The Dissertation Committee for John Martin Edwards
certifies that this is the approved version of the following dissertation:

Analysis-Ready Models of Tortuous, Tightly Packed Geometries

Committee:

Chandrajit Bajaj, Supervisor

Greg Plaxton

Robert van de Geijn

Alan Cline

Wenping Wang

Daniel Johnston

Analysis-Ready Models of Tortuous, Tightly Packed Geometries

by

John Martin Edwards, B.S., M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2013

To Angela.

Acknowledgments

Thanks to Prof. Kristen Harris and her lab at the Center for Learning and Memory for use of the high resolution data, contours, and helpful conversations. Thanks also to Dr. Justin Kinney (MIT) and Drs. Tom Bartol and Terrence Sejnowski (Salk Institute) for discussions and for generously imparting of their domain knowledge.

I am indebted to Prof. Wenping Wang and his entire lab, who provided a fertile research environment during my stay at the University of Hong Kong in the spring of 2012.

Members of the Computational Visualization Center have all been helpful, particularly Dr. Andrew Gillette, Dr. Alexander Rand, and Md. Muhibur Rasheed. Special thanks to E. Greg Daniel who was supremely helpful down the stretch with discussions, coding, and paper writing.

My advisor, Prof. Chandrajit Bajaj, has offered valued guidance, criticism, and encouragement along the way. His expertise and willingness to take time to share it with me made this dissertation possible. I am also indebted to my thesis committee for their valued contributions.

And finally, my most heartfelt thanks to Angela, Laren, Martin, Wesley, Dane, Lydia, and Sadie, who have made sacrifices for this document that far exceed my own.

Analysis-Ready Models of Tortuous, Tightly Packed Geometries

Publication No. _____

John Martin Edwards, Ph.D.
The University of Texas at Austin, 2013

Supervisor: Chandrajit Bajaj

Complex networks of cells called neurons in the brain enable human learning and memory. The topology and electrophysiological function of these networks are affected by nano and microscale geometries of neurons. Understanding of these structure-function relationships in neurons is an important component of neuroscience in which simulation plays a fundamental role. This thesis addresses four specific geometric problems raised by modeling and simulation of intricate neuronal structure and behavior at the nanoscale.

The first two problems deal with 3D surface reconstruction: neurons are geometrically complex structures that are tightly intertwined in the brain, presenting great challenges in reconstruction. We present the first algorithm that reconstructs surface meshes from polygonal contours that provably guarantees watertight, manifold, and intersection-free forests of densely packed

structures. Many algorithms exist that produce surfaces from cross-sectional contours, but all either use heuristics in fitting the surface or they fail when presented with tortuous objects in close proximity. Our algorithm reconstructs surfaces that are not only internally correct, but are also free of intersections with other reconstructed objects in the same region. We also present a novel surface remeshing algorithm suitable for models of neuronal dual space.

The last two problems treated by this thesis deal with producing derivative models from surface meshes. A range of neuronal simulation methodologies exist and we offer a framework to derive appropriate models for each from surface meshes. We present two specific algorithms that yield analysis-ready 1D cable models in one case, and proposed "aligned cell" models in the other. In the creation of aligned cells we also present a novel adaptive distance transform.

Finally, we present a software package called VolRoverN in which we have implemented many of our algorithms and which we expect will serve as a repository of important tools for the neuronal modeling community.

Our algorithms are designed to meet the immediate needs of the neuroscience community, but as we show in this thesis, they are general and suitable for a variety of applications.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

We present advancements in construction of spatially realistic models of nanoscale neuronal geometry. This work comprises contributions in the fields of neuroscience, computational geometry, and computer graphics. Reconstruction of surface representations of neurons is a challenging problem and, until now, no comprehensive solution to creating analysis-ready ultrastructure models has existed. We present our work here, which not only fills the need for such models, but also contributes to other fields, such as robotics, geospatial information systems, scientific visualization, and object representation. We motivate our work in the context of neuronal modeling (Section 1.1) and briefly describe each contribution we have made (Section 1.2). We then give an overview of the organization of this dissertation (Section 1.3).

1.1 Background

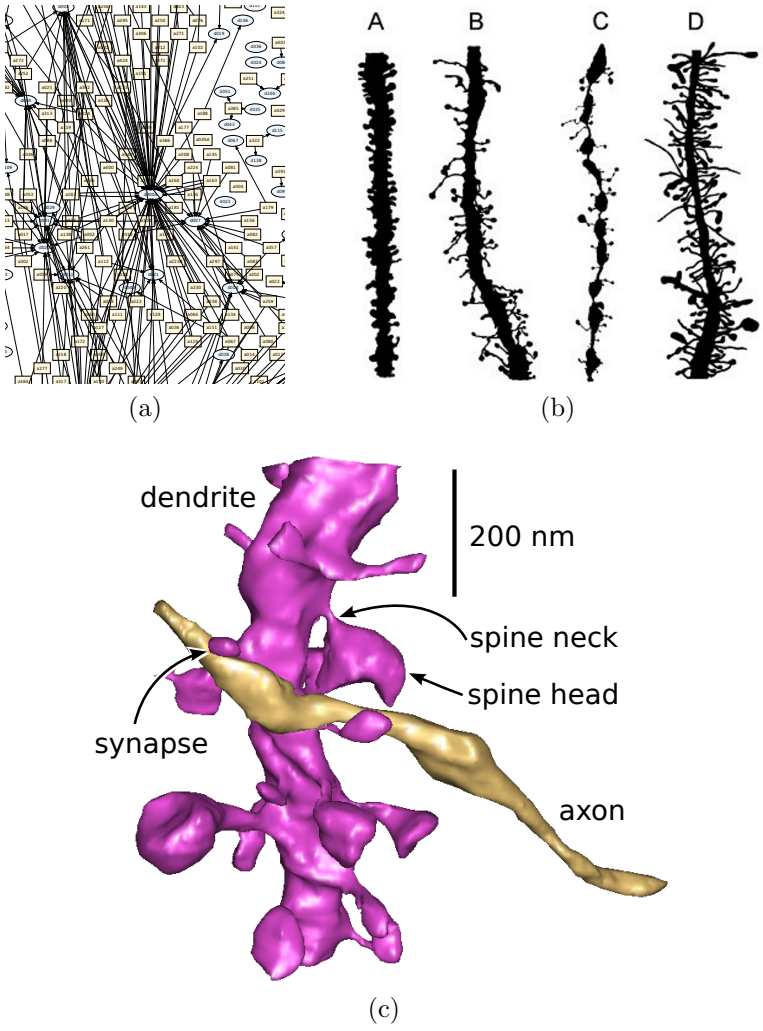
Learning and memory occurs through complex networks of cells called neurons. Neurons are highly specialized cells that have the ability to propagate electrical signals to and from other neurons via areas of close approach called synapses. It is in the topology and geometry of these synaptic connections

(Figure 1.1a) that memory and thought is encoded [6].

The number of neurons in the human brain is estimated to be 10^{10} with an estimated 10^{14} synaptic connections [43]. These numbers reveal the challenge of gaining understanding of the neuronal connection network and its relationship to learning modalities and pathologies (Figure 1.1b). With the recent successes of the human genome project, neuroscientists have launched into projects directed toward understanding the "comprehensive structural description of the network of elements and connections forming the human brain" [126], or connectome. Hagmann [65] describes the importance and magnitude of connectomics in these terms:

> It is clear that, like the genome, which is much more than just a juxtaposition of genes, the set of all neuronal connections in the brain is much more than the sum of their individual components....One could consider the brain connectome, the set of all neuronal connections, as one single entity, thus emphasizing the fact that the huge brain neuronal communication capacity and computational power critically relies on this subtle and incredibly complex connectivity architecture.

While many studies focus on the connectome as a whole, other studies focus on the ultrastructure, or nanoscale morphology, of the neurons themselves and how the ultrastructure affects the connectome [106]. Dendrites and axons are parts of the neuron (see Figure 1.2b), and at the level of light microscopy they appear as cylindrical cables. If we zoom in closer, however, we



(a) (b) (c)

Figure 1.1: (a) Neurons form complicated connection networks. (b) [54] Neuron geometries of neurological pathologies: [A] Neurologically normal. [B] Mentally retarded. [C] Severe neurobehavioral failure. [D] Fragile X syndrome. (c) Axons and dendrites form synapses in areas of close approach, typically on dendritic spine heads.

see that neurons are far from smooth cylinders. Dendrites in particular have complicated morphology, and it has been shown that their geometry at levels resolvable only by electron microscopy has a correlative relationship with brain function and that geometry varies across a range of neuropathologies [41, 101, 106, 114, 138]. These correlations are observable but not well understood. For example, dendrites in the hippocampal region of the brain are "spiny," meaning they have a main trunk and many branches called spines protruding from the trunk (Figure 1.1c). Spines are composed of two parts, the neck, or narrow region protruding from the trunk, and the head, which sits atop the neck. A synapse is a region where ions are passed from an axon of one neuron to the dendrite of another, and is typically located on a spine head. While the morphology of spines may appear arbitrary, many studies show a strong correlation between the geometries and brain function [41, 101, 106, 114, 138]. This is especially evident in comparisons of neurologically normal neurons with those from specimens with neuropathologies, where the numbers and shapes of spines vary widely (Figure 1.1b). These differences highlight the need to answer fundamental questions relating to neuronal structure-function relationships. Various functions of spines have been proposed [120]. A key question is what, if any, electrical function is served by spines. A debate is ongoing to discover whether spines induce electrical compartmentalization [7, 8, 62, 118, 135] or if they serve as biochemical and calcium compartments [23, 67, 74, 91, 131]. Even our understanding of the electrical differences between spiny and non-spiny dendrites is incomplete. For example,

it is unclear whether propagation of electrical signals is affected by possible diffusional traps in spine heads [117]. In relation to neuropathologies, questions include whether geometry is causal with respect to impaired function and whether any such causation is direct or if it is secondary by way of disruption of the neuronal connectivity graph.

In the absence of methods to physically measure geometry and signal propagation *in vivo* at the nanoscale, neuroscience relies on electrophysiological simulation to study high-resolution structure-function relationships. A common simulation methodology is that of cable model analysis [32, 70], which models a synaptic connectivity graph as a 1D circuit diagram. Charge is introduced in a region and potential is measured at certain timesteps in regions of interest in order to understand the propagation of the signal. Despite being a 1D model, geometry is implicitly encoded through resistances and capacitances. For example, a stretch of neuron will have higher resistance the longer it is. Cable model simulation is not an appropriate methodology for studies requiring high resolution in space or time, but it has proven extremely useful in macro- and microscale simulation. Microscale models for cable analysis are typically derived from light microscopy images using one of a number of effective staining and fluorescence techniques [124, 132]. Many robust tools are available to build such models from imagery [66, 141].

Cable model simulation of neuronal ultrastructure requires higher resolution than what traditional techniques can provide, and thus methods of generating models at the nanoscale are of increasing importance. Light can-

not resolve ultrastructure, and so electron microscopy (EM) techniques must be used.

Another simulation technique that requires nanoscale models is simulation of high-resolution 3D reaction-diffusion behavior of charge-carrying ions [87, 90]. Electrical brain signals are carried by ions such as Ca^+ and K^+. Neuroscientists study the spatio-temporal behavior of these ions in a very small region (e.g. in the region of a single spine head, or approximately $(40\text{ nm})^3$). While cable models reduce neurons to 1D models endowed with a small subset of geometric properties, reaction-diffusion studies require models of neuronal boundaries, typically described using surface meshes.

A surface mesh is a collection of polygons (typically triangles or quadrilaterals) that share edges. This dissertation deals almost exclusively with meshes composed of triangles. Physically realistic models have certain important characteristics, including, in our case, watertightness. Neurons are typically thought of as closed, or hole-free, surfaces, and so surface mesh models must accordingly be closed polyhedra.[1] More formally, every edge in a neuronal surface mesh must have exactly two incident triangles. Any edge with only one triangle neighbor constitutes a boundary and therefore, a hole, which is not physically realistic.

Two other important properties of surface meshes are manifoldness

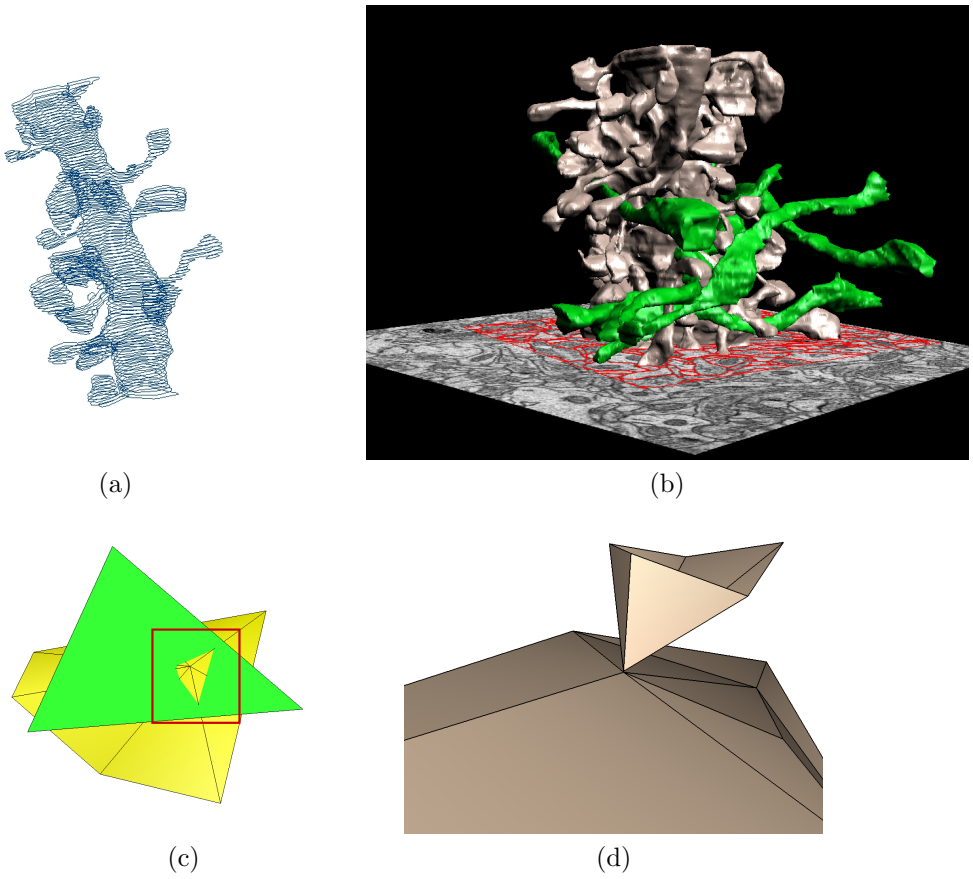
[1]We note that neurons do, in fact, have holes at the angstrom level that are used to send and receive neurotransmitters and ions, but reaction-diffusion simulations typically model these holes with time-varying ion densities, else the simulation become prohibitively complex and computationally expensive.

and lack of intersections. Intersections come in two forms. Self-intersections occur when the interiors of two or more triangles intersect. The other type of intersection is referred to as an object-object, or inter-object intersection. In this case two or more triangles from different surface meshes intersect (figure 1.2c).

A surface mesh is manifold if the intersection of an arbitrarily small ball with the surface at any point yields a topological disk, or a surface that is homeomorphic to a disk. As an example, a surface mesh is not manifold at an edge that have more than two incident triangles. Additionally, a mesh is not manifold at a vertex where the incident triangles cannot be uniquely ordered (figure

Other important properties of physically realistic surface meshes are treated in chapters 2 and 6.

While advanced tools for tracing light microscopy images for cable models and other studies are available, tools to build nanoscale models (figure 1.2b), both 1D cable and surface mesh, lag behind. This dissertation presents algorithms and software implementations to meet this need. With increasing availability of simulation tools such as MCell [88, 127], NEURON [32], and STEPS [68], tools that can automatically produce models are of growing importance. Until now, models have required many hours of manual manipulation in order to be suitable for analysis (e.g. [89, 90]). The algorithms presented in this document build on advances in neuroscience, computational geometry, and computer graphics to automate model construction, bringing



(a) (b) (c) (d)

Figure 1.2: (b) Surface meshes of a dendrite (tan) and axons (green) produced using our algorithms. (a) A stack of polygonal traces. (c) Object-object intersection. (d) The surface is nonmanifold at the point where the triangles meet.

to bear high-resolution data for 3D simulation.

At this stage we leave the term analysis-ready without a precise definition, as it is dependent on the type of simulation being performed. As each type of model created by our algorithms is presented we discuss specific suitability, or "readiness," requirements and how our models meet these requirements.

Our algorithms are tailored to geometric objects that are tightly-packed and highly tortuous. They are thus suitable for neuronal modeling, where objects that are hundreds of nanometers in diameter can be packed as closely as 10 nanometers. The intricate branching of dendrites and twisting of axons further complicate reconstruction. While the motivation and primary application of our algorithms is in neuronal modeling and simulation, the algorithms are general and, as is shown in each case, are applicable in a variety of settings.

1.2 Contributions

We now state the five primary contributions of this dissertation. Each statement is followed by a brief description which includes motivations, definitions, and justifications. Additional definitions are found in appendix A.

Contribution 1: Generate manifold, watertight, and non-intersecting surface meshes of object forests from stacks of polygonal contours [48]

As stated previously, manifoldness, watertightness, and lack of intersections are important properties of surface meshes in physical modeling. Neuronal

modeling, with twisting, branching, and meandering, make it particularly difficult to meet these criteria, particularly the requirement that meshes be free of self- and inter-object intersections. This is because of the physical imaging process from which models are derived.

Neuronal ultrastructure is resolvable only with physical imaging using electron microscopy (EM). Semi-automatic methods are used to trace polygonal contours around neuronal cross-sections using software such as RECONSTRUCT [53, 105], TrakEM2 [31], and ilastik [123]. These softwares are used interactively to generate stacks of polygons to which a surface mesh can be fit (figure 1.2a). Most contour tracing software packages do have surface mesh fitting capabilities, but the surface meshes have a large number of object-object intersections when objects are closely spaced. This is due to the fact that the spacing between polygon traces is very large compared to the tortuosity of the geometries. Intersections between objects are, of course, not a natural occurance and must be removed for a physically realistic simulation. Chapter 2 describes a novel algorithm for removing intersections as a post-reconstruction process with provable guarantees such as finiteness of the algorithm, preservation of contour interpolation, and surface separation distance. We describe the algorithm, discuss performance and show experimental results.

Contribution 2: Remesh surfaces for improved triangle quality without compromising manifoldness [49]

The process of remeshing is converting one mesh (surface meshes in our case) to another in order to reduce the number of triangles or to improve triangle

quality. Triangle quality is an additional important feature of surface meshes. The precise definition of quality is application dependent; in neuronal simulation we desire the triangles to be as close to equilateral as possible for fast and stable simulations. We use two measures of quality in this dissertation, both of which are maximized (or minimized) with equilateral triangles. Chapter 3 uses the Q measure (3.7), and Chapter 6 uses the triangle ratio (figure 6.11b) as well as minimum and maximum angle (table 6.2).

A popular remeshing algorithm that produces triangles of excellent quality [142] is the centroidal Voronoi tessellation (CVT). CVT is suitable for most meshes, but results in nonmanifold edges and vertices in areas of low curvature and small local feature size. Curvature is a measure of a surface's deviation from flatness. That is, areas of low curvature are flattish while areas of high curvature are tightly curving. Local feature size at a point is defined as the distance from the point to the medial axis, where the medial axis is the locus of points with at least two closest points on the surface. Two flattish regions that are close together are at risk of non-manifoldness when remeshed using CVT (figure 1.3). Some of our neuronal data exhibits these properties, inspiring our work on augmenting CVT to properly handle all cases. Chapter 3 describes what mesh qualities cause CVT to fail, explains our algorithm, and shows results on a variety of test cases.

Contribution 3: Accurately contract watertight polyhedra into 1D models of cylinder approximations [51]

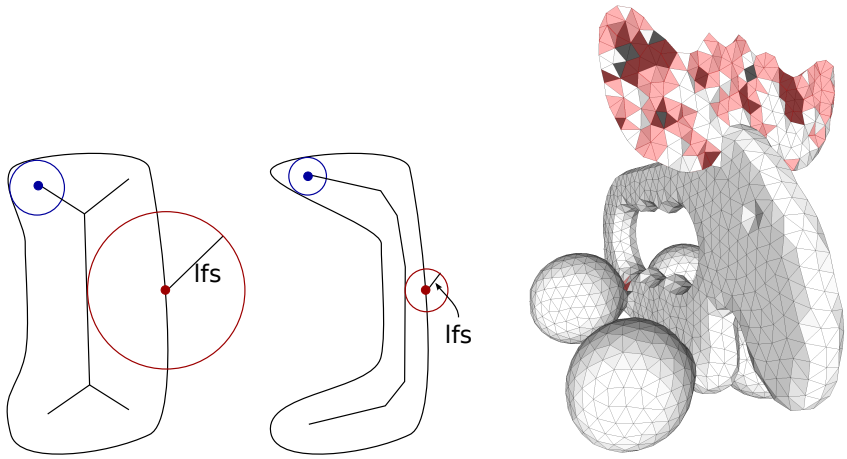


Figure 1.3: (a) A point where curvature is low but local feature size is high. (b) Curvature is identical but local feature size is small. (c) Effect of running standard CVT on a model with low curvature and low local feature size.

Simulation using cable theory [32] requires that neurons be described as a skeleton of segments, each with resistance and capacitance. Resistances and capacitances are typically derived from known physical properties of the neurons together with length and cross-sectional area of skeleton edges, conceptually "fitting" cylinders to best approximate neuronal surfaces. Models are typically generated directly from segmenting light microscopy images for microscale simulations. Because of recent work in simulation at the nanoscale level [88, 90], new models that are not obtainable from light microscopy are required. Skeletal cable models, where each edge of the skeleton is endowed with a length and cross-sectional area, can be derived by connecting the centers of adjacent contours, an approach used in the RECONSTRUCT software package. The primary problem with this approach is that it is restricted to

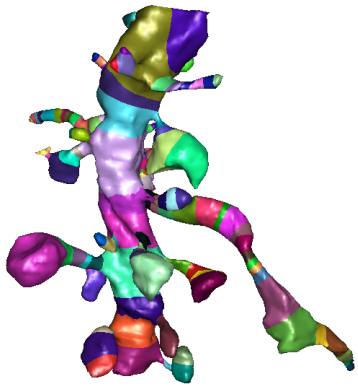


Figure 1.4: A segmented dendrite. Each colored region can be approximated with a cylinder. Surface area is computed exactly and length is approximated.

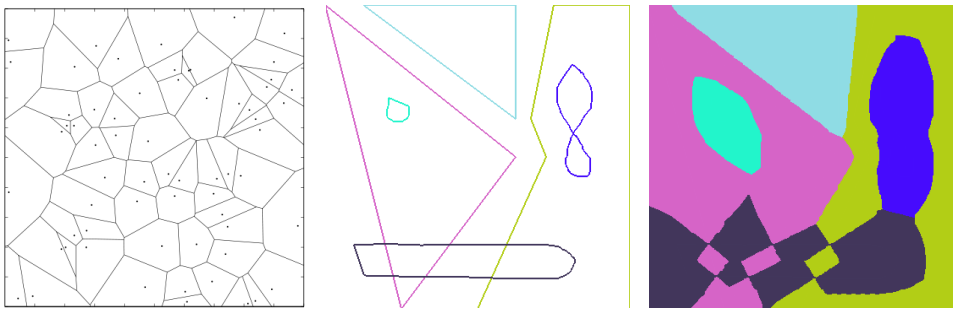
connecting centers in the image plane. If a portion of a neuron is oriented obliquely to the image plane, it may best be modeled with more segments within the inter-slice space.

We use an approach, based on previous work in skeletonization, that contracts a surface mesh repeatedly until triangles are nearly degenerate (i.e. have zero surface area). The contraction process implicitly generates a surface segmentation where triangles of the mesh are grouped into labeled regions. Using these labelings we compute edge lengths are volumes for each skeleton edge (figure 1.4). Our approach thus differs from previous approaches in that it derives cable circuits from surface meshes rather than stacked contours, giving us flexibility in "fitting" cylinders accurately. Chapter 4 describes and shows results of our approach to generating 1D cable models from surface meshes.

Contribution 4: Compute the generalized Voronoi diagram of tightly-packed objects [50]

The finite element method is a popular approach to solving differential equations over a spatial domain. It requires the domain to be split into elements, typically triangles or tetrahedra [42, 121]. The elements must respect domain boundaries, that is, no object boundary may intersect the interior of an element. Our neuronal data is very tightly packed, which means that a very large number of triangles will be required to appropriately mesh between objects. We propose a new "aligned" element, for use with meshless methods [144], that allows the boundary to pass through the interior of the element. The advantage of our proposed element is that regions that would require a prohibitive number of finite element tetrahedra can be modeled with a much smaller number of aligned elements, simplifying the simulation and rendering it more computationally feasible.

Tessellating, or splitting, a domain into aligned elements is a multi-step process, and Chapter 5 describes our contribution to that process, that of computing the generalized Voronoi diagram (GVD) of tightly-packed objects. The ordinary Voronoi diagram is defined for a finite set of sites $\{x_i\}$, or points, in a space X. The space is split into a complex of convex cells $\{VC_i\}$ such that $VC_i = \{p \in X | \text{dist}(p, x_i) < \text{dist}(p, x_j) \forall j \neq i\}$ (figure 1.5). Many efficient methods exist to compute the Voronoi diagram. What is less studied, however, is computation of the generalized Voronoi diagram, where sites are generalized to be any object. We describe our approach to computing the GVD using a so-called adaptive distance transform in an approximate, but error-bounded, way and give examples of a number of other applications our method is useful



(a) (b) (c)

Figure 1.5: (a) Voronoi diagram of a set of points. (b) A set of 2D objects. (c) Generalized Voronoi diagram (GVD) of the objects.

for.

Contribution 5: VolRoverN: software for modeling neuronal ultrastructure [51]

We have implemented our algorithms and housed many of them in the VolRoverN software package. Chapter 6 describes VolRoverN, its functionalities, comparisons with similar software, and interfacing abilities with simulation software.

1.3 Organization

This dissertation is organized as follows: chapters 2-6 treat the contributions listed above, one contribution per chapter. Chapter 7 contains concluding remarks and directions of future work. Appendix A contains a glossary of terms and mathematical symbols used throughout the dissertation.

Chapter 2

Intersection-Free Surface Reconstruction From Contours

This chapter presents a suite of algorithms that reconstruct surface mesh forests of objects such that the meshes are watertight, manifold, geometrically accurate, and intersection-free. The source data from which the surfaces are built are planar contours. This is a challenging problem when the objects are tightly-packed and geometrically complex, as is our neuronal modeling data. We first motivate and describe the problem in Section 2.1 and then give a brief outline of work that has been done both in single component and multi-component surface reconstruction from cross-sectional contour data (Section 2.2). We then build up a set of rules and theorems to prove correctness and robustness of our algorithm (Section 2.3), followed by a discussion of the

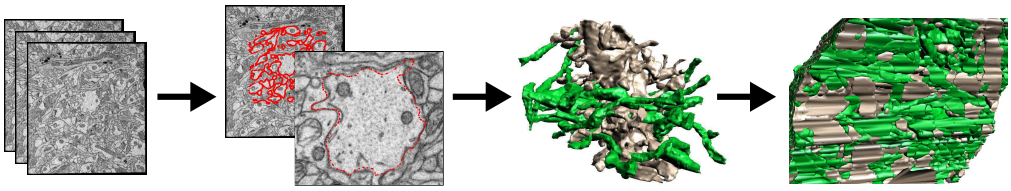


Figure 2.1: Overview of our automated neuronal reconstruction process. We begin with EM (TEM and SEM) images of the brain. We contour neuronal processes in 2D then generate each process individually. Finally we put everything together for a complete 3D reconstruction. See also Figure 2.14a.

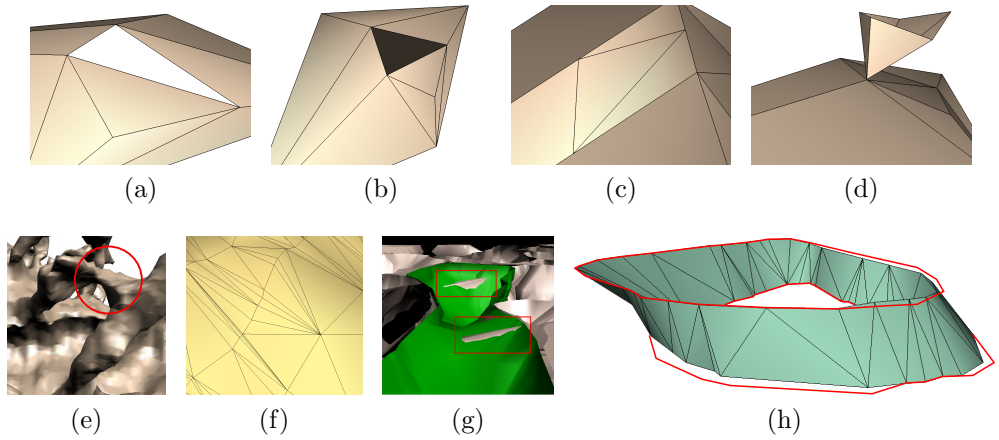


Figure 2.2: Quality requirements

single-component reconstruction improvements and intersection removal algorithm (Section 2.4). We then present results of our implementation (Section 2.5) followed by conclusions and future work (Section 2.6).

2.1 Introduction

3D models of neuronal ultrastructure are typically produced by first tracing polygonal contours around electron microscopy images of neurons and then fitting a surface to the contours in some meaningful way (Figure 2.1). In order to be suitable for analysis, surface mesh models must typically meet a number of criteria (Figure 2.2) including water-tight, manifold, and intersection-free. Much work on 3D reconstruction from planar cross-sectional data has been done in recent years. These single-object reconstruction methods are not sufficient, however, when faced with reconstructing models involving multiple objects or components, especially when the objects are intertwining and

tightly packed. This type of reconstruction is necessary in many different fields beyond neuronal modeling, including surgical planning and composite materials simulation. The problem is that reconstructing components one by one can yield intersections between components after compositing them into the same model, regardless of the guarantees made by the single component algorithm. The intersections occur frequently in data that is highly tortuous and densely packed, and is exacerbated further by highly anisotropic data, where the spacing of slices is large compared to the geometric behavior of the objects. The methods presented in this chapter aim to fill the gap in generating intersection-free multi-component models while maintaining the accuracy of existing single-component methods.

EM imagery used for neuronal reconstructions typically have xy-plane pixel spacing of roughly 2-5 nm, while spacing between slices is closer to 45 nm. Extracellular spacing (spacing between neuronal processes) is on the order tens of nanometers [90, 133]. This close spacing, combined with the comparatively large distance between slices can cause inter-object intersections between slices when using single-component reconstruction techniques. These intersections prohibit meaningful multi-component 3D simulation.

We have developed an intersection removal method that acts in concert with any surface reconstruction method, provided it conforms to several criteria. Our approach is to remove intersections by moving triangle vertices and induced points along the axis orthogonal to the slice plane. This approach can remove intersections efficiently and without causing additional intersec-

tions. Using an approach that moves vertices in the slice plane is possibly more intuitive, but yields considerably greater computational complexity [13].

We have also developed improvements to the single-component reconstruction approach, including dealing with several cases of numerical degeneracies and triangle shape improvement in some cases.

2.2 Related work

The single-component reconstruction problem is a well-studied topic. Fuchs et al. [58] presented the problem and proposed a solution based on triangulations guided by a toroidal graph. Barequet and Sharir [18] introduced a method using linear interpolations between slices of medical images. Bajaj et al. [11] expanded on their work by using medial axes to tile regions with no legal slice-to-slice tiling. Oliva et al. [109] specifically targeted difficult objects (objects with multiple branches, holes, and other irregularities) and used Voronoi diagrams to construct correct surfaces.

Somewhat more recent works interpolate contours using 2D skeletons in valid tile regions [17], Delaunay triangulation [139] and contour morphing [108]. Most works use some type of linear interpolation and thus require a smoothing post-processing step. One approach that performs non-linear smoothing during tiling is found in [19]. Other recent approaches reconstruct surfaces from non-parallel contours [20, 22, 25, 100]. The approaches in [22] and [136] are notable because they use implicit methods based on distance functions and radial basis functions, respectively, and naturally handle multi-

ple components. This is elegant, but implicit approaches can be troublesome when it comes to generating a geometric surface, as determining at what resolution to discretize the level-set isn't always straightforward. In addition, small components can be overwhelmed mid-slice by larger components and thus capped off at the slices rather than carried through the unknown inter-slice region. [100] uses a geometric approach but requires expensive calculation of both the arrangment of slices and the medial axis.

A different class of approaches that work directly from image data rather than contours exist, such as energy-minimizing 3D snakes [86] and a 3D extension to path cost-minimizing LiveWire [47]. These approaches require a level of user interactivity, however, and often don't scale well to massive datasets with large numbers of components.

Surprisingly little work has dealt with the issue of intersections between multiple components. Bajaj and Gillette [13] produced a method for removing intersections by removing contour overlaps in intermediate planes. This algorithm uses an inflated medial axis surface to separate mid-slice contours, but there is no guaranteed bound on the number of mid-slice contours to be separated. In addition, branching of components (where a single contour in one slice correlates to multiple contours in an adjacent slice) is treated in a pre-processing step, where branch points are moved as needed to enable intersection removal. Our algorithm has provably finite computational bounds and it also handles branching of components without treating them as a special case.

Of course, there are algorithms that use erosion morphological operators together with general boolean set operations on surfaces to solve the intersection problem [71]. These approaches, however, are more computationally expensive and don't preserve constraints imposed on the original initial single component reconstruction.

2.3 Rules

Our intersection removal algorithm is correct and robust and, in addition, it has bounded computational complexity and requires neither parameter optimization nor iterative mesh refinement. The only modifications made to the original surface are the addition of induced points and moving of points parallel to the axis orthogonal to the image plane, which we call the z axis. Restricting movement of offending points to the z axis we can provably remove intersections without causing additional intersections among other components. Allowing points to be moved in the x-y plane may yield better surfaces, but complicates intersection removal significantly. We discuss this in Section 2.6. Our method can also guarantee a minimum separation distance of δ. This is an important guarantee for applications that are sensitive to inter-component spacings.

2.3.1 Preconditions

We state here criteria on which our method relies, as well as various theorems which prove correctness and completeness. We define the xy plane

to be the plane of the original images (and slice contours), and the z axis to be perpendicular to the xy plane. For simplicity, we assume that each component has a unique color and each element belonging to that component (*e.g.* contour, boundary) inherits the same color. So if two contours have different colors, they belong to different components.

Criterion 1. *The reconstructed surfaces are piecewise closed polyhedra.*

Criterion 2. *Any vertical (parallel to the z axis) line segment between two adjacent slices either intersects a single component exactly once or not at all. Any intersection occurs either at a point or along a line segment.*

Criterion 3. *Slicing the reconstructed surface on any of the original slices produces exactly the input contours.*

Criterion 4. *All contours on the same slice have a minimum separation distance of δ.*

Criterion 5. *A contour cannot be nested inside a different colored contour. (This applies only to intersection removal. See discussion below.)*

Criteria 1-3 are borrowed from [11] and are required to ensure high quality and topologically correct (manifold) surfaces. Criterion 1 ensures that each component surface given to our algorithm are topologically correct and water-tight. Note that this does not guarantee that intersections between components won't exist. Criterion 2 also applies only to single components and helps to avoid topologies that are unlikely. Without it, a host of additional

correspondences are possible, most of which are incorrect. In order to avoid additional complexity and a large number of false positive correspondences, this criterion is carried through into our work. It assumes, however, that the slice spacing is close enough to enable reasonable reconstructions even with the criterion in place. Criterion 3 ensures that interpolation is bounded by only that required to generate a likely topology and avoids adding information to the original contours.

We enforce criterion 4 by adding a pre-processing step to our algorithm – that of separating contours by δ. Of course, if the application prohibits this then either δ can be decreased or application-specific contour separation methods can be used.

Criterion 5 is required as intersection removal between nested contours of different colors is not currently supported. In other words, components are treated as solids without any nested components. While this is a requirement of our current algorithm, we expect this criterion to be removed in future versions. See Section 2.6. Note that this does not preclude nested contours of the same color, which can occur when there are concavities in the surface.

2.3.2 Properties

Armed with these five preconditions we show that our algorithm has certain desirable properties that we mentioned earlier in the section and restate here for emphasis.

1. No more than two components can intersect in the same region (see

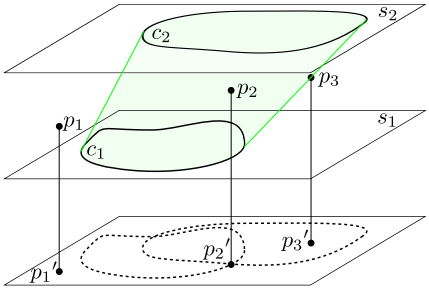


Figure 2.3: s_1 and s_2 are two adjacent slices and the lower plane is an arbitrary xy plane showing containing contours of various points. p_2 and p_3 are on the green component, while p_1 cannot be (see lemma 1 in Appendix B). $\mathscr{C}(p_1) = \emptyset$, $\mathscr{C}(p_2) = \{c_1, c_2\}$, $\mathscr{C}(p_3) = \{c_2\}$.

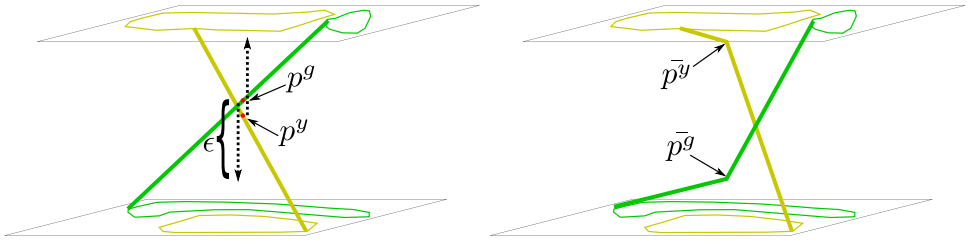


Figure 2.4: Conflict points are detected and removed by moving the points along the z axis. p^g and p^y are corresponding conflict points. The conflict is removed by moving p^g and p^y in the z direction by $s^g\epsilon$ and $s^y\epsilon$, respectively (equation 2.3) to produce new points $\bar{p^g}$ and $\bar{p^y}$.

theorem 1), so intersections can be resolved between components pair by pair.

2. Removing intersections in one region will not cause intersections in other regions (see theorem 3).

3. Pulling intersecting components apart at a finite number of appropriately chosen vertices will resolve all intersecting regions (see theorem 4).

We now argue these properties formally. In the following discussion, it is assumed that the data we are dealing with lies on and between a pair of adjacent slices. We also deal with only two components at a time, C^g and C^y, the green and yellow components, respectively. Theorem 1 shows that working with only two components at a time is justified. Proofs of the following theorems can be found in the appendix.

Definition 2.3.1. Suppose p' is the projection of point p onto the xy plane. We say that contour c is a *containing contour* of p if p' is inside or on the boundary of c' (see Figure 2.3). We define $\mathscr{C}(p)$ to be the set of all containing contours of point p and $\mathscr{C}^g(p)$ to be $\mathscr{C}(p)$ restricted to all green contours $\{c_i^g\}$, i.e., $\mathscr{C}^g(p) = \mathscr{C}(p) \cap \{c_i^g\}$.

Definition 2.3.2. S^g is the set of all points $p^g \in \partial C^g$ such that $|\mathscr{C}^g(p^g)| = 2$. These points lie "sandwiched" between two green contours. Similarly, U^g is the set of all points $p^g \in \partial C^g$ such that $|\mathscr{C}^g(p^g)| = 1$. These points have no containing contour on one of the slices. For a point $p^g \in U^g$, we call its single containing green contour the *penumbral contour* $\mathscr{P}(p^g)$. In Figure 2.3, $\mathscr{P}(p_3) = c_2$ and the penumbral contour for the other two points is undefined.

As it happens, points in S^g cannot lie on the inside of both contours, but must lie on the boundary of at least one, but this distinction is not necessary for our analysis here.

Definition 2.3.3. Suppose $p^g \in U^g$. Then $\mathscr{Z}(p^g)$ is the z coordinate value for $\mathscr{P}(p^g)$. We call this the *z-home* value for p^g. $\mathscr{Z}(q^g)$ is undefined for all

$q^g \notin U^g$.

Definition 2.3.4. Consider a sphere of radius δ centered at a point p. Consider also the cylinder of radius δ about the vertical line segment from p to p' where p' is the projection of p onto the slice at $\mathscr{Z}(p)$. The union of the open sphere and open cylinder is called the buffer region $\mathscr{B}(p)$ of p.

Definition 2.3.5. Point $p^g \in U^g$ is called a *conflict point* if there is some point $p^y \in U^y$ such that $p^y \in \mathscr{B}(p^g)$.

Conflict points are points at which two components are closer than δ or at which some point p^y is inside or on the surface of two components.

Theorem 1. *Let $p^g \in \partial C^g$ be a point conflicting with at least one point on another component C^y. Then for all components C^i such that $i \neq g$ and $i \neq y$, $C^i \cap \mathscr{B}(p^g) = \emptyset$.*

Theorem 1 shows that a point p can conflict with points of only one other component. This is important because it frees us to deal with only component pairs. That is, we are guaranteed that if component A and component B intersect, there is no other component C that intersects in the same region, even though C may intersect A and/or B elsewhere. In other words, there are no triple intersections. This naturally leads to the algorithm described in Section 2.4, in which conflict points are found and dealt with between pairs of components.

Theorem 2. *Two components C^g and C^y are within δ distance of each other if and only if there is at least one conflict point on the surface of either component.*

Corollary 1. *If two components C^g and C^y intersect then there is at least one conflict point on the surface of either component.*

By this theorem and corollary we see that removing all conflict points between two components is necessary and sufficient to guarantee that the components are not intersecting and are separated at every point by at least δ.

Theorem 3. *Moving any point $p^g \in U^g$ in the direction of $\mathscr{Z}(p^g)$ will not generate any additional conflict points among any pair of components.*

This theorem is important to justify removing conflict points between pairs of components. If it wasn't so then the algorithm would be computationally far more complex as we would have to continuously check previously resolved components for additional conflict points after any modification is made in the region.

Theorem 4. *Conflict points exist on the triangulated surfaces of two components if and only if at least one conflict point exists either at a triangle vertex or along a triangle edge of either component.*

2.4 Implementation

Our algorithm removes conflict points in a manner conforming to theorem 3, i.e., no additional conflict points are introduced at any step. And since removing all conflict points at locations defined in theorem 4 effectively removes all conflict points, our algorithm is bounded by the number of those locations.

The algorithm removes conflict points between pairs of components. We rely on theorem 3 to justify working with only two components at a time: resolving conflict points between two components will not increase the number of conflict points among any other pair of components.

In addition to working with only two components at a time, we also deal only with reconstructions between two given slices at a time. This is appropriate given that inter-slice interpolation is linear. There are no component intersections on the slices themselves (by criteria 3 and 4), and the algorithm does not modify any points on the slices. So the contours on the slices act as natural boundary points between intra-slice reconstructions, even after intersection removal.

The algorithm is as follows:

1. Separate contours in slices z_i and z_{i+1}.
2. Run single component tiling on each component in slices.
3. Determine conflict points.
4. Trace cut from conflict point to its exit in a triangle.

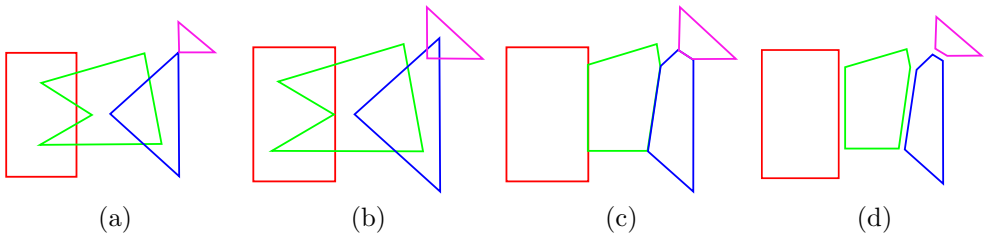


Figure 2.5: Contour intersection removal. (a) shows the original contours and (b) shows the contours after dilation by $\delta/2$. In (c) the dilated contours have been clipped and (d) shows the final result after erosion.

5. Triangulate (planar) polygons.
6. Adjust z values.

We now describe each step in detail.

2.4.1 Slice contour separation

Our 2D contour intersection removal algorithm is a simple and efficient algorithm aimed at separating contours by a value δ (see Figure 2.5). All contours in a given slice are first dilated by $\delta/2$ after which proper intersections between contours (proper, in this case, meaning intersections such that a segment from one contour touches both the interior and exterior of another contour) are found using a sweep line and marked. We are guaranteed that there are an even number of intersections as only proper intersections are marked. Intersection points are paired up such that the mid-point of the line segment defined between two intersections is in the interior of two contours. For each of these pairs, the intersecting contours are clipped along this line segment. The contours are then eroded back to roughly their original shape

minus the clipped areas.

This approach is simple and fast, but it does have its shortcomings. For one, it doesn't support intersections of more than two contours. This is, of course, possible, but generally doesn't happen often in the data we have dealt with. Another side effect is a smoothing of the contours, which is dependent on δ. In our case this is actually desirable, as the contours we generally deal with are rather noisy, which is why we have to perform the intersection removal in the first place.

2.4.2 Single component reconstruction

The single component reconstruction algorithm we use is adapted from [11]. It takes planar contours in adjacent slices as input and outputs a series of triangles, or "tiles", forming a surface between the contours. It supports branching and conforms to all of the criteria as they apply to single component reconstructions. The algorithm proceeds roughly as follows:

Given contours in adjacent slices,

1. find all contours with the same object labels
2. of these contours, find contours that correspond to (overlap with) each other
3. determine penumbral regions of corresponding contours
4. construct tiling between contours in penumbral regions (Figure 2.6a)
5. construct tiling in untiled regions (Figure 2.6b)

We defer to [11] for in-depth description of the algorithm except for three degenerate cases of tiling and the final step, where untiled regions are resolved.

There are three tiling cases that we detect and handle in our algorithm that aren't discussed in the original paper. These are degeneracies that we have found to occur in our data that we handle as special cases in order to make the tiling algorithm more robust with resorting to ϵ-perturbation. We discuss them informally and sketch our solutions. The first occurs when a chord is proposed as shown in Figure 2.7a. Here the projection of a chord (a, b) intersects with the projection of a vertex c in another contour. This chord is legal according to the original theorems, but can cause problematic tilings, *e.g.* if a chord (c, b) is proposed and accepted then criterion 2 will be violated. We detect this case and consider the chord (a, b) illegal.

The second case occurs when edges from adjacent contours overlap along a segment. Consider directional arrows on each contour traveling counter-clockwise. In the overlapping case shown in Figure 2.7b the arrows along the overlapping segment will be pointing different directions. The nature of this problem is such that it is difficult to solve in the context of the overall algorithm. As it happens rarely, our algorithm reports when it occurs and the regional tiling is corrected manually.

The third case occurs when a contour vertex a overlaps with an adjacent contour c, but both vertices adjacent to a are on the same side of the boundary of c, as in Figure 2.7c. We detect this case and treat a as if it were non-

overlapping.

We use a novel algorithm to triangulate untiled regions. Untiled regions occur when no legal slice chord can be placed between a vertex of contour c_i and a vertex of contour c_j. The approach to tiling these regions is to first approximate the medial axis of the projection of the region. We do this by decomposing the region into convex polygons. Once we have the convex decomposition, we connect the centers of each sub-polygon with the midpoints of the corresponding cut lines. See the dashed lines in Figure 2.6b. We then place the approximate medial axis into space between the slices and tile using chords from the vertices of the untiled region to the approximate medial axis line.

In the description thus far, our untiled region resolution algorithm matches that reported in [11]. But the previous approach uniformly placed the medial axis at $z = (z_i + z_{i+1})/2$. This ensures that the criteria are met (specifically criterion 3), but can cause bad triangles (see Figure 2.8a). Since the sub-polygons produced in the decomposition algorithm need meet only the criterion that they are convex, they can be arbitrarily bad, including sliver triangles and other undesirable shapes. When the centers of these sub-polygons are raised between the slices the tiling is jagged.

Ideally, medial axis vertex height should be interpolated using the vertices of the untiled regions. There are various interpolation approaches using barycentric coordinates for non-convex polygons [77, 82, 98]. In order to meet criterion 3 however, no point $v \in \Omega \setminus \partial\Omega$ where Ω is the untiled region and $\partial\Omega$

is the region boundary, can lie on z_i or z_{i+1}. Thus any barycentric approach would require that, given v, there must be at least one vertex v_j on each slice z_j such that the barycentric coordinate $\lambda_j(v) \neq 0$. A simple formulation of this requirement is $\lambda_i(v) \neq 0$ for all i and all points $v \in \Omega \setminus \partial\Omega$.

We use a simple algorithm to ensure that each point on the medial axis lies strictly between the slices. Let $\mathscr{S}$ be the union of the set of vertices of the polygon and the set of vertices of the medial axis. We compute Sibson's natural neighbor coordinates [122] for each vertex of the medial axis, such that the z-value at a medial axis vertex v is

$$v_z = \sum_{p \in \mathscr{S}} \lambda_p(v) p_z \tag{2.1}$$

We then compute v_z for every vertex v of the medial axis. At this point, at least one medial axis vertex u will be strictly in-between the slices, but many vertices may remain on the slices. But since vertices of the medial axis are neighbors of each other, we can iteratively perform interpolation and the z-value of u will propagate down the medial axis. At most n iterations, where n is the number of medial axis vertices, are required to ensure that all vertices are strictly between slices. The results are shown in 2.8b: the regions are smoothed out while still meeting criterion 3.

2.4.3 Determine conflict points

Once all components between two slices are found, we use a modified sweep line to find all tile edges of different colors that are closer than δ in

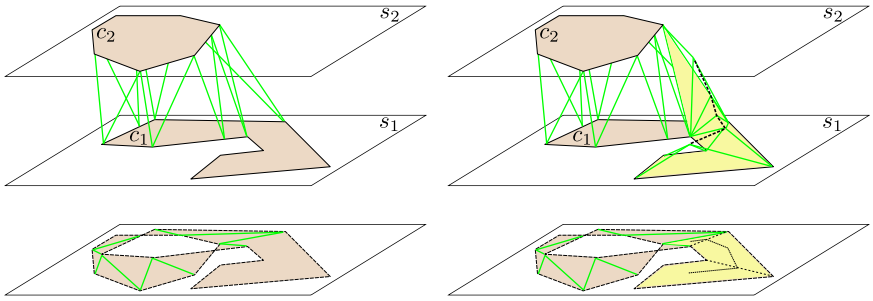


Figure 2.6: Single-component tiling algorithm. (a) shows the tiling after stage one of the algorithm. As highlighted in yellow in (b), there remains an untiled region that is then tiled by connecting contour edge segments to the medial axis of the untiled region. Our algorithm interpolates points of the medial axis to the appropriate locations between slices to avoid undesired artifacts, as shown in Figure 2.8.

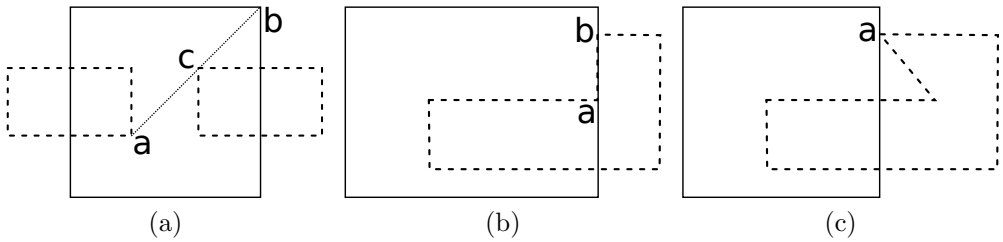
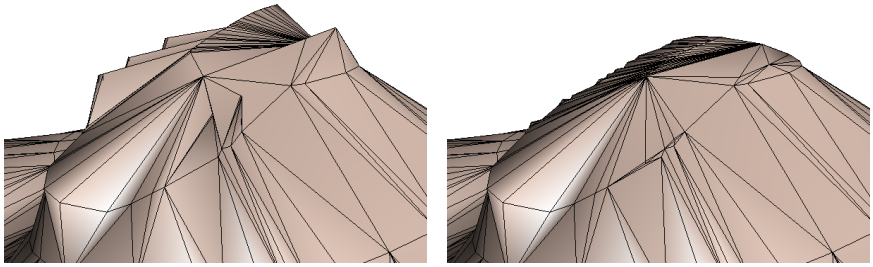
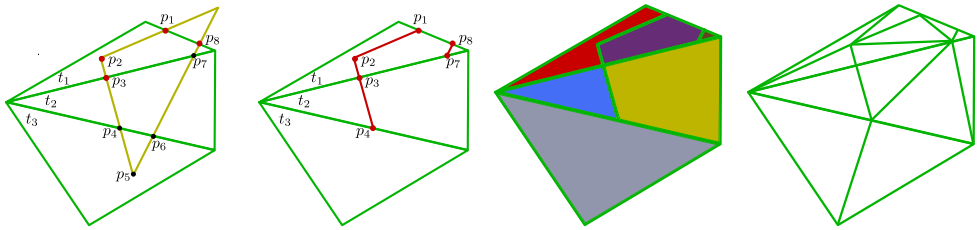


Figure 2.7: Three cases now detected and handled in augmented algorithm. The lower contour is solid while the upper contours are dashed. (a) The proposed chord (a, b) is now correctly labeled as illegal due to its intersection with vertex c. (b) No chords are legal between contours between a and b. (c) Vertex a is no longer tiled directly to the lower contour.



(a) (b)

Figure 2.8: Results of improvement to single contour reconstruction algorithm. 2.8a Shows jaggies resulting from the original algorithm placing medial axis vertices of untiled regions halfway between the two slices. 2.8b Our algorithm produces a more pleasing result by interpolating the medial axis points.



(a) (b) (c) (d)

Figure 2.9: Steps of the intersection removal algorithm. Conflict points are red while non-conflict approach points are black. (a) Conflict points on the green tile are detected. (b) Cut paths are traced. Note that cut paths occur along the yellow tile's edge and are only between two points of which at least one is a conflict point. Thus (p_3, p_4) is a cut path while (p_4, p_5) is not. (c) New polygons are induced by cut paths. The polygons are colored for clarity. (d) After triangulation of the polygons.

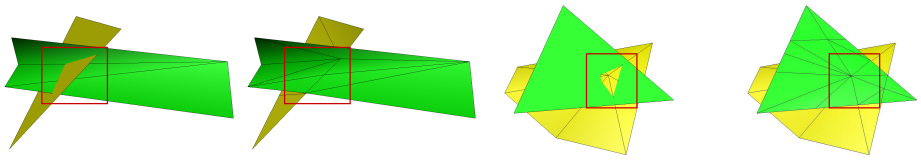
the xy plane. In addition, all tile vertices that are inside of a tile of another component (still in the xy projection of the tiles) are considered. We call these potential conflict points "approach" points. They are marked and stored in a data structure that maps the approach point to the two tiles and every edge passing through the point. There will usually be exactly two edges unless the approach point is at a tile vertex, in which case the number of edges is greater since every tile vertex touches at least two tiles.

We consider conflict points at only these approach points. We determine whether a point is conflicting or not by examining the minimum distance between the different colored edges on which the approach points lie. Then, if the minimum distance is less than δ, we mark each approach point as conflicting.

Figure 2.9(a) shows a yellow tile and three green tiles. The algorithm finds all approach points (black dots) and then determines which of these are conflict points (red dots).

2.4.4 Trace tile cuts

The algorithm for tracing tile cuts is as follows:



(a) (b)

Figure 2.10: Examples showing two interesting cases of intersection. The left figure of (a) shows a classic intersection between yellow and green tiles. The right figure shows the resolution of the intersection. The left figure of (b) shows a slightly more complicated case containing conflict points both at tile edges and at vertices. On the right is shown the resolution.

Algorithm 1: TRACE_CUTS

```
cuts := empty array of polylines
foreach point p in conflict points do
    p^y := projection of p onto yellow component
    t^y := yellow tile containing p^y
    t^g := green tile containing p^g
    polyline := empty array of points
    push p onto polyline
    dir := direction to travel on ∂t^y to go to interior of t^g
    boundary := ordered intersections on ∂t^y
    foreach approach point q^y in boundary do
        if q^y' ∈ t^g' ∪ ∂t^g' then
            q^g := projection of q^y onto green component
            q := q^g'
            push q onto polyline
        end
    end
    push polyline onto cuts
end
return cuts
```

For each conflict point, we must trace out cut polylines that we will use to induce new polygons that will then be triangulated. The way this is done is

to start at a conflict point p and find its projection onto the yellow component to get p^y (line 3). p^y will be on a yellow tile's boundary. Now follow the points on the boundary of the tile that have been marked as intersections from p^y to the exit point where the yellow tile exits the green tile (lines 8-10). Each point encountered in this trace are added to the polyline (line 14).

This can be seen visually in Figure 2.9(a). Point p_1 is a conflict point. The algorithm builds an ordered array of points following the yellow tile from p_1 all the way to point p_8. Then it loops over each point in the array checking to see if the current point p is inside of the projection of the green tile ${t_1}'$. If it is, then the point is added to the cut. So the cut from point $p_1 = [p_1, p_2, p_3]$. All cuts can be shown in red in Figure 2.9(b).

Cut polylines are specific to a tile. So cuts for the green component's tiles are first found, and then cuts for the yellow component. These polylines are superimposed onto the tiles to generate a set of induced polygons (Figure 2.9(c)). Even though the cuts are different for green tiles vs. yellow tiles, the projection of green tiles and cuts will be identical to that of the yellow.

Figure 2.9 shows an interesting case in that if the induced polygons are triangulated naively, an illegal triangulation can result. Consider point p_4 in the figure. There will be a triangle vertex at this point due to the triangulation of tile t_2. If, then, it is not a vertex in the triangulation of tile t_3, then the triangulation will not be legal. Because of this, the algorithm checks for any point on a tile edge that is involved in the triangulation of any adjacent tile, and induces that point onto all adjacent tiles. This ensures legal triangulations.

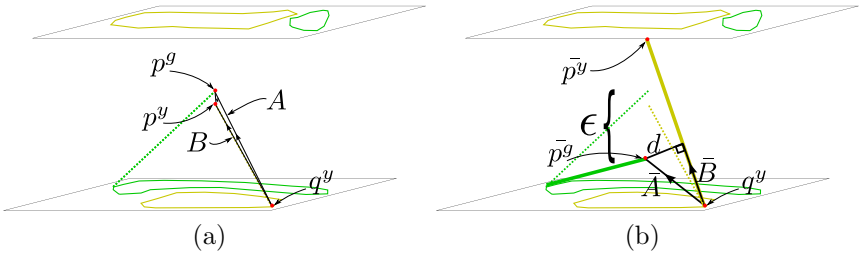


Figure 2.11: Calculation of ϵ. $\mathbf{A}$ and $\mathbf{B}$ are vectors from q^y to the original conflict points. $\bar{\mathbf{A}}$ and $\bar{\mathbf{B}}$ are vectors from q^y to the resolved conflict points. ϵ is calculated using these vectors and input minimum separation distance parameter δ.

2.4.5 Triangulate polygons

At this point we have polygons that are ready to be triangulated into a new induced surface. An intuitive approach would be to simply run a constrained Delaunay 2D triangulation on the xy projection of all induced points on the tile. This has two problems: first, the tiles can be vertical and thus the projections of the triangles are degenerate and second, inducing points on edges of triangles causes a large number of collinear points.

The first problem can be addressed by checking to see if the tile is vertical before triangulation. If it is, then rotate by 90 degrees and then triangulate. Happily, the tiles are still coplanar and so we can safely rotate the tile with induced points without worry of causing additional degeneracies.

The second problem is more troublesome. The combined problem of collinear points coupled with numerical error causing possible triangle edges outside of the original tile requires something more than a naive approach.

Our solution is to maintain a data structure mapping points to the edges of the original tile from which they were induced. Now a numerically error-prone check for collinearity is perfectly safe by simply doing a hash lookup for each point and comparing the original edges. If all three edges are the same then the points are collinear. If not, then they are not collinear, provided the original tiling algorithm returns non-degenerate tiles (which it does in our case).

We used a simple ear-cutting algorithm [110] using this data structure to ensure legal triangulations. Even with the check, an additional modification is required to ensure numerical stability. That is to first generate triangles involving at least one unused collinear point. Without this, the result often includes very long thin triangles if at least one induced point is very close to an existing vertex.

2.4.6 Adjust z values

The result of triangulating induced polygons is an induced triangulated surface, which still has conflict points between components. At this point we can adjust z values of all conflict points in the new triangulation to separate component surfaces. Each conflict point p is checked and its two associated component points p^g and p^y are given new z values as follows:

$$p_z^g = \frac{p_z^g + p_z^y}{2} + s^g \epsilon \tag{2.2}$$

where

$$s^g = \begin{cases} -1 & z^g > z^y \\ 1 & z^g < z^y \end{cases} \tag{2.3}$$

p_z^y is calculated similarly.

ϵ and δ are related but distinct. δ is the input parameter of minimum separation distance between components. ϵ is the distance along the z axis to move conflict points such that the new surfaces will be separated by δ. Determining ϵ to achieve the desired minimum distance δ between components is done as follows. A conflict point p^y is either an induced point on an edge or a vertex. Let $\mathbf{B}$ be the vector $p^y - q^y$ where q^y is either an induced point or vertex such that p_z^y is between q_z^y and $\mathscr{Z}(p^y)$. See Figure 2.11. Further, let $\mathbf{A}$ be the vector $p^g - q^y$. Now let $\bar{p^y} = \{p_x^y, p_y^y, p_z^y + s^y\epsilon\}$ and $\bar{p^g} = \{p_x^g, p_y^g, p_z^g + s^g\epsilon\}$. And lastly, $\bar{\mathbf{B}} = \bar{p^y} - q^y$ and $\bar{\mathbf{A}} = \bar{p^g} - q^y$. Distance d is

$$d = \frac{|\bar{\mathbf{A}} \times \bar{\mathbf{B}}|}{|\bar{\mathbf{B}}|} \tag{2.4}$$

Substituting for $\bar{\mathbf{A}}$ and $\bar{\mathbf{B}}$ and assuming that $s^g = 1$ we get

$$\begin{aligned} d^2 =& ((A_y(B_z - \epsilon) - (A_z + \epsilon)B_y)^2 \\ &+ ((A_z + \epsilon)B_x - A_x(B_z - \epsilon))^2 \\ &+ (A_xB_y - A_yB_x)^2)/(B_x^2 + B_y^2 + (B_z + \epsilon)^2) \end{aligned} \tag{2.5}$$

Factoring ϵ yields a quadratic:

$$\begin{aligned} 0 = \; & \epsilon^2((A_y + B_y)^2 + (A_x + B_x)^2 - d^2) \\ & +\epsilon(2)((A_x + B_x)(A_z B_x - A_x B_z) \\ & \quad - (A_y + B_y)(A_y B_z - A_z B_y) - d^2 B_z) \\ & +(A_y B_z - A_z B_y)^2 + (A_z B_x - A_x B_z)^2 \\ & +(A_x B_y - A_y B_x)^2 - d^2(B_x^2 + B_y^2 + B_z^2) \end{aligned} \tag{2.6}$$

We then substitute δ in for d and find the two roots for ϵ. It is possible that both roots yield shifts in the direction of $\mathscr{Z}(p^g)$. This occurs when the surface intersections are gross enough to cause conflict points that are more than δ apart. So we choose the solution for ϵ with the greatest value and then multiply by s^g (since we assumed that $s^g = 1$).

There is a denominator on the right hand side of (2.6) which we have omitted for brevity. But it should be clear from (2.5) that the denominator is zero only when $\bar{\mathbf{B}}$ is degenerate which cannot happen since p^y is moved in the direction of $\mathscr{Z}(p^y)$ for $\epsilon > 0$.

Theorem 5. *$\epsilon < |p^g - \mathscr{Z}(p^g)|$ and $\epsilon < |p^y - \mathscr{Z}(p^y)|$.*

This theorem bounds ϵ by the distance from the conflict points to the slices. In other words, no value of ϵ can cause a point shift in z such that the point crosses a slice boundary. This is important because any such shift would cause the reconstruction to violate criterion 3, not to mention all the criterion's dependent guarantees.

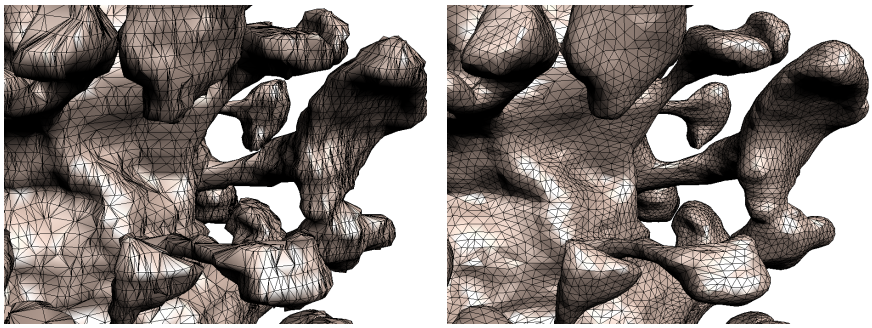
2.4.7 Computational complexity

The computational complexity of our algorithm is bounded by the number of conflict points. The maximum number of conflict points between the boundaries of two triangles is 12: 6 for the xy-plane intersections and 6 for the vertices. For m components, each with n_i triangles, the maximum number of conflict points is $12n_in_j$ where n_i and n_j are the two largest numbers of triangles in a single component. Thus, the computational complexity is $O(n^2)$ where n is the largest number of triangles in a single component.

In practice we have found that there are far fewer conflict points, even in highly tortuous datasets. Statistics of the reconstruction of which a small part is shown in Figure 2.14c is reported on line 1 of table 2.1. Between two of the slices there were 21330 total triangles before intersection removal. Among these triangles there were 7691 detected conflict points. Our testing of our most tightly-packed data showed similar ratios of number of triangles to number of conflict points.

2.4.8 Smoothing

One additional step in our pipeline is that of smoothing the surfaces. Initial surface reconstruction can introduce numerically-troublesome thin triangles, and the intersection removal adds $O(n^2)$ triangles. So at completion of intersection removal we run the surfaces through our quality improvement pipeline, which includes edge contraction using the QSlim software package [60] after which we decimate [149] and improve triangles [16]. These tasks are



(a) (b)

Figure 2.12: (a) An apical dendrite before smoothing. (b) After smoothing. The number of triangles composing the final, smoothed surface is a parameter. In this example the number of triangles was cut to roughly half the original number.

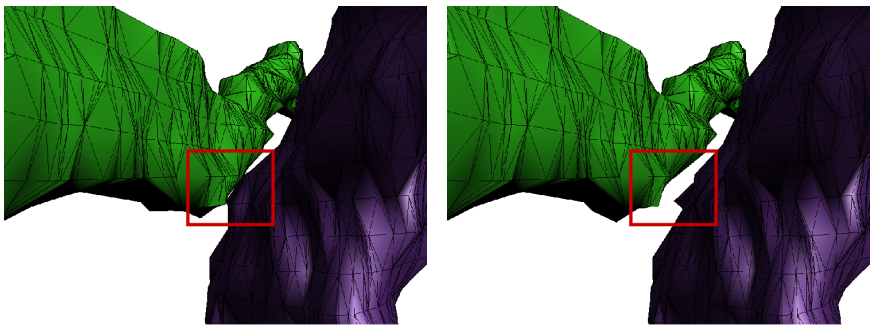
done using a library version of our Level Set Boundary Interior and Exterior Mesher [38] which is also embedded in our Volume Rover software package [39].

2.5 Results

We have implemented both the single component contour tiler and intersection removal algorithm.

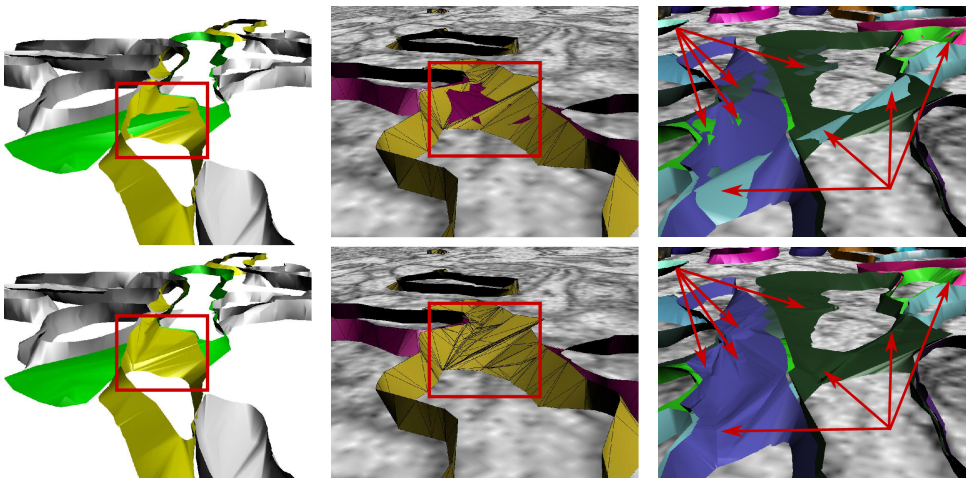
Figure 2.12 zooms in on a dendrite to show the effect of our smoothing algorithm, and Figure 2.13 shows the effects of varying the separation distance δ parameter.

Figure 2.10 shows some interesting cases of intersection between tiles (though it is by no means exhaustive). Figure 2.10a shows intersection of a



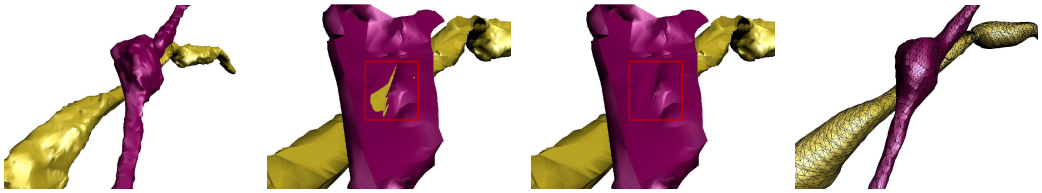
(a) (b)

Figure 2.13: Shows the effects of varying the separation distance δ parameter when reconstructing two axons that come very close in one region. (a) Separation $\delta = 0$. (b) $\delta = 40$ nm. Note that the surfaces are changed only in the region of close approach.



(a) (b) (c)

Figure 2.14: Results of running intersection removal on various portions of neuronal contour data. (a) Before and after intersection removal at branch point. (b) Result of intersection removal is shown on top of the original ssTEM data. (c) Shows prevalence of intersections. This small portion of the data alone has at least eight component intersections.



(a) (b) (c) (d)

Figure 2.15: Results of running intersection removal on two axons that intersect. (a) Two axons whose reconstructions intersect between slices. (b) Zoomed in with part of the top axon cut away to reveal the intersection. (c) Result of intersection removal. (d) After smoothing.

dataset	slices	tiling time	num triangles	num intersects	intersect removal	num triangles
CA1 (axons)	115-116	79.2s	21330	7691	85.1s	48778
CA1 (all)	61-62	503.3s	37849	26965	759.7s	105434
CA3	150-151	90.9s	9812	52	13.5s	10078

Table 2.1: Table of tiling timing and triangle statistics. Tiling time includes 2D contour curation and single contouring. Tests were performed on a Linux Kubuntu workstation with an Intel Xeon quad core CPU at 3.20 GHz with 4 GB memory. The CA1 dataset (Figure 2.1) was taken from the hippocampal region of the brain and has 452 axons and about 50 dendrites. The CA3 dataset is unreleased.

vertical tile. This requires the tile to be rotated by 90 degrees before being re-triangulated after new points are induced. 2.10b shows conflict points at only tile vertices and not xy-plane intersections. As can be seen this case is handled since conflict points are checked at tile vertices in addition to xy-plane intersections.

Figure 2.14 shows results of intersection removal from reconstructions of the hippocampal region of the brain. The contours used were hand-traced from 4K x 4K pixel resolution ssTEM images. Image pixels are approximately 2 nm square and inter-slice spacing is 45 nm. We show results from various regions of the dataset at different slices and using different components. The intersection removal algorithm is run immediately after all components between two slices are reconstructed. Only one pass over conflict points is made and, as can be seen, no additional conflict points are generated. These results show a number of interesting things: 1) Every intersection except for one in Figure 2.14c occur where one or both components is branching. This highlights the power of handling branching cases smoothly. 2) Figure 2.14b shows the mesh triangles and it is clear that new triangles are only induced from original triangles. It also shows that are large number of triangles are generated in intersection removal. 3) Figure 2.14c is striking in that it shows just how prevalent these intersections can be when using a linear interpolation reconstruction approach on tightly-packed anisotropic data. In that small region of the data (three slices and approximately 5 μ^2) there are more than eight distinct intersecting regions, some of them grevious. We emphasize that any interpolatory single-component

reconstruction method will run into this problem. Our algorithm detects and removes every intersection while maintaining original tiling criteria.

Figure 2.15 shows reconstructed data in the large. Two axons come in close proximity with each other, causing an intersection as can be seen in 2.15b after cutting away part of the red axon. The intersection is repaired in 2.15c and 2.15d shows the results of smoothing after intersection removal.

Figure 2.16 shows a complex dendritic structure with nested endoplasmic reticulum (ER). While both structures are neuronal, their geometries are vastly different, with the ER looking far more fractured than the bulbous branching of the dendrite. Our algorithm is oblivious to such varied geometries and handles each correctly.

Table 2.1 shows various statistics of our tests on neuronal data. In our most tightly-packed data (CA1), intersection removal took up roughly half the time and increased the number of total triangles to about double the original number. As noted earlier, the number of intersections was generally far less than the n^2 theoretical maximum.

An eventual goal is to reconstruct a global brain model, or even, somewhat more modestly, a reasonably large region of the brain. It turns out that both of these goals are ambitious, as the physical slicing and EM imaging process yields only very small data footprints [28]. But we are well-positioned to tackle the reconstruction challenge once the data becomes available as our algorithm will handle any arbitrary topologies and complicated geometries present

in different brain regions. Also, as discussed in Section 2.6, the algorithm will scale to support full brain reconstructions.

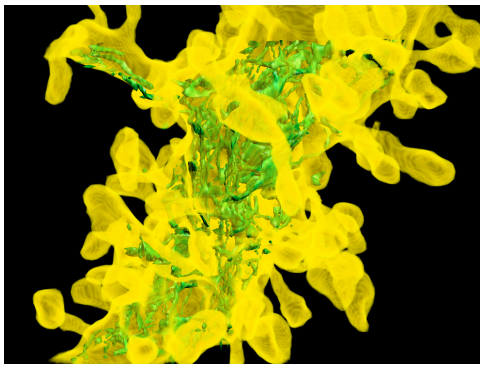


Figure 2.16: A zoomed-in view of the apical dendrite shown with transparency to reveal interior endoplasmic reticulum.

2.6 Discussion and future work

Without correct topologies and intersection-free surfaces, multiple component simulation is severely limited, and this work provides a solution to this important problem. This problem has not received significant attention due to the nature of data used in reconstructions in the past – automatic removal of intersections could be done manually because there were generally very few such intersections. But with the advent of reconstructions of tortuous, densely packed and anisotropic data, the importance of an automatic and robust method has increased. This chapter presented just such a method. Using single component reconstructions that adhere to certain guidelines, the method presented in this chapter can remove intersections between components correctly and robustly, preserving the guarantees of the single-

component reconstruction method: the output surface is water-tight, and any cross-section through an original slice yields precisely the input contours. It also guarantees a minimum separation distance between all components. This is important when dealing with *e.g.* neuronal data, as a rough idea of the average distance between components is generally well known.

The algorithm is efficient, performing the intersection removal in roughly the same amount of time as the original reconstruction by using efficient 2D geometric calculations. The algorithm is also scalable in the number of slices, since it reconstructs and resolves intersections slice-by-slice. Scalability in the number and complexity of components is slightly less straightforward, as the current implementation stores in memory all components between two slices. While this has not caused any problems in practice, it is possible that a very large number of components would not fit in computer memory. In this case, a simple heuristic could partition components into overlapping regions and work with them region-by-region.

Moving intersection points only in the z-axis enables proofs of correctness and solves the intersection problem, but it also restricts the algorithm. Enabling movement in the x-y plane may make it possible to produce smoother surfaces and better-behaved triangles from the intersection removal process. We are interested in determining whether the same correctness guarantees can be made while allowing conflict point movement in any direction. Of course, our smoothing process greatly improves any poor triangulations, but our current smoothing algorithm, which uses geometric flow, does not respect inter-

component spacing restrictions. A constrained smoothing algorithm would be beneficial.

Our algorithm does produce a large number of triangles in the process of removing intersections, and some of these triangles are poorly shaped. However, we have shown that our smoothing software is highly effective at transforming the surfaces into triangulations suitable for visualization and analysis. One question that needs to be addressed is how much does smoothing affect the minimum separation distance guaranteed by the intersection removal algorithm. We are interested in finding a quantitative measure and whether the error introduced is acceptable.

Another improvement that needs to be made is the removal of criterion 5, that a contour cannot be nested inside a different colored contour. This criterion prohibits intersection removal between nested components. In the case of neuronal modeling, so-called intracellular components such as endoplasmic reticulum and mitochondria are treated separately. But to get a truly accurate model, these will need to be included.

A fundamental issue still exists with criterion 2, which states that a vertical line cannot pass through more than one of a component's boundaries between any two given slices. This means that with very high anisotropy, an oblique component may be disconnected because its contours that in reality correspond may not be labeled as corresponding in the algorithm. As this criterion is a basis for many of our theorems and guarantees, an approach to resolving it may be to add a special case.

There are a number of interesting generalizations that may be possible in the framework of these methods. One is support of incomplete contours, which could generate either open polyhedra in the unknown regions, or closed polyhedra, thereby closing the original contour.

Another generalization is support for non-parallel slices. This would require re-visiting the fundamental theoretical guarantees of the single-component algorithm. The follow-up question would be whether the intersection removal algorithm could enjoy versions of the same guarantees our current parallel-slice version does, *e.g.*, no more than two components can intersect in the same region, all intersection points can be resolved by resolving a finite number of intersections, *etc.*

The contribution described in this chapter is part of a larger effort to build "analysis-ready" surface reconstructions, that is, geometric models that are water-tight and intersection free, among other properties. As shown, reconstructions using the method described here satisfy the two mentioned properties.

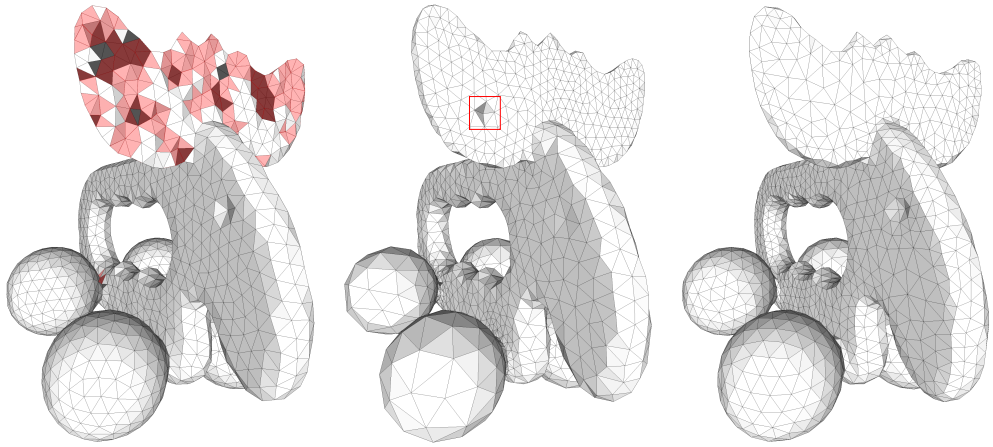
Chapter 3

Surface Segmentation for Improved Remeshing

This chapter presents a novel remeshing algorithm based on Centroidal Voronoi Tessellation (CVT). CVT can result in non-manifold vertices and edges in surface meshes that have areas of low curvature and low local feature size. Our processing pipeline is to first segment surface S into subsurfaces $\{M_i\}$. We then remesh each M_i individually using CVT, followed by stitching. Organization of this chapter is as follows. After motivating the problem (Section 3.1) and a discussion of related work (Section 3.2), we briefly describe the CVT algorithm and theorems related to topological correctness of remeshed surfaces (Section 3.3). We then discuss the curvature dominance property and surface segmentation (Section 3.4) followed by stitching (Section 3.5). We end with results and conclusions (Section 3.6).

3.1 Introduction

Surface remeshing is the process of transforming an input surface mesh S into an output surface mesh W. Often a given model needs to be remeshed to meet the needs of a given application. For example, the number of triangles



(a) (b) (c)

Figure 3.1: Remeshing the Toy Elk model using 2000 sample points. Non-manifold edges and vertices are highlighted in red. (a) Uniform CVT. A shortage of triangles in the horns results in topological errors. (b) *lfs* CVT. Despite a large number of triangles in the horn area there is still one non-manifold vertex. (c) Our method, κCVT. Our method produces improved meshes by distributing samples according to curvature rather than local feature size while avoiding topological errors.

may be too large for a graphics application to render efficiently, or low triangle quality may cause numerical instability in a FEM simulation. In the case of our neuronal modeling project, triangle quality needs to be high for good simulation results, and the number of triangles needs to be low in order for the simulation to fit into computer memory and complete in a reasonable amount of time. Triangle quality improvement and reducing the number of triangles (decimation) are two important, but often competing, goals in remeshing.

Many remeshing methods sample and optimize points directly on S and then use a triangulation of these points to produce W. These algorithms run into trouble when the number of sample points is very low. Not only does the geometric approximation suffer, but topological errors can be introduced, in the sense that W is not homeomorphic to S, and also that W may not be 2-manifold.

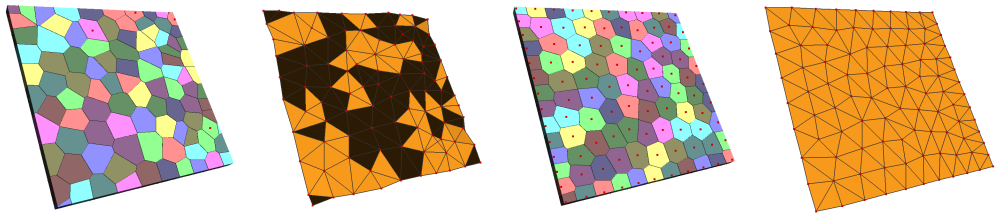
In areas where the surface has high curvature this phenomenon makes intuitive sense. But what can be at first surprising is that in many cases areas of very low curvature require an inordinately large number of triangles to approximate faithfully. This occurs when other sections of the surface, even if distant geodesically, come in close proximity to the flat region. The fact that topological and geometric errors increase with fewer triangles in such areas is an unfortunate side-effect of using the euclidean metric to approximate geodesic distances. One way to look at this is in terms of the local feature size (*lfs*), which is defined as the distance from a point $p \in S$ to the closest point on the medial axis of S. The r-sampling theorem, discussed further in

Section 3.3, states that the number of samples needed to ensure that W is homeomorphic to S is dependent on *lfs*. Thus, even if an area is nearly planar, large numbers of triangles will be needed if the *lfs* is low.

The effects are felt beyond these areas of low curvature. Since a large number of samples may be allocated to flat areas, an insufficient number of samples may be left for areas where they are inherently needed – areas that are highly detailed. The simple solution to this problem is to add more sample points, but if keeping the number of triangles low is important then an alternative solution is desired.

We propose an algorithm that largely removes the requirement for dense sampling in featurless areas. This is done through a surface segmentation algorithm that decomposes S into a set of subsurfaces $M = \{M_i\}$ such that for any $p \in M_i$, $lfs(p) \approx 1/\kappa(p)$ where κ is the maximum of the absolute values of the principal curvatures κ_1, κ_2 at p. We call a surface with this property "curvature dominant." Such surfaces with this property can be remeshed with enough samples to preserve features with high curvature, while flattish areas can be approximated with fewer triangles without risk of topological errors. Our remeshing of each individual subregion is done using CVT with density function $\rho = \sqrt{\kappa}$. Hence the moniker κCVT.

With remeshed subregions in place, our stitching algorithm composes them back into a single triangulation. The stitching algorithm uses information on connectivity between regions as a heuristic for accurately finding correspondences between triangles of different subregions.



(a) (b) (c) (d)

Figure 3.2: Examples of remeshing a thin box. (a) Voronoi diagram of seeds after running CVT to convergence. The Voronoi cells are badly shaped because seeds are influencing cells on the opposite side of the box. Seeds are shown in red. Most of the seeds have drifted inside the box and are not visible. (b) Dual of the Voronoi diagram. Triangles in black are facing away from the viewer – not only are many of the triangles poorly shaped, but there are topological errors as well. (c) Half of the box has been removed. With a single sheet the Voronoi cells are as expected: fairly regular hexagons. (d) The Dual has well-shaped triangles and no topological errors.

In datasets with flattish regions as described, our method requires fewer triangles to produce topologically correct meshes that are geometrically very accurate.

3.2 Related work

Much work has been done in the area of surface remeshing. Some remeshing approaches are geared toward decimation, or reduction of the number of triangles, and work directly on the mesh (e.g. [61, 75]) using a series of geometric operations such as edge collapse. These approaches typically have some error metric that is maintained and decimation halts once the metric reaches a threshold. Other similar approaches use optimization [76] that maintains topology but is computationally expensive. Another approach pro-

posed by Cheng et al. [35] uses Delaunay refinement to successfully remesh piecewise smooth meshes, including non-manifolds.

CVT is a popular remeshing technique that minimizes an energy function designed to simultaneously reproduce the input mesh faithfully while producing well-shaped triangles. A primer on CVT in its general formulation (not necessarily applied to surface remeshing) is given by Du et al. [44]. CVT has been applied to surface remeshing. Sample points are placed on the surface and their locations are optimized by minimizing the CVT energy function in order to produce quality triangles that approximate the surface well. CVT methods fall into two camps: those that optimize the points in parameter space and those that work directly on the mesh.

Of those that parametrize the surface, Alliez et al. [2] use a parameter representation of the surface as a whole. To avoid the difficulty of entire surface parametrization, various approaches parametrize locally [3, 130]. This simplifies parametrization, but returning to 3D involves stitching, and optimization is not done globally, yielding triangles that are not consistently uniform.

Another approach is to optimize sample points directly on the surface. This has the advantage of being a global optimization and triangles have been shown to be of higher quality [142]. These methods have been hampered by two issues. The first is performance. Lloyd's algorithm [103] has been the implementation of choice to minimize the CVT energy function despite its linear convergence rate [44]. Only recently was the CVT energy function shown by Liu et al. [102] to have C^2 smoothness, making it a candidate for

more efficient optimization techniques. In the same work the limited-memory BFGS method (L-BFGS) [99] was applied to the minimization of the CVT energy function with favorable results. Another work that further improved the performance of CVT is that of Yan et al. [142] which proposed an algorithm to efficiently and exactly compute the *Restricted Voronoi Diagram* (RVD), a necessary ingredient in direct methods.

The second issue with these methods is the need for high sampling rates to achieve topological correctness. While the method in [142] detects topological problems and corrects them by inserting additional samples, the requirement for large numbers of samples, possibly even in flat regions, is troublesome. Peyre et al. [112] use geodesic approximations directly, which largely obviates the sample density requirements. Their application is surface segmentation, and CVT is used to optimize two competing conditions (compactness and boundaries lying on sharp features) on surface regions. While effective for segmentation, in the context of remeshing, where the number of regions is very large, the performance is problematic. Our approach allows flat regions to be remeshed with few samples, regardless of local feature size, and does so while still using the euclidean metric.

Alliez et al. [4] provide a more thorough survey of remeshing techniques.

3.3 CVT remeshing

Our presentation of the CVT follows that given in [142].

The Voronoi Diagram, or Voronoi Tessellation, is defined as follows. Given n distinct sample, or seed, points $X = \{x_i\}_{i=1}^n$ in $\mathbb{R}^N$, each point x_i lies within a set of points

$$\Omega_i = \{x \in \mathbb{R}^N | \|x - x_i\| \leq \|x - x_j\|, \forall j \neq i\}. \tag{3.1}$$

The set of points Ω_i is called the Voronoi cell of x_i and the set of all Voronoi cells $V = \{\Omega_i\}_{i=1}^n$ is a decomposition of $\mathbb{R}^N$ and is called the Voronoi Diagram determined by X. The Centroidal Voronoi Tessellation (CVT) is a special case of the Voronoi Tessellation such that each seed $x_i \in X$ coincides with the center of mass of its corresponding Voronoi region Ω_i. Given a density function $\rho(x) > 0$, the center of mass x^* is defined as

$$x_i^* = \frac{\int_{\Omega_i} \rho(x) x \, d\sigma}{\int_{\Omega_i} \rho(x) \, d\sigma} \tag{3.2}$$

For computation purposes, the CVT can be formulated as a critical point of

$$F(X) = \sum_{i=1}^{n} \int_{\Omega_i} \rho(x) \|x - x_i\|^2 \, d\sigma \tag{3.3}$$

Equation (3.3) is known as the CVT energy function and is typically minimized using Lloyd's algorithm [103] or, more recently, the limited-memory BFGS method (L-BFGS) [99, 102].

In the case of surface remeshing, two modifications to CVT have been proposed. The first is Restricted CVT (RCVT). Given a surface $S \subset \mathbb{R}^3$, a set of seeds X, and the induced Voronoi Diagram V, the Restricted Voronoi Diagram (RVD) is the set of all restricted Voronoi cells (RVC) $\mathscr{R} = \{R_i\}_{i=1}^n$

where $R_i = \Omega_i \cap S$. In other words, each RVC is the Voronoi cell restricted to the surface S. RCVT uses a slightly modified energy function that utilizes the RVD:

$$F(X) = \sum_{i=1}^{n} \int_{R_i} \rho(x) \|x - x_i\|^2 \, d\sigma \tag{3.4}$$

Constrained CVT (CCVT) was introduced in [45] and is the same as RCVT except that the seed points are restricted to S, as given in

$$x_i^* = \arg\min_{y \in S} \int_{x \in R_i} \rho(x) \|y - x\|^2 \, d\sigma \tag{3.5}$$

The Restricted Voronoi Diagram is given by either CCVT or RCVT. The Restricted Delaunay Triangulation (RVT) is dual of the RVD, in that each vertex in the RVT corresponds to a cell in the RVD, two vertices share an edge if their corresponding RVD cells share an edge, and a triangle exists where three cells share a Voronoi vertex.

We use the efficient RVD computation given in [142] and the energy minimization given in [102]. For simplicity in writing, we use the term "CVT" to refer to the remeshing process comprising CCVT/RCVT and RVT.

3.3.1 Topological correctness

CVT does not guarantee that the output mesh W is homeomorphic to S. A theorem states a sufficient condition to avoid topological errors.

Theorem 6. *Topological ball property [46]. If each Restricted Voronoi Cell is a topological disk then the Restricted Delaunay Triangulation is homeomorphic to S.*

Figure 3.3 illustrates the topological ball property in 2D. If the RVD fails to meet this property, well-spaced sample points can be added, as implied in the figure. This solution is formalized in a theorem from the literature on surface reconstruction from point clouds.

Theorem 7. *r-sampling theorem [5]. If no point p on surface S is farther than $r \cdot \mathrm{lfs}(p)$ from a seed point $x \in X$ where r is a constant then the Restricted Delaunay Triangulation induced by X is homeomorphic to S.*

If W has topological errors, which can be detected using the topological ball property, then we simply add seed points and re-run the algorithm. This is the approach taken in [142]. Eventually there are enough seeds to meet the r-sample criterion given in theorem 7 and W will be homeomorphic to S (although termination has not been proven). Since *lfs* is based on euclidean distance, points on S that are geodesically very far from a given point $p \in S$ can cause *lfs*(p) to be small, requiring the sample density to be unduly high at p, even if the curvature at p is very low. This is the very artifact that we seek to avoid.

We briefly digress to note that our approach is intended to enable meeting the topological ball property with fewer triangles in areas of low curvature. It is not a general solution for all areas of S. To illustrate this point we mention two conditions that together are sufficient to meet the topological ball property, of which only the first condition is targeted by κCVT. Let $\mathscr{B}(p, r)$ be a ball of radius r centered at p.

Condition 1. *$\mathscr{B}(p, r) \cap S$ is a single connected component.*

Condition 2. *$\mathscr{B}(p, r) \cap S$ has a single boundary.*

Condition 1 is typically violated in areas where both *lfs* and curvature are small. Condition 2 is violated in areas where S is similar to a cylinder with small radius. In this case, $\mathscr{B}(p, r) \cap S$ may be a cylinder with open ends, which is a single connected component but has two boundaries, thus is not homeomorphic to a disk. While this can cause topological problems, there is nothing to be gained in segmentation since local feature size is already dominated by curvature.

3.3.2 Geometric accuracy and triangle quality

CVT has been shown to give excellent results in terms of mean Hausdorff error [142]. Further, the density function ρ is an intuitive way to control the trade-off between geometric accuracy and triangle quality. Meshes produced from CVT using a constant density function $\rho = c$ tend to have very well-shaped triangles of roughly uniform size. Using such uniform triangles can lead to reduced geometric accuracy in areas of high curvature, not to mention topological problems in areas of low feature size. The logical solution is to increase the number of triangles in these areas by setting $\rho = 1/lfs^2$. This produces gradation in the triangle sizes, affecting triangle quality.

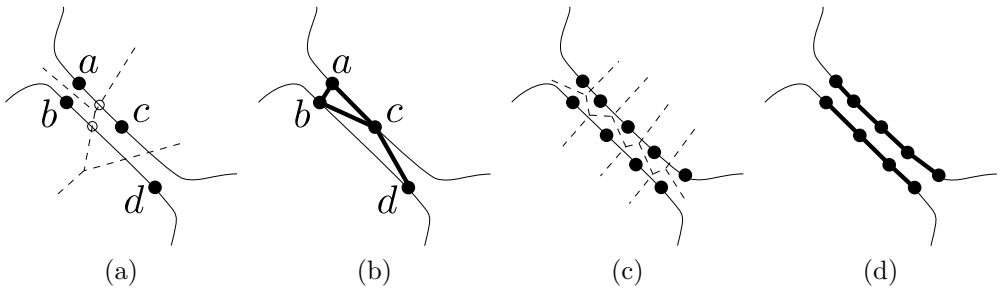


Figure 3.3: 2D graphic of the RVT. Resulting meshes are not necessarily homeomorphic to the input mesh when certain conditions are not met. (a) There are only 4 sample points (black dots) in the region of interest. The (unrestricted) Voronoi diagram is shown in dashed lines. Note that the Restricted Voronoi Cell $R_c = \Omega_c \cap S$ corresponding to point c has two connected components and thus is not a topological disk. Shared "edges" between samples a and c and samples b and c are circled. These induce edges between cells in the triangulation. (b) RDT induced by RVD. Edges between sample points (bold lines) are used to reconstruct the surface. Because samples are not dense enough, the resulting surface is not homeomorphic to the input. (c) Voronoi diagram of densely-sampled points. All RVCs are topological disks. (d) Resulting surface is homeomorphic.

3.4 Surface segmentation

If S has flat regions with low *lfs* there are two options when remeshing with CVT. The first is to use a constant density function $\rho = c$ and risk topological errors. The second option is to use $\rho = 1/lfs^2$ which may produce large numbers of triangles in those regions rather than allocating them to regions of high curvature. A surface that is curvature dominant, however, can use curvature as the density function. This ensures that few triangles will be used in flat regions while still maintaining good likelihood that the topological ball property will be met. We approximate curvature using CGAL's implementation of the approach in [34]. Segmentation requires local feature size for the vertices of S which we approximate using the approach given in [1] by computing the distance to the nearest pole [5].

We first fix some notation. If we let A and C denote closed, adjacent triangles in a 2-manifold triangulated surface S, then $A \cap C$ is the shared edge between them, which we denote AC. Let V_A be the set of three vertices defining triangle A. Further, let $d(p, \Delta) = \arg\min_{q \in \Delta} \|p - q\|$ be the minimum distance between p and 1- or 2-simplex Δ. We define $r_p = 2 \cdot \alpha \cdot lfs(p)$ and $r_A = \arg\min_{v_i \in V_A} r_{v_i}$. All of our experiments use $\alpha = 1.1$. Let $P_{A,\Delta}$ be the set of all points $p \in A$ that are within r_A of the simplex Δ. That is, $P_{A,\Delta} = \{p \in A | d(p, \Delta) < r_A\}$. We note that for any $p \in P_{A,\Delta}, \mathscr{B}(p, r_A) \cap \Delta \neq \emptyset$. See Figure 3.7. Let $lfs_S(p)$ be the local feature size at p with respect to surface S.

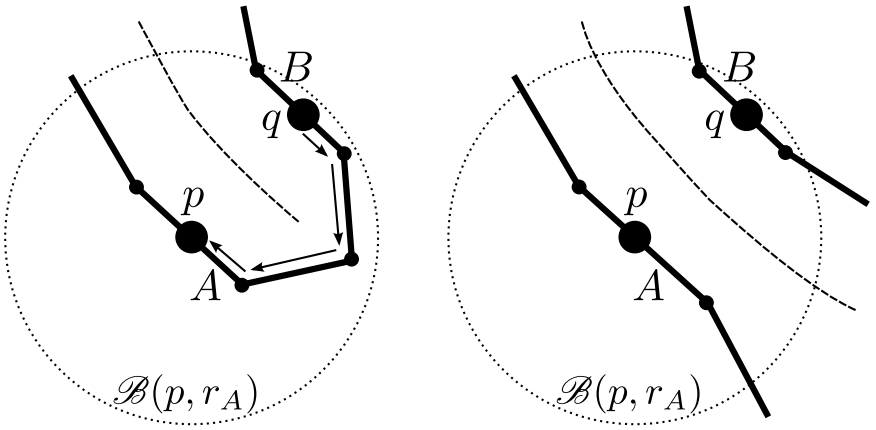
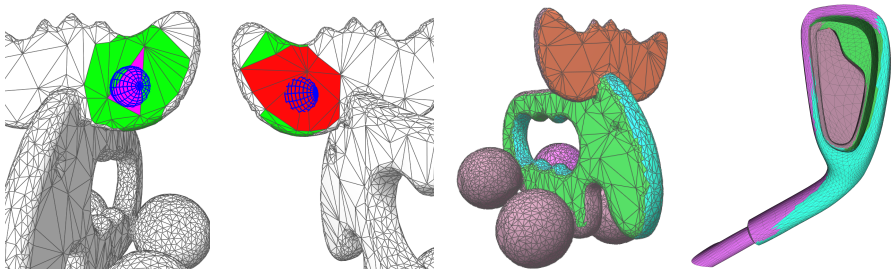


Figure 3.4: 2D graphic illustrating use of lfs_S to find incompatible triangles. (a) A and B are compatible. For any $p \in A$, $q \in \mathscr{B}(p, r_A) \cap B$ there is a path from q to p, similar to the path shown along the arrows. The path lies entirely in $\mathscr{B}(p, r_A) \cap S$. (b) A and B are incompatible. There is no path from q to p.

Compatibility table Our algorithm partitions a surface triangulation S into subsurfaces $M = \{M_i\}$ such that the ball $\mathscr{B}(p, r_A)$ centered at any point $p \in A \in M_i$ will yield a single connected component when intersected with M_i (Figure 3.4). The first step in our algorithm is to build a table of all "incompatible" pairs of triangles in S (Figure 3.5). Triangles $A \in S$ and $B \in S$ are incompatible if there exists any pair of points $p \in A$ and $q \in \mathscr{B}(p, r_A) \cap B$ such that there is no path from q to p residing entirely in $\mathscr{B}(p, r_A) \cap S$. (Note that a surface P is a single connected component *iff* for any two points $p, q \in P$ there exists a path $\Gamma_{q,p}$ from q to p such that $\Gamma_{q,p} \subset P$.) In this case, both A and B cannot exist in the same subsurface M_i without violating condition 1.

To build the compatibility table, we first construct an R-tree [64] with



(a) (b) (c) (d)

Figure 3.5: Identification of incompatible triangles. (a) The triangle of interest A is in magenta. The blue ball has radius r_A. Triangles compatible with A are shown in green. Any ball $\mathscr{B}(p \in A, r_A)$ intersected with the compatible triangles will yield a single connected component. (b) Rotated to see the opposite sheet near A. Triangles within a distance r_A of A that are incompatible with A are shown in red. (c), (d) Two examples of final segmentation. All triangles of the same color are compatible with each other.

all triangles in S. We then iterate through each triangle $A \in S$. Given A, we find, using the R-tree, the set of all triangles $B = \{B_i\}$ such that each B_i is within r_A of A (i.e. $\exists p \in A, q \in B_i$ s.t. $\|q - p\| < r_A$). We then compute P_{A,B_i}. We do this in three steps. First we construct a prism $Z = \{q | q' \in B_i, \|q - q'\| \leq r_A\}$ around B_i (Figure 3.6) where q' is the projection of q onto the plane defined by B_i. In other words, we sweep B_i in its normal direction by $-r_A$ to r_A. We then construct a set of three cylinders $X = \{X_{\{123\}}\}$ of radius r_A around each edge of B_i. Finally we construct a set of three spheres $Y = \{Y_{\{123\}}\}$ of radius r_A around each vertex of B_i. We can now compute $P_{A,B_i} = A \cap (Z \cup \bigcup_{X_i \in X} X_i \cup \bigcup_{Y_j \in Y} Y_j)$ and $P_{A,B_iC} = A \cap (X_j \cup Y_k \cup Y_l)$ where C is an edge-neighbor of B_i and X_j (Y_k, Y_l resp.) their shared edge (vertices). Since B_i is within r_A of A, $P_{A,B_i} \neq \emptyset$. For every point $p \in P_{A,B_i}$ and every

point $q \in \mathscr{B}(p, r_A) \cap B_i$ there must exist a path $\Gamma_{q,p} \subset \mathscr{B}(p, r_A) \cap S$. Let $\mathscr{N}_{B_i}$ be the edge-neighbors of B_i.

Our algorithm uses a breadth-first search from A over all $B_i \in B$, flagging each B_i as compatible or incompatible. A and B_i are incompatible if $\bigcup_{N_j \in \mathscr{N}_{B_i}} P_{A,N_jB_i} \setminus P_{A,B_i} \neq \emptyset$. See Figure 3.7. Each set of points $P_{A,\Delta}$ is the union of ellipses and polygons (Figure 3.6). Using these primitives directly makes for very expensive boolean set computations, so our implementation converts $P_{A,\Delta}$ to a raster representation so that less expensive bitwise boolean operations can be used.

Region merging Let Λ_i be a set of triangles. With the compatibility table built, we define a binary predicate operator $\Lambda_i \oplus \Lambda_j$ that yields true iff every triangle in Λ_i is compatible with every triangle in Λ_j. Segmentation proceeds as a region merge algorithm.

Algorithm 2: Segment via region merging

```
   Data: triangles S = {T_i}
   Result: regions M = {M_i}
1  Initialize one region Λ_i per triangle T_i
2  foreach region Λ_i do
3  |  foreach neighbor Λ̄_j of Λ_i do
4  |  |  if Λ̄_j ⊕ Λ_i then
5  |  |  |  merge Λ̄_j and Λ_i
6  |  |  end
7  |  end
8  end
9  Repeat lines 2-8 until no regions are merged
```

Region merging outputs a set of subsurfaces $M = \{M_i\}$. CVT with

density function $\rho = \sqrt{\kappa_S}$ is applied to each M_i. $\kappa_S(p)$ is the curvature at point p with respect to surface S. It is important to use the curvatures from the original surface, since edges with high curvature in S may be boundary edges in M_i with very low curvature. Even though the new boundary is flattish, we still desire high density there so that the stitching algorithm has border edges that have roughly the same spacing between two subsurfaces.

Remeshing With the surface S segmented into subsurfaces we now remesh subsurfaces individually using CVT with $\rho = \sqrt{\kappa}$. The N samples must be distributed among the different subsurfaces for initial placement and optimization. We allocate N_i seeds to each subsurface M_i where

$$N_i = N \int_{M_i} \rho(x)\, d\sigma / \int_S \rho(x)\, d\sigma$$

One subtlety in the remeshing stage is that we must ensure that adjacent boundaries of subsurfaces have compatible sample density so that subsequent stitching is uniform. For this reason, our density function uses the curvature computed with respect to the original surface S rather than with respect to a subsurface M_i.

In some cases, CVT remeshing of a segmented region yields a vertex that is shared by only two triangles such that the edges are not orderable. This is automatically detected and repaired by either breaking the triangles apart or adding two new triangles that connect them.

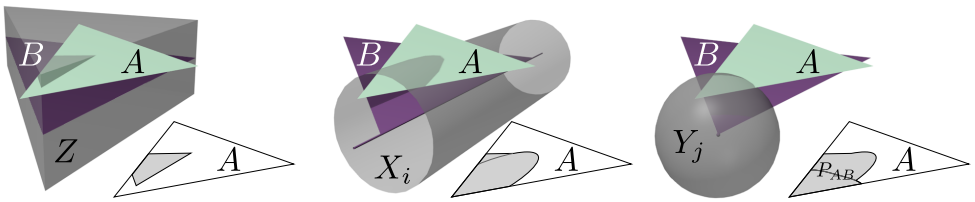


Figure 3.6: Construction of P_{AB}. Triangle A is in green and triangle B is in purple. (a) Intersection of A with prism Z. The shaded portion of the 2D graphic is the intersection restricted to A. (b) Intersection of A with cylinder X_i. The other cylinders are not shown. (c) Intersection of A with sphere Y_j. The other spheres are not shown. The 2D graphic shows P_{AB}.

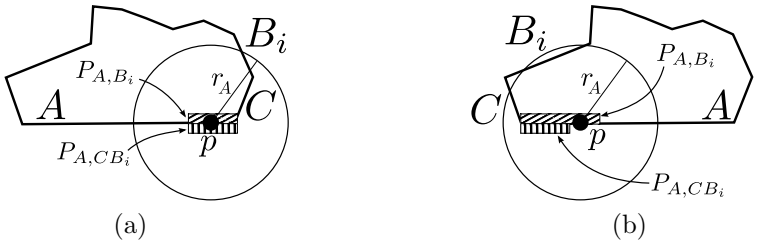


Figure 3.7: 2D illustration of $P_{A,\Delta}$. (a) In this case, given that C is compatible with A, $P_{A,B_i} \setminus P_{A,CB_i} = \emptyset$, so A and B_i are compatible. That is, $\mathscr{B}(p \in A, r_A) \cap S$ is a single connected component. (b) Given C is compatible with A, $P_{A,B_i} \setminus P_{A,CB_i} \neq \emptyset$, so A and B_i are incompatible. $\mathscr{B}(p \in A, r_A) \cap S$ yields two connected components.

3.5 Stitching

The segmentation algorithm described in Section 3.4 yields a set of subsurfaces $\{M_i\}$. After segmentation, but before remeshing, we build a table JUNCTIONS of n-way surface boundary intersections. If a vertex is shared by multiple subsurfaces, an entry is made in the table listing the index of each subsurface. After remeshing, we consult JUNCTIONS to find subsurface correspondences. Suppose an entry of the table consists of 3 subsurfaces M_i, M_j, and M_k. We first search to find a "connector triangle" t_c such that t_c shares a vertex with each of the three subsurfaces and is optimal in some sense. See Figure 3.8. Let E_i be the boundary edges of M_i and let V_i be the vertices in E_i. We iterate through each vertex of one set of edges. Let V_j^p (resp. V_k^p) be the closest n_α vertices in E_j (E_k) to p in terms of euclidean distance. The set of all candidate connector triangles is $\{(p \in V_i, q \in V_j^p, r \in V_k^p)\}$. For $n_\alpha = 4$ (the value used in our experiments) there are then $16 \cdot |V_i|$ candidate connectors.

For each candidate connector triangle t_c we stitch a distance of n_β triangles in each direction (in experiments we used $n_\beta = 5$, but we use $n_\beta = 2$ in the figure for clarity). Let T_c be the set composed of t_c and the $3 \cdot n_\beta$ neighborhood stitch triangles. We assign a score to t_c using the cost function

$$cost(t_c) = \sum_{t \in T_c} area(t) \cdot Q(t)^{-\gamma}. \tag{3.6}$$

γ is a user-defined parameter (we used $\gamma = 0.5$) and $Q(t)$ is the triangle quality

measure [56]

$$Q_t = \frac{6}{\sqrt{3}} \frac{r_t}{h_t} \tag{3.7}$$

where r_t and h_t are the inradius and longest edge length of t, respectively. Once all candidate triangles are scored we choose the one with the lowest score. This gives an approximately optimal triangle according to our cost measure that connects three surfaces. Once we've found connecting triangles for each entry in JUNCTIONS we simply search out from one such triangle, adding triangles along the way until we reach another connecting triangle.

3.6 Experimental results and conclusions

We compare our results with direct CVT methods and the method proposed by Fuhrmann et al. [59] which samples points directly on the mesh and then optimizes their placement in local parameter domains. The latter method isn't well-suited for aggressive decimation (the triangulation of initial samples fails), so we only report its results for higher sampling rates. We used 100 Lloyd iterations and used a density contrast exponent of 0.5 with no laplacian smoothing. In this discussion we refer to unsegmented CVT using the density function $\rho = 1$ as *uniform* CVT and unsegmented CVT with $\rho = 1/lfs^2$ as *lfs* CVT. We refer to our method of segmenting, CVT with $\rho = \sqrt{\kappa}$, and stitching as κCVT.

Figure 3.1 uses 2000 sample points to remesh the Toy Elk model. This is far too few sample points to meet the criteria for topological correctness when using uniform CVT, and many errors result. Even the *lfs* method produces

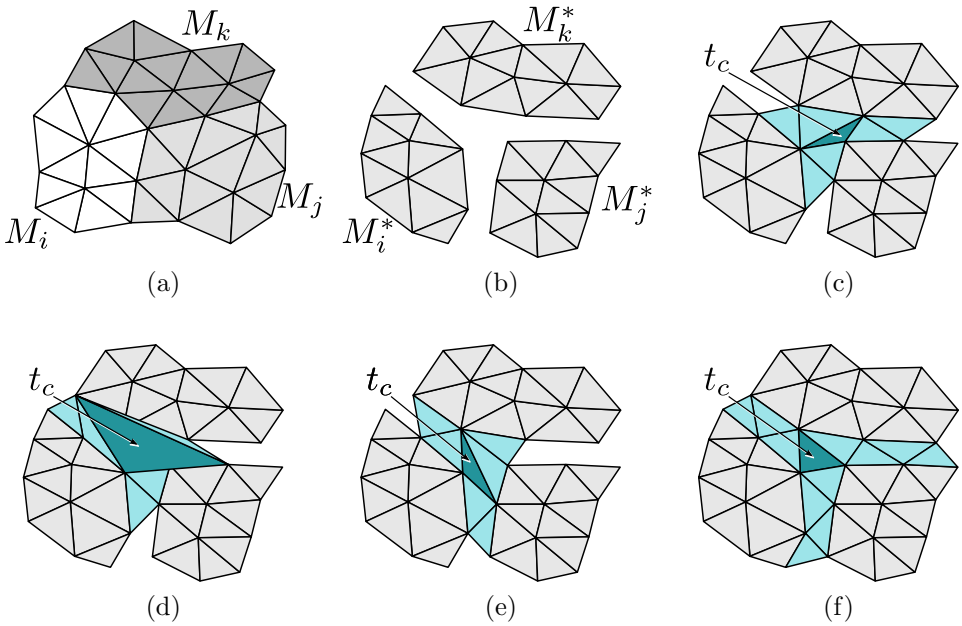


Figure 3.8: Finding connector triangles and stitching. (a) Segmented triangulation. $M_{\{ijk\}}$ are three subsurfaces in M. The table JUNCTIONS has an entry (i, j, k) since the three subsurfaces share a vertex. From the JUNCTIONS table we see that M_i^*, M_j^*, and M_k^* need to be stitched. (b) Segmented regions $M_{\{ijk\}}^*$ after remeshing using CVT. (c), (d), (e) Three candidate connector triangles. The connector is dark cyan. Neighbor stitches up to $N_\beta = 2$ distance in each direction are light cyan. (f) The connector triangle which, along with associated neighbor stitches, gives the best score. Final stitching is shown.

one non-manifold vertex in the horn region. Our method yields a mesh that is manifold and homeomorphic to S. In addition, our method also improves geometric error across almost all experiments, measuring geometric error with mean Hausdorff error (Figure 3.9 and table 3.1). In fact, our results show geometric error improvement of as much as 20% compared to the next-best method. In the single case that another method outperformed κCVT, our method had less than half the number of topological errors (see also Figure 3.10). In all other cases κCVT had identical or improved geometric error while reducing the number of topological errors to 0. In general there is some loss in average triangle quality when compared to uniform CVT. Similar to that of *lfs* CVT, the quality suffers due to the triangle size gradation. This is due to the fact that the optimization is not global and so even an optimal stitching algorithm cannot yield good triangles in every case.

Figure 3.10 is an example of our method avoiding topological errors that can result from extreme decimation. Uniform CVT has topological errors in the flat regions while *lfs* CVT has errors in the cylindrical regions (since it allocates most sample points to the flat regions). Our method distributes samples better, although it also has some errors in the cylindrical regions, as explained in Section 3.3.1.

Figure 3.11 shows results from remeshing the Fish model at two levels of decimation.

Figure 3.12 helps explain the gains in geometric error. The upper box highlights the horn region. The method from [59] suffers in high-curvature

regions due to local optimization. The uniform CVT method fails to allocate enough triangles around the edges of the horns. The *lfs* CVT method places far more triangles than needed in the flat horn region, causing a lack of triangles in the higher curvature but lower *lfs* ball region shown in the lower box. Our method uses far fewer triangles in the flat horn region, which doesn't affect geometric error, and allocates more triangles to the edges of the horns (a failing of uniform CVT) and to the ball region (a failing of *lfs* CVT).

We are interested in furthering this work by improving Q_{min} and θ_{min} by performing a final CVT energy optimization over the stitches and surrounding region. As we did not use feature preservation in this current work, we anticipate adding it in the future, similar to that done in [142]. We note that the method is parallelizable – after decomposition we can remesh the seperate segments in parallel, which we expect to implement in a distributed fashion.

Our method is a tool that can be used to more effectively control the trade-offs between geometric, topological, and triangle qualities. It is useful in cases where flat sheets pass close to each other, freeing the CVT algorithm from allocating inordinate numbers of samples into those regions while avoiding topological errors.

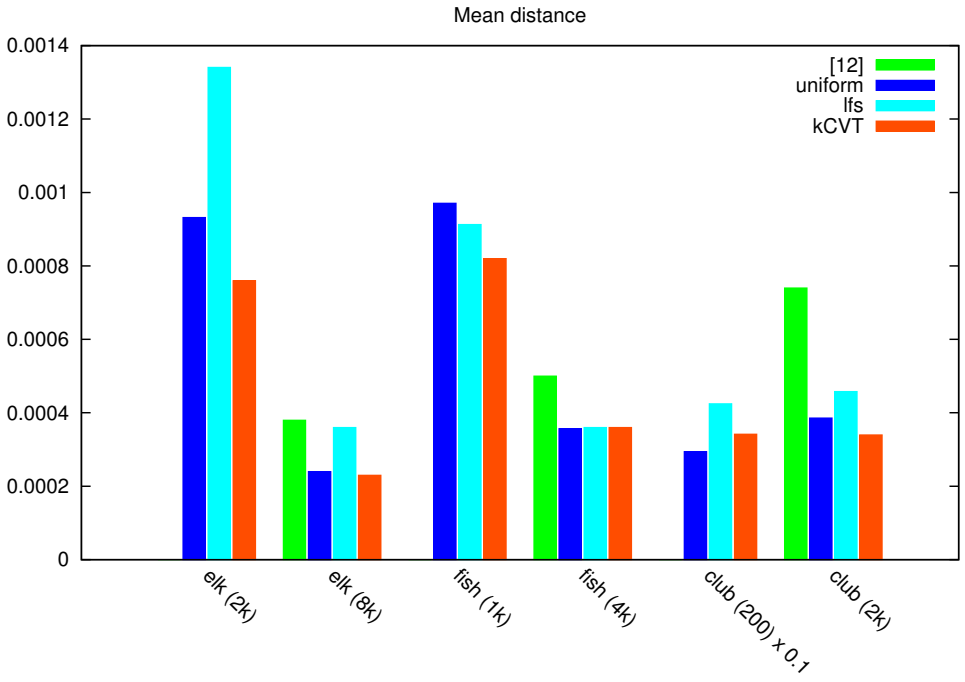
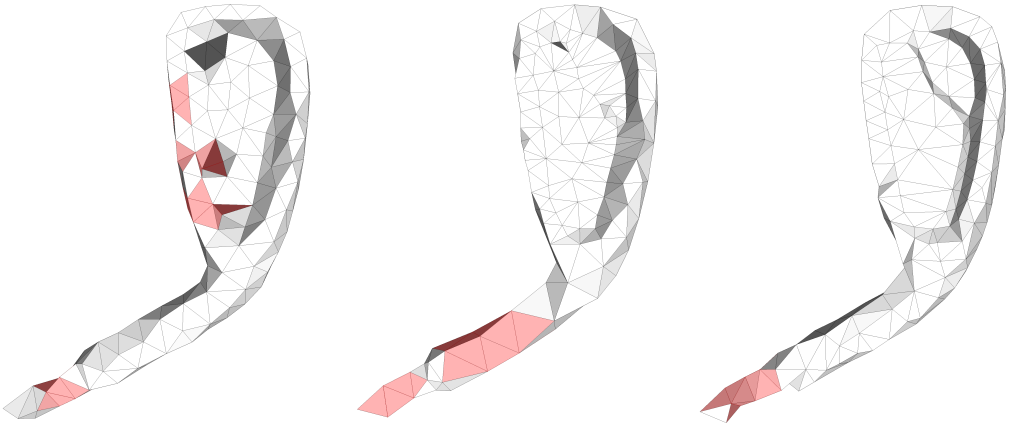
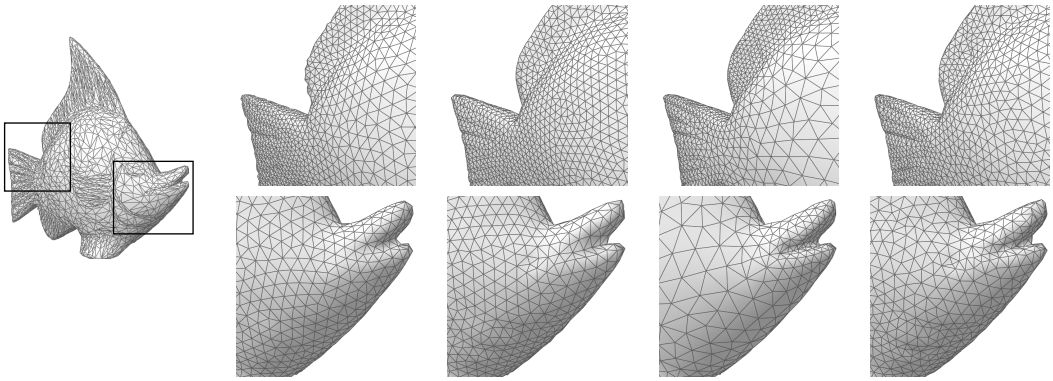


Figure 3.9: Graph of mean distance from input mesh to output mesh. κCVT performed equally or better in every test but one.



(a) (b) (c)

Figure 3.10: Remeshing the Club model with 200 sample points. Triangles with non-manifold edges are highlighted in red. (a) Uniform CVT. (b) *lfs* CVT. (c) κCVT.



(a) (b) (c) (d)

Figure 3.11: Remeshing the Fish model using 4000 sample points. Original model is shown on the left. (a) Algorithm from [59]. (b) Uniform CVT. (c) *lfs* CVT. (d) Our method, κCVT. Our method preserves features of the gills below the mouth (features that are lost using [59] and *lfs* CVT) while avoiding topological errors (uniform CVT has 11 non-manifold edges).

model	# samples	method	errors	$H_{mean} \times 10^3$	$H_{RMS} \times 10^3$	Q_{min}	Q_{ave}	θ_{min}	$\theta_{min,ave}$
Elk	2000	uniform	400	0.94	1.48	0.448	0.884	22.4	50.8
		lfs	15	1.31	1.76	0.347	0.858	19.3	48.7
		κCVT	0	0.76	1.00	0.220	0.849	11.7	48.1
Elk	8000	[59]	0	0.38	0.63	0.058	0.902	2.6	52.2
		uniform	0	0.24	0.37	0.509	0.916	24.4	53.2
		lfs	0	0.36	0.49	0.451	0.893	22.6	51.4
		κCVT	0	0.23	0.34	0.259	0.885	15.2	50.9
Fish	1000	uniform	95	0.97	0.16	0.525	0.872	28.6	49.7
		lfs	14	0.91	0.12	0.420	0.830	18.3	46.4
		κCVT	0	0.82	0.12	0.236	0.809	13.1	45.0
Fish	4000	[59]	0	0.50	0.85	0.070	0.898	2.7	51.8
		uniform	11	0.36	0.53	0.580	0.898	26.3	51.7
		lfs	0	0.36	0.51	0.407	0.864	19.4	49.1
		κCVT	0	0.36	0.58	0.160	0.863	6.5	49.0
Club	200	uniform	51	2.94	4.08	0.570	0.842	30.1	47.4
		lfs	31	4.25	6.36	0.362	0.770	13.8	41.8
		κCVT	19	3.42	4.92	0.173	0.728	9.2	39.5
Club	2000	[59]	-	0.74	1.54	~ 0	0.832	~ 0	47.5
		uniform	0	0.39	0.70	0.555	0.893	32.9	51.5
		lfs	0	0.46	0.85	0.314	0.834	12.5	46.8
		κCVT	0	0.34	0.62	0.082	0.855	4.5	48.5

Table 3.1: Table of quality statistics of our method compared to CVT without pre-segmentation. *errors* is the number of faces with non-manifold edges or vertices. H_{mean} and H_{RMS} are the mean and RMS of one-way distance, or error, from S to W divided by the bounding box diagonal, respectively. Error in the opposite direction is similar. Q_{min} (resp. Q_{ave}) is the minimum (average) Q-measure given by equation (3.7). θ_{min} (resp. $\theta_{min,ave}$) is the minimum (average minimum) triangle angle.

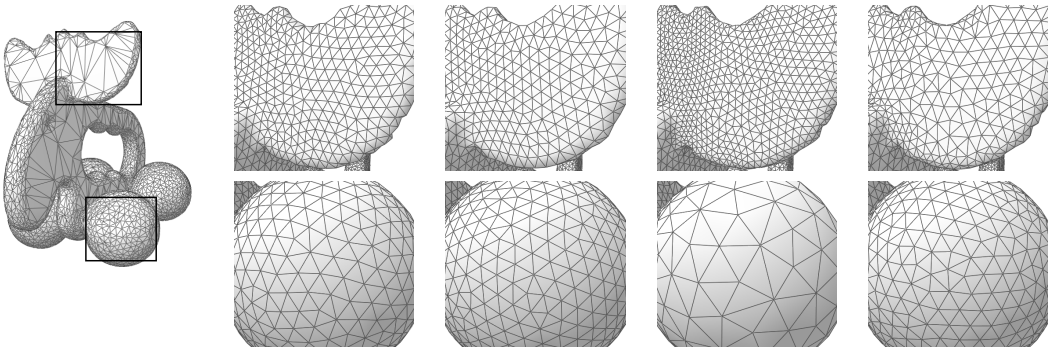


Figure 3.12: Remeshing the Toy Elk model using 8000 sample points. Original model is shown on the left. (a) Algorithm from [59]. (b) Uniform CVT. (c) *lfs* CVT. (d) Our method, κCVT. Our method produces improved meshes by distributing samples according to curvature rather than local feature size while avoiding topological errors.

Chapter 4

Segmentation and Reduction for Cable Analysis

We present an algorithm for deriving 1D cable models from surface meshes. Our algorithm uses a surface segmentation obtained during a skeletonization process and then computes surface areas and volumes to obtain geometric properties of skeleton segments. This chapter introduces the problem (Section 4.1), reviews related work (Section 4.2), describes the algorithm (Section 4.3) and presents results (Section 4.4).

4.1 Introduction

The cable model casts neuron skeletons into a 1D series of cylinders each endowed with a capacitance and resistance [32]. Such models are typically made directly from light microscopy imagery of whole or nearly-whole neurons [70, 107]. The practitioner "traces" trees of neurite segments [66] and assigns each segment a length and radius from which the volume and surface area can be computed. The resulting skeleton is then augmented with resistances and capacitances on each segment based on properties of the physical specimen (e.g. area of the brain, type of neuron).

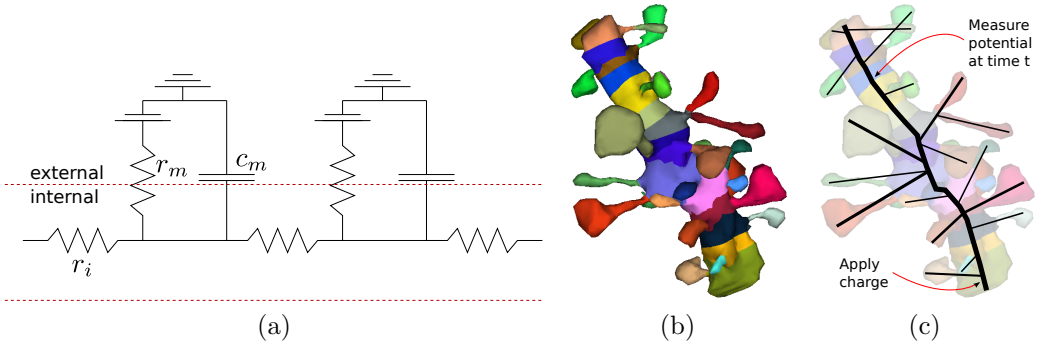


Figure 4.1: (a) The cable model comprises a set of segments with capacitances and resistances that can be simulated as an electrical circuit. (b) Our algorithm segments a surface mesh into roughly cylindrical regions. (c) Each region corresponds to a segment of the skeletonization.

With the neuron now modeled as a branching "cable," the problem becomes solving a partial differential equation over the 1D compartmentalized domain (Figure 4.1a).

While most cable simulations are done at the microscale level, simulating at the nanoscale level, or level at which the ultrastructure can be resolved, is a natural next step. Progress has been slow, however, because of a lack of models. As described in chapters 2 and 6, robust methods exist for making surface mesh models that capture neuronal ultrastructure. Our approach to creating cable models capturing nanoscale neuronal morphology is to derive the 1D model from an existing surface mesh. This approach is advantageous in four respects: 1) radii and surface areas can be computed directly from the surface mesh, giving more accurate measurements than are usually obtained from measuring 2D contours; 2) given a surface mesh, the 3D to 1D reduction

process we describe is automatic, so researchers with access to surface meshes can obtain cable models with little effort; 3) if, during cable simulation, a specific area of interest is identified, the neuroscientist has at her disposal the original, high resolution morphological surface model that can be used in high-resolution reaction-diffusion simulations; 4) as multiscale simulation technology progresses, models of the same specimen at multiple resolutions will be needed. Our algorithm not only generates a 1D cable model, but also includes a mapping between 1D and 3D representations, so it is straightforward to replace a 1D region with a 3D region, and vice-versa.

Our algorithm reduces watertight, manifold triangulated surface meshes to a 1D skeleton of segments. Each segment is assigned a surface area and volume (and therefore also cross-sectional area and length) that induces a logical cylinder approximation. We reduce the mesh by iterative geometric contraction until all triangles are approximately degenerate. Mappings between the resulting skeleton and original mesh provide for accurate area and volume computations. We review work that has been done in related areas, describe our algorithm, and show results.

4.2 Related work

Our goal is a volumetric decomposition. However, such a decomposition can often be found via either volumetric shape approximation or surface mesh segmentation. Shape approximation methods fit geometric primitives as proxies to a surface or volume. Variational shape approximation itera-

tively fits geometric proxies to a surface while minimizing an error term. The geometric proxies suggested originally were planar ellipses [36], but this was later extended to other primitives, including spheres, cylinders and rolling ball blend-patches [140]. These approaches do not necessarily preserve surface area or volume. An initial partitioning is required, which can use a curvature tensor field [37] to flood-fill regions based on curvature. The minimization algorithm proceeds as a variant of Lloyd's clustering algorithm [103]. Challenges in variational fitting approaches include determining the axis of a cylinder (which is particularly difficult in regions that don't map well to a cylinder), repartitioning of triangles based on a cylinder, and determining a guarantee of convergence.

Other segmentation approaches include those that use skeletonization. The algorithm of Li et al [95] first computes the skeleton and then cuts the surface mesh with a series of planes perpendicular to the skeleton edges. Lien et al [96] iteratively and simultaneously decompose the surface and skeletonize until an error threshold is met. Our approach skeletonizes the surface which induces a surface segmentation. We require the mapping to the original surface in order to compute surface area and volume of skeleton edges. We follow the approach of Au et al [9], in which a mapping is implicitly created during the skeletonization process.

4.3 Algorithm

Our algorithm finds a set of cylindrical models that approximate a triangulated surface mesh. The algorithm has four main steps:

1. Iteratively contract the surface mesh until it converges to a skeleton.
2. Map skeleton edges back to surface regions, inducing a surface segmentation.
3. Refine segmented regions, including smoothing of region interfaces and merging poorly-shaped regions.
4. Fit cylinders to regions.

After completing these steps we are left with a surface segmentation where each region is approximately circular and each interface between regions touches exactly two regions, i.e., interfaces are edge rings. Each region is then approximated with a cylinder and exported to cable model analysis software. Steps 1-2 of the algorithm use Au's method [9] while the remainder of the steps and the algorithm as a whole are contributions of this thesis. We now describe each step of the algorithm.

4.3.1 Contraction [9]

Given a surface mesh we perform geometric contraction iteratively using Laplacian smoothing. Laplacian smoothing attempts to solve the Laplace equation, thereby removing extrema, by moving vertices of the mesh along

their curvature-flow normals, that is, the approximate vertex normal weighted by one-ring area and local mean curvature $\frac{1}{2}(\kappa_1 + \kappa_2)$. Solving the Laplace equation on a surface mesh can be cast as a linear system $LV_{k+1} = 0$. Given a vertex x_i and a set $\{x_j\}$ of vertices connected to x_i by an edge, we define α_{ij} and β_{ij} to be the angles opposite the edge (i, j). Let E be the set of edges in the mesh and V_k the vertex positions at iteration k. Then we represent the weighted normals of all vertices as a matrix

$$L = \begin{cases} \lambda_{ij} = \cot \alpha_{ij} + \cot \beta_{ij} & \text{if } (i,j) \in E \\ \sum_{(i,l)\in E} \lambda_{il} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

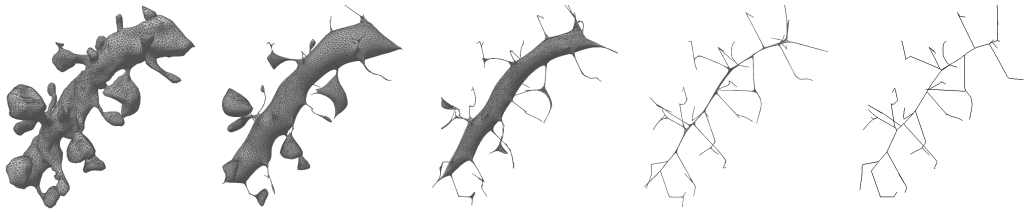
Note that the connectivity of the surface mesh doesn't change during contraction – only the vertex positions are modified. The rows of this system are called the contraction constraints.

Attraction constrains are added to the system to control contraction and pull the contracting geometry back toward the original mesh. The resulting system is

$$\begin{bmatrix} \Lambda_C L \\ \Lambda_A \end{bmatrix} V_{k+1} = \begin{bmatrix} 0 \\ \Lambda_A \end{bmatrix} V_k$$

Where Λ_C and Λ_A are contraction and attraction weighting parameters, respectively.

We iteratively solve this system using a sparse linear solver (we used the Eigen software library [63]) until the summed triangle surface area differences between iterations reaches a threshold, indicating that the triangles are approximately degenerate (Figure 4.2). We then perform a "surgery" step



(a) (b) (c) (d) (e)

Figure 4.2: Iterative mesh contraction to approximate degeneracy. The last step collapses degenerate triangles into line segments.

where the triangles are collapsed into line segments using an edge-collapse algorithm similar to that proposed by Garland and Heckbert [61]. The result of contraction and surgery is a skeleton of connected line segments. The skeleton may not be a tree – if the original mesh has genus greater than zero then the skeleton will have cycles.

4.3.2 Map skeleton segments to triangle regions [9]

During the surgery step, nearly degenerate triangles were collapsed into 1D line segments. During the process, a mapping M of triangles to segments is maintained, such that $M(T)$ for a given triangle yields the label of the skeleton segment that T was collapsed to. Grouping all triangles of surface S into regions gives a set of regions $\{R_i\}$ where $R_i = \{T \in S | M(T) = i\}$ (Figure 4.3).

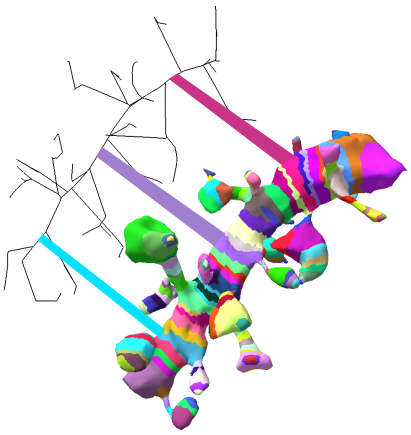
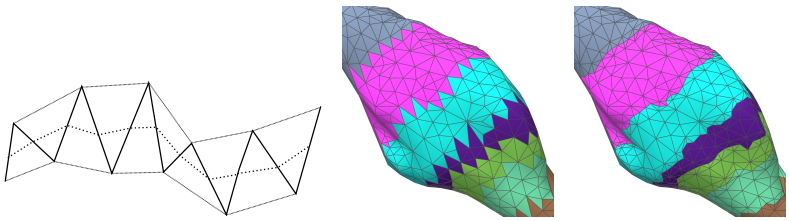


Figure 4.3: Mapping skeleton segments to regions of triangles in the original mesh.

4.3.3 Region refinement (contribution)

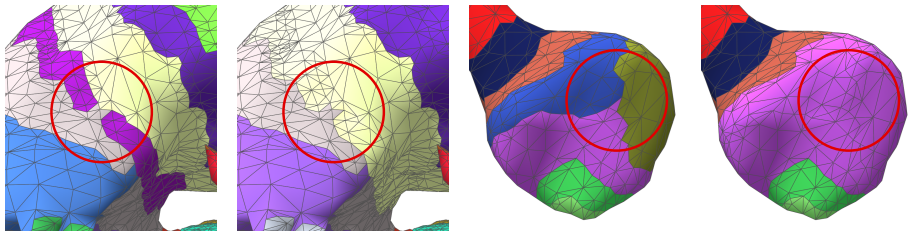
The first step in refinement of the regions is initial smoothing of the interface. The interface between two regions is defined to be the path of connected edges shared by the regions. More formally, an interface Γ_{ij} between regions labeled i and j is the set of edges $\{e_k | M(T_{e_k}) \in \{i, j\}\}$ where T_e is the pair of triangles incident to edge e. Initial smoothing simply connects the centroids of adjacent edges (Figure 4.4).

We then merge regions in illegal interfaces (Figure 4.5). An illegal interface is one that touches more than two regions, or equivalently, an interface is legal if and only if it exactly forms a graph with a single cycle and no branching, or a "ring." To resolve illegal interfaces we take a greedy approach. For an illegal interface Γ_{ij}, reassign each triangle $T \in R_j$ to label i, effectively merging the two incident regions.



(a) (b) (c)

Figure 4.4: Region interface smoothing. (a) The smoothed interface connects the centers of edges of the original interface. (b) Before smoothing. (c) After smoothing.



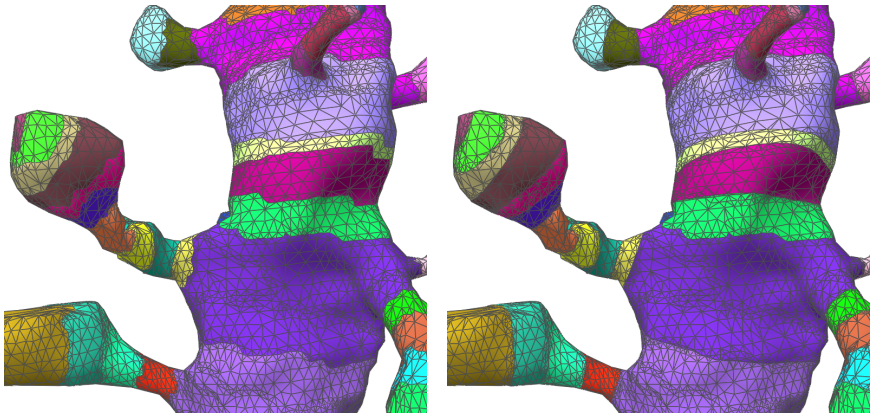
(a) (b) (c) (d)

Figure 4.5: Merging of two common illegal junctions.

The final step in region refinement is plane fitting (Figure 4.6. We consider each interface ring Γ_{ij} and fit the vertices in each ring to a plane using least-squares fitting. We then cut the surface with the plane and cut and reassign triangles as necessary so the new interface Γ'_{ij} between the two regions is planar.

The plane may intersect the surface at multiple locations, potentially yielding more than one intersecting ring. To find the ring that corresponds to Γ_{ij} we rely on the fact that some edge $e \in \Gamma_{ij}$ crosses the cut plane P, i.e., $e \cap P \neq \emptyset$ for some $e \in \Gamma_{ij}$. This is straightforward to show by contradiction: if every vertex $v \in V(\Gamma_{ij})$ is in the same halfspace P^* then moving P in the direction of the points will decrease the error. Thus not all points can be in the same halfspace of a least-squares-fit plane P. To find the correct ring, assume Γ_{ij} is in general position in the sense that no vertex $v \in V(\Gamma_{ij})$ in exactly in the plane P, and select edge $e \in \Gamma_{ij}$ such that $e \cap P \neq \emptyset$. Let $e_1 = e$ and let $p_1 = e_1 \cap P$ be the intersection point of the chosen edge. We define $\bar{e}_i$ to be a directed edge that assumes the orientation of its corresponding triangle, and $\bar{e}'_i$ to be the edge incident to $\bar{e}_i$ in the opposite direction. Then the cut path is defined as $p_{i+1} = (T_{\bar{e}'_i} \setminus \bar{e}'_i) \cap P$. This traces a planar path through triangle interiors.

The traced planar interface can be illegal if Γ'_{ij} intersects some triangle T with a label other than i or j. In this case we fail and set $\Gamma'_{ij} = \Gamma_{ij}$.



(a) (b)

Figure 4.6: Fitting planes to interfaces.

4.3.4 Fit cylinders to regions (contribution)

In order to fit a cylinder to a region R_i we first compute the surface area and volume, and then determine which two interface rings constitute the ends of the cylinder, inducing a cylinder axis.

Surface area is computed simply as $\sum_{T \in R_i} A(T)$ where $A(T)$ is the area of triangle T. Volume computation is more involved. Since regions are not closed polyhedra, volume is not defined. We use an approximation approach that preserves volume, that is, $V(S) = \sum_{R_i} V(\bar{R}_i)$ where $\bar{R}_i$ is equal to R_i with holes closed as described below, and $V(*)$ is the volume of a polyhedron. Our approach to closing holes in R_i is to triangulate each interface ring to its centroid. After closing all holes in R_i to form $\bar{R}_i$ we compute the volume of $\bar{R}_i$. This approach has two important properties. The first is that since an interface ring Γ_{ij} is shared by exactly two regions $R_{\{i,j\}}$, the closed ring is

exactly the same for both regions, to $\bigcup \bar{R}_i = S$ and $\bigcap(\bar{R}_i \setminus \partial\bar{R}_i) = \emptyset$. Thus, the closed regions are a tessellation of S. The second property is that our approach is robust to intersecting triangles in the closed holes. Triangulating a polygon to its centroid can yield intersecting triangles if the polygon is non-planar (meaning it failed plane-fitting in our case) or if the polygon is non-convex. Nevertheless, computing the volume remains consistent as long as it is computed using the standard, robust algorithm of $V(R) = \sum_{T \in R} V(\mathscr{T}(p, T))$ where p is an arbitrary point and $\mathscr{T}(p, T)$ is the tetrahedron formed by vertices $\{p, T_1, T_2, T_3\}$. It should be noted that the centroid of an interface rings is not necessarily inside S. Our implementation checks for this case and moves it inside if necessary.

The final step in fitting is to determine an axis. We don't fit an axis in the geometric sense: our purpose is to determine which interface rings correspond to the open ends of the conceptual cylinder. In cable model simulation, position along a cylinder is parameterized in the range $[0, 1]$ with branching points on the interior. So in a sense, we are simply determining which interface rings correspond to branching points and which two are the "start" and "end" of the region. We first consider a region R_i with only one interface ring Γ_{ij}. We assign Γ_{ji} on the adjacent region R_j to be the 0 ring. We then iterate through remaining interface rings of R_j and assign the ring Γ_{jk} with perimeter closest to the perimeter of Γ_{ij} to be the 1 ring. We then compute the parameter values for each remaining interface ring Γ_{jl}. Let $c(\Gamma_{ij})$ be the centroid of Γ_{ij} and let q be the midpoint between $c(\Gamma_{ji})$ and $c(\Gamma_{jk})$. Further, let $\mathbf{u}_{ji}$ be the

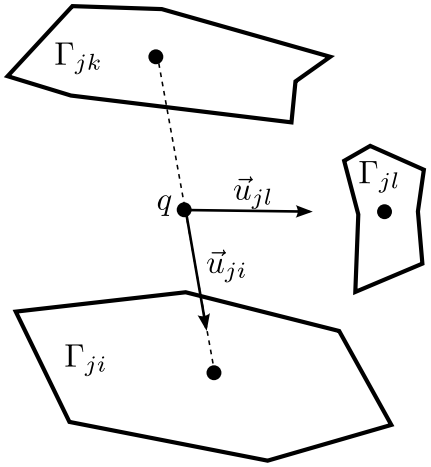


Figure 4.7: Computation of parameter values for interface rings.

normalized vector $c(\Gamma_{ji}) - q$. Then parameter $t_{jl} = \frac{1}{2}\left(1 - \mathbf{u}_{jl}^T \mathbf{u}_{ji}\right)$. See Figure 4.7.

4.4 Results and discussion

We report runtimes in table 4.1. Full models of neurons can be reduced to a 1D cable model in seconds, whereas it has previously been a labor-intensive process.

Our primary goal with this work is to produce neuronal models for cable model analysis. Figure 4.8 shows the reduced model of a dendrite along with a plot of electrical potentials measured at 3 points that was output from the NEURON simulation environment [32].

We anticipate extending this work, however, to be a more general surface segmentation method. With well-behaved ring-like segments we believe

dataset	# tris	contract	surgery	plane fit
dendrite	23966	10.8s	5.2s	61.3s
elk	10388	1.4s	2.1s	12.9s
club	4784	2.3s	0.9s	3.5s
gargoyle	5000	2.2s	1.0s	5.8s

Table 4.1: Table of segmentation timings. Times for contraction, surgery and plane fitting are reported. Times for initial interface smoothing, merging and axis computation are negligible. Tests were performed on a Linux Kubuntu workstation with an Intel Xeon quad core CPU at 3.20 GHz with 4 GB memory. The dendrite dataset is shown in Figure 4.8 and the elk, club, and gargoyle models are in Figure 4.9.

that a powerful segmentation method is possible. So-called superpixels [115] have gained significant attention in the image segmentation community. Superpixels over-segment an image, breaking the image into larger regions that can then more easily be segmented using an objective function than it would be using pixels directly. We are interested into subdividing our regions into a larger number of smaller regions by parameterizing each region to a 2D sheet and cutting along paths where $\kappa_1\kappa_2$ is small (corresponding to saddle points). Using these "superregions" the surface segmentation problem is simplified. As our initial segmentation algorithm generalizes well to models beyond neurons (Figure 4.9) we believe our algorithm to be a viable way of bootstrapping an initial segmentation.

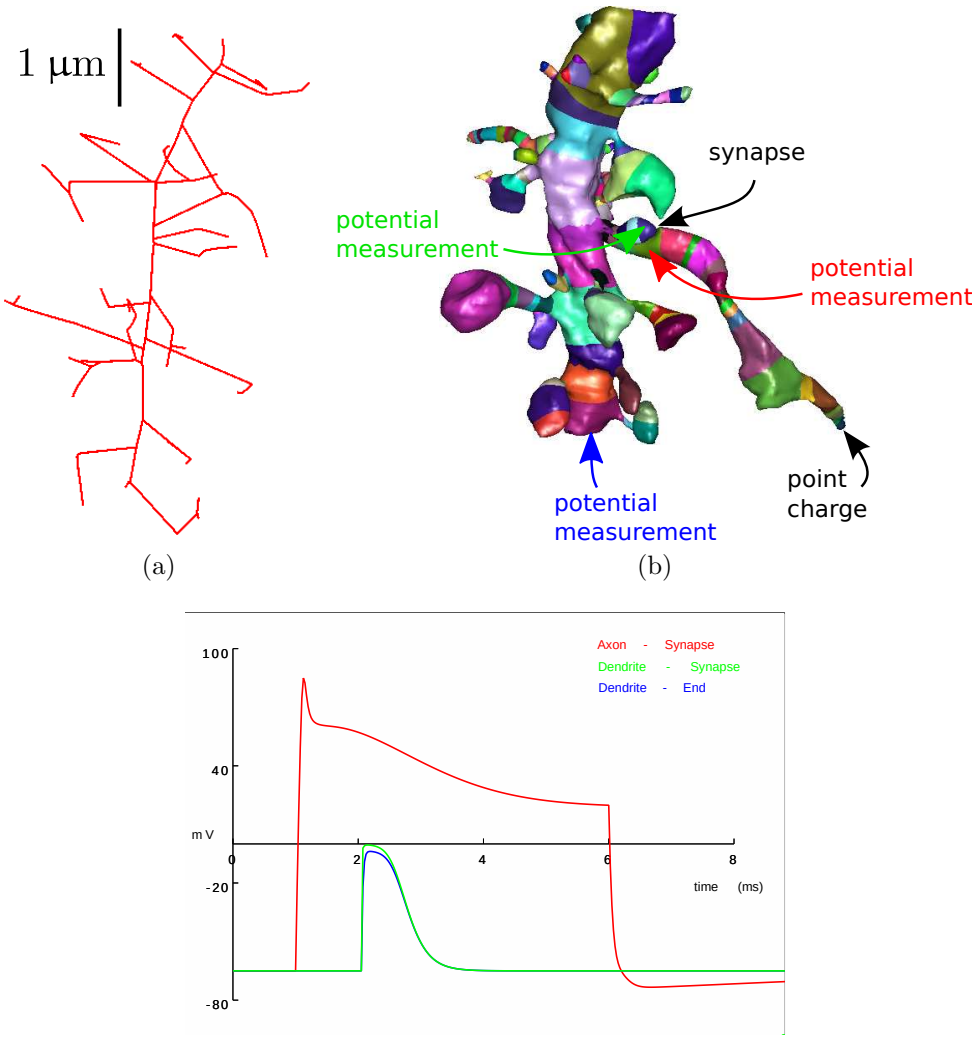


Figure 4.8: Multi-compartment model generation. Our surface segmentation first skeletonizes the mesh (a), which induces a segmentation (b). This graphic shows a simple cable model simulation. The compartmentalized versions of the axon and dendrite are input to NEURON. A synapse with a threshold and delay is added between the dendrite and axon and a point charge is placed at the end of the axon. Three potential measurements are made over time. Arrow colors correspond the potential measurements reported in the NEURON simulation graph in figure (c).

(a) (b) (c)

Figure 4.9: Our algorithm generalizes well to models beyond neurons.

Chapter 5

An Adaptive Distance Transform for Fast Voronoi Diagram Computation

This chapter presents a novel adaptive distance transform and surfacing algorithm for computing the generalized Voronoi diagram (GVD) efficiently. The motivation for this work is tessellation of neuronal domains into cells that are suitable for immersed boundary finite element analysis, but the applications of the GVD reach far beyond our neuronal modeling project. In this chapter we first motivate the problem in the context of electrophysiological simulation (Section 5.1), then give background on the GVD construct itself (Section 5.2). After reviewing related work both on distance transforms and GVD computation (Section 5.3) we discuss the four main parts of the algorithm: octree (Section 5.4), distance transform (Section 5.5), resolution of ambiguities (Section 5.6), and the GVD itself (Section 5.7). We then discuss our results (Section 5.8) and draw conclusions (Section 5.9).

5.1 Introduction

Nano-scale simulation of neurons includes modeling ion diffusion. The electrical current that travels along and between neurons via synapses is carried

by ions, and an important question in neuroscience is how the structure, or shape, of neurons affects the behavior of charge-carrying ions at the nano-scale level [90, 125]. Simulation helps to understand the relationships between neuronal structure and ion behavior.

A common type of simulation is the finite element method (FEM) [42]. FEM numerically solves differential equations over a cell complex of a spatial domain. A requirement of the complex is that it respect domain boundaries: the meshing algorithm must construct the spatial cells such that no domain boundary intersects the interior of a cell. This is most often done using a constrained triangulation. The problem that neuronal data present is that the range of spatial resolution is very large. That is, some objects in the domain, such as axons and dendrites, have large interiors which require few cells for a reasonable complex. Other areas, such as extra-cellular space (ECS), or the space between objects, require far smaller cells. The number of cells required to subdivide such a domain can be inordinately large (figures 5.1a-5.1b). As a result, most experiments are done on small regions in order to keep the number of cells small. Our goal is to produce complexes with fewer cells such that simulations can be done on larger volumes.

Figure 5.1 shows various types of complexes. Constrained Delaunay triangulation respects boundaries but often yields triangles with high aspect ratio which can cause instability in the numerical solution. Constrained conforming Delaunay triangulation yields higher-quality triangles, but at the expense of far more triangle elements. Meshless methods grid the space into uniform

cells and model the boundaries using alternative methods such as WEB-splines [10, 72, 73]. This is a promising approach but we still have the problem that a very large number of uniform cells would be required to ensure that each cell contains only a single connected component of the boundary.

We propose a new type of spatial decomposition to be used with immersed boundary methods [144]. The cells in this complex, which we call "aligned cells," have two properties: 1) cells are convex and 2) immersed boundaries have only one connected component inside a given cell (Figure 5.1g). Our proposed complex is aimed at producing fewer cells than a constrained, conforming Delaunay triangulation without comprising the numerical solution.

Generation of aligned cells has three main steps:

1. Compute a generalized Voronoi diagram V_G yielding a bisector B_G between objects.

2. Compute an ordinary Voronoi diagram V_O of the vertices of B_G. Let B_O be the bisector of V_O.

3. Compute the boundary complex of aligned cells as $B_G \cup B_O$.

Figure 5.2 shows the steps of the algorithm. Our approach requires computation of the generalized Voronoi diagram (GVD). The main contribution of this chapter is efficient computation of the GVD.

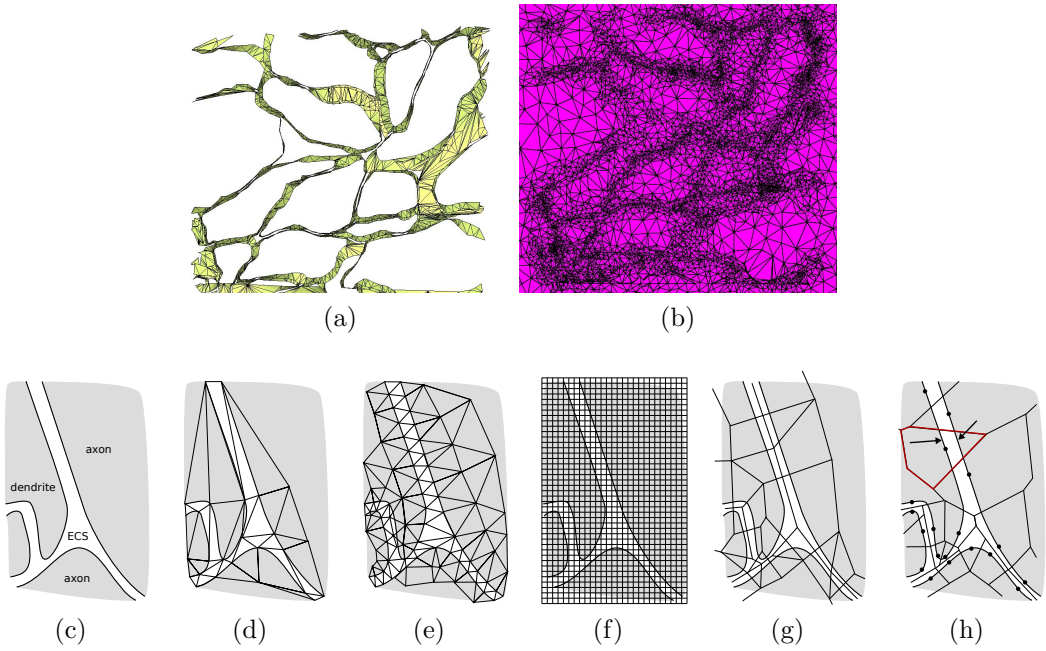


Figure 5.1: (a) Surface reconstruction between two slices. Volume of the domain is 1 µm x 1 µm x 100 nm. (b) The tetrahedralization contains 124458 tetrahedra. Integrating over even a small volume of 9 µm on a side yields 10^8 tetrahedra, corresponding to 10^8 variables to solve in a FEM simulation. (c) We desire to construct suitable cells for domains which have very close boundary spacing in some areas. (d) Constrained Delaunay triangulation yields triangles with large aspect ratio which causes badly conditioned linear systems and poor error convergence. (e) Constrained, conforming Delaunay triangulation gives good-quality triangles for well-conditioned systems, but the number of triangles can be very large. (f) Meshless methods use a uniform grid and represent boundaries as weighted b-splines. (g) Our complex of aligned cells uses few, well-formed elements. The intersection of any given cell with the surfaces gives a single connected component. (h) An example of a non-aligned cell: the red cell intersected with the surfaces has two connected components and fails property 2 for aligned cells.

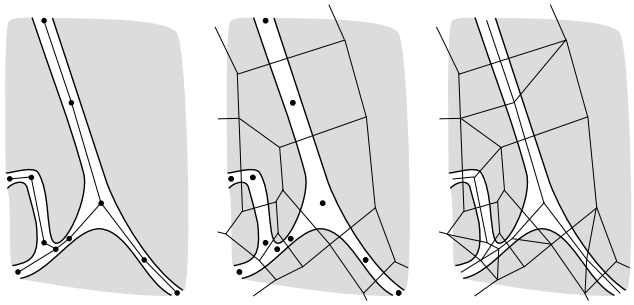


Figure 5.2: Algorithm outline for constructing aligned cells. (a) Find a bisector of the objects. (b) Compute the ordinary Voronoi diagram using the vertices of the bisector as sites. (c) Union the two and add edges using existing vertices to ensure convexity of the cells.

5.2 Background

An unsigned distance function $f : \mathbb{R}^n \to \mathbb{R}$ gives the minimum distance from a point p to an object S modulus some error, that is, $\inf_{q \in S} \text{dist}(p, q) + \epsilon(p)$ where $\text{dist}(p, q)$ is the distance between points p and q and $\epsilon(p)$ is an error measure. Unsigned distance functions, and especially their cousin, the signed distance function, which endows the distance with a sign depending on whether p is inside or outside of an object, are applicable in a large number of applications, including motion path planning, geospatial distance queries, object representation and processing, image quantization, and level-set functions. One important construct that can be derived from a distance function is the generalized Voronoi diagram (GVD), a generalization of the ordinary Voronoi diagram.

Given a set of seed points $\{x_i\}$, or sites, the ordinary Voronoi diagram

(VD) is the locus of points with at least two closest sites. It divides space into cells $\{VC_i\}$ such that every point in a cell VC_i is closer to x_i than to any other site. That is, given a space X where the distance between two points is given as $\text{dist}(a, b)$, the Voronoi cell for a site x_i is $VC_i = \{p \in X | \text{dist}(p, x_i) < \text{dist}(p, x_j), j \neq i\}$. If site x_i is given the label i then we say that every point $p \in VC_i$ has label i. The VD is a bisector of the space between sites and is piecewise linear.

The generalized Voronoi diagram lifts the restriction that sites be punctual, allowing arbitrary objects to be sites. The definition of the GVD is the same as that of the VD: given a set of objects $\{S_i\}$, a generalized Voronoi cell (GVC) is defined as $\text{GVC}_i = \{p \in X | \text{dist}(p, S_i) < \text{dist}(p, S_j), j \neq i\}$ where $\text{dist}(p, S_i) = \inf_{q \in S_i} \text{dist}(p, q)$ and can be approximated by a distance function. Because sites can be arbitrary objects, the GVD is higher order in general [26]. For example, the bisector of a point and a line is a quadratic curve. Generalized Voronoi diagrams are difficult to compute analytically in general [26, 69] and so most approaches use a distance field computed by a distance function to construct an approximate GVD.

In order for the distance function to be useful in building a GVD it must not only yield a distance, but also the label of the closest site, that is, $f : \mathbb{R}^n \to \mathbb{R} \times L$ where L is the set of all labels. The GVD is then exactly the set of critical points at which the label changes. We note that the medial axis is the analog to the bisector in a continuous setting, that is, when an infinite number of punctual sites are used.

Distance transforms (DTs) are a popular method of computing a distance field and include chamfer DTs, vector DTs, and the fast marching method [81]. The majority of existing distance transforms compute distances over a grid. While appropriate for many applications, uniform grids require a prohibitively large number of pixels (voxels) when the spatial extent is large compared to spacing between objects. Adaptive distance fields (ADFs) [57, 128] use spatial subdivision methods, such as quadtrees and octrees, to compute distances at higher resolution in areas of detailed object features.

Main contributions In computation of a GVD, we are less concerned with object detail; rather, we adapt subdivision to most efficiently compute a distance field for GVD computation. Our main contributions are fourfold.[1]

1. We present a novel adaptive subdivision scheme that subdivides as necessary to resolve objects. Object features are resolved only if they closely approach another object. Our octree data structure gains memory savings by storing cell vertices in an adjacency list rather than storing cells hierarchically, and by omitting spatial information from the data structure.

2. Our distance transform is computed after the octree is built and uses a scheme that requires $\mathrm{O}(F + N)$ distance computations for piecewise

[1]Our algorithms support both 2D and 3D. For simplicity, we will use 3D terms (e.g. octree rather than quadtree) when discussing dimension-independent concepts, and will revert to 2D terms only when contextually necessary.

linear site objects (e.g. polyhedra) where F is the number of faces and N is the number of octree leaf cells. Distances are computed on octree vertices and are error bounded.

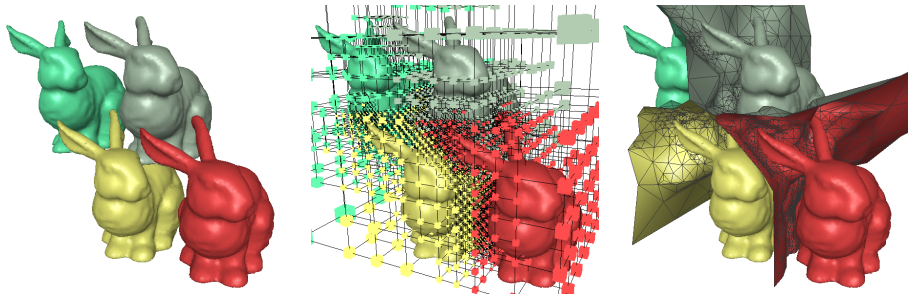
3. We trace out the GVD over the octree distance field using an efficient $\mathrm{O}(N)$ algorithm. The GVD is guaranteed to separate each object into its own generalized Voronoi cell, i.e., any path from a point $p \in S_i$ to a point $q \in S_j$ must intersect the GVD, a guarantee that is not usually made by uniformly-gridded methods.

4. We demonstrate various applications of the GVD in 2 and 3 dimensions, including motion path planning, pruning of occluding structures, and exploded diagrams.

Our algorithm has two primary products, a sampled distance field and a GVD. As the GVD is our primary motivation, the distance transform is designed to support efficient GVD computation. We have four main steps in our algorithm: 1) build the octree over the set of objects; 2) compute distances on octree vertices using a wavefront expansion; 3) resolve ambiguous cells through further subdivision; 4) compute the GVD surface by finding octree edges with differing end labels. We discuss each step in detail.

5.3 Related work

Distance fields Most sequential distance transforms, including chamfer distance transforms, vector distance transforms (VDT) and the fast marching



(a) (b) (c)

Figure 5.3: Given a set of objects, we use an adaptive distance transform over an octree to compute the generalized Voronoi diagram. The boundary of the generalized Voronoi diagram separates objects and bisects inter-object spacing. In the figure, triangles are colored with the color of the cell's contained object. (a) The original object data. (b) The octree with labeled vertices. (c) The GVD computed from the octree.

method (FMM) [81], lack adaptivity and thus require inordinate amounts of memory when feature size is small compared to bounding box size. However, there has been some excellent work in adaptive methods, mostly in the context of building a distance field for a single object, and thus adaptive in the sense of subdividing to full resolution anywhere near the surface, or adapting according to local feature size.

[128] computes the distance field exactly at vertices of an octree for redistancing when computing level sets. The octree is fully resolved everywhere on the surface. [57] proposed a similar approach that resolves the octree fully only in areas of small local feature size. Storage of the octree was later improved by [94] who introduced a compression scheme while maintaining fast random spatial acces. At least two works [21, 143] have implemented ADFs

using the GPGPU achieving speedups. Their purpose was in rendering and they thus adapt to object features. They encountered the expected challenges in storing an octree in GPU memory, which were overcome by using hashing schemes for vertex indexing.

All previous works in adaptive distance transforms focus on adapting resolution either by fully subdiving everywhere on surfaces or adapting subdivision to retain features in order to produce accurate distance fields. Our work is effective in computing distance fields, but is geared primarily toward the GVD for closely-spaced objects.

Generalized Voronoi diagrams A theoretical framework for generalized Voronoi diagrams can be found in Boissonnat et al. [26]. While ordinary Voronoi diagrams are well studied and efficient algorithms to compute them exactly exist [40], exact GVD algorithms are limited to a small set of special cases [84, 93]. In an early work, Lavender et al. [92] define and compute generalized Voronoi diagrams over a set of solid models using an octree. Their method also handles 2D regions that have two closest sites, a subtlety that our algorithm handles by arbitrarily choosing an owning site. Etzion and Rappaport [52] represent the GVD bisector symbolically for lazy evaluation, but are limited to sites that are polyhedra.

In recent years Voronoi diagram algorithms that take advantage of fast graphics hardware have become more common [81]. These algorithms are efficient and generalize well to the GVD [30, 69, 116, 129]. While GPGPU ap-

proaches introduce error by discretizing space and in other ways, the primary shortcoming that we address in this work is that they are not inherently adaptive, that is, a large number of voxels over the entire space may be required to resolve two closely-spaced objects.

5.4 Build octree

Unlike previous ADF approaches, we build the octree as a pre-processing step to computing a distance field. This decoupling allows us to optimize our octree construction by temporarily converting polyhedra to integer-based representations, and using entirely integer arithmetic. The size of integer is dependent on object spacing relative to bounding box size. Let d be the minimum distance between two objects. To resolve close objects the octree in the region must be subdivided down to cells of edge length $d/\sqrt{3}$ (see appendix C.1). We assume a minimum length of $d/2\sqrt{3}$ in order to provide for empty "buffer" cells between objects for ease of bisector computation (Section 5.7). If the length of the longest edge of the bounding box of all objects is L, then our integer representation must have at least $\log_2 2\sqrt{3}L/d$ bits. Our implementation uses 32-bit integers for a maximum octree level of 32, which has been sufficient in all of our tests.

Our octree data structure is not hierarchical. Since we have no need for spatial queries we store the vertices as an adjacency list, thereby avoiding redundancy inherent in the hierarchical format. Our distance transform also requires that, given a vertex v, we quickly find its "visible" neighbors

(Section 5.5), an operation well-suited to an adjacency structure. Additionally, vertices do not store their spatial position. Position is propagated during traversal, which results in memory savings of 3 floating point values per vertex, but it also means that a vertex position can only be computed by an octree traversal. Again, this is well-suited to our application in that all computations requiring vertex spatial locality are done during traversal. The only vertex position stored is that of an origin vertex from which all traversals start. As the structure is stored without hierarchy, the traversals are also done over the flat adjacency graph. As we will see later, vertices are also required to store a label and an index to a closest point.

With a flat octree data structure our octree construction algorithm, while intuitively similar to typical approaches, differs in the implementation in that subdividing an octree cell inserts vertices at vertex-vertex edges rather than adding child nodes to a parent node. Position information for a vertex is maintained by the stack during construction and then discarded when a cell and its children are in place.

We have two options for predicates to use in deciding whether to subdivide a cell. The first is for GVD computation. Given a cell c, subdivide if (1a) $|\{S_i | c \cap S_i \neq \emptyset\}| > 1$ (if the cell intersects more than one object), or (1b) $n \in nbrs(c) \cap S_j \neq \emptyset, j \neq i$ where $nbrs(c)$ is the set of edge-neighbors of c (if an edge-neighbor of c intersects with a different object). Test (1a) ensures that every leaf cell after construction will intersect at most one object, and test (1b) ensures that objects will be separated from each other by at least

one empty buffer cell.

The second predicate is for use in building a distance field suitable for distance queries at arbitrary points and ensures that object features are well-resolved (Figure 5.6). The simplest predicate is to subdivide if the cell is non-empty up to a maximum level. More sophisticated feature resolution predicates can be used (e.g. the bilinear interpolation test of [57]).

5.5 Distance transform

We compute the distance field using a novel wavefront distance transform that is similar in spirit to that of [27], with the fundamental difference being that we compute over an octree rather than a uniform grid. Each octree vertex has two properties – a label and a closest point. An octree cell c is said to be empty if $c \cap S = \emptyset$. An octree vertex is empty if all 4 incident cells are empty. The two main steps of distance computation are initialization of non-empty vertices and wavefront expansion. Algorithm `COMPUTE_DISTANCES` comprises both steps.

Define the distance between two points $\text{dist}(a, b)$ to be infinity if either a or b is `null`. Let $\{S_i\}$ be the set of objects intersecting octree cells incident to a vertex v.

Algorithm `COMPUTE_DISTANCES` initializes the point assignments of non-empty vertices (figures 5.4b and 5.4f). All closest points $point[v]$ are stored as an array and each vertex maintains an index into the array, taking advantage of

Algorithm 3: COMPUTE_DISTANCES

```
Input: Octree
// Initialization
foreach vertex v in Octree do
    if v is non-empty then
        point[v] := closest point on {S_i} to v
        label[v] := i
    end
    else
        point[v] := null
    end
end
V := the set of all vertices in Octree
// Wavefront expansion
while V is not empty do
    v := vertex in V with minimum |v-point[v]|
    remove v from V
    foreach visible vertex a from v do
        if dist(a, point[v]) < dist(a, point[a]) then
            point[a] := point[v]
            label[a] := label[v]
            reorder a in V
        end
    end
end
```

the fact that roughly half of computed closest points are shared among multiple vertices. The closest points are computed exactly: only surface points in cells incident to the vertex need be searched, making the loop an $\mathrm{O}(N)$ operation. It then iterates over vertex priority queue `V` in multiple expanding wavefronts, which are similar in behavior to Dijkstra shortest-cost path wavefronts in that the priority queue is sorted on distance to the nearest site. The vertex v at the front of the queue is the closest vertex to a site among all vertices in the queue, modulus an error term discussed below. The assigned closest point p is pushed to all visible neighbors of v. A visible neighbor n is defined as a vertex on a cell edge-incident to v such that a line segment $s = \overline{vn}$ between v and n does not intersect any other vertex and only intersects the octree edges incident to v (Figure 5.5). Visible neighbors are found efficiently using our vertex adjacency data structure.

Our algorithm does not give exact distances on vertices, but the error is bounded by $\frac{\epsilon(v)}{\delta(v)} \leq \frac{1}{2}$, where $e(v)$ is the error at a vertex v and $\delta(v)$ is the distance from v to the nearest point on S. The error bound proof is sketched in the appendix. In the case of objects that are piecewise linear with a total of N facets, distance transform time complexity is $\mathrm{O}(F)$ for the initialization step and $\mathrm{O}(N)$ for the wavefront expansion. Our complexity bound highlights an advantage of decoupling octree construction and distance computation. The top-down approaches of [128] and [57] compute exact distances at each octree level for $\mathrm{O}(LF)$ where L is the maximum octree level. Our initialization step is the only step dependent on the number of object facets, and so if the number

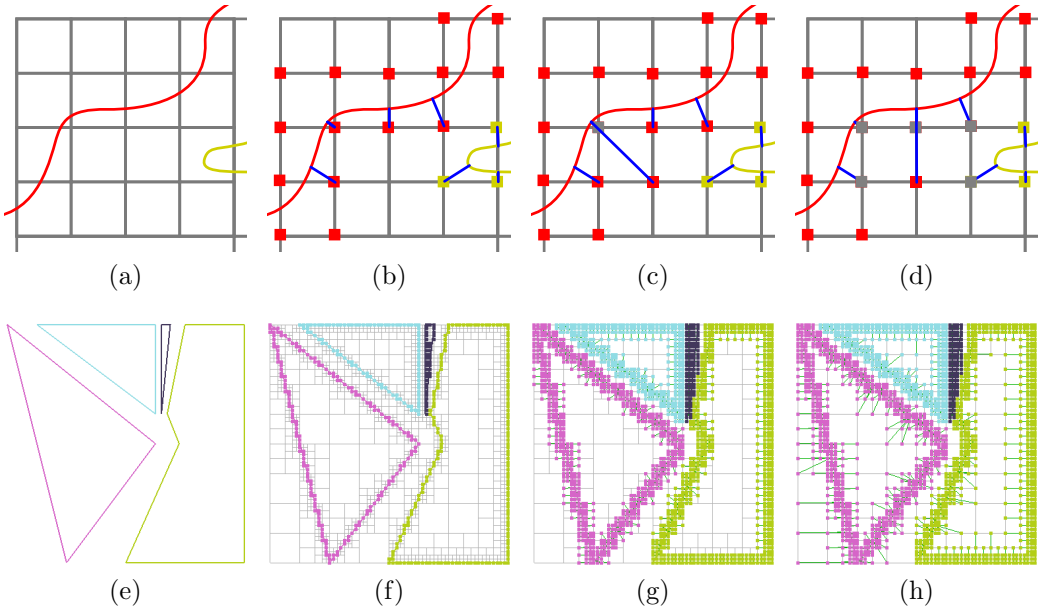


Figure 5.4: (a) Portions of the surfaces of two objects. (b) Every non-empty vertex (i.e. vertex adjacent to a non-empty cell) is assigned an exact closest point and added to the wavefront priority queue. (c) The top priority vertex (with the smallest distance) pushes its closest point to its neighbors. (d) The lower center vertex in red stores closest points to both the yellow and red objects. (e) - (h) Example of wavefront expansion over an entire space.

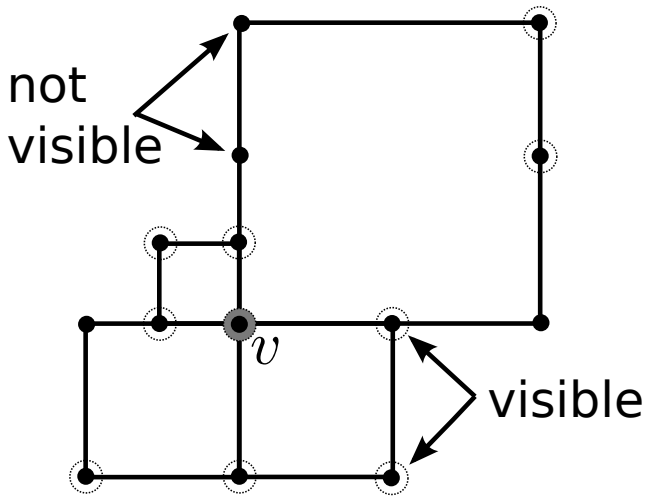


Figure 5.5: v is shaded and vertices that are visible from v are circled.

of facets is very high we gain significant speedup.

Distance field `COMPUTE_DISTANCES` finds distances on quadtree vertices. This labeled subdivision data structure can be used as a distance representation. That is, given any point we can quickly find an approximation of its distance from its closest site. If we desire to query our data structure for distances at arbitrary points (Figure 5.6) we should use the feature resolution predicate (Section 5.4) during octree construction and the octree should be stored as a hierarchy, which is better suited to point location queries (i.e. given a point p, find the octree leaf cell that contains p). The octree can either be stored hierarchically at the outset, requiring changes to the octree construction implementation, or it can be converted to a hierarchy from the adjacency representation.

The typical approach to computing the distance at a point given an ADF is to interpolate using distances of nearby vertices [57, 128]. This is advantageous in that vertices need store only a distance and not the actual closest point, and the distance field will be C^0 continuous. However, distance extrema and critical points will be limited to the octree vertices. Our approach is the following: given a point p and its containing leaf octree cell c, choose the closest point among the closest points of vertices on c. That is, $point[p] = \arg\min_{point[v \in c]} \text{dist}(p, point[v])$. While the distance field is not continuous, the benefit is that critical points can (and usually do) lie on the interiors octree cells.

5.6 Resolve ambiguities

Our distance function is built over a discretized space which yields ambiguities in the topology of the bisector. In Figure 5.7a, one cell has 4 edges that intersect the bisector. It is possible that there is a point inside the cell that is equidistant from 4 sites, but in practice it is unlikely. We use the following definition of an ambiguous cell. 0- and 1-dimensional cells are unambiguous. Let G' be a restriction of the octree vertex graph G to cell c. Collapse all edges in G' that have endpoints with identical labels. Call the new graph $\overline{G}$. A $(D > 1)$-dimensional cell c is unambiguous if $\overline{G}$ is a clique and all $D - 1$ faces are unambiguous. Ambiguous cells are subdivided recursively until the ambiguity is resolved or a threshold level is reached.

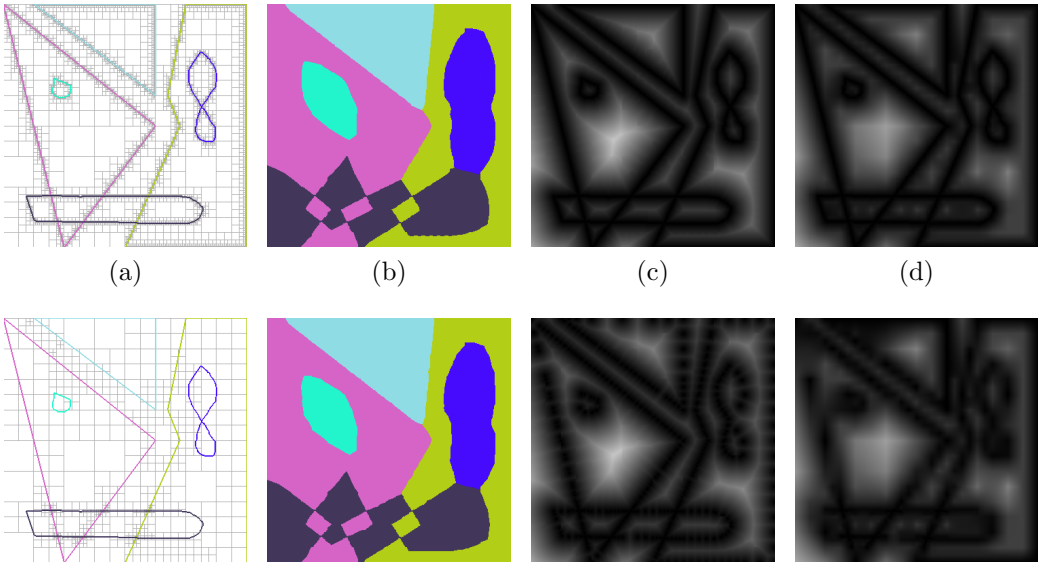
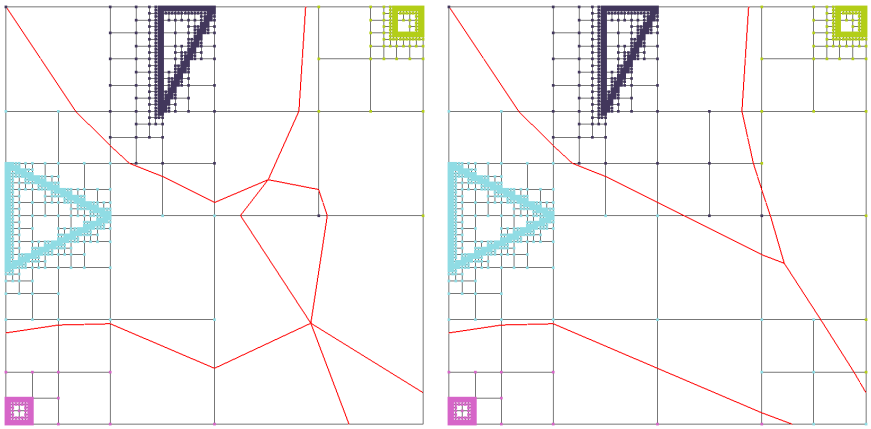
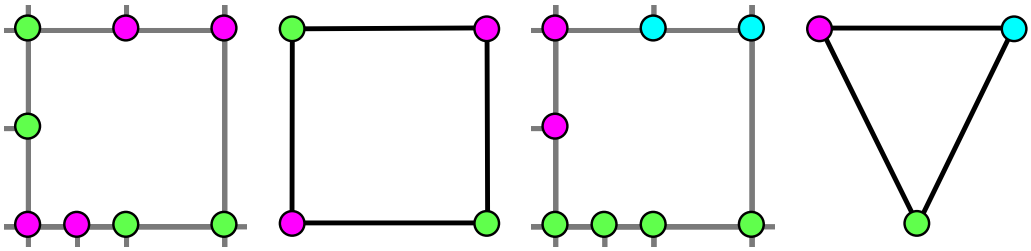


Figure 5.6: (a) The quadtree is subdivided to the limit for any leaf containing the surface. 7843 octree cells. (b) GVD. (c) Nearest point distance field. Critical points in cell interiors are preserved. (d) Interpolated distance field. This is the approach taken by [128] and [57]. We used Mean Value interpolants [55] in this example. Critical points and extrema are restricted to vertices. (e) The quadtree is subdivided only far enough so that there is a 1-cell buffer between objects. Note that cells are fully subdivided in areas of object-object intersections but not at self-intersections. 802 octree cells. (f) GVD. It is virtually identical to (b) despite using far fewer octree cells. (g) Nearest point distance field. (h) Interpolated distance field.



(a) (b)

Figure 5.7: In 2D, cells with more than 3 intersections are ambiguous and subdivided until a threshold is reached or the ambiguity is resolved. In 3D, ambiguities are detected using number of connected label components over a cell.



(a) (b) (c) (d)

Figure 5.8: (a) $G' = G$ restricted to c. (b) $\overline{G}$ is the collapsed version of G' as described in the text. It is not a clique, so the cell is ambiguous. (c) G'. (d) An unambiguous cell: $\overline{G}$ is a clique.

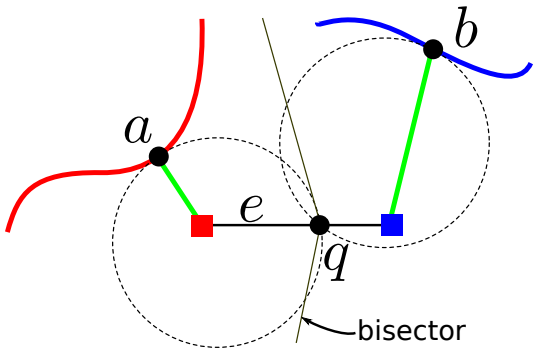


Figure 5.9: Computing the intersection of the bisector with a quadtree edge.

5.7 Compute GVD surface

With the distance function in place we compute the generalized Voronoi diagram, or the set of all points that have at least two closest points with differing labels. We store the GVD as sets of triangles, one set per object, representing the boundary of the GVC for that object. Each triangle is stored twice, once for each GVC is borders, with orientation pointing the triangle normal to the inside of the GVC.

We first compute which edges of the quadtree intersect the GVD by considering each quadtree edge e. If $\mathscr{L}(e_0) \neq \mathscr{L}(e_1)$ then e intersects the GVD. Let $a = p(e_0)$ and $b = p(e_1)$. Assume without loss of generality that e is horizontal at coordinate y. The vertical case is similar. We seek point $q = (x, y)$ such that $\text{dist}(a, q) = \text{dist}(b, q)$ (see Figure 5.9). In our euclidean setting this reduces to

$$x = \frac{2y(a_y - b_y) + b_x^2 - a_x^2 + b_y^2 - a_y^2}{2(b_x - a_x)} \tag{5.1}$$

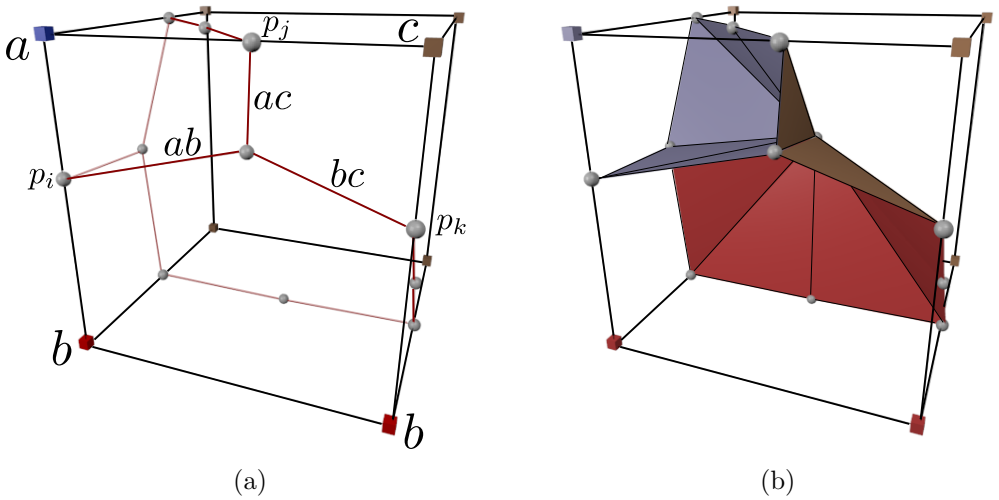
Once all edge bisectors for a cell are calculated we fit triangles (Figure 5.10a). Suppose bisector p_i lies on edge e^i. We connect bisector points on a 2D face f by creating edges between each bisector $p_i \in f$ and the centroid of all face bisectors $r_f = \sum p_i / |\{p_i\}|$. If the cell is unambiguous there will be at most 3 bisectors. In the case that the cell is subdivided to the lowest level without resolving ambiguity then the centroid r_f becomes an interface to 4 or more generalized Voronoi cells. We label each edge by the labels of the vertices adjacent to p_i, that is, $\mathscr{L}(p_i) = ab$ where $\mathscr{L}(e_0^i) = a$ and $\mathscr{L}(e_1^i) = b$.

Given a 3D cell c, we first find the edge complex of each 2D face c using the 2D algorithm and union all edges into a set E_c. We then form triangles from each edge $e \in E_c$ to the 3D centroid r_c of the bisectors (Figure 5.10b). Let triangle t_i have vertices (p_i, r_f, r_c) and let $\mathscr{L}(p_i) = ab$ and let T_a be the set of triangles assigned to the GVC of the object with label a. t_i is duplicated and added to both T_a and T_b. Let n_{t_i} be the normal of t_i and $v_{p_i}^a$ be the vertex adjacent to p_i such that $\mathscr{L} v_{p_i} = a$. If $n_{t_i} \cdot (v_{p_i} - r_c) < 0$ then we invert t_i before adding it to T_a.

5.8 Results

Our algorithm is designed primarily for memory savings, but it is has fast execution times as well. We report statistics and timings in table 5.1 for many of the examples shown in the paper.

We demonstrate our method and the GVD in three application settings: path planning, visualization of occluded structures, and exploded diagrams.



(a) (b)

Figure 5.10: GVD surface generation. (a) The 2D algorithm creates GVD edges from bisectors $\{p_i\}$ to the centroid. Each new GVD edge is given the two labels of the incident octree edge. (b) After finding the 2D GVD on its faces, the 3D cell fits triangles from 2D edges to the centroid. Each triangle is assigned to two sets of triangles, one for each label assigned to p_i.

dataset	objects	triangles ($\times 10^3$)	octree depth	cells ($\times 10^3$)	GVD (s)
Figure 5.3	4	65	8	6	1.7
Figure 5.11	470	5	24	158	2.0
Figure 5.12	35	1500	8	1043	63
Figure 5.13	11	100	24	106	35

Table 5.1: Table of statistics and timings. The *triangles* column gives the total number of triangles (edges in 2D) of all objects, *cells* gives the total number of leaf octree cells, and *GVD* gives the time to perform all steps of GVD computation. Number of voxels required to resolve all objects in a uniform gridding scheme is 2^{2n} where n is the octree depth.

5.8.1 Path planning

Planning a path for a robot or other mobile entity is a popular application of distance fields (e.g. [29, 134, 137]). A path that lies entirely on the GVD will have no intersections with objects. As the GVD is a reduction of the space by one dimension, using it as the search space an attractive optimization. Our GVD computation finds paths in environments that would quickly cause uniform grids to run out of memory. Domains with large numbers of objects or objects that are closely spaced require small pixels or voxels relative to the size of the space. Our adaptive method resolves tight packing with ease in both 2 and 3 dimensions. Figure 5.11 shows an example in 2 dimensions. We computed the GVD for 400 objects, with the quadtree reaching level 24 in order to resolve the smallest spacings. Quadtree and bisector graph computation took 1.3 seconds with the quadtree reaching level 24 and 140,680 cells (a uniform grid would require 2^{48} pixels to resolve the closest spacings). We then ran Dijkstra's shortest-cost path algorithm on the bisector graph (or GVD boundary complex) to compute the shortest cost path.

5.8.2 Occluded structures

Structures that are occluded from view by surrounding objects are often visualized using cut planes or by manually removing obstructing objects. Using the dual graph of the GVD we can quickly and easily visualize an object of interest with or without nearby objects. The dual graph is constructed by mapping objects to dual vertices; shared interfaces between GVD cells become

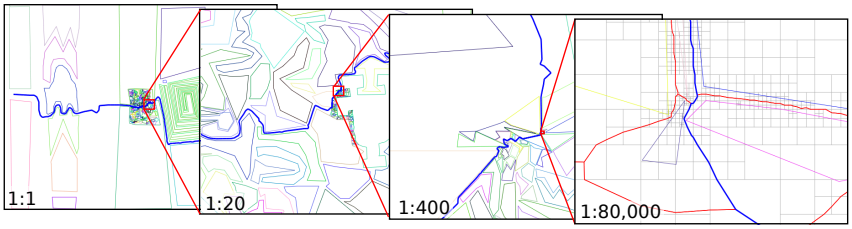


Figure 5.11: Path planning in 2D. We built a quadtree over hundreds of objects ranging in size and spacing over orders of magnitude. The quadtree reached level 24 before the closest spacings were resolved. The shortest-cost path between two points is shown in blue. The right-most figure shows the quadtree in gray and GVD boundary complex in red at 80,000x magnification.

edges. Thus, two dual vertices share an edge if their corresponding GVCs are adjacent. The topology of the dual graph has an identical relationship to the GVD as the Delaunay triangulation has to the ordinary Voronoi diagram. To visualize a given object the user controls a threshold parameter $t \in \mathbb{Z}^*$ and only objects in the $(i \leq t)$-ring of the object in the dual graph are rendered.

5.8.3 Exploded diagrams

An effective way of visualizing multiple objects in close proximity is to "explode" the objects away from each other in a meaningful way that retains some of the spatial coherence of the collection. Exploded diagrams are typically used in CAD applications, but their usefulness extends to other applications, such as molecular modeling. Virus molecules are composed of hundreds or thousands of atoms, often in symmetric structure that can be meaningfully segmented into constituent collections of atoms. Ofttimes many regions are occluded from view. Furthermore, molecules are often layered radially from

the center, forming shells. In the case of radial layering one need only translate objects radially away from the center, but this approach isn't effective if we wish to explode objects away from an anchor object that is not near the center. The same challenge is encountered in unstructured data. A naive approach to creating an exploded diagram is to choose a primary object C and move all objects along a vector derived from the object centroids. That is, given a object D, move $D \leftarrow D + \lambda(\mathscr{C}(D) - \mathscr{C}(C))$ where $\mathscr{C}(C)$ is the centroid of C and λ is an explosion constant. This approach works fine for objects that are roughly spherical, but in cases where objects are more complicated, the centroid of D may lie in a very different direction relative to $\mathscr{C}(C)$ than where D should intuitively travel. This is the case in Figure 5.12a.

Our approach is to utilize the GVD boundary complex to compute directions of travel for each object. Let C be the anchor object and let K be the dual graph of the GVD as described in Section 5.8.2. Further, define T_{CD} as the set of triangles adjacent to both GVC_C and GVC_D with normals oriented toward GVC_D. $\mathscr{R}_C(i)$ is the set of objects in the i-ring of C and $A_{CD} = \sum_{t \in T_{CD}} A_t$ is the summed areas of triangles in T_{CD}. Using graph K, if $D \in \mathscr{R}_C(1)$ then $D \leftarrow D + \mathscr{D}_1(D)$ where

$$\mathscr{D}_1(D) = \frac{\sum\limits_{t \in T_{CD}} n_t A_t}{A_{CD}}$$

Since the $(i > 1)$-ring neighbors of C have no direct interface to C they are moved in an average direction of the $(i-1)$-ring neighbors that are

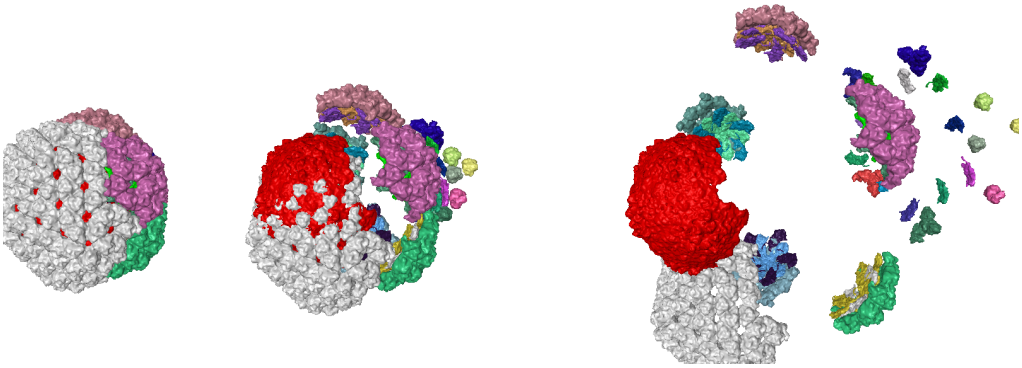
adjacent to D. That is, $D \leftarrow D + \mathscr{D}_i(D)$ where

$$\mathscr{D}_i(D) = \frac{\sum_{N \in \mathscr{R}_c(i) \cap \mathscr{R}_D(1)} \mathscr{D}_{i-1}(N) A_{DN}}{\sum_{N \in \mathscr{R}_c(i) \cap \mathscr{R}_D(1)} A_{DN}}$$

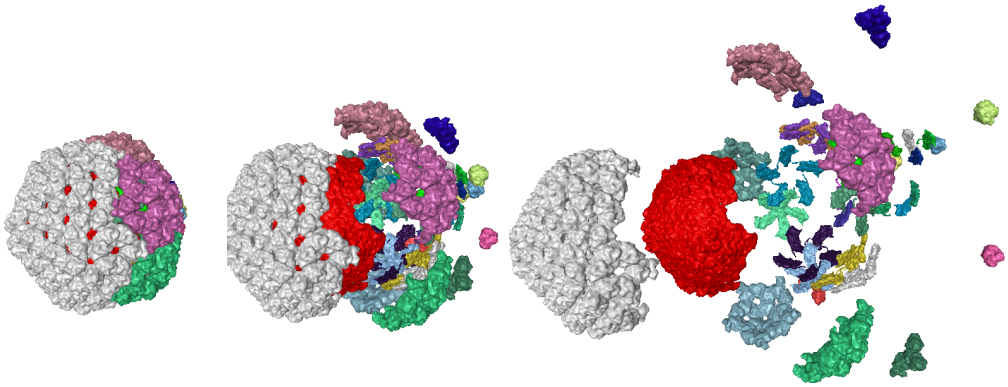
5.9 Conclusions

We have presented and demonstrated effectiveness of a novel adaptive distance transform and GVD algorithm, which include an octree subdivision algorithm and data structure suitable for the GVD. We have also shown, in addition to the popular motion path planning, important applications of the GVD in visualization. This raises the question of what other applications might benefit from a GVD algorithm specifically tailored to large sets of objects that are closely spaced. Our algorithm is general; while our implementation supports polyhedra, higher-order objects are equally as valid, and since initialization of the distance transform's wavefront is the only step dependent on the complexity of the objects, our approach is particularly suited to objects for which a point-object distance computation is expensive.

Our algorithm, much like feature-based subdivision schemes that become costly with large numbers of features, becomes expensive when the number of objects is large and object spacing is not small. As it is optimized for the GVD, our algorithm is not as accurate as other methods for distance field queries at arbitrary locations. That said, an advantage of our method is that it stores the closest object points, enabling critical points at locations other

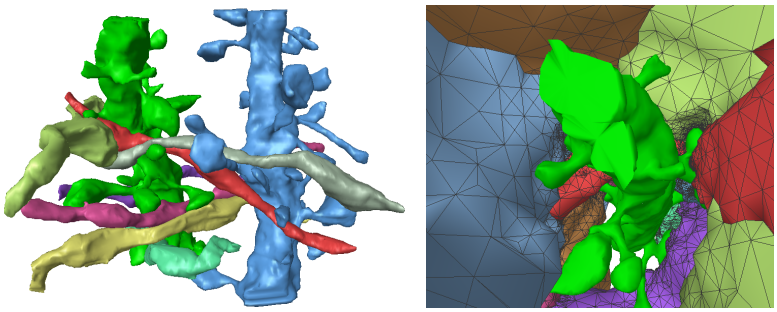


(a)



(b)

Figure 5.12: Explosion diagrams of a virus with symmetry and radial shelling. (a) The vectors objects travel along are computed using object centroids. Objects travel in non-intuitive directions. (b) Travel vectors are computed using triangles of the GVD boundary complex. The directions of travel are intuitive and separate the objects in a meaningful way.



(a) (b)

Figure 5.13: (a) Mammalian brain neurons are composed of dendrites and axons. This figure shows two vertically-oriented spiny dendrites with six nearby axons. (b) The inside of the green dendrite's generalized Voronoi cell. Boundary regions of the cell inherit the color of the opposite cell's object.

than octree vertices.

We are interested in deriving a tighter error bound on the vertex distances, which we conjecture to exist. We are also interested in alternative approaches to exploded views that expand yet more intuitively. We also intent to pursue a power diagram-like weighted generalized Voronoi diagram, which is suited to our framework.

Chapter 6

VolRoverN: Software for Modeling Neuronal Ultrastructure

In the previous chapters, we presented algorithms that further enable 3D modeling of neuronal ultrastructure. We have implemented these and other published algorithms in a software package called VolRoverN. In this chapter we describe VolRoverN, compare it to similar software and show examples of its capabilities. After a brief introduction (Section 6.1), we describe the functionality (Section 6.2), and give qualitative validation results (Section 6.3). We then discuss the impact that VolRoverN can potentially have on modeling of neuronal ultrastructure (Section 6.4).

6.1 Introduction

A number of software packages have been developed to support various tasks in cellular reconstruction including image alignment (RECONSTRUCT [53, 105], TrakEM2 [31]), image segmentation (RECONSTRUCT, TrakEM2, ilastik [123], NeRV [83], NeuroTrace [79]), and connectivity and hierarchy management (TrakEM2). While each of these tools accept EM images as input, provide image segmentation functionality, and create and export surface rep-

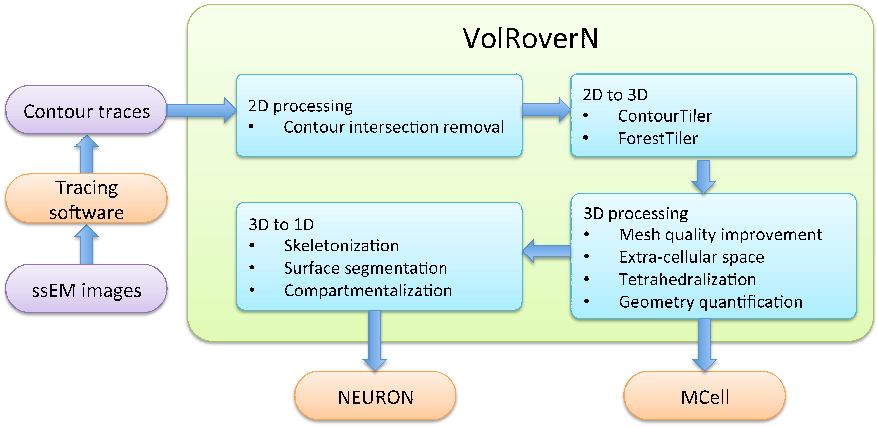


Figure 6.1: A high-level look at VolRoverN's functionality. There are four main phases: 2D processing, 2D to 3D reconstruction, 3D processing, and 3D to 1D reduction.

resentations (except NeuroTrace which uses volume rendering and does not create a surface mesh), the reconstructions are primarily created for visualization purposes only and are generally of insufficient quality to serve as a framework for dynamical simulations.

VolRoverN accepts as input the aligned and segmented images and contour tracings from existing software tools, and automatically generates reconstructions that are physiologically plausible and formatted for easy input into other software tools for simulation of neural dynamics. We begin by describing the functionality of VolRoverN, including 3D morphological reconstruction of neuropil and producing of derivative representations. We enumerate common errors in surface reconstruction and demonstrate VolRoverN's ability to produce error-free, quality reconstructions, including comparison to reconstructions produced using similar software.

6.2 Functionality

VolRoverN is freely downloadable at `http://cvcweb.ices.utexas.edu`. It is currently available on the Mac OS X platform, and we anticipate release for Linux and Windows platforms. With the VolRoverN download is a sample dataset with contours and images of 8 axons and 2 dendrites in the CA1 region of the hippocampus. All images in this chapter were produced using this dataset. A shared data respository will also be available where users of VolRoverN can share images, traces, 3D meshes, and simulation files for NEURON and MCell.

VolRoverN accepts RECONSTRUCT and TrakEM2 contour tracings as input. In the case of TrakEM2, the tracings are pixel-based and are automatically converted to polygonal representation by VolRoverN. Aligned and possibly segmented images can also be imported into VolRoverN for visualization purposes.

The software first fits a triangulated surface to contours such that the contours are exactly interpolated and the surface meets important quality criteria. We list and show examples of violations of these criteria in Figure 2.2. Properties of quality reconstructions include watertightness, manifoldness, lack of intersections, quality (close to equilateral) triangles, and geometric accuracy. With the surface mesh in place the user can make geometric queries, such as surface area and volume of a spine head. Further, VolRoverN provides tools to create derivative models, including 1D cable models. The various models can be saved in standard file formats, including Wavefront obj, OFF, ele/node,

MCell, and NEURON.

VolRoverN shares a code base with the related software package VolRover 2.0 [145] which performs image processing and visualization of molecular EM and includes 3DEM structure identification and model-based refinement. As such, VolRoverN and VolRover 2.0 have similar look and feel, but the tools included in VolRoverN are appropriate for neuronal modeling.

VolRoverN has 4 steps in producing models suitable for analysis (Figure 6.1). 1) Process 2D input. 2) Fit a 3D triangulated surface to the contours. 3) Process the 3D surface meshes, which includes improvement of the mesh. 4) Reduce the mesh to a 1D cable model. We now discuss each of these steps.

6.2.1 2D processing and 2D to 3D

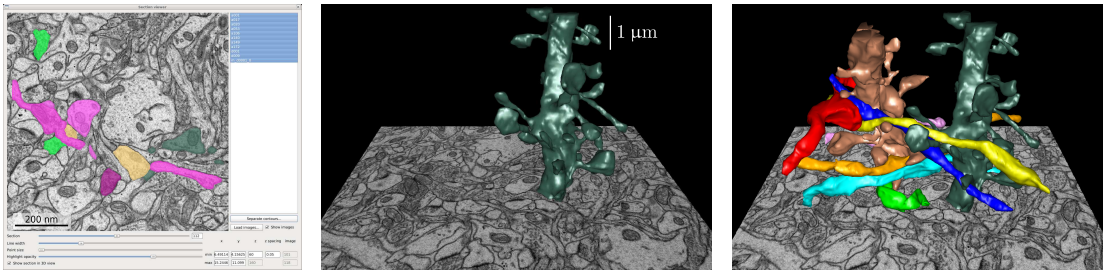
We have implemented the 2D contour intersection removal algorithm described in Section 2.4.1 (Figure 6.2; see also Figure 2.5) that not only ensures that contours don't intersect each other, but has an optional feature of guaranteeing a minimum separation distance. This is helpful especially with neuronal data since minimum spacing between cells is usually known [90].

VolRoverN also includes a tool called ForestTiler which is an implementation of the reconstruction and intersection removal algorithms described in Chapter 2 [11, 48]. Input contours are fitted with a surface mesh that is free of intersections with other objects (Figure 6.3). Like 2D contour curation, VolRoverN accepts a user-specified minimum spacing between 3D objects.

Standard algorithms such as marching cubes yield large numbers of

(a) (b)

Figure 6.2: 2D curation. Because components are usually traced independently of each other, intersection errors can occur. (a) A number of intersections and close approaches can be seen between contours. (b) The intersections have been removed and a specific contour spacing is enforced.



(a) (b) (c)

Figure 6.3: Reconstruction from 2D contours. (a) Input to VolRoverN is a set of 2D polygonal traces, or contours, derived from EM images. (b) Software embedded in VolRoverN called ContourTiler fits a triangulated surface to each set of contours to produce a 3D surface model. (c) Multiple components are combined using ForestTiler such that they are free of intersections.

intersections between objects when the objects are tightly packed (see Figure 6.10). Additionally, marching cubes produces blocky reconstructions due to its lack of interpolation between contours. This is especially evident with neuronal EM imagery, which is highly anisotropic. Our algorithm resolves both problems, producing surface meshes that are intersection-free and linearly interpolated between contours.

Additional qualities of ForestTiler meshes include watertightness, oriented normals, regularity, manifoldness, topological correctness, and interpolation of the original contours.

6.2.2 3D processing

Mesh quality improvement VolRoverN includes a suite of tools to produce meshes with good quality triangles, i.e. triangles that are close to equilateral. The first tool is decimation, which uses the QSlim algorithm [60] to reduce the number of triangles. QSlim is an edge-collapse algorithm that is popular because of its speed and robustness. We then use the geometric-flow mesh improvement algorithm of Zhang et al [147, 148] that produces a surface mesh with triangles of good aspect ratio (Figs 6.4a-6.4b). The mesh improvement algorithm can be iteratively applied for increasing triangle quality. Quality improved meshes are no longer guaranteed to meet the contour interpolation property. VolRoverN also has mesh repair utilities (Figs 6.4c - 6.4d) to repair errors such as holes, non-manifoldness and self-intersections.

Additional mesh repair and mesh improvement tools can be applied to

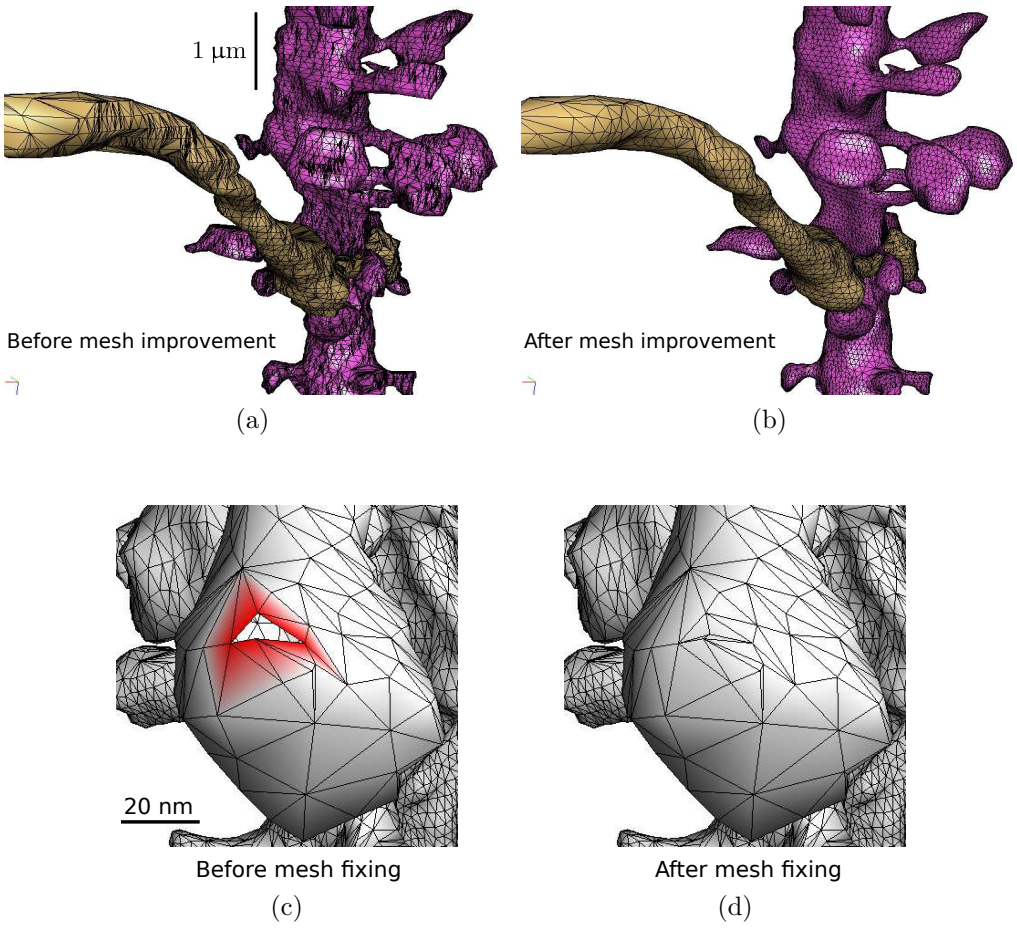


Figure 6.4: Mesh improvement. (a) The original reconstructed triangulation of a dendrite and axon. (b) The reconstruction after decimation and smoothing. The final triangulation has fewer than half the triangles as the original and the triangles have far better aspect ratio. (c) Repair utilities in VolRoverN include manifold correction and hole filling. The figure shows a hole created by removal of non-manifold edges. (d) After hole filling. The before/after ratio of total mesh surface area in this example was 32.7/32.8, for a total hole surface area of 0.3%.

our surface meshes by exporting to standard Wavefront obj or OFF files and importing into other mesh software.

Extra-cellular space and tetrahedralization A feature of VolRoverN is automatic extra-cellular space (ECS) generation from surface meshes. The user defines a bounding box and the ECS tool constructs a model of dual space, that is, a closed polyhedron with ECS in the interior (Figure 6.5). Models of ECS have been used in reaction-diffusion simulations [89, 90].

VolRoverN also has the capability of tetrahedralizing surface meshes [12, 150, 151] (Figure 6.5c). This uses an adaptive subdivision meshing algorithm contained in a library from the Level Set Boundary-Interior-Exterior (LBIE) software package [38]. Tetrahedra are exported in RAW and ele/node formats for ease of import into simulation packages such as STEPS [68].

Geometry quantification VolRoverN reports surface area and volume of regions of a reconstructed neurite. After surface segmentation, which is done when creating a cable model as reported in the 3D to 1D section, the user can discover geometric measurements of segments (e.g. Figure 6.6b) by clicking on the region.

MCell MCell [88, 127] is a software package that simulates multi-ion species reaction-diffusion using Monte Carlo algorithms over geometrically complex domains (Figure 6.7). The MCell export tool in VolRoverN writes a given

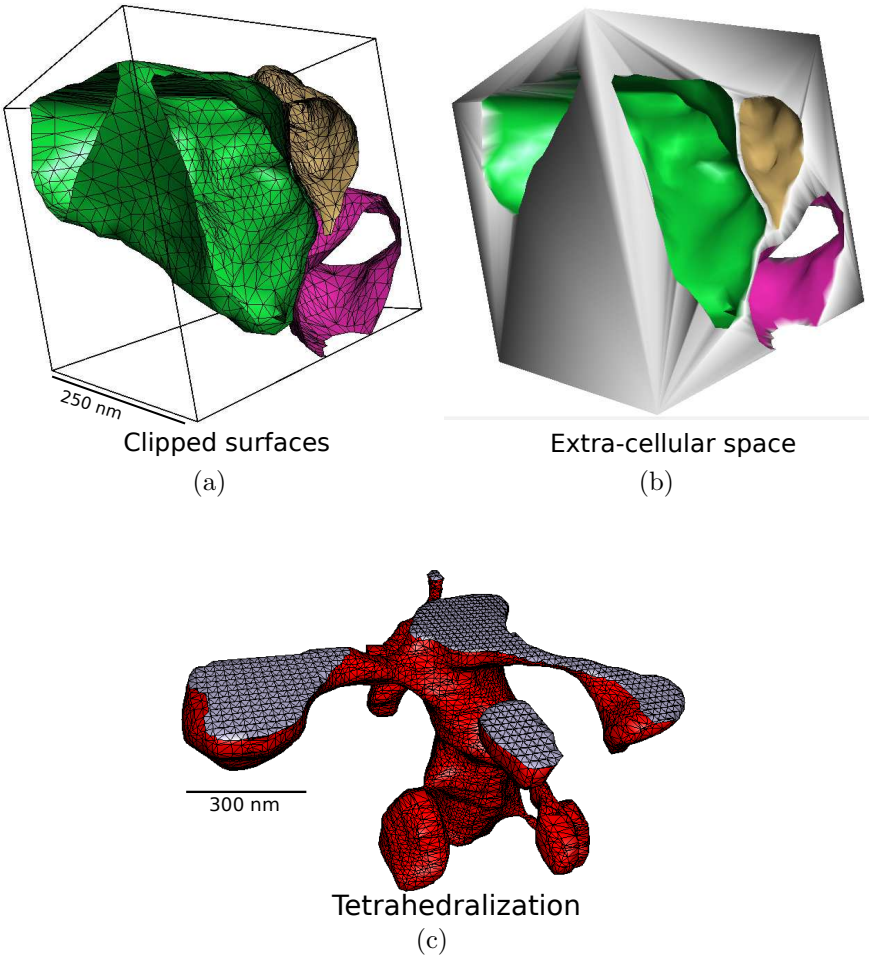


Figure 6.5: VolRoverN utilities. (a) A bounding box is placed around the surface meshes in an area of interest and the surface is clipped. (b) VolRoverN's ECS tool creates a closed polyhedron with ECS in the interior. (c) Tetrahedralization of a dendrite using VolRoverN's tetrahedralization tool.

surface mesh to an MCell MDL file. As noted, these meshes are required to be water-tight, manifold, and free of self-intersections which can be repaired, if necessary, using VolRoverN's mesh repair utilities.

6.2.3 3D to 1D

Cable model simulation requires 1D skeleton models of neurons. Cable models are typically created with neurite tracing software, but VolRoverN utilizes surface meshes to generate 1D models automatically.

VolRoverN decomposes the mesh into cylindrical chunks using the algorithm described in Chapter 4. The model description is then output to a NEURON [32] hoc file. NEURON is a simulation software that implements multi-compartment models of electrical signaling based on cable theory. The NEURON hoc file contains length and diameter properties for each region as well as connection properties defining their topological connectedness. It also contains a skeleton simulation function.

6.2.4 Additional tools

VolRoverN has a 2D image and contour display called the Section Viewer (Figure 6.3a) that enables navigation through sections while inspecting contours. The Section Viewer and 3D Viewer are linked: imagery and semi-transparent contours can be visualized in the 3D view alongside surface meshes and volumes (Figs 6.3b and 6.3c).

The signed distance function (SDF) in VolRoverN is a tool that pro-

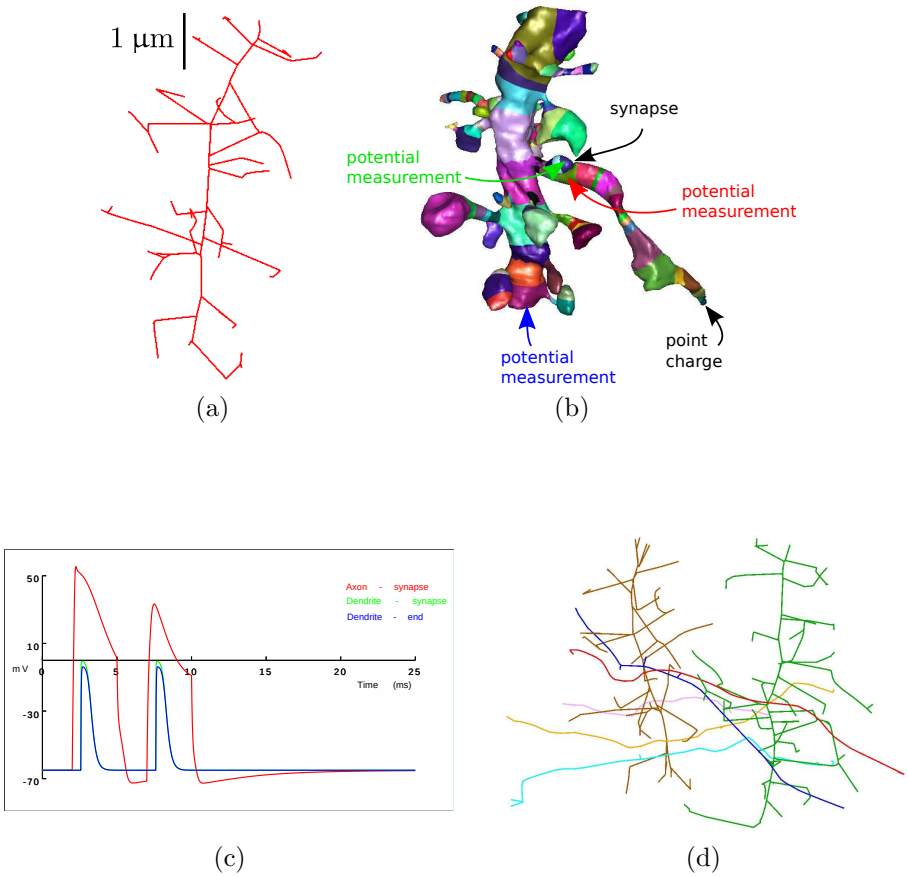
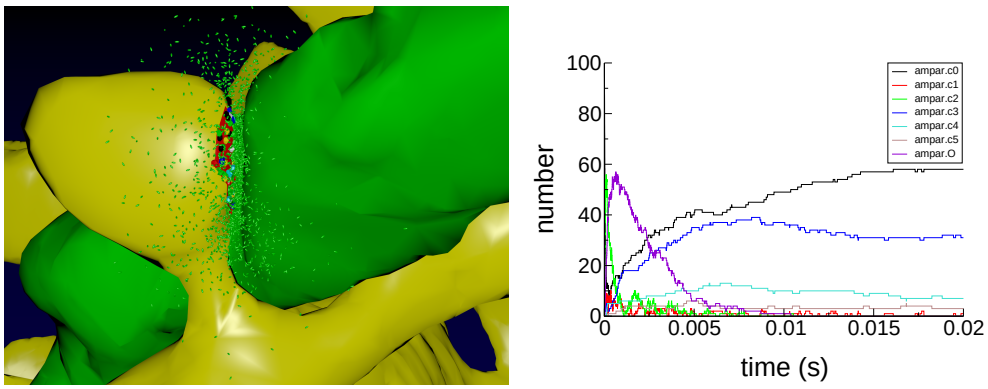


Figure 6.6: Multi-compartment model generation. Our surface segmentation first skeletonizes the mesh (a), which induces a segmentation (b). Each segment is in a different color in the figure. After correction, the segmentation can be used to produce surface area/volume statistics of different regions as well as labeling different regions for ion diffusion studies. This graphic shows a simple cable model simulation. The compartmentalized versions of the axon and dendrite are input to NEURON. A synapse with a threshold of -20 mV and delay of 0.5 ms is added between the dendrite and axon and point charges of 0.05 amps are placed at the end of the axon at 2 and 7 ms for 3 ms each. Potential measurements at three locations are made over time. Arrow colors correspond the potential measurements reported in the NEURON simulation graph in figure (c). (d) A view of a skeletonization of all axons and dendrites in the sample dataset (see Figure 6.3c). Skeletons can be saved in OFF and raw file formats.



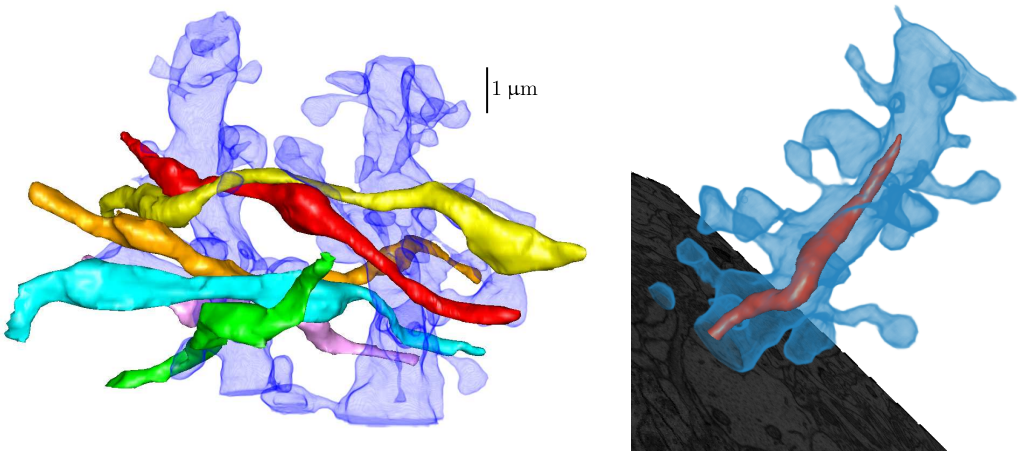
(a) (b)

Figure 6.7: MCell reaction/diffusion simulation of synaptic transmission from generated model. Generated meshes of axon (green) and dendrite (yellow) were imported into CellBlender to create an MCell simulation from the meshes. Images were rendered using CellBlender. (a) Visualization of synaptic transmission 100 microseconds after release of 2000 molecules of the neurotransmitter glutamate (small green ellipsoids). 10 NMDA receptors (NMDAR) and 100 AMPA receptors (AMPAR) were placed at the synaptic contact area between the axon and dendrite (small red patch of membrane on the dendrite). Color indicates state of activation of the receptors. At 100 microseconds, the glutamate has started to bind and activate some receptors and has started to spill out of the synaptic cleft space into the surrounding volume. (b) Time course of activation of AMPARs. AMPAR can be in 7 states: c0 (unbound state), c1 (one glutamate bound), c2 (two glutamate bound), c3 (one glutamate bound, desensitized state 1), c4 (two glutamate bound desensitized state 2), c5 (two glutamate bound, desensitized state 3), and O (two glutamate bound, ion channel open).

duces a volume of scalar values representing distance from a surface. SDF, together with VolRoverN's volume rending capabilities (Figure 6.8) and isocontouring tools, enables interactive exploration of topology and geometry at various isovalues. VolRoverN renders volumes (stored in HDF5 format) and geometries together seamlessly. The transfer function tool supports both color and transparency ramps across the spectrum of level-set values in a volume. The fast isocontouring tool [14] enables smooth and intuitive exploration of different isocontours (Figure 6.9). Isocontours are computed from surfaces produced by ForestTiler. A supporting tool is the contour tree [33, 146] which reveals topological connectedness of isosurfaces across isovalues [111]. The contour spectrum [15] gives additional insight, showing curves representing the signature of surface area, volume and gradient across isovalues (Figure 6.9e).

6.3 Validation

To validate the surface reconstruction algorithms in our software, we compare results from VolRoverN with two other reconstruction algorithms: marching cubes, which is used in most popular contouring software packages [31, 83, 123] and the Boissonnat algorithm [24], which is implemented in RECONSTRUCT. We use TrakEM2's implementation of marching cubes. We reconstructed the 8 axons and 2 dendrites in the sample dataset distributed with VolRoverN using both RECONSTRUCT and VolRoverN and compared the results by quantifying the most common errors of those described in Figure



(a) (b)

Figure 6.8: Volume rendering with geometry rendering. (a) Axons rendered with semi-transparent dendrites. (b) A dendrite is rendered with semi-transparent volume rendering to reveal mitochondria. ForestTiler naturally supports nested components.

2.2. Table 6.1 reports comparison results showing that VolRoverN produces meshes superior both in terms of errors and triangle quality.

VolRoverN meshes are free of intersections between multiple objects, in contrast to surfaces produced by marching cubes and the Boissonnat algorithm. Figs 6.10a-6.10d show reconstruction of two axons in close proximity (a001 and a020 from the sample dataset) using the three reconstruction methods. The RECONSTRUCT and TrakEM2 representations yield a large number of intersections, whereas the VolRoverN surfaces are entirely free of intersections. Further, VolRoverN meshes are guaranteed to have a user-specified minimum spacing between objects.

Meshes produced by VolRoverN interpolate, or pass exactly through,

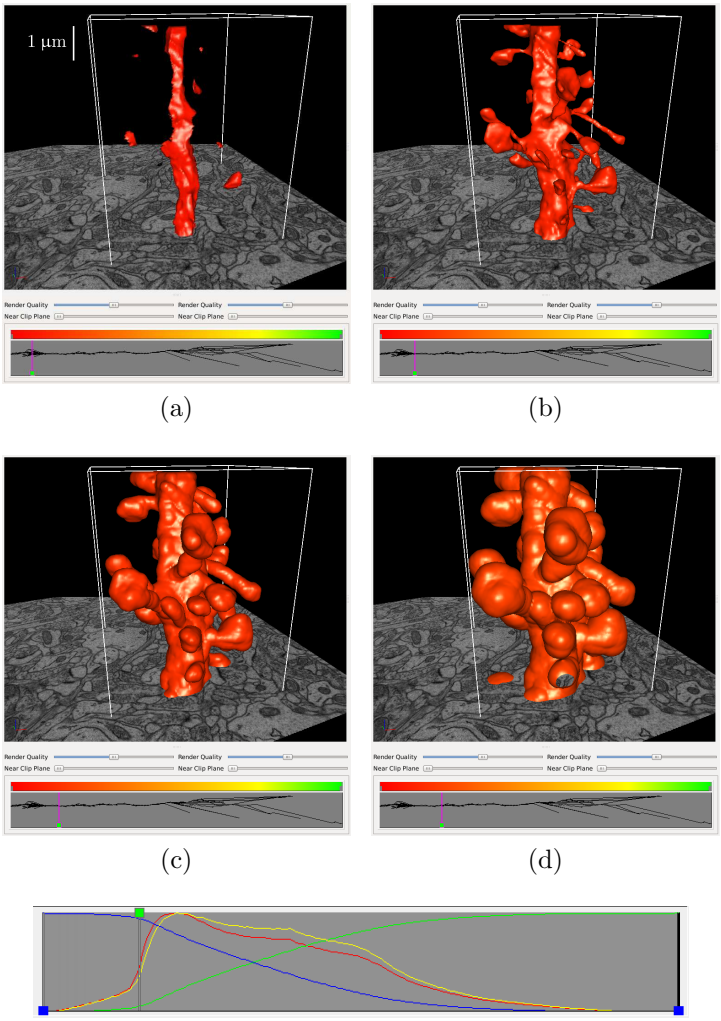


Figure 6.9: Isosurfaces of a dendrite at different isovalues. Isosurfaces are computed from surface geometries. The contour tree at bottom shows the topological branching structure of the isosurface. The vertical line in the contour tree shows the isovalue of the surface relative to the tree. Figure (b) is close to the true surface, as at that isovalue the contour tree is a confluence of branches into one. (e) The contour spectrum tool in the transfer function tool. Four attribute curves are shown: surface area (red); min volume (green); max volume (blue); gradient (yellow). The green isovalue node is close to an area of high gradient.

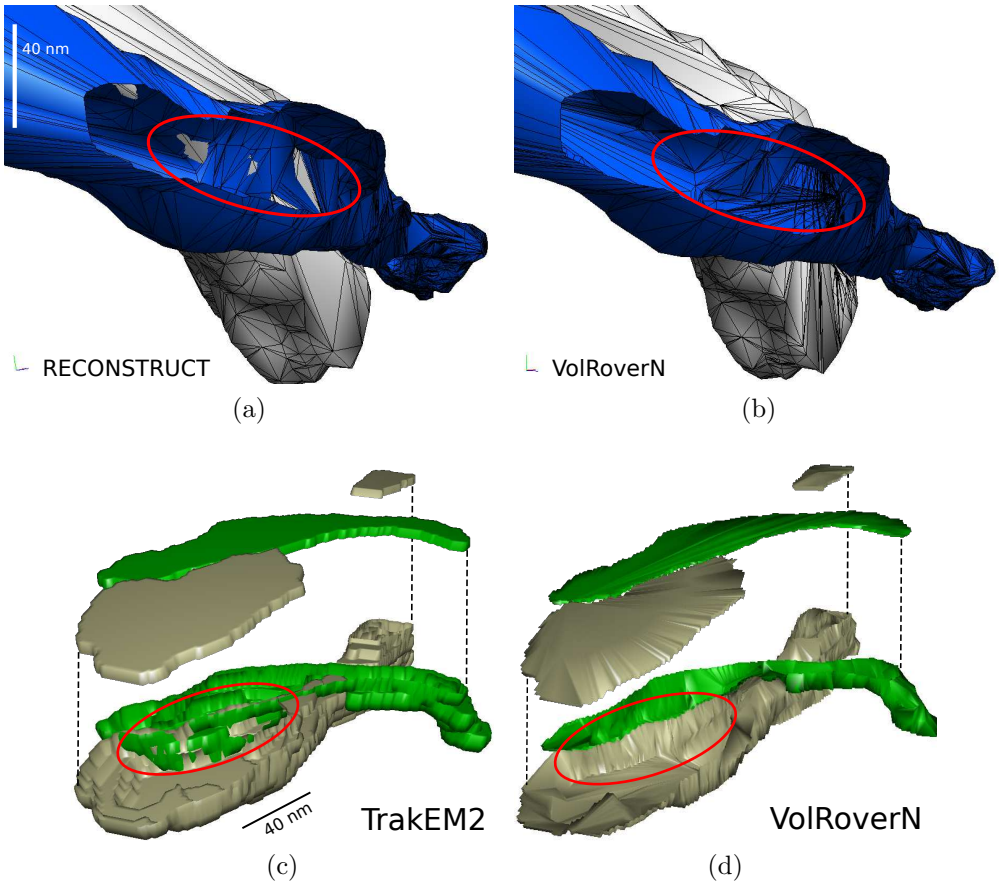
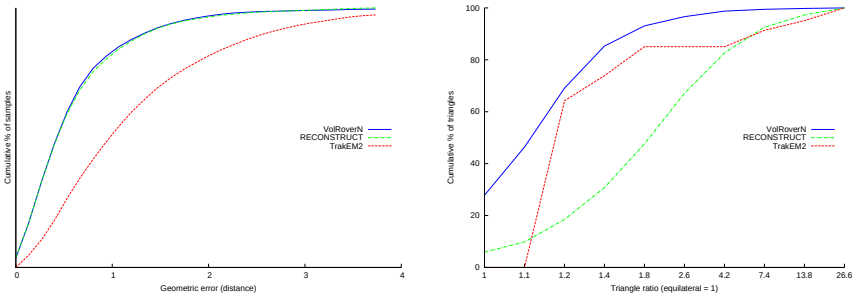


Figure 6.10: Intersection comparisons with RECONSTRUCT and TrakEM2. (a)-(b) Comparison between RECONSTRUCT and VolRoverN surfaces using axons a001 and a020 from the sample dataset. Part of a020 is cut out to see the interior intersections. The RECONSTRUCT surfaces yield a large number of intersections between objects. Output from ForestTiler is intersection-free. (c)-(d) Comparison between TrakEM2 and VolRoverN surfaces. A portion of two axons are reconstructed with TrakEM2's marching cubes implementation and the top is lifted to reveal the interior. While the triangles are of reasonably good quality and the surfaces are manifold and free of holes, there are numerous intersections between objects that are labor intensive to correct. Output from ForestTiler is intersection-free.



(a) (b)

Figure 6.11: Error and quality comparisons with RECONSTRUCT and TrakEM2. (a) Geometric error compared to a C^1-continuous approximating surface. This is a cumulative plot of percentage of samples within a given error. The great majority of samples from VolRoverN and RECONSTRUCT have small geometric error (measured as distance to the nearest point on the C^1-continuous surface). A larger number of samples from TrakEM2 have large error. The same number of samples were taken from the three surfaces. To create the C^1-continuous surface S_{C1}, we randomly choose 4 adjacent, non-bifurcating contours (called $c1, c2, c3$, and $c4$) and fit cubic B-splines to each of them using a least-squares fit. We then join the contours together with interpolating cubic curves, forming a patch that is C^1-continuous everywhere between $c1$ and $c2$. We then sample 100K points randomly between the $c1$ and $c2$ sections on the VolRoverN, RECONSTRUCT, and marching cubes surfaces (S_{ct} and S_{mc}, respectively) and find the distance from each sample to S_{C1} using the Newton-Raphson method. The data used in this test are from axon a001 (distributed with VolRoverN sample data) between slices 116 and 117. Contours were produced using TrakEM2 and were fitted using ContourTiler, RECONSTRUCT, and the marching cubes implementation in TrakEM2. (b) Comparison of quality of triangles between the three reconstruction methods. We define triangle ratio as $r_c/2r_i$ where r_c is the circumradius and r_i is the inradius of a triangle. The ideal triangle ratio, or the ratio of an equilateral triangle, is 1. The plot is a cumulative percentage of triangles below a given ratio. Statistics come from each method's reconstruction of a001. VolRoverN and RECONSTRUCT use contours traced in RECONSTRUCT and TrakEM2 uses its own tracings.

the input contours. In this sense, VolRoverN meshes are error-free, as are other methods such as marching cubes. To quantify surface error in regions between contours we compare surfaces produced by VolRoverN, RECONSTRUCT, and TrakEM2's implementation of marching cubes to a C^1-continuous surface fitted to the contours. The distribution of errors is reported in Figure 6.11a and shows that VolRoverN's reconstructions are geometrically very similar to RECONSTRUCT and are closer to a smooth approximation than marching cubes.

Triangle quality is an important measure of how successful a simulation will likely be in terms of error convergence. We compared VolRoverN's quality improved meshes with RECONSTRUCT and TrakEM2 meshes and report the results in Figure 6.11b. VolRoverN outperforms both other methods in terms of triangle aspect ratio (ratio of the circumradius to twice the inradius). Table 6.2 shows statistics of another measure of triangle quality: minimum and maximum angle. Triangles in VolRoverN reconstructions have angles closer to 60° in every measure but one.

To show that the reconstruction is useful for more than just visualization, we used VolRoverN's 3D to 1D tools to automatically reduce the surface representation to a multicompartmental cable model and simulated ion channel-driven dynamics of membrane voltage in NEURON (Figure 6.6).

VolRoverN also enables MCell simulations. VolRoverN's ForestTiler and mesh improvement tools were used to generate surface triangulations of an individual axon and dendrite from labeled ssTEM traces. The quality

	Holes	Non-man vertices	Non-man edges	Int tris
VolRoverN	0	0	0	0
RECONSTRUCT	3	154	88	9
TrakEM2	0	0	0	0

Table 6.1: Table comparing errors between VolRoverN, RECONSTRUCT and TrakEM2 surface meshes. Object a001 in the sample dataset was reconstructed (full reconstruction by VolRoverN and RECONSTRUCT, and partial reconstruction by TrakEM2). The common errors compared here are number of holes, number of non-manifold vertices, number of non-manifold edges and number of intersecting triangles. VolRoverN and TrakEM2 surfaces are error-free.

meshes were then exported as MCell MDL geometry files. A complete physiological simulation study was set up with the CellBlender software (`http://www.mcell.org`) using the MCell MDL geometry files. The geometric analysis tools in CellBlender confirmed that the imported neurite MDL meshes were of computational quality for use in simulations. CellBlender was then used to generate and run an MCell simulation of glutamatergic synaptic transmission at a synapse between the axon and dendrite (see Figure 6.7a). Figure 6.7b shows the time course of activation of synaptic receptors by diffusing neurotransmitter molecules released at the synapse.

The algorithms used in VolRoverN are scalable. Our reconstruction method linearly interpolates between sections, so only two sections need be stored in memory at one time, thus our memory requirements remain static with increasing stack size.

ForestTiler and associated mesh improvement tools in VolRoverN are

	Min angle	Avg min angle	Max angle	Avg max angle
VolRoverN	1.8	35.8	174.2	86.8
RECONSTRUCT	0.12	17.8	173.3	103.8
TrakEM2	0.23	32.6	160.3	90.4

Table 6.2: Table comparing triangle quality between VolRoverN, RECONSTRUCT and TrakEM2 surface meshes. Object a001 in the sample dataset was reconstructed (full reconstruction by VolRoverN and RECONSTRUCT, and partial reconstruction by TrakEM2). The quality statistics show min (resp. max) angles across all triangles as well as an average of the min (resp. max) angle of each triangle. Min and max angles should be as close to 60° as possible.

efficient. We tested reconstruction time on the sample dataset, which consists of portions of 8 axons and 2 dendrites with a combined 129.32 μm^2 surface area, 8.44 μm^3 volume and 204906 triangles. Reconstruction took 6 minutes and 40 seconds on a standard desktop computer. Decimation, triangle improvement and mesh fixing tools took an additional 56 seconds.

All rendered figures in this chapter were produced using VolRoverN except plots and the MCell simulation which used CellBlender (Figure 6.7a).

6.4 Discussion

Rapid, large-scale reconstructions enable a comprehensive taxonomy of neuroanatomy at the subcellular level. By directly measuring the geometry observed in the reconstructions one can begin a straightforward statistical sampling of the underlying distributions of cellular surface area and volume

[106] in the brain. Having large amounts of reconstructed data will make it easier, for instance, to find the structural correlates of learning and memory [85] as well as disambiguate normal variation in structure of dendrites, branching or arbors, spines, and glia from pathological morphology, all of which can be obtained with VolRoverN's robust reconstruction tool set. Also, reduced skeleton representations derived from VolRoverN's 3D to 1D tools facilitate the study of branching patterns in neurons and glia.

Furthermore, reconstructions serve as substrate for dynamical simulation of cellular activity. For example, representations of the cell surface enable simulations 3D boundary element methods (BEM) such as combined Monte Carlo simulation of particle diffusion and kinetic state-based modeling of protein dynamics at the microsecond time scale by MCell [89, 90]. 3D finite element method (FEM) simulations [10] of electro-diffusion with multi-species continuum concentrations are enabled by decomposing a reconstruction into a collection of small volumes. Also, by approximating neuronal geometry as a collection of cylindrical compartments each aligned to segments of the geometry [97], and modeling ionic conductance in each compartment with coupled differential equations, one arrives at the traditional cable simulation of electrical signals in the brain at the millisecond scale, as supported by simulation software such as NEURON [32]. Ideally, all of these different geometric representations of brain structure could be derived from each other to facilitate multi-scale simulations via coupled MCell, FEM and NEURON models.

VolRoverN accepts geometric contours as input. Thus, success in qual-

ity reconstruction is at least partly dependent on the quality of the contours produced using other software tools. Additionally, if surface reconstruction reveals errors in the contour tracings then the user must revert back to the original software for repair. This is largely mitigated by visual proofing tools in tracing software [31, 53, 105].

In one sense, VolRoverN plays a complementary role in the set of neuronal morphology software. It provides tools to enhance geometric understanding of 2D tracings and offers an alternative method of skeletonizing neurites. In another sense, VolRoverN fills a critical gap, in that it produces meshes that are manifold, geometrically accurate, water-tight, and free of intersections. Before now, producing such reconstructions required a large amount of manual work, but VolRoverN's powerful tools greatly reduce the amount of time and domain knowledge required to prepare reconstructions for geometric analysis.

Chapter 7

Conclusion

We have proposed algorithms to solve four specific geometric problems relating to modeling and simulation of neurons at the nanoscale. We have additionally described VolRoverN, software that implements many of these algorithms. Many exciting ideas are raised by this thesis that we are interested in pursuing. We break our future work into two sections – short term and long term goals.

7.1 Short term goals

Analysis of aligned cells The first task to accomplish is that of describing further exactly what properties aligned cells (Chapter 5) need to have in order to be a viable alternative to traditional finite elements. Specific questions include what edge length and face area ratios of cells are acceptable, as well as how much support a boundary needs to have, that is, how close to a cell edge an object boundary can be while remaining stable. This latter question was investigated by Höllig et al [73] and Höllig [72] in the context of WEB-splines, but needs to be addressed in terms of immersed methods using convex polyhedra.

Efficient implementation of CVT-driven segmentation Our surface segmentation algorithm for improved remeshing has three specific areas in which it can be improved: feature preservation, improved stitching, and enhanced performance. The latter is perhaps the task we will approach first by discretizing triangles into grids and using fast bit compare operations rather than our present exact boolean set operations that are rather expensive. We expect these tasks to be straightforward, at which point we will convert our conference paper into a journal publication.

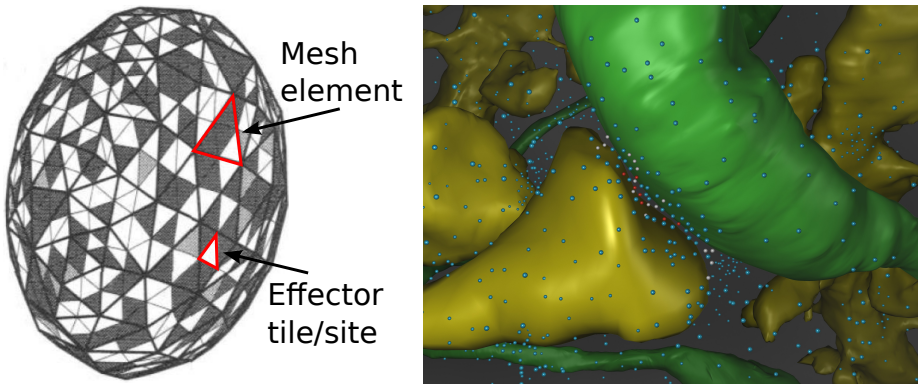
Superregions Judging from the current state of the art in surface mesh segmentation [119], there is a place for our over-segmented superregions concept that should naturally complement existing techniques. Our mesh segmentation is currently customized to our cable model generation work, but is suitable for bootstrapping a segmentation for later over-segmentation into superregions. In the meantime, we have submitted our work on 3D mesh to 1D cable model reduction as part of a larger work on VolRoverN [51]. VolRoverN fills a gap in the neuroscience community and we will continue to publicize it as an important tool for modeling of neuronal ultrastructure.

Error bounded surface meshes In Chapter 6 we performed a simple experiment to quantify error in our surface meshes. However, the ground truth surface that we used was, itself, an estimate. We are interested in investigating what possibilities there are for rigorously quantifying error in surface

reconstruction from contours. This would provide a framework for meaningful evaluation of different reconstruction methods as well as offer useful error measures of specific reconstructions to be used in downstream analysis.

MCell MCell is a Monte Carlo simulator for individual molecules in a neuron environment, specifically near a synaptic cleft [127]. It performs simulations over faceted ultrastructure domains. Each reacting and diffusing molecule is modeled as its own entity and Monte Carlo methods govern its motion (approximating brownian motion). Cell boundaries are modeled as polygonal (typically triangulated) meshes. Triangles are referred to as mesh elements. Neuronal cell boundaries are neither homogeneous nor everywhere permeable and the various receptor and transporter proteins and enzymes are modeled using barycentric subdivisions of mesh elements called effector tiles (ET). A barycentric subdivision in the case of a triangle breaks triangle edges in half and connects the new join points, forming 4 new triangles. Effector tiles can be simple non-reactive reflectors or they can be effector sites (ES), which are assigned reaction mechanisms depending on the type of protein or enzyme it corresponds to.

We are interested in augmenting VolRoverN's capabilities to more fully support MCell. This means that VolRoverN's modeling capabilities need to be extended to include user interface components for entering ET/ES information and then exporting the model and experimental parameters (i.e. time step, length of simulation) to MCell MDL file format.



(a) (b)

Figure 7.1: (a) MCell mesh elements and effector tiles [127]. (b) Type of results output from an MCell simulation.

The first step in setup of an MCell simulation is to create the surface mesh, which VolRoverN already provides. An important property of these meshes is watertightness – any holes in the mesh will allow molecules to pass through cell boundaries, bypassing microphysiological transport mechanisms.

Once the mesh is in place, size of ETs must be determined. MCell suggests that size be governed by an effector grid density (tiles/μm^2). After subdivision, properties on individual ESs are specified. Entering parameters for each individual ES, or even marking individual tiles to belong to a specific class would be time-consuming. So we plan to allow users to change a surface region and enter specific ES densities, and randomly assign ETs to specific ESs until the desired density is reached. Note that multiple site types can coexist in a single region, provided the densities don't exceed the surface area of the region. There are three specific types of sites: reflective, one-way transporter,

and two-way transporter. All three types require further reaction mechanism setup.

Integrating this setup could make VolRoverN a powerful complementary software to MCell, joining, for the first time, surface model preparation and experiment setup into a single software package.

7.2 Long term goal: multiscale modeling and simulation

We have focused on the ultrastructure of neurons. Large-scale ion reaction-diffusion simulations are still prohibitively expensive and will likely remain so for some time, so we are interested in pursuing multiscale simulation[1]. As an example, we would like to be able to run a local ion diffusion simulation simultaneously with a global cable model and field effects simulation, thereby influencing the local simulation with global effects. This requires either coupling of the governing equations or effects coupling. Figure 7.2 shows a continuum of scales in neuronal modeling and simulation.

Cable simulation An oft-used electrophysiological simulation methodology solves the cable equation to find potentials in space and time. The cable equation [80] is given as

$$\frac{1}{r_i}\frac{\partial^2 V}{\partial x^2} = c_m \frac{\partial V}{\partial t} + \frac{V}{r_m} \tag{7.1}$$

[1] see Horstemeyer [78] for an overview of multiscale modeling and simulation in the context of solid materials

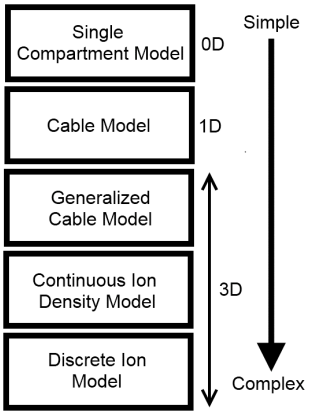


Figure 7.2: Scales for neuronal simulation.

where $r_{i,m}$ are the internal and membrane resistances of a compartment, or portion of a neuron, V is the potential on the cell membrane boundary, x is the position on the compartment and c_m is the capacitance of the membrane. The internal resistance and membrane current (V/r_m) are dependent on surface area, cross-sectional area and volume of the compartment. The compartments are combined using Kirchhoff's first law (conservation of current in a circuit) giving the final governing equations.

The cable equation is dependent on geometry, specifically for the parameters r_i, r_m, and c_m. The geometry is most often conceptualized as a skeleton of cylinders, where each segment's cylinder approximates the length and cross-sectional area of an object region. As the cylinder lengths approach zero the simulation becomes more accurate in solving equation (7.1), but a fundamental shortcoming is that homogeneity of ion concentrations in a given compartment is assumed.

Continuous ion simulation It has been shown [113] that ionic concentrations can rapidly change in thin sections of neurons, such as spine necks, so incorporation of ion concentration is important for nano-scale studies. Some recent work [104] has focused on finite-volume studies using the Nernst-Planck equation, which models ion concentrations with dependency on potential. The Poisson equation, which is coupled to the Nernst-Planck equation, models potential with dependency on ion concentrations. Specifically, the equations are [104]

$$\frac{\partial c_k}{\partial t} = \nabla \cdot [-\mathbf{J}_k] \tag{7.2}$$

$$\nabla \cdot [\epsilon(\mathbf{r}) \nabla V(\mathbf{r}, t)] = \rho(\mathbf{r}) \tag{7.3}$$

where c_k is the ion concentration of species k, $\mathbf{J}_k = D_k(\nabla c_k + \left(\frac{c_k}{\alpha_k}\right) \nabla V(\mathbf{r}, t))$ is the ion flux, $\epsilon(\mathbf{r})$ is the dielectric constant, $\{D_k, \alpha_k, z_k, F\}$ are constants, and the charge density is $\rho(\mathbf{r}) = \sum_k c_k z_k F$.

The finite-volume formulation of equations (7.2) and (7.3) is

$$\iiint_{\Omega} \frac{\partial}{\partial t} c_k d\Omega + \oiint_{S} (\mathbf{J}_k \cdot \hat{n}) dS = 0 \tag{7.4}$$

$$\oiint_{S} \hat{n} \cdot \epsilon(\mathbf{r}) \nabla V dS = - \iiint_{\Omega} \rho(\mathbf{r}) d\Omega \tag{7.5}$$

where $\hat{n}$ is the normal to the surface. In this form, the equations can be solved over a volumetric mesh, and this has been done for small volumes.

The primary issue with solving equations (7.4) and (7.5) over a volumetric mesh using FEM is the anisotropy of the data. The space between neurons, or extra-cellular space is extremely small due to the tight packing

of neurons. Most often, finite element methods use discretizations of Ω that are constrained meshes, or meshes where mesh element boundaries interpolate ∂D, which simplifies satisfying boundary conditions (the Neumann boundary condition in our problem is $\hat{n} \cdot \nabla c_k = 0$ to ensure that ions don't diffuse across boundaries). Constrained triangulations of packed neurons can yield prohibitive numbers of elements and alternative volume meshing methods, such as aligned cells for use with immersed boundary methods, are discussed in Chapter 5. However, even if aligned cells prove useful, the number of elements is still too high for a large-scale simulation.

Discrete ion simulation Discrete ion simulation models the motion of each individual ion using Monte Carlo methods to approximate brownian motion as well as binding probabilities. Rather than tetrahedralizing the entire volume, the model uses only surface meshes to represent neuronal boundaries. These meshes are broken up into multiple sub-meshes enabling receptor density simulation using counts of sub-elements assigned with given receptor properties. Geometric requirements on the mesh are not as strict as in the finite volume formulations, but they must be watertight and manifold. Monte Carlo methods are highly accurate, but they are also very computationally expensive.

Multi-scale simulation We have discussed three levels of simulation, the 1D cable model, 3D continuous ion density model, and 3D discrete ion model. We propose to advance the state of the art in two specific ways, by 1) ex-

tending the cable model to incorporate additional geometry and 2) coupling simulations at different scales.

The cable model supports tree topologies of 2D segments (2D because they each carry a length and a diameter). We propose to first augment the cable model by incorporating geometric information of organelles, such as mitochondria, which can affect electrical properties of neurons. The simplest approach is to compartmentalize into cylinders based not only on geometry of the neuron boundary, but also on presence and location of organelles. For example, a logical junction point between cylinder compartments might be at the endpoint of a mitochondria object. We also propose, which is somewhat more complex that consideration of organelles, to enhance the cable model by incorporating a third dimension of compartment geometry, that of shelling compartments radially. With accurate surface meshes obtained as described in this dissertation, we can determine interfaces between components not only axially but also radially. The challenge with this improvement is that while compartments are potentially still logical cylinders, the connectivity between components is much more dense and no longer a tree. Cycles are not supported by the NEURON simulation package, so potentially fundamental changes to the software that solves the cable equation are required.

Our second proposed long-term improvement to neuronal simulation is multi-scale simulation, which requires two types of coupling. The first is coupling of geometry. The cable model requires cylinders, while finite volume/immersed boundary methods require 3D tessellations and discrete ion

simulation uses surface meshes. We expect geometry coupling to be straightforward since in this dissertation we have presented algorithms to generate models at all ultrastructure scales from a single geometric representation: the surface mesh. Since volume mesh and compartment models are derivatives of the surface mesh, they are all in the same frame and easily registered with each other. The second coupling is that of the governing equations or, alternatively, effects coupling through dynamic boundary conditions. If we can successfully couple effects and geometry then multiscale simulation, which is not a trivial short-term work, could become a viable tool to bridge the gap between our understanding of neuronal behavior at the microscale and at the nanoscale.

Appendices

Appendix A

Terms and definitions

This chapter defines terms and sets notation that are used throughout the thesis. Individual chapters define additional terms as necessary for explanation of specific algorithms.

A.1 Symbols

1. $\mathscr{B}(c, r)$ - the r-ball, defined as a circle (2D) or sphere (3D) centered at point c with radius r
2. C^i - a surface S is C^i smooth at a point if the ith derivative is continuous at that point. The entire surface S is C^i smooth if every point $p \in S$ is C^i smooth.
3. $\text{dist}(P, Q)$ - distance between two objects P and Q. This thesis treats only euclidean distance unless otherwise specified. In euclidean space,

$$\text{dist}(P, Q) = \inf_{p \in P, q \in Q} |q - p|_2$$

4. $E(G)$
   (a) [graph] the set of all edges in graph G
   (b) [surface mesh] the set of all edges in mesh G (usually $E(M)$)

5. $F(M)$ - the set of all facets in mesh M
6. $\mathscr{R}(v, i)$ - given a graph G the i-ring is given as,

$$\mathscr{R}(v,i) = \begin{cases} \emptyset & i = -1 \\ \{v\} & i = 0 \\ \mathscr{N}(\mathscr{R}(v,i-1)) \setminus (\mathscr{R}(v,i-1) \cup \mathscr{R}(v,i-2)) & i > 0 \end{cases}$$

7. $\kappa(p)$ - we use Cauchy's definition of curvature which is, given two infinitely close points $p \in S, q \in S$, the curvature center c is the intersection of the normals at p and q, and the curvature is $1/|p-c|_2$
8. $\kappa_{\{1,2\}}(p)$ - (*see principal curvatures*)
9. *lfs*(p) - given a surface S, *lfs*$(p \in S) = \text{dist}(p, \mathscr{M}(S))$ where $\mathscr{M}(S)$ is the medial axis of S
10. $\mathscr{N}(v)$ - (neighbor) given a graph G, $\mathscr{N}(v) = \{n \in V(G) | (v,n) \in E(G)\}$
11. $V(G)$
    (a) [graph] the set of all vertices in graph G
    (b) [surface mesh] the set of all vertices in mesh G (usually $V(M)$)

A.2 Terms

1. adaptive distance field (ADF) - a distance field such that sample density is variable over the space
2. adaptive distance transform (ADT) - a distance transform that constructs an adaptive distance field
3. anisotropic [spatial]
    (a) [imaging] spacing between image slices is at a different scale than

image pixel resolution

   (b) [geometry] spacing between objects is at a different scale than object size

4. axon - a part of the neuron that carries ion current emanating away from the soma and synapses with dendrites
5. boundary
   (a) [continuous surface] a point p on a surface S lies on a boundary if $\mathscr{B}(p, \epsilon) \cap S$ lies entirely on one side of some plane passing through p
   (b) [surface mesh] an edge is a boundary edge if it is incident to exactly one facet
6. centroidal Voronoi tessellation - a Voronoi tessellation such that each site is the centroid of its respective Voronoi cell
7. complex - for the purposes of this thesis, a complex is equivalent to a tessellation
8. contour - *see trace*
9. conforming Delaunay triangulation - a constrained Delaunay triangulation that meets the requirements for a Delaunay triangulation
10. constrained Delaunay triangulation - a constrained triangulation such that the circumscribing circle of any facet contains no point visible from the facet
11. constrained Delaunay triangulation - a constrained triangulation such

that the circumscribing circle of any facet contains no point visible from the facet

12. constrained triangulation - a triangulation of a set of points such that a given set of edges connecting a subset of the points are facets in the triangulation
13. curation - removal of trace or surface mesh intersections
14. curvature (*see also Gaussian curvature, mean curvature*)
    (a) [continuous surface] *see* $\kappa(p)$
    (b) [surface mesh] an approximation of the curvature of the surface being modeled using one of various approximation methods
15. dendrite - a part of the neuron that carries ion current received from axons to the soma
16. Delaunay triangulation - a triangulation of a set of points such that the circumscribing circle of any facet of the triangulation contains no point in its interior
17. distance field (DF) - a sampled distance function
18. distance function - (*see also signed distance function*) a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ that gives the shortest distance from a point to a set of objects
19. distance transform (DT) - a method of computing a distance field that assumes a boundary condition close to the surface boundary
20. ϵ-ball - a circle (2D) or sphere (3D) of arbitrarily small radius
21. edge-neighbor [surface mesh] - given a surface mesh M and a facet $f \in$

$F(M)$, a facet $n \in F(M)$ is an edge-neighbor of f if $f \cap n \in E(M)$

22. extra-cellular space (ECS) - the space between neuron cells
23. feature [surface] - an area of a surface, usually with low curvatures or discontinities in the first derivative, that should be preserved in remeshing
24. Gaussian curvature - the product of principal curvatures $\kappa_1\kappa_2$
25. generalized Voronoi cell (GVC) - a single connected region bounded by a generalized Voronoi diagram, i.e. given a site x_i, $GVC_i = \{p \in X | \text{dist}(p, x_i) < \text{dist}(p, x_j), j \neq i\}$
26. generalized Voronoi diagram (GVD) - given a set of objects, the locus of points with at least two closest points on different objects
27. homeomorphic - a geometric object A is homeomorphic to a geometric object B if there exists a homeomorphism between the two
28. homeomorphism - a continuous function mapping a geometric object A to a geometric object B that preserves local topology at every time step (i.e. bending and stretching are allowed but tearing and joining are disallowed)
29. image plane - in the coordinate system of the electron microscope, the plane in which the physical slice lies
30. incident
    (a) [geometry] - an object a is incident to an object b if $a \cap b \neq \emptyset$
    (b) [graph] - given a graph G, a vertex $v \in V(G)$ is incident to a vertex $n \in V(G)$ if $(v, n) \in E(G)$

31. intersection (surface) - there are two types of surface intersections treated in this thesis
    (a) [self] a point p on a surface S is a self-intersection if there exist two points $q, r \in \mathscr{B}(p, \epsilon)$ such that the length of the shortest path between q and r restricted to S is greater than 2ϵ
    (b) [object-object or inter-object] given two objects A and B, a point p is an object-object intersection if $p \in A \cap B$
32. i-ring [graph] - *see* $\mathscr{R}(v, i)$
33. isotropic - not anisotropic
34. laplacian smoothing - an iterative surface mesh smoothing process that solves the Laplace equation ($\nabla^2 f = 0$) where the laplacian of each vertex is defined locally to its 1-ring
35. local feature size - *see lfs*
36. manifold - a geometric object that locally resembles euclidean space at every point. For example, a line L is a 1-manifold if, for every point $p \in L$, $\mathscr{B}(p, \epsilon) \cap L$ is a line segment between two points. A surface S is a 2-manifold if, for every point $p \in S$, $\mathscr{B}(p, \epsilon) \cap S$ is a disk.
37. mean curvature - sum of principal curvatures $(\kappa_1 + \kappa_2)/2$
38. medial axis - the locus of points with at least two closest points to a geometric object
39. neighbor
    (a) [graph] given a graph G, a set of edges $E(G)$ and vertices $V(G)$,

and a vertex $v \in V(G)$, a vertex $n \in V(G)$ is said to be a neighbor of v if edge $(v, n) \in E(G)$.

(b) [surface mesh] *see edge-neighbor, vertex-neighbor*

40. neuron - a cell used for communication using electrical signals. This thesis treats exclusively brain neurons, primarily from the hippocampal region of the mammalian brain.
41. neurotransmitter - a chemical that diffuses across the synaptic cleft from an axon to a dendrite signaling the arrival of an electrical signal
42. nonmanifold

    (a) [continuous] - a point p on a surface S is nonmanifold if $\mathscr{B}(p, \epsilon) \cap S$ is not a topological disk

    (b) nonmanifold [surface mesh]

        i. a vertex is nonmanifold if the incident triangles are not orderable

        ii. an edge is nonmanifold if there are more than 2 incident triangles

43. octree - a 3D spatial decomposition where a cube space is recursively subdivided into 8 cube octants
44. ordinary Voronoi diagram (VD) - *see Voronoi diagram*
45. orientable [surface mesh] - a surface mesh is orientable if it can be oriented
46. oriented [surface mesh] - two adjacent triangles are consistently oriented

if their shared edge has opposite orientations on each triangle, and a surface mesh is oriented if all pairs of adjacent triangles are consistently oriented

47. principal curvatures ($\kappa_{\{1,2\}}$) - given a point p on a surface S and the normal $\mathbf{n}$ at that point, the principal curvatures are the minimum and the maximum of the curvatures of the curves defined by $P \cap S$ where P is a plane passing through p with normal $\mathbf{n}'$ where $\mathbf{n}^T\mathbf{n}' = 0$
48. quadtree - a 3D spatial decomposition where a square space is recursively subdivided into 4 square quadrants
49. quality - this thesis treats only measures of isotropic quality:
    (a) a triangle is quality if it is close to equilateral. This thesis uses four measures of quality:
        i. ratio $r_c/2r_i$ where r_c is the circumradius and r_i is the inradius of a triangle
        ii. $Q_t = \frac{6}{\sqrt{3}}\frac{r_i}{h_t}$ where h_t is the longest edge length of the triangle
        iii. min triangle angle
        iv. max triangle angle
    (b) mesh quality is usually measured by the min and max of a triangle quality measure across all triangles
50. remeshing - the process of transforming a surface mesh into another surface mesh for the purposes of simplification or quality improvement
51. ring - *see i-ring*

52. section

    (a) the physical slice of brain tissue placed under an electron microscope

    (b) the EM image of a physical slice of brain tissue

53. signed distance function (SDF) - a distance function that reports distances as signed values, with sign depending on whether the point is inside or outside of the object

54. slice - *see section*

55. soma - the neuron cell body where the nucleus resides and from which dendrites and axons extend

56. surface mesh - an approximation of a continuous surface using incident polygons, usually triangles or quadrilaterals

57. synapse - an area of close approach between an axon and dendrite at which an electrical signal is propagated from an axon to a dendrite

58. synaptic cleft - the space between the axon and dendrite at a synapse

59. tessellation - the separating of a space S into individual geometric objects $\{C_i\}$ such that $\bigcup_i C_i = S$ and $\bigcap_i C_i = \emptyset$

60. tile - a geometric object in a tessellation. This thesis usually refers to tiles as triangles used in fitting a surface to planar contours.

61. trace - a polygonal outline of the cross-section of a neuron on an EM image

62. vertex-neighbor [surface mesh] - given a surface mesh M and a facet $f \in F(M)$, a facet $n \in F(M)$ is a vertex-neighbor of f if $f \cap n \in V(M)$

63. visible [geometry] - given two objects P and Q and a set of obstacles D, P is visible from Q (and vice-versa) if

$$\left(\bigcup_{p \in P, q \in Q} \overline{pq} \right) \cap D = \emptyset$$

where $\overline{pq}$ is the line segment between points p and q

64. Voronoi cell (VC) - a single connected region bounded by a Voronoi diagram, i.e. given a site x_i, $GVC_i = \{p \in X | \text{dist}(p, x_i) < \text{dist}(p, x_j), j \neq i\}$

65. Voronoi diagram (VD) - given a set of sites, or seed points, the locus of points with at least two closest sites

66. Voronoi tessellation - a tessellation such that, given a set of sites, each cell is a Voronoi cell

67. watertight [surface mesh] - a surface mesh with no boundary edges

Appendix B

Proofs of ForestTiler theorems

Our proofs rely on an additional theorem from [11], which we restate here in a slightly weaker formulation, but sufficient for our purposes:

Theorem 8 ([11], Theorem 2)**.** *For every point p^g on the surface ∂C^g of the green component,* $1 \leq |\mathscr{C}^g(p^g)| \leq 2$.

Lemma 1. *Suppose p^g is a point on the surface ∂C^g of component C^g. Then* $1 \leq |\mathscr{C}(p^g)| \leq 2$.

Proof. Since $|\mathscr{C}(p^g)| \supset |\mathscr{C}^g(p^g)|$, then $1 \leq |\mathscr{C}^g(p^g)|$ by theorem 8. We prove that $|\mathscr{C}^g(p^g)| \leq 2$ by contradiction. Suppose $|\mathscr{C}^g(p^g)| > 2$. Then by the pigeonhole principle, at least 2 contours must exist on some adjacent slice. These contours share the projection p_s^g onto the slice of point p_s^g. This violates criterion 4. □

This lemma states that any point on a surface must have at least one containing contour (of any color) but no more than two. In Figure 2.3, p_1 has no containing contour and so cannot exist on the surface of the reconstructed component.

Lemma 2. *If $|\mathscr{C}(p^g)| = 2$ then the two contours in $\mathscr{C}(p^g)$ lie on separate slices.*

Proof. If the 2 containing contours exist on the same slice, then they share the projection p^g_s onto the slice of point p^g_s. This violates criterion 4. □

Lemma 3. *No point $p^g \in S^g$ can be a conflict point.*

Proof. This is a proof by contradiction. Consider a point $p^g \in S^g$. Suppose that p^g is a conflict point. Then there exists some point p^y such that $p^y \in \mathscr{B}(p^g)$ (for the moment, set aside the fact that $\mathscr{Z}(p^g)$ is undefined). By lemma 1 we know that the projection of point p^g can have no more than 2 containing contours. By virtue of being in S^g, p^g indeed has 2 containing contours, both of which belong to the green component by definition. Since $p^y \in \mathscr{B}(p^g)$, the projections of p^y onto each adjacent slice are within δ of the projections of p^g. Then the projection of p^y onto $\mathscr{C}(p^y)$ (by theorem 8 there must be at least one) is within δ of a green contour $\mathscr{C}(p^g)$ which contradicts criterion 4. Thus no point p^y exists and p^g is not a conflict point. □

Lemma 4. *Let $p^g \in \partial C^g$ be a conflict point. Then there is no other point $q^g \in \partial C^g$ such that $p^{g\prime} = q^{g\prime}$.*

Proof. By lemma 3, $p^g \in U^g$ and thus cannot be on a vertical tile edge. Then, by criterion 2, a vertical line passing through p^g passes through no other point. □

Proof of theorem 1. By definition of a conflict point, there exists some p^y such that $p^{g\prime} = p^{y\prime}$. By lemma 4, p^g and p^y are the only points lying on the vertical line passing through them. In addition, these two points are unique, as the addition of a third point on the vertical line would violate theorem 8 and lemma 1. □

Proof of theorem 2. We first prove that if a conflict point $p^g \in \partial C^g$ exists, then the two components are, at some point, within δ of each other. Let $p^g \in \partial C^g$ be a conflict point. Let Γ be the vertical path from p^g to the slice at $\mathscr{Z}(p^g)$. By definition of a conflict point, Γ will pass within δ of some point $p^y \in \partial C^y$ before reaching the slice. By criterion 2, Γ intersects ∂C^g exactly once, at p^g, and since every point inside the planar contour $\mathscr{C}(p^g)$ is inside the green component, every point on Γ between p^g and the projection of p^g onto $\mathscr{Z}(p^g)$ is inside the green component. Therefore, some point $q^g \in (\Gamma \cup C^g)$ is within δ of ∂C^y.

If two components are within δ of each other, then by definition there exists a conflict point. □

Proof of corollary 1. If p^g is the intersection point between two components it is by definition a conflict point. □

Proof of theorem 3. Let $p^g \in U^g$ and let $\bar{p^g}$ be p^g shifted by ϵ in the direction of $\mathscr{Z}(p^g)$. $\mathscr{B}(\bar{p^g}) \subset \mathscr{B}(p^g)$ and therefore $\{p^y | p^y \in \mathscr{B}(\bar{p^g})\} \subset \{q^y | q^y \in \mathscr{B}(p^g)\}$. Therefore the number of conflict points will only decrease. □

Proof of theorem 4. We prove this by contradiction. Suppose there exists a single pair of conflict points $p^g \in \partial C^g$ and $p^y \in \partial C^y$ on the interiors of triangles $t^g \in \partial C^g$ and $t^y \in \partial C^y$, respectively. Further, suppose that every point $p \in (\partial C^g \cup \partial C^y)$ is not a conflict point. Then $|p^g - p^y| < \delta$ while the minimum separation distance between ∂C^g and ∂C^y is at least δ. This violates planarity of the triangles, and therefore there must be some conflict point $p \in (\partial C^g \cup \partial C^y)$. □

Proof of theorem 5. By definition, the projection $p^{g\prime}$ of p^g onto the slice at $z = \mathscr{Z}(p^g)$ lies inside or on the boundary of a green contour. It can lie on a boundary only if $p^g \in S^g$, so by lemma 3 we know that $p^{g\prime}$ lies on the interior of a green contour. Since the contours are separated by δ, and q^y lies on a yellow contour, then $|p^{g\prime} - q^y| > \delta$. Thus there is some point $\bar{p^g}$ on the vertical line between p^g and $p^{g\prime}$ such that $\frac{|\bar{\mathbf{A}} \times \bar{\mathbf{B}}|}{|\bar{\mathbf{B}}|} = \delta$ proving the first statement. The second follows with a similar argument. □

Appendix C

Proof of distance function error bound

C.1 Subdivision limit

Let d be the minimum distance between two objects.

Fact 1. *Subdividing an octree down to cells of edge length $d/\sqrt{3}$ ensures that all objects will be resolved.*

Proof. Consider figure C.1a. A cube with edge length $\alpha_0 \leq \frac{d}{\sqrt{3}}$ that intersects a surface point p cannot intersect any point further than d distance from p. Thus any single cube cell can intersect only one surface. □

Fact 2. *Subdividing an octree down to cells of edge length $d/2\sqrt{3}$ ensures that at least one empty buffer cell will exist between all object pairs.*

Proof. Consider figure C.1b. Let M be a line segment connecting p with its closest point q on another surface. A cube with edge length $\alpha_0 \leq \frac{d}{2\sqrt{3}}$ that intersects the midpoint of M cannot intersect p or q. A subdivision is a complex, so every point must be covered by a cell, and thus some empty cell exists that intersects the midpoint of M. □

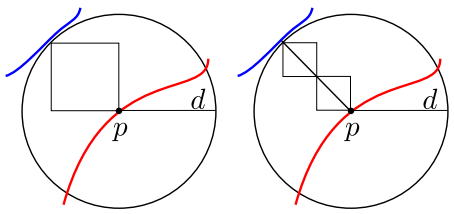


Figure C.1: Figures used in section C.1.

C.2 Distance error bound

This section derives a loose bound for the distance function over an octree. The bound is $\frac{\epsilon(v)}{\delta(v)} \leq \frac{1}{2}$*, where* $e(v)$ *is the error at a vertex* v *and* $\delta(v)$ *is the distance from* v *to the nearest point on* S*.*

See figure C.2. $\alpha(v)$ *is the shortest distance between* v *and any of* v*'s neighbors in the cardinal directions. Let* $p(v) = \arg\min_{p \in S} dist(p, v)$ *be the point on* S *that is closest to* v *and* $\delta(v) = dist(p(v), v)$*. With* $V(v)$ *as the set of all vertices visible to* v*, let* $\overline{p}(v) = \arg\min_{a \in V(v)} dist(\overline{p}(a), v)$ *be the octree approximation of* $p(v)$ *and* $\overline{\delta}(v) = dist(\overline{p}(v), v)$ *the octree approximation* $\delta(v)$*. Because* $\overline{p}(v)$ *is a point on the surface then we know that* $\overline{\delta}(v) \geq \delta(v)$*. Define* $\epsilon(v) = \overline{\delta}(v) - \delta(v)$*.* $\mathscr{B}(a, r)$ *is a ball centered at* a *with radius* r*.* $\hat{\delta}(b)$ *is the radius of the ball centered at* b *that is guaranteed not to contain* $\overline{\delta}(b)$ *by virtue of* $\mathscr{B}(v, \overline{\delta}(v))$*. And finally,* $k = \frac{\alpha(v)}{\alpha(a)}$*.*

Our proof builds geometric constructs (specifically, unions of circles) of regions Ω *for which* $\Omega \cap S = \emptyset$*. We find, for a given* Ω*, the closest point* $q(v) \in \Omega^C$ *to* v*, that is,* $q(v) = \arg\min_{p \in \Omega^C} dist(p, v)$*. The proof is not complete;*

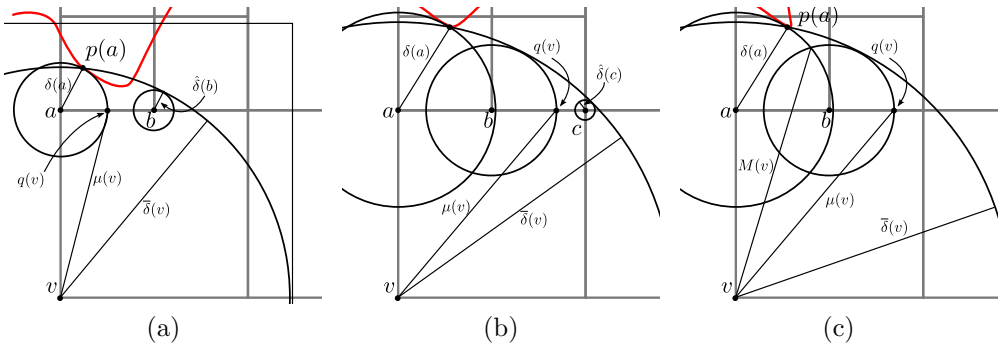


Figure C.2: Bounds proof. (a) Lemma 1 case. (b) Lemma 2 case. (c) Lemma 3.

specifically, we have one conjecture yet to prove (conjecture 1).

Approach *: We bound the error by showing that, in every case,* $\mu(v) = dist(q(v), v) \leq (1/2)\alpha(v)$*. Some proofs of lemmas are omitted for brevity.*

The first case is as shown in figure C.2a, and whether the case holds is governed by the following lemma:

Lemma 5 (Meeting circles lemma a)**.** *For* $\bar{p}(v) = p(a)$*,*

$$\delta(a) \leq \frac{1}{2}(1 - k + \sqrt{k^2 + 1})\alpha(a) \Leftrightarrow \mathscr{B}(a, \delta(a)) \cap \mathscr{B}(b, \hat{\delta}(b)) = \emptyset$$

Proof.

$$\begin{aligned}
\delta(a) &= \alpha(a) - \hat{\delta}(b) \\
&= \alpha(a) - (\bar{\delta}(v) - \sqrt{\alpha^2(v) + \alpha^2(a)}) \\
&\leq \alpha(a) - (\delta(a) + \alpha(v) - \sqrt{\alpha^2(v) + \alpha^2(a)}) \quad \text{(equal for } \bar{p}(v) \text{ on vertical)} \\
&= \alpha(a) - \alpha(v) + \sqrt{\alpha^2(v) + \alpha^2(a)} - \delta(a) \\
&= \frac{1}{2}(\alpha(a) - \alpha(v) + \sqrt{\alpha^2(v) + \alpha^2(a)}) \\
&= \frac{1}{2}(1 - k + \sqrt{k^2 + 1})\alpha(a) \qquad \text{(C.1)}
\end{aligned}$$

□

Corollary 2. *For* $\bar{p}(v) = p(a)$,

$$\delta(a) \geq \frac{\sqrt{2}}{2}\alpha(a) \Rightarrow \mathscr{B}(a, \delta(a) \cap \mathscr{B}(b, \hat{\delta}(b)) \neq \emptyset$$

The second case is as shown in figure C.2b, and whether the case holds is governed by the following lemma:

Lemma 6 (Meeting circles lemma b)**.** *For* $\bar{p}(v) = p(a)$ *and* $[b, c]$ *are* $[\alpha(v)/2, \alpha(v)]$, *respectively, from* a *in the positive* x *direction,*

$$\delta(a) \leq \frac{\sqrt{5} + 2\sqrt{2} - 3}{4}\alpha(v) \Leftrightarrow \mathscr{B}(b, \hat{\delta}(b) \cap \mathscr{B}(c, \hat{\delta}(c)) = \emptyset$$

Proof.

$$\hat{\delta}(b) + \hat{\delta}(c) = \frac{1}{2}\alpha(v)$$

$$\hat{\delta}(b) = \delta(v) - \sqrt{\alpha^2(v) + \frac{1}{4}\alpha^(v)} = \delta(v) - \frac{\sqrt{5}}{2}\alpha(v)$$

$$\hat{\delta}(c) = \delta(v) - \sqrt{\alpha^2(v) + \alpha^(v)} = \delta(v) - \sqrt{2}\alpha(v)$$

$$\begin{aligned}
\delta(a) &= \delta(v) - \alpha(v) \\
&= \frac{1}{2}(\hat{\delta}(b) + \hat{\delta}(c) + (\frac{\sqrt{5}}{2} + \sqrt{2} - 1)\alpha(v) \\
&= \frac{1}{2}(\frac{1}{2} + \frac{\sqrt{5}}{2} + \sqrt{2} - 1)\alpha(v) \\
&= \frac{\sqrt{5} + 2\sqrt{2} - 3}{4}\alpha(v)
\end{aligned}$$

□

Lemma 7 (Error lemma base case)**.** *For $\bar{p}(v) = p(a)$ and $\delta(a) \leq \frac{\sqrt{2}}{2}\alpha(a)$,*

$$\epsilon(v) = \epsilon_0(v, k) \leq \frac{1}{2}(2k + \sqrt{2} - \sqrt{4k^2 + 2})\alpha_0 \tag{C.2}$$

Proof. It follows from the algorithm that if $\delta(a) \leq \frac{\sqrt{2}}{2}\alpha(a)$ then $\alpha(a) = \alpha_0$.

$$\begin{aligned}
\epsilon(v) &= \alpha(v) + \bar{\delta}(a) - \sqrt{\alpha^2(v) + \delta^2(a)} \\
&= \alpha(v) + \delta(a) - \sqrt{\alpha^2(v) + \delta^2(a)} \qquad (\epsilon(a) = 0) \\
&\leq \alpha(v) + \frac{\sqrt{2}}{2}\alpha_0 - \sqrt{\alpha^2(v) + \frac{1}{2}\alpha_0^2} \\
&= \frac{1}{2}(2k + \sqrt{2} - \sqrt{4k^2 + 2})\alpha_0
\end{aligned}$$

□

Note that

$$\epsilon_0(v, k \leq 2) < 0.18 < e_0(v, k > 2) \tag{C.3}$$

Conjecture 1 (Intersecting circles lemma)**.** *See figure C.2c.*

$$\mu(v) < M(v)$$

Theorem 9. $\frac{\epsilon(v)}{\delta(v)} \leq \frac{1}{2}$.

Proof. We first show that $\epsilon(v) \leq C\alpha(v)$ by induction.

Case 1 (base) By definition, if v is non-empty, then $\epsilon(v) = 0$.

Case 2 By lemma 7 and equation (C.3), if v is empty and $\alpha(a) < \frac{1}{2}\alpha(v)$, then

$$\epsilon(v) = \epsilon_0(v, k) < \frac{1}{2}\alpha(v)$$

Induction $k > 2$. By conjecture 1, the error is maximized at $\delta(a) = \frac{\sqrt{5}+2\sqrt{2}-3}{4}\alpha(v)$. Then

$$\begin{aligned}
\epsilon(v) &\leq \alpha(v) + \delta(a) + \epsilon(a) - \sqrt{\alpha^2(v) + (\alpha(v) - \hat{\delta}(c))^2} \\
&\leq \alpha(v) + \frac{\sqrt{5}+2\sqrt{2}-3}{4}\alpha(v) - \sqrt{\alpha^2(v) + \left(\alpha(v) - \frac{1+\sqrt{5}-2\sqrt{2}}{4}\alpha(v)\right)^2} + \epsilon(a) \\
&= \left(1 + \frac{\sqrt{5}+2\sqrt{2}-3}{4} - \sqrt{2 - \frac{1+\sqrt{5}-2\sqrt{2}}{2} + \frac{(1+\sqrt{5}-2\sqrt{2})^2}{16}}\right)\alpha(v) + \epsilon(a) \\
&\leq 0.18\alpha(v) + \epsilon(a) \\
&\leq 0.18\alpha(v) + 0.37\alpha(a) \\
&\leq 0.18\alpha(v) + \frac{0.37}{2}\alpha(v) \\
&\leq 0.37\alpha(v)
\end{aligned}$$

If $\delta(v) < \alpha(v)$ then the error is zero and the theorem is proved by the base case. Otherwise,

$$\epsilon(v) \leq C\alpha(v) \leq C\delta(v)$$

. □

C.3 Resolve ambiguities algorithm

Algorithm 4: RESOLVE_AMBIGUITIES

```
Input: Octree
C := the set of all cells in Octree
while C is not empty do
    c := any cell in C
    remove c from C
    count := number of connected components in c
    num_labels := number of unique labels in c
    if count > 3 then
        D := subdivide c
        V_D := all vertices in D
        V := all vertices in c
        foreach vertex v in V_D \ V do
            point[v] := null
            dist[v] := ∞
            add v to V
        end
        Expand wavefront V
        add D to C
    end
end
```

Bibliography

[1] *P. Alliez, D. Cohen-Steiner, M. Yvinec, and M. Desbrun. Variational tetrahedral meshing.* ACM Transactions on Graphics (TOG), *24(3):617–625, 2005.*

[2] *P. Alliez, EC de Verdire, O. Devillers, and M. Isenburg. Isotropic surface remeshing. In* Shape Modeling International, 2003*, pages 49–58. IEEE, 2003.*

[3] *P. Alliez, M. Meyer, and M. Desbrun. Interactive geometry remeshing.* ACM Transactions on Graphics*, 21(3):347–354, 2002.*

[4] *P. Alliez, G. Ucelli, C. Gotsman, and M. Attene. Recent advances in remeshing of surfaces.* Shape analysis and structuring*, pages 53–82, 2008.*

[5] *N. Amenta and M. Bern. Surface reconstruction by voronoi filtering.* Discrete & Computational Geometry*, 22(4):481–504, 1999.*

[6] *Per Andersen, Richard Morris, David Amaral, Tim Bliss, and John O'Keefe.* The hippocampus book*. Oxford University Press, USA, 2006.*

[7] *Roberto Araya, Kenneth B Eisenthal, and Rafael Yuste. Dendritic spines linearize the summation of excitatory potentials.* Proceedings of the National Academy of Sciences*, 103(49):18799–18804, 2006.*

[8] *Roberto Araya, Jiang Jiang, Kenneth B Eisenthal, and Rafael Yuste. The spine neck filters membrane potentials.* Proceedings of the National Academy of Sciences*, 103(47):17961–17966, 2006.*

[9] *O. K-C. Au, C-L. Tai, H-K. Chu, D. Cohen-Or, and T-Y. Lee. Skeleton extraction by mesh contraction.* ACM Transactions on Graphics*, 27:44:1–10, 2008.*

[10] *C. Bajaj, R. Bettadapura, N. Lei, A. Mollere, C. Peng, and A. Rand. Constructing A-spline weight functions for stable WEB-spline finite element methods. In* Proceedings of the 14th ACM Symposium on Solid and Physical Modeling*, pages 153–158. ACM, 2010.*

[11] *C.L. Bajaj, E.J. Coyle, and K. Lin. Arbitrary topology shape reconstruction from planar cross sections.* Graphical Models and Image Processing*, 58(6):524–543, 1996.*

[12] *C.L. Bajaj, E.J. Coyle, and K. Lin. Tetrahedral meshes from planar cross sections. In* Computer Methods in Applied Mechanics and Engineering*, pages 31–52, 1999.*

[13] *C.L. Bajaj and A. Gillette. Quality meshing of a forest of branching structures. In* Proceedings of the 17th International Meshing Roundtable*,*

pages 433–449. Springer-Verlag, October 2008.

[14] C.L. Bajaj, V. Pascucci, and D.R. Schikore. Fast isocontouring for improved interactivity. In Proceedings of the 1996 symposium on Volume visualization*, pages 39–ff. IEEE Press, 1996.*

[15] C.L. Bajaj, V. Pascucci, and D.R. Schikore. The contour spectrum. In Proceedings of the 8th conference on Visualization'97*, pages 167–ff. IEEE Computer Society Press, 1997.*

[16] C.L. Bajaj, G. Xu, R.J. Holt, and A.N. Netravali. Hierarchical multiresolution reconstruction of shell surfaces. Computer Aided Geometric Design*, 19(2):89 – 112, 2002.*

[17] G. Barequet, M.T. Goodrich, A. Levi-steiner, and D. Steiner. Contour interpolation by straight skeletons, graphical models. In Graphical Models, v.66 n.4*, pages 245–260, 2004.*

[18] G. Barequet and M. Sharir. Piecewise-linear interpolation between polygonal slices. In Computer Vision and Image Understanding*, pages 93–102, 1994.*

[19] G. Barequet and A. Vaxman. Nonlinear interpolation between slices. In Proceedings of the 2007 ACM Symposium on Solid and Physical Modeling*, pages 97–107, New York, NY, USA, 2007. ACM.*

[20] G. Barequet and A. Vaxman. Reconstruction of multi-label domains from partial planar cross-sections. In SGP '09: Proceedings of the

Symposium on Geometry Processing, *pages 1327–1337, Aire-la-Ville, Switzerland, Switzerland, 2009. Eurographics Association.*

[21] Thiago Bastos and Waldemar Celes. Gpu-accelerated adaptively sampled distance fields. In Shape Modeling and Applications, 2008. SMI 2008. IEEE International Conference on, *pages 171–178. IEEE, 2008.*

[22] Amit Bermano, Amir Vaxman, and Craig Gotsman. Online reconstruction of 3d objects from arbitrary cross-sections. ACM Transactions on Graphics (TOG), *30(5):113, 2011.*

[23] Brenda L Bloodgood and Bernardo L Sabatini. Neuronal activity regulates diffusion across the neck of dendritic spines. Science Signaling, *310(5749):866, 2005.*

[24] J.D. Boissonnat and B. Geiger. Three-dimensional reconstruction of complex shapes based on the Delaunay triangulation. In Proceedings of SPIE, *volume 964, 1992.*

[25] J.D. Boissonnat and P. Memari. Shape reconstruction from unorganized cross-sections. In Proceedings of the fifth Eurographics symposium on Geometry processing, *pages 89–98. Eurographics Association, 2007.*

[26] Jean-Daniel Boissonnat, Camille Wormser, and Mariette Yvinec. Curved voronoi diagrams. In Effective Computational Geometry for Curves and Surfaces, *pages 67–116. Springer, 2006.*

[27] *David E Breen, Sean Mauch, and Ross T Whitaker. 3d scan conversion of csg models into distance volumes. In* Volume Visualization, 1998. IEEE Symposium on*, pages 7–14. IEEE, 1998.*

[28] *K.L. Briggman and W. Denk. Towards neural circuit reconstruction with volume electron microscopy techniques.* Current Opinion in Neurobiology*, 16(5):562–570, 2006.*

[29] *John Canny and Bruce Donald. Simplified voronoi diagrams.* Discrete & Computational Geometry*, 3(1):219–236, 1988.*

[30] *Thanh-Tung Cao, Ke Tang, Anis Mohamed, and Tiow-Seng Tan. Parallel banding algorithm to compute exact distance transform with the gpu. In* Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, pages 83–90. ACM, 2010.*

[31] *Albert Cardona, Stephan Saalfeld, Johannes Schindelin, Ignacio Arganda-Carreras, Stephan Preibisch, Mark Longair, Pavel Tomancak, Volker Hartenstein, and Rodney J Douglas. TrakEM2 software for neural circuit reconstruction.* PLoS One*, 7(6):e38011, 2012.*

[32] *N.T. Carnevale and M.L. Hines.* The NEURON Book*. Cambridge, UK: Cambridge University, 2006.*

[33] *Hamish Carr, Jack Snoeyink, and Ulrike Axen. Computing contour trees in all dimensions. In* Proceedings of the eleventh annual ACM-

SIAM symposium on Discrete algorithms, *pages 918–926. Society for Industrial and Applied Mathematics, 2000.*

[34] F. Cazals and M. Pouget. Estimating differential quantities using polynomial fitting of osculating jets. Computer Aided Geometric Design, *22(2):121–146, 2005.*

[35] S.W. Cheng, T.K. Dey, and E.A. Ramos. Delaunay refinement for piecewise smooth complexes. Discrete & Computational Geometry, *43(1):121–166, 2010.*

[36] D. Cohen-Steiner, P. Alliez, and M. Desbrun. Variational shape approximation. ACM Transactions on Graphics, *23(3):905–914, 2004.*

[37] D. Cohen-Steiner and J.M. Morvan. Restricted delaunay triangulations and normal cycle. In Proceedings of the nineteenth annual symposium on Computational geometry, *pages 312–321. ACM, 2003.*

[38] CVC. Lbie: Level set boundary interior and exterior mesher. `http://cvcweb.ices.utexas.edu/cvc/projects/project.php?proID=10`.

[39] CVC. Volume Rover. `http://cvcweb.ices.utexas.edu/cvcwp/?page_id=100`.

[40] Mark De Berg, Otfried Cheong, and Marc Van Kreveld. Computational geometry: algorithms and applications. *Springer, 2008.*

[41] *Silvia De Rubeis, Esperanza Fernández, Andrea Buzzi, Daniele Di Marino, and Claudia Bagni. Molecular and cellular aspects of mental retardation in the fragile x syndrome: from gene mutation/s to spine dysmorphogenesis. In* Synaptic Plasticity*, pages 517–551. Springer, 2012.*

[42] *Gouri Dhatt, Gilbert Touzot, et al.* Finite Element Method*. Wiley-ISTE, 2012.*

[43] *David A Drachman. Do we have brain to spare?* Neurology*, 64(12):2004–2005, 2005.*

[44] *Q. Du, V. Faber, and M. Gunzburger. Centroidal voronoi tessellations: Applications and algorithms.* SIAM review*, pages 637–676, 1999.*

[45] *Q. Du, M.D. Gunzburger, and L. Ju. Constrained centroidal voronoi tessellations for surfaces.* SIAM Journal on Scientific Computing*, 24(5):1488–1506, 2003.*

[46] *H. Edelsbrunner and N.R. Shah. Triangulating topological spaces.* International Journal of Computational Geometry and Applications*, 7(4):365–378, 1997.*

[47] *J. Edwards. Livemesh: An interactive 3d image segmentation tool. Master's thesis, Brigham Young University, Provo, Utah, May 2004.*

[48] *J. Edwards and C Bajaj. Topologically correct reconstruction of tortuous contour forests.* Computer-Aided Design*, 43(10):1296 – 1306, 2011.*

[49] *J. Edwards, W. Wang, and C. Bajaj. Surface segmentation for improved isotropic remeshing. In* Proceedings of the 21st International Meshing Roundtable*, pages 403–418. Springer-Verlag, October 2012.*

[50] *John Edwards, Eric Daniel, and Chandrajit Bajaj. An adaptive distance transform for fast voronoi diagram computation.* Manuscript*, 1, 2013.*

[51] *John Edwards, Eric Daniel, Justin Kinney, Tom Bartol, Terrence Sejnowski, Kristen Harris, Daniel Johnston, and Chandrajit Bajaj. VolRoverN: Automated reconstruction of cellular morphology for multiscale dynamical simulation of neural activity.* Manuscript submitted for publication*, 1, 2013.*

[52] *Michal Etzion and Ari Rappoport. Computing voronoi skeletons of a 3-d polyhedron by space subdivision.* Computational Geometry*, 21(3):87–120, 2002.*

[53] *J.C. Fiala. Reconstruct: a free editor for serial section microscopy.* Journal of Microscopy*, 218(1):52–61, 2005.*

[54] *J.C. Fiala, J. Spacek, and K.M. Harris. Dendritic spine pathology: cause or consequence of neurological disorders?* Brain Research Reviews*, 39(1):29–54, 2002.*

[55] *Michael S Floater. Mean value coordinates.* Computer aided geometric design*, 20(1):19–27, 2003.*

[56] *P.J. Frey and H. Borouchaki. Surface mesh evaluation. In* Proceedings of the 6th International Meshing Roundtable*, pages 403–418. Citeseer, 1997.*

[57] *Sarah F Frisken, Ronald N Perry, Alyn P Rockwood, and Thouis R Jones. Adaptively sampled distance fields: a general representation of shape for computer graphics. In* Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 249–254. ACM Press/Addison-Wesley Publishing Co., 2000.*

[58] *H. Fuchs, Z.M. Kedem, and S.P. Uselton. Optimal surface reconstruction from planar contours.* Communications of the ACM*, 20(10):693–702, 1977.*

[59] *S. Fuhrmann, J. Ackermann, T. Kalbe, and M. Goesele. Direct resampling for isotropic surface remeshing. In* Vision, Modeling, and Visualization*, pages 9–16, 2010.*

[60] *M. Garland. Qslim.* `http://mgarland.org/software/qslim.html`.

[61] *M. Garland and P.S. Heckbert. Surface simplification using quadric error metrics. In* Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, pages 209–216. ACM Press/Addison-Wesley Publishing Co., 1997.*

[62] Åsa Grunditz, Niklaus Holbro, Lei Tian, Yi Zuo, and Thomas G Oertner. Spine neck plasticity controls postsynaptic calcium signals through electrical compartmentalization. The Journal of Neuroscience*, 28(50):13457–13466, 2008.*

[63] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[64] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. In Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47 – 57. ACM, 1984.*

[65] P. Hagmann. From diffusion MRI to brain connectomics*. PhD thesis, Institut de traitement des signaux PROGRAMME DOCTORAL EN INFORMATIQUE ET COMMUNICATIONS POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES PAR Docteur en médecine, Université de Lausanne, 2005.*

[66] Maryam Halavi, Kelly A Hamilton, Ruchi Parekh, and Giorgio A Ascoli. Digital reconstructions of neuronal morphology: three decades of research trends. Frontiers in Neuroscience*, 6, 2012.*

[67] K.M. Harris and J.K. Stevens. Dendritic spines of CA 1 pyramidal cells in the rat hippocampus: serial electron microscopy with reference to their biophysical characteristics. The Journal of Neuroscience*, 9(8):2982, 1989.*

[68] *Iain Hepburn, Weiliang Chen, Stefan Wils, and Erik De Schutter. STEPS: efficient simulation of stochastic reaction-diffusion models in realistic morphologies.* BMC Syst Biol, *6(35):1752–0509, 2012.*

[69] *Kenneth E Hoff III, John Keyser, Ming Lin, Dinesh Manocha, and Tim Culver. Fast computation of generalized voronoi diagrams using graphics hardware. In* Proceedings of the 26th annual conference on Computer graphics and interactive techniques, *pages 277–286. ACM Press/Addison-Wesley Publishing Co., 1999.*

[70] *Dax A Hoffman, Jeffrey C Magee, Costa M Colbert, and Daniel Johnston.* K^+ *channel regulation of signal propagation in dendrites of hippocampal pyramidal neurons.* Nature, *387(6636):869–875, 1997.*

[71] *C.M. Hoffmann.* Geometric and Solid Modeling: An Introduction. *Morgan Kaufmann Pub, 1989.*

[72] *K. Höllig.* Finite element methods with B-splines, *volume 26. Society for Industrial Mathematics, 2003.*

[73] *K. Höllig, U. Reif, and J. Wipper. Weighted extended b-spline approximation of dirichlet problems.* SIAM Journal on Numerical Analysis, *pages 442–462, 2002.*

[74] *William R Holmes. Is the function of dendritic spines to concentrate calcium?* Brain research, *519(1):338–342, 1990.*

[75] H. Hoppe. Progressive meshes. In Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 99–108. ACM, 1996.*

[76] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 19–26. ACM, 1993.*

[77] K. Hormann and M.S. Floater. Mean value coordinates for arbitrary planar polygons. ACM Transactions on Graphics*, 25(4):1424–1441, 2006.*

[78] Mark F. Horstemeyer. Multiscale modeling: a review. In Practical Aspects of Computational Chemistry: Methods, Concepts and Applications*, pages 87–136. Springer, 2010.*

[79] W.K. Jeong, J. Beyer, M. Hadwiger, R. Blue, C. Law, A. Vázquez-Reina, R.C. Reid, J. Lichtman, and H. Pfister. Ssecrett and neurotrace: interactive visualization and analysis tools for large-scale neuroscience data sets. IEEE Computer Graphics and Applications*, 30(3):58, 2010.*

[80] D. Johnston, S.M.S. Wu, and R. Gray. Foundations of cellular neurophysiology*. MIT press Cambridge, MA:, 1995.*

[81] Mark W Jones, J Andreas Baerentzen, and Milos Sramek. 3d distance

fields: A survey of techniques and applications. Visualization and Computer Graphics, IEEE Transactions on, *12(4):581–599, 2006.*

[82] P. Joshi, M. Meyer, T. DeRose, B. Green, and T. Sanocki. Harmonic coordinates for character articulation. ACM Transactions on Graphics, *26, July 2007.*

[83] Elizabeth Jurrus, Shigeki Watanabe, Richard J Giuly, Antonio RC Paiva, Mark H Ellisman, Erik M Jorgensen, and Tolga Tasdizen. Semi-automated neuron boundary detection and nonbranching process segmentation in electron microscopy images. Neuroinformatics, *pages 1–25, 2012.*

[84] Menelaos I Karavelas. A robust and efficient implementation for the segment voronoi diagram. In International symposium on Voronoi diagrams in science and engineering, *pages 51–62. Citeseer, 2004.*

[85] Haruo Kasai, Masahiro Fukuda, Satoshi Watanabe, Akiko Hayashi-Takagi, Jun Noguchi, et al. Structural dynamics of dendritic spines in memory and cognition. Trends in neurosciences, *33(3):121, 2010.*

[86] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. International Journal of Computer Vision, *1(4):321–331, 1988.*

[87] D.X. Keller, T.M. Bartol, J.P. Kinney, M.B. Kennedy, C.L. Bajaj, K.M. Harris, and T.J. Sejnowski. Synaptic Calcium Transients in Reconstructed Dendritic Spines on Hippocampal CA1 Pyramidal Neurons

are Regulated by Calcium Pumps. Manuscript submitted for publication*, 2011.*

[88] R.A. Kerr, T.M. Bartol, B. Kaminsky, M. Dittrich, J.C.J. Chang, S.B. Baden, T.J. Sejnowski, and J.R. Stiles. Fast Monte Carlo simulation methods for biological reaction-diffusion systems in solution and on surfaces. SIAM Journal on Scientific Computing*, 30(6):3126, 2008.*

[89] J. Kinney. Investigation of neurotransmitter diffusion in three-dimensional reconstructions of hippocampal neuropil. *PhD thesis, University of California, San Diego, 2009.*

[90] Justin P Kinney, Josef Spacek, Thomas M Bartol, Chandrajit L Bajaj, Kristen M Harris, and Terrence J Sejnowski. Extracellular sheets and tunnels modulate glutamate diffusion in hippocampal neuropil. Journal of Comparative Neurology*, 521(2):448–464, 2013.*

[91] Christof Koch and Anthony Zador. The function of dendritic spines: devices subserving biochemical rather than electrical compartmentalization. J. Neurosci*, 13(2):413–422, 1993.*

[92] David Lavender, Adrian Bowyer, James Davenport, Andrew Wallis, and John Woodwark. Voronoi diagrams of set-theoretic solid models. Computer Graphics and Applications, IEEE*, 12(5):69–77, 1992.*

[93] Der-Tsai Lee. Medial axis transformation of a planar shape. Pattern

Analysis and Machine Intelligence, IEEE Transactions on, *pages 363–369, 1982.*

[94] Sylvain Lefebvre and Hugues Hoppe. Compressed random-access trees for spatially coherent data. In Proceedings of the 18th Eurographics conference on Rendering Techniques, *pages 339–349. Eurographics Association, 2007.*

[95] Xuetao Li, Tong Wing Woon, Tiow Seng Tan, and Zhiyong Huang. Decomposing polygon meshes for interactive applications. In Proceedings of the 2001 symposium on Interactive 3D graphics, *pages 35–42. ACM, 2001.*

[96] Jyh-Ming Lien, John Keyser, and Nancy M Amato. Simultaneous shape decomposition and skeletonization. In Proceedings of the 2006 ACM symposium on Solid and physical modeling, *pages 219–228. ACM, 2006.*

[97] K.A. Lindsay, J.R. Rosenberg, and G. Tucker. From maxwell's equations to the cable equation and beyond. Progress in Biophysics and Molecular Biology, *85:71–116, 2004.*

[98] Y. Lipman, J. Kopf, D. Cohen-Or, and D. Levin. GPU-assisted positive mean value coordinates for mesh deformations. In Proceedings of the fifth Eurographics symposium on Geometry processing, *pages 117–123. Eurographics Association, 2007.*

[99] *D.C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization.* Mathematical programming, *45(1):503–528, 1989.*

[100] *L. Liu, C. Bajaj, L.O. Deasy, D.A. Low, and T. Ju. Surface reconstruction from non-parallel curve networks.* Computer Graphics Forum, *27(2):155–163, 2008.*

[101] *Meng Liu, James Duggan, Thomas E Salt, and M Francesca Cordeiro. Dendritic changes in visual pathways in glaucoma and other neurodegenerative conditions.* Experimental eye research, *92(4):244–250, 2011.*

[102] *Y. Liu, W. Wang, B. Lévy, F. Sun, D.M. Yan, L. Lu, and C. Yang. On centroidal voronoi tessellationenergy smoothness and fast computation.* ACM Transactions on Graphics (ToG), *28(4):101, 2009.*

[103] *S. Lloyd. Least squares quantization in pcm.* Information Theory, IEEE Transactions on, *28(2):129–137, 1982.*

[104] *C.L. Lopreore, T.M. Bartol, J.S. Coggan, D.X. Keller, G.E. Sosinsky, M.H. Ellisman, and T.J. Sejnowski. Computational modeling of three-dimensional electrodiffusion in biological systems: Application to the node of ranvier.* Biophysical Journal, *95(6):2624–2635, 2008.*

[105] *J. Lu, J.C. Fiala, and J.W. Lichtman. Semi-automated reconstruction of neural processes from large numbers of fluorescence images.* PLoS One, *4(5):5655, 2009.*

[106] *Y. Mishchenko, T. Hu, J. Spacek, J. Mendenhall, K.M. Harris, and D.B. Chklovskii. Ultrastructural analysis of hippocampal neuropil from the connectomics perspective.* Neuron*, 67(6):1009–1020, 2010.*

[107] *R Narayanan, K J Dougherty, and D Johnston. Calcium store depletion induces persistent perisomatic increases in the functional density of* h *channels in hippocampal pyramidal neurons.* Neuron*, 68(5):921–935, 2010.*

[108] *O. Nilsson, D. Breen, and K. Museth. Surface reconstruction via contour metamorphosis: An eulerian approach with lagrangian particle tracking. In* Proceedings of IEEE Visualization*, pages 407–414, 2005.*

[109] *J-M. Oliva, M. Perrin, and S. Coquillart. 3d reconstruction of complex polyhedral shapes from contours using a simplified generalized voronoi diagram.* Computer Graphics Forum*, 15(3):397–408, 1996.*

[110] *J. O'Rourke.* Computational Geometry in C*. Cambridge University Press, 1994.*

[111] *Valerio Pascucci and Kree Cole-McLaughlin. Efficient computation of the topology of level sets. In* Proceedings of the conference on Visualization'02*, pages 187–194. IEEE Computer Society, 2002.*

[112] *G. Peyre and L. Cohen. Surface segmentation using geodesic centroidal tesselation. In* 3D Data Processing, Visualization and Transmission,

2004. 3DPVT 2004. Proceedings. 2nd International Symposium on, *pages 995–1002. IEEE, 2004.*

[113] N. Qian and TJ Sejnowski. An electro-diffusion model for computing membrane potentials and ionic concentrations in branching dendrites, spines and axons. Biological Cybernetics, *62(1):1–15, 1989.*

[114] Omar A Ramírez and Andrés Couve. The endoplasmic reticulum and protein trafficking in dendrites and axons. Trends in cell biology, *21(4):219–227, 2011.*

[115] Xiaofeng Ren and Jitendra Malik. Learning a classification model for segmentation. In Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on, *pages 10–17. IEEE, 2003.*

[116] Guodong Rong and Tiow-Seng Tan. Variants of jump flooding algorithm for computing discrete voronoi diagrams. In Voronoi Diagrams in Science and Engineering, 2007. ISVD'07. 4th International Symposium on, *pages 176–181. IEEE, 2007.*

[117] Fidel Santamaria, Stefan Wils, Erik De Schutter, and George J Augustine. Anomalous diffusion in purkinje cell dendrites caused by spines. Neuron, *52(4):635–648, 2006.*

[118] Idan Segev, Alon Friedman, Edward L White, and Michael J Gutnick. Electrical consequences of spine dimensions in a model of a cortical spiny

stellate cell completely reconstructed from serial thin sections. Journal of computational neuroscience, *2(2):117–130, 1995.*

[119] Ariel Shamir. A survey on mesh segmentation techniques. In Computer graphics forum, *volume 27, pages 1539–1556. Wiley Online Library, 2008.*

[120] GORDON M Shepherd. The dendritic spine: a multifunctional integrative unit. Journal of neurophysiology, *75(6):2197–2210, 1996.*

[121] J.R. Shewchuk. What is a good linear finite element? interpolation, conditioning, anisotropy, and quality measures (preprint). University of California at Berkeley, *2002.*

[122] R. Sibson. A brief description of natural neighbour interpolation. Interpreting multivariate data, *pages 21–36, 1981.*

[123] Christoph Sommer, Christoph Straehle, Ullrich Koethe, and Fred A. Hamprecht. ilastik: interactive learning and segmentation toolkit. In 8th IEEE International Symposium on Biomedical Imaging (ISBI 2011), *2011.*

[124] C.K. Song, L.W. Enquist, and T.J. Bartness. New developments in tracing neural circuits with herpesviruses. Virus research, *111(2):235–249, 2005.*

[125] *Karin E Sorra and Kristen M Harris. Overview on the structure, composition, function, development, and plasticity of hippocampal dendritic spines.* Hippocampus*, 10(5):501–511, 2000.*

[126] *O. Sporns, G. Tononi, and R. Kotter. The human connectome: a structural description of the human brain.* PLoS Comput Biol*, 1(4):e42, 2005.*

[127] *Joel R Stiles, Thomas M Bartol, et al. Monte carlo methods for simulating realistic synaptic microphysiology using mcell. In* Computational neuroscience: Realistic modeling for experimentalists*, pages 87–128. CRC Press, Boca Raton, FL, 2001.*

[128] *J. Strain. Fast tree-based redistancing for level set computations.* Journal of Computational Physics*, 152(2):664–686, 1999.*

[129] *Avneesh Sud, Naga Govindaraju, Russell Gayle, and Dinesh Manocha. Interactive 3d distance field computation using linear factorization. In* Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 117–124. ACM, 2006.*

[130] *V. Surazhsky, P. Alliez, and C. Gotsman. Isotropic remeshing of surfaces: A local parameterization approach. In* In Proceedings of 12th International Meshing Roundtable*, 2003.*

[131] *Karel Svoboda, David W Tank, Winfried Denk, et al. Direct measurement of coupling between dendritic spines and shafts.* SCIENCE-NEW

YORK THEN WASHINGTON-, *pages 716–718, 1996.*

[132] Justin W Taraska and William N Zagotta. Fluorescence applications in molecular neurobiology. Neuron*, 66(2):170–189, 2010.*

[133] R.G. Thorne and C. Nicholson. In vivo diffusion analysis with quantum dots and dextrans predicts the width of brain extracellular space. Proceedings of the National Academy of Sciences of the United States of America*, 103(14):5567, 2006.*

[134] Sebastian Thrun. Learning metric-topological maps for indoor mobile robot navigation. Artificial Intelligence*, 99(1):21–71, 1998.*

[135] David Tsay and Rafael Yuste. Role of dendritic spines in action potential backpropagation: a numerical simulation study. Journal of neurophysiology*, 88(5):2834–2845, 2002.*

[136] G. Turk and J.F. O'Brien. Shape transformation using variational implicit functions. In SIGGRAPH'99*, pages 335–342, 1999.*

[137] L Vachhani, Arun D Mahindrakar, and K Sridharan. Mobile robot navigation through a hardware-efficient implementation for control-law-based construction of generalized voronoi diagram. Mechatronics, IEEE/ASME Transactions on*, 16(6):1083–1095, 2011.*

[138] Myrrhe van Spronsen and Casper C Hoogenraad. Synapse pathology in psychiatric and neurologic disease. Current neurology and neuroscience reports*, 10(3):207–214, 2010.*

[139] D. Wang, O. Hassan, K. Morgan, and N. Weatherill. Efficient surface reconstruction from contours based on two-dimensional delaunay triangulation. In In Proc. Int. J. Numer. Meth. Engng*, pages 734–751, 2006.*

[140] J. Wu and L. Kobbelt. Structure recovery via hybrid variational surface approximation. In Computer Graphics Forum*, volume 24, pages 277–284. Wiley Online Library, 2005.*

[141] J. Xie, T. Zhao, T. Lee, E. Myers, and H. Peng. Automatic Neuron Tracing in Volumetric Microscopy Images with Anisotropic Path Searching. Medical Image Computing and Computer-Assisted Intervention–MICCAI 2010*, 6362:472–479, 2010.*

[142] D.M. Yan, B. Lévy, Y. Liu, F. Sun, and W. Wang. Isotropic remeshing with fast and exact computation of restricted voronoi diagram. Computer graphics forum*, 28(5):1445–1454, 2009.*

[143] Kangxue Yin, Youquan Liu, and Enhua Wu. Fast computing adaptively sampled distance field on gpu. In Pacific Graphics Short Papers*, pages 25–30. The Eurographics Association, 2011.*

[144] L. Zhang, A. Gerstenberger, X. Wang, and W.K. Liu. Immersed finite element method. Computer Methods in Applied Mechanics and Engineering*, 193(21):2051–2067, 2004.*

[145] Q. Zhang, R. Bettadapura, and C. Bajaj. Macromolecular structure modeling from 3d em using volrover 2.0. Biopolymers*, 97(9):709–731, 2012.*

[146] Xiaoyu Zhang, Chandrajit L Bajaj, Bongjune Kwon, Todd J Dolinsky, Jens E Nielsen, and Nathan A Baker. Application of new multiresolution methods for the comparison of biomolecular electrostatic properties in the absence of global structural similarity. Multiscale Modeling & Simulation*, 5(4):1196–1213, 2006.*

[147] Y. Zhang, C. Bajaj, and B.S. Sohn. 3D finite element meshing from imaging data. Computer methods in applied mechanics and engineering*, 194(48-49):5083–5106, 2005.*

[148] Y. Zhang, C. Bajaj, and G. Xu. Surface smoothing and quality improvement of quadrilateral/hexahedral meshes with geometric flow. Communications in Numerical Methods in Engineering*, 25(1):1–18, 2009.*

[149] Y. Zhang, C.L. Bajaj, and G. Xu. Surface smoothing and quality improvement of quadrilateral/hexahedral meshes with geometric flow. In In Proceedings, 14th International Meshing Roundtable*, pages 449 – 468. John Wiley & Sons, 2005.*

[150] Yongjie Zhang and Chandrajit Bajaj. Adaptive and quality quadrilateral/hexahedral meshing from volumetric data. Computer methods in applied mechanics and engineering*, 195(9):942–960, 2006.*

[151] *Yongjie Zhang, Thomas JR Hughes, and Chandrajit L Bajaj. An automatic 3d mesh generation method for domains with multiple materials.* Computer Methods in Applied Mechanics and Engineering (CMAME)*, 199(5):405–415, 2010.*

Vita

John Martin Edwards was born in Logan, Utah on 3 December 1974, the son of Dr. W. Farrell Edwards and Ann P. Edwards. He received the Bachelor of Science degree in Computer Science from Utah State University in 1999 and the Master of Science degree in Computer Science from Brigham Young University in 2004. He was employed as a research and development engineer by Rigaku, Inc., ProLogic, Inc., and Autonomous Solutions, Inc. He returned to pursue Ph.D. studies in Computer Science at The University of Texas at Austin in June, 2009.

Permanent address: 2317 Speedway, 2.302
Austin, Texas 78712

This dissertation was typeset with LaTeX† by the author.

†LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.