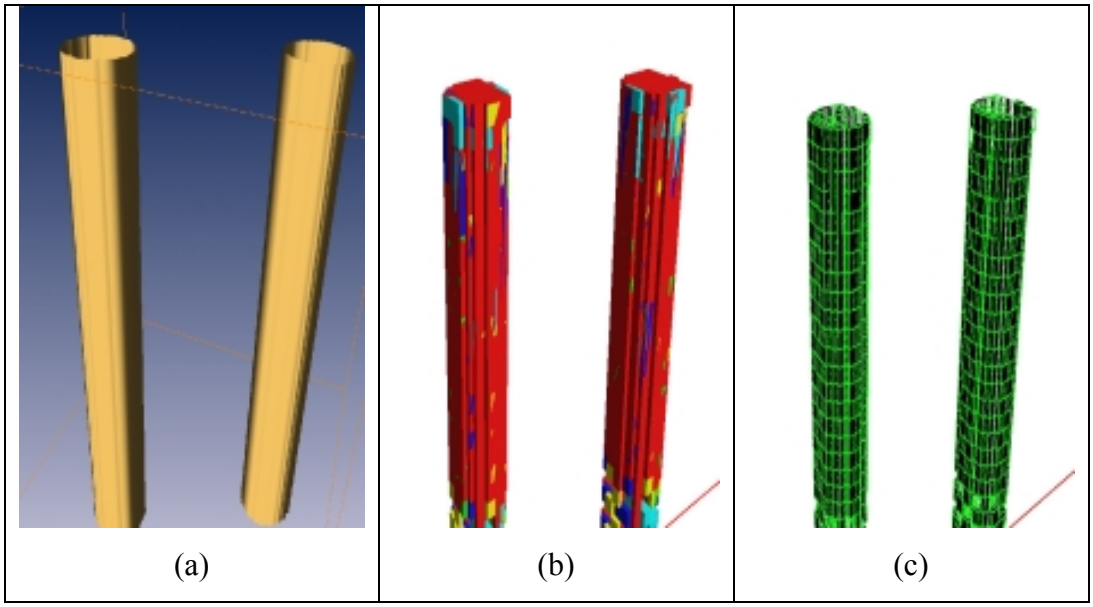


Figure 27. (a) Amira reconstruction of the data set with cylindrical structures; (b) L-blocks, colored according to their sizes; (c) L-blocks along with their contents.

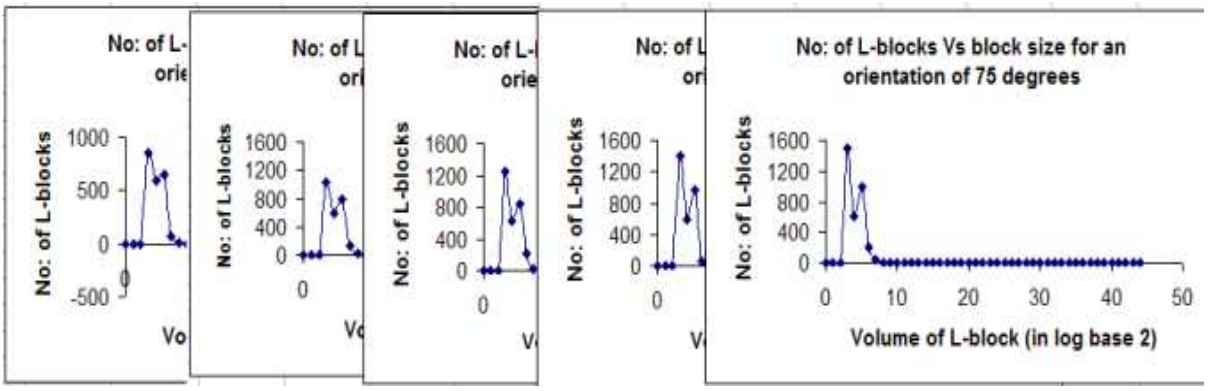


Figure 28. Plots of number of L-blocks vs volume of L-block (in log base 2) for three ellipsoids with maximum cross-sectional diameter of 12, 22, and 42 pixels respectively.

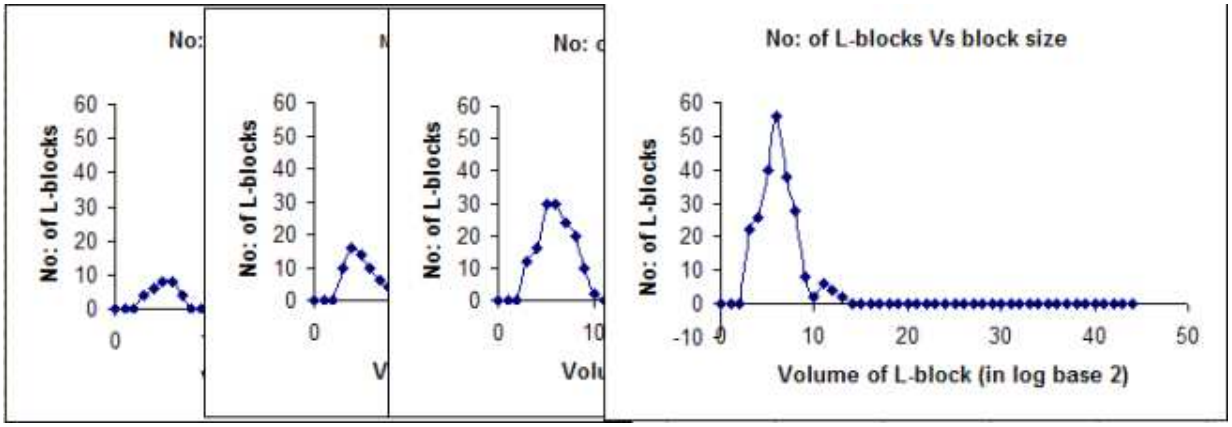


Figure 29. Plots of number of L-blocks vs volume of L-block (in log base 2) for four different data sets consisting of cylindrical structures with cross-sectional diameter of 8, 14, 22, and 45 pixels respectively.

CHAPTER V

L-BLOCK COVERINGS APPLIED TO THREE-DIMENSIONAL MICROSCOPY DATA OF MOUSE BRAIN MICROSTRUCTURE

1. Golgi-Stained Tissue

The Golgi staining technique is characterized by fixation in an aldehyde-osmium-dichromate solution, followed by impregnation with silver salts. Its uniqueness lies in the fact that only one percent of the neurons are stained. Unlike the Nissl stain, which stains only the cell bodies, Golgi stains the entire structure of the neuron. Since Golgi stains neurons only selectively, we can trace these stained neurons without interference from other unstained neurons in cell-dense portions of the brain. One disadvantage of the Golgi staining process is that it randomly stains neurons and therefore we cannot reproduce this stain pattern in another brain. Scanned data of Golgi-stained tissue creates one other problem for reconstruction. While scanned data contains the entire neuron, both the cell bodies and the arbors, the amount of stain taken up is proportional to the diameter of the cell body or arbor. Hence the integrated intensity value of arbors is relatively less than that of cell bodies. This variation of intensity makes the choice of threshold difficult.

Figure 30 shows a sample image from a stack of images generated from Golgi-stained tissue data, and also shows the results of reconstruction using both the Amira software [28] and the polymerization algorithm [1], [2]. Reconstructions with and without noise reduction are also shown. From these images we can see the advantage of using the noise removal techniques, specially when reconstructing Golgi-stained tissue data.

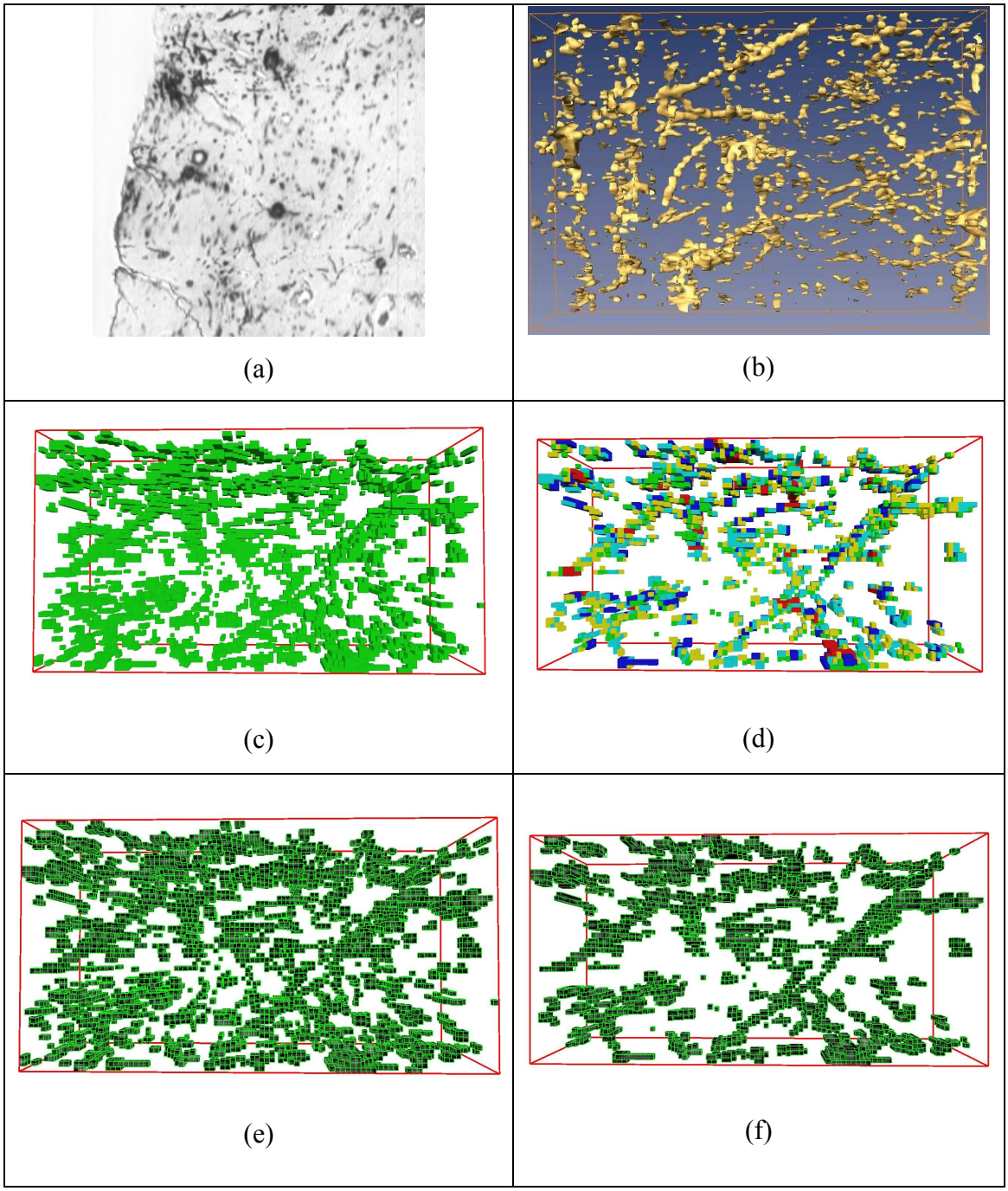


Figure 30. (a) 10X image of a slice of Golgi-stained tissue data (set1) ; (b) Amira reconstruction of the corresponding image stack; (c) and (e) L-blocks reconstruction without any noise reduction or compression; (d) and (f) L-blocks reconstruction with noise removal and compression.

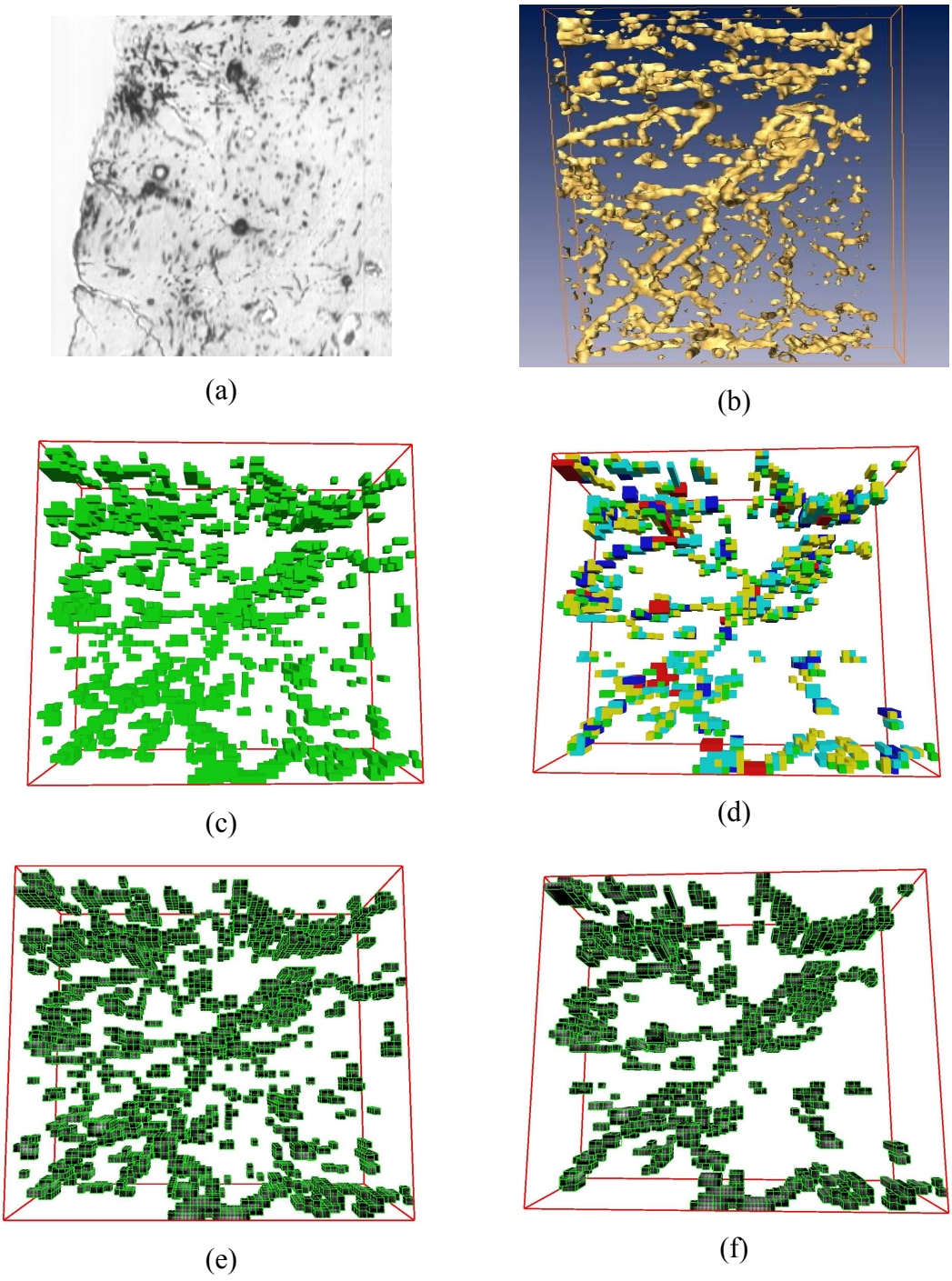


Figure 31. (a) 10X image of a slice of Golgi-stained tissue data (set2); (b) Amira reconstruction of the corresponding image stack; (c) and (e) L-blocks reconstruction without noise reduction or compression; (d) and (f) L-blocks reconstruction with noise removal and compression.

Figure 31 shows Golgi-stained tissue data, but a different part of the same data set was used for reconstruction. The cell bodies are less dense in this part of the data set. Though less dense, the noise reduction techniques show significant difference in the reconstruction. The polymerization algorithm tries to reconstruct a thread-like structure.

2. Nissl-Stained Tissue

Nissl stains the RNA in the cytoplasm of all neurons and also the DNA in the cell bodies. The advantage of this stain is that it stains all cell bodies, and hence we can view the distribution of the cell bodies in the entire tissue. Since only cell bodies are considered, we have little problem in setting a proper threshold, unlike in Golgi-stained tissue data. One disadvantage of Nissl data is that reconstruction takes more computational time than for Golgi-stained tissue data: The reason here is the sheer number of cell bodies to reconstruct and space occupied by each of the cell bodies.

Figure 32 shows a slice of the Nissl-stained tissue data, and the results of reconstruction using both Amira software and the polymerization algorithm. From the raw image, we can see that the scanning artifacts lead to lesser contrast in the images, making reconstruction of these images difficult. To improve the constrast as an aid in reconstruction, we have used contrast enhancement and image homogenization (results shown in Figure 33). This additional filtering diffrentiates the cell bodies from the background. From the reconstructed images we can see the density of the cell bodies in a given unit volume and also the effectiveness of the polymerization algorithm. Unlike Golgi, Nissl stains only the cell bodies, leading to less noise in the scanned data. Therefore the results show less improvement with noise reduction.

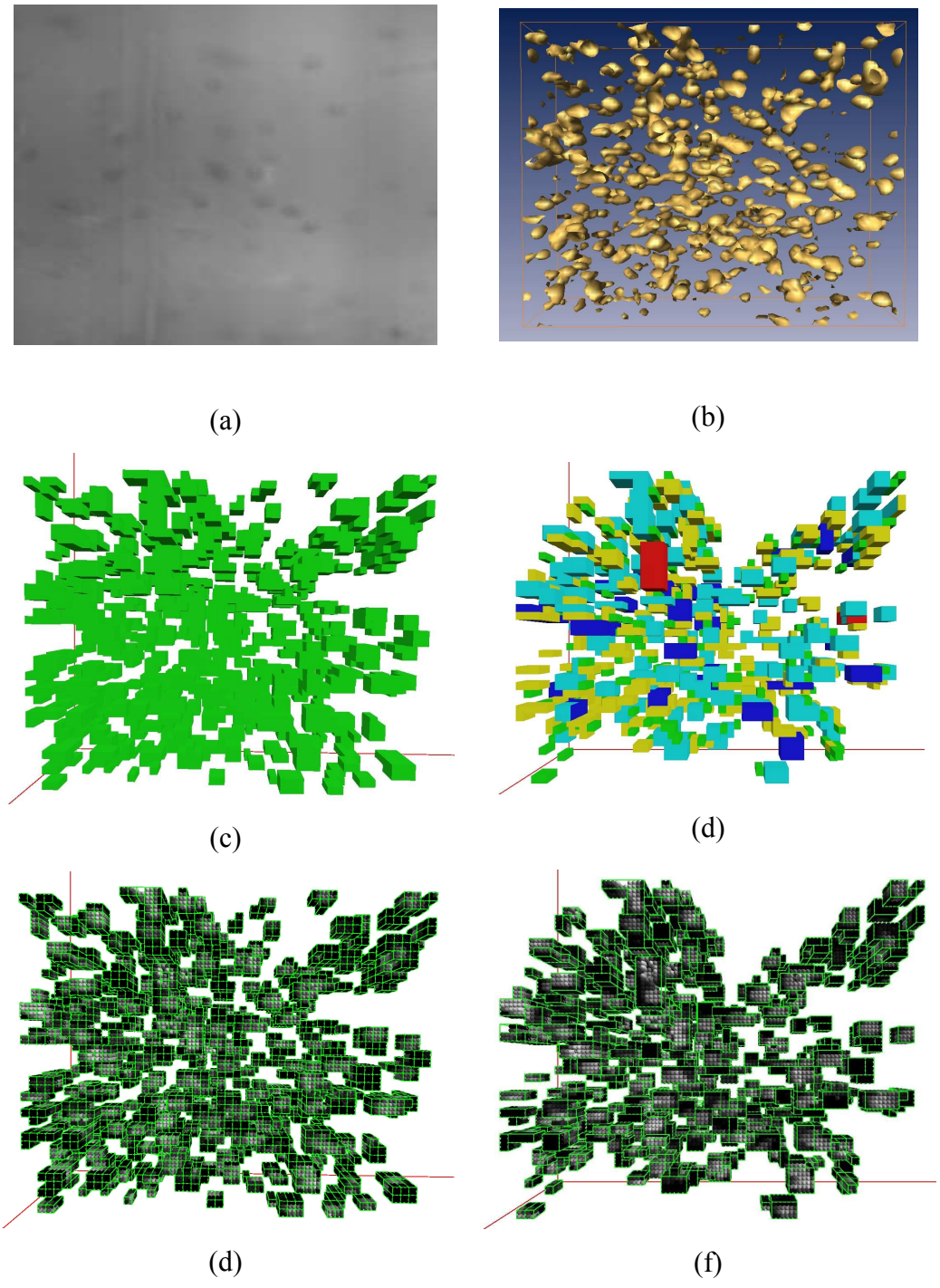


Figure 32. (a) 10X image of a slice of Nissl-stained tissue data (set1); (b) Amira reconstruction of the corresponding image stack; (c) and (e) L-blocks reconstruction without noise reduction or compression; (d) and (f) L-blocks reconstruction with noise removal and compression.

Figure 34 shows the same Nissl-tissue stained data set as earlier, but from a cell-body dense part of the tissue. This image shows densely packed nature of the cell bodies. This image also has the same problems of contrast as earlier and the result of applying contrast enhancement and image homogenization on the raw image can be seen in Figure 35. This image set was selected to test the performance of the polymerization algorithm when the image stack is dense and has less contrast.

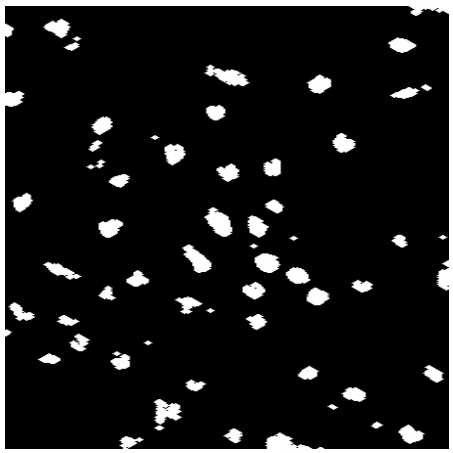


Figure 33. Resulting image after applying contrast enhancement and image homogenization on less cell - dense image.

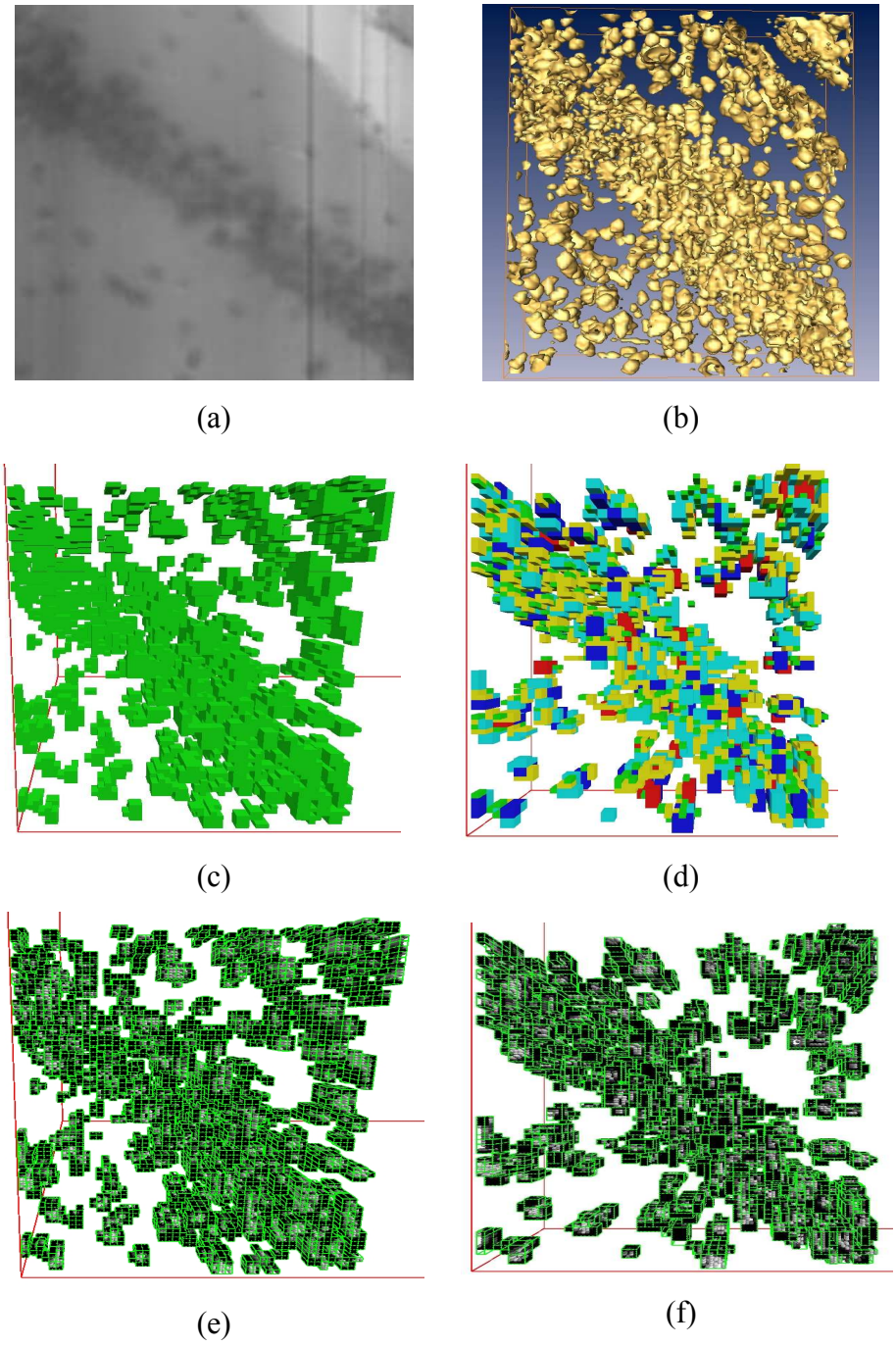


Figure 34. (a) 10X image of a slice of Nissl-stained tissue data (set2); (b) Amira reconstruction of the corresponding image stack; (c) and (e) L-blocks reconstruction without noise reduction or compression; (d) and (f) L-blocks reconstruction with noise removal and compression.

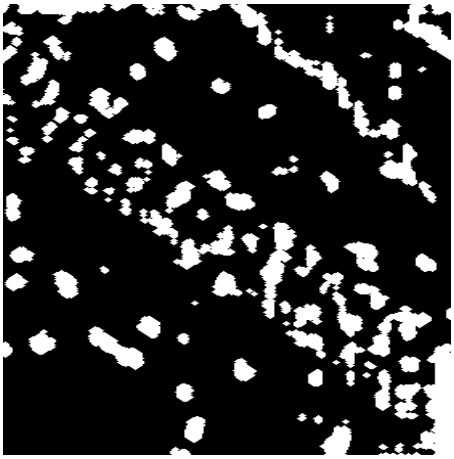


Figure 35. Resulting image after applying contrast enhancement and image homogenization on cell-dense image.

CHAPTER VI

RATIONALE FOR VECTOR-BASED TRACING AND ITS USAGE WITH THE POLYMERIZATION ALGORITHM

1. Pathological Deficiencies Prevalent in Large-Scale Filamentary Volume Data Sets

Neuronal volumetric data sets considered in this work have many distinguishing features that makes reconstruction difficult. Pathological deficiencies prevalent in large-scale filamentary volume data sets lead to errors in the reconstruction caused by:

a. Loss of connectivity information due to missing data (gaps), and
b. Confusion in identifying individual fibers in fiber-dense regions.

Some of these pathologies are:

- *Stain drop-out*. Caused by non-uniform stain uptake, this pathology leads to loss of tracking information in weakly stained data.
- *Weakly articulated fine fibres* These can result from scanning/imaging artifacts, sectioning chatter, or irregularity in illumination.
- *Loss of contrast in fine fibres*. Loss of contrast can be the result of both inadequate staining and limited scanning/imaging spatial resolution. Staining artifacts are caused due to inadequate penetration of the stain into fine fibers. Scanning/imaging resolution is proportional to the dominant illumination wavelength and inversely proportional to the numerical aperture (NA) of the microscope objective. Axons in the mouse average 300nm diameter, comparable to the sampling interval of the scanning process.

- *Rampant branching and neuropil mats.* Regions of volume data with high fiber density cause this pathology. High fiber density regions can have :
    i. Fibers running parallel to each other at very close distances
    ii. Fibers crossing each other creating a mesh-like structure (neuropil mats), and
    iii. Fibers crossing in all possible directions.

In all these cases distinguishing each individual fiber and separating the fiber from others in its proximity is a difficult task.

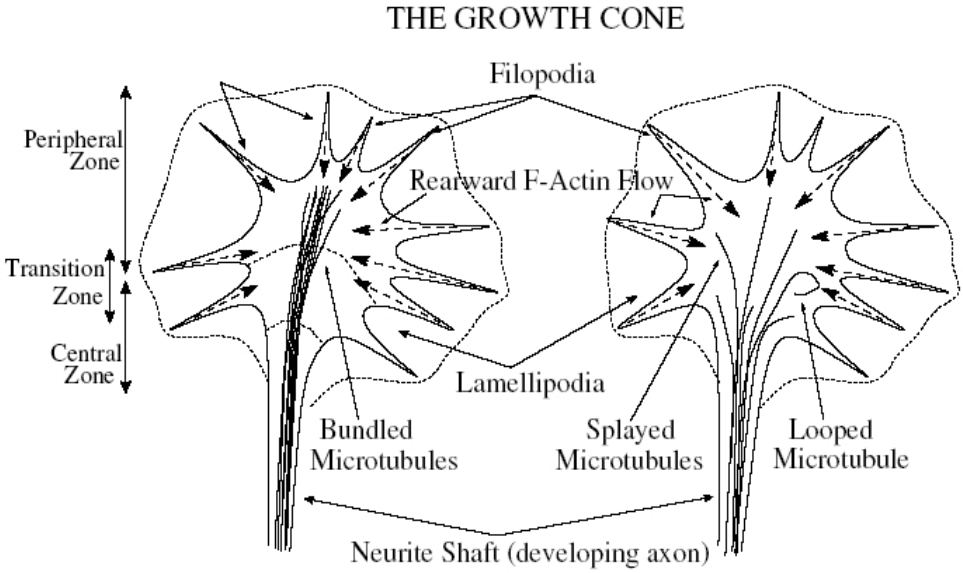


Figure 36. Growth cone with the filapodia, from Ref. [31]

2. The "Growth Cone"-Mediated Polymerization Process

The growth cone, first named by Santiago Ramón y Cajal, the famous Spanish neuroanatomist [29][30], is the structure present at the end of a neurite which guides the developing neurite to its target cells. Each neurite, from its initiation at the soma/cell-body, mends its way towards its target cells, thus forming a network which mediates the exchange of information/signals. Each growth cone has long, thin spike-like fingers at its tip, called *filopodia*. Filapodia act like antenna, continuously exploring the extracellular

environment surrounding the growth cone. Figure 36 shows an image of growth cone with filapodia, taken from Hely[31]. Filapodia help in steering the growth cone in a particular direction, either by extending or retracting the antenna, depending on the environment. The filapodia interact with other cells, causing the flow of ions and other molecules between the growth cone and other cells.

Adapting this concept to the reconstruction of neuronal data, a growth-cone mediated polymerization process can be used to overcome pathological deficiencies in the segmentation defined by a L-block covering of the filamentary volume data set. This growth-cone mediated strategy uses the edited filamentary volume data on a sample volume data set, learns its growth strategy from this "training set", and generalizes the growth process to apply to similar homogeneous filamentary volume data sets. Such methods typically estimate the suitability of taking a step to a nearby position. After a few steps if the algorithm finds that the path taken is not the right direction, it can back off to an earlier positions and restart again from the previous position, much in the manner of neuronal growth cones.

3. Rationale for Vector-Based Tracing

Prior to vectorization, most existing methods for neuronal reconstruction were either skeletonization methods or methods based on edge/line enhancement. In both these methods, each and every pixel/voxel present in the data set is processed, with numerous operations on each. In such cases even simple operation per pixel (voxel), can lead to large computational cost. The computational costs also increases linearly with the increase in data size. When dealing with filamentary structures, the amount of valid data is often a small percentage of the whole data set. In such cases it is not advisable to process all voxels present. Compared to previous skeletonization methods, vector-based tracing [4], [5] offers the following advantages:

- Does not require preprocessing or expensive deconvolution
- Works in a recursive manner
- Processes only a minimal necessary fraction of voxels in a exploratory manner

- Highly adaptive, relying only on local image information.
- Can be automated completely
- Avoids any kind of prior operations on the images, like thresholding, edge detection, etc.
- Scales well with the image size
- Can be used for real-time processing
- Gives very high speed-ups in processing

4. Review of the Neuron Tracing Literature

Vectorization algorithms or exploratory algorithms can be broadly classified into three types described below.

The first technique is commonly used in Quantitative Coronary Angiography (QCA) [32]. This is a semi-automatic technique, in which the user manually enters the initial and final endpoints of the vessel (and sometimes also its direction and width). These techniques provide accurate results; however they are not preferred for large volumetric data sets. They are also not suitable if the vessels branch or have intersections among themselves.

In the second technique the user manually enters the initial point and direction[33]. The system then recursively traces the entire neurite tree using a depth-first search, without manual intervention. The drawback of this approach is that it can only trace a single axonal or dendritic arbor that is efferent from a single soma and is spatially isolated from neighboring neurons. All neurites need not be continuous, given the pathological deficiencies described above. The strictly recursive approach also breaks down when it encounters missing data during its traversal.

In the third technique, the algorithm calculates the starting points (seed points) [4], [34]-[37] by itself and extracts the neuronal structure without user input. These techniques work well for coronary angiograms and also for three-dimensional reconstruction. These techniques are not necessarily efficient in tracking neurons in congested regions with many neighboring neurons.

5. Overview of the Vector-Based Tracing Extension

The vectorization method, as described in the paper by Al-Kohafi et al., [4], [5], tries to exploit local image properties to trace the neuronal structures recursively. In this method, dendritic and axonal segments are approximated by generalized cylinders (over a short distance), with elliptical cross sections and some gradual curvature along the trajectory of the cylinder. A 5xK kernel is termed as a "template", where K is the number of voxels considered in the direction of the neurite trajectory. For each of the K voxels we have a 5x1 matched filter $\mathbf{[-1, -2, 0, 2, 1]^T}$, applied to it [37], and a moving average is computed over the K voxels. For the 3D case, the rotation space for both θ and ϕ (explained in Chapter VIII), is discretized/quantized into either 16 or 32 orientations for each degree-of-freedom. For each [θ , ϕ] orientation four templates (one each for appliation to the right, top, left, and bottom boundaries of the elliptical cross section of the structure), are precomputed. For a 2D case, the rotation space in the X-Y plane is discretized/quantized into either 16 or 32 orientations and for each orientation two templates (one each for appliation to the left and right boundaries of the structure), are precomputed. Starting from a seed point (required for initiation of vector-tracing algorithm, as explained in Chapter IX) on the center line and an initial estimate of the tangent to the trajectory of generalized cylinder, the structure is recursively traced by estimating successive points (using the respective equations described for 2D and 3D cases in Chapter VIII) along the neurite trajectory at each step. Figure 47 and Figure 48 illustrate the 2D and 3D recursive tracing algorithm respectively. This process is stopped when it satisfies a stopping criterion (explained in Chapter IX). Criterion have to be defined for selection of seed points, detection of branch points, and also for stopping the recursion. In this method a variable- length template is used, which adapts itself to the gaps and noise present in the data, and the local curvature of the neurite.

Combining the polymerization algorithm [1], [2] with the vectorization method, as explained above, can be done in either of two ways, either polymerization followed by vectorization or vectorization followed by polymerization.

In the first process, polymerization followed by vectorization, we apply the polymerization algorithm on the given data set and generate L-block coverings/clusters, as explained in earlier chapters. The end points of the clusters generated serve as seed points for vector tracing. For initial orientations, we estimate the orientation (say R1) of the fibers at these end points using matched filters. Then we generate orientation (say R2), which is the exact opposite orientaion of R1. With the seed points so chosen and an intial set of orientations, we start the vector tracing recursion in both the directions R1 and R2. We stop the recursion if these recursions satisfy the defined stopping criterion.

In the second process, vectorization followed by polymerization, we first perform the vector tracing, obtain the reconstructed data and use it and neighboring data as input volumetric data for the polymerization algorithm. This allows us to add local structure (such as spines) to the generalized cylinder model of neuronal segments, used by vector tracing.

In the Al-Kofahi implementation of the vector tracing algorithm, seed points are generated by projecting the image stack into a single 2D image plane, processing this projected image, and then reprojecting seed points back into 3D space. This construction is computationally experiment when the image stack containing multiple neurons. The data sets used in our work typically consists of thousands of neurons, while the data set used in [4], [5] consists of a single isolated neuron. Distinguishing multiple neurons in the projected 2D image quickly becomes computationally expensive. Since the technique used to generate seed points is neither computationally efficient nor gives accurate results, this second process is not necessarily feasible for the volumetric data set used in our work.

CHAPTER VII

MATCHED FILTERS FOR EDGE DETECTION

1. Membrane or Neurite-Filling Stains

Types of stains commonly used with brain tissue and their properties are listed in the Table 1 below. Each of these stains is specifically targetted to label certain parts or the whole of the tissue. For example, Golgi-Cox stains both the cell body and its neurites (axons and dentrites), while Nissl stains mostly cell bodies. Since Nissl stains acidic structures (DNA, RNA and cytoplasm), it stains the nucleus (due to DNA) and cytoplasm (in a punctate manner depending on the amount of rough endoplasmic reticulum and polyribosomes present). Green Flourescence Protein is a protein that flouresces, so it can stain either the membrane or neuron depending on the location of the protein attached to it.

Table 1. Typical stains used with brain tissue.

Name of the Stain	**Type of the Stain Filling (Membrane/Neuron)**
Golgi-Cox	Neuron
Nissl (Cresyl violet)	Neuron (not entirely)
Green Fluorescence Protein / XFP (other Fluorescence Protein)	Membrane/Neuron (depending on the location of the protein)
Osmium Tetroxide	All membranes

The optimum choice of matched filters used for edge detection varies with the portions of the cell that are stained, e.g., membrane-only or cell-filling. Confocal

microscopy and multi-photon microscopy, optically section tissue. In these techniques a membrane stain Vs. a filling stain might be indistinguishable. In knife-edge scanning microscopy, the tissue is physically sectioned, so the type of filling affects the scanned output data. For example, a membrane stain (osmium tetroxide) is easily distinguished from Golgi-stained neuron.

2. Image Formation: The Point-Spread Function

When imaging under diffraction-limited optics, certain blur is introduced into the scanned images due to diffraction of light during the scanning process. So during the generation of synthetic images, we differentiate between imaging in *geometric optics* Vs imaging in *diffraction-limited optics*. Imaging in geometric optics gives us the images of graphical primitives, if appropriately large, e.g., as seen by the naked eye. After generating the hard-edged two-dimensional images in the synthetic data set, we simulate the blurring effect in diffraction-limited optics by convolving a point-spread function once each two-dimensional image in the synthetic data set. This strategy is appropriate for knife-edge scanning microscopy; Confocal microscopy and multi-photon microscopy however would require using a three-dimensional point-spread function. The reason is, in KESM, the tissue present above/below the present layer, does not affect the section of tissue being scanned. So only that particular section requires 2D-convolution blurring, generated by using a simple point spread function. The following function was used to generate point-spread function:

$$y = (2 * J_1(x) / x)^2$$

Here x is the radial distance from the center of the kernel, y is the magnitude that has to be multiplied with the gray-scale value at x, and $J_1(x)$ is the bessel function of the first kind of order 1 of x.

3. Geometric Primitives Viewed at Different Orientations

Synthetic data was generated consisting of geometric primitives placed at different orientations. This aids us in computing matched filters appropriate for a given orientation, which are merely snapshots centered across the object boundary. The generated matched filters were tested for translational invariance. These tests suggest better matched filters to be used on real scanned tissue data. For each of the generated image, we consider a pixel on the boundary of the geometric primitive in the image and read the gray-scale values of a 5X6 template around this pixel. We then substract the mean of the pixel values in the 5X6 template from each of the template's elements. The values thus obtained may not be integral values and are then quantized in the interest of minimizing computational complexity. Later we subtract a constant from each gray scale value, such that the sum of these template values (weights) is either zero or close to zero. We generate such matched filters for a cylinder with a couple of orientations. Later we perform the same process for synthetic data as would be seen with diffraction-limited optics and also test the translational invariance of these filters by calculating them at different positions along the same boundary of the object.

For an orientation of 30 degrees, the template for one of the border points and the reduced filter are shown in Figure 37 and Figure 38 respectively. Figure 39 is the template taken from the same image, after convolution with the point spread function. Figure 40 is the resultant template after mean substraction and adjusting the values (such that they are close to zero). The orientation of the template considered is axis-aligned, which explains the concentration of zeros at the center. Figure 41 shows a template taken along the boundary of a cylinder(oriented 30 degrees to the x-axis) in the same image and Figure 42 shows the resultant template obtained.

210	210	210	210	210	210
210	210	210	210	210	210
210	210	210	210	30	30
210	210	30	30	30	30
210	30	30	30	30	30

Figure 37. 5X6 template of gray-scale values taken from a cylinder oriented at 30 degrees.

1	1	1	1	1	1
1	1	1	1	1	0
1	1	1	1	-1	-2
1	1	0	-2	-2	-2
0	-2	-2	-2	-2	-2

Figure 38. Result of substracting the mean and adjusting the template values.

13	27	47	66	81	90
42	63	79	89	93	96
42	62	78	88	93	96
42	62	78	88	94	96
75	87	93	96	97	98

Figure 39. 5X6 axis-aligned template of gray scale values taken from a cylinder oriented at 30 degrees, after simulation of diffraction-limited optics.

-2	-2	-1	0	0
-1	0	0	0	1
-1	0	0	0	1
-1	0	0	0	1
0	0	1	1	1

Figure 40. Result of substracting the mean and adjusting the template values, for the cylinder oriented at 30 degrees, after simulation of diffraction-limited optics.

1	2	5	11	6	13
4	2	5	11	24	42
9	20	37	24	42	62
20	38	58	75	62	78
53	38	59	75	87	93

Figure 41. 5X6 template of gray scale values taken along the boundary of a 30 degree oriented cylinder (under diffraction-limited optics).

-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	0	0
-1	0	0	0	0	1
0	0	1	1	1	1
1	0	1	1	2	2

Figure 42. Result of substracting the mean and adjusting the template values for the 30 degree orientated cylinder (under diffraction-limited optics).

4. Comparison of Discrete Matched Filters with RPI-Type Matched Filters

The performance of our discrete matched filters is compared with the matched filters designed by Al-Kofahi et al., (RPI-type matched filters) [4], [10], [37]. This comparision is done by executing the extended polymerization algorithm [1], [2] on a synthetic data set. Figure 43 shows the result of extended polymerization algorithm on sythetic data set (torus), once using matched filter used in the paper by Al-kofahi et al., (Figure 43(a)) and using discrete matched filters generated in the previous sections for the second image (Figure 43(b)). As seen from the figure, the performance of the algorithm using the discrete matched filters is comparable to RPI-type matched filters. Except for the circled portion in Figure 43(b), where we can see a break, the algorithm traces the underlying structure nicely. Only the traced seed points are shown and not the elliptical structure that encloses them.

Testing the performance of discrete matched filter performs in comparision to RPI-type mached filter for different orientations of the object is done by running the extended polymerization algorithm on synthetic data sets with objects (torus) placed at different orientations. Figure 44 plots Number of Seed points traced Vs Orientation of the objects in the synthetic data sets. Both the plots follow a similar pattern.

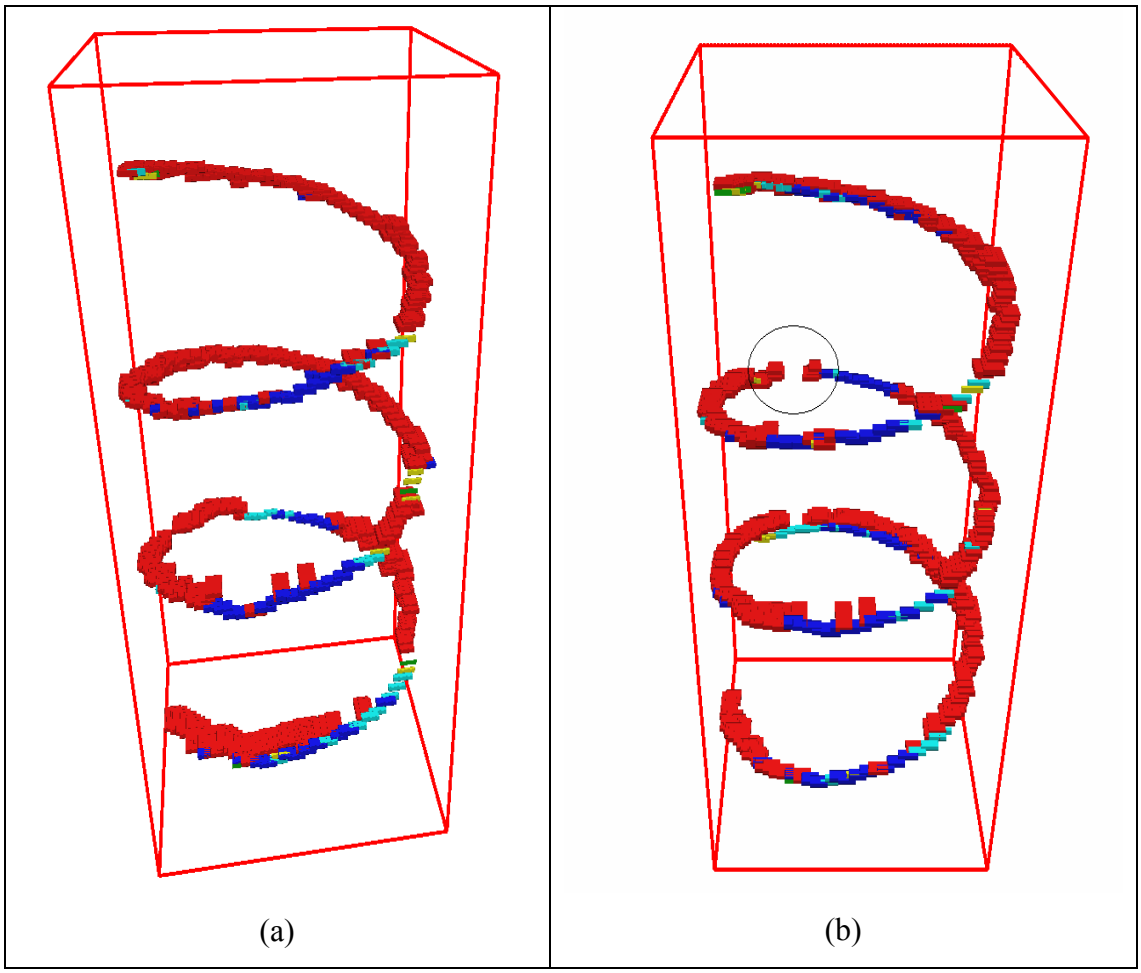


Figure 43. (a) Result of running extended polymerization algorithm, using RPI-type matched filter on synthetic data set. (b) Result of running extended polymerization algorithm, using discrete matched filter on synthetic data set.

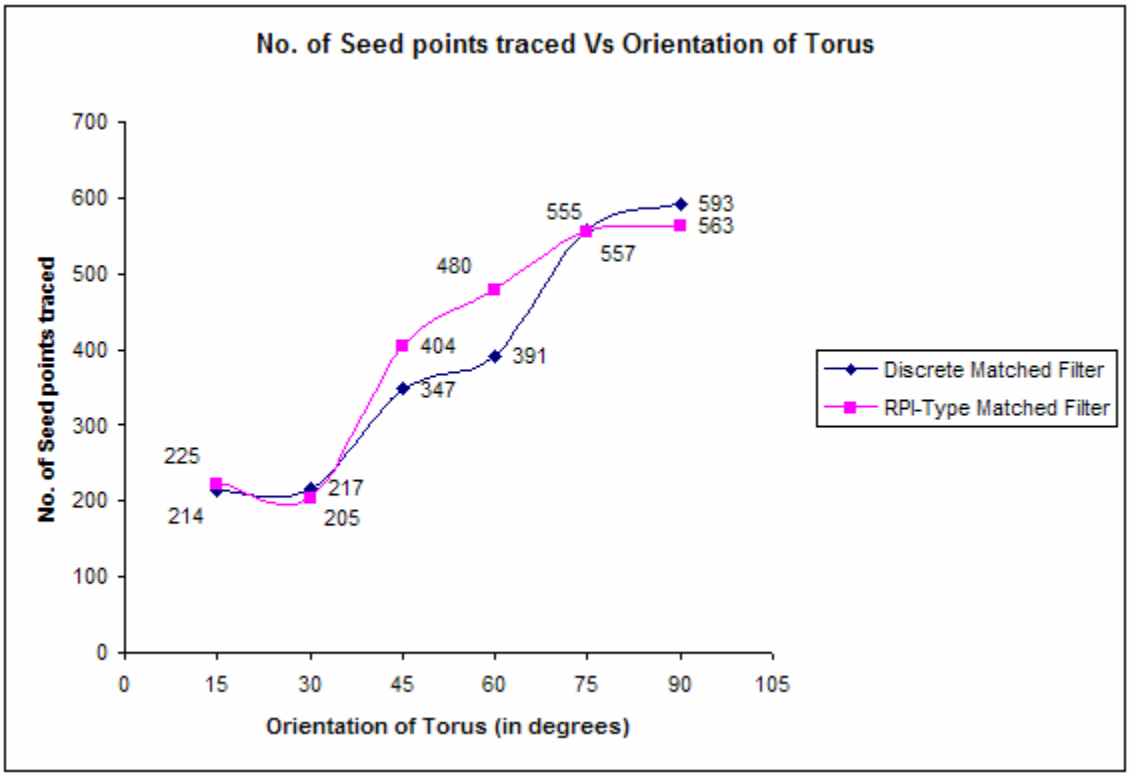


Figure 44. Plot of number of seed points traced vs orientation of synthetic data (Torus), comparing the performance of using the polymerization algorithm with discrete matched filter and RPI-type matched filter.

CHAPTER VIII

ITERATIVE/RECURSIVE VECTOR TRACING BASED ON PREDICTOR-CORRECTOR METHOD

Theory explained in this chapter is taken from the paper by Al-Kofahi et al. [4], [5]; The text is mainly an explanation of the author's algorithm. For a more detailed description, the reader is refered to the paper by Al-Kofahi et al. Rationale for using vector-based tracing and its usage in combination with the polymerization algorithm [1], [2] in the final reconstruction stage has been explained in Chapter VI.

Before describing the algorithm, we discuss the pre-requisites: the input parameters needed to start and perform the computations of the algorithm. Later in the chapter the notation required to explain the formulae of algorithm is explained.

1. Prerequisites

The main parameters required to perform the iterations are pre-computed templates, the number of neighboring directions to be considered, seed points, and initial trajectory orientations at these seed points.

i. Pre-Computed Templates Separate templates for application at different boundaries of the neurite for different orientations are "pre-computed". These "pre-computed" templates speed up execution time as they reduce the number of floating point operations/computations. For the purpose of this algorithm, orientation space is discretized into N different orientations for each rotational degrees-of-freedom.

For 2D space with one rotational degree-of-freedom, the orientation space is discretized into 32 different orientations. For each orientation we compute two templates, one each for the right and left boundary of the structure. These "pre-computed" templates are stored in the memory and are accessed as required, thus avoiding floating point computation. Figure 45 shows a sample two-dimensional

template computed for 22.5º orientation. The same template would be used for any angle between 22.5º and 45º.

For 3D space, directions are described in terms of two angles θ and ϕ, corresponding to the two-rotational-degrees of freedom. Angle θ describes a rotation around Z axis, and ϕ describes a rotation around the rotated Y axis (after an angle of θ around the Z axis). Figure 46, illustrates the coordinate system for performing the rotations. Now each of the angle/orientation space is discretized into N = 32 different orientations, resulting in a total of 1024 (N^2 = 32 x 32) unique directions. For each of these directions we have four sets of templates, one each for right, left, top and bottom boundaries (Figure 48). So totally we have 4096 (4 N^2 = 4 x 1096) pre-computed templates.

ii. Neighboring Directions Considered To reduce the computation time, for each iteration we limit on the number of neighboring directions considered. This reduces the redundant computations that are performed when all the 1024 orientations, in case of 3D, and 32 orientations in case of 2D, are considered. The rationale behind this reduction in the number of orientations considered is the limited curvature of the neurite structures. So the number of directions to be considered is denoted by Σ, where

$$\Sigma = \{ \tilde{\boldsymbol{u}}^{i} + [\pm\partial s_1, \pm\partial s_2]^{T} \mid \partial s_1, \partial s_2 = 0, 1, \ldots, \Delta\Sigma. \},$$

Here $(2\Delta\Sigma + 1)$ is the maximum number of neighboring directions for each degree of freedom, Here $\tilde{\boldsymbol{u}}^{i}$, is the predicted direction of the i^{th} iteration.

iii. Seed Points Seed points are the initial points where we start iteration. A complete set of seed points present in the data set is needed for tracing the entire structure.

iv. Initial Orientations at Seed Points Given the seed points, the initial estimate of the orientations along which the structure to be traced is required. This initial orientations is important when we consider the limited number of neighboring directions normally considered. We might miss the structure entirely if the initial orientation is way

off the actual orientation or if that orientation is not among the limited neighboring orientations considered.

				-1	-1	
	-1	-1	-1	-2	-2	
-1	-2	-2	-2	0	0	
-2	0	0	0	2	2	2
	0	2	2	1	1	1
	2	1	1			
	1					

Figure 45. A 5X6 pre-computed 2D rotation template for 22.5 degrees orientation. The entries are kernel values to be multiplied with the corresponding gray values.

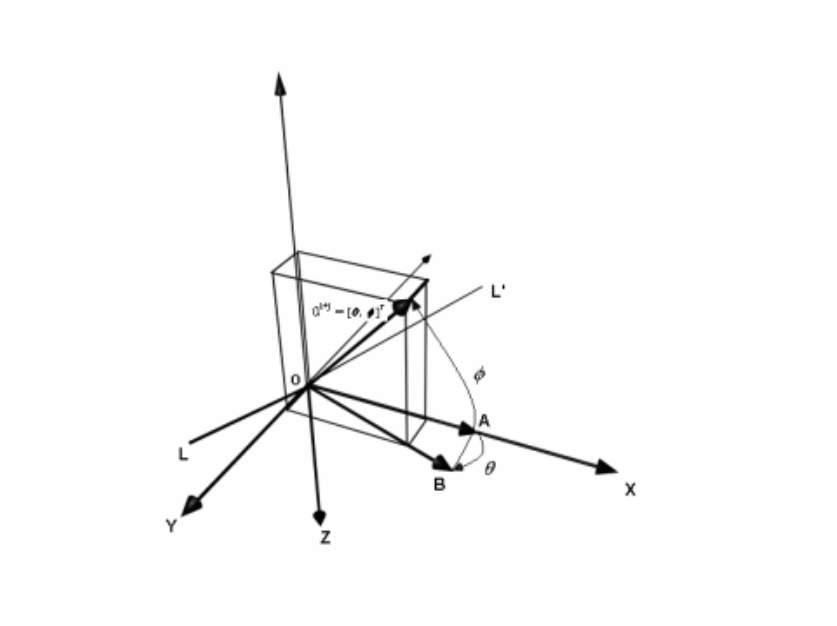


Figure 46. The coordinate system for specifying angular directions. The vector $\mathbf{u}^{i+j}$ is obtained by rotating the vector OA by θ^o relative to the x axis in the x-y plane, and then rotating the resulting vector (i.e., OB) by ϕ^o relative to the x-y plane.

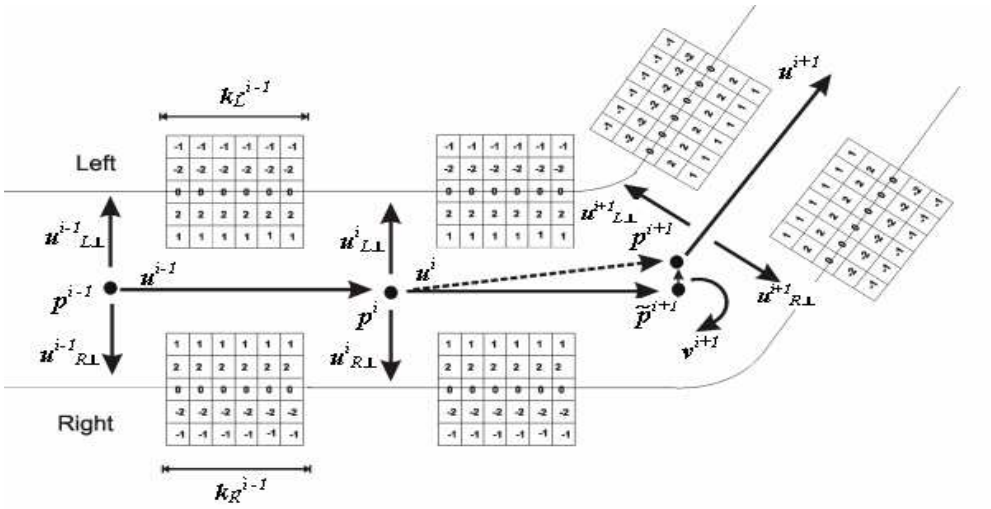


Figure 47. An (intermediate) step of 2D-tracing algorithm. P^{i-1}, P^{i} and P^{i+1} are the previous, current and next centerline points in the iteration.The next postion $P\tilde{}^{i+1}$ in the iteration is computed based on the current position, current direction u^{i}, and the right and left boundary points calculated along perpendicular directions $\boldsymbol{u^{i}_{R\perp}}$ and $\boldsymbol{u^{i}_{L\perp}}$; P^{i+1} is corrected by vector $\boldsymbol{v^{i+1}}$ to the next centerline point; directions of maximal response kernels and u^{i} together define the next tracing direction; k_L^{i-1} and k_R^{i-1} are variable kernel lengths.

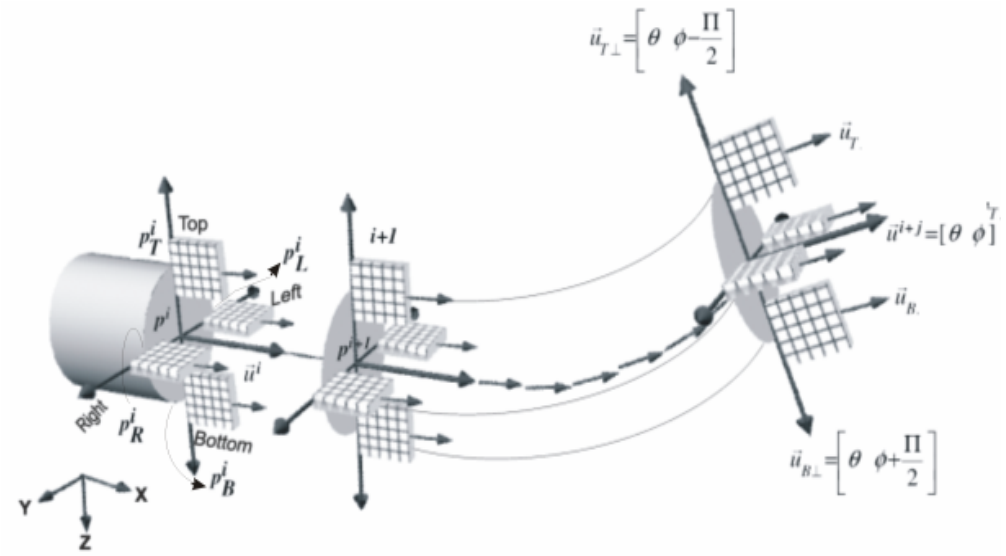


Figure 48. The 3D-tracing algorithm. P^{i+1} is the centerline point with P^{T}, P^{R}, P^{B} and P^{L} as the center points of top, right, bottom and left kernels respectively. Direction u^{i} is calculated based on the directions of the strongest kernel responses along the four perpendicular directions $\boldsymbol{u_L}$, $\boldsymbol{u_R}$, $\boldsymbol{u_T}$, $\boldsymbol{u_B}$. Θ and Φ define an NXN angular direction space in which each kernel's response is computed.

2. Notation

Figure 48 describes the relative placement of kernels in the vector-tracing algorithm.

- To index the discrete angular orientations in 3D, we use the notation s_1 and s_2, where s_1 and $s_2 \in \{0,\ldots..., \text{N-1}\}$. Given a unit vector $\boldsymbol{u}$ with the orientation $[\theta, \phi] = [\ 2\pi s_1/\text{N}, 2\pi s_2/\text{N}]$, this orientation can be expressed in terms of s_1 and s_2 as $[s_1, s_2]^{\text{T}}$. The templates are correlated repeatedly to search for structure boundaries in a direction $\boldsymbol{u}_{\perp}$, that is perpendicular to the direction of the structure orientation u.
- k is the length of the template, i.e., the number of voxels in the direction of the structure that are to be considered. The template length k has a lower bound by ,

  $$k \geq \lceil\ 1/\sin(2\pi/N) \rceil.$$

  At equality there is a difference of at most one voxel between the templates along adjacent vectors at their far end.
- $R(\mathbf{u}_R, k, \text{p})$ denotes the correlation response of a right template of length k, and orientation $\mathbf{u}_R$, with the image data $\text{I}(x, y, z)$ when the template is centered at the image point $\text{p}(x, y, z)$. Similarly we can define for left, top and bottom templates the correlation response as $L(\mathbf{u}_R, k, \text{p})$, $T(\mathbf{u}_R, k, \text{p})$, and $B(\mathbf{u}_R, k, \text{p})$ respectively.
- $\{p^{\text{i}}_R, p^{\text{i}}_L, p^{\text{i}}_T, p^{\text{i}}_B\}$ are the points along the directions Right, Left, Top and Bottom that produce maximum template responces at the structure boundaries respectively. The corresponding local direction estimates are $\{\boldsymbol{u}^{\text{i}}_R, \boldsymbol{u}^{\text{i}}_L, \boldsymbol{u}^{\text{i}}_T, \boldsymbol{u}^{\text{i}}_B\}$. For the top template this can be defined as,

  $$(p_T^i, u_T^i) = \underset{\{(p,u_T)\,|\,p=\tilde{p}^i+mu_{T\perp},\ m=1,\ldots,\ M/2,\ and\ u_T\in\Sigma\}}{\arg\max} T(u_T, k, p)$$

  where M is the maximum expected dendrite/axon diameter. The definitions for the rest of the directions are similar. $\boldsymbol{u}_{\text{T}\perp}$ is a function of u_{T}, as defined in Table 2.
- $\hat{R} = R(u_R^i, k, p_R^i)$ denotes the maximal response of the right template at the boundary point estimated.

Table 2. Perpendicular shift directions for the four templates: right, left, top, and bottom, from Ref. [5].

TEMPLATE	DIRECTION	PERPENDICULAR SHIFT DIRECTION
RIGHT	$u_R = \begin{bmatrix} s_1 & s_2 \end{bmatrix}^T$	$u_{R\perp} = \begin{bmatrix} s_1 + \frac{N}{4} & s_2 \end{bmatrix}^T$
LEFT	$u_L = \begin{bmatrix} s_1 & s_2 \end{bmatrix}^T$	$u_{L\perp} = \begin{bmatrix} s_1 - \frac{N}{4} & s_2 \end{bmatrix}^T$
TOP	$u_T = \begin{bmatrix} s_1 & s_2 \end{bmatrix}^T$	$u_{T\perp} = \begin{bmatrix} s_1 & s_2 - \frac{N}{4} \end{bmatrix}^T$
BOTTOM	$u_B = \begin{bmatrix} s_1 & s_2 \end{bmatrix}^T$	$u_{B\perp} = \begin{bmatrix} s_1 & s_2 + \frac{N}{4} \end{bmatrix}^T$

3. Iteration

For a 2D case, we obtain the point in the next iteration, using the point and the direction in the previous iteration and the equation is defined by,

$$p^{i+1} = p^i + \alpha^i \boldsymbol{u}^i$$

where α is a step size, defined by,

$$\alpha^i = \max\{3, \frac{1}{4}\min(k_R^i, k_L^i, k_T^i, k_B^i)\}$$

But when we have high local curvature, the above equation produces nonsmooth traces. In order to correct this, we add a fine-tuning step to the above equation, resulting in

$$\tilde{p}^{i+1} = p^i + \alpha^i u^i$$

$$p^{i+1} = \tilde{p}^{i+1} + \alpha^i v^{i+1}$$

where $\mathbf{v}^{i+1}$ is a correction (fine-tuning) vector, and "~" indicates approximation. Figure 47, illustrates the 2D-tracing algorithm.

For 3D case the method to refine the location and direction estimates $\tilde{p}^i$ and $\tilde{u}^i$ can be defined as,

$$p^i = \begin{bmatrix} x^i & y^i & z^i \end{bmatrix}^T$$

$$= \frac{\tilde{p}^i}{2} + \begin{bmatrix} \dfrac{\hat{R}^i x_R^i + \hat{L}^i x_L^i}{2(\hat{R}^i + \hat{L}^i)} & \dfrac{\hat{R}^i y_R^i + \hat{L}^i y_L^i}{2(\hat{R}^i + \hat{L}^i)} & \dfrac{\hat{T}^i z_T^i + \hat{B}^i z_B^i}{2(\hat{T}^i + \hat{B}^i)} \end{bmatrix}^T$$

$$u^i = \begin{bmatrix} s_1^i & s_2^i \end{bmatrix}^T$$

$$= \frac{\tilde{u}^i}{2} + \begin{bmatrix} \dfrac{\hat{R}^i \tilde{s}_{1R}^i + \hat{L}^i \tilde{s}_{1L}^i}{2(\hat{R}^i + \hat{L}^i)} & \dfrac{\hat{T}^i \tilde{s}_{2T}^i + \hat{B}^i \tilde{s}_{2B}^i}{2(\hat{T}^i + \hat{B}^i)} \end{bmatrix}^T$$

Using these equations, we can define the estimates for the location and direction of the next centerline point,

$$\tilde{p}^{i+1} = p^i + \alpha^i u^i$$

$$\tilde{u}^{i+1} = u^i$$

Figure 48 illustrates the 3D-tracing algorithm.

CHAPTER IX

SELECTION OF SEED POINTS, DETECTION OF BRANCH POINTS, AND STOPPING CRITERION

1. Selection of Seed Points

Reconstruction efficiency in part is based on effective of the selection of seed points. As mentioned earlier, the strategy used by Al-Kofahi et al., [5], is neither effective nor feasible for the volumetric data set used in our work. The number of sectional 2D images that we have in their volumetric data set are large. Projecting all these stacks of images into a single 2D image plane, processing this projected image, and then reprojecting seed points back into 3D space, is computationally costly. In part one can overcome this problem by considering only a few images at a time. However the volumetric data sets used in this work are large in comparision to those used in Al-Kofahi et al., paper, i.e. number of neurons to be reconstructed are large, in range of thousands, when compared to isolated neurons used there. Considering the computation time required to distinguishing these multiple neurons within the projected 2D image is like distinguishing individual trees in winter time when viewed from a distance across a field. Figure 16, Figure 20 and Figure 22 drawn by Santiago Ramón y Cajal are the 2D-projections of the 3D-scenes. We cannot distinguishing individual fibers from these 2D-images. This method of seed point generation was not used.

To avoid these problems we employ a different seed-point selection scheme. This scheme stems from the hybrid approach of combining the polymerization algorithm with vector tracing. We first apply the polymerization algorithm [1], [2] over a given data set, and output a network of L-block clusters, which are not necessarily connected. It is logical to assume that each cluster is a part of the neurite or a neuron, and that neighbouring clusters might be connected. So we start by considering mid/center point

of each cluster. For each center point we use a brute-force approach to check for the specific orientation that gives maximum response. We employ this approach only once per cluster. No additional computational time, except for calculation of the initial orientation, is associated with this seed-point selection process, as the output of polymerization process provides a list of clusters. For each cluster we store the position of the cluster center and both the cluster orientation and its reverse oreintation. The rational for considering two opposite directions is that each cluster might have lost connections in either directions. The selected seed points are further filtered, based on size of the clusters. Clusters of smaller size normally are result of noise rather than data, so we remove these clusters and their associated seed points.

2. Detection of Branch Points

During the iterative tracing of the neurite, the algorithm might encounter a junction where the neurite branches. These points have to be detected before the tracing selects one of the arbitrary directions. For tracing technique used here we ignore detection of branch points. During the tracing process we proceed to trace the structure depending on the maximum response obtained at each iteration. When the algorithm encounters a branch point, it continues tracing, depending only on the branch that gives the maximum response. The bypassed branch is traced at a later stage, when the cluster to which the branch belongs is traced. As mentioned earlier, from the given center point of each cluster we trace the neurite in both the (oppposite) directions. This ensures that every bypassed branch is covered during the tracing process. Consider Figure 49, assuming that the underlined structure (in grey), is enclosed by the three clusters respectively. For each cluster we calculate the initial orientation and an orientation opposite to it from the center points of the cluster. Starting with cluster I (center point A and initial orientation), we trace the structure. When we reach point D (the branch point), we get two maximum responces respectively for the two clusters (cluster II and III). The algorithm can take any of the clusters to proceed with the tracing. Assuming that the algorithm has taken the path leading to cluster III, it might trace the structure till it

satisfies a stopping criterion. Cluster II would be traced later, when we start processing it at a later stage, starting from its center point B. Explained in the next section, we have taken care of repetitions during the tracing process.

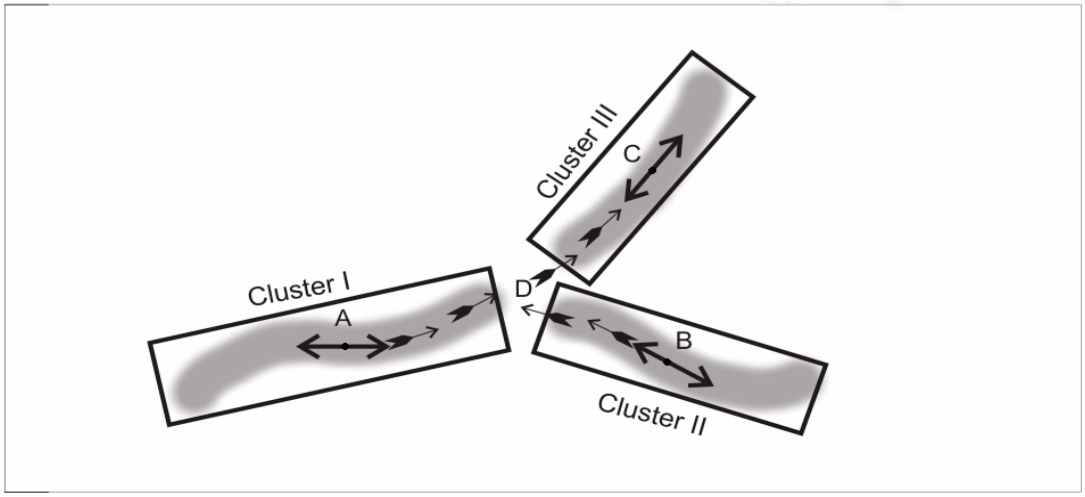


Figure 49. Neuron tracing at a branch point. Points A, B and C are the center points of Clusters I, II and III respectively. Point D is the intersection/branch point of the three clusters. The algorithm starts at A(Cluster I), and traces the underlying structure. At D, the algorithm might decide to trace Cluster III, Cluster II is traced when it starts tracing from the seed point B at a later stage.

3. Stopping Criteria Membrane

The tissue in the input data images dealt in this work is lighter than the background. As the intensity of the background is zero, it aids us in differentiating the tissue (which has a higher intensity value) from the background. We use this difference as a stopping criterion for the hybrid algorithm. Given the list of clusters, we consider one cluster at a time and trace the underlying structure until all clusters are covered. For a given cluster, we start the tracing from the center point of the clusters and proceed by calculating template responses for each iteration. As mentioned earlier, the next position and orientation of the next point during the tracing is based on the calculated maximum template responces. When the iterations are in the region of no-tissue or purely background, the template responses are either zero or close to zero. We stop iteration of

the algorithm (for that particular cluster) when the summation of the magnitudes of the four templates responces is zero. This stopping criterion is computationally inexpensive and gives good results as can be seen in Chapters X and XI. In order to prevent the algorithm from accidently blowing into a no-tissue/background region we use variable-width templates, depending on the curvature of the neurite. The width of the template is adaptively changed according the the curvature of the structure being traced.

Along with defining a stopping criterion, we must take care of repetitions that might occur during the tracing process. This process takes advantage of the L-block data structure. During the iterations, the data structure stores a list of all voxels traced so far. For each iteration we search this list to see if the voxel to be considered in the next iteration has already been traced by the algorithm. We stop the iterations for the cluster, if we find the pixel in the list. Else we add the voxel to the list and proceed with iteration. Checking for an already traced voxel ensures that we do not trace the same branch or the same structure again. Using this process, in Figure 49, the hybrid algorithm avoids tracing either Cluster I or Cluster III, when we start tracing from point B.

CHAPTER X

APPLICATION OF THE EXTENDED POLYMERIZATION ALGORITHM ON SYNTHETIC VOLUME DATA SETS

The extended polymerization algorithm is tested here on synthetic data sets before applying it on biological volumetric data. This chapter concentrates more on tracing the neuron structures highlighting some of the drawbacks of the polymerization algorithm [1], [2]. Though tracing blob data is not generally of interest, a couple of examples are given to show the performance of the algorithm.

Synthetic data sets used for testing purposes have been divided into three categories: *special data sets, filamentary data sets*, and *blob data sets*. Similar data sets to the last two are encountered in the scanned mouse brain tissue. The rationale for selecting specific data sets has been explained in Chapter IV. For each data set we have provided its visualization obtained from commercial software, Amira [28]. This software, as mentioned earlier, constructs an iso-surface for a given data set. Presently we represent every point that is traced by a 2x2x2 L-block, resulting in not so smooth structures.

1. Special Data Sets

i. Cork-Screw Spiral

This data set mimics the neuronal fibers (with curvature) seen in the mouse brain tissue. Figure 50 shows the synthetic data set (spiral) reconstructed using Amira software and traced by the extended polymerization algorithm. Isolated points, seen in Figure 50, were introduced in the data set to test the performance of the algorithm when it encounters isolated points. These isolated points resemble noise or unwanted data resulting in the scanned tissue data. To increase the complexity of the data set and to reflect scanning and staining artifacts, small gaps are introduced within the structure.

From the figure it can be seen that, given a seed point and initial direction, the algorithm traces underlying structure elegantly. The isolated points are not traced, as desired.

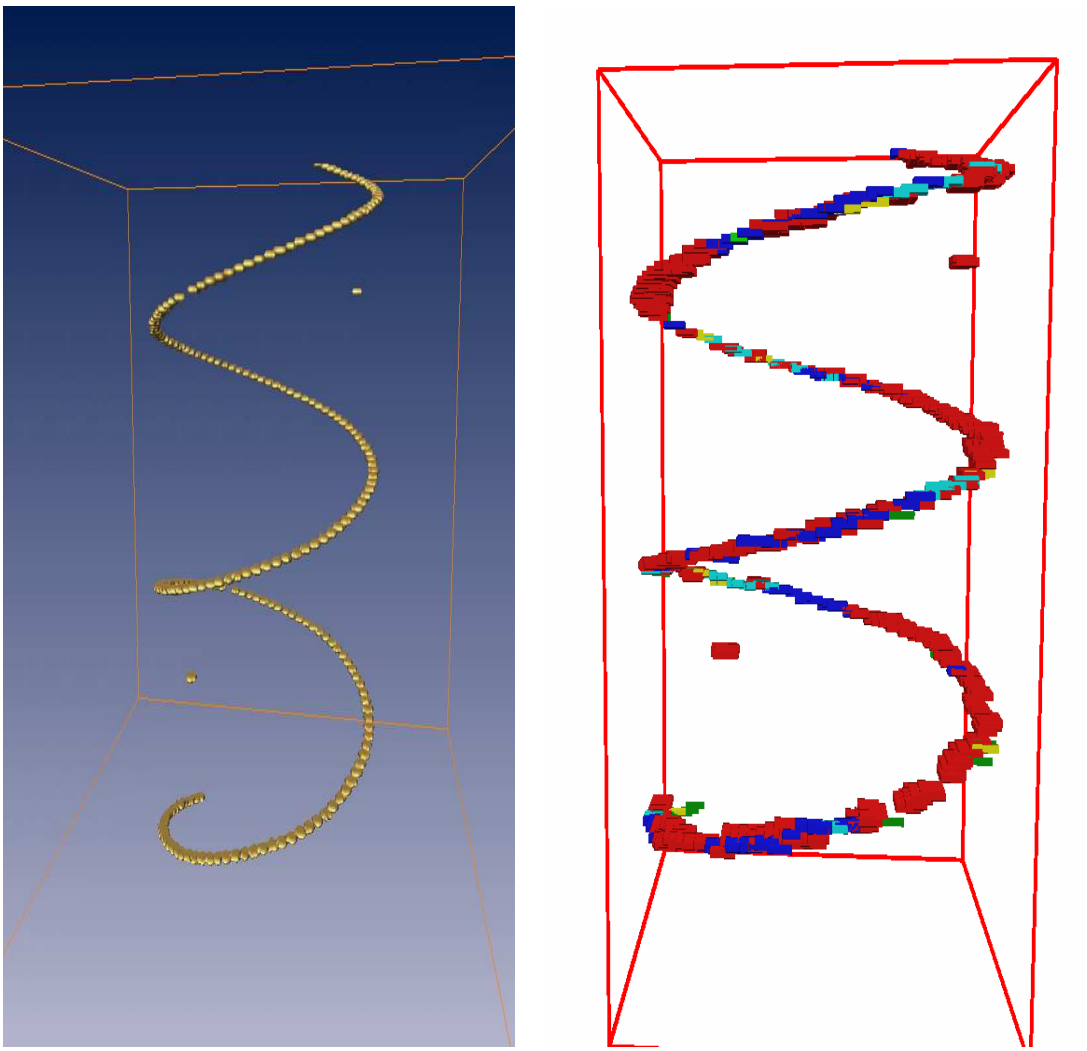
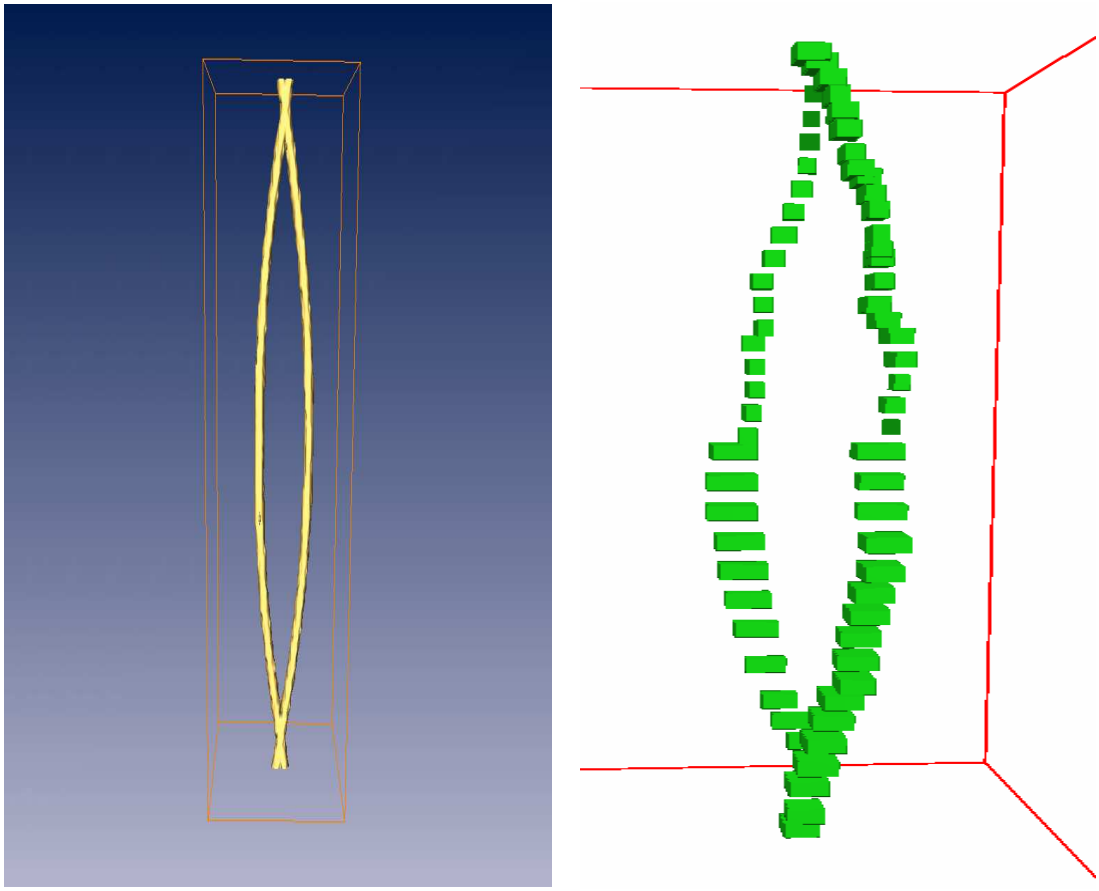


Figure 50. (a) Amira reconstruction of cork-screw spiral; (b) result of applying the extended polymerization algorithm on the same data set.

v. Torus

This data set resembles loops present among fibers, where they branch off and intersect at a later stage. Figure 51 shows the synthetic data set (torus) reconstructed using Amira software and traced by the extended polymerization algorithm. From the figure it can be seen that the algorithm traces the structure correctly. The points seen in the figure are the points traced during the iteration process.



(a) (b)

Figure 51. (a) Amira reconstruction of torus; (b) result of applying the extended polymerization algorithm on the same data set.

2. Filamentary Data

This data set, which has gaps with in the fibers, is taken from Chapter IV. As seen in the reconstructed results of Chapter IV, both polymerization algorithm and Amira software could not reconstruct these fibers completely. Inspite of using higher adjacency edges, the polymerization algorithm could not bridge these gaps, due to the large gaps within the structures.

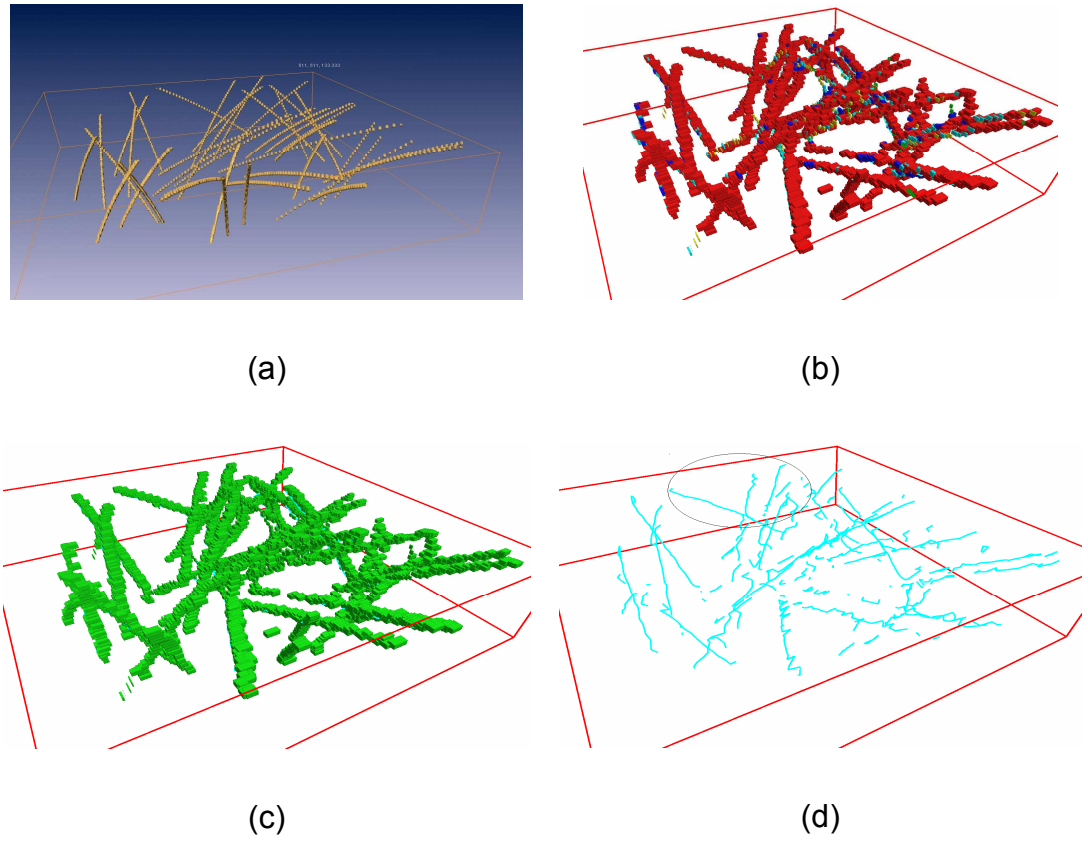


Figure 52. (a) Amira reconstruction of fine fibers; (b) (c) the result of applying extended polymerization algorithm on the same data set ; (c) colored L-blocks where the colors represent different sizes; (d) the result of thread generation after application of extended polymerization.

Figure 52 shows the reconstructed synthetic data set both using Amira software and the extended polymerization algorithm. It can be seen that the performance of the hybrid algorithm is better than both Amira software (Figure 53) and the polymerization algorithm (Figure 15). The extended polymerization algorithm bridges the large gaps between the fibers elegantly.

Figure 54 shows the enlarged region (highlighted elliptical region) shown in Figure 52 (d). Figure 54(a) is the result of applying thread generation after polymerization. Figure 54(b) is the result of applying thread generation after application of the extended polymerization algorithm. Figure 54(c) is the snapshot of Amira software reconstruction. It can be seen that the hybrid algorithm performs far better than the others. It traces the underlying structure by bridging the large gaps between them.

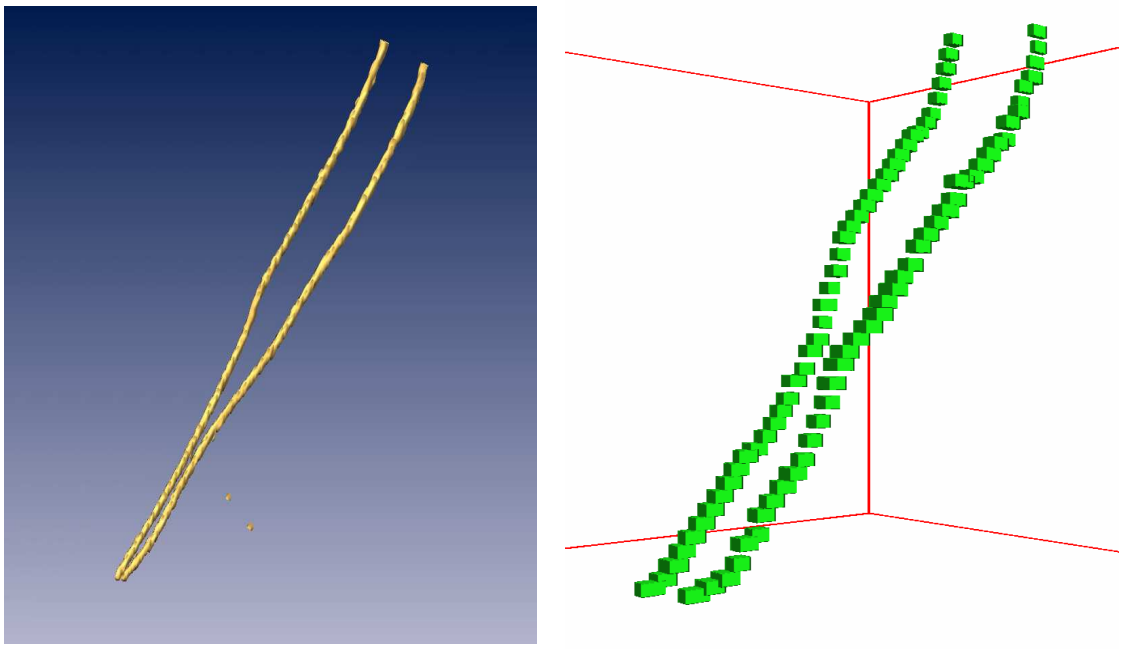
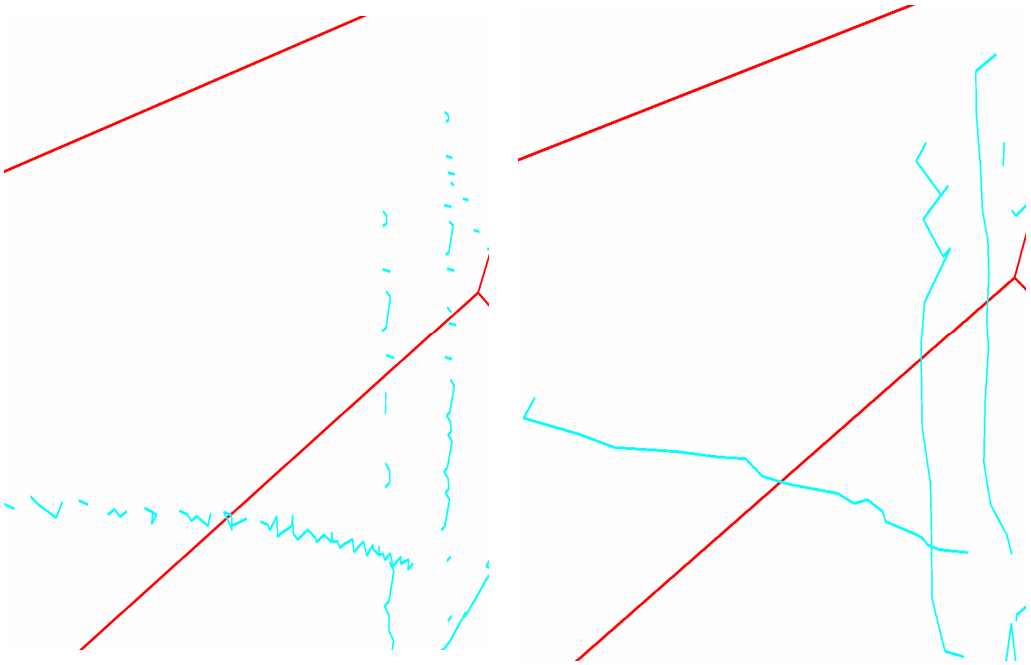
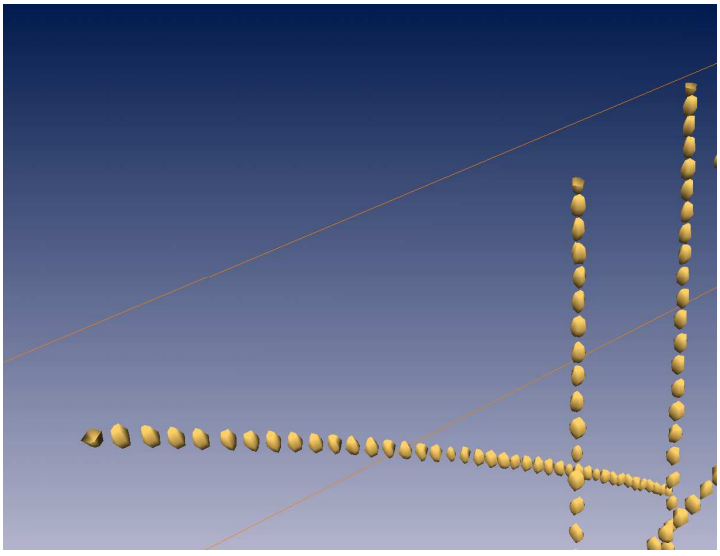


Figure 53. (a) Amira reconstruction of two parallel fibers; (b) result of applying extended polymerization algorithm on the same data set.



(a)

(b)



(c)

Figure 54. (a) Result of thread generation after polymerization; (b) result of thread generation after application of extended polymerization; (c) Amira reconstruction of the same data set.

Figure 53 shows two parallel fibers with little distance between them. This data set tests if the extended algorithm can resolve two nearby parallel fibers. Figure 53(b) shows the result of applying the extended polymerization algorithm on the data set. It can be seen that the algorithm traces the structures without confusing among close fibers.

3. Blob Data

Extended polymerization algorithm was mainly aimed at tracing filamentary structures. To test the performance of the algorithm in the presence of blob data, we have used a couple of data sets from Chapter IV.

Figure 55(a) shows reconstruction of a cylindrical structure using the polymerization algorithm, while Figure 55(b) shows the traced structure using the extended polymerization algorithm. The points shown in the figure are the traced points during the iterations. The algorithm traces the center of the cylinder nicely without being confused by surrounding data.

Figure 56(a) shows reconstruction of a ellipsoidal structure using the polymerization algorithm and Figure 56(b) shows the traced structure using the extended polymerization algorithm. As earlier the points shown in the figure are the traced points during the iterations. From the figure we can see that though the tracing branches of initially, but later it merges to the center of the ellipsoid.

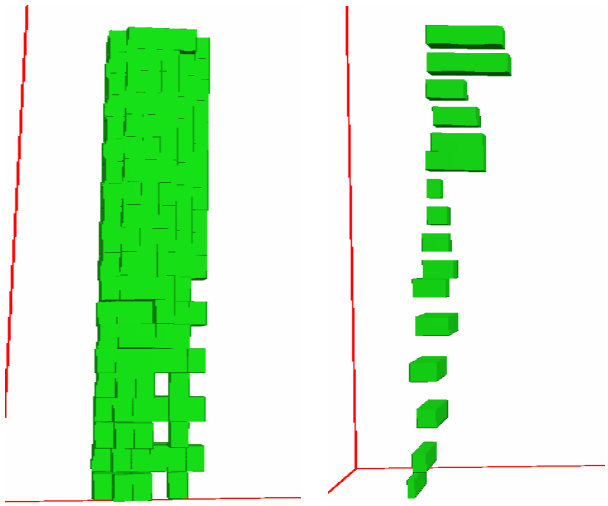


Figure 55. (a) Result of reconstruction using polymerization algorithm on a cylinder; (b) result of application of extended polymerization algorithm on the same data set.

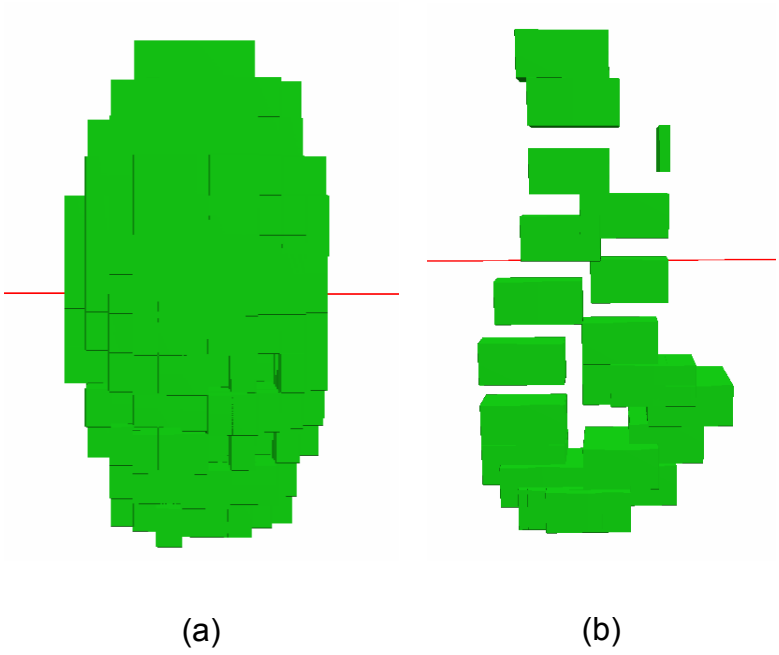


Figure 56. (a) result of reconstruction using polymerization algorithm on an ellipsoid; (b) result of application of extended polymerization algorithm on the same data set.

CHAPTER XI

APPLICATION OF THE EXTENDED POLYMERIZATION ALGORITHM ON NEURONAL VOLUME DATA SETS

1. Golgi-Stained Tissue

The extended polymerization algorithm was tested on neuronal volume data sets. One of the neuronal data sets was obtained by sectioning and scanning Golgi-stained tissue. Features of Golgi-stained tissue data have been described in Chapter V.

Figure 57 shows a sample image from the stack of images generated from Golgi-stained tissue data (Figure 57(a)), and also shows the results of reconstruction using the Amira software [28] (Figure 57(b)), the polymerization algorithm [1], [2] (Figure 57(e),(f)) and the extended polymerization algorithm (Figure 57(c),(d)). In this case the polymerization algorithm has got rid of some of the data, taking it for noise. The hybrid algorithm tries to recover this noise/missing data, either due to staining artifacts or scanning artifacts, and extracts the underlying neuronal structure. The results show that hybrid algorithm has the advantages of both the polymerization algorithm and neuron tracing.

Figure 58 shows the same slice of Golgi-stained tissue data. But a different part of the same data was used for reconstructions, as shown in the figures. Data is more dense in this region. It can be seen that the increase in density doesnot affect the algorithm and it performs well with out confusing the directions during the tracing process. The extended polymerization algorithm tries to reconstruct a thread-like structures.

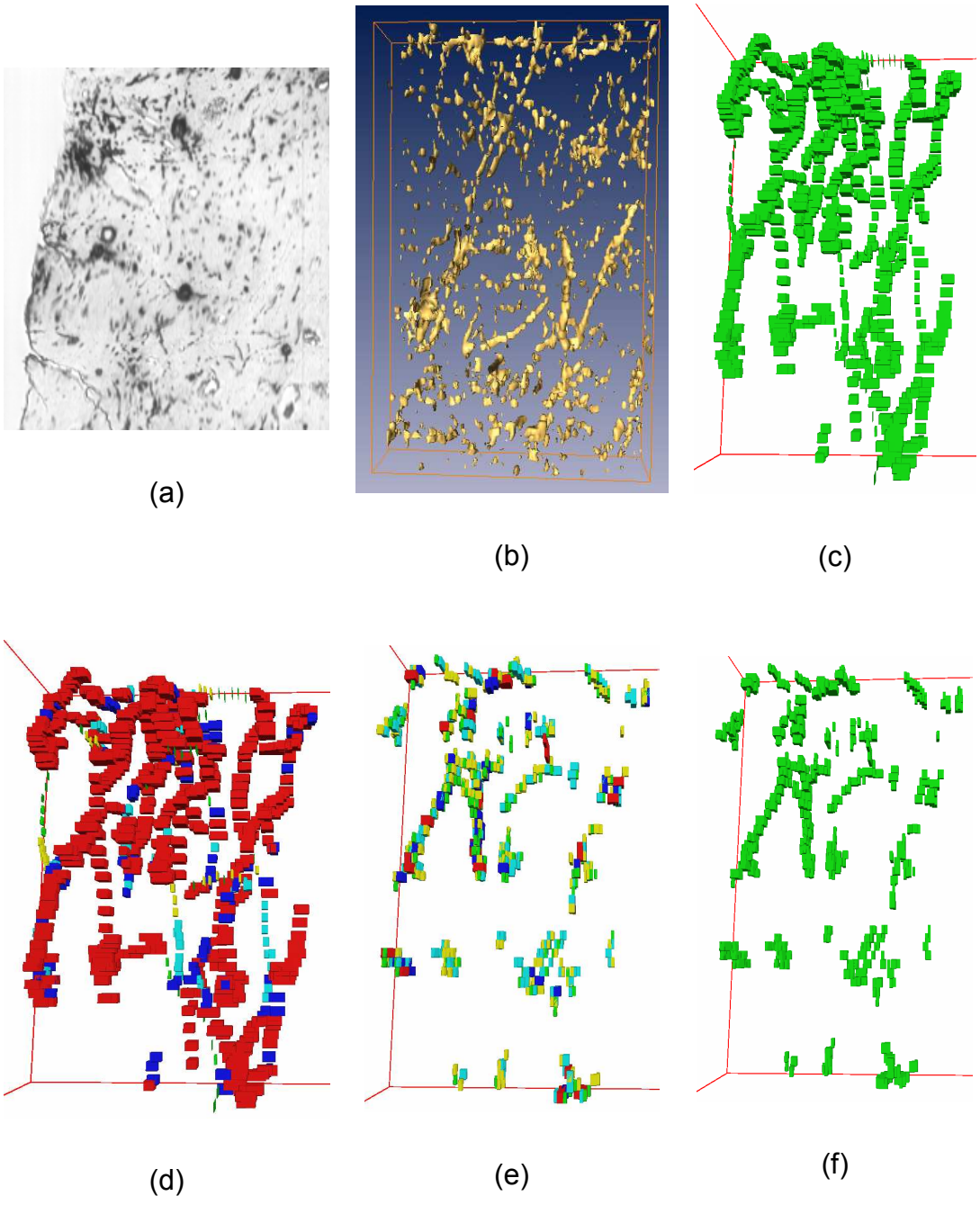


Figure 57. (a) 10X image of a slice of Golgi-stained tissue data; (b) Amira reconstruction of the corresponding image stack; (c) (d) result of applying extended polymerization algorithm on the same data set; (e) (f) result of polymerization algorithm on the same data set.

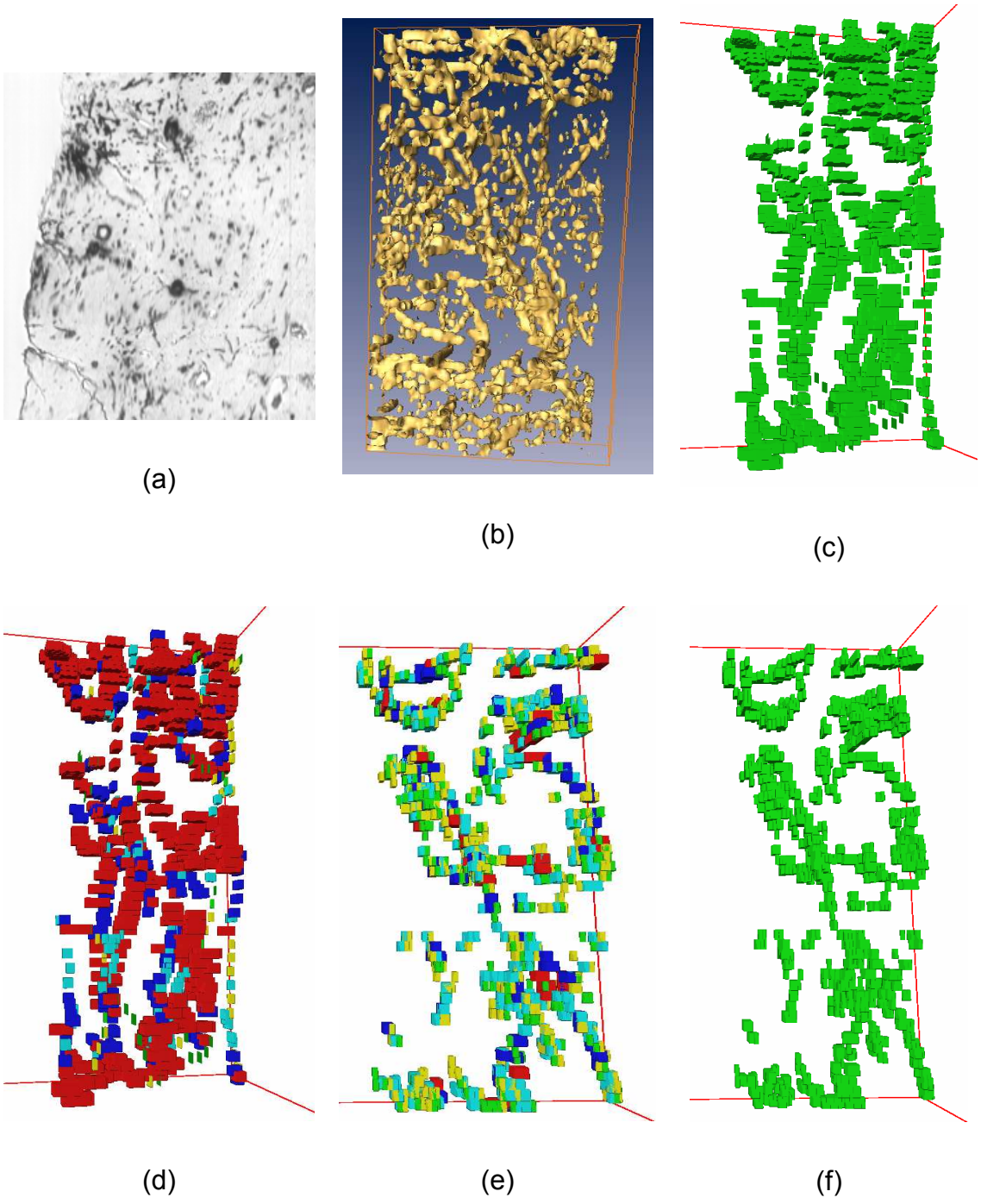


Figure 58. (a) 10X image of a slice of Golgi-stained tissue data; (b) Amira reconstruction of the corresponding image stack; (c) (d) result of applying extended polymerization algorithm on the same data set; (e) (f) result of polymerization algorithm on the same data set.

2. Nissl-Stained Tissue

The second set of neuronal data sets used to test the performance of extended polymerization algorithm was obtained by sectioning and scanning Nissl-stained tissue. Features of Nissl-stained tissue data have been previously explained in Chapter V.

The reconstruction of Nissl-stained tissue data takes more time than for Golgi-stained tissue data. The reason here is the sheer number of cell bodies to reconstruct and space occupied by each of the cell bodies.

Figure 59 shows a slice of the Nissl-stained tissue data(Figure 59(a)), and the results of reconstruction using the Amira software (Figure 59(b)), the polymerization algorithm (Figure 59(e),(f)) and the extended polymerization algorithm(Figure 59(c),(d)). The extended polymerized algorithm does not perform well when compared to polymerized algorithm in this case. The reason here is that almost the entire data set appears to be connected. The algorithm tries to trace the entire connected structure and this property affects the performance in turn.

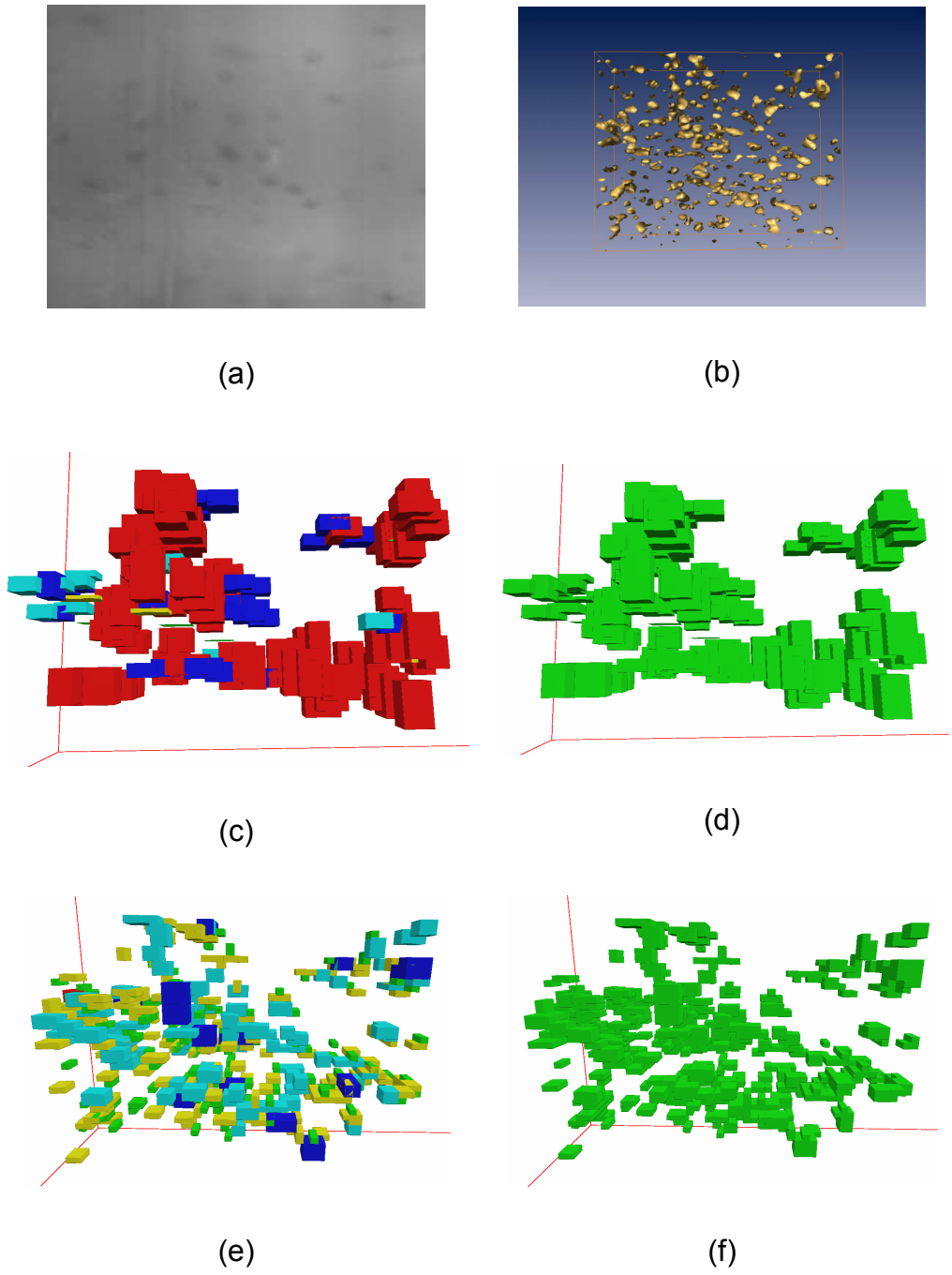


Figure 59. (a) A 10X image of a slice of Nissl-stained tissue data; (b) Amira reconstruction of the corresponding image stack; (c) (d) result of applying extended polymerization algorithm on the same data set; (e) (f) result of polymerization algorithm on the same data set.

CHAPTER XII

CONCLUSIONS AND FUTURE STUDY

1. Conclusions

This thesis has presented hierarchical segmentation strategies for three-dimensional reconstruction and visualization of scanned tissue data using the L-block data structure. This work shows the feasibility of reconstructing neuronal data sets, i.e., reconstructing stacks of consecutive raw input two-dimensional images into three-dimensional neuronal structures.

Polymerized volume data sets and their characteristics have been presented. Relevent existing data structures were presented and their drawbacks when representing neuronal data have been explained. The features of raw scanned data from brain tissue and problems associated in reconstructing the data have been described. Certain preprocessing techniques, tuned to specific sources of raw data to remove noise in the data are also presented.

The L-block data structure, L-block coverings/partitions, and some operations on L-blocks required for the reconstruction process are defined and explained. The polymerization algorithm has been described, as have also the steps involved in the three-dimensional reconstruction pipeline for volumetric data, from generation of the data through visualization of the reconstructed data.

A hybrid algorithm that extends the polymerization algorithm has been presented, as is the rationale for using vector-based tracing in this hybrid method. The existing literature on vector based tracing was surveyed, providing an overview of the vector tracing algorithm developed by Al-Kofahi et al. [5]. The design of matched filters used for edge detection in vector tracing have also been described.

Results are presented of the application of the polymerization algorithm and its extension by vector tracing, on both synthetic data (correlating real tissue data), and

three-dimensional light microscopic data. An analysis of the results is made showing the advantages and drawbacks of both algorithms. The efficiency of the algorithms in reconstructing synthetic data has been evaluated and the rationale for developing specific synthetic data has been explained.

2. Future Study

During the process of reconstruction we maintain a list of L-blocks processed so far. The size of this list increases in proportion to the size of the data set, leading to an increase in computation time. This computational time can be improved by maintaining smaller lists and processing them in parallel.

Another drawback of polymerization algorithm is, the computational cost when a data set having huge chucks of data/tissue in it is encountered. The algorithm tries to process the big chunk/blocks by processing smaller L-blocks. This extravelence can be avoided by having a mechanism that would identify such blocks and mark them as big blocks or clusters, so that no further processing is done on these blocks.

Presently we just trace the neuronal structure using the vector-based tracing. Futurework could provide geometric modeling of the neuronal structure.

During the process of reconstructing the tissue, the extended polymerization algorithm ignores the fine structure present in the tissue. For example, if we have a neurite with multiple spines attached to it, the algorithm doesnt pick them up during the reconstruction, giving us only neurites with smooth surfaces. We need to device a way to reconstruct these fine structures also using the information packaged by the polymerization algorithm.

REFERENCES

1. B. H. McCormick, B. Busse, P. Doddapaneni, Z. Melek, and J. Keyser, "Compression, segmentation, and modeling of filamentary volumetric data," in *Proceedings of 9th ACM Symposium on Solid Modeling and Applications*, Genova, Italy, June 9-11, 2004, pp. 333-338.
2. B. H. McCormick, 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, December 2002. Available from http://research.cs.tamu.edu/bnl.
3. B. H. McCormick, and P. Aragonda, "Volume data set enhancement I: Basic theory," (in preparation).
4. A. Can, H. Shen, J. N. Turner, H. L. Tanenbaum, and B. Roysam,"Rapid automated tracing and feature extraction from retinal fundus images using direct exploratory algorithms," *IEEE Trans. Inf. Technol. Biomed.*, vol. 3, pp. 125–138, June 1999.
5. K. A. Al-Kofahi, S. Lasek, D. H. Szarowski, C. J. Pace, G. Nagy, J. N. Turner, and B. Roysam, "Rapid automated three-dimensional tracing of neurons from confocal image stacks," *IEEE Trans. Inf. Technol. Biomed.*, vol. 6, no. 2, pp. 171–187, June 2002.
6. A. S. Winter, "Volume graphics, field based modelling and rendering," Ph.D. Thesis, Department of Computer Science, University of Wales, Swansea, December 2002.
7. J. F. Thompson, B.K. Soni, and N.P. Weatherill, "Handbook of grid generation," Boca Raton, Florida: CRC Press, 1999, pp. 1-4.
8. A. A. G. Requicha, "Representations for rigid solids: theory, methods and systems," *Computing Surveys*, vol. 12, no. 4, pp. 437-464, 1980.

9. J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes. “Computer graphics principles and practice,” second edition, Boston: Addison-Wesley, 1996, pp. 548-557.
10. L. March, and P. Steadman. “The geometry of environment,” Cambridge, MA: MIT Press, 1974.
11. C. Jackins, and S. L. Tanimoto. “Oct-trees and their use in representing 3-D objects,” *Computer Graphics and Image Processing*, vol. 14, pp. 249-270, 1980.
12. H. Samet, “The quadtree and related hierarchical data structures,” *ACM Computing Survey*, vol. 16, no. 2, pp.187-260, 1984.
13. H. Samet, and R. E. Webber, “Hierarchical data structures and algorithms for computer graphics, part 1: fundamentals,” *IEEE Computer Graphics and Applications*, vol. 5, pp. 48-68, 1988.
14. H. Samet, and R. E. Webber, “Hierarchical data structures and algorithms for computer graphics, part 2: applications,” *IEEE Computer Graphics and Applications*, vol. 7, pp. 59-75, 1988.
15. M. H. Overmars, and J. Van Leeuwen. “Dynamic multidimensional data structures based on quad- and k-d trees,” *Acta Inform*, vol. 17, pp. 235-267, 1982.
16. G. Van den Bergen, “Efficient collision detection of complex deformable models using AABB trees,” *Journal of Graphics Tools*, vol 2, no. 4, pp. 1-13, 1997.
17. B. H., McCormick, “Brain tissue scanner enables brain microstructure surveys,” *Neurocomputing*, vol. 44-46, pp. 1113-1118, 2002.
18. B. H., McCormick, “Design of a brain tissue scanner,” *Neurocomputing*, vol. 26-27, pp 1025-1032, 1999.
19. B. H. McCormick, “Development of the brain tissue scanner,” Technical Report, Department of Computer Science, Texas A&M University, College Station, March 18, 2002. Available from http://research.cs.tamu.edu/bnl.

20. B. P. Burton, B. H. McCormick, R. Torp, and J. H. Fallon. “Three-dimensional reconstruction of neuronal forests,” *Neurocomputing*, vol. 38-40, pp. 1643-1650, 2001.
21. B. H. McCormick, Y. Choe, W. Koh, L. C. Abbott, J. Keyser, Z. Melek, P. Doddapaneni, and D. Mayerich. “Construction of anatomically correct models of mouse brain networks,” *Neurocomputing*, 2004. In press.
22. M. M. David, “Acquisition and reconstruction of brain tissue using knife-edge scanning microscopy,” M.S. Thesis, Department of Computer Science, Texas A&M University, College Station, December 2003.
23. W. Koh, and B.H. McCormick, “Specifications for volume data acquisition in three-dimensional microscopy,” Technical Report, Department of Computer Science, Texas A&M University, College Station, December 2002. Available from http://research.cs.tamu.edu/bnl.
24. A. Prathyusha, “Strategy for construction of polymerized volume data sets,” M.S. Thesis, Department of Computer Science, Texas A&M University, College Station, December 2004.
25. R. Klette, and A. Rosenfeld, “Digital geometry: Geometric methods for digital picture analysis,” San Francisco, CA: Morgan Kaufmann, 2004, pp. 35-76.
26. T. Y. Kong, and A. Rosenfeld. “Digital topology: Introduction and survey,” *Computer Vision, Graphics, and Image Processing*, vol. 48, pp. 357-393, 1989.
27. R. S. Avila, L. M. Sobierajski, and A. E. Kaufmann, “Visualizing nerve cells,” *IEEE Computer Graphics and Applications*, vol. 14, pp.11-13, September 1994.
28. Amira - visualization and reconstruction for 3D image data. http://www.amiravis.com (2003).
29. S. R. Cajal, “Histology of the nervous system of man and vertebrates, Vol I,” N. Swanson, L. W. Swanson, Trans. New York: Oxford University Press, 1995, pp. 194, 256 & 577.

30. S. R. Cajal, “Histology of the nervous system of man and vertebrates, Vol II,” N. Swanson, L. W. Swanson, Trans. New York: Oxford University Press, 1995, pp. 418.
31. T. A. Hely, “Computational models of developing neural systems,” Ph.D. Thesis, Department of Cognitive Science, University of Edinburgh, Scotland, October 1999.
32. L. Zhou, M. S. Rzeszotarski, L. J. Singerman, and J. M. Chokreff, “The detection and quantification of retinopathy using digital angiograms,” *IEEE Trans. Med. Imag.,* vol. 13, pp. 619–626, December 1994.
33. I. Liu, and Y. Sun, “Recursive tracking of vascular networks in angiograms based on detection-deletion scheme,” *IEEE Trans. Medical Imaging*, vol. 12, pp. 334–341, 1993.
34. H. Shen, B. Roysam, C. V. Stewart, J. N. Turner, and H. L. Tanenbaum, “Optimal scheduling of tracing computations for real-time vascular landmark extraction from retinal fundus images,” *IEEE Trans. Inform.Technol. Biomed.*, vol. 5, March 2001.
35. R. Collorec and J. L. Coatrieux, “Vectorial tracking and directed contour finder for vascular network in digital subtraction angiography,” *Pattern Recog. Lett.,* vol. 8, no. 5, pp. 353–358, December 1988.
36. R. D. T. Janssen, and A. M. Vossepoel, “Adaptive vectorization of line drawing images,” *Computer Vision Image Understanding*, vol. 65, no. 1, pp. 38–56, 1997.
37. Y. Sun, R. J. Lucariello, and S. A. Chiaramida, “Directional low-pass filtering for improved accuracy and reproducibility of stenosis quantification in coronary arteriograms,” *IEEE Trans. Medical Imaging*, vol. 14, pp. 242–248, 1995.

VITA

Venkata Purna Doddapaneni was born on June 10, 1979 in Andhra Pradesh, India. In 2002 he graduated from Indian School of Mines(I.I.T.), India with a Bachelor of Technology in computer science and engineering. He received a Master of Science degree in computer science from Texas A&M University in December 2004. This thesis marks the end of his long trek through graduate school.

Starting in July 2004, he began work as a Software Quality Assurance and Development Engineer with NVIDIA Corporation, Autin. He can be reached by e-mail at purnad@nvidia.com or at the following address

12501 Tech Ridge Blvd, #215,

Austin, Texas 78753.