Automatic Mesh Generation and Processing for Biomedical Geometries

A Dissertation Presented

by

Volodymyr Dyedov

to

The Graduate School

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Applied Mathematics and Statistics

Stony Brook University

August 2012

Stony Brook University

The Graduate School

Volodymyr Dyedov

We, the dissertation committee for the above candidate for the Doctor of Philosophy degree, hereby recommend acceptance of this dissertation.

This dissertation is accepted by the Graduate School.

Charles Taber
Interim Dean of the Graduate School

Abstract of the Dissertation

Automatic Mesh Generation and Processing for Biomedical Geometries

by

Volodymyr Dyedov

Doctor of Philosophy

in

Applied Mathematics and Statistics

Stony Brook University

2012

Introduction of advantages of fluid dynamics simulations into clinical practice depends on ability to quickly create patient-specific computational models from computed tomography or magnetic resonance images with minimal human interaction. Our research addresses three steps in this process. First is identifying lung airways in radiological images. Our approach is based on a combination of geometric analysis of gradient vector flow and graph-based algorithms. Second is a novel method for robust and efficient computation of medial curves for complex biomedical geometries. This algorithm is based on three key concepts: a local orthogonal decomposition of geometry into substructures, a differential concept called the interior center of curvature, and integrated stability and consistency tests. Finally, we introduce variational method for generating prismatic boundary-layer meshes. Compared to existing methods, our approach is novel in the following aspects: it relies on feature size to make resulting mesh scale invariant and prevent global self-intersections; it uses face-offsetting method to propagate surface mesh to generate the high-quality prismatic layers; finally, it allows to add prismatic boundary layer to any tetrahedral mesh while preserving structural qualities of original discretization.

Contents

Contents **iv**

List of Figures **vi**

List of Tables **xi**

1 Introduction **1**

2 Segmentation **5**
- 2.1 Introduction . . . 5
- 2.2 Gradient Vector Flow Field . . . 7
- 2.3 Flow of the Algorithm . . . 8
- 2.4 Random Walks . . . 10
  - 2.4.1 Implementation Details . . . 11
  - 2.4.2 Random Walker Weights . . . 12
- 2.5 Results . . . 14

3 Medial Curves **17**
- 3.1 Previous Work . . . 17
  - 3.1.1 Volume-Based Approaches . . . 17
  - 3.1.2 Surface-Based Approaches . . . 18
- 3.2 Requirements for the Medial Curve . . . 19
- 3.3 Geometry and Topology of Medial Curves . . . 21
  - 3.3.1 Canal Surfaces and Centers of Curvature . . . 21
- 3.4 Local Orthogonal Decomposition . . . 23
  - 3.4.1 Decomposition of Objects . . . 24
  - 3.4.2 Approximation by Orthogonal Cuts . . . 25
- 3.5 Computation of Medial Curves . . . 26
  - 3.5.1 Computing Differential Quantities . . . 26
  - 3.5.2 Local Dimensionality of Object . . . 27
  - 3.5.3 Identifying Stable Tips . . . 28

3.5.4 Fine-Tuning Cuts 37
3.5.5 Reconstructing Medial Curve 39
3.5.6 Implementation Details 40
3.5.7 Experimental Results 41
3.5.8 Comparison with Other Methods 44

4 Boundary Layer Meshes **46**
4.1 Previous work 46
4.2 Feature Size Guided Face Offsetting 47
4.2.1 Gradient Limited Face Offsetting 48
4.2.2 Advancing Layers by Face Offsetting 51
4.3 Variational Optimization of Prismatic Layers 54
4.3.1 Elemental Energy for Triangle Shapes 56
4.3.2 Elemental Energy for Edge Orthogonality 57
4.3.3 Energy Minimization 59
4.3.4 Overall Smoothing Algorithm 59
4.4 Robustness and Implementation Issues 60
4.4.1 Positivity of Jacobian 60
4.4.2 Adaptive Step Size Control 62
4.4.3 Enforcement of Boundary Constraints 62
4.4.4 Generation of Interior Tetrahedral Meshes 63
4.5 Experimental Assessment and Comparisons 65
4.5.1 Quality Measures for Prisms 65
4.5.2 Rat Lung 66
4.5.3 Human Heart 72
4.5.4 Comparison with Commercial Software 76
4.6 Variational Generation of Hybrid Mesh 78
4.6.1 Initial Generation of Tetrahedra 79
4.6.2 Initial Generation of Prisms and Shrinking Tetrahedra . . . 84
4.6.3 Expanding Prisms 86
4.7 Results 89
4.7.1 Rat Lung 89
4.8 Conclusions and Discussions 93

5 Conclusions **95**

Bibliography **97**

List of Figures

2.1 Response of Median Tube-Likeliness Filter and ridges extracted from it. 6
2.2 Flow of our method. 7
2.3 Slice of the image of Gradient Vector Flow field magnitudes. Extremums correspond to boundaries of the objects. 8
2.4 Subset of the image that can be extracted based on analysis of singular value decomposition and Hessians of local GVF and normalized GVF. 8
2.5 Panel **A**: Subset of the image with divergence of normalized GVF less then -0.001. Panel **B**: subset of the image extracted by intensity based connected threshold algorithm within subset in Panel **A**. 9
2.6 Figure on the left shows subset of the image corresponding to airways and bones colored by probability that random walker starting at the vertex will first reach seed in the airways. On the right are voxels at which the probabilities are greater then 0.5. 11
2.7 Airways and bones colored same as in the original CT image. Color histograms of bones and airways. 13
2.8 Panel **A** shows pre-segmentation of the image by connected intensity threshold algorithm within the subset of the voxels where divergence of normalized inversed GVF field is < -0.0001. Airways and bones are connected on slices 4 to 120. Panel **B** shows section yz of the sub-image and slices 4, 60 and 120. 14
2.9 Seeds for initializing Random Walker Algorithm. From top left to bottom right corner seeds are [188, 131, 18] (bones); [184,159,43] (airways); [181,158,85] (airways) and [181,148,136] (airways) and [201,280,136] (bones). 15

2.10 Solutions of the discrete Dirichlet problems associated with Random Walker algorithm. Panel **A** is the sub-image colored by the probabilities that Random Walker starting from the vertex will first reach on of the seeds inside the rib-cage. Panel **B** depicts solution to second linear system, which equals to vector of 1's minus first solution. On the right image are cross-sections of the image on Panel **A** 16
2.11 Segmentation of the airways. 16

3.1 Examples of medial curves (red) of a cylinder, a torus, and a surface generated by a sphere of radius $0.3 + 0.15\sin(3t)$ moving on a unit circle. 21
3.2 Illustration of tip, segment, and junction, along with orthogonal cuts (blue planes). Red curves show medial curves. 24
3.3 Post-processing of local orthogonal decomposition, before and after. 39
3.4 Medial curve of three biological surfaces. **A**) A mouse coronary artery from CT data. **B**) A monkey pulmonary airway from MRI data. **C**) A high-resolution rat pulmonary airway from CT data. . . . 42
3.5 Noisy dataset of mouse coronary artery and the computed medial curve. 44
3.6 Comparison of our results against two other methods for the smoothed and unsmoothed coronary geometry. Artifacts in the results are indicated by black arrows. Panels A-D contrast the medial-geodesic method with our method; Panels E-H contrast the mesh contraction method 45

4.1 Development of a scale-invariant surface mesh from MRI data: A) binary segmentation of the imaging data of a monkey lung cast; B) isosurface generation; C) detail of the isosurface; D) surface triangulation registered to the feature size; E) feature size field. . . . 50
4.2 Without gradient limiting $F[x_i]$ can discontinuously jump with a small perturbation in the surface S due to borderline clipping of local diameter ray. With gradient limiting, this jump is generally limited to $G\|x_i - x_j\|$, where x_j is a neighbor whose local diameter ray is robustly clipped. 51
4.3 Advective vs. wavefrontal propagation at expansion. 53
4.4 Illustration of offsetting each face. 53
4.5 Examples of poor-shaped prisms due to poor triangles with too large or too small angles (left two) and lack of orthogonality due to twisting or shifting (right two). 55

4.6 Naming convention of (a) vertices, edges, face normals, and (b) orthogonal directions of prism for energy computation. Underlined symbols indicate edges. 56
4.7 Schematic of mapping from master element to actual element for prisms. 60
4.8 2-D illustration of constraints. 64
4.9 Example of evolution of surface under boundary constraints. 64
4.10 Boundary layer mesh of a rat lung. Panel A shows the feature size fields on the lung surface, as well as the locations of the cutaway sections in panels B–E. Panel B shows the boundary prism and interior tetrahedra at a region where a large airway , with f(x) greater than , transitioning to a smaller airway. Panels C and D show the sections through the trachea and through a pair of airways downstream of the bifurcation, respectively. Panel E shows the resulting surface mesh in the neighborhood of several outlets, where the faces are constrained. 67
4.11 Mesh-quality statistics for the prismatic layers and interior tetrahedra of the rat lung mesh generated using our proposed method. Panel A shows the triangle angles for each surface layer, starting with the original surface mesh (60° is ideal). Panel B shows the edge distortions of each prismatic layers (0° is ideal). Panel C shows the scaled aspect ratios of prisms for each prismatic layers (1.0 is ideal, and a negative value would indicate an inverted prism). Finally, panel D shows the aspect ratios for tetrahedral elements (1 is ideal) 68
4.12 Transient CFD results for the pulmonary airways of 9-wk-old, male rat. Flow data (inset) were experimentally determined from a mechanical ventilator and prescribed to the trachea. Downstream outlets were uniformly prescribed zero pressure boundary conditions. Columns $\mathbf{A}\&\mathbf{B}$ correspond to time points $\mathbf{A}\&\mathbf{B}$ on the inset velocity curve, roughly corresponding in turn to peak inhalation and peak exhalation. Velocity vectors and velocity contours through each domain (prismatic or tetrahedral) are shown in respectively in rows $\mathbf{1}-\mathbf{3}$. 70
4.13 Comparative statistics for prismatic layers using different alternatives. First panel shows the options used for each series and also their corresponding marched distances as percentage of GLFS. Last three panels show quality metrics of selected methods after propagating 15% of GLFS. 72

4.14 Multi-material boundary layer mesh of a human heart, consisting of heart tissue, blood boundary layer and blood domains. Panels A and B show the feature size field on the outer surface of the heart tissue domain and the blood domain, respectively. Panel B shows the feature size field on the outer surface of the blood domain. Panel C shows a cut through the heart muscle tessellated with layered tetrahedra [44], where the orientation of the cut plane is indicated in Panel A. Panel D shows the detail of the layered tetrahedral mesh of the heart tissue. Panels E and F are zoomed-in views on the regions of the cardiac valves, showing the prismatic boundary layer alone. Panels G, H and I show the prismatic boundary layer at about 25% of GLFS of the blood domain, sandwiched between the layered tetrahedra of the tissue and the Delaunay tetrahedra of the blood. Heart model and surface mesh are courtesy of the NYU Medical Center and Zhang et al. 73
4.15 Mesh quality statistics for the prismatic layers and interior tetrahedra of the human heart mesh generated using our proposed method. Panel A shows the angles for each surface layer ($60°$ is ideal). Panel B shows the edge distortions of each prismatic layer ($0°$ is ideal). Panel C shows the the scaled aspect ratio for prisms of each prismatic layer (1 is ideal, and a negative value would indicate an inverted prism). Finally, panel D shows the aspect ratio for tetrahedral elements (1 is ideal) 75
4.16 Comparison of prismatic boundary layer generated using a leading commercial CFD mesh generator and our method. 78
4.17 Comparison of mesh quality of prismatic boundary layer generated using commercial software and our method. Solid curves indicate ridge curves in the surface mesh with dihedral angles close to $90°$. . 79
4.18 Comparison of mesh quality statistics of prismatic boundary layer generated using commercial software and our method. 80
4.19 Naming convention for tetrahedron. Underlined symbols indicate edges. 84
4.20 Illustration of construction of reference tetrahedra for vertices 1 and 4 of a prism. (a) the actual (in dark edges) and ideal prism. (b) Reference tetrahedron (in dark edges) at vertex 1. (c) Reference tetrahedra at vertex 4. 87
4.21 Quality of tetrahedral mesh before and after compressing by 40% of GLFS. Number of tetrahedra : 526393. 90

4.22 Quality (percentage points) of tetrahedral core compressed by 40% of GLFS, dark columns correspond to compressed core, light to tetrahedra generated by tetgen based on inner surface of boundary layer. Total number of tetrahedra in compressed core was 526393, in constrained tetrahidralization by TetGen - 623473. 91
4.23 Comparison between quality of compressed layered and constrained Delaunay tetrahedral cores. Dark columns correspond to compressed mesh, light to constrained Delaunay. Values in tables are percentage points. 92
4.24 Comparison between quality of prismatic boundary layers. 93

List of Tables

3.1 Results for three test geometries in Figure 3.4. 42
3.2 Results for different meshes for the mouse coronary. 43
3.3 Comparison of our method with alternative surface-based methods for the mouse coronary artery. For Au et al., numbers of tips and junctions were counted visually, as their code did not allow saving the result. All times are in seconds. 45

4.1 Marched percentage of the GLFS for heart geometry using different methods. “Prop.” and "Interp." refer to propagated and interpolated layers, respectively. 76

Chapter 1

Introduction

Computational continuum physics has become a platform for discovery in many scientific and engineering disciplines, including biomedical research. For example, the understanding of blood shear stress profiles requires numerical simulations, which can shed light on the pathophysiology of thrombus formation or rupture in cerebral aneurysms [35, 64]. Also, evaluating risk of cerebral aneurysm rupture can be modeled as a fluid-structure interaction problem. If risk is deemed high enough to warrant a surgery, the operation has to be planned with high precision because the effects of an increased wall stress caused by the intervention will only be evident over time. Similarly, fluid and particle dynamics in the lung can shed light on airway smooth muscle spasm in asthma or abnormal clearance in cystic fibrosis [12, 41]. Many of these biomedical problems involve complex geometries, which are increasingly derived from imaging modalities such as magnetic resonance or computed tomography. Image processing and isosurface extraction have been the subject of much research (e.g. [49, 57, 58, 73]). While these techniques have become relatively mature in recent years to extract surface geometries from medical images, the whole process of extracting topologically correct volumetric image still requires a lot of manual interaction and is time-consuming. Geometries extracted from medical images are noisy and can be substantially more complex than the man-made CAD models in classical engineering problems. Therefore, significant challenges remain in the pre-processing and generation of computational meshes from these complex biological geometries.

We have been working on framework that will provide a pipeline from medical images to simulation of physics of fluids. Patient-specific simulations require ability to quickly build computational models of the organs from radiological data with minimum of human interaction. In this work we addressed three steps in the process. First, airways or blood vessels have to be identified in the computed tomography or magnetic resonance image. Images we have been working with are CT scans of rodents whose airways were injected with silicone. Density of the silicone filling is close to that of surrounding rib cage and spine which makes it difficult to differentiate between them based on intensity, moreover, they appear connected in the image. Local structure of airways and bones is also similar as they both can be viewed as collection of tubes. Finally, the cast material often forms bubbles which appear as darker points in the domain and areas of brightness where contrast agent dissolved unevenly forming pockets. Our approach is based on a combination of geometric analysis of gradient vector flow and graph-based algorithms. We divide a problem of identifying airways into sub-tasks of extracting connected airways and surrounding bone structure and then separating them from each other.

Result of the image segmentation is a volume mesh. Surface mesh extracted directly from this representation has to be pre-processed to accommodate boundary conditions. Imposing proper outlet boundary conditions for accurate simulations requires the identifications of outlets. Introduction of orthogonal boundary patches - outlets, into treed structures, such as arterial or airway trees, is a challenging part of transforming images into computational mesh. Traditionally, this process is accomplished interactively using some surface mesh editor. But with recent advances in image acquisition and processing, it is now common to resolve tens of thousands of dense branches in these geometries, like in lung airways. So manual manipulation becomes impossible. We introduced an efficient and robust approach to identifying and truncating the outlets for complex biomedical geometries. This technique led to developing the method of medial axis computation for certain types of geometries. The computation of medial curves poses significant challenges, both in terms of theoretical analysis and practical efficiency and reliability. In biomedical engineering medial curves are important for multi-scale modeling [24], morphometry [17, 80], physiology [53], image analysis [50], and computer assisted surgery [78]. For example, the medial curves can form the basis of a morphometric analysis of tree-structured airways in the lung [17] or coronary vessels in the heart [25], by providing statistics for segment length, diameter, and bifurcation angle. Since the medial curves embed the topology of the tree and distill its geometry, these data can then be analyzed for fractal relationships that reveal genetic decision making in morphogeneis, pathological remodeling or anatomic variation. Furthermore, in multiscale modeling, these data can form the basis of physiological multiscale boundary conditions in the form of lower-dimensional differential equations that represent the biophysics of subsets of the vessel or airway tree at a scale that is not practical to represent as a 3D continuum [24]. In endoscopy or virtual surgery, the medial curves can help to pilot diagnostic devices to a specific location in the airway or vessel tree [78].

In second part of the writeup, we propose a definition and analysis of medial curves and also describe an efficient and robust method called *local orthogonal cutting* (*LOC*) for computing medial curves. Our approach is based on three key concepts: a local orthogonal decomposition of objects into substructures, a differential geometry concept called the *interior center of curvature* (ICC), and integrated stability and consistency tests. These concepts lend themselves to robust numerical techniques and result in an algorithm that is efficient and noise resistant. Next part will talk about hybrid mesh generation based on the input surface preprocessed by LOC.

Third part is devoted to generation of hybrid prismatic-tetrahedral volume mesh. The grid is the link between geometry and the resolution of the flow physics. Its density and quality is of significant importance for the solution accuracy. For most applications two most important types of interpolation errors are the difference between the interpolated function and the true function, and the difference between the gradient of the interpolated function and the gradient of the true function. Errors in the gradient can be extremely important for blood flow simulation. Pathology in wall shear stress can have dramatic consequences for patients so gradient error is the primary arbiter of element quality. In finite element methods they contribute to discretization errors. If the true function is smooth, the error in the interpolated function can be reduced simply by refining the mesh. However, the error in the gradient is strongly affected by the shape of the elements as well as their size. Finite element method rely on the availability of meshes whose elements have the right shapes and sizes. Element shape has a strong influence on matrix conditioning. Condition number of the stiffness matrix associated with the method should be kept as small as possible. Poorly conditioned matrices affect linear equation solvers by slowing them down or introducing large roundoff errors into their results.

Viscous flows like air and blood flow tend to have strong gradients at the boundary due to wall shear stress, an effect that has been shown to be extremely important in biomechanics. We generate volume mesh based on a given surface. In boundary layer regions of the flow field derivatives normal to the wall can be much greater than they are parallel to the flow. We fill this region with several layers of prisms to resolve these features of the flow by concentrating computational grid at the wall while keeping the overall computational cost of the problem tractable. The use of tetrahedra to fill the rest of the domain allows single-block generation for extremely complex geometries since the tetrahedron is the simplex element in 3-D. This approach has the advantages of creating elements that are mostly orthogonal to the wall in regions of high solution gradients while essentially decoupling the grid density in the normal and tangential directions. In addition, prismatic layers have been shown to be effective in reducing errors in turbulence, particle deposition, and mass-transfer problems. In biological fluid-structure interaction problems with a Lagrangian interface, prismatic boundary layers are also desirable near the interfaces, such as closing cardiac valves. In these cases, the direction of impaction is locally normal to the boundary, so it is aligned with axis of the prisms. Thus, prisms as opposed to tetrahedra are less prone to entanglement.

Generating such a mesh based on surface of biomedical object faces some significant challenges. First of all, they can have large variations in curvature. Unlike man-made engineering models, biomedical geometries have no well-defined sharp features but combination of high curvature and coarse resolution can introduce near-singularities. In addition, biological geometries often span multiple spatial scales. The lumen of a coronary artery, for example, may be several orders of magnitude smaller than the span across a ventricle, and thus there is a need to equilibrate error over a range of meaningful scales. These challenges are not limited to the heart. The airways of the lung for example range from a several centimeters to hundreds of microns.

We propose a new method for generating prismatic boundary-layer meshes that is suitable for complex biological geometries. Compared to existing methods, our method is novel in the following aspects:

1) We use the feature size defined to control the height of the prisms to make the resulting mesh scale invariant and prevent global self-intersections.

2) We use the face-offsetting method to propagate the surface mesh to generate the prismatic layers, which provides reliable directions of vertex motions and prevents local self-intersections.

3) We develop a variational mesh optimization for prismatic layers, which improves not only the shapes of the triangles but also the orthogonality of the prisms.

4) We propose an approach to introduce high quality boundary layer to any tetrahedral mesh while preserving structural qualities of the original discretization.

Chapter 2

Segmentation

2.1 Introduction

The heart mesh used in our simulations is an idealized model. On the other hand, surfaces of airways in lungs and coronary vessels were extracted from radiological images, e.g. computer tomography. In this section we will address the problem of identifying airways in the chest image. This is an example of image segmentation problem, which had been an active area of research in computer science and electrical engineering. Segmentation of airways for generating meshes used in this work had been done through manually intensive and time consuming process. This section describes our work on automating the task of identifying airways in the CT chest images.

Problem of segmentation of airways in medical images attracted a lot of attention. Evaluation study was held in 2009 to compare performances of different airways extraction algorithms [47]. We used the study as a starting point in our work. One of the best results was shown by algorithm by team from Institute for Computer Graphics and Vision at Graz University of Technology lead by Christian Bauer. In a nutshell, his method consists of identifying tube-like objects in the image, constructing their centerlines, constructing a tree-like structure based on subset of centerlines satisfying certain heuristics, which is thought of as centerline of the airways, and finally, employing graph cut algorithm initialized with the centerline and seeds assumed to be outside the airways. He also proposes novel segmentation method. This approach is described in many details in his excellent thesis [6], so we started our research by implementing certain parts of his algorithm. The images we work for are of rats who had silicone casting material injected into their lungs.

The cast image differs from typical image of lung airways in three important ways. Firstly, there is a high degree of contrast between the airways and the parenchyma, i.e. airways and alveolar regions wherein the cast material did not penetrate. Secondly, the intensity of the cast material is close to that of bone and not far from that of the pulmonary vasculature. In other words, the casted airways are on the white end of the spectrum rather than the typically black end. Finally, the cast material often forms bubbles which appear as darker points in the domain and areas of brightness where contrast agent dissolved unevenly forming pockets.

Both airways and surrounding bone structure are locally tube like. Moreover their intensity values are close enough to not be a reliable criteria for distinction. Because of this similarities, tube-likeliness filters and further ridge extraction process in Bauer's algorithm recover centerlines of bones as well. That makes structural analysis in his approach too expensive. We also found the algorithm to fail on higher resolution images because tube filters tend to find several lines for each tube.

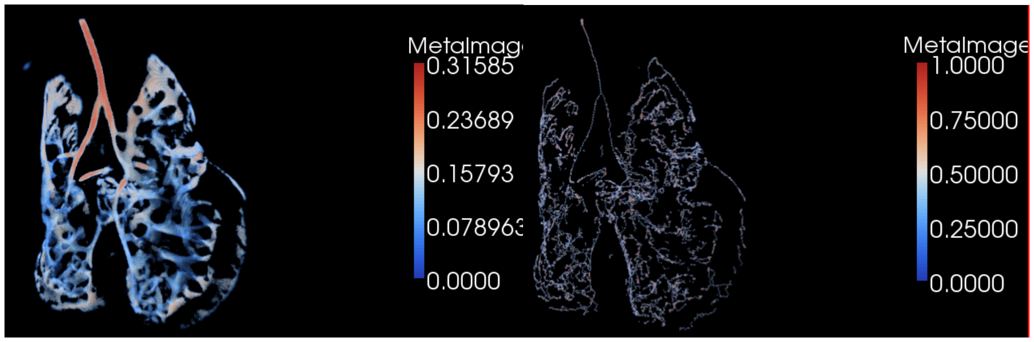


Figure 2.1: Response of Median Tube-Likeliness Filter and ridges extracted from it.

Arguably the most important contribution in [6] is application of Gradient Vector Flow field to this problem. Bauer used it to replace multi-scale Gaussian diffusion for tube-likeliness filters and for the segmentation. We found GVF to be extremely useful meta image and used it extensively in our approach.

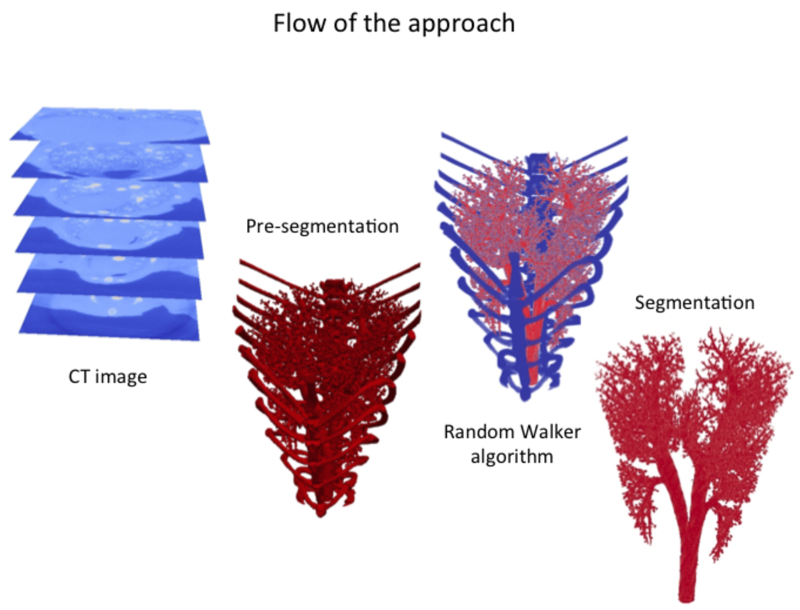


Figure 2.2: Flow of our method.

2.2 Gradient Vector Flow Field

Gradient Vector Flow (GVF) field was first introduced in [81] as an external force for active contours with the purpose of improving behavior of the snakes near concave boundaries. It is a diffusion of intensity gradients.

Given initial gradient vector field $F^n(x)$, GVF is defined as vector field $G(x)$ that minimizes

$$E(x) = \int\int\int_{\Omega} \mu|\nabla G(x)|^2 + |F^n(x)|^2|G(x) - F^n I(x)|^2 dx, \tag{2.2.1}$$

where Ω is the image domain and x is a voxel. μ is regularization parameter that can be adapted according to the amount of noise. As a result, gradients are smoothed out in homogeneous areas of the image and point towards objects (if background is dark compared to the object). At the objects boundaries gradients magnitudes have local extremums.

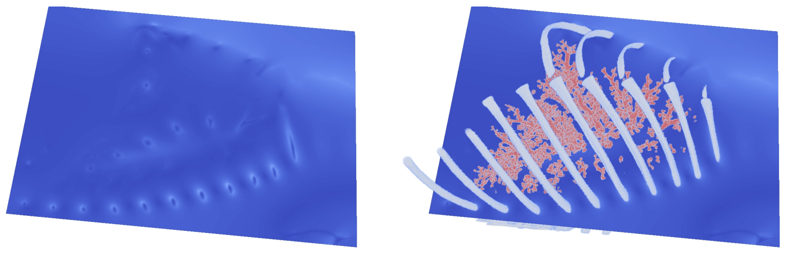


Figure 2.3: Slice of the image of Gradient Vector Flow field magnitudes. Extremums correspond to boundaries of the objects.

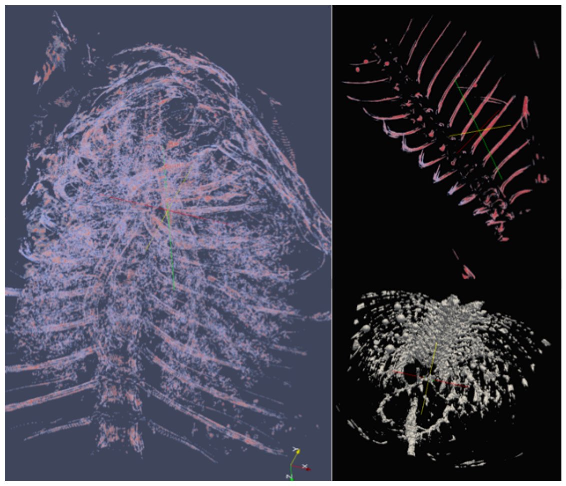


Figure 2.4: Subset of the image that can be extracted based on analysis of singular value decomposition and Hessians of local GVF and normalized GVF.

To minimize errors in estimated gradient directions, we compute first partial derivatives of the image using the coefficients given by Farid and Simoncelli ([18])
.

2.3 Flow of the Algorithm

The presence of contrast in the casted image might seem to suggest a connected threshold or similar intensity based algorithm, but such an approach quickly fails due to the presence of noise and the similarities between cast material, bone and blood vessels. Moreover, an intensity-based approach would leave gaps where bubbles have formed in the cast. Completely immersed bubbles could be filled with little effort, but bubbles on the boundary of the cast material would need to be manually filled. The shear number of bubbles makes such a strategy inadvisable. To some extent, application of an 'edge-preserving' filter would help to address the influence of noise, but the improvement would come inevitably at the loss of geometric detail, particularly at the smaller scale. In addition, the gradients become weaker at the smaller scale.

Gradient vectors are pointing away from dark areas of the image towards lighter areas. Both airways and bones are colored lighter then surrounding areas, while being relatively homogeneous in color themselves. Therefore direction of intensity gradients in the interior of the airways and bones are generally chaotic. Diffusing the gradient vector field by solving 2.2.1, results in the smooth field $G(x)$, at each vertex pointing towards the boundary of the nearest object. Since both airways and bones can be views as a collection of tube-like objects, GVF points away from centerlines. Then inverted field $-G(x)$ points towards centerlines and away from boundaries. Of course, this property is not unique to centerlines points but it does imply that superset of centerlines can be detected by local analysis of the GVF field. This assumption lies at the core of the method in [6]. Their execution of the analysis showed to be unreliable for datasets we used though. Power watersheds method [11] combined with GVF analysis seems to be a promising approach, but in this work though we used simplified method. We notice, that within airways and bones, $\operatorname{div}(-G(x)) < 0$, in particular $-1 \leq \operatorname{div}\left(-\frac{G(x)}{\|G(x)\|_2}\right) < 0$, where divergence equals to -1 at points located exactly on the centerline (Figure 2.5).

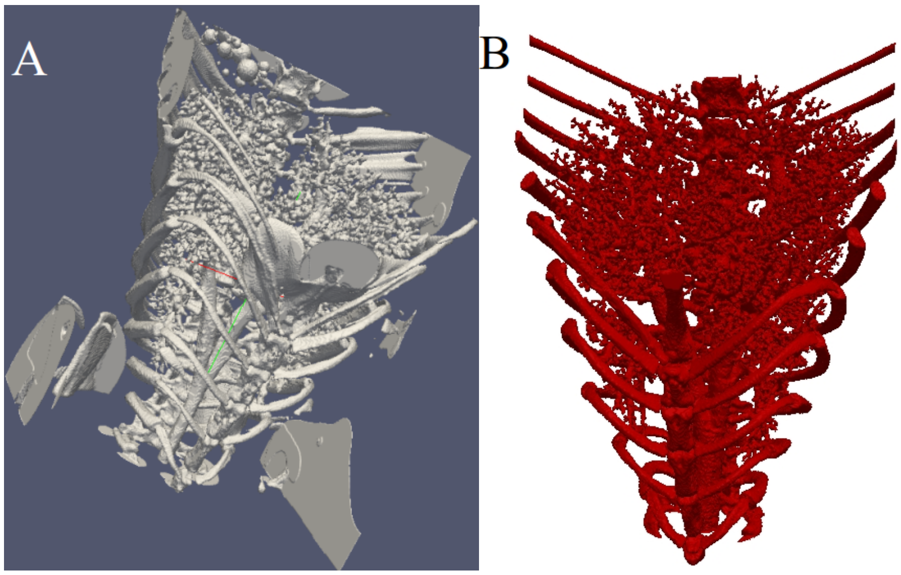


Figure 2.5: Panel **A**: Subset of the image with divergence of normalized GVF less then -0.001. Panel **B**: subset of the image extracted by intensity based connected threshold algorithm within subset in Panel **A**.

While this criteria alone is not strong enough to select a meaningful subset of the image, introducing it into intensity-based connected threshold algorithm allows for extraction of the image subset corresponding to the union of airways and surrounding rib cage (Figure 2.5). Connected threshold method has two parameters: initial voxel located within airways, and image intensity range. Seed can be chosen interactively by the user, or automatically, utilizing iterative clustering scheme ([79]) for finding a seed within trachea. Range of image intensity is chosen by the user.

2.4 Random Walks

To separate bones and airways we rely on a variant of Random Walker Algorithm[20]. We ask user to mark the airways and bones by selecting one or more voxel within each object, and assign labels $\{l_1, l_2\}$ to the selected points. Then we build a weighted graph $G = (V, E)$, where V are set of vertices v_i, $i = \overline{1, N}$, corresponding to image voxels; and E is a set of edges e_{ij} describing chosen connectivity pattern between neighboring voxels. We consider two neighborhood connectivity options: "$6-$connectivity" - voxels are connected to six immediate neighbors, so E is a subgraph of a square grid, and "$26-$ connectivity", which also includes diagonals of a square grid subgraph. For each unmarked vertex v_i we compute probability p_i^1 that a random walker, traveling from the vertex along edges of the graph, will first reach one of the seeds labeled l_1. Since there are only two labels, corresponding probability p_i^2 for random walker first reaching l_2, is $1 - p_i^1$. The likelihood of random walker traveling over the edge e_{ij} is proportional to the weight w_{ij}. Weighting scheme is chosen to bias random walker to avoid cross-boundary trajectories. Given the probabilities, each vertex v_i is labeled l_k, where $k = \arg\max_{k\in\{1,2\}}(p_i^k)$.

Probabilities at unseeded vertices are solutions to linear system of equations arising from discrete Dirichlet problem [16, 36]. Let $\boldsymbol{L}$ be the combinatorial Laplacian matrix

$$\boldsymbol{L}_{ij} = \begin{cases} d_i & \text{if } i = j, \\ -w_{ij} & \text{if } \exists e_{ij} \in E, \\ 0 & \text{otherwise} \end{cases} \tag{2.4.1}$$

where $d_i = \sum_{\text{all } j} w_{ij}$. Let $\boldsymbol{x}_M$ be a vector of all labeled vertices, and $\boldsymbol{x}_U$ be a vector of unlabeled vertices (we assume without loss of generality that vertices are numbered so that labeled vertices go first). Relationship between them is described by

$$\boldsymbol{L}_U \boldsymbol{x}_U = -\boldsymbol{B}^T \boldsymbol{x}_M, \tag{2.4.2}$$

where $\boldsymbol{L} = \begin{bmatrix} \boldsymbol{L}_M & \boldsymbol{B} \\ \boldsymbol{B}^T & \boldsymbol{L}_U \end{bmatrix}$ (from minimization of Dirichlet integral [20]). From the point of view of electrical circuit theory, which is the most intuitive, this system describes relationship between node's potentials satisfying Ohm's and Kirchhoff's laws. For each unseeded voxel there is a corresponding row (and column) in the system. To find probabilities $l_i^1$for all unlabeled vertices we set potentials of seeds labeled l_1 to 1, and seeds labeled by l_2 to 0.

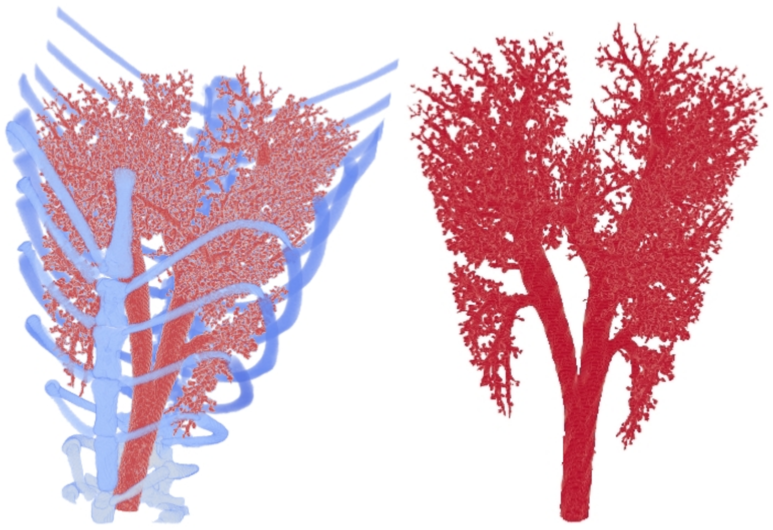


Figure 2.6: Figure on the left shows subset of the image corresponding to airways and bones colored by probability that random walker starting at the vertex will first reach seed in the airways. On the right are voxels at which the probabilities are greater then 0.5.

2.4.1 Implementation Details

We design all the functions in the framework to be able to work with images with resolution up to $1024 \times 1024 \times 1024$. If we apply Random Walker to this image directly, for example by assigning zero weights to all edges incident on at least one vertex corresponding to a voxel outside preselected area, the resulting system would have 1024^3 equations. Most of the rows would be zeros, but common sparse matrix structure used, for example, by MATLAB, has $nnz+n+1$ integers of overhead for storing $m \times n$ matrix with nnz non-zero elements. So for $n = 1024^3$ that is around 4Gb of additional memory used just to account for the matrix size. This matrix is reducible and can be treated as such by MATLAB in all relevant operations, but not without some efficiency costs. To avoid them we construct graph only based on preselected vertices.

Let $\tilde{X}$ be a set of preselected voxels, $\tilde{G} = \{\tilde{V}, \tilde{E}\}$ corresponding graph. Furthermore, we denote $\tilde{V}^{\text{in}}$ vertices corresponding to voxels whose all immediate neighbors are within the preselected area and $\tilde{V}^{\text{bd}}$ those on the boundary of preselected area. Assuming $6-$connectivity pattern chosen for the graph, there are $7 \cdot |\tilde{V}^{\text{in}}| + 4 \cdot |\tilde{V}^{\text{bd}}|$ non-zero elements in this matrix (for each interior vertex there are 6 edge weights and a diagonal value, for boundary vertices there are 3 edge weights and a diagonal value). Since matrix is symmetric number of non-zero elements stored can be decreased by only keeping non-zero elements in upper triangular part. Also, customized solver can recreate diagonal elements based on non-diagonal ones. Resulting matrix $\tilde{\boldsymbol{L}} \in R^{|\tilde{V}|\times|\tilde{V}|}$, so overhead for storing it in sparse form is only $|\tilde{V}|$ instead of n. We still need to keep the mapping from the vertices in the graph to corresponding voxels in the image, as it is not as straightforward as when graph is build on the whole image. The map requires additional $|\tilde{V}|$ integers of storage, but it can be deleted for the run-time of linear system solution method and recomputed later. We used this approach for most of the cases as it allows to leverage strength of the parallelized built-in MATLAB preconditioned conjugate gradient linear solver. Some of the disadvantages of this approach are additional operations needed for mapping between graph and image voxels and that MATLAB can only create sparse matrices with double precision (at least in the version we are using), while satisfactory results can be achieved with single precision data. So we also implemented matrix-free conjugate gradient method for solving the system which operates directly on the edge list and corresponding weights. Edge list adds $|\tilde{E}|$ integers to the amount of memory needed to store weights, which is somewhat less memory efficient then sparse matrix form for $\tilde{G}$.

2.4.2 Random Walker Weights

Common way of biasing random walker to avoid crossing boundaries is to weight the graph edges based on intensity levels at vertices. Likelihood of random walker crossing the edge is proportional to the weight. Therefore, when weight of the edge is a monotonic function of the inverse difference between colors of incident vertices, random walker tends to travel between most similarly colored voxels. When objects are intuitively defined as relatively homogeneous areas of different colors, random walker approach on a graph weighted this way handles weak boundaries remarkably well. In our case though, because of similarity between densities of the bones and the silicone filling, colors of the airways and bones are very similar (Figure 2.7). Under these circumstances, the length of the shortest path between the point and seeds is the main contributor to the solution.

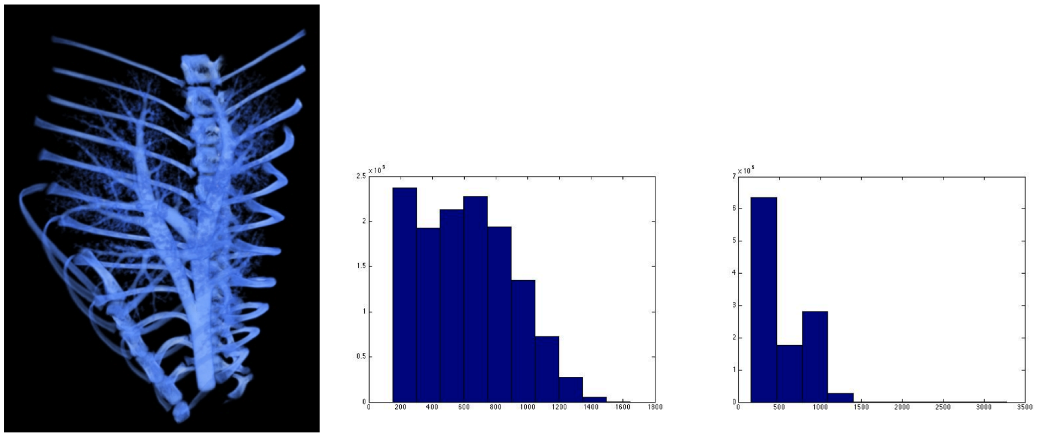


Figure 2.7: Airways and bones colored same as in the original CT image. Color histograms of bones and airways.

Since a random walker is likely to first reach the seed that is the closest, solution is highly dependent on seeds locations. The algorithm still provides good results when seeds with different labels are close enough to the boundary between objects, but fails otherwise. Since there is no other signal in the image to bias random walker but the intensity levels, we need some more sophisticated way to utilize the intensity information. We notice that the path between most of the points inside airways and most of the points inside bones would drastically bend at the boundary between them. While this is also true for many paths within rib cage, it doesn't matter because we are looking for most probable first seed. Based on this observation we would like to include shape of the path into weighting scheme to encourage random walker from taking a path that has globally sharp turn. Gradient Vector Flow field provides an excellent way to introduce awareness of local geometry into random walker algorithm. GVF propagates information about geometry of local boundary throughout interior in a vector field form. Angle between GVF vectors at vertices incident on edge gives an idea how much the local boundary is bended. The difference of GVF magnitudes is an indicator of a degree to which edge is aligned with a centerline. For those trajectories that do not bend when crossing the boundary, intensity biasing is quite effective as we mentioned earlier. Therefore we include difference in both magnitude and direction of GVF vectors at incident vertices into the formula for computing edge weights, as well as difference in intensity levels. The formula for the weight w_{ij} of the edge between v_i and v_j is

$$w_{ij} = e^{-\left(\alpha(1+\cos(\widehat{G_i,G_j}))+\beta|\|G_i\|_2-\|G_j\|_2|+\gamma|I_i-I_j|\right)}, \tag{2.4.3}$$

where G_i, G_j - Gradient Vector Flow Field at voxels corresponding to graph vertices v_i, v_j; I_i, I_j denote corresponding intensity levels, and coefficients α, β and γ are weights reflecting relative importance of each metric. Including GVF into consideration decreases dependence on the choice of seeds locations, and improves condition number of the matrix, which in turn, improves efficiency of iterative solvers. (condition number for matrix based on intensity weighting, condition number of matrix corresponding to weighting with GVF included).

2.5 Results

We illustrate our approach on one of the computed tomography images provided by our collaborators at The Pacific Northwest National Laboratory. The dataset represents the chest image of specific-pathogen-free adult male Sprague-Dawley rat (Charles River Laboratory, Raleigh, NC). The animal was sacrificed with CO_2 asphyxiation and prepared for lung airway casting, as described by Perry et al. [61]. Silicone casting material, consisting of 10.5 g Dow Corning 734 flowable sealant, 2.5 g Dow Corning 200 fluid (20 CS), and 1 ml Ultravist 300 (Bayer Healthcare) contrast agent, was thoroughly mixed and degassed under vacuum. Approximately 1.5 ml of the mixture was injected into the lungs. After about 1 hour, the cast was CT imaged in situ with the following parameters: 100 kVp, 50 mA, 20 ms exposure time. The image has resolution of $[384 \times 347 \times 512]$ voxels.

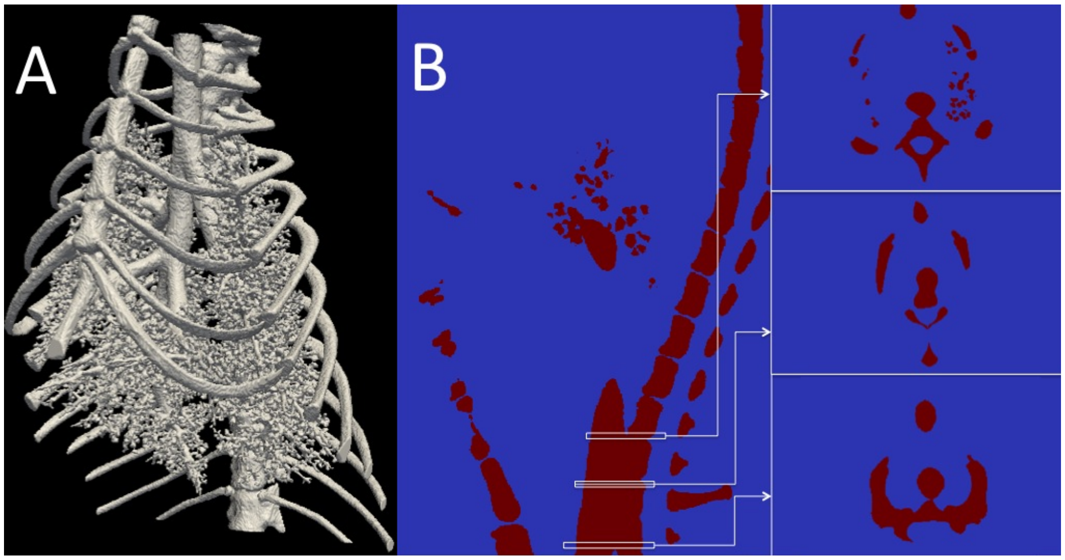


Figure 2.8: Panel **A** shows pre-segmentation of the image by connected intensity threshold algorithm within the subset of the voxels where divergence of normalized inversed GVF field is < -0.0001. Airways and bones are connected on slices 4 to 120. Panel **B** shows section yz of the sub-image and slices 4, 60 and 120.

Gradient Vector Flow field of the image was computed in 128.30 seconds on 8 core workstation with 26GB of RAM. Seed within trachea was manually selected and intensity range specified to be $[150, \max(I(x))]$. Pre-selection takes 7 seconds and result is shown on Figure 2.8. Next step is choice of seeds for Random Walker algorithm (Figure 2.9). Five seed vertices were chosen, 2 within bones and 3 within airways. Seeds were chosen arbitrarily, the only guideline was to pick at least one seed within both spine and sternum.

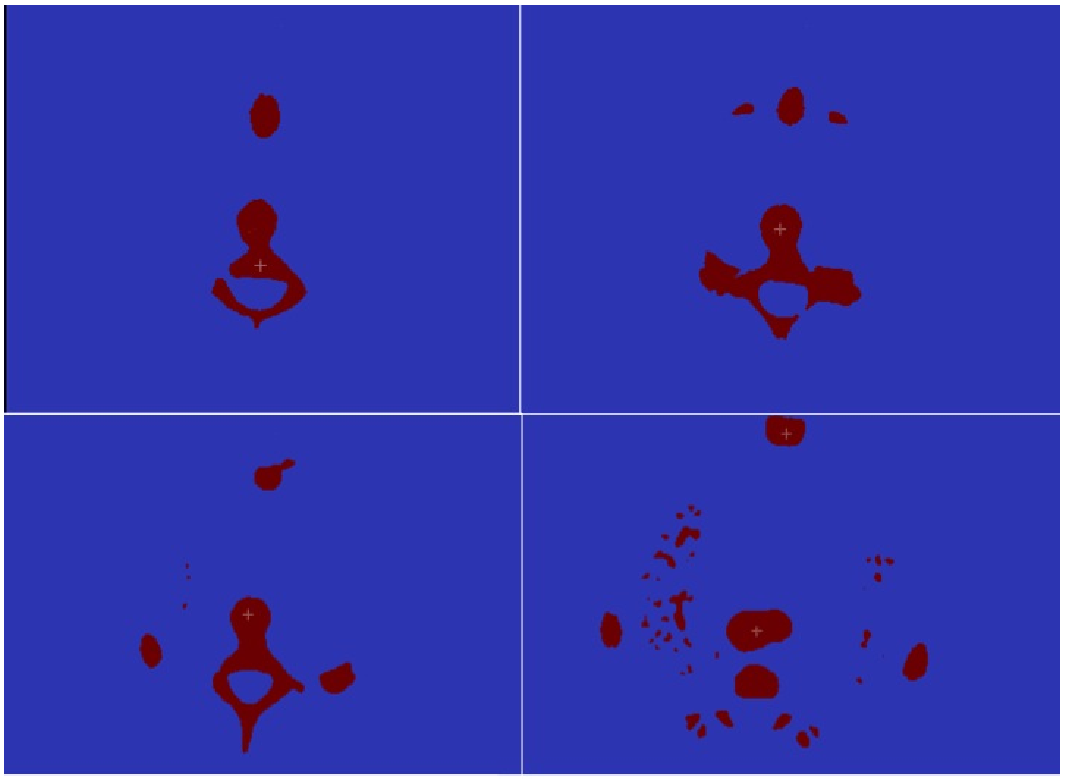


Figure 2.9: Seeds for initializing Random Walker Algorithm. From top left to bottom right corner seeds are [188, 131, 18] (bones); [184,159,43] (airways); [181,158,85] (airways) and [181,148,136] (airways) and [201,280,136] (bones).

Volume of the preselected subset of the image is only 5.7% of the whole image. Constructing the graph, including weights computation took 40 seconds. There are two linear systems of equations, one per each label in the image, constructed based on the graph and the choice of seed points. Their solutions add to $\mathbf{1}$, so one system was solved using preconditioned conjugate gradient method with incomplete Cholesky precondtioner, and solution to another one was found by simple subtraction of the first solution from $\mathbf{1}$. It took 29.13 minutes.

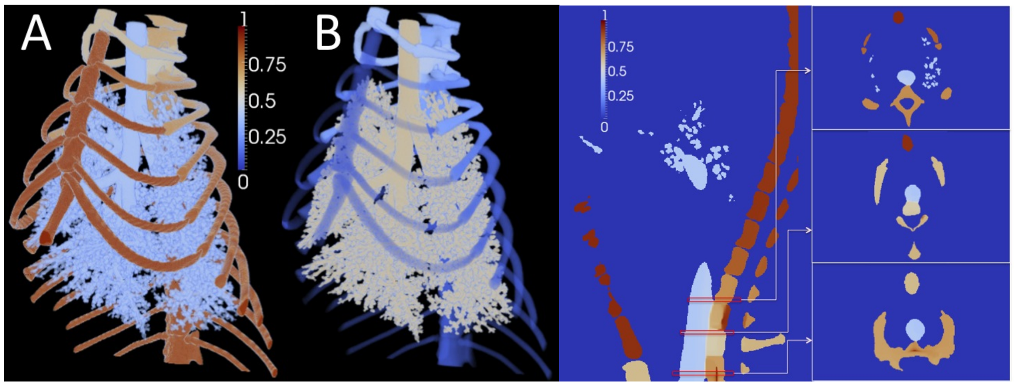


Figure 2.10: Solutions of the discrete Dirichlet problems associated with Random Walker algorithm. Panel **A** is the sub-image colored by the probabilities that Random Walker starting from the vertex will first reach on of the seeds inside the rib-cage. Panel **B** depicts solution to second linear system, which equals to vector of 1's minus first solution. On the right image are cross-sections of the image on Panel **A**

Final segmentation (Figure 2.11) is obtained by labeling voxels according to maximum probability at the point.

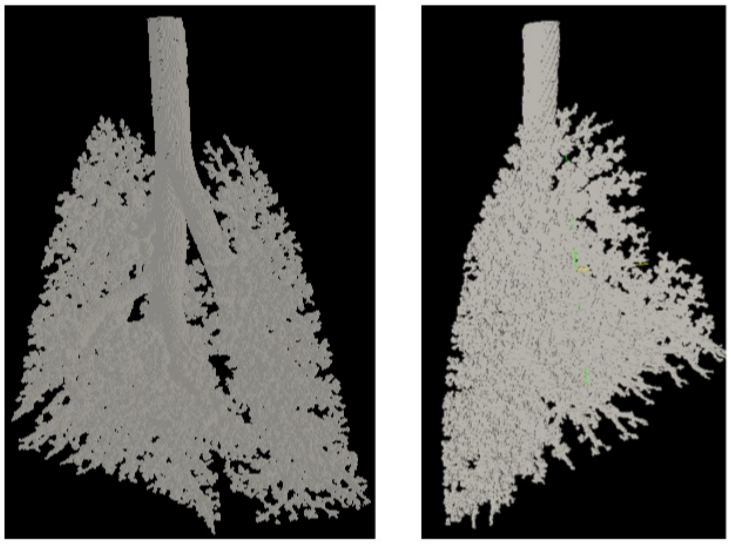


Figure 2.11: Segmentation of the airways.

Chapter 3

Medial Curves

Excerpts from this chapter appeared in [30] (c) SIAM.

The subject of this part is the construction of a 1-D representation (commonly known as the *curve skeleton*) of a 3-D object that is mostly composed of tubular substructures. Curve skeletons have a number of variants [9]. We define and compute a special type of curve skeleton, which we refer to as the *medial curves*.

3.1 Previous Work

There is a vast literature on extracting curve skeletons or similar 1-D curves from an explicit or an implicit representation of an object. We will only review a few representative references. We divide the approaches into two categories: algorithms that are based on the volume, and those based on the surface.

3.1.1 Volume-Based Approaches

Most volumetric methods begin with a binary voxelized image, derived from MRI (magnetic resonance) or CT (computed tomography) scans. Among these methods, *thinning* is probably the most popular paradigm for computing skeletons, due to its relative simplicity and efficiency. A thinning algorithm typically iteratively erodes the contours in a binary image until a single layer of pixels remain, and these pixels are then connected to form the skeleton. The process is based on Blum's grass-fire analogy of the medial axis [8]. There are many variants of thinning algorithms. See for example [45] for a survey on thinning in 2-D and see [46, 59] for algorithms in 3-D. Because the resolution of quantized to the voxel size, the precision of these methods are limited, so the resulting curves in general are not smooth and have limited accuracy. In addition, they are sensitive to the noise in the image. This sensitivity is often manifested as spurious branches, leading to incorrect structures.

In [77], smoothness is improved by adding a shrinking step before thinning and sensitivity to noise is alleviated by a pruning step, but centeredness and accuracy of the skeletons may still suffer.

Another class of methods is distance transform, which computes distance field for each interior point or voxel center (see e.g. [51], [7]). Ridges of the distance field are detected, yielding an approximate medial surface, which is then pruned. Similar to other voxel-based methods, the accuracy of these methods is constrained by voxel resolution. In addition, these methods can be sensitive to noise due to the instability of the medial axis. To alleviate the sensitivity to noise and improve accuracy, Cornea et al. proposed to compute the distance field based on a simulated electric potential [10]. However, their method is computationally prohibitive for practical image sizes. Wischgoll et al. proposed a variation of this approach, which positions discrete points on the boundary of the object with subvoxel precision based on intensity gradients in the voxelized image [80]. This approach considerably reduces the computational overhead, but it does not guarantee topological correctness.

3.1.2 Surface-Based Approaches

Surface-based approaches have two potential advantages over volumetric approaches. First, they are not limited by the voxel resolution. Second, they can potentially be much more efficient, as the number of surface points is usually orders of magnitude smaller than the number of image voxels for large datasets. The most common class of surface-based approaches begins by computing the medial surface from the Voronoi diagrams [2]. The Voronoi-based medial surfaces are then distilled into medial curves or medial skeletons by either clustering points, or by extracting the internal edges of the Voronoi diagram [14]. This approach is quite expensive. In addition, since small perturbations in the original surface can induce large perturbations in the medial axis, this approach is sensitive to noise, and the resulting medial curves are in general not smooth. In addition, approaches based on the medial surfaces can produce structurally incorrect medial curves for many biomedical applications. For example, the stable medial-axis based skeleton of a cylinder is a centered line that has four legs reaching out at 45 degrees to the top and bottom of the cylinder, while the desired solution is a straight, centered line that extends all the way from one end of the cylinder to the other.

Another class of surface-based approaches involves mesh contraction. For example, in a recent approach meshes are operated on directly by shrinking the boundary to a 1D graph, by applying a curvature flow and implicit Laplacian smoothing with global positional constraints [4]. This approach produces medial curves that are homotopic to the object, and it is reasonably efficient. However, the endpoints tend to retract into the geometry, and the medial curve tends not to be well centered.

Another type of method is based on Reeb and Morse graphs. Such methods are typically very efficient and can encode the correct surface topology [60], and they are closely related to many mesh segmentation methods. Typically, level set functions are defined on the surface and stable feature points are defined as the extrema of these so-called mapping functions. Various mapping functions have been defined for Reeb-graph methods, but it is in general difficult to define mapping functions that yield a geometrically correct skeleton. It is also difficult to transform a topological graph into a geometrically accurate skeleton [72]. Another approach to Reeb-graphs is to identify feature points on the surface before defining mapping functions by locating vertices at the extremities of surface components [38, 54]. However, stable feature point identification is challenging by itself and may be sensitive to noise.

Other related surface based approaches examine contours of intersection with the surface rather than contours of a mapping function [55, 56]. These methods are based on the observation that tubular networks can be decomposed into tubular and non-tubular regions by examining the intersection curves of inflated spheres. In the first of these, the spheres were centered at the vertices [55]. The study of the evolution of the intersection curves, γ_i, as function of the radius in terms of the number of connected components i, determines the local topology. Where $i = 1$ the neighborhood will be an extremity; where $i = 2$ the neighborhood will be tubular; where $i \geq 3$ the neighborhood will be a branch. Since every point on a medial curve is an endpoint, a segment or a branch point, the connection to medial curves is obvious. However, the sizes of the neighborhoods must be subjected to application specific parameters due to multiple ambiguities. In [56], only tubular sections were extracted, by first finding two intersection curves from spheres inflated at seed vertices, termed medial loops. The medial loops were then swept along the tubular section by collocating spheres located at the centroids of the medial loops rather than the seed vertices.

3.2 Requirements for the Medial Curve

Medial curves and other curve skeletons have a wide range of applications in geometric modeling and analysis, such as shape matching and shape retrieval, which are surveyed thoroughly in [9], though we mostly are interested in biomedical applications.

Biomedical applications imply following key requirements for the medial curve:

Topological and structural correctness: The medial curve should be *homotopic* to the object, and should represent the *structure* of the object by correctly labeling the *ends* and *branches*.

Smoothness: The medial curve should be *smooth* except at branches and distinct sharp turns.

Centeredness: The medial curve should be near the "*centers*" of the object.

In addition, an algorithm for computing medial curves must satisfy the following requirements.

Robustness: The algorithm must handle complex geometries with tens of thousands of branches without user intervention.

Noise resistance: The algorithm should tolerate moderate level of noise, which are unavoidable in image-derived geometries.

Efficiency: The algorithm must be fast and have a small memory footprint (i.e., have close to linear complexity in time and space) to handle large datasets with millions of triangles.

We focus on imaging-derived biomedical geometries. We assume the input geometry is given by a surface triangulation, such as that obtained by an isosurfacing algorithm (see [57] for a survey of such algorithms). The decision of using surface triangulations (instead of using volumetric data) is for the considerations of accuracy, efficiency, and application requirements.

We take a novel perspective on defining and computing medial curves, and make a few contributions toward the better understanding, and more robust computation, of medial curves. Our first contribution is a concept called the *interior center of curvature* (ICC) of surfaces. For canonical shapes (such as spheres, cylinders and tori) and more generally the *canal surfaces* [21, Sec. 20.2], the ICCs of the surface give precisely their medial curves. For general surfaces, we define the *average interior center of curvature* (AICC) as approximations of ICC with improved noise resistance, and use AICCs as a starting point for finding the medial curves.

Our second contribution is to define the medial curve based on a *local orthogonal decomposition* of the object into tips, segments, and junctions. We define this decomposition based on geometric quantities (including AICC and orthogonality) and topological invariance (the Euler characteristic). The medial curve is then the graph connecting the centers of these substructures and of the cutting planes between them.

Third contribution is a new algorithm for the computation of medial curves, called *local orthogonal cutting* (*LOC*). Our algorithm is robust and resistant to noise, and it has nearly linear time and space complexities. We demonstrate the effectiveness of our method for complex biomedical geometries derived from medical images, including lung airways and blood vessels, and present comparisons of our method with some existing methods.

3.3 Geometry and Topology of Medial Curves

In this section, we present some basic geometric and topological concepts. We consider a volumetric object Ω embedded in $\mathbb{R}^3$. The *surface* is the boundary of the object, denoted by $\partial\Omega$, so it is by definition a 2-manifold without boundary. We will define the *medial curve* of the object Ω (or equivalently, the *medial curve* of its surface $\partial\Omega$). We assume the surface is given by a triangulation. By convention, surface normals are outward, so the curvature is positive if the object is locally concave and is negative if it is locally convex. Without loss of generality, we assume the object contains a single connected component. For an object with multiple connected components, all of our discussions apply to each connected component of the object.

3.3.1 Canal Surfaces and Centers of Curvature

To define the medial curve, we first consider a special but important class of surfaces called *canal surfaces* [21, Sect. 20.2], which are closely related to the medial axis and centers of curvatures.

Canal Surfaces and Medial Axis

A canal surface is the envelope of a 1-parameter family of spheres in $\mathbb{R}^3$. The notion of canal surface is quite general: all tubes and most surfaces of revolution are canal surfaces. The curve formed by the centers of these spheres is called the *center curve* of the canal surface [21, page 646]. Figure 3.1 shows some examples of canal surface and their center curves. In this paper, we refer to the center curve as the *medial curve*, as it is a special case of the *medial axis* for general surfaces, as we will explain next.

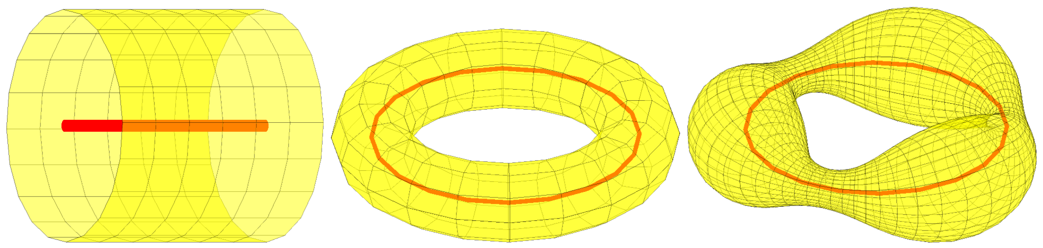


Figure 3.1: Examples of medial curves (red) of a cylinder, a torus, and a surface generated by a sphere of radius $0.3 + 0.15\sin(3t)$ moving on a unit circle.

Although the medial axis is typically explained using the grass-fire analogy, it was originally motivated by the *medial axis transformation* [8], which is the process of computing the medial axis and the local feature size (lfs) and then approximating the surface (or curve) using the envelope of the spheres (or circles) that are centered on the medial axis and have radii equal to the lfs. The local feature size is therefore also called the *radius* of the object. If the medial axis is a curve (namely, the *medial curve*), then the envelope of these spheres is either a canal surface (if the medial curve is a closed curve) or a collection of canal surface patches (if the medial curve has branches). Many geometries, especially biomedical geometries, can be well approximated as a union of canal surfaces (more precisely, the union of the volume bounded by canal surfaces). Therefore, it is important and valuable to first understand canal surfaces and their medial curves. For more complex geometries, we will develop numerical approximations and introduce other measures to address the additional issues such as junctions and open-ends.

Interior Center of Curvature

To help understanding canal surfaces, we introduce some new concepts. The *interior curvature* (*IC*) is the most negative curvature at a point. We refer to the reciprocal of the absolute value of IC as the *interior radius of curvature* (*IRC*). The point that has a distance equal to IRC along the inward surface normal of $\boldsymbol{p}$ will be called the *interior center of curvature* (*ICC*). More precisely, given a point $\boldsymbol{p} \in \partial\Omega$, let $\hat{\boldsymbol{n}}_p$ denote the unit outward surface normal to Γ at $\boldsymbol{p}$, κ_{p1} and κ_{p2} denote the principal curvatures at $\boldsymbol{p}$. Then the *interior curvature*, denoted by κ_p^-, is the more negative value of κ_{p1} and κ_{p2}, i.e.,

$$\kappa_p^- = \begin{cases} \kappa_{p1} & \kappa_{p1} < 0 \text{ and } \kappa_{p1} \leq \kappa_{p2}, \\ \kappa_{p2} & \kappa_{p2} < 0 \text{ and } \kappa_{p2} < \kappa_{p1} \\ -\epsilon & \kappa_{p1} \geq 0 \text{ and } \kappa_{p2} \geq 0 \end{cases} \tag{3.3.1}$$

where ϵ is an infinitesimal number (e.g., 10^{-100} for double-precision floating-point arithmetic) and is introduced to avoid division by zero. The *interior center of curvature* (*ICC*) at $\boldsymbol{p}$ is then

$$\boldsymbol{c}_p = \boldsymbol{p} + \frac{1}{\kappa_p^-}\hat{\boldsymbol{n}}_p. \tag{3.3.2}$$

The ICC is the center of an *osculating sphere*, which is tangent to the surface at $\boldsymbol{p}$ and has a radius equal to the IRC.

The above definitions of IRC and ICC seem to be new and are applicable to any smooth surfaces. For a canal surface Γ, the IRC is the local feature size of the surface, and the IRCs form the medial curve of Γ. The osculating sphere intersects a canal surface Γ at a circle, which we refer to as an *intersection circle*. If the radius of the canal surface is constant, then this circle is a principal curve and is orthogonal to Γ (see Theorem 20.14 in [21, page 649]).

Average Interior Center of Curvature

The curvatures are second-order differential quantities, so the ICC are inherently sensitive to noise. To improve noise resistance for surface triangulations, we define the *average interior center of curvature* (AICC) of a surface patch sampled by a discrete point set $P = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_m\}$ to be a weighted average of the ICCs at these points, i.e.,

$$\bar{\boldsymbol{c}} = \frac{\sum_{i=1}^{m} w_i \boldsymbol{c}_i}{\sum_{i=1}^{m} w_i}, \tag{3.3.3}$$

where $\boldsymbol{c}_i$ denote the ICC at point $\boldsymbol{p}_i$. We choose the weight w_i to be $\left(\kappa_i^-\right)^2$ times a control area a_i of the vertex i (where the control area is chosen to be one third of the sum of the areas of the incident triangles of $\boldsymbol{p}_i$). This weighting scheme gives higher priorities to more curved areas and lower priorities to flatter areas. Therefore, it tends to underestimate the average radius of curvature and in turn stabilizes the AICC. For better robustness, we sometimes set the weights to zeros for the points that are "invisible" from a particular viewpoint, as we will explain in Section 3.5. The AICC of a sphere is exactly the center of the sphere. The AICC of a small patch of a tube is close to the medial axis, and it converges to a point on the medial axis as the size of the patch tends to zero. Therefore, AICC preserves the key features of ICC while allowing better noise resistance.

AICC-Centered Minimum Containing Sphere

Besides ICC, the osculating sphere is an important concept for canal surfaces. An osculating sphere intersects a canal surface at a circle. Such an intersection is unstable to compute in the presence of numerical errors. For better stability, we approximate an osculating sphere by a sphere that is centered at the AICC with a slightly larger radius, such that this enlarged sphere would intersect the surface at a finite area. We refer to the enlarged sphere as the *AICC-Centered minimum containing sphere* (MCS) of the area, and it will be useful in our computation of medial curves later.

3.4 Local Orthogonal Decomposition

Our objective now is to define the medial curves for surfaces that are mostly composed of canal surfaces. We propose a decomposition of an objects into three different types of sub-structures, and the definition of medial curves will naturally follow from this decomposition.

3.4.1 Decomposition of Objects

Suppose a given object is mostly composed of a set of canal surfaces, referred to as the *component canal surfaces* (CCS). Imagine moving a planar knife to cut the object through the intersection circles of the CCSes. We refer to the cross-section of the knife with the object as a *cut*. In general, the cuts do not intersect with each other, and any point in the object belongs to at most one cut. If the medial axis γ of one CCS is not closed, then the cut corresponding to an open-end of γ cuts off a *tip* of the object. At an intersection of multiple CCSes, the cutting planes are ill-defined, but such an intersection may be identified by the well-defined cuts from their adjacent CCSes.

In summary, the above process decomposes an object into three types of substructures: the *segments*, each of which is composed of continuous cuts and bounded by two cuts, the *tips*, each of which is incident on one cut, and the *junctions*, each of which is incident on three or more cuts. The cuts also decompose the surface of the object into surface patches, and for convenience we refer to these surface patches as tips, segments, and junctions as well. Figure 3.2 shows an illustration of these three types of local structures and their corresponding cuts.

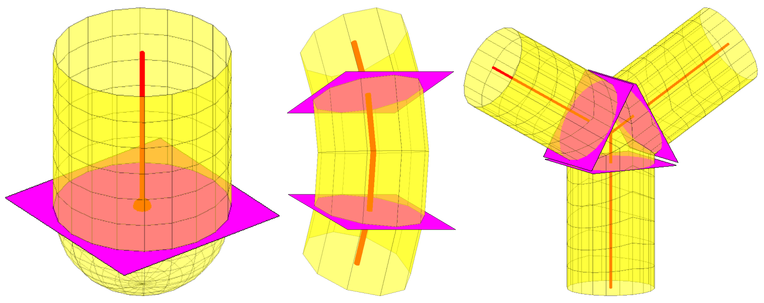


Figure 3.2: Illustration of tip, segment, and junction, along with orthogonal cuts (blue planes). Red curves show medial curves.

Given this above decomposition, we define the *medial curve* of the surface to be the union of the medial curves of the segments as well as the edges that connect the "junction centers" with the end-points of the medial curves of their adjacent segments. This decomposition also determines the topology of the medial curve: the *end-points* of medial curve correspond to the tips, and the *bifurcations* and *multifurcations* of the medial curve correspond to the junctions. The key of computing the medial curve is then the computation of such a decomposition of the object, which we describe in the next subsection.

Remark 1. Note that for a given object, there are a constant number of tips and junctions, but there can be an arbitrary number of cuts. For efficiency, it is desirable to have the lengths of the segments to be proportional to the local diameter of the object and potentially be dependent on the curvature, so that the medial curve of each segment can be approximated by a line segment, and the number of edges in the medial curve is independent of the resolution of the surface triangulation.

3.4.2 Approximation by Orthogonal Cuts

Computing the exact intersection circles is a difficult and potentially unstable process. For simplicity, we assume the radii (i.e., the local feature sizes) of the component canal surfaces do not change rapidly, so that the cutting plane are nearly orthogonal to the surface and the ICCs are approximately at the centroid of the cutting plane. Thereafter, we can locate the cuts by computing cutting planes that are as orthogonal to the surface as possible. This concept of near orthogonality will lend itself to robust numerical optimization techniques.

A plane is uniquely determined by a point and a normal direction. We will determine the points based on the AICC and the centroid. For the normal direction, we measure orthogonality using two different types of quantities: *angle-based* and *eigenvalue-based*. Given a plane P, let $\hat{\boldsymbol{n}}_P$ denote the unit normal to P, and $\hat{\boldsymbol{n}}_\tau$ denote the unit face normal of a triangle τ. The dihedral angle between P and τ is $\arccos|\hat{\boldsymbol{n}}_P^T\hat{\boldsymbol{n}}_\tau|$, which ideally should be $90°$ but can be as small as $0°$. We therefore define the angle-based measure as the minimum dihedral angle between P and the triangles intersecting P.

The minimum dihedral angle is useful for measuring the quality of a given plane, but by itself it cannot facilitate optimizing orthogonality. For the latter purpose, we introduce an eigenvalue-based measure. Consider the matrix

$$\boldsymbol{N} = \sum_{\tau} w_\tau \hat{\boldsymbol{n}}_\tau \hat{\boldsymbol{n}}_\tau^T, \tag{3.4.1}$$

which we refer to as the *normal covariance matrix*. If the plane P intersects a cylinder nearly orthogonally, then all the normals $\hat{\boldsymbol{n}}_\tau$ are coplanar, and $\boldsymbol{N}$ is a rank-two matrix. Let σ_2 and σ_3 denote the two smaller eigenvalues of $\boldsymbol{N}$, and then $\rho = \sigma_3/\sigma_2$ is then a measure of how close the surface is to a cylinder locally. In general, ρ is between 0 and 1, and the smaller the better. If ρ is sufficiently small, then the eigenvector of $\boldsymbol{N}$ corresponding to σ_3 may provide a better normal for the plane.

Remark 2. The angle-based and eigenvalue-based measures complement each other in algorithm design. The former is a measure of a given plane, whereas the latter is a measure for potentially a better plane, and the corresponding eigenvectors can facilitate optimizing the normals of the plane. Both of these measures are numerical in nature, so some tolerances are needed, which we will discuss in Section 3.5.3.

Topological Invariants

Besides geometric measures, important information about the local structure of the object can be obtained by analyzing the topology of a surface patch P. In particular, we classify P based on whether it is connected, whether it is closed, and whether it has tunnels. To capture the first two criteria, we consider the number of connected components of P and the number of connect components of the border curves of P and a simple counting. The definitions of these quantities are trivial, and their computations require only a breadth-first search. For the third criterion, we must consider the Euler characteristic of P. Given a polygonal surface (such as a triangulated surface), the *Euler characteristic* is the alternating sum:

$$\chi \equiv \#\text{vertices} - \#\text{edges} + \#\text{faces}. \tag{3.4.2}$$

For a tip, segment, or junction, the surface patch is an open surface. The Euler characteristic is 1 for the surface of a tip, 0 for a segment, and a negative number for a junction. Therefore, χ gives an alternative means for classifying tip, segment, and junction. In the next section, we present an algorithm for decomposing an object using the AICC, orthogonality, and Euler characteristic.

3.5 Computation of Medial Curves

In the preceding section, we defined the medial curves based on a decomposition of an object into tips, segments, and junctions, and introduced measures for these local structures and for the cuts between them. We hereafter present an algorithm, called *local orthogonal cutting* (*LOC*), based on these intuitive definitions. At a high level, our algorithm has the following five steps: 1) compute the differential quantities, 2) identify stable tips, 3) identify other patches, 4) fine-tune cuts and segments, and 5) reconstruct the medial curve. We describe each of these steps in the following subsections and present the analysis, data structures, and some implementation issues in Section 3.5.6.

3.5.1 Computing Differential Quantities

The identification of candidate cuts relies on the concept of interior center of curvature. Thus, we begin by computing the geometric quantities of the surface, including the normals and principal curvatures at the vertices as well as the normals and areas of the triangles. The surfaces in our applications are inherently noisy, as they are typically derived from medical images. For stability and noise resistance, we compute these vertex quantities by applying a quadratic fitting over the two-ring neighborhood of each vertex (i.e., the set of vertices that are within two edges away from the vertex) and solve the fitting using the weighted least squares framework in [33]. From the normals and principal curvatures, we obtain the interior curvature (IC) and the interior center of curvature (ICC) using Eqs. (3.3.1) and (3.3.2), and then compute the AICC of a face incident on each vertex using Eq. (3.3.3) for improved noise resistance.

3.5.2 Local Dimensionality of Object

The medial curve is a lower-dimensional representation of an object. To understand the geometry and topology of the medial curve, we must therefore have some conceptual understanding of the object itself. One particularly important concept is the *local dimension* of some finite neighborhood of the surface around a point.

Given an object Ω, let $\boldsymbol{p}$ be a point in Ω. Consider a neighborhood of $\boldsymbol{p} \in \Omega$ of some finite size that is greater than the local diameter of the object. This neighborhood of $\boldsymbol{p}$ may have different characteristics depending on its location in Ω, in that it may appear to be more like a tube, a disk, or a ball.

Mathematically, let $N(\boldsymbol{p})$ denote the neighborhood of $\boldsymbol{p}$ in Ω. The centroid, or the center of mass, of $N(\boldsymbol{p})$ is a weighted average of the positions. More formally, the centroid of $N(\boldsymbol{p})$ is

$$\boldsymbol{c} = \left(\int_{N(\boldsymbol{p})} \boldsymbol{x} d\boldsymbol{u} \right) \bigg/ \left(\int_{N(\boldsymbol{p})} d\boldsymbol{u} \right), \tag{3.5.1}$$

where $\boldsymbol{u}$ denotes a conceptual local parametrization of $N(\boldsymbol{p})$. Consider the 3×3 covariance matrix

$$\boldsymbol{M} \equiv \int_{N(\boldsymbol{p})} (\boldsymbol{x} - \boldsymbol{c})(\boldsymbol{x} - \boldsymbol{c})^T d\boldsymbol{u}. \tag{3.5.2}$$

In general, $\boldsymbol{M}$ is symmetric and positive semi-definite (i.e., its eigenvalues are non-negative). Consider its eigenvalue decomposition

$$\boldsymbol{M} = \boldsymbol{X} \boldsymbol{\Lambda} \boldsymbol{X}^{-1}, \tag{3.5.3}$$

where $\boldsymbol{X}$ is an orthogonal matrix composed of the eigenvectors of $\boldsymbol{M}$, and $\boldsymbol{\Lambda}$ is the diagonal matrix $\operatorname{diag}(\lambda_1, \lambda_2, \lambda_3)$ composed of the eigenvalues λ_i with $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0$. Assume the neighborhood has a size that is larger than the local diameter of the object. The local dimensionality of the neighborhood, denoted by d, is then the number of relatively large eigenvalues of $\boldsymbol{M}$, i.e., $\lambda_1 \approx \cdots \approx \lambda_d \gg \lambda_{d+1}$. This eigenvalue-based interpretation will be helpful for understanding the definitions and algorithms in this paper. Note that the criteria for $\approx$ and $\gg$ may be problem dependent. There may also be some inherent ambiguities if $\lambda_1 \gtrapprox \lambda_2$ and $\lambda_2 \gtrapprox \lambda_3$ over the whole object. In such situations, the decomposition is intrinsically ambiguous and hence inherently unstable.

3.5.3 Identifying Stable Tips

As defined in Section 3, a tip is a topological disk in the surface. However, not every disk corresponds to a tip. The objective of this step is to determine the disks that are *stable tips*, which are critical for the stability of the medial curve. Our goal is to avoid false positiveness while being as accurate as possible. Due to its complexity, we outline a high-level pseudocode of this step in Algorithm 1. The procedure has some key aspects, which we describe in more detail as follows.

Algorithm 1 Identify stable tips.

```
 1: for each convex seed vertex $v$, in ascending order of IRC
 2:     if $v$ has been visited; then continue
 3:     $S \leftarrow$AICC-centered MCS of a face incident on $v$;
 4:     $\chi_{\text{old}} \leftarrow 1$;
 5:     for $i = 1, 2, ...$
 6:         obtain patch $\Gamma$ in $S$ and cut $P$;
 7:         fill in small holes in $\Gamma$;
 8:         $\chi \leftarrow$Euler characteristic of $\Gamma$;
 9:         if $\chi \neq 1$ and $\chi_{\text{old}} = 1$
10:             save non-disk patch;
11:         elseif $\chi \neq 1$
12:             resort to saved tip; break;
13:         else
14:             if $P$ is highly orthogonal to $\partial\Omega$
15:                 save patch as tip; break;
16:             elseif $P$ is more orthogonal than last saved
17:                 save $P$ and patch as tip for resort;
18:             end if
19:         end if
20:         $S \leftarrow$ MCS of $\Gamma$ centered at its AICC;
21:         $\chi_{\text{old}} \leftarrow \chi$;
22:     end for
23:     if any tip was saved
24:         mark triangles intersecting or below $P$ by tip ID;
25:         mark vertices of the above triangles as visited;
26:         associate cut with a seed face incident on border edge;
27:     else
28:         mark vertices in saved non-disk patch as visited;
29:     end if
30: end for
```

Selecting and Sorting Seed Vertices To identify stable tips, we select a dense sampling of mesh vertices as seeds, based on their IRCs. We need at least one seed per tip. A tip must have a vertex that is convex. Therefore, we require each seed vertex to be convex, i.e., whose principal curvatures are both negative. To reduce the number of seeds, we impose an upper bound on the IRC of seed vertices. A tight upper bound would be the local radius of the object in the neighborhood near the vertex, but estimating the local radii is overly complex and expensive for our purpose. Instead, we set the upper bound to be half of the minimum dimension of the bounding box of the object.

The ordering of seed vertices is also important. With an arbitrary ordering, a large feature might subsume a smaller one and in turn cause small but important tips to be missing. Thus, we sort the seed vertices in ascending order of IRC (cf. line 1 of Algorithm 1), so that smaller features would be processed first. We go through the seed vertices one by one, and skip those that have been marked as visited in earlier iterations (cf. lines 2, 25 and 28). In the following we describe the sub-steps for each individual seed vertex.

Growing Candidate Patch Starting from a seed vertex, we grow a candidate patch to seek a stable tip. We obtain a candidate patch starting from an incident face of the seed vertex, and grow the patch through the aid of the MCS of the candidate patch centered at its AICC (cf. lines 3, 6 and 20). To increase the growth rate, we enlarge the MCS slightly (typically by 5%). To guard against strongly bended areas, we include a simple "visibility condition" in the computation of AICC (lines 3 and 20). Specifically, let $\boldsymbol{o}$ denote the AICC of the surface patch in the previous iteration. Given a vertex $\boldsymbol{p}$ in the surface patch, let $\hat{\boldsymbol{n}}$ denote the outward unit normal at $\boldsymbol{p}$. We set the weight for $\boldsymbol{p}$ to 0 in (3.3.3) if $\hat{\boldsymbol{n}}^T(\boldsymbol{p} - \boldsymbol{o}) \leq 0$.

In line 6 of Algorithm 1, we obtain the triangles that intersect or fall within the sphere and that at the same time are not yet assigned to any tip. These triangles are found efficiently through a breadth-first search on the dual graph of the mesh. To reduce sensitivity of the topology to small perturbations, we fill in small holes in the candidate patch (cf. line 6), including the holes that each correspond to a single face, two adjacent faces along an edge, or the one-ring neighbor of a vertex (or more concisely, the *star* of a face, edge, or vertex). From the candidate patch, we also obtain a candidate cutting plane P that is nearly orthogonal to the surface. The determination of the cutting plane is fairly involved, which we defer to Section 3.5.3.

We continue to grow a candidate patch until it reaches some *steady state* topologically and geometrically, so that a small perturbation would less likely affect the qualitative behavior of the medial curve. Topologically, we consider a patch to be non-disk only if it is non-disk for at least two consecutive iterations (cf. lines 9–12). This persistency requirement avoids the situation where a non-disk is introduced due to a small perturbation during an intermediate step. A variable χ_{old} was introduced to facilitate this persistency checking (cf. lines 4, 9, and 21). Geometrically, we consider the orthogonality of the cut, measured by the minimum dihedral angle as defined in Section 3.4.2. Our basic idea is to look for the patch with the most orthogonal cut (cf. lines 16–17). For better efficiency, we consider the plane as *highly orthogonal* if the minimum angle is close to $90°$ (say $\geq 75°$). A topological-disk patch with a highly orthogonal cut is accepted as a tip immediately (cf. lines 14–15). For robustness, we consider a cut as *not orthogonal* if the minimum angle is smaller than $45°$, and we impose an upper bound on the number of iterations of the growth (cf. line 5). In practice, we found an upper bound of 20 to 30 works well. After identifying a tip, we label the triangles that intersect or are below the cut within the candidate patch by a unique integer ID for the tip (cf. lines 24). In addition, we associate the cut with a seed face that is incident on the border edge of the surface patch (cf. lines 26). This seed face will be used in the next step for determining the connectivity of the tips with adjacent patches.

Computing and Optimizing Cut To determine stable tips, the most involved and critical sub-step is the determination of the cut for a given candidate patch, as described in more detail as follows. This procedure is employed in line 6 of Algorithm 1. It will also be a basic building block in later steps of our overall algorithm.

Given a candidate patch Γ that is a topological disk, we determine a cutting plane in an iterative manner starting from its border curve. A cutting plane is uniquely determined by its normal direction and a point, which we refer to as an *anchor point* of the plane. To determine a candidate cut, we first determine a normal direction using a principal component analysis of the border curve $\partial\Gamma$. In particular, let $\boldsymbol{c} = \int_{\partial\Gamma} \boldsymbol{x} ds / \int_{\partial\Gamma} ds$ be the centroid of the border curve, where s denotes a parametrization of $\partial\Gamma$. The matrix

$$\boldsymbol{A} = \int_{\partial\Gamma} (\boldsymbol{x} - \boldsymbol{c})(\boldsymbol{x} - \boldsymbol{c})^T ds \tag{3.5.4}$$

is the covariance matrix of the curve. We obtain an initial normal direction $\boldsymbol{n}_0$ to the plane as the third eigenvector of $\boldsymbol{A}$ (i.e., the eigenvector corresponding to the smallest eigenvalue of $\boldsymbol{A}$). Let $\boldsymbol{s}$ denotes the area-weighted average of the centroids of the triangles incident on $\partial\Gamma$. We choose the sign of $\boldsymbol{n}_0$ to satisfy $(\boldsymbol{c} - \boldsymbol{s})^T \boldsymbol{n}_0 > 0$, so that $\boldsymbol{n}_0$ points outwards with respect to the region between Γ and the plane.

After obtaining the normal, we select an anchor point for the plane so that all the triangles incident on the border curve are just above the plane. This condition ensures that the surface patches corresponding to different tips would not overlap at any vertex. It will be important in ensuring the remainder of the surface can be decomposed into valid patches, as we will explain shortly. We consider the plane as the *base cut* if it is orthogonal to the surface, and in addition if the AICC is below the plane. If any of these conditions is violated, we cannot extract a cut and instead would continue grow the patch.

After determining a base cut, we further optimize its location and orthogonality. Algorithm 2 shows the pseudocode for this procedure. Ideally, the cut should contain the AICC, but there may not exist a plane passing through the AICC while being orthogonal to the surface. For robustness, we instead search for an orthogonal cut that is as close to the AICC as possible. This requires adjusting both the position as well as the orientation of the cut. We achieve this in a double loop: In the outer loop, we perform a bisection procedure to adjust the position, and in the inner loop we adjust the orthogonality.

Algorithm 2 Optimize position and orientation of cut starting from base cut $P_0 = (\boldsymbol{p}_0, \boldsymbol{n}_0)$ and desired center $\boldsymbol{o}$.

```
1: $\boldsymbol{p} \leftarrow \boldsymbol{o}$;
2: for $i = 1, 2, \ldots$
3:     $P \leftarrow (\boldsymbol{p}, \boldsymbol{n}_0)$;
4:     for $j = 1, 2, \ldots$
5:         Compute $\tilde{\boldsymbol{n}}$ from normal covariance matrix of $P \cap \Gamma$;
6:         $\tilde{P} \leftarrow (\boldsymbol{p}, \tilde{\boldsymbol{n}})$;
7:         if $\tilde{P}$ is safe and is more orthogonal than $P$
8:             $P \leftarrow \tilde{P}$;
9:         else
10:            break;
11:        end if
12:    end for
13:    if $P$ is orthogonal
14:        return $P$;
15:    end if
16:    $\boldsymbol{p} \leftarrow (\boldsymbol{p} + \boldsymbol{p}_0)/2$;
17: end for
18: return base cut $P_0$;
```

We first describe the inner loop. Our key idea is to use the normal covariance matrix to obtain an improved normal direction. In particular, let γ denote the intersection curve of a candidate plane P with the surface patch Γ. The normal covariance matrix is $\boldsymbol{N} = \int_\gamma \hat{\boldsymbol{n}}\hat{\boldsymbol{n}}^T ds$, computed as in (3.4.1). Consider the eigenvalue decomposition of $\boldsymbol{N}$. If the smallest eigenvalue of $\boldsymbol{N}$ is far smaller than the other two (in practice, $\lambda_3 \leq 0.5\lambda_2$), let $\tilde{\boldsymbol{n}}$ be the third eigenvector of $\boldsymbol{N}$ with a sign such that $\tilde{\boldsymbol{n}}^T\boldsymbol{n}_0 > 0$. Let $\tilde{P}$ denote a plane centered at the same anchor point as P but with normal $\tilde{\boldsymbol{n}}$. We check whether $\tilde{P}$ intersects only the interior triangles of Γ (i.e., the triangles not incident on $\partial\Gamma$), and whether the triangles intersecting and below $\tilde{P}$ constitute a disk. If $\tilde{P}$ satisfies both of these conditions, it is then considered *safe*. We accept $\tilde{P}$ as an improved plane if it is safe and it is more orthogonal to the surface than P. We repeat the inner loop for up to four iterations to obtain the most orthogonal plane.

In the outer loop, we start by trying to place the anchor point at the AICC and improving the orthogonality by invoking the inner loop. Let $\boldsymbol{p}$ denote the anchor point of the current trial plane. If the trial plane is not orthogonal, we then move the anchor point of the trial plane to the mid-point between $\boldsymbol{p}$ and the anchor point of the base cut. This process is repeated for a few (typically up to four) iterations. If none of the trial planes is orthogonal enough, we then accept the base plane.

Identifying Other Patches

After finding stable tips, we then identify the other patches in the remainder of the surface. These other patches include segments, junctions, and potentially some tips that were designated as unstable in the previous step. This step shares some similarities with the previous one. However, there are also some key differences in terms of seed selection, growth of patches, and stopping criteria. In addition, in this step we must address the additional complexity of having potentially too many branches in a junction. We hereafter describe the algorithm and focus on these key issues. Algorithm 3 outlines a high-level pseudocode for this step.

Algorithm 3 Identification of remaining patches.

```
1: for each seed vertex $v$, in ascending order of IRC
2:     while any incident face $\sigma$ of $v$ is not assigned
3:         $S \leftarrow$MCS of $\sigma$ centered at its AICC;
4:         obtain patch $\Gamma$ in $S$;
5:         for $i = 1, 2, \ldots$
6:             $\chi \leftarrow$Euler characteristic of $\Gamma$;
7:             if $\chi = 1$
8:                 identify cut or grow AICC-centered MCS;
9:             else
10:                for each border curve $\gamma$ of $\Gamma$
11:                    identify cut or grow sphere at $\gamma$;
12:                end for
13:                if $\chi < 0$
14:                    move cut inwards toward junction center;
15:                end if
16:            end for
17:    end while
18:    if patch has more than three cuts
19:        attempt to decompose junction;
20:    end if
21:    mark triangles of the patch by patch ID;
22:    associate each cut of patch with a seed face;
23: end for
```

Selecting and Sorting Seeds We search for the patches starting from a series of seed vertices. A junction or segment may be free of convex points, but it cannot be composed of only concave points. Therefore, the seed vertices must include not only those at convex areas but also at saddle points. We select the candidate seed vertices as those whose IRCs are smaller than half of the smallest dimension of the bounding box of the object, and sort them in ascending order of the IRC (cf. line 1 of Algorithm 3). We go through the seed vertices one by one, and skip a vertex if all of its incident faces have been assigned to a patch (a tip, segment, or junction). Checking incident faces (instead of checking only vertices) is necessary, because some triangles may not have been assigned to any patch even after all vertices have been assigned.

Growing Candidate Patch Similar to the previous step, we grow a candidate patch starting from an unassigned face incident on a seed vertex, by locating the triangles that intersect or fall within its AICC-centered MCS and at the same time are not yet assigned to another patch. Although the candidate patch starts as a topological disk, it would inevitably grow into a non-disk at segments and junctions, for which we must consider the potential interactions of the different cuts of a patch. Therefore, we grow a candidate patch differently depending on its topology, as we will describe shortly. Note that it is possible for a candidate patch to change its topology during the growth.

Given a patch, we define a border curve of the patch as a single closed curve composed of border edges of the patch. For each patch, we continue to grow it until each border curve has a corresponding cut. In general, each cut has two adjacent patches. For each border curve, its corresponding cut is therefore either matched with a cut that was identified in an adjacent patch earlier, or is created as a new cut and to be matched by another patch later. After locating a patch, we assign an integer ID to it, mark all the faces in it by the patch ID, and associate each newly created cut with a seed face that is incident on a border edge of its corresponding border curve. This seed face will be used for matching the cut later.

Given a candidate patch that is topologically a disk, we grow it as in Algorithm 1 by using the AICC-centered MCS of the candidate patch. A minor difference is that when filling small holes, we consider not only those within the candidate patch but also those between it and previously identified patches. Since we have identified the tips that are stable by themselves, in this step we accept a disk-patch as a tip only if its border curve γ is matched with an existing cut. We match γ with a cut P in an adjacent patch Γ if every border edge of γ is incident on a triangle assigned to Γ, and P is the cut in Γ whose seed face is incident on γ.

When a candidate patch grows into a non-disk, we then change to grow the patch along each of its border curves separately to identify a cut for each of its border curves. For a patch with two border curves, if one of its border curves γ does not match any existing cut, we attempt to construct a cutting plane from γ, measure its orthogonality of the candidate cut in terms of the minimum dihedral angle, and improve its orthogonality based on the normal covariance matrix. This process is the same as that for stable tips in the previous step. In addition, we emphasize that we place the cut so that all the triangles incident on the border curve are above the cutting plane, so that the remainder of the surface patch would not have any isolated triangles and can be further decomposed into valid sub-structures. If the candidate cut is sufficiently orthogonal, we accept it as the cut for the curve. If an orthogonal plane cannot be obtained, we grow the candidate patch outward at γ. Specifically, we construct a sphere centered at the centroid of γ and containing the triangles incident on γ, and then locate the triangles that fall within or intersect the sphere and are not yet assigned to a different patch, accomplished by a breadth-first search from the triangles incident on γ. We add these triangles into the candidate patch and then re-attempt to match a cut or determine a new cut for the patch. We continue to grow the patch until a cut is identified for the border curve or the number of border curves has changed.

For a candidate patch with more than two border curves, it is desirable for the bounding cuts to be as close to the junction as possible while maintaining the orthogonality of the cuts, so that the junction itself would be more likely to be convex and be closer to a ball-shape. For this reason, we relax the tolerance of orthogonality for junctions. For each border curve γ, we attempt to match a cut, determine a new cut, or grow the patch at γ as above. Starting from each of the identified cuts (either matched or newly created), we try to move it closer to the junction using a bisection procedure. In particular, we try to move the cut toward the centroid of the junction by half of its distance and then optimize its orthogonality. If the plane is not orthogonal enough, we move it back closer to the border curve; otherwise we move the plane closer to the centroid. This process is repeated for a few (typically four) iterations. We identify the cuts one by one and accept the patch as a junction if a cut is identified for each border curve.

Decomposing Multi-Cut Junctions When growing a patch, it is probable for a junction to have more than three border curves. Some of these junctions may be real multi-furcations in the geometry, while other are artifacts. These artifacts are especially likely to occur if the geometry has regions where junctions are close to each other. To resolve this issue, we attempt to decompose any multi-cut junctions into bifurcations. In particular, if a candidate junction with more than three cuts is obtained, we sort its cuts by their maximum radii (computed as the maximum distance from the points on the curve to the centroid) in ascending order, and then try to recompute a patch from the seed face of each of these cuts. If we reach at the same number of cuts starting from different seeds, we accept the multi-cut junction as a real multi-furcation. We accept a recomputed patch if it is a segment or a bifurcation. If a recomputed patch is neither a segment nor a bifurcation but has fewer cuts, we then try to recompute for this smaller patch recursively.

In practice, multi-furcations typically have few cuts. For better efficiency, we set an upper bound on the number of cuts of a junction. We stop the growth of a patch if its number of cuts is larger than the upper bound and resort to recomputing the junction from these cuts as above. In practice, an upper bound 10 works well for a variety of geometries in our test. This upper bound is rarely reached even without the decomposition.

Remark 3. The two steps (steps 2 and 3) described in Section 3.5.3 and 3.5.3 identify an initial decomposition of the object into tips, segments, and junctions. This initial decomposition determines the topology of the medial curve. In step 3 we preserved all the stable tips identified in step 2. For better efficiency, it is possible to merge part of step 3 into step 2. In particular, we can identify the segments and junctions along with the stable tips in step 2, but allow stable tips to take over the triangles assigned to segments and junctions and remove segments and junctions that overlap with tips. Then, in step 3 we need to identify only the remaining patches from faces that are not yet assigned to any patch. However, such a reorganization complicates the logic of the algorithm, so we sacrificed efficiency slightly for clarity in our algorithm.

3.5.4 Fine-Tuning Cuts

The preceding steps identified the local sub-structures mostly independently of each other, except for the care taken in matching the cuts and ensuring topological consistencies. Those steps could not optimize the cuts with respect to each other, because some adjacent patches might still be unknown. Thus, the lengths of adjacent segments may differ drastically. In addition, our algorithm might have pushed some cuts too deep into the junction in order to keep the junctions as compact as possible. It is therefore desirable to have an additional step to fine-tune the cuts. Our algorithm allows for such optional step.

We first adjust location of the cuts at junctions. Those that are adjacent to some stable cut can be improved by moving them closer to the stable cut. Way that the seeds were originally chosen implies that high curvature regions between two consecutive cuts should be rare. So while in the patch, orthogonality of the cut is reasonably monotonic function of its distance to known stable cut. Based on this assumption we use bisection to optimize orthogonality of the junction cut while keeping junction as compact as possible. This time we use more strict tolerance for orthogonality than in 3.5.3. Note that cut might be deleted if it can't be improved or is too close to next cut. Adjacent (but not incident to same junction) cut might also be unstable. If the patch between two is a segment we try to replace them both with cut in the middle of the segment. In all other cases we leave it alone.

Algorithm 4 Optimize unstable junction cut $U(p_u, n_u)$ based on adjacent stable cut $S(p_s, n_s)$.

```
1:  for each junction cut $U(p_u, n_u)$
2:      $S(p_s, n_s)$-adjacent cut
3:      if $S$ is not stable
4:          if patch between $S$ and $U$ is segment
5:              $n_o = n_s$ or $n_o = n_u$whichever is more orthogonal
6:              adjust $O \leftarrow (p_o = \frac{p_u+p_s}{2}, n_o)$
7:              if $O$ is not orthogonal enough
8:                  continue
9:              else
10:                 replace $U$ with $O$
11:             end if
12:         else
13:             continue
14:         end if
15:     else
16:         $M \leftarrow (p_m = \frac{p_u+p_s}{2}, n_m = n_s)$;
17:         $L = U$; $R = S$;
18:         $O = S$;
19:         for $i = 1 : 4$
20:             adjust $M$
21:             if $M$ is orthogonal enough
22:                 move cut toward junction center
23:                 $M \leftarrow (\frac{p_m+p_L}{2}, n_m)$;
24:                 $R = M$;
25:                 $O = M$;
26:             else
27:                 see if cut can be improved by moving closer to stable one
28:                 $M \leftarrow (\frac{p_m+p_R}{2}, n_R)$;
29:                 $L = M$;
30:             end if
31:         end for
32:         if $O = S$
33:             remove unstable cut $U$
34:         else
35:             replace $U$ with $O$
36:         end if
37:     end if
38: end for
```

After optimizing the cuts at junctions, we then perform an iterative Laplacian smoothing to redistribute the cuts between segments and optimize their orthogonality, while preserving the cuts at the tips and junctions.

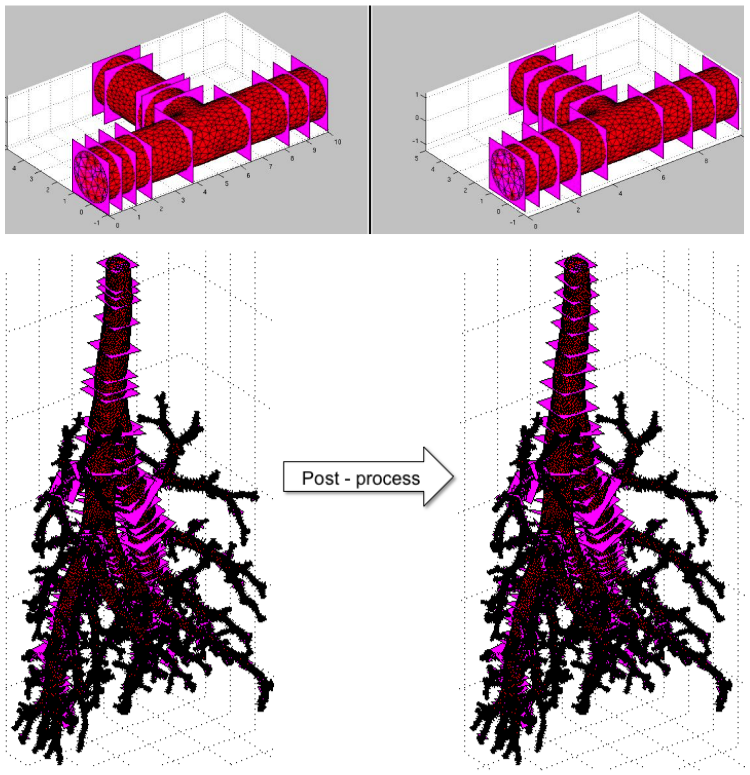


Figure 3.3: Post-processing of local orthogonal decomposition, before and after.

3.5.5 Reconstructing Medial Curve

After decomposing an object into tips, segments, and junctions, we reconstruct its medial curves based on the connectivity of these patches. Specifically, we create a vertex in the medial curve for each cut center and for the center of each segment and junction, and assign consecutive integer IDs to the vertices. After creating the vertices, we create an edge in the medial curve to connect each cut center with the center of each of its incident segments or junctions, and identify each edge by a pair of vertex IDs. We choose the cut center to be the centroid (i.e., the center of mass) of the cross section of the cutting plane with the surface. The centroid is defined as $\int_C \boldsymbol{x} dxdy / \int_C dxd$, which we compute by decomposing the cross section into triangles apexed at a point in the cross section and then computing the signed-area-weighted average of the centroids of these triangles. For each segment, we compute its center as the average of the cut centers. For each junction, we compute its center as the volume centroid of the junction, and compute it as a signed-volume-weighted average of the centroids of the pyramids that are apexed at the same point and based at the cuts or based at the parts of the faces that fall within the patch. The centroid of a pyramid is simply three fourths of the centroid of the base plus one forth of the apex.

3.5.6 Implementation Details

Our algorithm requires only the vertex coordinates and element connectivity of the triangles as input. Internally, our algorithm requires some data structures to store the mesh, the cuts, the patches, and their correlations with each other. We hereafter describe these data structures.

For the mesh, we need a data structure that can facilitate neighbor-to-neighbor traversal. We use the well-known *half-edge data structure* (also known as doubly-connected edge list) [13, §2.2], and in particular an array-based version described in [1]. At a high-level, each edge in general has two opposite, directed half-edges in its two incident triangles. Our data structure identifies each half-edge in a face by a face ID and a local edge ID within the face, encoded in a single integer, and stores the mapping from each half-edge to its opposite half-edge in a 2-D array of size $m \times 3$, where m is the number of triangles. Together with the element connectivity and a mapping from each vertex to one of its incident half-edges, we have a compact and complete half-edge data structure. This data structure can be constructed efficiently in linear time. We group these arrays along with other geometric quantities (including normals, curvatures, and surface areas) into an object (a.k.a. a structure).

A key output of our algorithm is the medial curve. Similar to the input mesh, we store the medial curve simply by a list of vertex coordinates and the element connectivity of the edges. For later processing, a data structure is also needed for neighbor-to-neighbor traversal of the curve. We define a vertex-based data structure for storing the medial curve, analogous to the array-based half-edge data structure. Specifically, we refer to a vertex within each edge as a "*half-vertex*" and identify it by the edge ID and a local vertex ID within the edge, encoded in a single integer. Since the medial curve in general is not a manifold, a vertex can have more than two incident edges. We construct a cyclic mapping for the half-vertices corresponding to the same vertex (instead of a mappings between opposite half-vertices in a manifold mesh). This mapping is stored in a 2-D $n \times 2$ array, where n is the number of edges in the medial curve. We refer to this as an *extended half-vertex data structure* for non-manifold curves.

In our algorithm, we assign an integer ID for each cut and patch, and store the data for them in arrays grouped with an aggregate object. We assign patch IDs in the order of being identified. Note that each cut is incident on two patches. We consider the patch that is identified first as the the *owner patch* of the cut. We assign IDs to the cuts consecutively in ascending order of their owner patches. For each cut, we store its centroid, its normal outward to its owner patch, as well as the IDs of its incident patches. For each patch, we store the cuts owned by it consecutively and also store a list of the IDs of the matched patches. These data allow the complete reconstruction of the medial curve as well as the computation of the statistics of the medial curve for applications. For best efficiency, our algorithm preallocates buffer spaces used by the algorithm and group them into a single object.

We have implemented our algorithm in MATLAB, which is feasible because our data structures are all array-based, without using any pointers. The use of the high-level programming language MATLAB and its interactive visualization environment eases implementing and debugging of the complex logic of our algorithm. We then converted our MATLAB code into efficient and portable ANSI C code using the Agility MCS software (`www.agilityds.com`; acquired by Mathworks in early 2009).

3.5.7 Experimental Results

In this section, we report some experimental results with our method for complex, image-derived biomedical geometries and compare our method against some others. The medial-curve algorithms are largely independent of the image processing method. In our setting, we used a fuzzy-connected threshold approach for segmentation [74] and a variant of the well-known marching cubes algorithm for the isosurface extraction [57]. To assess the robustness and noise resistance, we will report results for raw, unsmoothed surface meshes from isosurfacing, smoothed meshes with a volume-conserving smoothing algorithm [43], as well as those after adaptive remeshing using the method in [44].

Results for Smoothed Geometries

We first present results for three complex biomedical geometries. All these geometries are smoothed, as they typical are in real applications. Table 3.1 shows the statistics of the datasets and summarizes our results. Our first test geometry is a mouse coronary artery network, derived from the CT dataset courtesy of Dr. Ghassan Kassab [52]. Figure 3.4 shows the geometry, superimposed on the medial curve computed by our algorithm; see row "Coronary" in Table 3.1 for the statistics of the mesh and the medical curve. Since this geometry is much simpler than the other two (with fewer than 100 tips and junctions), we could visually inspect the accuracy of the result. Our inspection indicated that the structure of the medial curve was 100% accurate (i.e., fully agreed with the intuition of a trained biologist). This accuracy is remarkable, as evident from the comparative study that we will show in Section 3.5.8. Our algorithm took 0.97 seconds for this case, including 0.36 seconds to identify the tips, on a Linux computer with a 3GHz Intel Duo Core processor and 2GB memory.

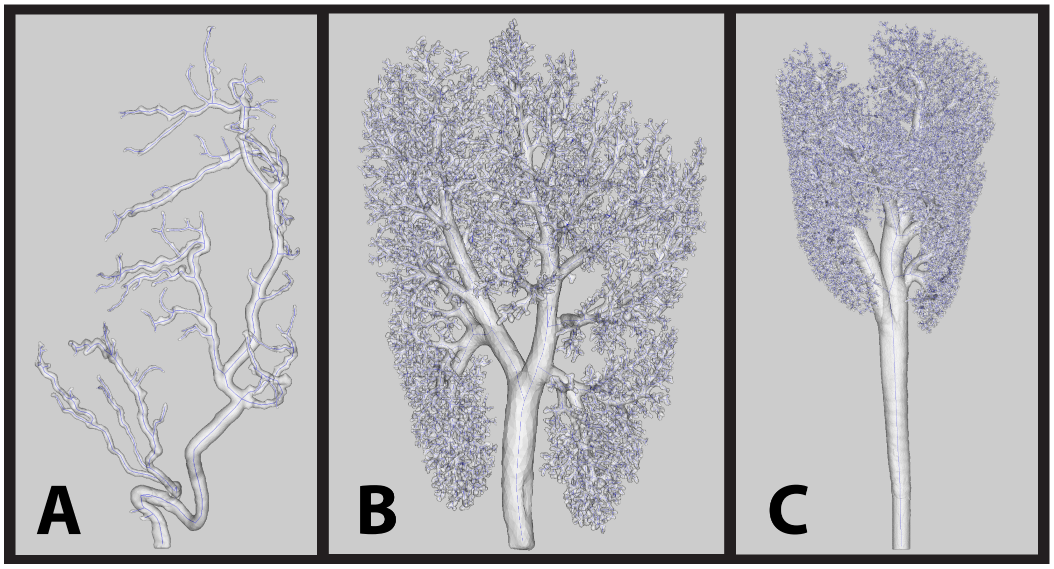


Figure 3.4: Medial curve of three biological surfaces. **A**) A mouse coronary artery from CT data. **B**) A monkey pulmonary airway from MRI data. **C**) A high-resolution rat pulmonary airway from CT data.

Table 3.1: Results for three test geometries in Figure 3.4.

	Input Mesh		**Elements**		**Run Times (sec)**		
Case	vertices	triangles	tips	junctions	tips	others	total
Coronary	57,008	114,012	96	85	0.36	0.60	0.97
Monkey Lung	895,915	1,791,826	6,214	7,121	13.3	14.6	27.9
Rat Lung	3,975,949	7,951,894	17,416	12,525	55.8	312.7	368.5

Our second test geometry is a monkey pulmonary airway tree (see Figure 3.4) and row “Monkey Lung” in Table 3.1), derived from the MRI dataset courtesy of Dr. Kevin Minard. Algorithm processed the dataset in about 27.89 seconds, and identified 6,214 tips and 7,121 junctions. The medial curve is homotopic to the object, as guaranteed by our algorithm. Visual inspection at sampled locations indicates that the curve was smooth, well centered, and structurally correct almost everywhere.

Our third test geometry is a high-resolution rat pulmonary airway tree (see Figure 3.4**C** and row "Rat Lung" in Table 3.1), derived from the CT dataset courtesy of Drs. Timothy Cox, Richard Jacobs and James Carson. It took about six minutes for the algorithm to identify 17,416 tips and 12,525 junctions.

Assessment of Sensitivity to Noise

To further assess the noise resistance of our method, we apply our algorithm to a series of meshes for the mouse coronary artery, including an unsmoothed triangulation of the isosurface from marching cubes, a smoothed triangulation, as well as three meshes that were registered to the gradient-limited feature size as described in [44] with three different levels of refinement and derefinement. Table 3.2 summarizes the numbers of vertices, triangles, identified tips, and identified junctions of these cases.

Table 3.2: Results for different meshes for the mouse coronary.

	Input Mesh		**Elements**	
Case	vertices	triangles	tips	junctions
Unsmoothed	57,008	114,012	96	83
Smoothed	57,008	114,012	96	85
Adaptive 1	76,143	152,282	92	81
Adaptive 2	71,130	142,256	91	83
Adaptive 3	64,685	129,366	91	79

The unsmoothed mesh is a challenging test, as the noise in this dataset is high-frequency and has a magnitude of up to one voxel. Figure 3.5 shows the tips, junctions and medial curve for this dataset. Our method identified the same number of tips with and without smoothing, although it produced different number of junctions (because some high-degree junctions in the unsmoothed mesh were broken into low-degree junctions). This result shows that our method is relatively insensitive to noise. For the adaptively refined meshes, our method slightly fewer tips and junctions (by about 7%), probably because the additional smoothing involved in the mesh adaptation process eliminated some short branches and their corresponding tips.

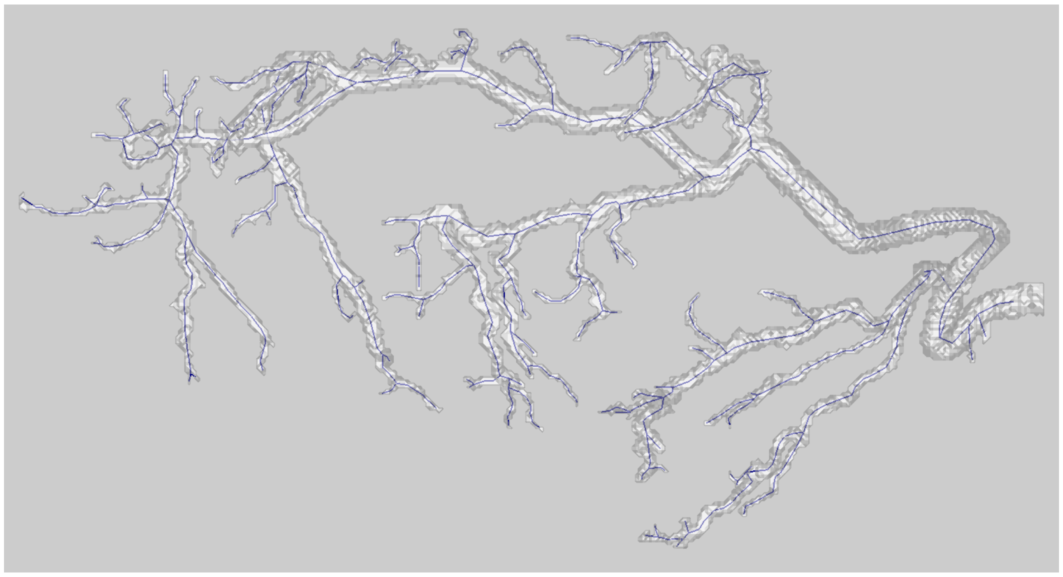


Figure 3.5: Noisy dataset of mouse coronary artery and the computed medial curve.

3.5.8 Comparison with Other Methods

We compare our method against the publicly available implementations of two other surface-based approaches [4, 14]. The method of Dey and Sun in [14] is based on an analysis of a medial geodesic function on the Voronoi-based medial surface [14], and the method of Au et al. in [4] is based on mesh contraction. Table 3.3 shows the numbers of identified tips and junctions as well as the overall run times for our method and those two methods for the smoothed and unsmoothed coronary geometries shown in Figures 3.4 **A** and 3.5. Figure 3.6 shows close-up views of the results from these methods. Note that the result of our method for the smoothed geometry has verified manually, so we use it as the reference solution. The Dey-and-Sun's approach in [14] produced nearly twice as many tips and junctions as our method. From Figures 3.6 **A** and 3.6 **C**, it is obvious that there were many short, spurious branches. Besides spurious branches, Dey-and-Sun's approach also missed several actual branches as can be seen in Figure 3.6 **C**. In addition, the method in [14] produced far more tips and junctions for the unsmoothed mesh than for the smoothed mesh, and the medial curve tends to be highly jagged, showing its sensitivity to noise both topologically and geometrically. In terms of computational cost, the runtime of Dey-and-Sun's method was more than three orders of magnitude slower than ours for this mesh. This ratio grows substantially as the size of the mesh increases, and their code never terminated for the meshes with one million or more faces in our test.

Table 3.3: Comparison of our method with alternative surface-based methods for the mouse coronary artery. For Au et al., numbers of tips and junctions were counted visually, as their code did not allow saving the result. All times are in seconds.

Case	**Ours**			**Dey and Sun** [14]			**Au et al.** [4]		
	tips	junctions	time	tips	junctions	time	tips	junctions	time
Smooth	96	85	0.97	146	142	1294.3	84	79	35.3
Noisy	96	83	1.10	163	161	1439.3	87	81	32.6

In contrast to [14], the mesh contraction method of Au et al. in [4] appears to be relatively insensitive to noise. However, as shown in Figure 3.6**E**, their method tends to miss tips and branches. Furthermore, the medial curve from their method was not well centered, with some vertices outside of the object (see Figure 3.6**F**). In terms of running time, their method is more than 30 times slower than ours. From these results, we conclude that for biomedical geometries, which are the interests of our applications, our algorithm delivers a better combination of structural accuracy, smoothness, centeredness, robustness, noise resistance, and efficiency, compared to the other state-of-the-art methods.

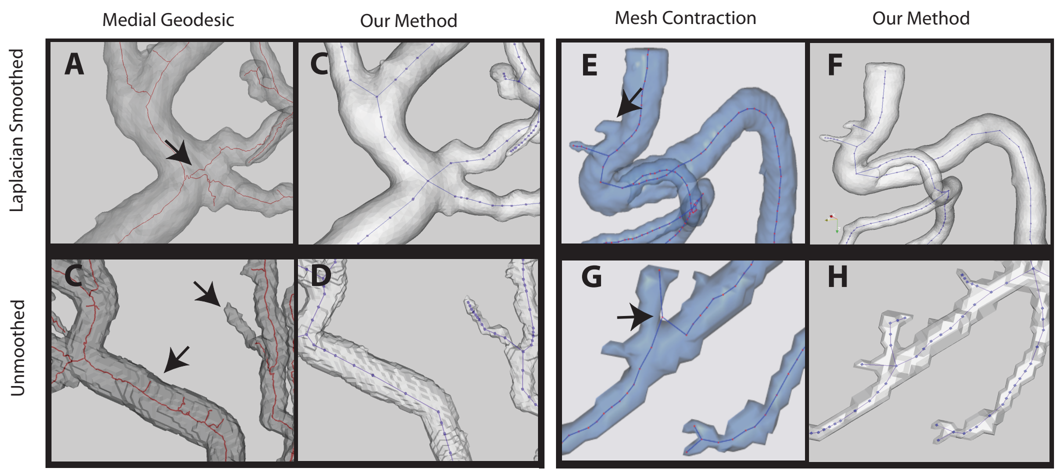


Figure 3.6: Comparison of our results against two other methods for the smoothed and unsmoothed coronary geometry. Artifacts in the results are indicated by black arrows. Panels A-D contrast the medial-geodesic method with our method; Panels E-H contrast the mesh contraction method

Chapter 4

Boundary Layer Meshes

4.1 Previous work

Traditionally, mesh generation is broadly classified as *structured* or *unstructured* (see [71] for a comprehensive survey). Structured meshes are known to produce higher accuracy results but are too time consuming (if even possible) to generate for complex biomedical geometries. On the other hand unstructured meshes can be generated fast and without manual interaction which is desirable for clinical applications of CFD. In recent years, the generation of *semi-structured* boundary-layer meshes has attracted significant interest (see e.g. [3, 5, 19, 22, 26, 27, 37, 40, 62, 63, 76]), as these meshes are effective for capturing large gradients and resolving viscous flows near the boundary and can be generated much faster then structured meshes. We review some of the literature on the generation of three-dimensional boundary-layer meshes, focusing on those that have played a notable historical role or represent the current state of the art.

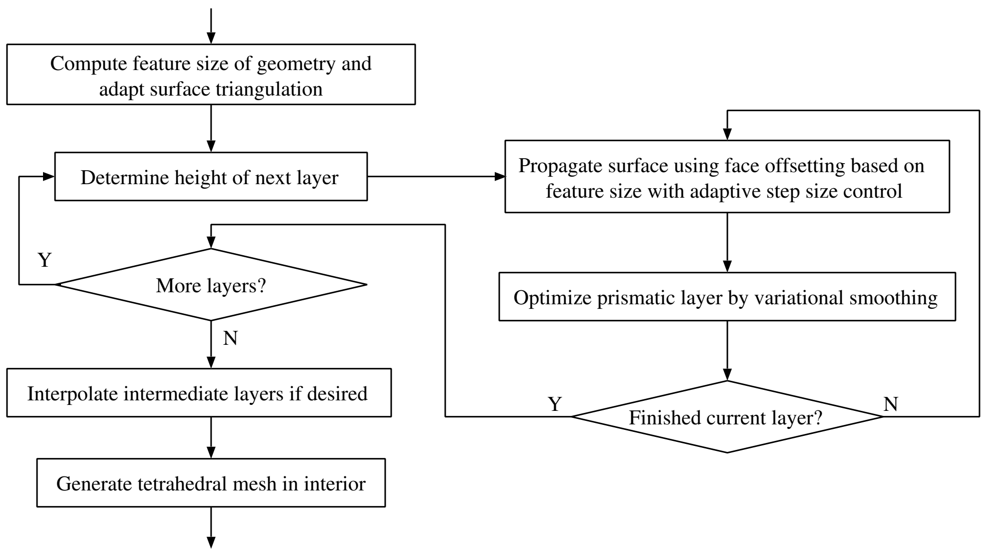
Most of existing methods, including those in the literature or commercial software, generate boundary layers by marching the vertices of the surface and smooth the vertex positions to improve mesh quality. For example, the *advancing-layers method* in [62] extrudes the surface mesh by moving its vertices nearly orthogonally to the surface to create thin layers of elements. For a triangular surface mesh, prisms are typically generated by connecting the vertices of adjacent surface layers, such as in [37, 40, 66]. The remainder of the domain away from the boundary is typically meshed with tetrahedra, using advancing front or Delaunay methods. Sometimes, the prisms are subsequently subdivided into tetrahedra for compatibility with some simulation codes [62, 63].

Some key challenges in advancing-layers methods include enforcing the validity of the prismatic layers (e.g. to avoid inverted elements and self-intersections) and obtaining high quality elements in the mesh. Some earlier methods could only generate very thin layers and were not robust for complex geometries or large displacements. Garimella and Shephard [19] later proposed a generalization of the advancing-layers method to enhance robustness by adopting multiple marching directions at *convex* sharp edges and applying smoothing, shrinking, and pruning at *concave* sharp edges. The idea of multiple marching directions was also used and advocated by other authors (see e.g.,[5, 26, 27, 48]). Athanasiadis and Deconinck [3] introduced more sophisticated methods to address these sharp features, but their method required many different types of elements (including degenerate prisms and degenerate hexahedra), which introduce additional complexities to the algorithms and may not be suitable for many simulation codes. These existing techniques are mostly designed for CAD models with piecewise smooth surfaces, and they invoke special treatments at well-defined sharp features (i.e., ridges and corners). They do not address the "near singularities" caused by large curvatures or coarse resolution of the surface mesh, both of which are common for biological geometries.

A key difficulty associated with meshing concave sharp areas is the development of "swallow-tails" [65]. Recently, Wang *et al.* proposed generating boundary-layer meshes in three dimensions using a fast marching method with unit speed [75, 76]. However, their method does not allow for the adaptation of the marching based on the geometry or physics, and difficulties still remain in reconstructing the connectivities [75]. We propose hybrid technique for generating hybrid meshes with high-quality boundary layers. It can very efficiently generate hybrid tetrahedral-prismatic mesh with high-quality boundary layer and isometric tetrahedral core. In case when running time is less important constraint then properties of terahedral core, high quality boundary layer can be added to any teraherdralization of the geometry while preserving structural characteristics of the core. Our approach combines the efficiency and flexibility of advancing layers with the "entropy condition" of the level set methods. This combination is made possible by the face-offsetting method (*FOM*) for surface propagation [29], which relies on a mathematical formulation called the *generalized Huygens' principle* to impose the entropy condition to explicit dynamic surfaces. However, unlike level set methods, FOM directly propagates a surface mesh, and unlike fast marching methods, FOM allows adapting the heights of the layers through a "speed function" to generate high-quality meshes and avoid self-intersections. We apply a modification of the face-offsetting method to create hybrid meshes in challenging biomedical geometries by computing the speed function based on a feature size of the geometry.

4.2 Feature Size Guided Face Offsetting

First we will describe out approach to fast generation of unstructured tetrahedral-prismatic mesh based on the isosurface. As an input method accepts a suitable, geometrically and topologically correct isosurface that has been appropriately manipulated to accommodate the definition of proper boundary conditions. Figure~4.2 shows the overall control flow of our method, given that isosurface. Like some existing methods, we generate layers with prisms near the boundary and fill the interior with tetrahedra. However our method is novel in the following aspects: 1) We use the feature size defined in [44] to control the height of the prisms to make the resulting mesh scale invariant and prevent global self-intersections. 2) We use the face-offsetting method [29] to propagate the surface mesh to generate the prismatic layers, which provides reliable directions of vertex motions and prevents local self-intersections. 3) We develop a variational mesh optimization for prismatic layers, which improves not only the shapes of the triangles but also the orthogonality of the prisms. 4) We introduce simple and effective measures to guarantee the validity of the resulting prisms.


Compute feature size of geometry and adapt surface triangulation
Determine height of next layer
Propagate surface using face offsetting based on feature size with adaptive step size control
Y
More layers?
N
Optimize prismatic layer by variational smoothing
Interpolate intermediate layers if desired
Y
Finished current layer?
N
Generate tetrahedral mesh in interior

First we estimate feature size and use it to adapt the surface triangulation. This method was developed by our collaborators in The Pacific Northwest National Laboratory [44]. Then we use feature size to provide a "speed function" of face offsetting for the generation of prismatic boundary layers with the face-offsetting method developed by Professor Jiao [29]. This novel and natural integration will allow the generation of high-quality, adaptive boundary-layer meshes when fortified by the optimization techniques that we will introduce in later sections.

4.2.1 Gradient Limited Face Offsetting

One of the key concepts in our method for generating boundary-layer meshes is the *gradient-limited feature size* (or *GLFS*), introduced in [44]. The GLFS is essentially a rough estimation of the *local feature size* of a surface (where the local feature size is the shortest distance form every point on the surface to its medial axis), computed without constructing the medial axis. It is useful to control the grading of the tetrahedral and surface meshes, as well as the thickness of the boundary layers.

At the high-level, the GLFS is defined and computed as follows. Let S be an oriented closed triangulated surface, derived from imaging data. We allow S to consist of one large enclosing closed surface S_0 and possibly any number of closed surface holes $S_1, \dots S_n$ inside S_0. Any of the surfaces $S_0, S_1, \dots S_n$ may have handles (i.e., have nonzero genus). For any vertex $X \in S$, we define the *raw feature size* $F[X]$ as the length of the line segment formed by first shooting a ray from X in the direction of $\hat{n}_{\text{in}}[X]$, the inward normal at X, and then truncating the ray at its first intersection with S, i.e.,

$$F[X] \equiv \min\{\lambda > 0 \mid X + \lambda \hat{n_{in}}[X] \in S\}. \qquad (4.2.1)$$

Some user-specified lower and upper bounds may also be imposed on $F[\boldsymbol{x}]$. However, this raw feature size may be sensitive to abrupt changes in the geometry, so we perform a gradient-limiting procedure as follows: first initialize $f[X]$ to $F[X]$. Given a bound on the surface gradient of $f[X]$, we iteratively relax $f[X_i]$ to satisfy the gradient limit. The resulting feature size f is then the gradient-limited feature size (or GLFS).

We use GLFS to adapt the surface mesh and as the speed function for propagating the surface layers using face offsetting. We modify S based on GLFS by performing the operations of smoothing, refinement and de-refinement while limiting perturbations to a small fraction of a voxel. Resulting high-quality surface mesh S' has local resolution proportional to local scale (Figure~4.1).

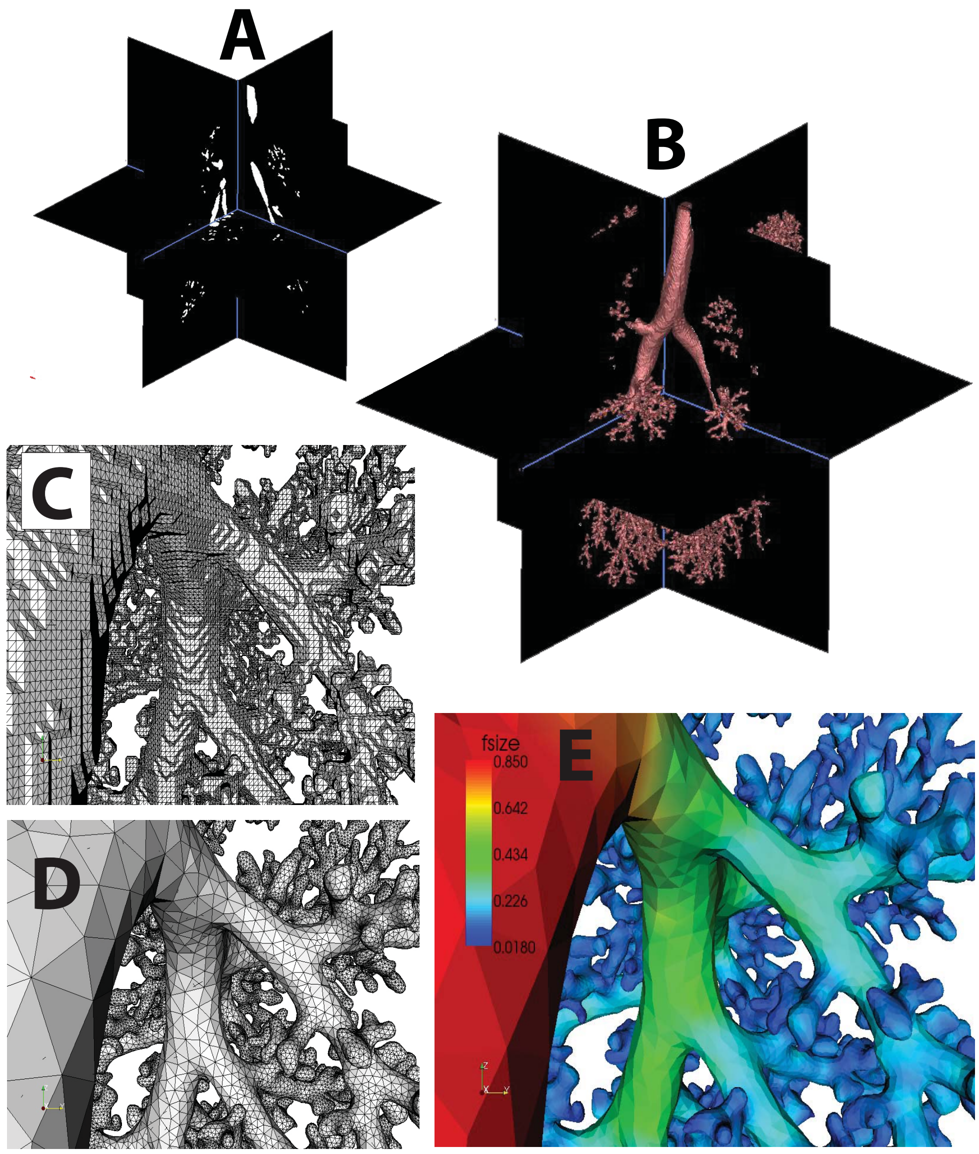


Figure 4.1: Development of a scale-invariant surface mesh from MRI data: A) binary segmentation of the imaging data of a monkey lung cast; B) isosurface generation; C) detail of the isosurface; D) surface triangulation registered to the feature size; E) feature size field.

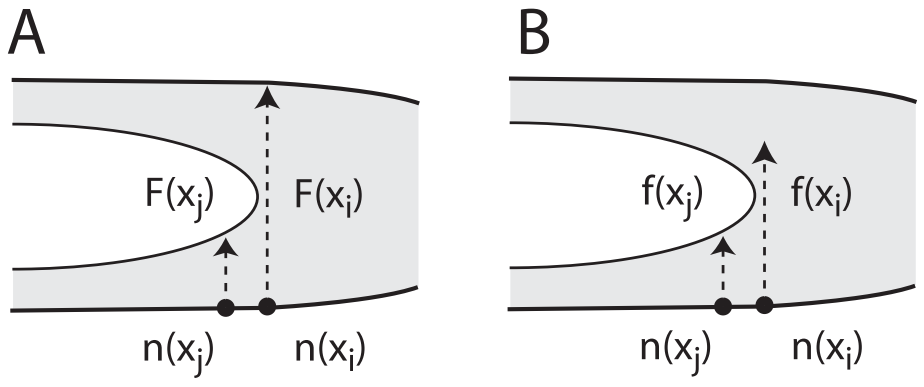


Figure 4.2: Without gradient limiting $F[x_i]$ can discontinuously jump with a small perturbation in the surface S due to borderline clipping of local diameter ray. With gradient limiting, this jump is generally limited to $G\|x_i - x_j\|$, where x_j is a neighbor whose local diameter ray is robustly clipped.

4.2.2 Advancing Layers by Face Offsetting

Our method advances a surface layer by solving the Lagrangian evolution equation,

$$\frac{\partial \boldsymbol{x}}{\partial t} = f(\boldsymbol{x}, t)\hat{\boldsymbol{n}}, \tag{4.2.2}$$

where t denotes time, $\hat{\boldsymbol{n}}$ denotes unit surface normal, and $f(\boldsymbol{x}, t)$ denotes the speed function as we will explain shortly. In principle, generating a layer of prisms reduces to marching the vertices in time by discretizing (4.2.2).

The method provides two options in generating a multi-layered prismatic mesh. The first option is to generate one layer of prisms per time step by connecting the vertices before and after propagation, and generate a layered mesh by taking multiple time steps. Another option is to march the surface by the desired total height of all the layers and then interpolate the intermediate layers. We refer to the two options as *propagated layers* and *interpolated layers*, respectively. Both options can generate quality, graded prismatic meshes, as we will demonstrate in later experimental studies. We will focus our discussions on the first option as it can be easily extended to support the second one.

In (4.2.2), the speed function serves two purposes: 1) it controls the density of the prisms along normal direction, and 2) it can slow down the propagation of the surface at small features to avoid collisions. We note that the gradient-limited feature size (GLFS) seems to be ideal to fulfill these purposes. First, the surface mesh is adapted commensurate with GLFS, so adapting the height of the boundary layers gives some control of the aspect ratios of the prisms. Second, GLFS is approximately twice of the surface's local feature size, so it can control the distance of the surface to the medial axis to avoid collisions. Therefore, we choose GLFS as the speed function, and "time" is then a measure of the height of the boundary layer versus GLFS. Note that unlike surface mesh adaptation, we need not limit the speed function based on the maximum curvature, and we may use different L_{min} and L_{max} parameters when computing GLFS.

Given the speed function, a key question in integrating Eq.~(4.2.2) is the evaluation of surface normals, which is challenging for coarse meshes of noisy surfaces. After obtaining the GLFS, we could simply move each vertex along the normal direction by the desired distance, but we observed that a simple weighted-averaging of face normals can lead to unreliable normal estimates for imaging-derived geometries in our applications. In addition, it is well known that the Lagrangian equation can encounter difficulties (such as "swallowtails" or severe lagging) at strongly concave regions and large curvatures. We address these issues using the *face offsetting method* in [29], which is based on a geometric construction called *the generalized Huygens' principle* and numerical techniques of least-squares approximation and eigenvalue analysis.

The basic idea of the face-offsetting method (*FOM*) is to integrate the Lagrangian evolution equation in time by advancing the faces (instead of vertices) along normal directions and then reconstructing the vertices from the displaced faces. We first advance each triangle by moving its ith vertex normal to the face for a distance proportional to the GLFS at the vertices. After displacing the faces, we reconstruct the vertices based on the *generalized Huygens' principle* [29]. This principle states that every point of an interface has a field of influence (such as the trajectory of a particle in advection or a spherical neighborhood of a heat source in burning), and the new interface is the boundary of the union of the influences of all the points on the interface. Unlike the classical Huygens' principle, this principle is applicable to both advective and wavefrontal motions, which have different behaviors at expansions as illustrated in Figure~4.3. In terms of mesh generation, a wavefrontal motion may be more desirable than advective motion, as it would smooth out convex sharp edges and in turn is likely to be more robust than advective motion, which would preserve convex sharp edges. Computationally, solving advective motion is a prerequisite for solving wavefrontal motion in FOM, as we will explain shortly. If the interface remains smooth, this principle results in Eq.~(4.2.2) and also the well-known level set equation [65]. Unlike those PDEs, however, this construction is well defined even at singularities.

We first describe the numerical solution of vertex reconstruction for advective motion. Based on the generalized Huygens' principle, each vertex should move to the intersection of the offsets of its incident faces. In three dimensions, the intersection may be ill-defined, and we therefore formulate the intersection at each vertex

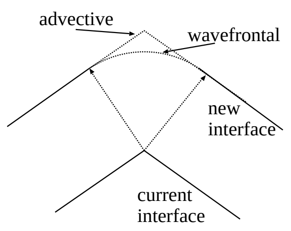


Figure 4.3: Advective vs. wavefrontal propagation at expansion.

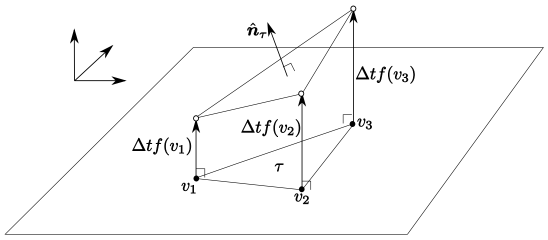


Figure 4.4: Illustration of offsetting each face.

as an ill-conditioned least squares approximation and solve it using an eigenvalue analysis as follows. At a vertex v, consider the computation of the displacement vector $\boldsymbol{d}$ from the vertex to its new position. Let $\hat{\boldsymbol{n}}_\tau$ denote the unit normal to the offset face incident on v after displacing its vertices by $\Delta t f_v$ (see Figure~4.4), where f_v denotes the GLFS at v. For each face incident on v, we obtain an equation $\hat{\boldsymbol{n}}_\tau^T \boldsymbol{d} \approx \Delta t f_v$. Suppose vertex v has m incident faces. We then obtain an $m \times 3$ linear system

$$\boldsymbol{N}\boldsymbol{d} \approx \boldsymbol{f}. \tag{4.2.3}$$

Let ω_τ denote the area of face τ, $\boldsymbol{W} = \text{diag}(\omega_1, \dots, \omega_m)$, and $\sqrt{\boldsymbol{W}} = \text{diag}(\sqrt{\omega_1}, \dots, \sqrt{\omega_m})$. We pose (4.2.3) as a weighted least squares problem,

$$\min_{\boldsymbol{d}} \|\boldsymbol{N}\boldsymbol{d} - \boldsymbol{f}\|_{\boldsymbol{W}} \equiv \min_{\boldsymbol{d}} \|\sqrt{\boldsymbol{W}}\boldsymbol{N}\boldsymbol{d} - \sqrt{\boldsymbol{W}}\boldsymbol{f}\|_2, \tag{4.2.4}$$

which reduces to a linear system

$$\boldsymbol{A}\boldsymbol{d} = \boldsymbol{b}, \text{ where } \boldsymbol{A} = \boldsymbol{N}^T\boldsymbol{W}\boldsymbol{N} \text{ and } \boldsymbol{b} = \boldsymbol{N}^T\boldsymbol{W}\boldsymbol{f}. \tag{4.2.5}$$

Because $\boldsymbol{A}$ is symmetric positive semi-definite, it has an eigenvalue decomposition $\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T$, where $\boldsymbol{\Lambda} = \text{diag}(\lambda_1, \lambda_2, \lambda_3)$ is composed of the eigenvalues of $\boldsymbol{A}$, with $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0$. The ith column vectors of $\boldsymbol{V}$, denoted by $\hat{\boldsymbol{e}}_i$, is the eigenvector corresponding to λ_i. We refer to the space spanned by the eigenvectors corresponding to very "small" eigenvalues as the *null space* (or the *numerical null space*). For numerical stability, we filter out the components of the solution to (4.2.5) in the null space and then obtain the normal displacement for vertex v as

$$\boldsymbol{u}_v = \boldsymbol{d} \approx \sum_{i=1}^{k} \hat{\boldsymbol{e}}_i \hat{\boldsymbol{e}}_i^T \boldsymbol{b}/\lambda_i. \tag{4.2.6}$$

In our setting, the surface meshes are typically fairly coarse and do not have sharp features except for the boundary curves of inlet and outlet walls. Therefore, we use a simple procedure to consider an eigenvalue λ_i to be "small" if $\lambda_i \leq \epsilon\lambda_1$ for some ϵ (such as $\epsilon = 0.003$ as suggested in [29]), so the summation in (4.2.6) becomes $\boldsymbol{d} \approx \sum_{\{i|\lambda_i \geq \epsilon\lambda_1\}} \hat{\boldsymbol{e}}_i \hat{\boldsymbol{e}}_i^T \boldsymbol{b}/\lambda_i$.

For generating boundary-layer meshes, it is often desirable to use wavefrontal motion, which produces smoother surfaces at convex regions and in turn can lead to better robustness. We solve the wavefrontal motion using a predictor-corrector procedure: first, we "predict" a direction of motion $\hat{\boldsymbol{d}} = \boldsymbol{d}/\|\boldsymbol{d}\|$ for vertex v, where $\boldsymbol{d}$ is computed using (4.2.6). Second, we "correct" the displacement along $\hat{\boldsymbol{d}}$ as follows. For each face σ incident on v, let $\boldsymbol{s}_\sigma$ denote the vector from v to the centroid of the face after applying the displacements. The displacement of v within face σ is then determined based on whether the face is contracting or expanding locally at v, i.e.,

$$\boldsymbol{d}_\sigma = \begin{cases} \boldsymbol{d} & \text{if } \boldsymbol{s}_\sigma^T \boldsymbol{d} > 0 \text{ (i.e., contracting)} \\ \Delta t |f_v| \hat{\boldsymbol{d}} & \text{otherwise (i.e., expanding).} \end{cases} \tag{4.2.7}$$

As in (4.2.4), let ω_σ denote the area of σ. The weighted least-squares approximation of the displacement along $\hat{\boldsymbol{d}}$ results in a simple weighted averaging

$$\boldsymbol{u}_v = \left(\sum_{\{\sigma|v\in\sigma\}} \omega_\sigma \boldsymbol{d}_\sigma \right) \Bigg/ \left(\sum_{\{\sigma|v\in\sigma\}} \omega_\sigma \right). \tag{4.2.8}$$

We refer readers to [29] for more details. The face-offsetting method provides us an effective method to propagate the surfaces and generate prismatic mesh. However, the resulting prisms do not necessarily have good quality and may even cause inverted elements. We address these quality and robustness issues in the next two sections.

4.3 Variational Optimization of Prismatic Layers

Accuracy and efficiency of numerical simulations depends on shapes of the mesh elements. In this section we focus on prisms comprising the boundary layer. Shapes of the base triangles and the orthogonality of the side edges are particularly important. The former can affect the aspect ratios of the prisms in the boundary layers and of their adjacent tetrahedra in the interior, while the latter can affect the skewness of the prisms. Orthogonality is important for producing smoother boundary layer meshes and for capturing the boundary layer effects in simulations more accurately. Figure~4.5 shows some examples of poorly-shaped prisms due to poorly-shaped triangles or lack of orthogonality.In our framework, the displacements of the vertices are decomposed into normal and tangential motions, where the tangential motion controls mesh quality.

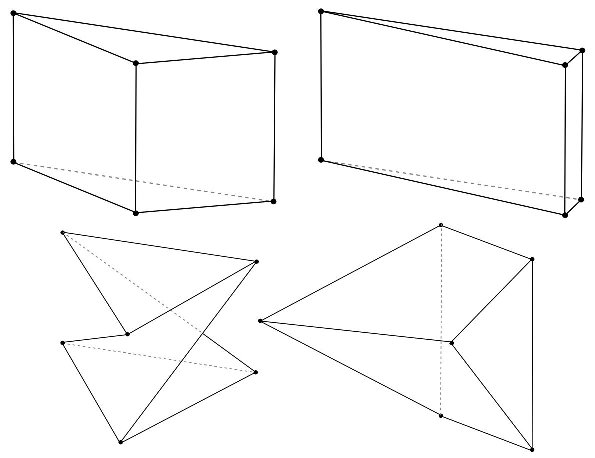


Figure 4.5: Examples of poor-shaped prisms due to poor triangles with too large or too small angles (left two) and lack of orthogonality due to twisting or shifting (right two).

For flexibility, we optimize the prisms using a variational approach by minimizing a weighted sum of two potential energy functions to control triangle shapes and side-edge orthogonality, denoted by E_θ and $E_\perp$, respectively. We compute these energies element by element and refer to the energy of each element as *elemental energy*. The total energy is then the sum of the elemental energies. For efficient minimization, we need to evaluate the gradient and Hessian of the energies with respect to the vertex positions of the prism. We hereafter describe the computation of the elemental energies and their derivatives and then present our overall smoothing algorithm.

For the convenience of presentation, let us define a special naming convention of the prism and some additional notation. As shown in Figure~4.6(a), let the vertices of the prism be $\boldsymbol{x}_j$ for $j = 1, \dots, 6$. We refer to the triangle $\boldsymbol{x}_1\boldsymbol{x}_2\boldsymbol{x}_3$ and $\boldsymbol{x}_4\boldsymbol{x}_5\boldsymbol{x}_6$ as bottom and top triangles, respectively. Let the opposite edge of $\boldsymbol{x}_j$ in the top or bottom triangle be $\boldsymbol{t}_j$, and let the $\boldsymbol{s}_i = \boldsymbol{x}_{i+3} - \boldsymbol{x}_i$ be the ith side edge for $i = 1, 2, 3$. In the following, i ranges between 1 and 3 and j ranges between 1 and 6, unless otherwise stated. Let us define the shorthands

$$i+ = \begin{cases} i+1 & i < 3 \\ 1 & i = 3 \end{cases} \text{ and } i- = \begin{cases} i-1 & i > 1 \\ 3 & i = 1 \end{cases} \tag{4.3.1}$$

and then $\boldsymbol{t}_i = \boldsymbol{x}_{i-} - \boldsymbol{x}_{i+}$ and $\boldsymbol{t}_{i+3} = \boldsymbol{x}_{(i-)+3} - \boldsymbol{x}_{(i+)+3}$. Let $\boldsymbol{n}_t = \boldsymbol{t}_5 \times \boldsymbol{t}_4$ and $\boldsymbol{n}_b = \boldsymbol{t}_1 \times \boldsymbol{t}_2$, which are the inward normals to the top and bottom faces,

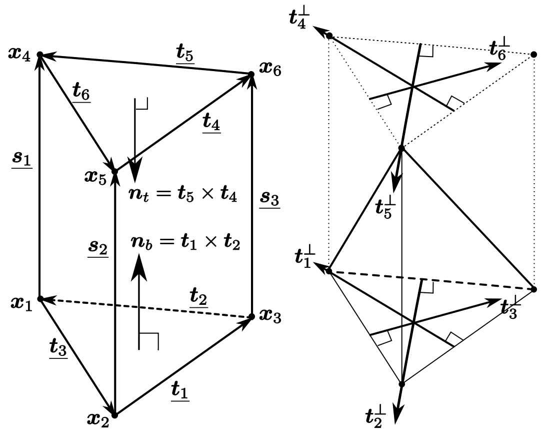


Figure 4.6: Naming convention of (a) vertices, edges, face normals, and (b) orthogonal directions of prism for energy computation. Underlined symbols indicate edges.

respectively. Let $\|\boldsymbol{x}\|$ denote the 2-norm of a vector $\boldsymbol{x}$. Let l_i denote $\|\boldsymbol{s}_i\|$, i.e. the lengths of the ith side edges. Let $a_t = \|\boldsymbol{n}_t\|$ and $a_b = \|\boldsymbol{n}_b\|$, i.e., twice of the areas of the top and bottom triangles, respectively. Let $V_i = \boldsymbol{s}_i^T \boldsymbol{n}_b$, i.e., six times of the tetrahedra spanned by the bottom triangle and the side edge $\boldsymbol{s}_i$, and similarly $V_{i+3} = -\boldsymbol{s}_i^T \boldsymbol{n}_t$. Let $\boldsymbol{t}_i^\perp = -\boldsymbol{t}_i \times \hat{\boldsymbol{n}}_b$ denote the $90°$ rotation of $\boldsymbol{t}_i$ in the bottom triangle pointing toward $\boldsymbol{x}_i$, and $\boldsymbol{t}_{i+3}^\perp = \boldsymbol{t}_{i+3} \times \hat{\boldsymbol{n}}_t$ be the $90°$ rotation of $\boldsymbol{t}_{i+3}$ in the top triangle pointing toward point $\boldsymbol{x}_{i+3}$, as illustrated in Figure~4.6(b). Let $\hat{\boldsymbol{s}}_i = \boldsymbol{s}_i / l_i$, $\hat{\boldsymbol{n}}_t = \boldsymbol{n}_t / a_t$, and $\hat{\boldsymbol{n}}_b = \boldsymbol{n}_b / a_b$, i.e., the unit vectors of $\boldsymbol{s}_i$, $\boldsymbol{n}_t$, and $\boldsymbol{n}_b$, respectively.

4.3.1 Elemental Energy for Triangle Shapes

We first consider the energy for the shapes of the top and bottom triangles. Following [31], we define the energy to measure the discrepancy between the actual triangle and a reference triangle, and compute the energy in $\mathbb{R}^3$ (instead of in $\mathbb{R}^2$). Let $E_{\theta,b}$ and $E_{\theta,t}$ denote the energies for the bottom and top triangles, respectively, and the elemental energy of triangle distortion of a prism τ is then $E_\theta(\tau) = E_{\theta,b} + E_{\theta,t}$. We describe only $E_{\theta,b}$, as $E_{\theta,t}$ can be obtained easily by symmetry.

As shown in [31], an effective energy for $E_{\theta,b}$ is the ratio between the sum of the squared edge lengths versus twice of the triangle area, derived from a conformal mapping between the actual triangle with a reference equilateral triangle. Its gradient and Hessian with respect to the vertex positions have simple closed forms.

In particular,

$$E_{\theta,b} = \frac{1}{a_b}\sum_{i=1}^{3}\|\boldsymbol{t}_i\|^2, \tag{4.3.2}$$

$$\nabla_{\boldsymbol{x}_i} E_{\theta,b} = \frac{1}{a_b}(2\boldsymbol{t}_{i+} - 2\boldsymbol{t}_{i-} - E_{\theta,b}\boldsymbol{t}_i^{\perp}), \tag{4.3.3}$$

$$\nabla^2_{\boldsymbol{x}_i} E_{\theta,b} \approx \frac{4}{a_b}\boldsymbol{I} - \frac{1}{a_b}\left(\left(\nabla_{\boldsymbol{x}_i} E_{\theta,b}\right)\boldsymbol{t}_i^{\perp T} + \boldsymbol{t}_i^{\perp}\left(\nabla_{\boldsymbol{x}_i} E_{\theta,b}\right)^T\right), \tag{4.3.4}$$

where $\boldsymbol{I}$ is the 3×3 identity matrix. The residual in (4.3.4) is $-\,(E_{\theta,b}\|\boldsymbol{t}_i\|^2/a_b^2)\,\hat{\boldsymbol{n}}_b\hat{\boldsymbol{n}}_b^T$, which is negligible because the vertices move nearly tangentially. From (4.3.4), it can be easily shown that $\nabla^2_{\boldsymbol{x}_i} E_{\theta,b}$ is symmetric positive definite as long as $a_b > 0$.

4.3.2 Elemental Energy for Edge Orthogonality

We derive a new energy function for measuring the orthogonality of the side edges. Let ϕ_i be the angle between $\boldsymbol{s}_i$ and $\boldsymbol{n}_b$ and ϕ_{i+3} the angle between $-\boldsymbol{s}_i$ and $\boldsymbol{n}_t$, i.e.,

$$\phi_i = \arccos\left(\frac{\boldsymbol{s}_i^T\boldsymbol{n}_b}{\|\boldsymbol{s}_i\|\|\boldsymbol{n}_b\|}\right) \text{ and } \phi_{i+3} = \arccos\left(\frac{-\boldsymbol{s}_i^T\boldsymbol{n}_t}{\|\boldsymbol{s}_i\|\|\boldsymbol{n}_t\|}\right). \tag{4.3.5}$$

Assuming the prism remains valid during mesh optimization, the quality $1/\cos\phi_j$ is minimized if $\phi_j = 0$ and approaches ∞ if $\phi_j = 90°$, so $1/\cos\phi_i$ and $1/\cos\phi_{i+3}$ effectively measure the deviation from orthogonality of $\boldsymbol{s}_i$ with respect to $\boldsymbol{n}_b$ and $\boldsymbol{n}_t$. Therefore, we define the elemental orthogonality energy for a prism τ as

$$E_{\perp}(\tau) = \sum_{j=1}^{6}\frac{1}{\cos^p\phi_j} = \sum_{i=1}^{3}\left(\left(\frac{\|\boldsymbol{s}_i\|\|\boldsymbol{n}_b\|}{\boldsymbol{s}_i^T\boldsymbol{n}_b}\right)^p - \left(\frac{\|\boldsymbol{s}_i\|\|\boldsymbol{n}_t\|}{\boldsymbol{s}_i^T\boldsymbol{n}_t}\right)^p\right). \tag{4.3.6}$$

In practice, we use $p = 1$, as it leads to simpler formulas. Like E_θ, $E_\perp$ is highly nonlinear. However, we can also derive closed-form formulas of the gradient and Hessian of $E_\perp$ with respect to the vertex positions.

For convenience, let $E_i = l_i a_b/V_i$ and $E_{i+3} = l_i a_t/V_{i+3}$. First, note that

$$\nabla_{\boldsymbol{x}_j} V_i = \begin{cases} \boldsymbol{s}_i \times \boldsymbol{t}_j & j = i\pm \\ \boldsymbol{t}_j \times (\boldsymbol{x}_{i-} - \boldsymbol{x}_{i+3}) & j = i \\ \boldsymbol{n}_b & j = i+3 \\ \boldsymbol{0} & \text{otherwise} \end{cases},$$

and

$$\nabla_{\boldsymbol{x}_j} V_{i+3} = \begin{cases} \boldsymbol{s}_i \times \boldsymbol{t}_j & j = (i\pm) + 3 \\ -\boldsymbol{t}_j \times (\boldsymbol{x}_{(i-)+3} - \boldsymbol{x}_i) & j = i + 3 \\ \boldsymbol{n}_t & j = i \\ \boldsymbol{0} & \text{otherwise} \end{cases}.$$

We obtain the following formulas for the gradients:

$$\nabla_{\boldsymbol{x}_j} E_i = \frac{1}{V_i} \begin{cases} l_i \boldsymbol{t}_j^{\perp} - E_i \nabla_{\boldsymbol{x}_j} V_i & j = i\pm \\ \cdots - a_b \hat{\boldsymbol{s}}_i & j = i \\ a_b \hat{\boldsymbol{s}}_i - E_i \nabla_{\boldsymbol{x}_j} V_i & j = i + 3 \\ \boldsymbol{0} & \text{otherwise,} \end{cases} \tag{4.3.7}$$

and

$$\nabla_{\boldsymbol{x}_j} E_{i+3} = \frac{1}{V_{i+3}} \begin{cases} l_i \boldsymbol{t}_j^{\perp} - E_{i+3} \nabla_{\boldsymbol{x}_j} V_{i+3} & j = (i\pm) + 3 \\ \cdots + a_t \hat{\boldsymbol{s}}_i & j = i + 3 \\ -a_t \hat{\boldsymbol{s}}_i - E_{i+3} \nabla_{\boldsymbol{x}_j} V_{i+3} & j = i \\ \boldsymbol{0} & \text{otherwise,} \end{cases} \tag{4.3.8}$$

where "$\cdots$" denotes the repetition of the previous row. To compute the Hessian, let us define a matrix

$$\boldsymbol{B}_{kj} = \left(\left(\nabla_{\boldsymbol{x}_j} E_k \right) \left(\nabla_{\boldsymbol{x}_j} V_k \right)^T + \left(\nabla_{\boldsymbol{x}_j} V_k \right) \left(\nabla_{\boldsymbol{x}_j} E_k \right)^T \right), \tag{4.3.9}$$

where $k = 1, ..., 6$. We obtain the following formulas for the Hessian:

$$\nabla^2_{\boldsymbol{x}_j} E_i = \frac{1}{V_i} \begin{cases} \frac{l_i}{a_b} \|\boldsymbol{t}_j\|^2 \hat{\boldsymbol{n}}_b \hat{\boldsymbol{n}}_b^T - \boldsymbol{B}_{ij} & j = i\pm \\ \cdots + \frac{a_b}{l_i} \left(\boldsymbol{I} - \hat{\boldsymbol{s}}_i \hat{\boldsymbol{s}}_i^T \right) - \left(\hat{\boldsymbol{s}}_i \boldsymbol{t}_j^{\perp T} + \boldsymbol{t}_j^{\perp} \hat{\boldsymbol{s}}_i^T \right) & j = i \\ \frac{a_b}{l_i} \left(\boldsymbol{I} - \hat{\boldsymbol{s}}_i \hat{\boldsymbol{s}}_i^T \right) - \boldsymbol{B}_{ij} & j = i + 3 \\ \boldsymbol{0} & \text{otherwise,} \end{cases} \tag{4.3.10}$$

and

$$\nabla^2_{\boldsymbol{x}_j} E_{i+3} = \frac{1}{V_{i+3}} \begin{cases} \frac{l_i}{a_t} \|\boldsymbol{t}_j\|^2 \hat{\boldsymbol{n}}_t \hat{\boldsymbol{n}}_t^T - \boldsymbol{B}_{(i+3)j} & j = (i\pm) + 3 \\ \cdots + \frac{a_t}{l_i} \left(\boldsymbol{I} - \hat{\boldsymbol{s}}_i \hat{\boldsymbol{s}}_i^T \right) + \left(\hat{\boldsymbol{s}}_i \boldsymbol{t}_j^{\perp T} + \boldsymbol{t}_j^{\perp} \hat{\boldsymbol{s}}_i^T \right) & j = i + 3 \\ \frac{a_t}{l_i} \left(\boldsymbol{I} - \hat{\boldsymbol{s}}_i \hat{\boldsymbol{s}}_i^T \right) - \boldsymbol{B}_{(i+3)j} & j = i \\ \boldsymbol{0} & \text{otherwise.} \end{cases} \tag{4.3.11}$$

It is fairly easy to implement the above formulas. As for $\nabla^2_{\boldsymbol{x}_j} E_\theta$, we can omit the $\hat{\boldsymbol{n}}_b \hat{\boldsymbol{n}}_b^T$ and $\hat{\boldsymbol{n}}_t \hat{\boldsymbol{n}}_t^T$ terms in $\nabla^2_{\boldsymbol{x}_j} E_i$ and $\nabla^2_{\boldsymbol{x}_j} E_{i+3}$.

4.3.3 Energy Minimization

Given the gradient and Hessian of the elemental energies, the gradient and Hessian of the total energy with respect to each vertex v is equal to the sum of those in the incident prisms of the vertex. Therefore,

$$\nabla_{\boldsymbol{x}_v} E = \sum_{\{\tau \mid v \in \tau\}} \left(\mu \nabla_{\boldsymbol{x}_v} E_\theta(\tau) + (1-\mu) \nabla_{\boldsymbol{x}_v} E_\perp(\tau)\right), \tag{4.3.12}$$

$$\nabla^2_{\boldsymbol{x}_v} E = \sum_{\{\tau \mid v \in \tau\}} \left(\mu \nabla^2_{\boldsymbol{x}_v} E_\theta(\tau) + (1-\mu) \nabla^2_{\boldsymbol{x}_v} E_\perp(\tau)\right), \tag{4.3.13}$$

where μ controls the importance of triangle shapes versus edge orthogonality.

In practice, we give higher priority to orthogonality and choose $\mu = 0.2$. After obtaining the gradient and Hessian, one could apply one step of Newton's method to determine a displacement $\boldsymbol{d}_v$ for the vertex v, i.e., by solving the equation $\left(\nabla^2_{\boldsymbol{x}_v} E\right) \boldsymbol{d}_v = -\nabla_{\boldsymbol{x}_v} E$. To preserve the height of the prisms, we constrain the vertex to move nearly tangentially within a layer. For a vertex of the current surface layer, let $\hat{\boldsymbol{t}}_1$ and $\hat{\boldsymbol{t}}_2$ denote two orthonormal tangent vectors at v. Let $\boldsymbol{T} = [\hat{\boldsymbol{t}}_1 \mid \hat{\boldsymbol{t}}_2]$. We obtain the displacement by solving $\left(\boldsymbol{T}^T \left(\nabla^2_{\boldsymbol{x}_v} E\right) \boldsymbol{T}\right) \boldsymbol{y} = -\boldsymbol{T}^T \nabla_{\boldsymbol{x}_v} E$ and then $\boldsymbol{d}_v = \boldsymbol{T}\boldsymbol{y}$, i.e.

$$\boldsymbol{d}_v = -\boldsymbol{T} \left(\boldsymbol{T}^T \left(\nabla^2_{\boldsymbol{x}_v} E\right) \boldsymbol{T}\right)^{-1} \boldsymbol{T}^T \nabla_{\boldsymbol{x}_v} E. \tag{4.3.14}$$

4.3.4 Overall Smoothing Algorithm

For the overall algorithm, it is most efficient and convenient to compute the gradient and Hessian of E_θ and $E_\perp$ over all the triangles, accumulate them at each vertex, and then compute the displacements for all the vertices concurrently. This is similar to the control flow of a typical finite-element code. When applying the displacements, we determine a relaxation factor α_v for each vertex and then add $\alpha_v \boldsymbol{d}_v$ to the vertex, where the relaxation factor is chosen to ensure mesh validity, as we explain in Section~4.4.2. This variational algorithm is essentially a block-diagonal solver for one step of Newton's method. Our algorithm repeatedly invokes this procedure for a desired number of iterations or until the mesh no longer improves. This procedure is simple and efficient and delivers sufficient accuracy for optimizing prisms.

4.4 Robustness and Implementation Issues

The preceding sections established the theoretical foundation of our framework for generating prismatic boundary layer meshes. To obtain a robust method for practical biological computations, we must address a few additional issues, including the verification of mesh validity, adaptation of step sizes, enforcement of boundary constraints, and generation of tetrahedral mesh of the interior.

4.4.1 Positivity of Jacobian

In finite element methods, the computation over a prismatic element is based on a mapping from a master element to the actual element. As illustrated in Figure~4.7, the mapping maps a point $\boldsymbol{\xi} = (\xi, \eta, \zeta)$ in the master element to a point $\boldsymbol{x}(\boldsymbol{\xi}) \equiv \sum_{i=1}^{6} N_i(\boldsymbol{\xi})\boldsymbol{x}_i$, where $\boldsymbol{x}_i = (x_i, y_i, z_i)$, and N_i is the shape function of the element associated with its ith vertex. For accuracy and stability, it is important that the Jacobian (aka the Jacobian determinant) of the mapping is positive everywhere within the element. Because the shape functions are nonlinear, it is nontrivial to verify the positivity of a prismatic element. We describe a simple and efficient technique to verify the positivity of the Jacobin for prisms in this subsection. This technique will also serve as the foundation of our method for preventing mesh folding in the next subsection.

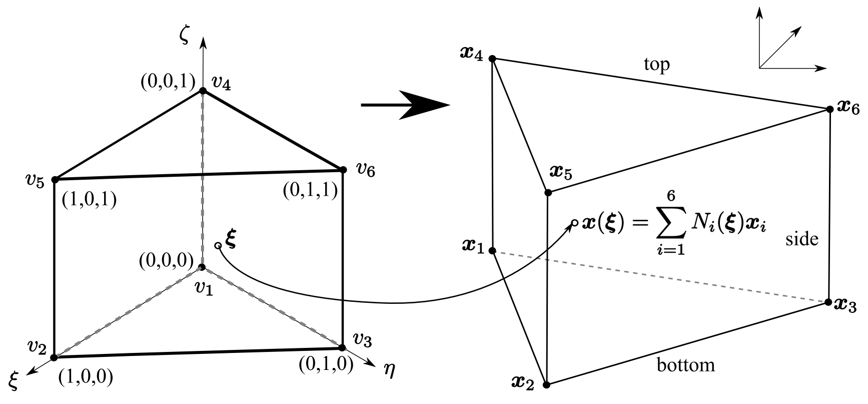


Figure 4.7: Schematic of mapping from master element to actual element for prisms.

Our technique is based on the following proposition, proved by professor Jiao: *The Jacobian is positive everywhere within a prism if and only if it is positive along all the side edges of the prism.* For a prismatic element, the standard shape functions of the prism are

$$\begin{array}{ll} N_1 = (1-\xi-\eta)(1-\zeta) & N_4 = (1-\xi-\eta)\zeta \\ N_2 = \xi(1-\zeta) & N_5 = \xi\zeta \\ N_3 = \eta(1-\zeta) & N_6 = \eta\zeta. \end{array} \tag{4.4.1}$$

Using the naming conventions in Section~4.3, the Jacobian matrix at a point $\boldsymbol{x}(\xi, \eta, \zeta)$ is

$$\boldsymbol{J} \equiv \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} & \frac{\partial x}{\partial \zeta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} & \frac{\partial y}{\partial \zeta} \\ \frac{\partial z}{\partial \xi} & \frac{\partial z}{\partial \eta} & \frac{\partial z}{\partial \zeta} \end{bmatrix} = \left[\begin{array}{c|c|c} \boldsymbol{j}_1 & \boldsymbol{j}_2 & \boldsymbol{j}_3 \end{array} \right], \tag{4.4.2}$$

where

$$\boldsymbol{j}_1 = \boldsymbol{t}_2 + \zeta(\boldsymbol{s}_2 - \boldsymbol{s}_1)\boldsymbol{j}_2 = \boldsymbol{t}_3 + \zeta(\boldsymbol{s}_3 - \boldsymbol{s}_1)\boldsymbol{j}_3 = \boldsymbol{u}_1 + \xi(\boldsymbol{s}_2 - \boldsymbol{s}_1) + \eta(\boldsymbol{s}_3 - \boldsymbol{s}_1).$$

The Jacobian is then $\det(\boldsymbol{J})$. It is obvious that $\det(\boldsymbol{J})$ is quadratic, because the terms $\zeta^2\xi$ and $\zeta^2\eta$ vanish. Furthermore, $\det(\boldsymbol{J})$ is linear in ξ and η. Note that the Jacobian is nonpositive if and only if the minimum Jacobian within the prism is zero or negative. Suppose a point $\boldsymbol{x}_0 = \boldsymbol{x}(\xi_0, \eta_0, \zeta_0)$ has the minimum Jacobian. This point is contained in the triangle with vertices $\boldsymbol{x}(0, 0, \zeta_0)$, $\boldsymbol{x}(0, 1, \zeta_0)$, $\boldsymbol{x}(1, 0, \zeta_0)$ (which are in general not the vertices of the prism). Because the Jacobian varies linearly within the triangle, the Jacobian attains its minimum at a vertex of the triangle, which is contained in a side edge. Therefore, the minimum Jacobian is nonpositive within a prism if and only if it is nonpositive along a side edge.

Based on this observation, we can therefore check the positivity of a prism by considering only the side edges. As in Section~4.3, let $\boldsymbol{s}_i = \boldsymbol{x}_{i+3} - \boldsymbol{x}_i$ for $i = 1, 2, 3$. Let $\boldsymbol{x}_{i,k}$ denote $\boldsymbol{x}_k - \boldsymbol{x}_i$ and $\boldsymbol{s}_{i,k}$ denote $\boldsymbol{s}_k - \boldsymbol{s}_i$. Note that ξ and η are constants along the ith side edge for $i = 1, 2, 3$. The Jacobian at the points on the edge is then a quadratic function in ζ, $J_i(\zeta) = a_i\zeta^2 + b_i\zeta + c_i$, where

$$a_i = (\boldsymbol{s}_{1,2} \times \boldsymbol{s}_{1,3})^T \boldsymbol{s}_i b_i = (\boldsymbol{s}_{1,2} \times \boldsymbol{x}_{1,3} + \boldsymbol{x}_{1,2} \times \boldsymbol{s}_{1,3})^T \boldsymbol{s}_i c_i = (\boldsymbol{x}_{1,2} \times \boldsymbol{x}_{1,3})^T \boldsymbol{s}_i.$$

The problem then reduces to the determination of the positivity of $J_i(\zeta)$ within the interval $\zeta \in [0, 1]$. In the presence of round-off errors, the equation $J_i(\zeta) = 0$ can be solved in a numerically stable fashion using the alternative quadratic formulas (see e.g. [23, p. 26]). Suppose $J_i(0) > 0$ for $i = 1, 2, 3$. For robustness, we consider J_i to be positive within the ith side edge only if none of its solutions is in the interval $[-\varepsilon_1, 1 + \varepsilon_2]$ for some positive ε_1 and ε_2. In practice, we choose $\varepsilon_1 \approx 10^{-6}$ to guard against round-off errors, and choose $\varepsilon_2 \approx 0.05$ or 0.1 so that the prism is reasonably far from degeneracy.

4.4.2 Adaptive Step Size Control

In our advancing layers algorithm, if the prescribed height of a layer is too large, some prisms may become folded. Such foldings can be detected by checking the Jacobian as described in the above. More importantly, we can extend the checking procedure to control the adaption of the step sizes for ensuring the validity of the mesh. At a high level, our basic idea is to verify the positivity of all the prisms, and if any prism is invalid, we then repeatedly reduce the vertex displacements and re-perform the checking until all elements are valid. We apply two different variants of this procedure to normal and tangential displacements, respectively.

Let us first describe the scaling procedure for normal motions. A difficulty arises because the user seems to have lost control of the height of the prismatic layers due to the scaling of the displacements. This problem is easily overcome by employing a "subcycling" technique. In particular, within each time step we take a few sub-steps toward the prescribed height of the current layer while ensuring the validity of the prisms. Let $\boldsymbol{x}_i$ denote the vertex coordinates of a triangle in the previous layer, $\boldsymbol{d}_i$ the accumulated displacements of the previous sub-steps of the current time step, and $\boldsymbol{u}_i$ be the displacements of the current sub-step. We determine a scaling factor α_g so that the prisms composed of vertices $\boldsymbol{x}_i$ and $\boldsymbol{x}_i + \boldsymbol{d}_i + \alpha_g \boldsymbol{u}_i$ are all positive. For simplicity, we determine α_g using bisection by starting with $\alpha_g = 1$ and repeatedly reduce α_g by half until all prisms are valid. After each sub-step, we perform a few iterations of mesh smoothing for the intermediate prismatic layer. This procedure automatically invokes more smoothing steps for more skewed meshes.

Mesh smoothing itself must also preserve the validity of the mesh. However, instead of scaling all of the vertex displacements by a global α_g, we compute a different scaling factor for each vertex for more effective smoothing. In particularly, we first determine an α_τ for each element τ by bisection so that the prism with vertices $\boldsymbol{x}_i$ and $\boldsymbol{x}_i + \boldsymbol{d}_i + \alpha_\tau \boldsymbol{u}_i$ is positive, where $i = 1, 2, 3$ and denotes the vertices of a triangle in the surface layer. We then set the scaling factor α_v for vertex v to be the minimum of α_τ among its incident triangles, i.e., $\alpha_v = \min_{\{\tau | v \in \tau\}} \alpha_\tau$. If $\alpha_v < 1$ for any vertex, we scale its displacement and recompute α_τ and α_v in its neighborhood. We repeat this process until $\alpha_v \geq 1$ for all vertices, which typically terminates in very few iterations.

4.4.3 Enforcement of Boundary Constraints

In biomechanical simulations, it is often necessary to impose inflow/outflow conditions, such as for specifying the velocity of the blood flow at the end of a blood vessel. Boundary layers are not desired for the inflow/outflow walls, so we must extend our algorithm to accommodate these needs. For simplicity, we assume the inlet/outlet walls are flat as they are in our applications.

For the purpose of meshing, we consider the inlet/outlet walls as *constrained patches*, on which vertices can move only tangentially (i.e., only smoothing is allowed), and consider the other surface patches as *advancing patches* (see Figure~4.8). In our framework, we simply set the speed function of the constrained faces to zero when computing the vector $\boldsymbol{f}$ in (4.2.3) and then solve (4.2.5) as usual. For the vertices at the curves between the advancing and constrained patches, the displacement $\boldsymbol{d}$ from (4.2.5) would then be nearly tangential to the constrained patch and nearly orthogonal to the advancing patch, just as desired. For better accuracy, we further project $\boldsymbol{d}$ orthogonally onto the constrained surface. Figure~4.9 shows an example of the boundary layer mesh under boundary constraints, where the initial triangulated inlet wall in Figure~4.9(a) is replaced by a hybrid triangle-quadrilateral mesh in Figure~4.9(b).

Finally, for completeness **Algorithm 5** summarizes our overall algorithm for advancing the layers with subcycling and boundary constraints. Note that the time step Δt may differ from layer to layer to allow gradation of the mesh. Note that in the case of propagation and interpolation, we simply run the algorithm with $\mathrm{nlayers}$ equal to 1 and then subdivide the side edges into subintervals to obtain the intermediate layers. We have implemented our algorithm for the generation of boundary-layer meshes in MATLAB and converted it into C code using Agility MCS (www.agilityds.com).

4.4.4 Generation of Interior Tetrahedral Meshes

In general, there are two paths to follow in order to fill out the interior with tetrahedra: we can construct a tetrahedral interior and project the prisms to the original surface, or we can construct the prismatic boundary layer and fill in the tetrahedral core. The former allows more freedom in the construction and improvement of the tetrahedral core, but that freedom comes at the cost of constraints on the construction of the prismatic boundary layer. The latter path restricts the construction of the tetrahedral core in a very specific way: it must be boundary constrained. In other words, the triangulation of the interface between prismatic boundary layer and tetrahedral core must be unchanged. We generate boundary constrained tetrahedral core using constrained Delaunay tetrahedralization in Tetgen (http://tetgen.berlios.de/) developed by Si [70]; for more details on constrained Delaunay tetrahedralization, see e.g., [68, 70].

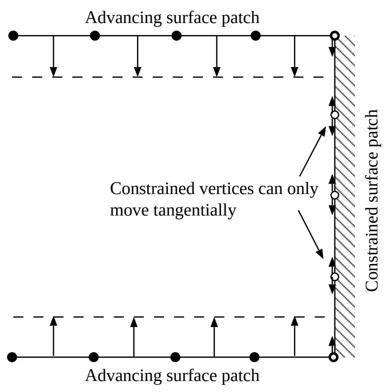
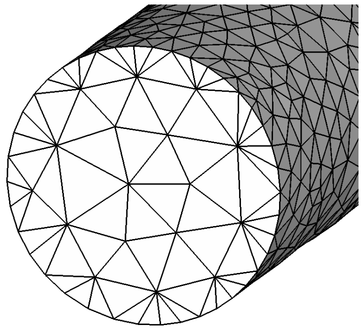
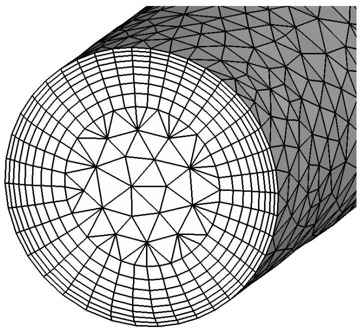

Advancing surface patch
Constrained vertices can only move tangentially
Constrained surface patch
Advancing surface patch

Figure 4.8: 2-D illustration of constraints.



(a) Initial surface mesh.



(b) Final surface mesh.

Figure 4.9: Example of evolution of surface under boundary constraints.

Algorithm 5 Advancing layers by face-offsetting.

```
1: for i = 1 .. number of layers
2:     Comment: construct layer by layer
3:     $\delta t \leftarrow \Delta t_i$
4:     while $\delta t > 0$
5:         compute normal displacements $\boldsymbol{u}_v$ at all vertices with GLFS as speed function;
6:         impose boundary constraints;
7:         determine global rescaling factor $\alpha_g \in (0, 1]$
8:         if $\alpha_g$ is too small
9:             stop and return layers up to $i - 1$
10:        end if
11:        for desired number of iterations
12:            Comment: Perform mesh smoothing
13:            compute tangential displacements $\boldsymbol{t}_v$ at all vertices;
14:            determine local rescaling factors $\alpha_v$ for each vertex v;
15:            move each vertex to $\boldsymbol{x}_v \leftarrow \boldsymbol{x}_v + \alpha_v \boldsymbol{t}_v$;
16:        end for
17:        $\delta t \leftarrow (1 - \alpha_g)\delta t$;
18:    end while
19:    save $\boldsymbol{x}_v$ as the $i$th surface layer;
20: end for
```

4.5 Experimental Assessment and Comparisons

In this section, we assess the robustness and quality of our proposed method for generating prismatic boundary layer meshes. We report the results with two complex biological geometries: a rat lung and a human heart, and compare our method with some other alternatives. We evaluate the meshes in terms of four quality measures, which we define next.

4.5.1 Quality Measures for Prisms

Good quality measures are critical for assessing and comparing different methods. To the authors' knowledge, there is no previously well-established quality measures for prisms. A measure sometimes used in the literature is the so called *scaled Jacobian* (or *minimum scaled Jacobian*, see e.g., [15]). At each vertex of a prism, the scaled Jacobian is the determinant of the matrix composed of the unit vectors of its three edges, and the quality of the prism is then measured by the minimum among the vertices. The scaled Jacobian was initially defined for hexahedra [42], where the three edges should ideally be mutually orthogonal. However, this measure is not ideal for prisms, because a prism may be inverted even if the Jacobian (and in turn the scaled Jacobian) is positive at all the vertices. In addition, the pointwise scaled Jacobian does not attain its maximum value at the vertices of an ideal prism.

We propose a different quality measure for prisms, referred to as the {\em scaled aspect ratio}, which resolves the above issues of scaled Jacobian. First, we define a pointwise measure at each point within the prism as

$$\rho(\boldsymbol{\xi}) = \left(\frac{2\sqrt{3}\boldsymbol{j}_1 \times \boldsymbol{j}_2}{\|\boldsymbol{j}_1\|^2 + \|\boldsymbol{j}_2\|^2 + \|\boldsymbol{j}_1 - \boldsymbol{j}_2\|^2} \right)^T \frac{\boldsymbol{j}_3}{\|\boldsymbol{j}_3\|}, \tag{4.5.1}$$

where $\boldsymbol{j}_i$ is given in (4.4.2). For a prism τ that has positive Jacobian everywhere, we define the *scaled aspect ratio* of τ, denoted by $\rho(\tau)$, to be the minimum of ρ among the six vertices; for a prism that has nonpositive Jacobian, we define $\rho(\tau)$ to be the minimum of $\rho(\boldsymbol{\xi})$ among the points that attain the most negative value of $\det(\boldsymbol{J})$ in the side edges, i.e.,

$$\rho(\tau) = \begin{cases} \min\{\rho(\boldsymbol{v_i}) \mid i = 1, \dots, 6\} \text{if } \tau \text{ is not inverted} \\ \min\{\rho(\boldsymbol{\xi}_k) \mid \det(\boldsymbol{J}(\boldsymbol{\xi}_k)) \text{ is most negative along } k\text{th side edge}\} \text{otherwise.} \end{cases} \tag{4.5.2}$$

Note that our boundary-layer algorithm guarantees positive Jacobian. However, some other methods may not guarantee positive Jacobian, so the general definition above is necessary for comparison purposes.

This scaled aspect ratio has both geometric and algebraic meanings. The magnitude of the first term in (4.5.1) measures the aspect ratio of the triangle with edges $\boldsymbol{j}_1$, $\boldsymbol{j}_2$, and $\boldsymbol{j}_1 - \boldsymbol{j}_2$, and it resembles the reciprocal of the triangle-shape energy (4.3.2). The inner product in (4.5.1) resembles those in the edge-orthogonality energy (4.3.6). Therefore, this quality measure in effect combines the measures of triangle shapes and edge orthogonality, which we optimize in our mesh smoothing step. Note that $\rho(\tau) \in [-1, 1]$, $\rho = 1$ for ideal prisms, and $\rho \leq 0$ indicates an inverted element. Algebraically, we can rewrite (4.5.1) as

$$\rho(\boldsymbol{\xi}) = \frac{2\sqrt{3}\det(\boldsymbol{J})}{\|\boldsymbol{j}_3\|\left(\|\boldsymbol{j}_1\|^2 + \|\boldsymbol{j}_2\|^2 + \|\boldsymbol{j}_1 - \boldsymbol{j}_2\|^2\right)}, \tag{4.5.3}$$

where $\boldsymbol{J}$ is given in (4.4.2). From (4.5.3), it is clear that the scaled aspect ratio involves a simple "fix" to the scaled Jacobian for prisms.

Besides scaled aspect ratio, we also consider the statistics of the angles of the triangles ($60°$ is ideal) and the distortions of the side edges with respect to the face normals, which we refer to as *triangle angles* and *edge distortions*, respectively. We compute the latter as the angles between the side edges and the upward normals to the triangles of the prisms, so $0°$ is ideal. These can be considered as measures of triangle shapes and edge orthogonality, respectively. Although the scaled aspect ratio already contains information about both of these measures, these statistics are more intuitive to understand and sometimes revealing. For the tetrahedral interior, we report the *aspect ratio* χ of a tetrahedron, defined as

$$\chi \equiv 2\sqrt{6}\frac{\text{inscribed radius}}{\text{length of longest edge}} \tag{4.5.4}$$

which has a maximum value of 1.0 for regular tetrahedra.

4.5.2 Rat Lung

Our first example geometry, as shown in Figure~4.10, is a rat lung, whose surface has genus 0. Magnetic imaging data was acquired using silicone casts of pulmonary airways in a 9-wk-old, male, Sprague-Dawley rat. The surface was extracted with a variant of the marching cubes algorithm [73].

Panel A shows the gradient-limited feature sizes (GLFS) $f[x]$ and $f_{out}[x]$. Adapted surface mesh contained 162,923 vertices and 325,842 triangles.

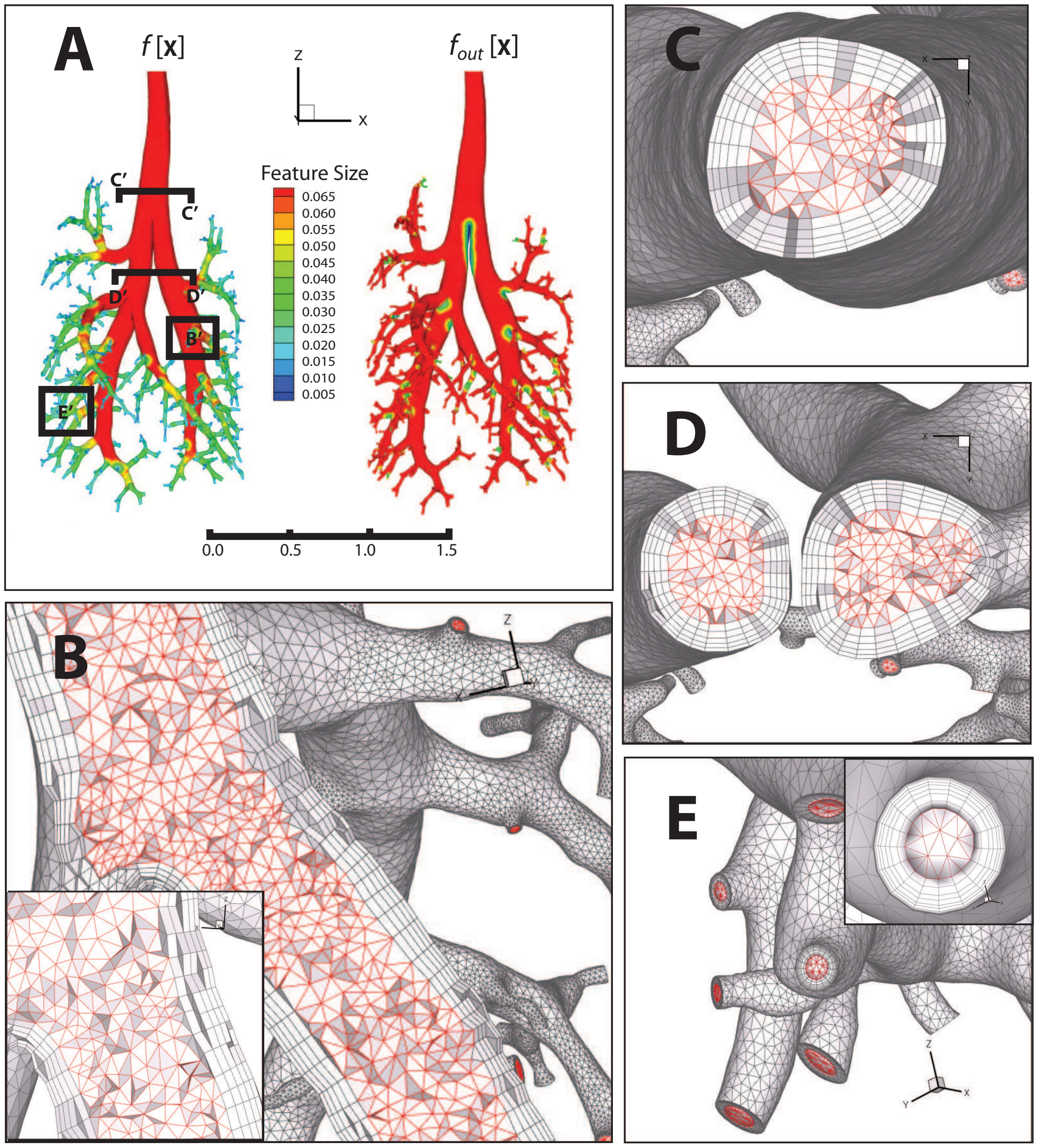
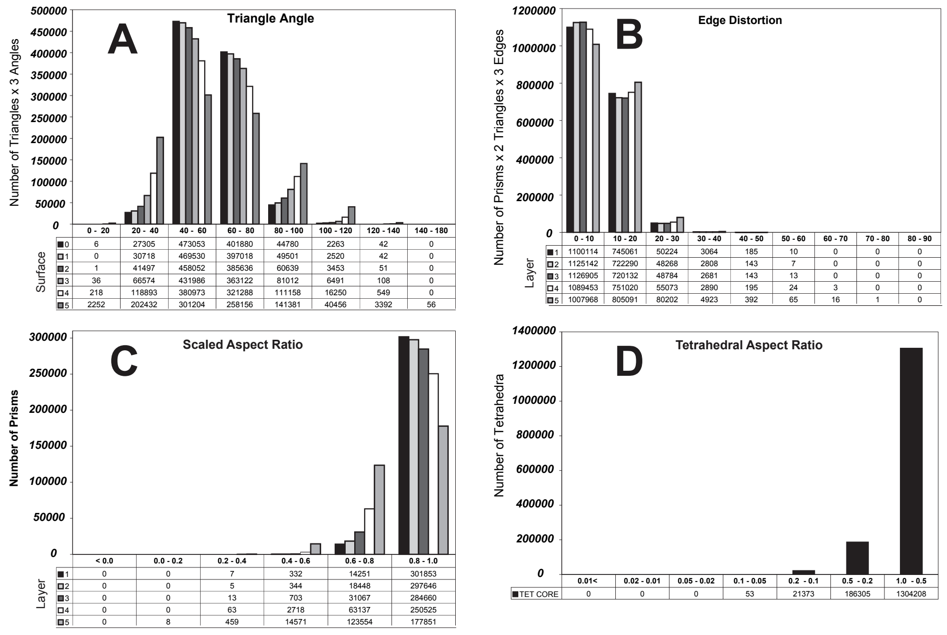


Figure 4.10: Boundary layer mesh of a rat lung. Panel A shows the feature size fields on the lung surface, as well as the locations of the cutaway sections in panels B–E. Panel B shows the boundary prism and interior tetrahedra at a region where a large airway , with f(x) greater than , transitioning to a smaller airway. Panels C and D show the sections through the trachea and through a pair of airways downstream of the bifurcation, respectively. Panel E shows the resulting surface mesh in the neighborhood of several outlets, where the faces are constrained.

Quality Measurements

Our algorithm generated five layers of prisms for a total height of about 40% of the GLFS. The layers were graded with an exponent of 1.2, so the height of first layer was about 4.48% and that of the last layer was about 11.15%. We used face-offsetting with both shape and orthogonality optimization and with interpolated layers. The generation of the prismatic layers took about 36 minutes on a PC with a 3GHz Intel Core 2 Duo CPU. The generation of the tetrahedral mesh took about 2 minutes. The resulting hybrid mesh is shown in Figure~4.10.

Panel B shows the prismatic boundary layer and interior tetrahedra in a large airway transitioning to a smaller airway. Panels C and D show sections through the trachea and a pair of airways downstream of the bifurcation, respectively. Panel E shows both the external boundary and the prismatic layer in the neighborhood of several outlets that are roughly 1/30 of the diameter of the trachea. The outlet surfaces in panel E illustrate the enforcement of the boundary constraints of the prismatic boundary layers described in subsection~4.4.3.



Surface	0 - 20	20 - 40	40 - 60	60 - 80	80 - 100	100 - 120	120 - 140	140 - 180
■ 0	6	27305	473053	401880	44780	2263	42	0
□ 1	0	30718	469530	397018	49501	2520	42	0
▨ 2	1	41497	458052	385636	60639	3453	51	0
▨ 3	36	66574	431986	363122	81012	6491	108	0
□ 4	218	118893	380973	321288	111158	16250	549	0
▨ 5	2252	202432	301204	258156	141381	40456	3392	56

Layer	0 - 10	10 - 20	20 - 30	30 - 40	40 - 50	50 - 60	60 - 70	70 - 80	80 - 90
■ 1	1100114	745061	50224	3064	185	10	0	0	0
□ 2	1125142	722290	48268	2808	143	7	0	0	0
▨ 3	1126905	720132	48784	2681	143	13	0	0	0
□ 4	1089453	751020	55073	2890	195	24	3	0	0
▨ 5	1007968	805091	80202	4923	392	65	16	1	0

Layer	< 0.0	0.0 - 0.2	0.2 - 0.4	0.4 - 0.6	0.6 - 0.8	0.8 - 1.0
■ 1	0	0	7	332	14251	301853
□ 2	0	0	5	344	18448	297646
▨ 3	0	0	13	703	31067	284660
□ 4	0	0	63	2718	63137	250525
▨ 5	0	8	459	14571	123554	177851

	0.01<	0.02 - 0.01	0.05 - 0.02	0.1 - 0.05	0.2 - 0.1	0.5 - 0.2	1.0 - 0.5
■ TET CORE	0	0	0	53	21373	186305	1304208

Figure 4.11: Mesh-quality statistics for the prismatic layers and interior tetrahedra of the rat lung mesh generated using our proposed method. Panel A shows the triangle angles for each surface layer, starting with the original surface mesh ($60°$ is ideal). Panel B shows the edge distortions of each prismatic layers ($0°$ is ideal). Panel C shows the scaled aspect ratios of prisms for each prismatic layers (1.0 is ideal, and a negative value would indicate an inverted prism). Finally, panel D shows the aspect ratios for tetrahedral elements (1 is ideal)

Figure 4.11 shows three prism statistics: triangle angle, edge distortion and scaled aspect ratio for the five layers generated, along with the conventional aspect ratio for tetrahedra. For both aspect-ratio measures, the vertical axes are the numbers of their corresponding elements. However, for the triangle angles, the vertical axis is equal to the number of triangles times three, and for edge distortions it is equal to the number of prisms times six (since there are two triangles and three side edges per prism). As can be readily seen, the peak angles of the original surface mesh and the final offset surface were between $40°$ and $80°$. For edge distortion, the peak remained in the $0° - 10°$ range. For scaled aspect ratio, the peak was in the $0.8 - 1.0$ range, and very few prisms were in the $0.0 - 0.2$ range in the final layer, with a minimum of 0.113. It is beyond the scope of this paper to report detailed simulation results using boundary-layer meshes, which we will report elsewhere. Nevertheless, Figure~4.12 shows the result from a transient CFD simulation using our meshes, demonstrating the usability of our method for our targeted biomedical applications.

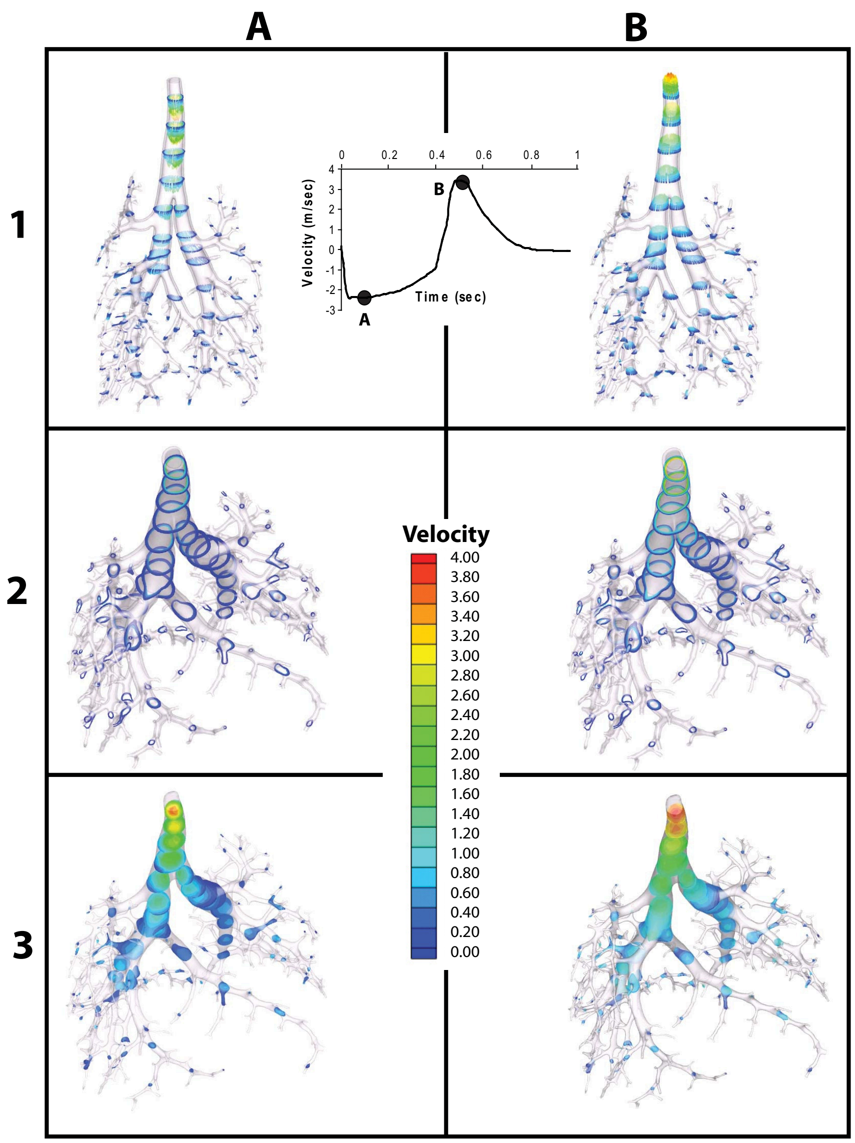


Figure 4.12: Transient CFD results for the pulmonary airways of 9-wk-old, male rat. Flow data (inset) were experimentally determined from a mechanical ventilator and prescribed to the trachea. Downstream outlets were uniformly prescribed zero pressure boundary conditions. Columns $\mathbf{A}\&\mathbf{B}$ correspond to time points $\mathbf{A}\&\mathbf{B}$ on the inset velocity curve, roughly corresponding in turn to peak inhalation and peak exhalation. Velocity vectors and velocity contours through each domain (prismatic or tetrahedral) are shown in respectively in rows $\mathbf{1} - \mathbf{3}$.

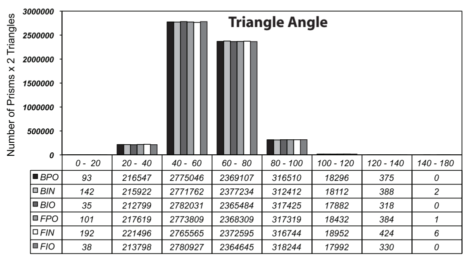
Comparison of Different Alternatives

To get an indirect comparison with other methods, we note that most competing algorithms are based on vertex propagation rather than face offsetting. To our knowledge, few (if any) of the existing methods optimize the orthogonality of prisms. We therefore compared our method against vertex propagation. We consider two alternatives of normal computations for vertex propagation: area-weighted vertex normals, which we refer to as *AVP*, and the normal computation proposed by Kallinderis and Ward in [37], which we refer to as *BVP*. To evaluate the effectiveness of our mesh-optimization and layer-interpolation strategies, we ran 18 different cases, as indicated in the first panel in Figure~4.13, where "Interp." indicates the propagation by a single big step followed by interpolating the intermediate layers, and "Prop." indicates that each layer is generated by propagating layer by layer. For each case, we let each method propagate up to 40% of the GLFS, invoked subcycling whenever necessary, and stopped the propagation if the subcyling time step became too small ($\leq 10^{-5}$). As shown in the first panel in Figure~4.13, only the methods with our full optimization strategy (i.e., optimizing both triangle shape and edge orthogonality) marched the desired distance, demonstrating the effectiveness of our smoothing strategy in enhancing the robustness for prismatic boundary layers. Note that when optimizing only triangle angles, all the methods failed after mere 0.13% propagation because of severe edge distortions. In terms of the propagated and interpolated layers, the former was more robust without mesh smoothing, but the latter became more robust with our optimization. Furthermore, a layer-by-layer propagation could not produce the desired number of layers if it fails in the middle, but the interpolation schemes can always produce the desired number of layers even if the propagation fails prematurely, and it is also convenient to change the number of layers. Therefore, we advocate using interpolated layers with variational optimization for better robustness and flexibility. Between vertex propagation and face offsetting, it seemed to be a tie for this particular geometry, but face offsetting will be shown to be more robust for a more complex geometry in the next subsection.

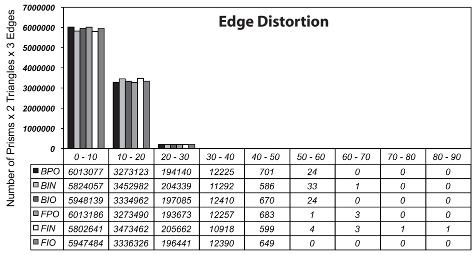
To assess the quality of the resulting meshes, we compare six out of the 18 cases to propagate, encoded as BPO, BIN, BIO, FPO, FIN, and FIO, where the first letter (B or F) indicates BVP versus FOM, the second letter (P or I) indicates Prop. versus Interp., and the third letter (N or O) indicates no-optimization versus full optimization. For fairness of comparison, we reran the six cases to generate boundary layers with a total height equal to 15% of the GLFS, which were about the maximum distance that BIN and FIN could propagate. The last three panels in Figure~4.13 shows the statistics of the prismatic layers for these six cases. Overall, different methods produced similar overall distributions of mesh quality, but FIO was notably better than the others in terms of the worst edge distortion and scaled

Marched distances of different alternatives

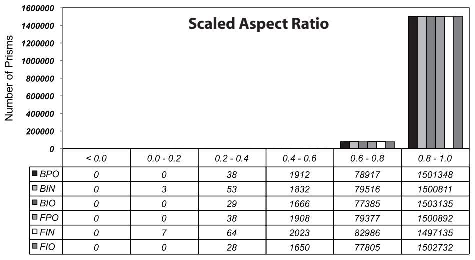
	Smoothing	AVP	**B**VP	**F**OM
Prop.	None	19.5%	19.6%	19.6%
	Triangle only	0.12%	0.13%	0.13%
	Triangle & **O**rth.	28.8%	**40%**	**40%**
Interp.	**N**one	18.0%	**14.5%**	**15.4%**
	Triangle only	0.12%	0.13%	0.13%
	Triangle & **O**rth.	40%	**40%**	**40%**



	0 - 20	20 - 40	40 - 60	60 - 80	80 - 100	100 - 120	120 - 140	140 - 180
■ BPO	93	216547	2775046	2369107	316510	18296	375	0
□ BIN	142	215922	2771762	2377234	312412	18112	388	2
■ BIO	35	212799	2782031	2365484	317425	17882	318	0
■ FPO	101	217619	2773809	2368309	317319	18432	384	1
□ FIN	192	221496	2765565	2372595	316744	18952	424	6
■ FIO	38	213798	2780927	2364645	318244	17992	330	0



	0 - 10	10 - 20	20 - 30	30 - 40	40 - 50	50 - 60	60 - 70	70 - 80	80 - 90
■ BPO	6013077	3273123	194140	12225	701	24	0	0	0
□ BIN	5824057	3452982	204339	11292	586	33	1	0	0
■ BIO	5948139	3334962	197085	12410	670	24	0	0	0
■ FPO	6013186	3273490	193673	12257	683	1	3	0	0
□ FIN	5802641	3473462	205662	10918	599	4	3	1	1
■ FIO	5947484	3336326	196441	12390	649	0	0	0	0



	< 0.0	0.0 - 0.2	0.2 - 0.4	0.4 - 0.6	0.6 - 0.8	0.8 - 1.0
■ BPO	0	0	38	1912	78917	1501348
□ BIN	0	3	53	1832	79516	1500811
■ BIO	0	0	29	1666	77385	1503135
■ FPO	0	0	38	1908	79377	1500892
□ FIN	0	7	64	2023	82986	1497135
■ FIO	0	0	28	1650	77805	1502732

Figure 4.13: Comparative statistics for prismatic layers using different alternatives. First panel shows the options used for each series and also their corresponding marched distances as percentage of GLFS. Last three panels show quality metrics of selected methods after propagating 15% of GLFS.

aspect ratio. Finally, we note that vertex propagation was about $10 - 20\%$ slower than FOM, because the former required far smaller time steps than FOM to avoid negative Jacobian.

4.5.3 Human Heart

Our second example, as shown in Figure~4.14, is an idealized human heart model. This model was originally from the NYU Medical Center. It is relatively realistic and highly complex topologically, with a genus 12. The heart mesh is composed of two distinct material regions: the heart musculature and the enclosed blood. The blood region in turn consists of two disconnected components, the left and right ventricular and atrial cavities. Panels A and B of Figure~4.14 show the GLFS of the tissue $f_{Tissue}[x]$ and blood $f_{Blood}[x]$, respectively.

For the heart, we constructed three, instead of two mesh domains. The tissue of the heart muscle was tessellated by a layered tetrahedral meshing algorithm described in [44]. The interior surface of this layered tetrahedral region was the boundary of the blood region that was to be offset. Within the offset boundary, we produced a boundary-constrained tetrahedral core. Surface mesh has $64,865$ vertices and $129,722$ triangles.

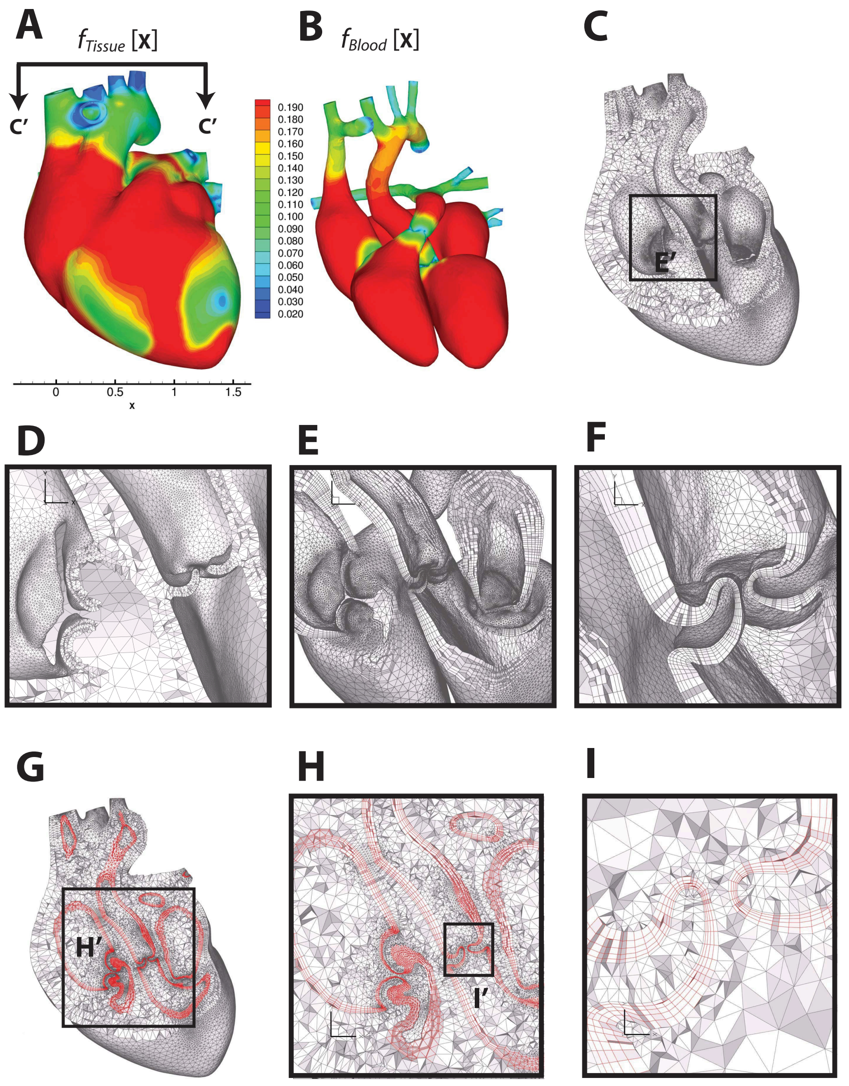


Figure 4.14: Multi-material boundary layer mesh of a human heart, consisting of heart tissue, blood boundary layer and blood domains. Panels A and B show the feature size field on the outer surface of the heart tissue domain and the blood domain, respectively. Panel B shows the feature size field on the outer surface of the blood domain. Panel C shows a cut through the heart muscle tessellated with layered tetrahedra [44], where the orientation of the cut plane is indicated in Panel A. Panel D shows the detail of the layered tetrahedral mesh of the heart tissue. Panels E and F are zoomed-in views on the regions of the cardiac valves, showing the prismatic boundary layer alone. Panels G, H and I show the prismatic boundary layer at about 25% of GLFS of the blood domain, sandwiched between the layered tetrahedra of the tissue and the Delaunay tetrahedra of the blood. Heart model and surface mesh are courtesy of the NYU Medical Center and Zhang et al. .

Quality Measurements

We first generated a six-layer prismatic mesh with our method by marching for about 27% of the GLFS of the blood domain, again using the default mode of our proposed method, namely face offsetting with full optimization and with interpolated layers. The propagation and construction of the prismatic layers took about 20 minutes on a PC in the same setting as for the lung mesh. The generation of the tetrahedral mesh for the heart tissues took 30 minutes, and that for the blood region took 5 minutes. The resulting hybrid, multi-material mesh is shown in Panels C–I of Figure 4.14.

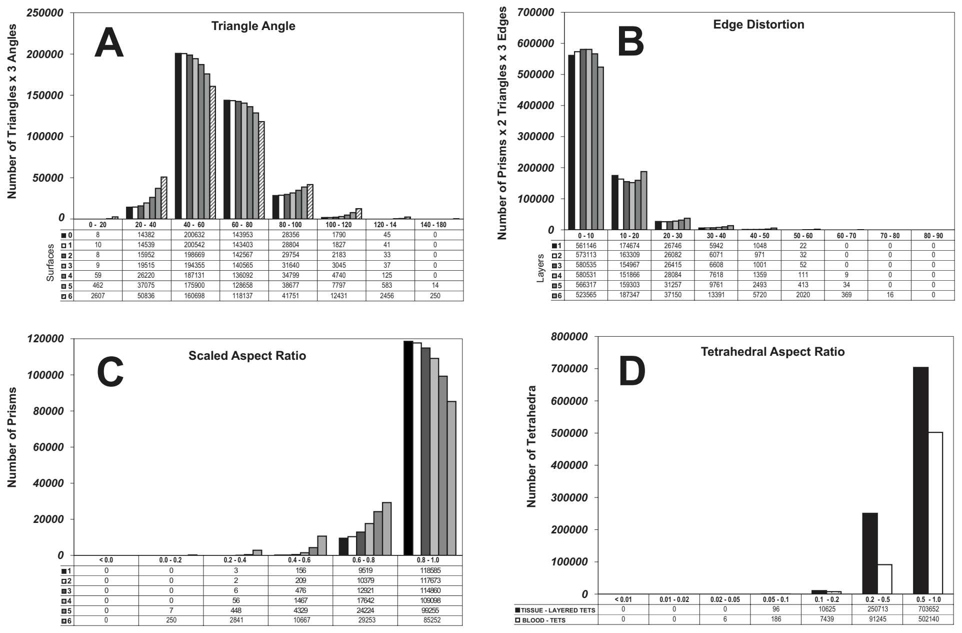
Panels C and D show the layered tetrahedral tissue mesh in isolation. In this case, we chose to generate three layers of tetrahedra, corresponding to the histologically distinct myocardial, valvular and arteriovenous layers of the tissue. The prismatic boundary layer is shown in Panels E and F. Note the ordered layers of prisms transitioning from the larger myocardial scale to the lower valvular scale. Panels G, H and I show the three regions integrated at various levels of scale.

Figure 4.15 shows the three prism statistics: triangle angle, edge distortion and scaled aspect ratio for the six layers generated, as well as the conventional aspect ratio for tetrahedra.

The distribution of triangle angles did not significantly degrade in the final offset surface. The same trend can be seen in the edge distortion statistics, where peak distortion remained in the $0° - 10°$ range. Only a few prisms had a distortion greater than $70°$, with a maximum of $77°$. In terms of the scaled aspect ratio, the majority of the prisms are close the 1.0, the ideal value. Only a handful were in the 0.0-0.2 range, with a minimum of 0.085.

Comparison of Different Alternatives

As for the lung mesh, we compared a number of methods for generating prismatic boundary layers, including face offsetting (FOM) as well as vertex propagation (AVP and BVP), and ran the same 18 combinations. For each variant, we march the surface as far as possible, and recorded the marched percentages of GLFS in Table. Most methods marched less than 10% of the GLFS. The clear winner was our proposed method, namely face offsetting with full optimization and layer interpolation, which reached 27%. The second place was BVP, namely vertex propagation using the normal computation of Kallinderis and Ward in [37], augmented with our mesh optimization and layer interpolation, which marched 17.1%. These results once again show that our mesh optimization and layer interpolation techniques are effective in enhancing the robustness of prismatic boundary layers. In addition, this example also demonstrates that FOM is more robust than vertex propagation for complex geometries.



Surfaces	0 - 20	20 - 40	40 - 60	60 - 80	80 - 100	100 - 120	120 - 14	140 - 180
■ 0	8	14382	200632	143953	28356	1790	45	0
□ 1	10	14539	200542	143403	28804	1827	41	0
■ 2	8	15952	198669	142567	29754	2183	33	0
■ 3	9	19515	194355	140565	31640	3045	37	0
■ 4	59	26220	187131	136092	34799	4740	125	0
■ 5	462	37075	175900	128658	38677	7797	583	14
▨ 6	2607	50836	160698	118137	41751	12431	2456	250

Layers	0 - 10	10 - 20	20 - 30	30 - 40	40 - 50	50 - 60	60 - 70	70 - 80	80 - 90
■ 1	561146	174674	26746	5942	1048	22	0	0	0
□ 2	573113	163309	26082	6071	971	32	0	0	0
■ 3	580535	154967	26415	6608	1001	52	0	0	0
■ 4	580531	151866	28084	7618	1359	111	9	0	0
■ 5	566317	159303	31257	9761	2493	413	34	0	0
■ 6	523565	187347	37150	13391	5720	2020	369	16	0

	< 0.0	0.0 - 0.2	0.2 - 0.4	0.4 - 0.6	0.6 - 0.8	0.8 - 1.0
■ 1	0	0	3	156	9519	118585
□ 2	0	0	2	209	10379	117673
■ 3	0	0	6	476	12921	114860
■ 4	0	0	56	1467	17642	109098
■ 5	0	7	448	4329	24224	99255
■ 6	0	250	2841	10667	29253	85252

	< 0.01	0.01 - 0.02	0.02 - 0.05	0.05 - 0.1	0.1 - 0.2	0.2 - 0.5	0.5 - 1.0
■ TISSUE - LAYERED TETS	0	0	0	96	10625	250713	703652
□ BLOOD - TETS	0	0	6	186	7439	91245	502140

Figure 4.15: Mesh quality statistics for the prismatic layers and interior tetrahedra of the human heart mesh generated using our proposed method. Panel A shows the angles for each surface layer (60° is ideal). Panel B shows the edge distortions of each prismatic layer (0° is ideal). Panel C shows the the scaled aspect ratio for prisms of each prismatic layer (1 is ideal, and a negative value would indicate an inverted prism). Finally, panel D shows the aspect ratio for tetrahedral elements (1 is ideal)

There appear to be two main reasons for vertex propagation to be less robust than FOM. First, the heart mesh contained some convex areas with large curvatures, which quickly evolve into sharp edges after a moderate amount of propagation due to the coarse mesh resolution. These sharp edges cause difficulties for normal computations, even for the normal computation in [37], causing some vertex normals to point opposite to some face normals and in turn negative Jacobian. FOM overcomes the problem using an eigenvalue analysis to deliver more reliable directions for the propagation. The second reason is that unlike FOM or the level set methods, vertex propagation is not based on the Huygens' principle (or its generalization) and can lead to severe lagging at sharp convex regions. This deficiency of vertex propagation tends to accelerate the formation of sharp edges and cause early termination of the propagation. Note that these difficulties with convex sharp edges are the main motivations for the use of multiple normal directions in previous methods [19, 26, 27], which are unfortunately difficult to apply for complex biological geometries, because the sharp edges do not correspond to well-defined sharp features in the surface and they emerge dynamically.

4.5.4 Comparison with Commercial Software

Our comparisons with other methods for the lung and heart meshes were based on our own implementations of different alternatives, so they only serve as indirect comparisons with other methods. Due to the lack of standard benchmarks, it is admittedly difficult to compare against other software implementation. For completeness, we nevertheless report a comparison against a mesh generator from a leading CFD company. Like a few others, this software can import a surface mesh to reconstruct the geometry (a discrete CAD model) and then generate a boundary-layer mesh using different algorithms. The algorithms supported by this software include one that produces uniform heights of the prisms, one that adapts the heights

Table 4.1: Marched percentage of the GLFS for heart geometry using different methods. "Prop." and "Interp." refer to propagated and interpolated layers, respectively.

	AVP		BVP		FOM	
	Prop.	Interp.	Prop.	Interp.	Prop.	Interp.
No mesh optimization	3.8%	2.7%	3.5%	5.8%	3.6%	8.0%
Optimize triangle only	6.0%	5.9%	6.1%	6.1%	10.0%	9.1%
Optimize triangle & orth.	8.0%	12.5%	9.5%	17.1%	9.9%	**27%**

of the prisms based on the edge lengths of the initial input surface, and one that adapts the heights of the prisms based on the edge lengths of the propagated surface. None of these options allows the adaptation of the boundary layer based on a user-specified field, such as the GLFS. However, since our surface meshes are adapted based on the GLFS, the latter two options can indirectly adapt the heights of the prisms based on GLFS. Therefore, we therefore compare our method against the latter two algorithms and refer to them as Commercial A and Commercial B, respectively.

We first attempted to generate prismatic boundary layers from the lung and the heart surface meshes. Even after our best effort (including consultation with the principal developer of this capability in the commercial software), for the lung mesh the commercial software failed to properly reconstruct the surface patches, which is a pre-requisite for the software to generate boundary-layer meshes. For the heart mesh, the software successfully reconstructed the surface patches, but it repeatedly generated prismatic meshes that had self intersections or that were too skewed for that software itself to produce a valid tetrahedral mesh. With our best possible effort, we were able to generate only a very thin layer of prisms to cover approximately 3% of the domain using Algorithm B of the software. Figure~4.16 shows a visual comparison of the resulting meshes using the commercial software and our method.

The above tests demonstrated the tremendous difficulties in generating prismatic boundary-layer meshes for complex geometries even for the state-of-the-art commercial software. This is not surprising, because as we noted earlier, most existing methods and software for boundary layers are designed for CAD models, so they are not well suited for complex geometries from medical imaging. The commercial software with which we compared was no exception. In order to compare the quality of the meshes generated by our method against commercial software, we resorted to a simpler geometry. We chose a simple T shaped intersection of tubes. After many unsuccessful trials, the commercial tool reported successful generation of prismatic elements using either of the two algorithms, which apparently marched about 35 to 45% of the GLFS. The left two panels in Figures~4.17 show the resulting meshes, which were visibly jagged and skewed. Such jagged boundary-layer meshes appear to be common for some other software as well. In this comparison, we set our method to march 50% of the GLFS for this geometry, and the last panel in Figures~4.17 shows the resulting mesh using our method. In contrast, the mesh from our method was visibly far smoother because of its variational nature.

Figure 4.18 shows the statistics of the prismatic layers using the commercial software as well as our method in terms of the triangle angle, edge distortion, and scaled aspect ratio. It is worth noting that the commercial software appears to deliver better triangle angles than ours. However, because that software does not

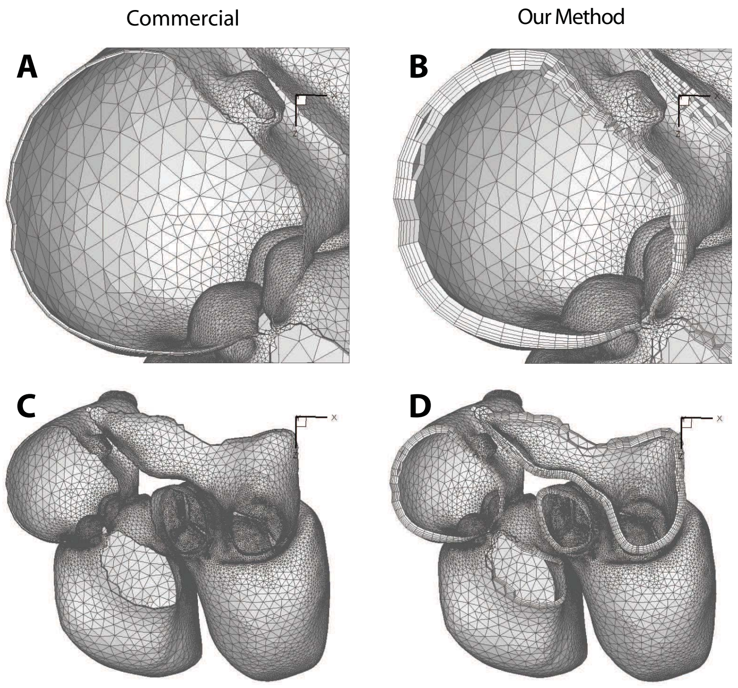


Figure 4.16: Comparison of prismatic boundary layer generated using a leading commercial CFD mesh generator and our method.

optimize the orthogonality of the prisms, it caused severe edge distortions and even inverted elements. Overall, the mesh from our method is superior in terms of the scaled aspect ratio. This result is consistent with our observations for the lung mesh. These results demonstrate that our method is more robust and can produce higher quality meshes than this (and likely some other) state-of-the-art, commercial software.

4.6 Variational Generation of Hybrid Mesh

Generating prisms prior to tetrahedralization of the interior quickly produces thick boundary layers with well shaped prisms, but it doesn't allow flexibility in choosing properties of terahedral core.

The reason is that after propagation, the domain is shrunk. Tetrahedral core has to be generated by a constrained tetrahedralization, which cannot add Steiner points on the surface. As a result, the quality of the resulting tetrahedral mesh is limited. To maintain the consistencies of prisms, the tetrahedral mesh generator has more constraints than usual, making it difficult to produce high-quality tetrahedra. To overcome this limitation, we propose an alternative strategy. The method has four

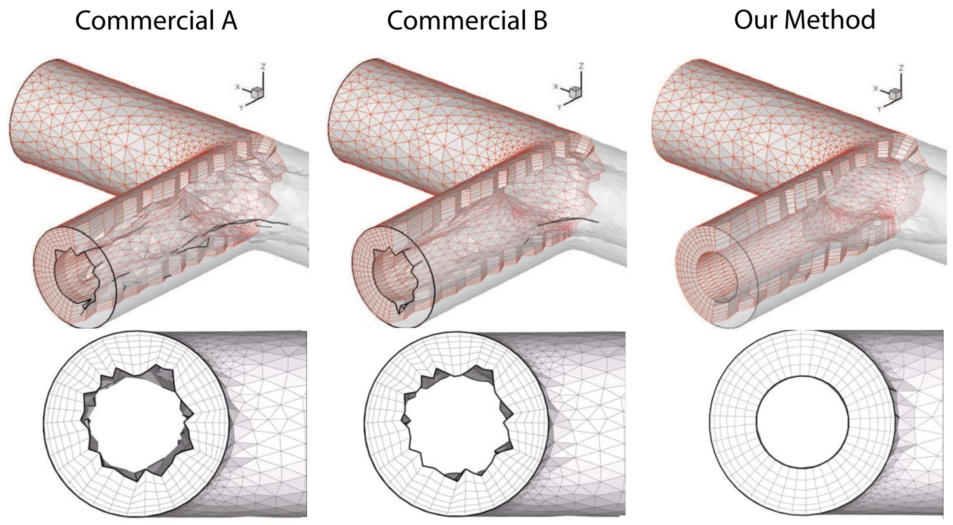


Figure 4.17: Comparison of mesh quality of prismatic boundary layer generated using commercial software and our method. Solid curves indicate ridge curves in the surface mesh with dihedral angles close to 90°.

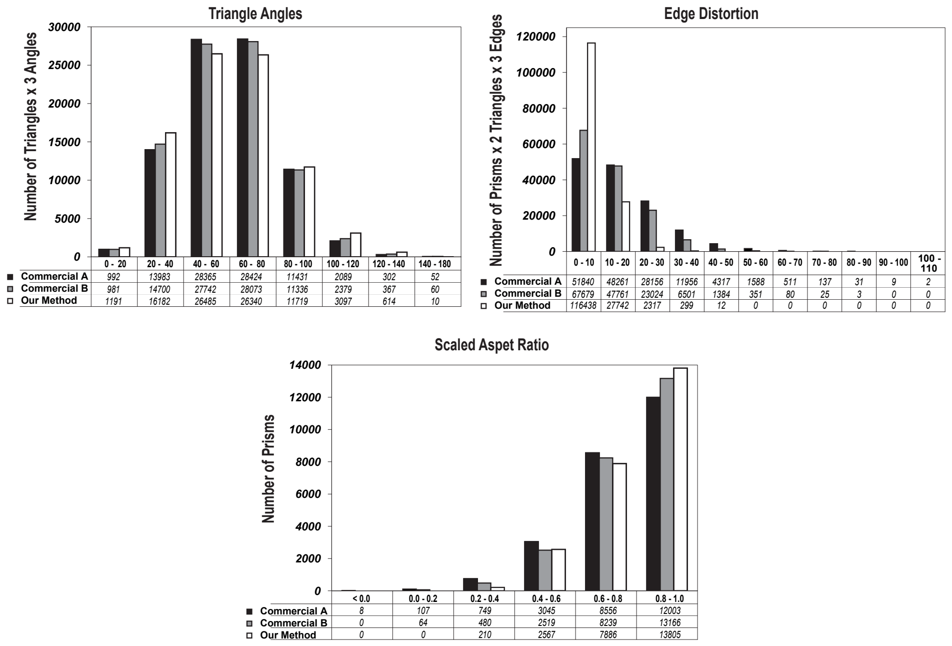
key steps:

1. generate an initial tetrahedral mesh;
2. propagate an initial thin-layer of prisms and shrink the tetrahedral core;
3. further expand prisms and shrink tetrahedra by variational smoothing;
4. enhance overall hybrid mesh using local mesh adaptivity.

We discuss each of these steps in the following subsections.

4.6.1 Initial Generation of Tetrahedra

In this section we describe our own technique for terahedralization of the surface, though our method can add prismatic boundary layer to any tetrahedral mesh. For our purpose, the boundary triangulation of the tetrahedral mesh should have a good mesh quality, as the prisms will be generated from it. The interior of the tetrahedral mesh has more flexibility: it may be anisotropic and/or layered, and it does not need to have perfect mesh quality since it will be deformed and enhanced later. However, in general it is desirable to avoid high-valence vertices and very small tetrahedra for better effectiveness and higher efficiency.



	0 - 20	20 - 40	40 - 60	60 - 80	80 - 100	100 - 120	120 - 140	140 - 180
■ Commercial A	992	13983	28365	28424	11431	2089	302	52
▨ Commercial B	981	14700	27742	28073	11336	2379	367	60
□ Our Method	1191	16182	26485	26340	11719	3097	614	10

	0 - 10	10 - 20	20 - 30	30 - 40	40 - 50	50 - 60	60 - 70	70 - 80	80 - 90	90 - 100	100 - 110
■ Commercial A	51840	48261	28156	11956	4317	1588	511	137	31	9	2
▨ Commercial B	67679	47761	23024	6501	1384	351	80	25	3	0	0
□ Our Method	116438	27742	2317	299	12	0	0	0	0	0	0

	< 0.0	0.0 - 0.2	0.2 - 0.4	0.4 - 0.6	0.6 - 0.8	0.8 - 1.0
■ Commercial A	8	107	749	3045	8556	12003
▨ Commercial B	0	64	480	2519	8239	13166
□ Our Method	0	0	210	2567	7886	13805

Figure 4.18: Comparison of mesh quality statistics of prismatic boundary layer generated using commercial software and our method.

In our implementation, we generate the tetrahedral mesh by first generating a surface mesh and also a point set within the volume, with a distribution in proportion to the GLFS. Our current approach differs from the algorithm detailed in [44], in several key ways. First, the entire algorithm is implemented in our Matlab-based numerical geometry library, NumGeom. Second, we use TetGen [69] interoperably linked to NumGeom as our Delaunay connection engine. Importantly, since TetGen performs semi-constrained Delaunay tetrahedralization, inclusion of the exterior feature size (F_{out} in [44]) in the GLFS and subsequent adaption of the surface mesh which includes this exterior value is no longer necessary. It is important to note that we simply ask TetGen to connect the points. We put no requirements on quality, dihedral angle or volume. Since our point placement is very deliberate, we wish to improve the quality of the initial tetrahedral mesh in a way that is inherently compatible with that placement. Secondly, we adapt the outlet surface triangulation with Triangle [67], similarly interoperably linked to NumGeom.

Algorithm 6 GENERATELAYEREDTETS

```
1: [Given a closed triangulated surface $S$ with planar boundary outlets $S_{fac}^{1...n}$, generate a quality, scale-invariant layered tetrahedral grid]
2: Inputs: minimum and maximum feature size limits $L_{\min}$ and $L_{\max}$, a collision tolerance $\Phi$, a curvature parameter $\gamma$, and the desired number of layers $nlayers$
3: $\alpha = {}^{nlayers}/_{nlayers+1}$
4: Compute vertex normals and curvatures [33]
5: Compute GLFS [44? ] to return $F[\mathbf{x}]$ the raw ray shoot, and $f[\mathbf{x}]$ its gradient-limited value, where here $F[\mathbf{x}]$ and $f[\mathbf{x}]$ are considered radii not diameters
6: for each boundary facet $S_{fac}^{1...n}$
7:     delete facet triangles
8:     for each point $\mathbf{x}_{fac}$ on the border
9:         project vertex normals $\mathbf{n}_{fac}$ onto the facet
10:        for j $\leftarrow 1 \ldots nlayers$
11:            $dist = \alpha f[\mathbf{x}_{fac}] \cdot j\ nlayers$
12:            generate points on the interior of the facet $\mathbf{x}_{new} \leftarrow \mathbf{x}_{fac} + dist \cdot \mathbf{n}_{fac}$; $new++$
13:    create a kdtree structure of all of the new points
14:    sort new points in order of descending $SR = {}^{f[\mathbf{x}]}/_{F[\mathbf{x}[}$
15:    for each new point $x_i^{new}$ from sorted list
16:        if $\neg\, dudded[x_i^{new}]$
17:            use the kdtree to find points, $\mathbf{x}_{found}$, in $\mathbf{x}_{new}$within $\Phi \cdot {}^{f[\mathbf{x}_i^{new}]}/_{nlayers}$
18:            $dudded[\mathbf{x}_{found}] \leftarrow TRUE$
19:    call TRIANGLE [67] to connect the surviving new points and the contour
20:    add the new triangulated facet to $S$
21: for each point $\mathbf{x_i} \in S$ that is not a facet point
22:    for j $\leftarrow 1 \ldots nlayers$
23:        $dist = \alpha f[\mathbf{x}_i] \cdot j\ nlayers$
24:        generate points on the interior of $S$ $\mathbf{x}_{new} \leftarrow \mathbf{x}_i + dist \cdot \mathbf{n}_{fac}$; $new++$
25:    create a kdtree structure of all of the new points
26:    sort new points in order of descending $SR = {}^{f[\mathbf{x}]}/_{F[\mathbf{x}[}$
27:    for each new point $x_i^{new}$ from sorted list
28:        if $\neg\, dudded[x_i^{new}]$
29:            use the kdtree to find points, $\mathbf{x}_{found}$, in $\mathbf{x}_{new}$within $\Phi \cdot {}^{f[\mathbf{x}_i^{new}]}/_{nlayers}$
30:            $dudded[\mathbf{x}_{found}] \leftarrow TRUE$
31: call TETGEN [69] to connect the surviving new points and $S$ into tetrahedral mesh $\mathcal{T}$
32: map the boundary conditions to $\mathcal{T}$ [28]
```

Algorithm 7 IMPROVETETMESH

```
1: [Given a tetrahedral mesh $\mathcal{T}$ generated with Algorithm 6, apply edge flipping
   and r-adaption to improve the mesh quality]
2: while $RMS^{\mathcal{T}} > tol$
3:     while $nflips > 0$
4:         perform a sweep of 3-2, 2-3, 2-2 and 4-4 flips to decrease the error
5:     perform $n$ iterations of variational smoothing
6:     $RMS^{\mathcal{T}} \leftarrow \left( \frac{1}{N} \sum_{i=1}^{N} \|\boldsymbol{d}_v\|^2 \right)^{1/2}$, where $\boldsymbol{d}_v$ is the change in the position of
   the node during
```

There are also differences in terms of the distribution of points on the interior. In particular, we compute a shoot ratio ($SR[\mathbf{x}]$) which is the ratio of the GLFS ($f[x]$) to the length of the ray ($F[\mathbf{x}]$). It should be noted that $SR[\mathbf{x}]$ is a measure of our confidence that the gradient limited radius is actually on the medial axis. For example, $SR[x_i] \approx 1$ implies that the GLFS at x_i underwent very little gradient limiting. Given this measure of confidence we sort all of the candidate interior (and outlet) points by $SR[\mathbf{x}]$ in descending order and use this sorted list within a kdree [39] to delete points that are within a fraction of the point offset times a collision parameter Φ. This is a natural way to remove troublesome points that would wreck the layer structure and create small badly shaped elements. The details of the overall point generation and connection method are given in Algorithm 6. Generation of this initial layered tetrahedral mesh is followed by a simple but effective tetrahedral improvement strategy that pairs well with our tetrahedral mesh generation strategy. It consists of alternating topological flip operations with nodal smoothing (Algorithm 7). For both operations, we define an energy for each tetrahedron with respect to some "ideal" reference tetrahedron, similar to that presented in [32]. We define this energy in detail below since it will be applied during the prism propagation and hybrid mesh improvement stages, as well.

Variational Tetrahedral Energy

Given a tetrahedron $\tau = \boldsymbol{x}_0\boldsymbol{x}_1\boldsymbol{x}_2\boldsymbol{x}_3$, and its corresponding reference tetrahedron, $\boldsymbol{p}_0\boldsymbol{p}_1\boldsymbol{p}_2\boldsymbol{p}_3$, let $\boldsymbol{J}_1$ and $\boldsymbol{J}_2$ denote their respective Jacobians with respect to the standard master element, i.e., $\boldsymbol{J}_1 = [\boldsymbol{x}_1 - \boldsymbol{x}_0 | \boldsymbol{x}_2 - \boldsymbol{x}_0 | \boldsymbol{x}_3 - \boldsymbol{x}_0]$, and $\boldsymbol{J}_1 = [\boldsymbol{p}_1 - \boldsymbol{p}_0 | \boldsymbol{p}_2 - \boldsymbol{p}_0 | \boldsymbol{p}_3 - \boldsymbol{p}_0]$. We define

$$\boldsymbol{G}_1 = \boldsymbol{J}_1^T \boldsymbol{J}_1 = \begin{bmatrix} g_{11} & g_{12} & g_{13} \\ g_{21} & g_{22} & g_{23} \\ g_{31} & g_{32} & g_{33} \end{bmatrix} \text{ and } \boldsymbol{G}_2 = \boldsymbol{J}_2^T \boldsymbol{J}_2 = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}, \tag{4.6.1}$$

and let $\boldsymbol{g}_j$ and $\boldsymbol{h}_j$ denote the jth columns of $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$, respectively. Let $\nu = \sqrt{|\boldsymbol{G}_1|}$ and $V = \sqrt{|\boldsymbol{G}_2|}$. As illustrated in Figure 4.19, we label the edges of the tetrahedra as $\boldsymbol{l}_i$ and $\boldsymbol{r}_i$ for $i = 1, 2, 3$, where $\boldsymbol{l}_i = \boldsymbol{x}_i - \boldsymbol{x}_0$ and $\boldsymbol{r}_i = \boldsymbol{x}_{i-} - \boldsymbol{x}_{i+}$, with

$$i+ = \begin{cases} i+1 & i < 3 \\ 1 & i = 3 \end{cases} \text{ and } i- = \begin{cases} i-1 & i > 1 \\ 3 & i = 1 \end{cases}. \tag{4.6.2}$$

Let $\boldsymbol{n}_0 = \boldsymbol{r}_2 \times \boldsymbol{r}_1$, the normal to the opposite triangle of $\boldsymbol{x}_0$ pointing toward $\boldsymbol{x}_0$ with magnitude equal to twice of the triangle area. For any $\boldsymbol{a} \in \mathbb{R}^3$, let sum$(\boldsymbol{a})$ denote $a_1 + a_2 + a_3$. The energy $E_\theta(\tau)$ for a tetrahedron τ in $\mathbb{R}^3$ is

$$E_\theta = \frac{1}{V^{2/3}} \sum_{i=1}^{3} \left(\alpha_i \|\boldsymbol{l}_i\|^2 + \beta_i \|\boldsymbol{r}_i\|^2 \right), \tag{4.6.3}$$

where $\alpha_i = \text{sum}(\boldsymbol{g}_{i+} \times \boldsymbol{g}_{i-})/\nu^{4/3}$ and $\beta_i = (g_{ii} g_{i-,i+} - g_{i,i-} g_{i,i+})/\nu^{4/3}$. If the ideal tetrahedra are regular, then $\alpha_i = \beta_i = 2^{-4/3}$.

Nodal Smoothing of Tetrahedral Mesh

Smoothing sweep functional is the error defined in Equation 4.6.3. In this application, the reference element is a regular tetrahedron, and thus again $\alpha_i = \beta_i = 2^{-4/3}$, as defined above. During smoothing the neighborhood on which the error is defined is the polyhedron of elements surrounding each node. Nodal smoothing is subject to the constrained degrees of freedom of the point. Boundary surface nodes that are not on the interior of a plane are currently excluded from the sweep.

We add up the energy of all the tetrahedra and then try to minimize the total energy by moving vertices. In our optimization algorithm, we compute the gradient and Hessian of the energy with respect to vertex coordinates. For the elemental energy E_θ, the gradient $\nabla_{\boldsymbol{x}_0} E_\theta$ and Hessian $\nabla^2_{\boldsymbol{x}_0} E_\theta$ with respect to $\boldsymbol{x}_0$ are

$$\nabla_{\boldsymbol{x}_0} E_\theta = -\frac{2}{V^{2/3}} \left(\sum_{i=1}^{3} \alpha_i \boldsymbol{l}_i \right) - \frac{2}{3V} E_\theta \boldsymbol{n}_0 \tag{4.6.4}$$

$$\nabla^2_{\boldsymbol{x}_0} E_\theta = \left(\frac{2}{V^{2/3}} \sum_{i=1}^{3} \alpha_i \right) \boldsymbol{I} - \frac{2}{3V} \boldsymbol{B} - \frac{2}{9V^2} E_\theta \boldsymbol{n}_0 \boldsymbol{n}_0^T, \tag{4.6.5}$$

where $\boldsymbol{I}$ denotes the 3×3 identity matrix and $\boldsymbol{B} = \left(\nabla_{\boldsymbol{x}_0} E_\theta\right) \boldsymbol{n}_0^T + \boldsymbol{n}_0 \left(\nabla_{\boldsymbol{x}_0} E_\theta\right)^T$.

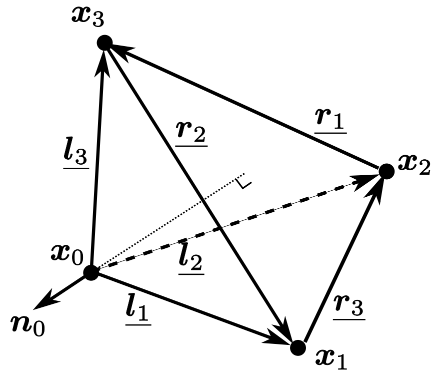


Figure 4.19: Naming convention for tetrahedron. Underlined symbols indicate edges.

Given Equations 4.6.3, 4.6.4 and 4.6.5, we apply Newton's method to determine a displacement $\boldsymbol{d}_v$ for the vertex, i.e., by solving

$$\left(\nabla^2_{\boldsymbol{x}_v} E_\theta\right) \boldsymbol{d}_v = -\nabla_{\boldsymbol{x}_v} E_\theta. \tag{4.6.6}$$

In the case of constrained nodes with 1 or 2 degrees of freedom, we solve a reduced system that is either along an edge or in the plane of a planar boundary (typically an outlet). Solution of Equation 4.6.6 or the reduced system leads to proposed nodal displacement $\boldsymbol{d}_v$ that is scaled back - if necessary - to avoid folding; if any vertex is scaled back, we recompute the gradient and Hessian of its surrounding elements and compute its displacement. Smoothing based upon the minimization of E_θ has a very favorable characteristic. The tetrahedra do not invert under adaptive smoothing, owing to the presence of the tetrahedral volume in the denominator.

4.6.2 Initial Generation of Prisms and Shrinking Tetrahedra

We add a layer of prisms outside the tetrahedral mesh to set up the topology of the prismatic-tetrahedral hybrid mesh. This is done by 1) propagating the surface mesh to generate a thin layer of prisms, and 2) shrinking the tetrahedral mesh to make space for the prisms.

Initial Propagation

We generate an initial prismatic mesh using a simplified version of our previous algorithm. We retain the core ideas of 1) using GLFS to control the depth of the prisms, 2) using face offsetting to propagate the surfaces, and 3) applying variational smoothing and step size control to ensure high-quality prisms and positive Jacobians. This combination has been demonstrated to be robust and efficient. However, because our goal is to just produce a thin layer of prisms, we omit some optimizations in the original algorithms in favor of faster execution, as we describe in more detail below.

Overall, the initial propagation proceeds as follows:

1. compute vertex displacements for Δt using face offsetting with GLFS as speed function;

2. check positivity of Jacobian of the prisms and reduce Δt to prevent mesh folding;

3. apply variational smoothing to optimize triangle shapes and side-edge orthogonality;

4. if it has propagated sufficiently (say 50% of the desired depth of the prisms) or the time step was reduced to some too small value, then stop; otherwise return to step 1.

Shrinking Tetrahedra

After a thin boundary layer has been generated, we deform the tetrahedral mesh using a variational smoothing procedure:

1. add displacements computed from the initial propagation to boundary vertices; if the displacements cause tetrahedral elements to fold, scale back the displacements;

2. smooth the interior tetrahedral with fixed boundary;

3. if displacements were scaled back, go back to step 1 and repeat step 1 and 2, until all displacements have been applied or the displacements are scaled back to nearly zero.

Because the process does not change the connectivity of the mesh, it is possible that the procedure stops before all the propagated distances of the prismatic layer are transferred to the tetrahedra. This is not a problem, as our goal is simply to generate a thin layer of prisms with consistent tetrahedra.

We smooth the tetrahedra using a modified algorithm of that in [32]. In particular, we define an energy for each tetrahedron with respect to some "ideal" reference tetrahedron, according to the energy defined in Equations 4.6.3, 4.6.4 and 4.6.5. Since the input mesh may be graded or layered, we try to preserve the shapes of the initial tetrahedra, we use the elements in the input mesh as references for smoothing. Thus, the shape parameters $\alpha_i = \text{sum}(\boldsymbol{g}_{i+} \times \boldsymbol{g}_{i-})/\nu^{4/3}$ and $\beta_i = (g_{ii}g_{i-,i+} - g_{i,i-}g_{i,i+})/\nu^{4/3}$ are computed directly from the reference mesh. This is reasonable since if the mesh does not deform, this choice of reference elements would exactly preserve the character of the original tetrahedral mesh. However, there are two potential issues about using the input elements as references. First, since the tetrahedra are shrunk during the process, the angles cannot be reconstructed exactly. However, in our experiments the tets tend to shrink nearly uniformally, and hence the layers are preserved reasonably well.

When using the input tetrahedra as references, the quality of the resulting mesh may suffer if there are slivers in the input mesh. To address this issue, we regularize the energy by adding a component with respect to the regular elements. In particular, let

$$c = \frac{1}{1 + \exp(-\rho(\theta - \frac{5}{6}\pi))}, \tag{4.6.7}$$

where θ denote the largest dihedral angle of the reference element, and ρ is a controllable parameter. Then we set

$$\tilde{\alpha}_i = (1 - c)\alpha_i + c2^{-4/3}. \tag{4.6.8}$$

It is obvious that $c \approx 1$ for $\theta \gg 150^\circ$ and $c \approx 0$ for $\theta \ll 150^\circ$, so that $\tilde{\alpha}_i$ is dominated by the regular elements if the input element is close to slivers, and is dominated by the input elements otherwise. We chose ρ to be 40 based on empirical evidence.

4.6.3 Expanding Prisms

The previous step generated and smoothed prisms and tetrahedra independently of each other. During the process, if some tetrahedra have very large aspect ratios, the propagation of the prismatic boundary layers would terminate prematurely. We address the issue in this step by smoothing the prisms and tetrahedra simultaneously. Our goals are twofold: 1) to propagate the prisms further while continuing to shrink the tetrahedra, and 2) to allow local adaptation of prism heights based on the overall mesh quality. We assume that the input surface triangulation has a reasonable quality, and we do not move the vertices on the boundary, except for those on the outlet walls, which must slide along the walls.

Definition of Energy

To couple prism expansion with tetrahedral core compression, we define energy for prisms based on tetrahedral energy formulation. Given a triangle σ on the propagating boundary, let $\hat{\boldsymbol{n}}_\sigma$ be the unit normal toward the direction of propagation, and d_σ be the average of the desired propagated distance of its three vertices. We define a target prism for the given triangle to have points $\boldsymbol{v}_i$, where

$$\boldsymbol{v}_i = \begin{cases} \boldsymbol{x}_i & i = 1,2,3 \\ \boldsymbol{x}_{i-3} + d_\sigma \hat{\boldsymbol{n}}_\sigma & i = 4,5,6 \end{cases}. \tag{4.6.9}$$

To define the energy for a prism, for simplicity we sample a tetrahedron at each vertex of the prism, and use the corresponding vertices in the actual and target prisms as the actual and reference tetrahedra. For example, at vertex 1 the actual tetrahedron is $\boldsymbol{x}_1\boldsymbol{x}_2\boldsymbol{x}_3\boldsymbol{x}_4$ and the reference tetrahedron is $\boldsymbol{v}_1\boldsymbol{v}_2\boldsymbol{v}_3\boldsymbol{v}_4$. At vertex 4, the actual tetrahedron is $\boldsymbol{x}_4\boldsymbol{x}_5\boldsymbol{x}_6\boldsymbol{x}_1$ and the reference tetrahedron is $\boldsymbol{v}_4\boldsymbol{v}_5\boldsymbol{v}_6\boldsymbol{v}_1$, as shown in Figure 4.20. The constructions for other vertices are similar. For each pair of actual and reference tetrahedra, we substitute them into 4.6.3 to compute its energy, and the energy for the prism is the sum of the energy of the six tetrahedra. We refer to this energy as the *height energy* of the prism, , denoted by E_h, as its main objective is to control the height of the prisms.

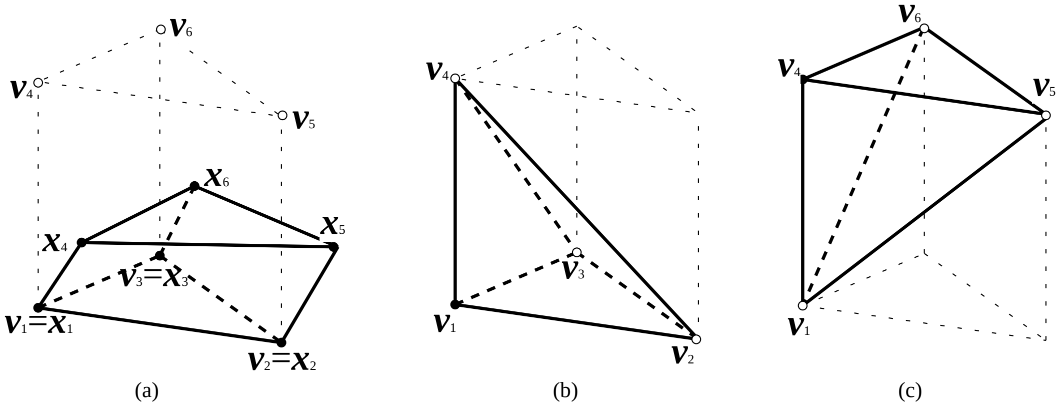


Figure 4.20: Illustration of construction of reference tetrahedra for vertices 1 and 4 of a prism. (a) the actual (in dark edges) and ideal prism. (b) Reference tetrahedron (in dark edges) at vertex 1. (c) Reference tetrahedra at vertex 4.

In addition to the height energy, we also would like to optimize the orthogonality of the side edges of prisms. Note that the height energy also penalizes edges that are far from orthogonal, but it is not strong enough to enforce orthogonality. We use the same energy defined in 4.3.6.

Given these elemental energy, our overall prism expansion algorithm computes the total energy E as a weighted sum of the height energy E_h of the prisms, the orthogonality energy $E_\perp$, and the shape energy E_t of the interior tetrahedra, i.e.,

$$E = w_t \sum E_t + \sum \left(w_h E_h + w_\perp E_\perp \right), \tag{4.6.10}$$

w denotes the corresponding weights. We set the weights to $w_h = 4$ and $w_\perp = w_t = 1$, where the higher weights for E_h allows more substantial growth of the prisms. We minimize this energy similarly as for tetrahedra, as summarized by the following procedure:

1. Loop through the interior tetrahedra, and for each tetrahedron compute the gradient and Hessian of E_t with respect to its vertex coordinates, and add them to the corresponding vertex values;
2. Loop through the prisms, and for each prism compute the gradient and Hessian of E_h and $E_\perp$ with respect to its vertex coordinates, and add them to the corresponding vertex values;
3. Loop through all vertices to divide the gradient by the Hessian to obtain a displacement; for vertices on the outlet walls, constrain the displacement within the wall or along the ridge;
4. Scale back displacements to avoid folding; if any vertex is scaled back, recompute the gradient and Hessian of its surrounding elements and compute its displacement;
5. Repeat steps 1 through 4 for a few times.

Remarks Note that unlike the energy for interior tetrahedra, we do not blend the objective function for regular tetrahedra, because it may complicate the control of the height of the prisms. In addition, it is not necessary because by construction slivers are not possible, and very poorly shaped triangles are prevented by the energy of the interior tetrahedra.

4.7 Results

We consider the qualities of the generated prisms and tetrahedra. To measure the quality of prisms, we use the *scaled aspect ratio* defined via Jacobian, deviation of prisms edges from normals to bases and angles of surfaces triangles.

To measure the quality of tetrahedra, we use the *aspect ratio* quality measure, defined as

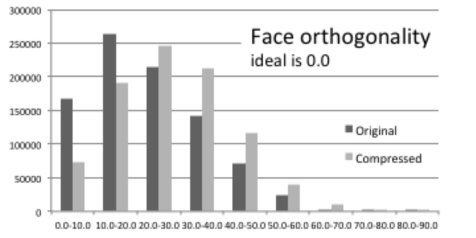
$$\chi \equiv 2\sqrt{6}\frac{\text{inscribed radius}}{\text{length of longest edge}} \tag{4.7.1}$$

which has a maximum value of 1.0 for regular tetrahedra.

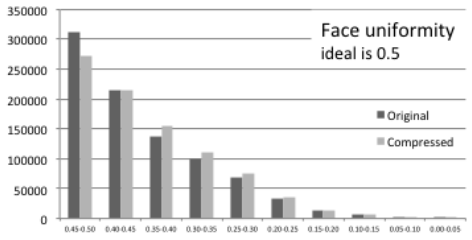
In addition to the element based quality metrics above, we report the face-based quality statistics defined in [34] that are important for finite volume codes: *face-orthogonality*, *face-skewness* and *face-uniformity*. Each of these metrics relates the ray, *d*, between the centroid of a polyhedra and that of its face-neighbor to the shared face between them. *Face-orthogonality* (α) is the angle between *d* and the face normal, with an *face-orthogonality* of 0.0 being optimal. *Face-skewness* (ψ) is the distance between the intersection of *d* with the plane of the face and the face centroid, normalized by the length of the *d*. A *face-skewness* of 0.0 is optimal. Lastly, *face-uniformity* (f_x) is the ratio of the distance between the element centroid and the plane of their common face with *d*. A *face-uniformity* of 0.5 means that the two neighbors are roughly the same size and is optimal. Details of these face-based statistics can be found in [34]. All of our experiments were conducted on a PC with a 3.16GHz Intel® Core-2 Duo E8500 CPU and 4GB of RAM, running 64-bit Linux 2.6.31 with gcc 4.4.1.

4.7.1 Rat Lung

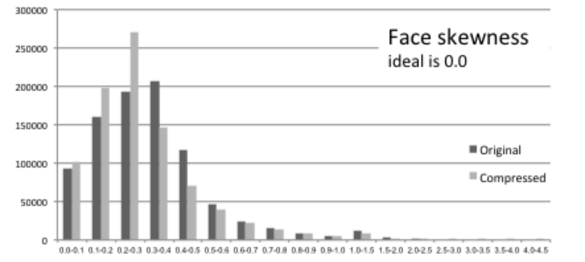
We evaluate new approach on the same surface mesh as in 4.5.2. Based on this surface we generate four volume meshes. First, we evaluate how good the method is in preserving the quality of the tetrahedral mesh it compresses. To compare the quality of tetrahedra before and after compressing we let TetGen produce unconstrained Delaunay tetrahedralization of the surface. Then we proceed to generate boundary layer on top of the core with the height of about 40% of GLFS. Comparison (Figure 4.21) shows that quality deterioration is acceptable, and is mostly pronounced in the face orthogonality.



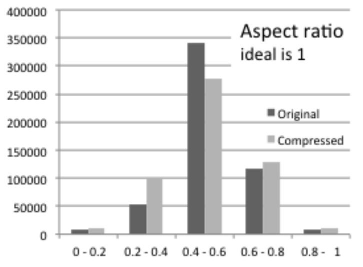
	0.0-10.0	10.0-20.0	20.0-30.0	30.0-40.0	40.0-50.0	50.0-60.0	60.0-70.0	70.0-80.0	80.0-90.0
Original	168287	264202	215009	142441	72188	23988	3277	198	1
Compressed	72327	191699	245385	213009	116319	39727	9537	1512	76



	0.45-0.50	0.40-0.45	0.35-0.40	0.30-0.35	0.25-0.30	0.20-0.25	0.15-0.20	0.10-0.15	0.05-0.10	0.00-0.05
Original	312176	215478	138112	100291	68406	32888	12677	7028	2504	30
Compressed	273166	214956	154312	111598	74447	36120	14415	7389	2890	298



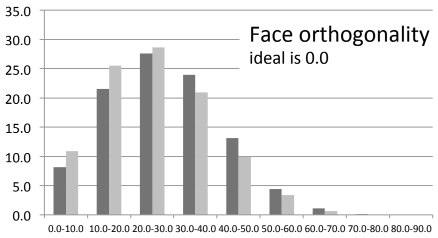
	0.0-0.1	0.1-0.2	0.2-0.3	0.3-0.4	0.4-0.5	0.5-0.6	0.6-0.7	0.7-0.8	0.8-0.9	0.9-1.0	1.0-1.5	1.5-2.0	2.0-2.5	2.5-3.0	3.0-3.5	3.5-4.0	4.0-4.5
Original	93898	160791	193642	206367	117093	47440	24308	15334	9435	5449	12819	2874	141	0	0	0	0
Compressed	101249	198879	270754	147101	71326	39568	23193	13697	8472	5192	8347	1432	270	74	16	5	5



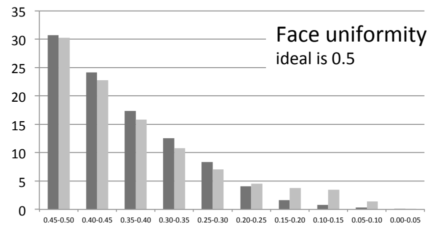
	0 - 0.2	0.2 - 0.4	0.4 - 0.6	0.6 - 0.8	0.8 - 1
Original	7155	53973	340265	115818	9182
Compressed	11450	100735	276942	127554	9712

Figure 4.21: Quality of tetrahedral mesh before and after compressing by 40% of GLFS. Number of tetrahedra : 526393.

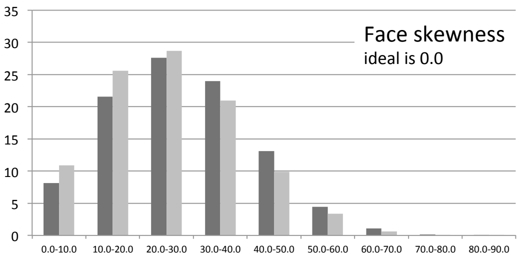
Next we compare the quality of compressed tetrahedral core with the quality of constrained Delaunay tetrahedralization of the inner boundary layer surface (Figure 4.22). This is somewhat different from our original approach, as we allow TetGen to add Steiner points to original surface, then generate boundary layer, then invoke TetGen again for constrained tetrahedralization. In this case both meshes have same boundary layer, so we only compare quality of tetrahedra. Results show that both approaches generate tetrahedral meshes of comparable quality, though compressing the tetrahedral mesh is much more computationally expensive.



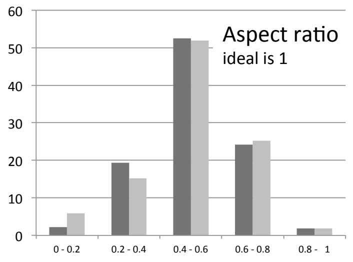
	00.0-10.0	10.0-20.0	20.0-30.0	30.0-40.0	40.0-50.0	50.0-60.0	60.0-70.0	70.0-80.0	80.0-90.0
Compressed	8.1	22	28	24	13	4.47	1.07	0.17	0.009
Constrained	10.9	26	29	21	10	3.40	0.63	0.02	0.011



Uniformity	0.45-0.50	0.40-0.45	0.35-0.40	0.30-0.35	0.25-0.30	0.20-0.25	0.15-0.20	0.10-0.15	0.05-0.10	0.00-0.05
Compressed	30.71	24.16	17.35	12.54	8.37	4.06	1.62	0.83	0.32	0.03
Constrained	30.24	22.79	15.84	10.83	7.03	4.57	3.76	3.50	1.42	0.03



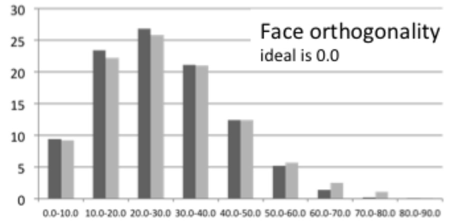
	0.0-0.1	0.1-0.2	0.2-0.3	0.3-0.4	0.4-0.5	0.5-0.6	0.6-0.7	0.7-0.8	0.8-0.9	0.9-1.0	1.0-1.5	1.5-2.0	2.0-2.5	2.5-3.0	3.0-3.5	3.5-4.0	4.0-4.5
CMP	11	22	30	17	8	4.4	2.6	1.5	1.0	0.584	0.938	0.161	0.030	0.008	0.002	0.001	0.001
CNS	9	18	23	20	13	7.2	4.2	2.4	1.4	0.727	0.818	0.087	0.005	0.000	0.000	0.000	0.000



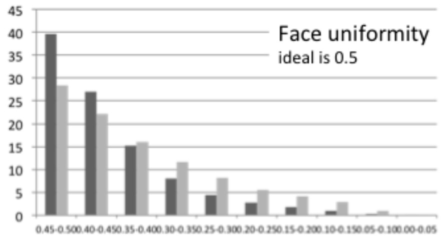
Aspect Ratios	0 - 0.2	0.2 - 0.4	0.4 - 0.6	0.6 - 0.8	0.8 - 1
Compressed	2	19	52	24	2
Constrained	6	15	52	25	2

Figure 4.22: Quality (percentage points) of tetrahedral core compressed by 40% of GLFS, dark columns correspond to compressed core, light to tetrahedra generated by tetgen based on inner surface of boundary layer. Total number of tetrahedra in compressed core was 526393, in constrained tetrahidralization by TetGen - 623473.

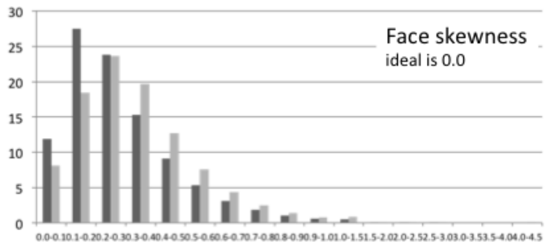
The case that shows clear advantage of modified approach is when boundary layer is introduced on top of terahedral mesh generated as described in 4.6.1. We compare it with mesh generated by **Algorithm 5** . Both meshes have five layers of prisms with total height of about 25% of GLFS. Layered mesh has 5926260 tetrahedra, about ten times more then constrained Delaunay tetrahedralization of the surface by TetGen, which has 585104 tetrahedra.



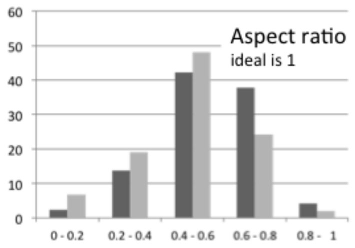
	00.0-10.0	10.0-20.0	20.0-30.0	30.0-40.0	40.0-50.0	50.0-60.0	60.0-70.0	70.0-80.0	80.0-90.0
Layered	9.43	23.43	26.79	21.15	12.45	5.19	1.39	0.18	0.00
Constrained	9.22	22.22	25.80	21.00	12.37	5.75	2.45	1.12	0.06



	0.45-0.50	0.40-0.45	0.35-0.40	0.30-0.35	0.25-0.30	0.20-0.25	0.15-0.20	0.10-0.15	0.05-0.10	0.00-0.05
Layered	39.61	27.06	15.16	8.03	4.43	2.73	1.77	0.92	0.26	0.02
Constrained	28.32	22.21	16.09	11.66	8.11	5.58	4.13	2.90	0.98	0.03



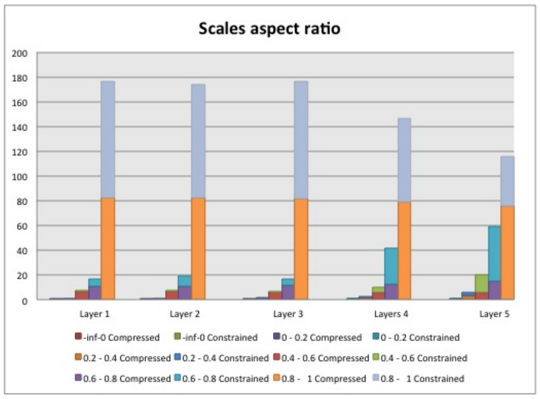
	0.0-0.1	0.1-0.2	0.2-0.3	0.3-0.4	0.4-0.5	0.5-0.6	0.6-0.7	0.7-0.8	0.8-0.9	0.9-1.0	1.0-1.5	1.5-2.0	2.0-2.5	2.5-3.0	3.0-3.5	3.5-4.0	4.0-4.5
Layered	11.88	27.51	23.82	15.34	9.13	5.33	3.11	1.80	1.01	0.54	0.52	0.01	0.00	0.00	0.00	0.00	0.00
Constrained	8.14	18.47	23.62	19.69	12.66	7.54	4.35	2.49	1.39	0.73	0.85	0.07	0.00	0.00	0.00	0.00	0.00

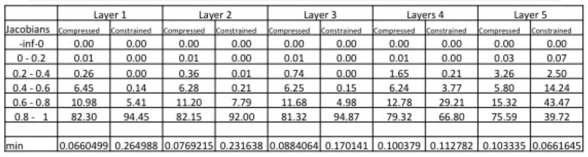


Aspect Ratios	0 - 0.2	0.2 - 0.4	0.4 - 0.6	0.6 - 0.8	0.8 - 1
Layered	2.29	13.65	42.16	37.76	4.13
Constrained	6.75	19.08	48.00	24.21	1.96

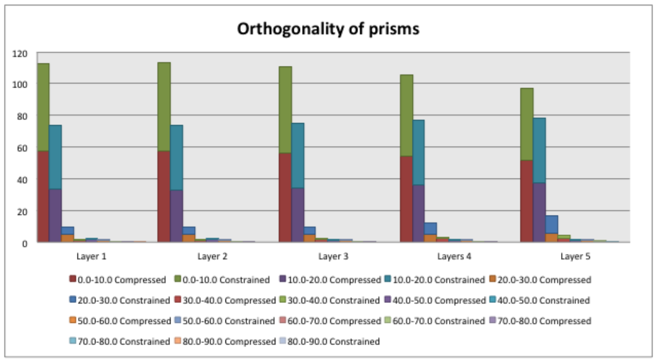
Figure 4.23: Comparison between quality of compressed layered and constrained Delaunay tetrahedral cores. Dark columns correspond to compressed mesh, light to constrained Delaunay. Values in tables are percentage points.

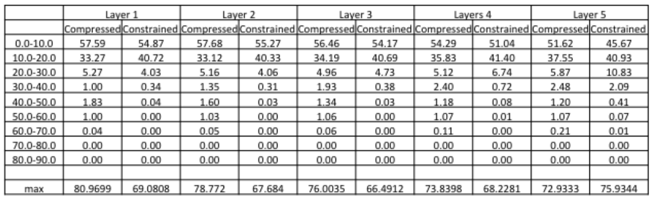
Quality of the compressed layered tetrahedral core is far superior, especially considering it has ten times more elements. We also show quality of the boundary layer prisms for both cases.





	Layer 1		Layer 2		Layer 3		Layers 4		Layer 5	
Jacobians	Compressed	Constrained	Compressed	Constrained	Compressed	Constrained	Compressed	Constrained	Compressed	Constrained
-inf-0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0 - 0.2	0.01	0.00	0.01	0.00	0.01	0.00	0.01	0.00	0.03	0.07
0.2 - 0.4	0.26	0.00	0.36	0.01	0.74	0.00	1.65	0.21	3.26	2.50
0.4 - 0.6	6.45	0.14	6.28	0.21	6.25	0.15	6.24	3.77	5.80	14.24
0.6 - 0.8	10.98	5.41	11.20	7.79	11.68	4.98	12.78	29.21	15.32	43.47
0.8 - 1	82.30	94.45	82.15	92.00	81.32	94.87	79.32	66.80	75.59	39.72
min	0.0660499	0.264988	0.0769215	0.231638	0.0884064	0.170141	0.100379	0.112782	0.103335	0.0661645





	Layer 1		Layer 2		Layer 3		Layers 4		Layer 5	
	Compressed	Constrained	Compressed	Constrained	Compressed	Constrained	Compressed	Constrained	Compressed	Constrained
0.0-10.0	57.59	54.87	57.68	55.27	56.46	54.17	54.29	51.04	51.62	45.67
10.0-20.0	33.27	40.72	33.12	40.33	34.19	40.69	35.83	41.40	37.55	40.93
20.0-30.0	5.27	4.03	5.16	4.06	4.96	4.73	5.12	6.74	5.87	10.83
30.0-40.0	1.00	0.34	1.35	0.31	1.93	0.38	2.40	0.72	2.48	2.09
40.0-50.0	1.83	0.04	1.60	0.03	1.34	0.03	1.18	0.08	1.20	0.41
50.0-60.0	1.00	0.00	1.03	0.00	1.06	0.00	1.07	0.01	1.07	0.07
60.0-70.0	0.04	0.00	0.05	0.00	0.06	0.00	0.11	0.00	0.21	0.01
70.0-80.0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
80.0-90.0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
max	80.9699	69.0808	78.772	67.684	76.0035	66.4912	73.8398	68.2281	72.9333	75.9344

Figure 4.24: Comparison between quality of prismatic boundary layers.

So modified method applied to layered tetrahedral mesh produces volume mesh that has higher quality of both boundary layer and interior.

4.8 Conclusions and Discussions

We introduced a new method for generating prismatic boundary-layer meshes for complex biological geometries. Our method uses a gradient-limited feature size (GLFS) to guide the generation of the prismatic mesh at the boundary layers and the tetrahedral mesh in the interior. Unlike previous approaches, which propagate vertices along some vertex normals, our algorithm uses the face-offsetting method (FOM) to advance the surfaces. In addition, we introduced a novel prismatic variational smoothing procedure to improve bases triangle shapes and edge orthogonality. Our experimental study showed that the face-offsetting method is more robust than traditional methods based on vertex propagation, and our variational smoothing further enhances the robustness substantially and enables high-quality prismatic meshes. In addition, our experimentations also demonstrated that it is in general more robust to propagate the surface and then interpolate the intermediate layers than propagating the surface layer by layer. Combined with a robust, high-quality tetrahedral mesh generation, our overall framework produces high-quality hybrid meshes that adapt to the local scales of the geometry. Moreover, we introduced method to add thick high-quality prismatic boundary layer to any tetrahedral mesh. We demonstrated our method for two complex geometries, including a rat lung and an idealized human heart. We have conducted preliminary numerical simulations using our hybrid meshes and demonstrated that our method is effective and promising for biomedical computations. The output of our method can be also used to generate high-quality polyhedral meshes.

Chapter 5

Conclusions

We introduced a series of new methods for automatic processing and generation of hybrid prismatic-tetrahedral meshes for complex biological geometries. Each of the methods has value and applications by itself, and put together they provide a sequence of steps for generating high quality computational hybrid mesh for numerical simulation of fluid dynamics from radiological data.

Starting with CT scan of the chest of a rat injected with silicon solution, our method identifies airways in the image. Our approach is based on a combination of Gradient Vector Flow field analysis and Random Walker algorithm. We introduced novel scheme for computation of weights for Random Walker and proposed a way to state a subproblem which can be effectively solved with this Random Walker modification. We pre-select subset of the image, corresponding to union of airways and surrounding rib cage by applying region growing method to both image and meta-image of divergence of normalized Gradient Vector Flow Field. We continue work on this step, trying to replace region growing method with Power Watersheds approach combined with analysis of Gradient Vector flow field.

A triangulated surface extracted out of a volumetric voxel mesh originated from MRI or CT scan can not be directly used for generating high quality tetrahedral or hybrid volume mesh. It has to be preproccesed to accommodate boundary conditions and scale-invariability. In order to produce a mesh that is useful for numerical analysis, we identify outlets and perform the operations of smoothing, refinement and de-refinement to the surface, subject to volumetric constraint, by limiting perturbations to a small fraction of a voxel. We presented a new method for computing the medial curves, which includes identifying and truncating the boundary outlets. We defined the medial curves based on a decomposition of objects into three different types of substructures, including tips, junctions, and segments, which naturally map to substructures of the medial curve. To facilitate the identification of these substructures, we introduced a new geometric concept called the interior center of curvature (ICC), which constitute the medial axis of canal surfaces, and the average interior centers of curvature (AICC), which inherits the key features of the ICC while being more noise resistant. We then proposed robust numerical techniques, including eigenvalue analysis, weighted least squares approximations, and minimization algorithms, to enhance the noise resistance. We have illustrated the effectiveness and robustness of our approach with some highly complex, large-scale, noisy biomedical geometries derived from medical images, and presented a comparison of our method with some existing methods.

Having truncated the boundary outlets we can proceed to generation of the volume mesh. We introduced a new method for generating prismatic boundary-layer meshes for complex biological geometries. Our method uses a gradient-limited feature size (GLFS) to guide the generation of the prismatic mesh at the boundary layers and the tetrahedral mesh in the interior. Unlike previous approaches, which propagate vertices along some vertex normals, our algorithm uses the face-offsetting method (FOM) to advance the surfaces. In addition, we introduced a novel prismatic variational smoothing procedure to improve bases triangle shapes and edge orthogonality. Our experimental study showed that the face-offsetting method is more robust than traditional methods based on vertex propagation, and our variational smoothing further enhances the robustness substantially and enables high-quality prismatic meshes. In addition, our experimentations also demonstrated that it is in general more robust to propagate the surface and then interpolate the intermediate layers than propagating the surface layer by layer. Combined with a robust, high-quality tetrahedral mesh generation, our overall framework produces high-quality hybrid meshes that adapt to the local scales of the geometry. We demonstrated our method for two complex geometries, including a rat lung and an idealized human heart. We have conducted preliminary numerical simulations using our hybrid meshes and demonstrated that our method is effective and promising for biomedical computations. Moreover, we extended this method to be able to add prismatic boundary layer to any tetrahedral mesh while preserving its quality. This allowed us to accommodate a method for generation of anisotropic scale-invariant layered tetrahedra meshes developed by our collaborators at PNL which has several significant advantages over isotropic generators.

Bibliography

[1] T. Alumbaugh and X. Jiao. Compact array-based mesh data structures. In *Proc. 14th Int. Meshing Roundtable*, pages 485–504, 2005.

[2] N. Amenta, S. Choi, and R. K. Kolluri. The power crust. In *Proc. 6th ACM Symposium on Solid Modeling and Applications*, pages 249–266, 2001.

[3] A. N. Athanasiadis and H. Deconinck. A folding/unfolding algorithm for the construction of semi-structured layers in hybrid grid generation. *Comput. Meth. Appl. Mech. Engrg.*, 194:5051–5067, 2005.

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

[5] R. Aubry and R. Löhner. Generation of viscous grids at ridges and corners. *Int. J. Numer. Meth. Engrg.*, 2008. to appear.

[6] C. Bauer. *Segmentation of 3D Tubular Tree Structures in Medical Images*. PhD thesis, Graz University of Technology, Austria, April 2010.

[7] I. Bitter, A. E. Kaufman, and M. Sato. Penalized-distance volumetric skeleton algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):496–508, 2001.

[8] H. Blum. A transformation for extracting new descriptors of shape. In *Models for the Perception of Speech and Visual Form*, pages 362–380. MIT Press, Cambridge, MA, 1967.

[9] N. D. Cornea, D. Silver, and P. Min. Curve-skeleton properties, applications, and algorithms. *IEEE Trans. Vis. Comput. Graph.*, 13:530–548, 2007.

[10] N. D. Cornea, D. Silver, X. Yuan, and R. Balasubramanian. Computing hierarchical curve-skeletons of 3D objects. *Vis. Comput.*, 21(11):945 – 55, 2005.

[11] C. Couprie, L. Grady, L. Najman, and H. Talbot. Power watersheds: A new image segmentation framework extending graph cuts, random walker and optimal spanning forest. In *International Conference on Computer Vision 2009*, 2009.

[12] J. De Backer, W. Vos, A. Devolder, S. Verhulst, P. Germonpré, F. Wuyts, P. Parizel, and W. De Backer. Computational fluid dynamics can detect changes in airway resistance in asthmatics after acute bronchodilation. *J. Biomech.*, 41:106–113, 2007.

[13] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008.

[14] T. Dey and J. Sun. Defining and computing curve-skeletons with medial geodesic function. In *Proc. Eurographics Symp. Geometry Proc.*, pages 143–152, 2006.

[15] A. Dheeravongkit and K. Shimada. Inverse adaptation of a hex-dominant mesh for large deformation finite element analysis. *Comput. Aid. Des.*, 39:427–438, 2007.

[16] P. Doyle and L.Snell. *Random walks and electric networks*. Carus mathematical monographs. Mathematical Association of America, 1984. Washington, D.C.

[17] D. R. Einstein, B. Neradilak, N. Pollisar, K. R. Minard, C. Wallis, M. Fanucchi, J. P. Carson, Kuprat, S. Kabilan, R. E. Jacob, and R. A. Corley. An automated self-similarity analysis of the pulmonary tree of the sprague-dawley rat. *Anat. Rec.*, 291:1628–1648, 2008.

[18] H. Farid and E. P. Simoncelli. Differentiation of discrete multidimensional signals. *IEEE Transactions on Image Processing*, 13:496–508, 2004.

[19] R. V. Garimella and M. Shephard. Boundary layer mesh generation for viscous flow simulations. *Int. J. Numer. Meth. Engrg.*, 49:193–218, 2000.

[20] L. Grady. Random walks for image segmentation. *IEEE Trans Pattern Anal Mach Intell*, 28(11):1768–83, Nov 2006.

[21] A. Gray, E. Abbena, and S. Salamon. *Modern Differential Geometry of Curves and Surfaces with Mathematica*. Chapman & Hall/CRC, 3rd edition, 2006.

[22] O. Hassan, K. Morgan, E. Probert, and J. Peraire. Unstructured tetrahedral mesh generation for three-dimensional viscous flows. *Int. J. Numer. Meth. Engrg.*, 39:549–567, 1996.

[23] M. T. Heath. *Scientific Computing: An Introductory Survey*. McGraw-Hill, New York, 2nd edition, 2002.

[24] Y. Huo and G. S. Kassab. A hybrid one-dimensional/Womersley model of pulsatile blood flow in the entire coronary arterial tree. *Am J Physiol Heart Circ Physiol.*, 292(6):H2623–33, 2007.

[25] Y. Huo and G. S. Kassab. A scaling law of vascular volume. *Biophys J.*, 96(2):347–53, 2009.

[26] Y. Ito and K. Nakahashi. Improvements in the reliability and quality of unstructured hybrid mesh generation. *Int. J. Numer. Meth. Fluids*, 45:79–108, 2004.

[27] Y. Ito, A. M. Shih, B. Soni, and K. Nakahashi. Multiple marching direction approach to generate high quality hybrid meshes. *AIAA J.*, 45:162–167, 2007.

[28] R. K. Jaiman, X. Jiao, P. H. Geubelle, and E. Loth. Conservative load transfer along curved fluid-solid interface with non-matching meshes. *J. Comput. Phys.*, 218:1, 2006.

[29] X. Jiao. Face offsetting: A unified approach for explicit moving interfaces. *J. Comput. Phys.*, 220:612–625, 2007.

[30] X. Jiao, D. R. Einstein, V. Dyedov, and J. P. Carson. Automatic identification and truncation of boundary outlets in complex imaging-derived biomedical geometries. *Med. Bio. Engrg. Comput.*, 2009. To appear.

[31] X. Jiao, D. Wang, and H. Zha. Simple and effective variational optimization of surface and volume triangulations. In *Proc. 17th Int. Meshing Roundtable*, 2008. to appear.

[32] X. Jiao, D. Wang, and H. Zha. Simple and effective variational optimization of surface and volume triangulations. In *Proc. 17th Int. Meshing Roundtable*, pages 315–332, Pittsburgh, PA, 2008.

[33] X. Jiao and H. Zha. Consistent computation of first- and second-order differential quantities for surface meshes. In *Prof. ACM Solid and Physical Modeling Symposium*, pages 159–170, 2008.

[34] F. Juretic. *Error Analysis in Finite Volume CFD*. PhD thesis, Imperial College, University of London, 2005.

[35] R. Kadirvel, Y. Ding, D. Dai, H. Zakaria, A. Robertson, M. Danielson1, D. Lewis, H. Cloft, and D. Kallmes. The influence of hemodynamic forces on biomarkers in the walls of elastase-induced aneurysms in rabbits. *Neuroradiology*, 49:1041–1053, 2007.

[36] S. Kakutani. Markov processes and the dirichlet problem. In *proc. Jap. Acad.*, volume 21, pages 227–233, 1945.

[37] Y. Kallinderis and S. Ward. Prismatic grid generation for three-dimensional complex geometries. *AIAA J.*, 31:1850–1856, 1993.

[38] S. Katz, G. Leifman, and A. Tal. Mesh segmentation using feature point and core extraction. *Vis. Comput.*, 21:865–875, 2005.

[39] A. Khamayseh and G. Hansen. Use of the spatial kd-tree in computational physics applications. *Communications in Computational Physics*, 2:545–576, 2007.

[40] A. Khawaja, H. McMorris, and Y. Kallinderis. Hybrid grids for viscous flows around complex 3-D geometries including multiple bodies, 1995. AIAA-95-1685.

[41] C. Kleinstreuer, H. Shi, and Z. Zhang. Computational analyses of a pressurized metered dose inhaler and a new drug-aerosol targeting methodology. *J. Aerosol Med.*, 20(3):294–309, 2007.

[42] P. M. Knupp. Achieving finite element mesh quality via optimization of the Jacobian matrix norm and associated quantities. Part II–a framework for volume mesh optimization and the condition number of the Jacobian matrix. *Int. J. Numer. Meth. Engrg.*, 48:1165–1185, 2000.

[43] A. Kuprat, A. Khamayseh, D. George, and L. Larkey. Volume conserving smoothing for piecewise linear curves, surfaces, and triple lines. *J. Comput. Phys.*, 172:99–118, 2001.

[44] A. P. Kuprat and D. R. Einstein. An anisotropic scale-invariant unstructured mesh generator suitable for volumetric imaging data. *J. Comput. Phys.*, 228:619–640, 2009.

[45] L. Lam, S.-W. Lee, and C. Y. Suen. Thinning methodologies—a comprehensive survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14:869–885, 1992.

[46] T.-C. Lee, R. L. Kashyap, and C.-N. Chu. Building skeleton models via 3-D medial surface/axis thinning algorithms. *CVGIP: Graph. Models Image Process.*, 56(6):462–478, 1994.

[47] P. Lo, B. van Ginneken, J. M. Reinhardt, and M. de Bruijne. Extraction of airways from ct. In *Conf Proc EXACT'09*, volume 2009, 2009.

[48] R. Löhner and J. Cebral. Generation of non-isotropic unstructured grids via directional enrichment. *Int. J. Numer. Meth. Engrg.*, 49:219–232, 2000.

[49] W. Lorensen and H. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *Proc. SIGGRAPH 87*, volume 21, pages 163–169, 1987.

[50] M. Maddah, A. Afzali-Kusha, and H. Soltanian-Zadeh. Efficient center-line extraction for quantification of vessels in confocal microscopy images. *Med Phys*, 30(2):204–11, 2003.

[51] M. Maddah, H. Soltanian-Zadeh, and A. Afzali-Kusha. Snake modeling and distance transform approach to vascular centerline extraction and quantification. *Comput. Med. Imaging Graph. (UK)*, 27(6):503 – 12, 2003.

[52] S. Molloi, G. S. Kassab, and Y. Zhou. Quantification of coronary artery lumen volume by digital angiography: In-vivo validation. *Circulation*, 104:2351, 2001.

[53] R. C. Molthen, K. L. Karau, and D. C. A. Quantitative models of the rat pulmonary arterial tree morphometry applied to hypoxia-induced arterial remodeling. *J Appl Physiol*, 97(6):2372–84, 2004.

[54] M. Mortara and G. Patane. Affine-invariant skeleton of 3d shapes. In *Shape Modeling International*, pages 245–252, 2002.

[55] M. Mortara, G. Patane, M. Spagnuolo, B. Falcidieno, and J. Rossignac. Blowing bubbles for the multi-scale analysis and decomposition of triangle meshes. *Algorithmica, Special Issues on Shape Algorithms*, 38(2):227–248, 2004.

[56] M. Mortara, G. Patane, M. Spagnuolo, B. Falcidieno, and J. Rossignac. Plumber: a method for a multi-scale decomposition of 3d shapes into tubular primitives and bodies. In *ACM Symposium on Solid Modeling and Applications*, 2004.

[57] T. Newman and H. Yi. A survey of the marching cubes algorithm. *Computers & Graphics*, 30:854–879, 2006.

[58] G. M. Nielson. On marching cubes. *IEEE Trans. Vis. Comput. Graph.*, 9(3):283–97, 2003.

[59] K. Palágyi and A. Kuba. A 3d 6-subiteration thinning algorithm for extracting medial lines. *Pattern Recognition Letters*, 19:613–627, 1998.

[60] V. Pascucci, G. Scorzelli, P.-T. Bremer, and A. Mascarenhas. Robust on-line computation of Reeb graphs: simplicity and speed. *ACM Trans. Graph.*, 26(3):58, 2007.

[61] S. Perry, A. M. Purohit, S. Boser, I. Mitchell, and F. H. Green. Bronchial casts of human lungs using negative pressure injection. *Exp Lung Res*, 26:27–39, 2000.

[62] S. Pirzadeh. Unstructured viscous grid generation by advancing-layers method. *AIAA J.*, 32:1735–1737, 1994.

[63] S. Pirzadeh. Three-dimensional unstructured viscous grids by the advancing-layers methods. *AIAA J.*, 34:43–49, 1996.

[64] C. Schirmer and A. Malek. Wall shear stress gradient analysis within an idealized stenosis using non-Newtonian flow. *Neurosurgery*, 61:853–863, 2007.

[65] J. Sethian. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge University Press, 1999.

[66] D. Sharov and K. Nakahashi. Hybrid prismatic/tetrahedral grid generation for viscous flow applications. *AIAA J.*, 36:157–162, 1998.

[67] J. R. Shewchuk. Triangle: Engineering a 2d quality mesh generator and Delaunay triangulator. In *Lecture Notes in Computer Science*, volume 1148, pages 203–222, 1996.

[68] J. R. Shewchuk. General-dimensional constrained Delaunay and constrained regular triangulations, i: combinatorial properties. *Discrete Comput. Geom.*, 39:580–637, 2008.

[69] H. Si. Tetgen, a quality tetrahedral mesh generator and three-dimensional Delaunay triangulator v1.4, 2006.

[70] H. Si. Adaptive tetrahedral mesh generation by constrained Delaunay refinement. *Int. J. Numer. Meth. Engrg.*, 2008. http://dx.doi.org/10.1002/nme.2318.

[71] J. F. Thompson and B. Soni, editors. *Handbook of Grid Generation*. CRC Press, 1999.

[72] J. Tierny, J.-P. Vandeborre, and M. Daoudi. Enhancing 3D mesh topological skeletons with discrete contour constrictions. *Vis. Comput.*, 24(3):155–172, March 2008.

[73] G. M. Treece, R. W. Prager, and A. H. Gee. Regularised marching tetrahedra: improved iso-surface extraction. *Computers & Graphics*, 23(4):583–598, 1999.

[74] J. K. Udupa, P. K. Saha, and R. A. Lotufo. Relative fuzzy connectedness and object definition: Theory, algorithms, and applications in image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(11):1485–1500, 2002.

[75] Y. Wang, F. Guibault, and R. Camarero. Eikonal equation-based front propagation for arbitrary complex configurations. *Int. J. Numer. Meth. Engrg.*, 73:226–247, 2008.

[76] Y. Wang and S. Murgie. Hybrid mesh generation for viscous flow simulations. In *Proc. 15th Int. Meshing Roundtable*. Springer, 2006.

[77] Y.-S. Wang and T.-Y. Lee. Curve-skeleton extraction using iterative least squares optimization. *IEEE Trans. Vis. Comput. Graph.*, 14:926–936, 2008.

[78] D. R. Whittaker, J. Dwyer, and M. F. Fillinger. Prediction of altered endograft path during endovascular abdominal aortic aneurysm repair with the gore excluder. *J Vasc Surg*, 41(4):575–83, 2005.

[79] R. Wiemker, T. Bulow, and C. Lorenz. A simple centricity-based region growing algorithm for the extraction of airways. In *Proc. of the Second International Workshop on Pulmonary Image Analisys*, pages 309–314, 2009.

[80] T. Wischgoll, J. S. Choy, E. L. Ritman, and G. S. Kassab. Validation of image-based method for extraction of coronary morphometry. *Ann Biomed Eng*, 36(3):356–68, 2008.

[81] C. Xu and J. L. Prince. Gradient vector flow: A new external force for snakes. In *IEEE Proc Conf on Comp Vis Patt Recog*, pages 66–71, 1997.