Summer 8-2020

Automatic Linear and Curvilinear Mesh Generation Driven by Validity Fidelity and Topological Guarantees

Jing Xu
Old Dominion University, jingxu0513@gmail.com

Recommended Citation
Xu, Jing. "Automatic Linear and Curvilinear Mesh Generation Driven by Validity Fidelity and Topological Guarantees" (2020). Doctor of Philosophy (PhD), Dissertation, Computer Science, Old Dominion University, DOI: 10.25777/31kp-az96
https://digitalcommons.odu.edu/computerscience_etds/127

AUTOMATIC LINEAR AND CURVILINEAR MESH GENERATION DRIVEN BY VALIDITY FIDELITY AND TOPOLOGICAL GUARANTEES

by

Jing Xu
B.S. June 2005, Information Engineering University, China
M.S. June 2009, Soochow University, China

A Dissertation Thesis Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

COMPUTER SCIENCE

OLD DOMINION UNIVERSITY
August 2020

ABSTRACT

AUTOMATIC LINEAR AND CURVILINEAR MESH GENERATION DRIVEN BY VALIDITY FIDELITY AND TOPOLOGICAL GUARANTEES

Jing Xu
Old Dominion University, 2020
Director: Dr. Andrey Chernikov

Image-based geometric modeling and mesh generation play a critical role in computational biology and medicine. In this dissertation, a comprehensive computational framework for both guaranteed quality linear and high-order automatic mesh generation is presented. Starting from segmented images, a quality 2D/3D linear mesh is constructed. The boundary of the constructed mesh is proved to be homeomorphic to the object surface. In addition, a guaranteed dihedral angle bound of up to 19.47° for the output tetrahedra is provided. Moreover, user-specified guaranteed bounds on the distance between the boundaries of the mesh and the boundaries of the materials are allowed. The mesh contains a small number of mesh elements that comply with these guarantees, and the runtime is compatible in performance with other software. Then the curvilinear mesh generator allows for a transformation of straight-sided meshes to curvilinear meshes with C^1 or C^2 smooth boundaries while keeping all elements valid and with good quality as measured by their Jacobians. The mathematical proof shows that the meshes generated by our algorithm are guaranteed to be homeomorphic to the input images, and all the elements inside the meshes are guaranteed to be with good quality. Experimental results show that the mesh boundaries represent the objects' shapes faithfully, and the accuracy of the representation is improved compared to the corresponding linear mesh.

ACKNOWLEDGMENTS

My first gratitude goes to my advisor, Dr. Andrey Chernikov. I greatly value the opportunity and experience to work as his Ph.D. student. His rigorous attitude on the technology, his persistence to innovations, and his continuous support and patience are the bright lights on my doctoral research way. I thank him for his support of many insightful discussions that helped me get a better understanding of my research problems, and for his patience in going through all my drafts of my papers.

I would also like to thank my committee members Dr. Danella Zhao, Dr. Jiangwen Sun and Dr. Nail Yamaleev, for their helpful comments and invaluable suggestions to improve the contents of this work. I gratefully acknowledge the generous support of Old Dominion University Modeling and Simulation fellowship on this research.

I am deeply grateful to the friends who have shared both my joyful and upset moments: Panagiotis Foteinos, Fotis Drakopoulos, Eleni adam, Christos Tsolakis, Polykarpos Thomadakis, Thomas Kennedy, Juliette Pardue, Kevin Garner and Yixun Liu.

Finally, I am especially grateful to my family. They always stand by me, support me, and love me, regardless of my successes or failures.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 MOTIVATION

Discretizations of complex shapes into simple elements are widely used in various computing areas that require a quantitative analysis of spatially dependent attributes. One, traditional, area is the finite element analysis [1] which is used to numerically solve partial differential equations derived using solid mechanics and computational fluid dynamics approaches. With this approach one starts with the knowledge of the constitutive physical laws and initial (boundary) conditions and obtains a prediction of the observable properties of objects of interest. Another, emerging, area is the use of discretizations for delineating homogeneous spatial zones within objects that can be represented as units for an overall object description. With this approach one starts with the knowledge of observable object properties and uses statistical methods to infer the processes that govern the formation of the object. Therefore, the second approach can be viewed as a reversal of the first approach, that still relies on a similar discretization technique. This second approach is a useful tool for bioinformatics applications, for example gene expression pattern analysis [2, 3, 4, 5].

Said discretizations of objects are usually called meshes, and the simple elements that they consist of are either triangles and tetrahedra (in two and three dimensions, respectively), or quadrilaterals and hexahedra. Furthermore, elements can have either straight or curved sides. In previous work [2, 3] triangular meshes with straight sides were used to discretize images of fruit fly embryos. However, the embryos, like most biological objects, have curved shapes, and their discretizations with straight-sided elements have limited accuracy. To obtain much higher accuracy one needs to use curved-sided elements that match the curves of object boundaries.

Image-based mesh generation is a relatively new field. The research on mesh generation is dominated by tetrahedra meshing algorithms, which can be grouped into Delaunay refinement methods, advancing front methods, and space-tiling methods.

Delaunay refinement methods [6, 7] use the Delaunay criterion for guiding element construction after point insertion. These methods allow for the mathematical proofs of element quality and good grading. Quality is traditionally defined as the ratio of the circumradius of the element to the length of its shortest edge, which does not imply a bound on dihedral angles in 3D. Advancing front methods [8, 9, 10] build the mesh in layers starting by dividing the boundaries of the mesh into triangular faces and work toward the center of the region being meshed. Some of the advancing front methods [11] combine the Delaunay triangulation and advancing front ideas, however, no theoretical guarantees are usually available. Space-tiling methods [12, 13, 14, 15, 16] build uniform meshes from equal-sized standard cubic grid or body-centric cubic lattice, and graded meshes from octree. Sometimes theoretical guarantees on the quality in terms of dihedral angle are provided. This work falls into the third category.

The key challenges scientists face on image-based mesh generation are:

- Automatic meshing techniques for heterogeneous domains with non-manifold boundaries;
- Robust unstructured mesh generation with topology validation;
- High-order mesh construction techniques for complicated domains;
- Guaranteed validity and effectively improved quality of high-order elements.

In this dissertation, a comprehensive computational framework for both guaranteed quality linear and high-order automatic mesh generation for heterogeneous domains is presented. It is composed of two algorithms. Starting from a segmented image with multiple materials, the first algorithm automatically constructs a two- or a three-dimensional unstructured linear mesh that satisfies the quality, fidelity, and topological guarantees. The second algorithm takes the two-dimensional linear mesh as the input, and it transforms the straight-sided mesh to a curvilinear mesh with C^1 or C^2 smooth boundaries while keeping all elements valid and with good quality as measured by their Jacobians.

1.2 AUTOMATIC LINEAR TRIANGULAR/TETRAHEDRAL MESH CONSTRUCTION

With volumetric data sets, one can generate conforming quality meshes of the spatially realistic domains that help to produce computer-aided visualization, manipulation, and quantitative analysis of the multi-dimensional image data. The domain of interest often includes heterogeneous materials that specify functionally different characteristic properties. In finite element analysis, each material is assigned individual attributes. Thus, meshes with conforming boundaries describing each of the partitioned material regions need to be generated. In this dissertation, images that are segmented into multiple homogeneous regions are our concern, unsegmented images with noise are out of the scope.

An algorithm for constructing tetrahedral volume meshes from segmented multi-material images is presented. It generates a mesh which satisfies all the following criteria:

1. The mesh offers a faithful topology to the materials, i.e., there is a homeomorphism between the boundary of the mesh and the boundary of the materials. The term homeomorphism, also called a continuous transformation, is a topological notion of equivalence. Since grid-based schemes can miss important details of the surface, correct topology usually needs extra efforts to be guaranteed. For implicit surfaces, tools such as critical points [17, 18], Lipschitz conditions [19, 20, 18] and interval arithmetic [21] can be used to capture the topological details. For images, we give a sufficient condition (*single manifold condition*, see section 2.4) for the approximation to offer homeomorphism.

2. Elements with arbitrarily small angles which cause the stiffness matrix in FE analysis to be ill-conditioned do not appear in the mesh. Specifically, we guarantee that all dihedral angles are above a user-specified lower bound which can be set to any value up to 19.47°. The algorithm exposes the parameter that allows for a trade-off between the minimum dihedral angle and the final number of elements. In contrast, the state-of-the-art guaranteed quality Delaunay method only allows for a bound on the circumradius-to-shortest-edge ratio of tetrahedra, which in three dimensions is not equivalent to a bound on the smallest dihedral angles.

3. The mesh is able to offer a close representation (fidelity) of the underlying materials. In particular, the two-sided Hausdorff distance between the boundaries of the mesh and the boundaries of the materials respects the user-specified fidelity

bounds. The Hausdorff distance measures how far two subsets of a metric space are from each other (see the definition in section 2.2). This bound can be zero, which means that the mesh is a strict matching to individual pixels boundaries of the object. However, a strict matching to image boundaries will produce a large number of elements that will slow down the solver. In some cases, the FE analysis requires a loose fidelity bound but a faster solver, for example, for image registration [22]. Our solution exposes parameters that allow for a trade-off between the fidelity and the final number of elements.

4. The mesh contains a small number of elements that comply with the three guarantees above. The reason for this criterion is that the cost of assembling and solving a sparse system of linear equations in the FE method directly depends on the number of tetrahedra [1, 23]. To achieve this goal, our method is able to offer the ability to grade (by an octree) from small to large elements over a relatively short distance, not only in volume but also on the surface. We also implemented a specialized post-processing procedure to decimate the mesh such that the number of tetrahedra in the mesh is as small as possible.

5. The mesh can be constructed within tight real-time time constraints enforced by clinical simulation and applications that require interactive re-meshing such as surgical simulations and image-guided interventions. The efficient implementation described below enables our software compatible in performance with other software such as a state-of-the-art Delaunay code.

Meshing techniques have been proposed to mesh the domain of a *Piecewise Linear Complex* (PLC). It can be composed of polygons of any shape. The challenge is that the quality of the input PLC affects the quality of the final mesh, because the mesh has to match exactly to the boundaries of the model. For images, the faces of each boundary voxel can be considered as the input PLC, since these faces meet at angles of 90° or 180°, thus alleviates this challenge. The most popular technique for generating tetrahedral meshes in this category of techniques is Delaunay refinement [24, 25, 26, 27, 28, 29]. However, the problem with Delaunay refinement is that it only satisfies a bound on the circumradius-to-shortest-edge ratio, which works well in two dimensions, but in three dimensions it does not imply a bound on dihedral angles. As a consequence, tetrahedra with very small dihedral angles called slivers can survive. *Sliver exudation* [27, 30, 31] or other optimization post-processing techniques [32, 24, 26, 33, 34] can be used to eliminate degenerated elements, but those methods are unable to guarantee a meaningful dihedral angle bounds(they exist in theory but too small to even be computed).

Meshing techniques also have been proposed to mesh the domain through an implicit function $f : R^3 \to R$ such that points in different regions of interest evaluate f differently. One guaranteed-quality technique in this category is based on the Delaunay refinement [35, 17, 36, 37], where a piecewise-linear approximation of a surface is generated from a finite set of sufficiently dense sample points [38, 19, 20]. Another guaranteed-quality technique in this category employs a space-tiling background grid to guide the creation of a mesh [14, 15, 39, 40, 16], the focus of this work.

Isosurface Stuffing [14] is a guaranteed-quality tetrahedral meshing algorithm for general surfaces under the assumption that the surface is a smooth 2-manifold. It offers the one-sided Hausdorff distance guarantee from the mesh to the model. If the surface is a smooth manifold with bounded curvature, it also provides the one-sided Hausdorff distance guarantee from the model to the mesh. Using interval arithmetic to account for vertex movement, the dihedral angles for the mesh with uniform-sized boundary are proved to be above 10.7°. However, the method presented in this dissertation is able to achieve a minimum dihedral angle bound of 19.47°.

Expanding on the ideas from Isosurface Stuffing, Walker [15] proposed a method for generating quasi-uniform tetrahedral meshes of solids whose boundary is a smooth surface using edge rearrangement. If the lattice spacing is smaller than the local feature size, then the dihedral angles are bounded above 11.4°. As the Isosurface Stuffing, if the surface has bounded curvature and if the background grid is sufficiently fine, then the Hausdorff distance guarantee is two-sided. The proposed algorithm is able to achieve a higher minimum dihedral angle bound even with a fully graded case using an octree decomposition.

Bronson et al. designed software called Cleaver [39] based on Isosurface Stuffing for generating surface and volumetric meshes from three-dimensional imaging data. By designing a generalized stencil for lattice tetrahedra that contains different materials, their method is able to handle volumetric domains consisting of multiple materials. The method guarantees the element quality by proving that the dihedral angles are bounded above 2.8°. It offers a one-sided Hausdorff distance guarantee from the surface mesh to the boundary of the materials. However, it is not proven that the Hausdorff distance bound is two-sided and the mesh is topologically accurate.

Liang and Zhang [40] proposed an octree-based dual contouring algorithm with guaranteed mesh quality for closed smooth surfaces. The algorithm first generates an octree, then it adjusts the octree grid points to the input surface if they are too close to the input surface. Finally, a dual contouring method with two minimizers is applied to generate the tetrahedral mesh. However, this method only handles the single-material geometry model, and there is no proof of geometric accuracy. The minimum dihedral angle bound (proved to be 12.04° with a small perturbation) is smaller than our minimum dihedral angle bound 19.47°.

Foteinos et al. [37] present a Delaunay meshing algorithm with several mathematical guarantees. Using a strategy called ϵ*-sample* [19, 20], the mesh boundary is proved to be ambient isotopic to the object surface. Although the circumradius-to-shortest-edge ratio of the tetrahedra in the output mesh is proved to be less than 1.93, they didn't provide any dihedral angle bound. Even if the radius-edge ratio is very small, it can not avoid the almost flat tetrahedra. Furthermore, the two-sided Hausdorff distance between the object surface and mesh boundary is bounded by $O(\delta^2)$, where δ is a small constant 0.0168. However, if the constant implied in Big-O is very large, the two-sided Hausdorff distance bound could also be very large.

This work builds upon the Lattice Decimation (LD) method [16]. LD is a tetrahedral image-to-mesh conversion algorithm that allows for guaranteed bounds on the smallest dihedral angle (up to 35.26°) and on the 2-sided Hausdorff distance between the boundaries of the mesh and the boundaries of the materials. The LD algorithm constructs an octree and refines it until no leaf contains voxels from multiple materials. Then it fills the octree leaves with high-quality elements. This initial mesh has a large number of elements, because it satisfies the highest dihedral angle bounds (35.26°) and fidelity bounds (zero). The authors designed a post-processing decimation step using vertex removal operation [41, 42] while at all times maintaining the required fidelity and quality bounds. However, the decimation step is a greedy algorithm which was not designed for a smooth transition in element size. In fact, it can produce clusters of smaller elements surrounded by much larger elements. In this work, the octree is refined to a smaller depth so that the tiny elements are avoided, therefore, the issue can be mitigated. Moreover, although LD maintains the material connectivity, it does not guarantee topological fidelity. A new technique is designed based on the enforcement of a *single manifold condition* that solves this problem.

The algorithm proposed in this dissertation simultaneously satisfies the quality, fidelity, and topology requirements. At the same time, the number of tetrahedra in the mesh is low subject to the three requirements above. Compared to the LD algorithm, a coarser initial mesh is constructed before decimation. As a result, the geometric and topological fidelity needs to be taken care of both before and after decimation.

Using a pre-defined look-up table, the boundary of the materials is approximated with a set of triangular patches in each octree leaf. The triangular patches all together form a water-tight surface mesh. The topological faithfulness is feasible due to the *single manifold condition*. The proof of the homeomorphism between the boundaries of the mesh and the boundary of the materials relies on it. The two-sided Hausdorff distance between the triangular patches and the boundaries of the materials respects a user-specified fidelity bound. This goal is achieved by constructing a sequence of homeomorphic water-tight surface meshes until the fidelity condition is satisfied. A second pre-defined look-up table is used to fill the octree leaves with high-quality elements. The quality of the final mesh is proved by analyzing all possible predefined shapes of the tetrahedra filling a cubic leaf of the octree. During decimation, the initial mesh is coarsened to a much lower number of elements while at all times the quality, fidelity, and topological requirements are maintained. Our measurements show that it meshes three-dimensional images of practically significant sizes in time that matches the performance of the LD method and a Delaunay open-source mesh generator Computational Geometry Algorithms Library (CGAL).

1.3 AUTOMATIC TWO-DIMENSIONAL CURVILINEAR MESH GENERATION

For automatically generating valid high-order meshes to represent curvilinear domains with smooth global mesh boundaries, cubic Bézier polynomial basis is selected for the geometric representation of the elements, because it provides a convenient framework supporting the smooth operation and mesh validity verification. We highlight the three contributions:

1. The curved mesh boundary is globally smooth. It satisfies C^1 or C^2 smoothness requirement chosen by the user.

2. The proposed approach is robust in the sense that the invalid elements are

eliminated, and the mesh quality is enforced.

3. The method provides higher accuracy compared to the linear discretization.

The procedure starts with the automatic construction of a linear mesh. The edges of those linear elements which are classified on the boundary are then curved using cubic Bézier polynomials such that these boundary edges constitute a smooth closed curve. Once the validity verification procedure detects invalid elements, the method next curves the interior elements by iteratively solving for the equilibrium configuration of an elasticity problem until all the invalid elements are eliminated.

Various procedures have also been developed and implemented by other authors to accomplish the generation of a curvilinear mesh. Sherwin and Peiro [43] adopted three strategies to alleviate the problem of invalidity: generating boundary conforming surface meshes that account for curvature; the use of a hybrid mesh with prismatic and tetrahedral elements near the domain boundaries; refining the surface meshes according to the curvature. The mesh spacing is decided by a user-defined tolerance ϵ related to the curvature and a threshold to stop excessive refinement. In this work, a method was developed that allows for an all triangle mesh which simplifies and unifies both meshing and analysis. Persson and Peraire [44] proposed a node relocation strategy for constructing well-shaped curved meshes. Compared to the proposed method which iteratively solves for the equilibrium configuration of a linear elasticity problem, they use a nonlinear elasticity analogy, and by solving for the equilibrium configuration, vertices located in the interior are relocated as a result of a prescribed boundary displacement. Luo et al. [45] isolate singular reentrant model entities, then generate linear elements around those features, and curve them while maintaining the gradation. Local mesh modifications such as minimizing the deformation, edge or facet deletion, splitting, collapsing, swapping as well as shape manipulation are applied to eliminate invalid elements whenever they are introduced instead of our global node relocation strategy. George and Borouchaki [46] proposed a method for constructing tetrahedral meshes of degree two from a polynomial surface mesh of degree two. *Jacobian* is introduced for guiding the correction of the invalid curved elements. In contrast, the proposed method does not require a starting curved boundary mesh, as well as produces more flexible cubic elements.

CHAPTER 2

PRELIMINARIES

2.1 MULTI-MATERIAL IMAGE, IMAGE BOUNDARY, MESH BOUNDARY AND BOUNDARY LEAF

Let $\Omega \subset \mathcal{R}^3$ be the domain of a multi-material segmented image composed of a collection of voxels V. Ω contains the object $\Phi \subset \Omega$ to be meshed. Φ is composed of a finite set of distinct materials $\Phi = \bigcup_{i=1}^{n} \Phi_i$. A function $f : V \rightarrow \{0, 1, ..., n\}$ is defined such that each voxel v is assigned a label of a single material or of the background. In particular, v evaluates f to a positive integer i if it belongs to the material Φ_i, or to 0 if it lies in the background.

Image boundary is the union of voxel faces shared by voxels of different colors. It includes the interior boundary and exterior boundary. The exterior boundary is a closed manifold (or manifolds) and separates the materials from the background. The interior boundary separates different materials.

An *image boundary edge* is an edge that two pixels/voxels of different colors incident. In the three-dimensional case, an *image boundary face* is the face of a voxel upon which a voxel of a different color is incident. The endpoints of image boundary edges and the corner points of image boundary faces are called *image boundary vertices.*

An octree (or a quadtree) leaf is called a *proper boundary leaf* if it contains more than one material; an octree (or a quadtree) leaf is called a *non-proper boundary leaf* if it contains only one material, but at least one of the voxels that are incident upon the cube faces is adjacent to a voxel of a different color through voxel face. A *boundary leaf* is either a proper boundary leaf or a non-proper boundary leaf.

The notation used in the rest of the dissertation is presented below. The image boundary and the mesh surface generated by the algorithm are denoted as $\mathcal{B}$ and $\mathcal{S}$ respectively. Given an octree leaf L, $\mathcal{B}_L = L \cap \mathcal{B}$ denotes $\mathcal{B}$ restricted to the leaf L and $\mathcal{S}_L = L \cap \mathcal{S}$ denotes $\mathcal{S}$ restricted to the leaf L. Here L includes its boundary.

2.2 HAUSDORFF DISTANCE AND 2-TO-1 RULE

For image boundary $\mathcal{B}$ and surface mesh $\mathcal{S}$, the *two-sided Hausdorff distance* is:

$$H(\mathcal{B}, \mathcal{S}) = \max\{h(\mathcal{B}, \mathcal{S}), h(\mathcal{S}, \mathcal{B})\},$$

where

$$h(X, Y) = \max_{x \in X} \min_{y \in Y} d(x, y),$$

and d is the Euclidean distance.

Two octree (quadtree) leaves are neighbors if they share a face (an edge). An octree (quadtree) subdivision is considered satisfying the *2-to-1 rule* if any two neighbors differ at most by a factor of two in size.

2.3 HOMEOMORPHISM AND D-MANIFOLD WITH BOUNDARY

An *open ball* in $\mathcal{R}^n$ is the collection of points $B = B(o, r) = \{a \in \mathcal{R}^n | \quad |oa| < r\}$ for some point $o \in \mathcal{R}^n$ and some positive r. o is the *center* and r is the *radius* of B. Let $\mathcal{X}$ be a topological space. For $\mathcal{Y} \subseteq \mathcal{X}$ a *neighborhood* of $\mathcal{Y}$ in $\mathcal{X}$ is an open subset of $\mathcal{X}$ that contains $\mathcal{Y}$.

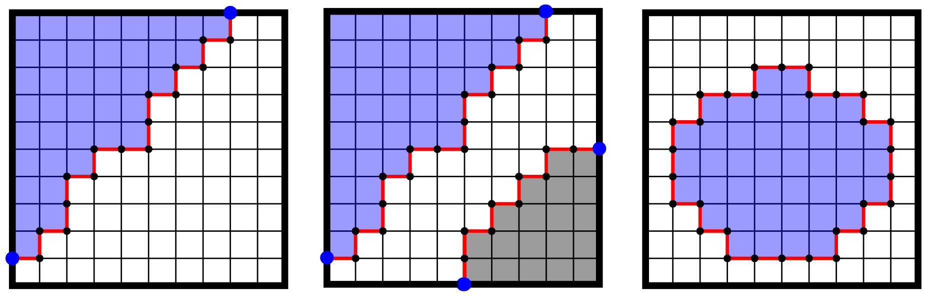
Homeomorphism is an equivalence relation and one-to-one correspondence between points in two topological spaces that is continuous in both directions. Let $\mathcal{X}$ and $\mathcal{Y}$ be two topological spaces, $\mathcal{X}$ and $\mathcal{Y}$ are *homeomorphic*, written $\mathcal{X} \approx \mathcal{Y}$, if there is a bijective map $\mu : \mathcal{X} \to \mathcal{Y}$ so that μ and μ^{-1} are continuous. μ is a *homeomorphism* between $\mathcal{X}$ and $\mathcal{Y}$. Let $d \geq 0$ be the dimension and o be the origin of $\mathcal{R}^n$, $H^d = \{x = (\xi_1, ..., \xi_d) \in \mathcal{R}^d | \quad \xi_d \geq 0\}$, and $B^d = \{x \in \mathcal{R}^d | \quad |ox| \leq 1\}$. An *open d-ball* is homeomorphic to $\mathcal{R}^d$, a *half-open d-ball* is homeomorphic to H^d, and a *closed d-ball* is homeomorphic to B^d.

$\mathcal{X} \subseteq \mathcal{R}^n$ is a *d-manifold without boundary* if each $x \in \mathcal{X}$ has an open *d-ball* as a neighborhood in $\mathcal{X}$. $\mathcal{X} \subseteq \mathcal{R}^n$ is a *d-manifold with boundary* if each $x \in \mathcal{X}$ has an open or half open *d-ball* as a neighborhood in $\mathcal{X}$, and there is at least one $x \in \mathcal{X}$ that has no open *d-ball* as a neighborhood. The set of points without open *d-ball* neighborhood forms the boundary. The boundary of an d-manifold with boundary is an $(d-1)$-manifold without boundary.

A mesh generation algorithm needs more than good elements and geometric fidelity; it also needs to produce a mesh that is a topologically accurate approximation of the domain it is supposed to represent. One of our goals is to ensure that the algorithm generates a mesh whose surface $\mathcal{S}$ is topologically equivalent to the image boundary $\mathcal{B}$. This can be guaranteed by making sure that both the image boundary and the surface mesh have the same topology within each leaf, i.e. each of the $\mathcal{S}_L$ and $\mathcal{B}_L$ is a *d-manifold with boundary*. The single manifold condition below applies to the boundary leaves. It defines the image boundary topology within each leaf.

2.4 SINGLE MANIFOLD CONDITION

Definition 1 (***Single manifold condition***) The intersection of the image boundary with each of the boundary leaves is a $(n-1)$-manifold with boundary, n being the dimension.

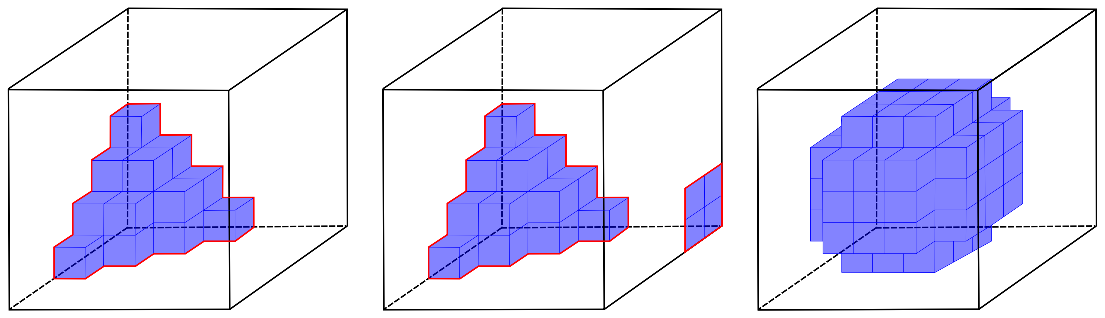

(a) A boundary quadtree leaf respects the single manifold condition

(b) A boundary quadtree leaf violates the single manifold condition

(c) A boundary quadtree leaf violates the single manifold condition

FIG. 1: An illustration of the two-dimensional single manifold condition on a quadtree leaf. Different colors in the leaf show different materials. The red segments represent the image boundary edges. The black points show the image boundary vertices that have two neighbors, and the blue points show the image boundary vertices that have only one neighbor.

Specifically, in the two-dimensional case, the image boundary edges form a 1-manifold with boundary. The 1-manifold with boundary is a simple chain composed of image boundary edges. On this chain, every image boundary vertex has two image boundary edges adjacent to it, except the first and the last ones, which have only one adjacent image boundary edge. The endpoint of the chain, a 0-manifold, is the boundary of the 1-manifold. The closed cycle is not considered, because it has no boundary. Figure 1 is an illustration of the two-dimensional single manifold condition on a quadtree leaf. Figure 1a shows a boundary quadtree leaf that contains two materials (white and blue). It respects the single manifold condition because there is a 1-manifold with boundary. Figure 1b shows a quadtree leaf that contains three materials (white, blue, and gray). It violates the single manifold condition because



(a) A boundary octree leaf respects the single manifold condition

(b) A boundary octree leaf violates the single manifold condition

(c) A boundary octree leaf violates the single manifold condition

FIG. 2: An illustration of the three-dimensional single manifold condition on an octree leaf. Each octree leaf contains a material displayed with blue, against the white background. The blue squares show the image boundary faces. The blue edges show the 2-degree image boundary edges, and the red edges show the 2-degree image boundary edges.

there is more than one 1-manifold with boundary. Figure 1c shows a boundary quadtree leaf that contains two materials (white and blue). It disobeys the single manifold condition because the 1-manifold has no boundary.

In the three-dimensional case, the *degree* of the image boundary edge is defined by the number of boundary faces that are incident upon the edge. Notice that the two neighboring voxels whose colors are different share only one boundary face. In a boundary leaf, the image boundary faces form a 2-manifold with boundary. The 2-manifold with boundary is a simple sheet on which the 1-degree image boundary edges form a cycle and all the other image boundary edges are 2-degree edges. The cycle composed by the 1-degree image boundary edges, a 1-manifold, is the boundary of the 2-manifold. A closed surface is not considered as satisfying the condition, because it is a 2-manifold without boundary. Figure 2 is an illustration of the three-dimensional single manifold condition on an octree leaf. Figure 2a shows a boundary octree leaf which respects the single manifold condition. In this leaf, the image boundary faces form a sheet which is a 2-manifold with boundary. Figure 2b shows a boundary octree leaf that violates the single manifold condition. In this leaf, there is more than one 2-manifold with boundary. Figure 2c shows a boundary leaf that disobeys the single manifold condition because the 2-manifold has no boundary.

2.5 BÉZIER CURVES

We express Bézier curves in terms of Bernstein polynomials. A n-th order Bernstein polynomial is defined explicitly by

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}, \quad i = 0, ..., n, \quad t \in [0,1],$$

where the binomial coefficients are given by

$$\binom{n}{i} = \begin{cases} \frac{n!}{i!(n-i)!} & \text{if } 0 \leq i \leq n \\ 0 & \text{otherwise.} \end{cases}$$

One of the important properties of the Bernstein polynomials is that they satisfy the following recurrence:

$$B_i^n(t) = (1-t)B_i^{n-1}(t) + tB_{i-1}^{n-1}(t),$$

with

$$B_0^0(t) \equiv 1, \quad B_j^n(t) \equiv 0 \quad for \quad j \in 0, ..., n.$$

Now the Bézier curve of degree n can be defined in terms of Bernstein polynomials as

$$b^n(t) = \sum_{i=0}^{n} B_i^n(t) P_i,$$

where the set of points $P_0, P_1, ..., P_n$ are called *control points*, and the polygon P formed by points $P_0, P_1, ..., P_n$ is called *control polygon* of the curve b^n.

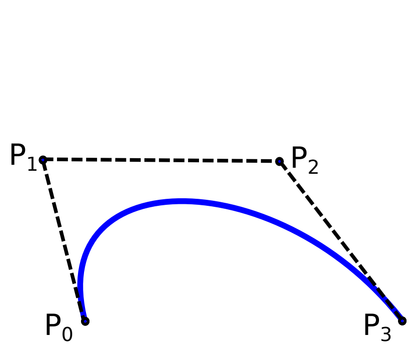
The barycentric form of Bézier curves demonstrates its symmetry property nicely. Let u and v be the barycentric coordinates, $u \in [0,1]$ and $v \in [0,1]$, $u + v = 1$, the

$$b^n(u,v) = \sum_{i+j=n} B_{ij}^n(u,v) P_{ij},$$

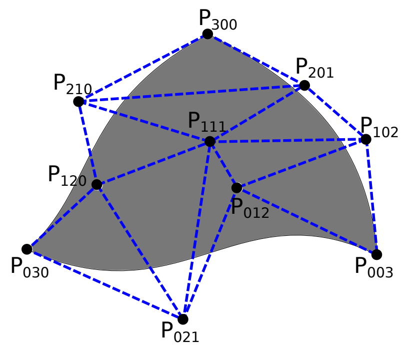
where $B_{ij}^n(u,v) = \frac{n!}{i!j!} u^i v^j$, $P_{ij} \in \mathcal{R}^2$ are the control points. Note that this and the following equations are in fact two equations corresponding to the two spatial coordinates.

Specifically, the cubic Bézier curve can be written in terms of the barycentric coordinates:

$$b^3(u,v) = \sum_{i+j=3} B_{ij}^3(u,v) P_{ij} = u^3 P_{03} + 3u^2 v P_{12} + 3uv^2 P_{21} + v^3 P_{30}.$$



(a) A cubic Bézier curve



(b) A cubic Bézier triangle

FIG. 3: A cubic Bézier curve and a cubic Bézier triangle.

Figure 3a gives an example of the cubic Bézier curve with its control polygon formed by four control points.

2.6 BÉZIER TRIANGLES

Univariate Bernstein polynomials are the terms of the binomial expansion of $[t + (1 - t)]^n$. In the bivariate case, a n-th order Bernstein polynomial is defined by

$$B^n_{\mathbf{i}}(\mathbf{u}) = \binom{n}{\mathbf{i}} u^i v^j w^k,$$

where

$$\mathbf{i} = \{i, j, k\}, \quad |\mathbf{i}| = n, \quad \mathbf{u} = \{u, v, w\},$$

$u \in [0, 1]$, $v \in [0, 1]$ and $w \in [0, 1]$ are the barycentric coordinates and $u + v + w = 1$. It follows the standard convention for the *trinomial coefficients* $\binom{n}{\mathbf{i}} = \frac{n!}{i!j!k!}$.

This leads to a simple definition of a Bézier triangle of degree n

$$\mathcal{T}^n(\mathbf{u}) = \sum_{i+j+k=n} B^n_{\mathbf{i}}(\mathbf{u}) P_{\mathbf{i}},$$

where the set of points $P_{\mathbf{i}}$ are *control points*, and the net N formed by points $P_{\mathbf{i}}$ is called *control net* of the Bézier triangle $\mathcal{T}^n$.

Specifically, the Bézier triangle of degree three can be written as

$$\begin{aligned}\mathcal{T}^3(\mathbf{u}) &= P_{300}u^3 + P_{030}v^3 + P_{003}w^3 \\ &\quad + 3P_{201}u^2w + 3P_{210}u^2v + 3P_{120}uv^2 \\ &\quad + 3P_{102}uw^2 + 3P_{021}v^2w + 3P_{012}vw^2 \\ &\quad + 6P_{111}uvw.\end{aligned}$$

Figure 3b gives an example of the cubic Bézier triangle with its control net formed by ten control points.

2.7 THE JACOBIAN

We explore the concept of a derivative of a coordinate transformation, which is known as the *Jacobian* of the transformation.

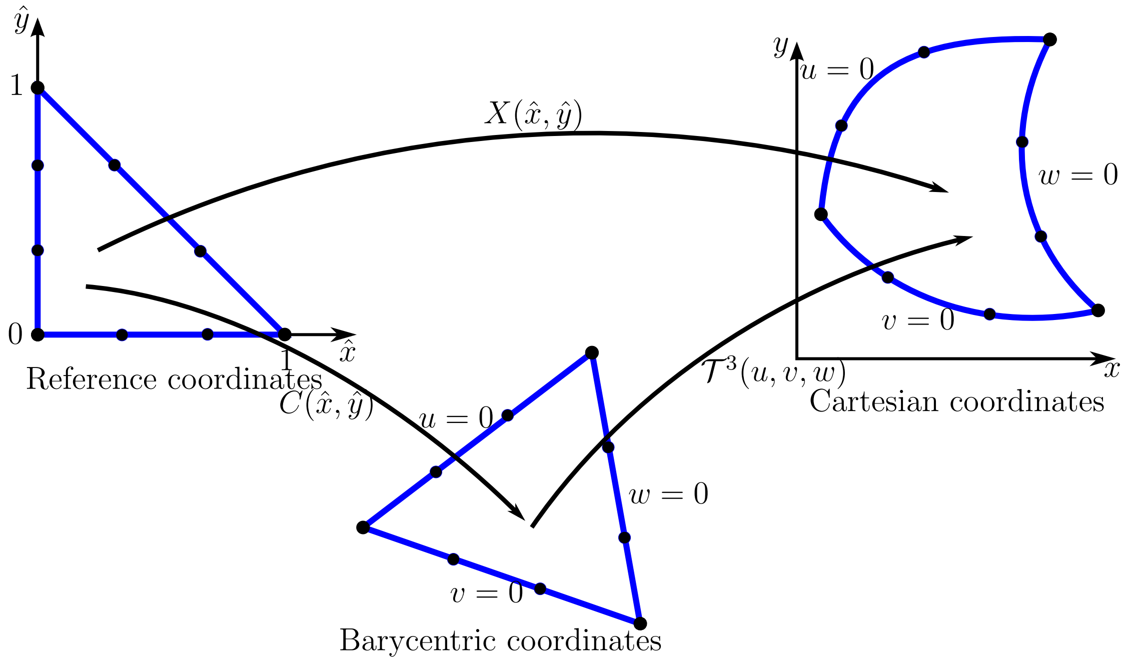


FIG. 4: Reference unit triangle in local coordinates $(\hat{x},\hat{y})$ and the mappings $\mathbf{X}(\hat{x},\hat{y})$, $\mathbf{C}(\hat{x},\hat{y})$ and $\mathcal{T}(u,v,w)$. A general principle for the transformations: an one-to-one correspondence between coordinates systems.

Because the cubic Bézier triangle is defined in terms of the barycentric coordinates (u,v,w) with the form:

$$\mathcal{T}^3(u,v,w) = \sum_{i+j+k=3} B^3_{ijk}(u,v,w)P_{ijk},$$

the reference triangle is first mapped to a triangle in barycentric coordinates (by the mapping $\mathbf{C}(\hat{x}, \hat{y})$) and then mapped to a curved triangle in global (x, y) coordinates (by the mapping $\mathcal{T}(u, v, w)$). This two-step mapping is presented in Figure 4.

The mapping should be bijective, because there should not be overlapped regions inside the element. This implies that the sign of the *Jacobian* of the transformation has to be strictly positive everywhere on this element. *Jacobian* is the determinant of the Jacobian matrix J which is defined by all first-order partial derivatives of the transformation:

$$J = \begin{bmatrix} \frac{\partial \mathcal{T}_x^n}{\partial \hat{x}} & \frac{\partial \mathcal{T}_x^n}{\partial \hat{y}} \\ \frac{\partial \mathcal{T}_y^n}{\partial \hat{x}} & \frac{\partial \mathcal{T}_y^n}{\partial \hat{y}} \end{bmatrix}.$$

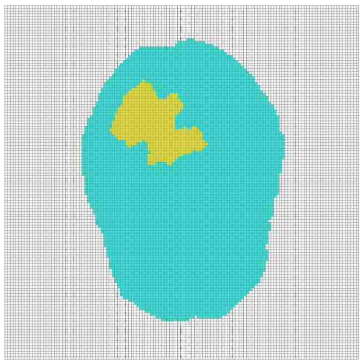
CHAPTER 3

TWO- AND THREE-DIMENSIONAL LINEAR TRIANGULAR/TETRAHEDRAL MESH CONSTRUCTION

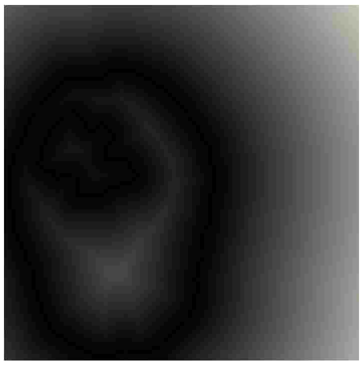
3.1 METHODOLOGY

The proposed algorithm is described as follows: the algorithm takes a two- or a three-dimensional multi-material image as its input. The user also specifies as input the target fidelity bounds (two-sided Hausdorff distance) and the desired angle lower bound. The angle lower bound should be less than or equal to the lower dihedral angle bound of the initial mesh (this bound is proved to be 19.47° in section 3.2.1). The algorithm outputs a quality mesh which is a good geometric and topological approximation of the underlying object. The algorithm first builds an octree from which it generates a waterproof surface mesh. The surface mesh is proved to be homeomorphic to the boundary of the image. The two-sided Hausdorff distance between the surface mesh and the boundaries of the image respects the user-specified fidelity bounds. Then the octree leaves are filled with high-quality tetrahedra which compose the volume mesh whose surface is exactly the same as the surface mesh. As the last step, the algorithm coarsens the mesh to a much lower number of elements while maintaining the quality, fidelity, and homeomorphism (the homeomorphism is proved in section 3.2.3). Algorithm 1 is a high-level description of the mesh generation algorithm.

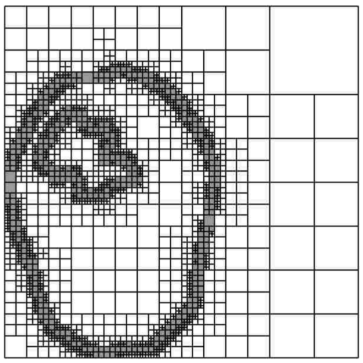
The main steps performed by the algorithm are illustrated in Figure 5. It shows a two-dimensional image being converted into a triangular mesh. The size of the image is 128×128 voxels. It defines a slice of a simplified human brain (shown in cyan) with a randomly selected region (which could be a tumor or a tissue with certain properties) (shown in yellow) against a white background.



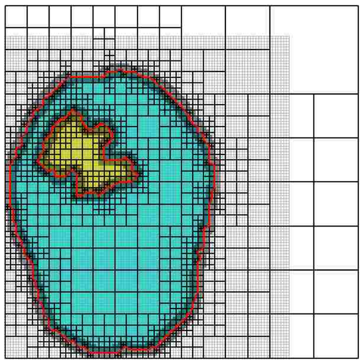
(a) The input two-dimensional image of size 128 $\times$ 128. The input angle bound is set to 19.47°, and the fidelity bounds are both set to two voxel side lengths



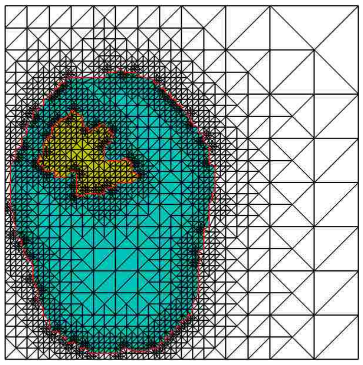
(b) Euclidean distance transform of the input image. Darker shades correspond to the voxels that are closer to image boundaries



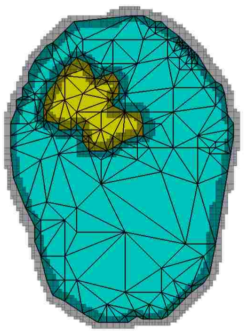
(c) The first quadtree $\mathcal{O}_1$. The leaves whose all corresponding pixels have the distance of EDT that are smaller than or equal to the fidelity bound are marked with grey color



(d) The second quadtree $\mathcal{O}_2$. The leaves were refined to meet the single manifold condition and the 2-to-1 rule. The red segments show the surface mesh that respects the fidelity condition, which is two-voxel side lengths in this example



(e) The fine mesh. 2678 triangles inside the object, 4408 triangles total. The minimum angle is 19.47°. The mesh boundary is exactly the same as the surface mesh in (d)



(f) The decimated mesh which maintains quality, fidelity and homeomorphism of the underlying object and the fidelity bound. 378 triangles inside the object, the outside triangles are removed.

FIG. 5: An illustration of the main steps performed by the algorithm.

3.1.1 CONSTRUCTION OF THE OCTREE

Our octree construction algorithm is shown in Algorithm 2 (the actual implementation is slightly more involved to support efficient data structures). The algorithm

Algorithm 1: A high-level description of the mesh generation algorithm

```
1  Algorithm: Construction($\mathcal{I}$, $\bar{\theta}$, $\bar{h}_1$, $\bar{h}_2$)
   Input  : $\mathcal{I}$ is a two- or a three-dimensional multi-material image containing $\Phi$,
            $\bar{\theta}$ is the lower bound on the minimum angle bound,
            $\bar{h}_1$ and $\bar{h}_2$ are the upper bounds on one-sided Hausdorff distances.
   Output: A tetrahedral mesh $\mathcal{M}$ that approximates $\mathcal{I}$'s boundary $\mathcal{B}$ well in terms
            of fidelity and topology and is composed of tetrahedra (triangles) whose
            dihedral (planar) angles are larger than or equal to $\bar{\theta}$.
2  Construct an octree $\mathcal{O}_1$ that completely encloses $\Phi$;
3  Extend $\mathcal{I}$ to the size of $\mathcal{O}_1$;
4  Compute the EDT on the extended $\mathcal{I}$;
5  Split $\mathcal{O}_1$ until no leaf contains both pixels that have the distance of EDT that are
     larger than MAX($\bar{h}_1$, $\bar{h}_2$) and pixels that have the distance of EDT that are
     smaller than or equal to MAX($\bar{h}_1$, $\bar{h}_2$);
6  Mark the leaves of $\mathcal{O}_1$ whose all corresponding pixels have the distance of EDT
     that are smaller than or equal to MAX($\bar{h}_1$, $\bar{h}_2$);
7  $\mathcal{O}_2$ = OCTREE_CONSTRUCTION($\bar{h}_1$, $\bar{h}_2$, $\mathcal{I}$, $\mathcal{O}_1$) ;        /* Algorithm 2 */
8  $\mathcal{M}$ = VOLUME_CONSTRUCTION($\mathcal{O}_2$) ;                   /* Algorithm 3 */
9  Find the connectivity among the tetrahedra of $\mathcal{M}$;
10 Assign colors to the tetrahedra using flood filling method;
11 $\mathcal{M}$ = DECIMATION($\mathcal{M}$, $\bar{\theta}$, $\bar{h}_1$, $\bar{h}_2$, $\mathcal{O}_1$) ;           /* Algorithm 4 */
12 return $\mathcal{M}$;
```

first constructs an octree (root) that completely encloses all the materials (except for the background voxels) from the bitmap. An extra space between the materials and the exterior boundaries of the octree should be equal to or larger than the user-specified fidelity bounds. We maintain a queue Q of octree leaves that are candidates for splitting. The queue is first initialized to contain the octree root only. For every leaf in Q, check if the leaf satisfies the simple manifold condition, and the leaf with any its neighbor differs at most by a factor of two in size. If the leaf failed either of the two conditions, push all neighbors into Q, and split the leaf into 8 children and push them into Q. Otherwise, generate surface triangles for the leaf. If the surface triangles do not form a topological disk (the image boundary in the leaf is a simple manifold), or the surface triangles do not respect the user-specified two-sided fidelity bound, push all neighbors into Q, and split the leaf and push the children into Q. The reason we push all neighbors of the leaf into Q is that it's likely that the leaf that after split and one of its neighbor do not differ at most by a factor of two in size. The algorithm terminates when Q is empty.

Waterproof surface mesh

When an octree leaf respect the single manifold condition, and the leaf with any its neighbor differ at most by a factor of two in size, the algorithm generates triangular patches from each octree leaf such that the union of all the patches forms a waterproof surface mesh. To generate the waterproof surface mesh, an approach reminiscent of the Marching Cubes (MC) algorithm [12] is used. MC processes the uniform-sized cubes of the rectilinear grid one by one, and evaluates cut function f (evaluates to positive or negative) given the vertices of each cube, then approximates the intersection of the isosurface with that cube using triangles from a pre-defined look-up table. In contrast, the proposed algorithm processes on an octree, and it iterates over the six faces of the leaves, applying a modified marching-squares-like algorithm. The selection of the appropriate stencil for each face of the leaf is determined by a key which is an ordered concatenation of label for each vertex of the face. These labels indicate whether a vertex is inside, outside, or directly on the image boundary. The vertices of a leaf can be evaluated to three labels: zero, positive, and negative. A vertex of a leaf evaluates to zero if the vertex is located exactly on the boundary between the materials. The positive label is assigned to the vertex lying inside the material, and the negative label is assigned to the vertex lying outside the material. Globally, assigning a vertex label is ambiguous, meaning that a vertex that is positive from one side is negative from another side since the vertex could be inside of one material but be outside of another. However, this problem can be bypassed by assigning the labels locally in each leaf. If a proper boundary leaf contains only two materials, choose any material and label the points inside it as positive, and label the points in the other material as negative. However, when a proper boundary leaf satisfies the simple manifold condition, it does not necessarily contain only two materials, it could contain 3 or more materials (though such cases are very rare). A pre-processing is done before initialing Q that the octree is split until no leaf contains more than two materials. Instead of initializing Q with octree root, initialize it with all the leaves after split with no more than two materials.

MC approximates the intersection of the isosurface using triangles from a look-up table defined based on a cube. However, for the proposed algorithm, the templates on cubes would be cumbersome, because there would be 3^8 cases in the look-up table in the uniform background grid case or even much more for the graded case. To construct such a huge table is labor-intensive and error-prone. Instead, an intersection

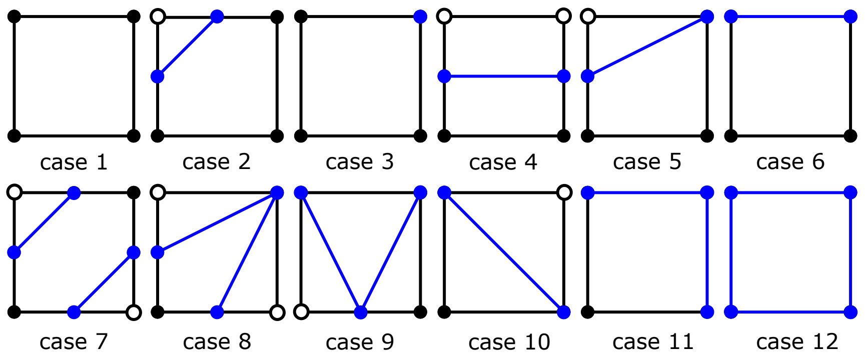
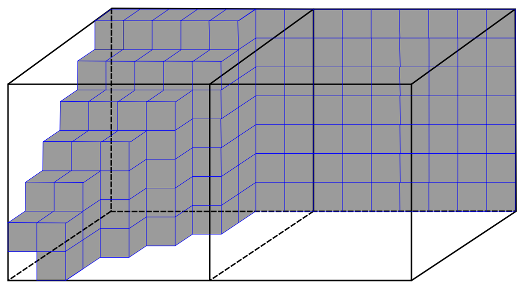
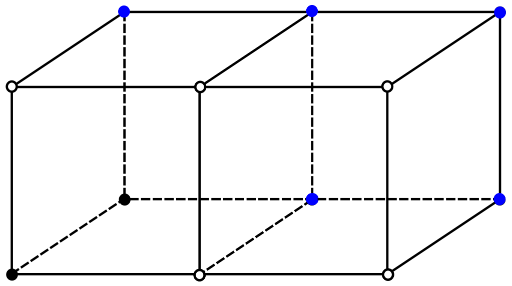


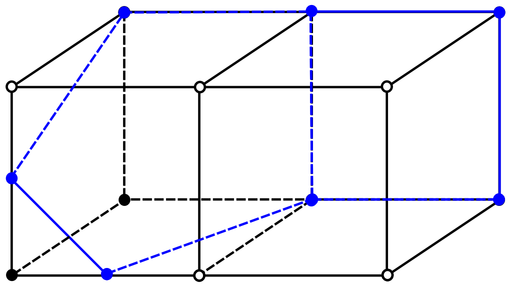
FIG. 6: All possible stencils for creating intersection vertices and intersection edges by grouping cases using symmetries. Black circles show the positive vertices, white circles show the negative vertices, and blue circles show the zero vertices. Blue segments show the intersection edges created by the algorithm.

edge look-up table is designed on the square for each of the cube faces, so there are totally only 3^4 entries in the table. In the boundary leaves, the vertices that sample mesh surface separating two materials are called *intersection vertex*. Any edge on the stencils that has two ends with opposite signs (positive and negative) contains at least one intersection vertex. We sample an intersection vertex on the center of each such edge. Vertices on the squares that labeled 0 are considered as intersection vertices as well. Within each stencil, the *intersection edges* separating vertices with a positive value from those with a negative value are defined by connecting the various intersection vertices. Figure 6 lists 12 stencils. Two different symmetries of the square reduce the stencils from 81 cases to 12 cases: cases that the vertex label in all corners reversed are grouped into one case, and rotationally symmetric cases are also grouped into one case. The figure is used for illustration purpose, 81-case stencils are used in the algorithm to avoid the ambiguity.

For each of the proper boundary leaves, the intersection edges on all the cube faces are generated by the intersection edge look-up table (partly shown in Figure 6). The union of the intersection edges, if forms one and only one closed chain, are called *intersection polygon* associated with the leaf. By connecting the vertices of the intersection polygon with the center of the cube, a 2-manifold with boundary is constructed (proved in section 3.2.3). For the non-proper boundary leaves, the intersection polygon is exactly the same as the boundary of the cube face if all voxels adjacent to this face are adjacent to another material (case 12 in Figure 6). In such cases, a diagonal of this cube face is used to generate the triangular patches,


(a) Step1: An octree is constructed enclosing all the materials. This figure shows a part of the octree (two leaves of the same size) with image boundary (grey)


(b) Step2: the signs of the cube vertices are identified


(c) Step3: intersection edges for cube faces is generated from the intersection edge look-up table. The intersection edges in each leaf form a closed loop

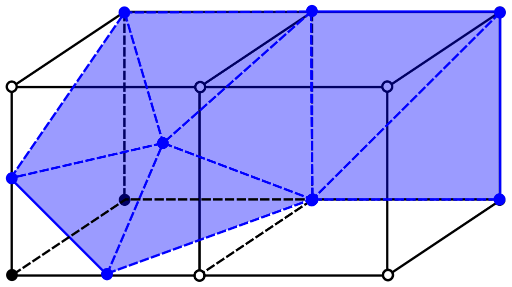
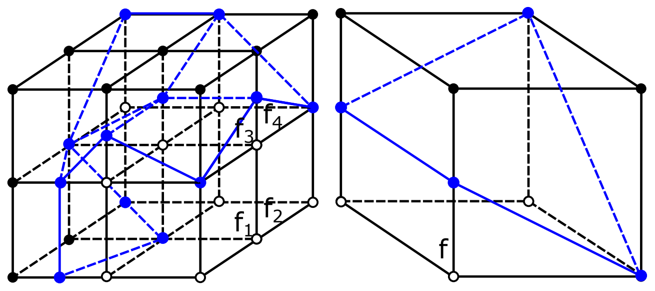

(d) Step4: triangular patches for each of the intersection edge loop is created

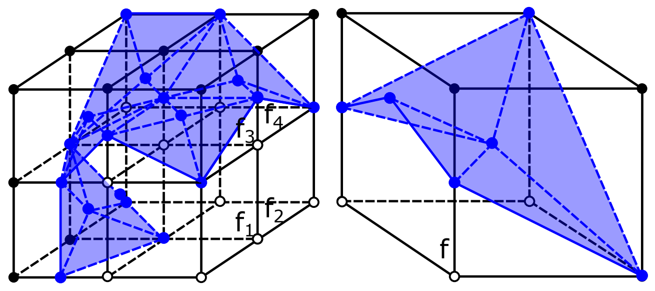
FIG. 7: An illustration of generating triangular patches to approximate the image boundary in two leaves of the same size. Black circles and white circles show the positive and negative vertices respectively, blue circles show the intersection vertices, blue edges on cube faces and blue triangles show the intersection edges and the triangular patches respectively.

resulting in two equally sized right triangles. In those leaves, the surface mesh is exactly the same as the image boundary. The steps of generating triangular patches approximating the image boundary are shown in Figure 7.

One important issue that needs extra concern is the consistency between neighboring leaves. Extracting the surface mesh from the octree representation is, however, more complicated than extracting it from the uniformly sized grid. If the background grid is composed of uniformly sized cubes, each cube can be processed independently without paying extra attention to the consistency. Since the same decision is made on both sides of a cube face, the extracted triangulation is guaranteed to be watertight. However, when octree leaves of different sizes meet, processing octree leaves



(a) If leaves are processed independently, the intersection edge defined by face f (the left face of the right cube) will not align with the intersection edges defined by the finer faces f_3 and f_4 (the right faces of the two top adjacent neighbors)

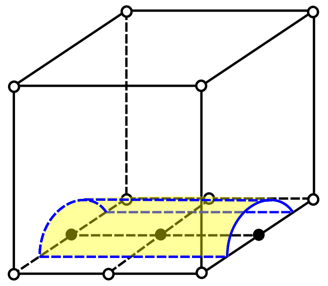


(b) Inconsistency solved by replacing the intersection edge defined by f with intersection edges defined by the finer faces f_3 and f_4

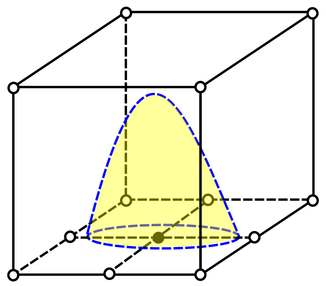
FIG. 8: An illustration of generating triangular patches to approximate the image boundary in a leaf whose face is adjacent to finer neighbors. Blue circles show the intersection vertices, blue segments on cube faces show the intersection edges and blue triangles show the triangular patches.

independently can result in gaps in the faces. Figure 8a shows an example of an inconsistent definition of edges across a face which is adjacent to four finer neighboring cubes. The inconsistency in the shared face of neighboring leaves leads to a "hole" in the surface mesh. In order to solve this problem, the octree leaves are processed in the order of their size, starting with the smallest. In the case that two face-adjacent leaves are of different sizes, the intersection edges from the shared faces of finer neighbors are duplicated to the coarser face (Figure 8b).

Two special cases need to be taken care of separately. One is that the boundary surface exhibits multiple intersections along an edge of a leaf (see Figure 9a), another is that the boundary surface exhibits no intersection along the edges of a leaf (see



(a) The piece of boundary surface exhibits multiple intersections along the edges of the leaf (the left and right edge of the bottom face). Those edges connect two equally signed corners and the midpoint is differently signed



(b) The piece of boundary surface exhibits no intersections along the edges of the leaf. For the bottom face, the four corners all agree in sign but disagree with the center of the face

FIG. 9: Two examples illustrate that the piece of boundary surface fulfills the single manifold condition, but exhibits multiple intersections or no intersection along the edges of the leaf, leading to different topology. These leaves need to be split.

Figure 9b), but both of the two cases fulfill the single manifold condition. In those cases there is no cube edge has opposite signs, and the number of intersection edges in such cubes is zero. This means that no mesh surface is generated in the cube, but the cube does contain one piece of image boundary, leading to a different topology. Same topological criteria warrant the subdivision of the cube: the leaf needs to be refined if the intersection edges form zero closed chain.

Two-sided Hausdorff distance

The mesh has to provide a close geometric approximation of the object shape. We measure the closeness by the fidelity tolerance, quantified by the two-sided Hausdorff distance from the mesh to the image and the image to the mesh.

To measure the Hausdorff distance from the surface mesh to the boundaries of the image, we compute the Euclidean distance transform (EDT) [47] of the extended image, i.e., the image is padded with imaginary background voxels (or truncated of the extra background voxels) to the size of the octree root node. A Euclidean distance transform of an image is an assignment to every voxel of a distance to the nearest boundary of the image in Euclidean metric (see Figure 5b as an example, where darker shades correspond to the voxels that are closer to image boundaries). The fidelity zone is the octree (quadtree) leaves whose all corresponding voxels have the distance of EDT that are smaller than or equal to the fidelity bound. To compute the fidelity zone, the second octree is constructed, and is split until the EDT values of all the voxels in each leaf are either larger than the input fidelity bound, or smaller than or equal to the input fidelity bound. In the implementation, we make the two octrees one with common ancestor for efficiency and storage purpose. Then the leaves whose all corresponding pixels have the distance of EDT that are within the input fidelity tolerance are marked, composing the fidelity zone (see the leaves in transparent gray in Figure 5c). If one triangle of the surface mesh intersects at least one of the leaves marked as outside the fidelity zone, the fidelity condition is violated and the leaf is split.

To measure the other side, the shortest distance is computed from each image boundary vertex in the leaf to the triangular patches of the surface mesh. If one of the image boundary vertices has a shortest distance larger than the fidelity tolerance, the fidelity condition is violated and the leaf is split. If the fidelity condition is satisfied in the leaf, each of the image boundary vertices is assigned to its closest triangular patch for later use in the mesh decimation step.

3.1.2 FILLING IN THE OCTREE

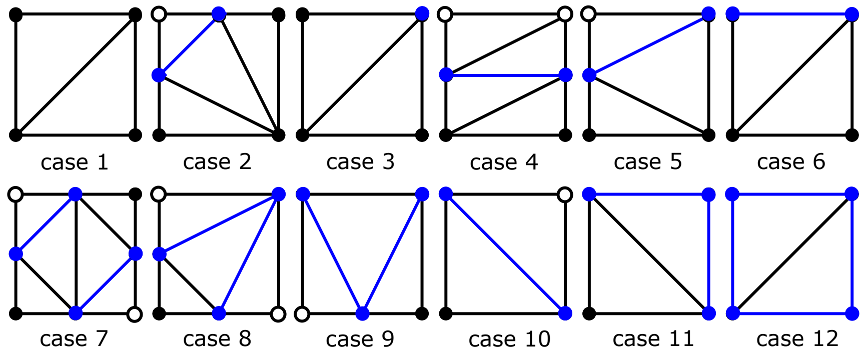
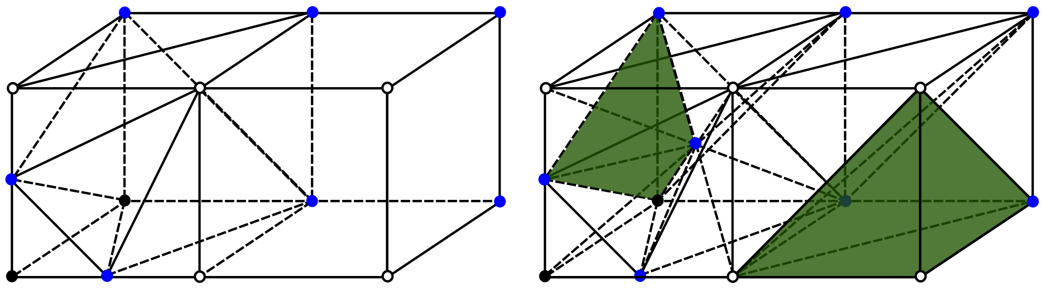


FIG. 10: The stencils of the triangle look-up table that correspond to the stencils of the intersection edge look-up table for creating triangles on cube faces. The blue segments show the intersection edges. The template triangles respect the intersection edges.

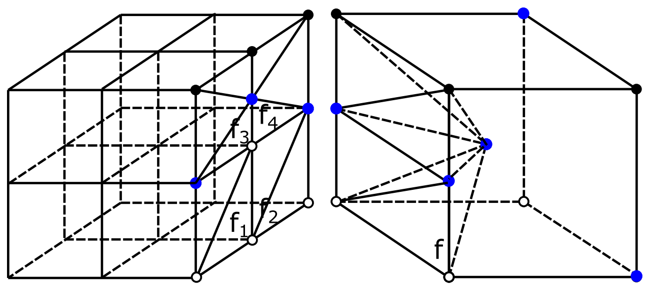
Now that the octree satisfies the single manifold condition, 2-to-1 rule and resulting surface mesh respects the user-specified fidelity bounds, the volume mesh generation can be facilitated, by subdividing the octree leaves into tetrahedra. See Algorithm 3. For each proper boundary leaf, the faces are triangulated according to the second pre-defined triangle look-up table in Figure 10 (here we show 12 cases corresponding to the stencils of the intersection edge look-up table in Figure 6, but we use 81 cases table in the proposed algorithm). The template triangles of the triangle look-up table respect the intersection edges of the intersection edge look-up table, i.e., the intersection edges should be the edges of the template triangles. For non-proper boundary leaves and non-boundary leaves, if one edge of the leaf is split by a middle point, triangulate each face of the leaf which contains the edge by introducing the center of the face and connecting it with all vertices on the face, and triangulate other faces into two triangles by introducing a diagonal to it. Once all faces of the octree leaf are triangulated, the leaf is tetrahedralized by constructing a tetrahedron for each triangle through connecting it to the center of the cube. If no edge of the cube is split by a midpoint, the central point of the cube is not introduced, the leaf is tetrahedralized into six tetrahedra. Figure 11 illustrates the tetrahedralization steps.


(a) Step1: triangulating cube faces from the triangle look-up table

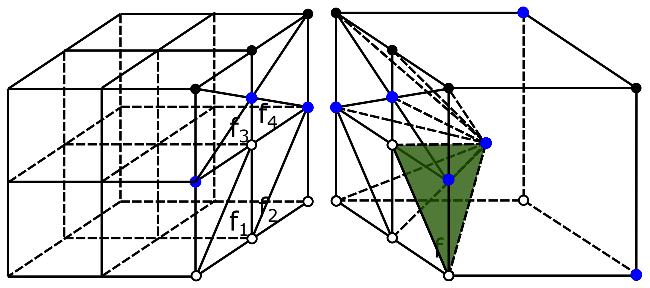
(b) Step2: tetrahedralizing the leaves by connecting each triangle on the cube faces to the center of the cube. If no edge of the cube is split by a midpoint, the leaf is tetrahedralized into six tetrahedra.

FIG. 11: An illustration of tetrahedralization in two leaves of the same size. Black circles show the positive vertices, white circles show the negative vertices, blue circles show the zero vertices. Black edges on cube faces show how the faces are triangulated by the triangle look-up table. Two tetrahedra are marked by green.

Similar to the surface construction, the consistency between neighboring leaves still needs to be considered when the octree is tetrahedralized. When octree leaves at different resolutions meet, an inconsistent definition of triangles across the shared face occurs. An example is illustrated in Figure 12a. To solve this problem, the octree leaves are processed from the smallest to the largest. In the case that a face of



(a) If leaves are processed independently, triangles defined by face f (the left face of the right cube) will not align with triangles defined by face f_1, f_2, f_3 and f_4 (the right faces of the four adjacent finer neighbors of the right cube)



(b) Inconsistency solved by replacing the triangles defined by f with triangles defined by f_1, f_2, f_3 and f_4

FIG. 12: An illustration of tetrahedralization in a leaf whose face is adjacent to finer neighbors. Black circles show the positive vertices, white circles show the negative vertices, blue circles show the zero vertices. Black edges on cube faces show how the faces are triangulated.

a leaf is adjacent to four finer neighbors, the triangles defined from the shared faces of finer neighbors are duplicated to the coarser face, see Figure 12b for the illustration.

If a face of a cube is shared by four finer neighbors, but another face of the cube which is adjacent to it is shared by a neighbor of the same size, a hanging point may appear. To eliminate the hanging points, if an edge of a triangle on a cube face has a hanging point in the middle, the triangle is re-triangulated into two triangles by connecting the hanging point with the vertex of the triangle which is opposite to the edge on which the hanging point is lying.

When all the leaves are filled with tetrahedra, the connectivity among the tetrahedra are identified, i.e., identifying face-adjacent tetrahedra for later use in the decimation procedure.

The octree (quadtree) leaves and tetrahedra (triangles) are assigned colors based on their location with respect to the materials in the image. If a leaf contains only one material, it derives the color from the block of voxels that it encloses. Each tetrahedron (triangle) in such leaves derives its color from the leaf. For proper boundary leaves, we assign the color of tetrahedra (triangles) using the flood-fill method. The flood-fill method starts with a tetrahedron of a known color, and assign the same color to its neighours whose color is unknown if the shared face of the two tetrahedra is not a mesh boundary face. Therefore, all tetrahedra (triangles) are classified with respect to the boundary of underlying materials.

3.1.3 MESH DECIMATION

Similar to the LD method [16], the vertex removal operation is used to coarsen the mesh. A vertex is merged to a destination vertex if the vertex and the edge between the vertex and its destination are removed from the mesh. As a consequence, all tetrahedra (triangles) incident upon the removed edge are also removed from the mesh. The remaining edges that were incident upon the vertex became incident upon its destination. To achieve this goal, a queue of mesh vertices is maintained for merging. Initially, all mesh vertices are added to the queue. Then the vertices are removed from the queue and are evaluated if they pass the check for a merge. After the initialization, only the vertices whose adjacent vertices were merged can be added to the queue. The decimation procedure terminates when none of the vertices in the queue passes the check for a merge and the queue becomes empty. The detailed operation steps see Algorithm 4, here we only discuss the merging conditions. A vertex cannot be merged along an edge to another vertex if it violates the following requirements:

1. The quality requirement, i.e., if at least one of the newly created elements, as a result of a sequence of merges, is inverted or its dihedral angle is smaller than the input quality angle bound, the merge is discarded.

2. The fidelity requirement, i.e., if at least one of the newly created mesh boundary faces (edges in two-dimensional case and triangles in three-dimensional case) has at least one-sided Hausdorff distance larger than the input fidelity bound, the merge is discarded. This check consists of two parts, for each of the one-sided Hausdorff distances. To evaluate the Hausdorff distance from the boundaries of the submesh to the boundaries of the corresponding material, if one of the newly created boundary

faces intersects at least one of the leaves marked as outside the fidelity tolerance, the fidelity requirement is violated. To evaluate the Hausdorff distance from the boundary of each material to the boundary of the corresponding submesh, for each boundary face we maintain a cumulative list of the image boundary vertices. The image boundary vertices were first assigned to the boundary faces during the fidelity check of the octree construction. Each image boundary vertex was assigned to the closest mesh boundary face. If at least one of the image boundary vertices, as a result of a sequence of merges, is further away from its closest mesh boundary face than the fidelity tolerance, the merge is discarded. After each merge, the image boundary vertices of the mesh boundary faces which are incident upon the merged vertex are reassigned to the closest newly created boundary faces.

3. The topological equivalence requirement, i.e., if the homeomorphism is not maintained during operation of a merge, the merge is discarded. We apply the following rules to maintain the original structure of both the exterior and interior boundaries:

(1) Boundary vertices merge to boundary vertices. Applying this rule is not necessary for the topological correctness, but it helps create a smoother mesh boundary after decimation.

(2) Do not merge if a tetrahedron (triangle) has all edges are boundary edges.

(3) The topological correctness in each merge also relies on the single manifold condition. Instead of checking if the single manifold condition is fulfilled in every single leaf during the surface mesh construction, this check is conducted on a union of several octree leaves. A cumulative list of the octree leaves that each boundary face lies in is maintained. The merge happens if the union of octree leaves of the boundary faces the merged vertex and the destination incident upon respects the single manifold condition. Suppose a boundary vertex a merges to the destination boundary vertex b. F_a and F_b denote the set of boundary faces that a and b incident upon respectively. Notice that $F_a \cap F_b$ may not be empty. Let F_{ab} denote $F_a \cup F_b$, and let $L_{F_{ab}}$ denote the set of octree leaves that the faces of F_{ab} lie in. If all the leaves of $L_{F_{ab}}$ as a whole satisfy the single manifold condition, the merge operation can be executed. Specifically, in the two-dimensional case, all the image boundary edges in leaves of $L_{F_{ab}}$ form a simple chain on which only the first and the last image boundary vertices have only one adjacent image boundary edge, all the other boundary vertices have two image boundary edges adjacent to it. Two examples are shown in Figure 13

for an illustration. In the three-dimensional case, the union of the image boundary faces in leaves of $L_{F_{ab}}$ form a simple sheet on which the 1-degree image boundary edges form a cycle and all the other image boundary edges are 2-degree edges. (4) Each tetrahedron keeps the original color after it changes shape due to the vertex merge. As a result, all tetrahedra (triangles) are correctly classified with respect to the underlying materials after decimation.

3.2 GUARANTEES

The algorithm achieves the following mathematical guarantees on the output mesh: (1) the tetrahedra of the interior mesh have good dihedral angles, (2) the boundaries of the output mesh is geometrically close to the boundaries of the materials, and (3) the boundaries of the output mesh are homeomorphic to the boundaries of the materials.

3.2.1 DIHEDRAL ANGLES

Theorem 3.2.1. *All dihedral angles in the mesh produced by Algorithm 1 are above a user-specified lower bound which can be set to any value up to* $19.47°$.

Proof. The key to proving theorem 3.2.1 is to prove that the bound on the dihedral angles before mesh decimation is $19.47°$. Since there is only a finite number of stencils, and the mesh vertices have a finite number of locations in the octree leaf (either on the corners or the center of the leaf, or on the quarters of the leaf edges), a minimum dihedral angle of $19.47°$ is obtained by analyzing all possible resulting leaf triangulations; see Figure 14. So the user-specified lower bound can be set to any value up to $19.47°$. During the decimation, if at least one newly created angle is smaller than the input quality angle bound, the merge is discarded. Therefore, the lower bound on the dihedral angles produced by our algorithm are correct as written. □

3.2.2 GEOMETRIC FIDELITY

Since the centers of proper boundary leaves are located on the mesh surface, they are considered as intersection vertices. By the process that the triangular patches in these leaves are constructed, the intersection vertices always connect to intersection vertices. By inspection of the intersection edge look-up table, the proposed algorithm never connects a vertex inside the image boundary (labeled +) to one outside (labeled -), therefore the mesh respects the image boundary. Furthermore, the boundary of the mesh produced by the algorithm approximates the image boundary by two-sided Hausdorff distance bound both before decimation and after decimation, therefore, every point on the mesh surface is geometrically close to image boundary.

3.2.3 TOPOLOGICAL FIDELITY

The following lemmas help guarantee that the surface meshes of our algorithm are topologically equivalent to the image boundary.

Lemma 3.2.2. *The surface triangulation $\mathcal{S}_L$ in each boundary leaf L is a 2-manifold with boundary.*

Proof. The proof distinguishes two cases: the first case is on proper boundary leaves, and the second case is on boundary leaves that are not proper.

– The intersection polygon associated with any proper boundary leaf L is a closed chain composed of the intersection edges on faces of L. Filling the intersection polygon with triangles by connecting the intersection edges with the center of the cube, the surface triangulation $\mathcal{S}_L$ is obtained. This cone-like triangular patch $\mathcal{S}_L$ satisfies the definition of a 2-manifold with boundary.

– By the algorithm, in all the non-proper boundary leaves, the intersection polygon is exactly the same as the boundary of the cube face (see Figure 4. case 12). A diagonal of this cube face is used to generate the triangular patches, resulting in two equally sized right triangles. In this case, the surface triangulation $\mathcal{S}_L$ is exactly the same as the image boundary $\mathcal{B}_L$. Similarly, by the definition, this square patch $\mathcal{S}_L$ is a 2-manifold with boundary.

□

The following lemma proves the watertight property of the surface mesh.

Lemma 3.2.3. *The surface mesh generated by the algorithm is watertight.*

Proof. The polygons must satisfy two properties to result in a watertight surface mesh: the extracted intersection edges on cube faces must be closed loops, and each intersection edge must be shared exactly twice. Since the intersection polygon associated to any boundary leaf is a closed chain composed of the intersection edges on faces of the leaf, the first property is satisfied.

To prove the second property, for the intersection edges lying on the cube face, there are two cases. The first case is that when an intersection edge is lying on the cube face which is adjacent to a neighbor of the same size. Since the same decision is made on both sides of the cube face, the intersection edge is shared by the two triangles defined by the two ends of the intersection edge and the center of each of the neighboring cubes. The second case is that when an intersection edge is lying on the cube face which is adjacent to four finer neighbors. Since the intersection edge is obtained by duplicating the corresponding intersection edge from one of the shared faces of the finer neighbors, it is shared exactly twice. Thus, the second property is fulfilled. Therefore, the surface mesh of our algorithm is guaranteed to be watertight. □

Lemma 3.2.4. *Let $\mathcal{B}$ be the image boundary, and $\mathcal{S}$ be the mesh boundary before decimation. $\mathcal{B} \approx \mathcal{S}$.*

Proof. Let L be any boundary leaf. By the simple manifold condition, $\mathcal{B}_L$ is a 2-manifold with boundary. By Lemma 3.2.2, $\mathcal{S}_L$ is a 2-manifold with boundary. Thus, $\mathcal{B}_L \approx \mathcal{S}_L$. Since the surface mesh generated by the algorithm is watertight (Lemma 3.2.3), $\mathcal{B} \approx \mathcal{S}$. □

Lemma 3.2.5. *Under the decimation rule of the algorithm, a 2-manifold with boundary with at least two triangles after a merge from one of its vertices to an adjacent vertex is a 2-manifold with boundary.*

Proof. In this proof, the vertices refer to the vertices of the triangles that compose the 2-manifold with boundary. We prove this lemma by case analysis.

- An interior vertex merges to an interior vertex, see Figure 15a. After the merge the boundary of the 2-manifold with boundary is unchanged.
- An interior vertex merges to a boundary vertex, see Figure 15b. Similar to the previous case, the boundary of the 2-manifold with boundary is unchanged.
- A boundary vertex merges to an interior vertex. In this case, the boundary of the 2-manifold with boundary is changed, but after the merge, it still is a

2-manifold with boundary. In fact, we omit this case in the decimation rule so that after merge we have smoother boundaries.

– A boundary vertex merges to a boundary vertex, see Figure 15c. In this case, the number of edges of the boundary of the 2-manifold with boundary is decreased by one, but it doesn't create a new boundary or the boundary is vanished (the number of boundaries is one). Moreover, by the decimation rule, the 2-manifold with boundary never became degenerated or split to two 2-manifolds with boundary that incident on one common vertex, thus after the merge, the topology is unchanged.

Due to the quality requirement, the merge is discarded if a newly created element is inverted, the 2-manifold with boundary after a merge won't lead to a self-intersecting boundary. By enumerating all the four cases, we proved that a 2-manifold with boundary after a merge is a 2-manifold with boundary under the decimation rule of the algorithm. □

Theorem 3.2.6. *The boundary of the mesh is homeomorphic to the boundary of the input image.*

Proof. We prove this theorem by induction. We show that the basis step is true, i.e., there is a homeomorphism between the image boundary and the surface mesh before decimation. This step is proved by Lemma 3.2.4.

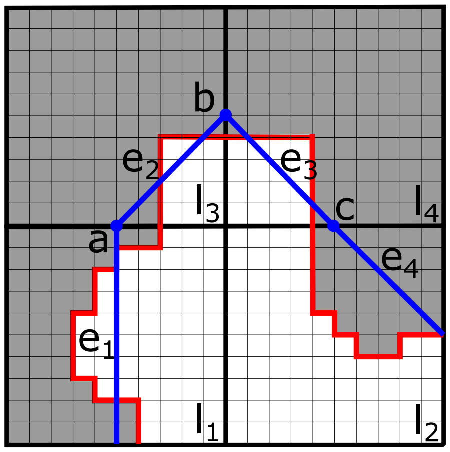
We prove the inductive step is also true. For the inductive hypothesis we assume that after the k-th merge, the boundary of the mesh is homeomorphic to the boundary of the image. Suppose in the $(k+1)$-th merge, a boundary vertex a merges to boundary vertex b. The inductive hypothesis implies that in the leaves of $L_{F_{ab}}$ if the image boundary is a 2-manifold with boundary, the mesh boundary is a 2-manifold with boundary. This means that if the leaves of $L_{F_{ab}}$ satisfy the simple manifold condition, the image boundary faces in the leaves of $L_{F_{ab}}$ is a 2-manifold with boundary as well. According to Lemma 3.2.5, the mesh boundary is a 2-manifold with boundary after vertex a merged to vertex b. Therefore, after $(k+1)$-th merge, the boundary of the mesh is homeomorphic to the boundary of the image. □

Algorithm 2: A high-level description of the octree construction algorithm.

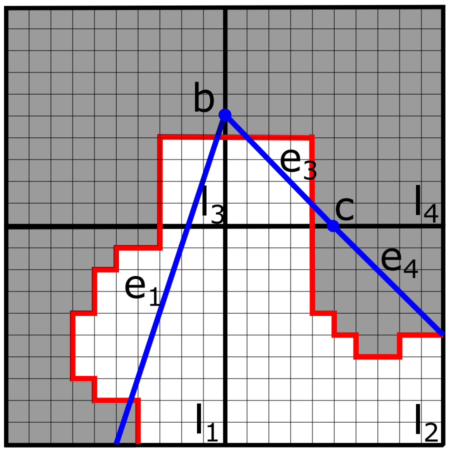
```
Algorithm: octree_construction($\bar{h}_1$, $\bar{h}_2$, $\mathcal{I}$, $\mathcal{O}_1$)
Input  : $\bar{h}_1$ and $\bar{h}_2$ are the upper bounds on one-sided Hausdorff distances,
         $\mathcal{I}$ is a two- or a three-dimensional multi-material image containing $\Phi$,
         $\mathcal{O}_1$ is the octree with the leaves marked within MAX($\bar{h}_1$, $\bar{h}_2$).
Output: An octree from which the surface mesh $\mathcal{S}$ satisfies fidelity requirement $\bar{h}_1$ and $\bar{h}_2$
         and are homeomorphic to $\mathcal{I}$'s boundary $\mathcal{B}$.
$\mathcal{S} = \emptyset$;
Construct an octree root $\mathcal{O}_2$ that completely encloses $\Phi$;
Initialize a queue $Q$ to the octree root $\mathcal{O}_2$;
while $Q \neq \emptyset$ do
    Pick $node \in Q$;
    $Q \longleftarrow Q \setminus \{node\}$;
    if !SINGLE_MANIFOLD_CONDITION(node) $\vee$ !BALANCED(node) then
        $Q \longleftarrow Q \cup Neis$ ;          /* Let $Neis$ be the set of $node$'s neighbors */
        Split node;
        $Q \longleftarrow Q \cup C$ ;               /* Let $C$ be the set of $child_1 \ldots$ $child_8$ */
    else
        if node is a proper boundary leaf then
            for each vertex $x$ of node do
                Compute sign of $x$ ;  /* determine whether sign of $x$ is positive,
                 negative or zero */
            end
            $\mathcal{E} = \emptyset$ ;            /* let $\mathcal{E}$ be the intersection edge set of node */
            for each face $f$ in node do
                $index \longleftarrow$ INDEX(SIGN($x_1$), SIGN($x_2$), SIGN($x_3$), SIGN($x_4$)) ;  /* $x_1$, $x_2$,
                 $x_3$ and $x_4$ are the four vertices of $f$ */
                $\mathcal{E} \longleftarrow \mathcal{E} \cup$ TABLE1_LOOKUP($index$);
            end
            if the intersection edges in $\mathcal{E}$ form a closed loop then
                $\mathcal{S} \longleftarrow \mathcal{S} \cup$ PATCHING($\mathcal{E}$) ; /* function PATCHING($\mathcal{E}$) introduces the
                 center of $l$ and connects it with the two ends of each
                 intersection edge in $\mathcal{E}$ */
            else
                $Q \longleftarrow Q \cup Neis$;
                Split node;
                $Q \longleftarrow Q \cup C$;
            end
        else
            for each face $f$ of node do
                if all pixels adjacent to $f$ are adjacent to another material then
                    generate two triangles using the diagonal with boundaries of $f$;
                end
            end
        end
    end
    else if !FIDELITY_SATISFIED(node, $\mathcal{S}$, $\mathcal{O}_1$, $\bar{h}_1$, $\bar{h}_2$) then
        $Q \longleftarrow Q \cup Neis$;
        Split node;
        $Q \longleftarrow Q \cup C$;
end
return $\mathcal{O}_2$;
```

Algorithm 3: A high-level description of the volume meshing algorithm

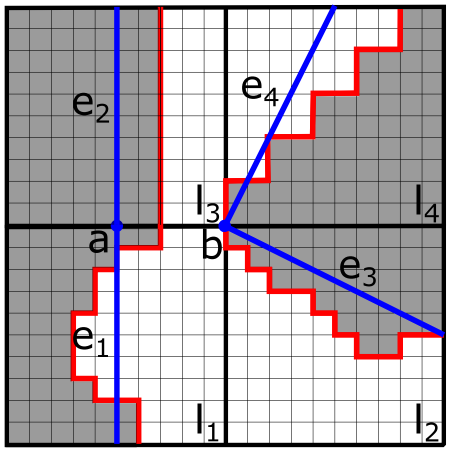
```
1  Algorithm: volume_construction($\mathcal{O}_2$)
   Input  : $\mathcal{O}_2$ is an octree from which a surface mesh $\mathcal{S}$ is constructed.
   Output: A tetrahedral mesh $\mathcal{M}$ that approximates $\mathcal{I}$'s boundary $\mathcal{B}$ well in terms
           of fidelity and topology and is composed of tetrahedra (triangles) whose
           dihedral (planar) angles are larger than or equal to 19.47°.
2  $\mathcal{M} = \emptyset$;
3  for each leaf $l$ in $\mathcal{O}_2$ do
4      if $l$ is a proper boundary leaf then
5          $\mathcal{T} = \emptyset$ ;              /* let $\mathcal{T}$ be the triangle set of faces of $l$ */
6          for each face $f$ in $l$ do
7              $index \longleftarrow$ INDEX(SIGN($x_1$), SIGN($x_2$), SIGN($x_3$), SIGN($x_4$)) ;    /* $x_1$,
                $x_2$, $x_3$ and $x_4$ be the four vertices of $f$ */
8              $\mathcal{T} \longleftarrow \mathcal{T} \cup$ TABLE2_LOOKUP($index$);
9          end
10         $\mathcal{M} \longleftarrow \mathcal{M} \cup$ TETRAHEDRALIZATION($\mathcal{T}$) ;                  /* function
            TETRAHEDRALIZATION($\mathcal{T}$) introduces the center of $l$ and
            connects it with the vertices of the triangles in $\mathcal{T}$ */
11     else
12         if at least one edge of $l$ is split by a middle point then
13             $\mathcal{T} = \emptyset$;
14             for each face $f$ in $l$ do
15                 if at least one edge of $f$ is split by a middle point then
16                     $\mathcal{T} \longleftarrow \mathcal{T} \cup$ TRIANGULATION_1($f$) ;            /* function
                        TRIANGULATION_1($f$) triangulates $f$ by introducing
                        the center of $f$ and connecting it with all vertices
                        on $f$ */
17                 else
18                     $\mathcal{T} \longleftarrow \mathcal{T} \cup$ TRIANGULATION_2($f$) ;            /* function
                        TRIANGULATION_2($f$) triangulates $f$ into two
                        triangles by introducing an diagonal to it */
19                 end
20             end
21             $\mathcal{M} \longleftarrow \mathcal{M} \cup$ TETRAHEDRALIZATION($\mathcal{T}$);
22         else
23             $\mathcal{M} \longleftarrow \mathcal{M} \cup$ TETRAHEDRALIZATION($l$) ; /* TETRAHEDRALIZATION($l$)
                triangulates $l$ into six tetrahedra */
24         end
25     end
26 end
27 return $\mathcal{M}$;
```



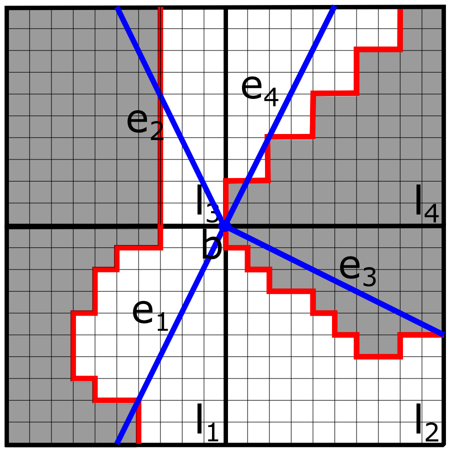
(a) $F_a = \{e_1, e_2\}$, $F_b = \{e_2, e_3\}$, $F_{ab} = F_a \cup F_b = \{e_1, e_2, e_3\}$, and $L_{F_{ab}} = \{l_1, l_3, l_4\}$. Because the image boundary edges (red segments) in $L_{F_{ab}}$ form a simple chain, a can merge to b



(b) In (a), after a merged to b, e_2 is removed from the mesh. The set of octree leaves in e_1 became $\{l_1, l_3\}$. The topology of the mesh boundary is still the same as the topology of the image boundary



(c) $F_a = \{e_1, e_2\}$, $F_b = \{e_3, e_4\}$, $F_{ab} = F_a \cup F_b = \{e_1, e_2, e_3, e_4\}$, and $L_{F_{ab}} = \{l_1, l_2, l_3, l_4\}$. Because the image boundary edges (red segments) in $L_{F_{ab}}$ do not form a simple chain, a cannot merge to b



(d) In (c), if a is merged to b, the topology of the mesh boundary is not the same as the topology of the image boundary

FIG. 13: An illustration of the topological equivalence requirement during decimation. The red segments represent the image boundaries, and the blue segments represent the mesh boundaries. a is the merged vertex, b is its destination.

Algorithm 4: A high-level description of the decimation algorithm

```
1  Algorithm: Decimation($\mathcal{M}$, $\bar{\theta}$, $\bar{h}_1$, $\bar{h}_2$, $\mathcal{O}_1$)
   Input   : $\mathcal{M}$ is the initial mesh,
             $\bar{\theta}$ is the lower bound on the minimum angle bound,
             $\bar{h}_1$ and $\bar{h}_2$ are the upper bounds on one-sided Hausdorff distances,
             $\mathcal{O}_1$ is the octree with the leaves marked as the fidelity zone.
   Output: A tetrahedral mesh $\mathcal{M}$ that approximates $\mathcal{I}$'s boundary $\mathcal{B}$ well in terms
             of fidelity and topology and is composed of tetrahedra (triangles) whose
             dihedral (planar) angles are larger than or equal to $\bar{\theta}$.
2  Initialize a queue $Q$ to the set of all vertices in $\mathcal{M}$;
3  while $Q \neq \emptyset$ do
4      Pick $\mathbf{v}_i \in Q$;
5      $Q \longleftarrow Q \setminus \{\mathbf{v}_i\}$;
6      Find $A = \{\mathbf{v}_j\}$ the set of vertices adjacent to $\mathbf{v}_i$;
7      for each $\mathbf{v}_j \in A$ do
8          if Quality($\mathbf{v}_j$, $\bar{\theta}$) $\wedge$ Fidelity($\mathbf{v}_j$, $\mathcal{O}_1$, $\bar{h}_1$, $\bar{h}_2$) $\wedge$ Topology($\mathbf{v}_j$, $\mathcal{M}$)
            then
9              Merge $\mathbf{v}_i$ to $\mathbf{v}_j$, update $\mathcal{M}$;
10             $Q \longleftarrow Q \cup A$;
11             break;
12         end
13     end
14 end
15 return $\mathcal{M}$;
```

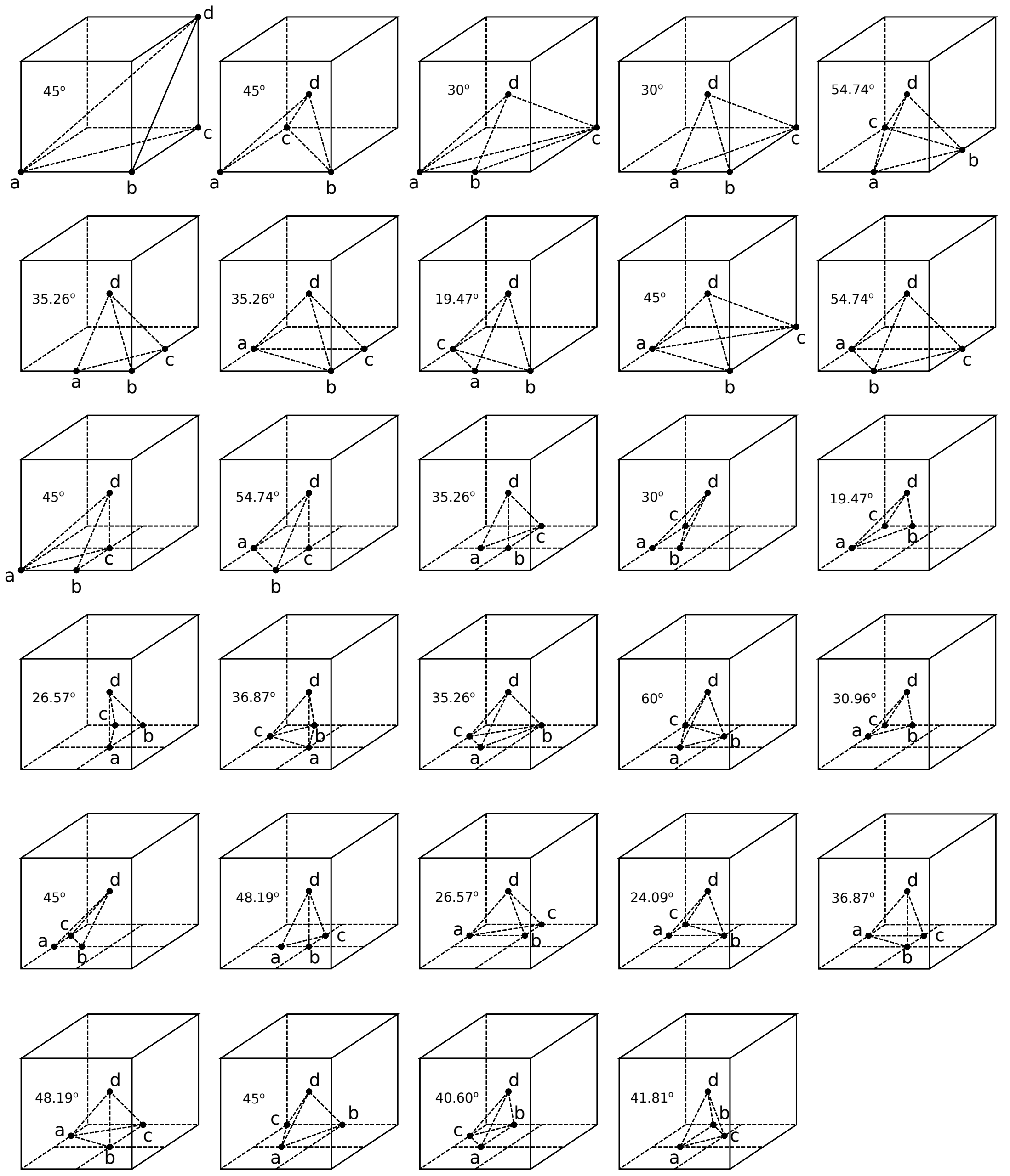
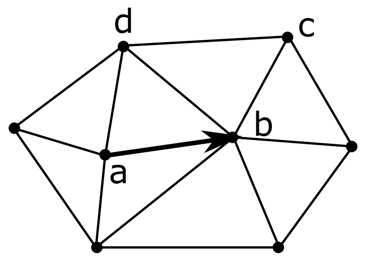
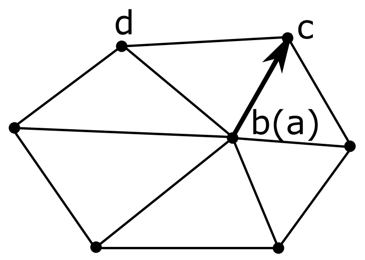


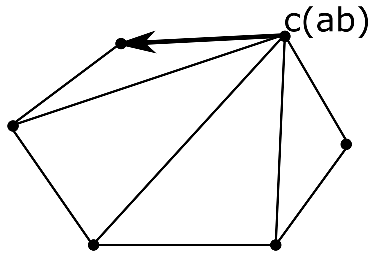
FIG. 14: All possible shapes of the initial tetrahedra (abcd) filling a cubic leaf of the octree, up to symmetry. The smallest dihedral angles are listed on the figure. Therefore, 19.47° is the lower bound for the dihedral angle in the initial three-dimensional tetrahedralization of the octree.



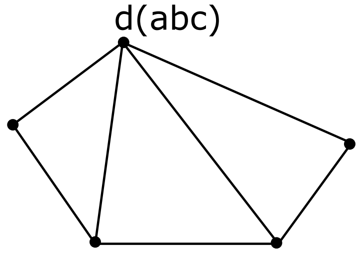
(a) A topological disk before vertex merge operation.



(b) An interior vertex a merged to an interior vertex b on the topological disk.



(c) An interior vertex b merged to a boundary vertex c on the topological disk.



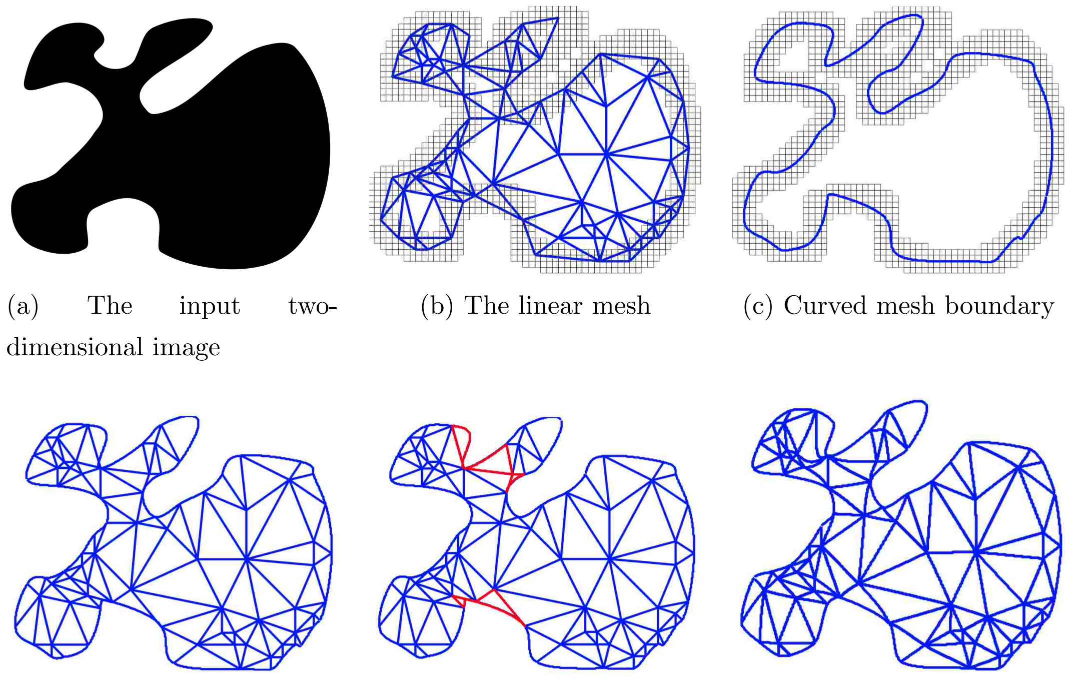
(d) A boundary vertex c merged to a boundary vertex d on the topological disk.

FIG. 15: An illustration of the vertex merge operation on a topological disk. The list of vertices in brackets shows merge history.

CHAPTER 4

TWO-DIMENSIONAL CURVILINEAR TRIANGULAR MESH GENERATION

Given a bounded curved domain $\Omega \subset \mathcal{R}^2$, the algorithm outputs a curvilinear mesh of the *interior* of Ω with globally smooth boundary. Figure 16 illustrates the main steps performed by our algorithm. The details are elaborated below.



(a) The input two-dimensional image

(b) The linear mesh

(c) Curved mesh boundary

(d) Smooth curved boundary with linear edges in the interior

(e) Invalid elements are detected

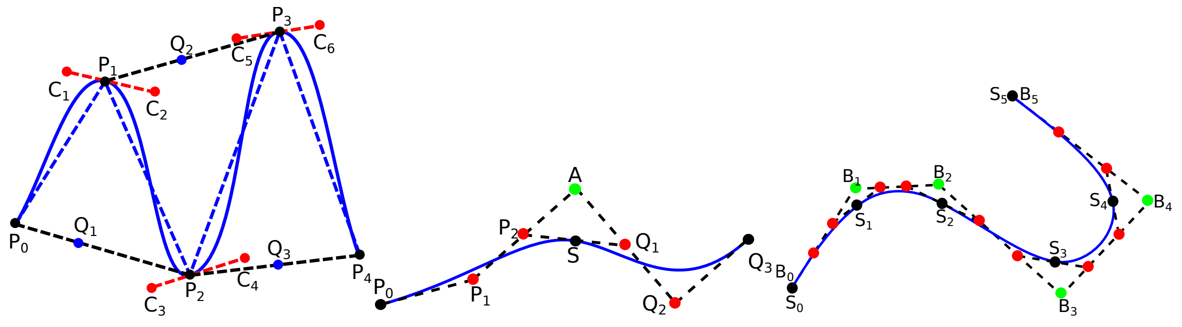
(f) Valid high quality curvilinear mesh

FIG. 16: An illustration of the main steps performed by our algorithm.

4.1 SMOOTH BOUNDARY CONSTRUCTION

We aim to find a smooth curve interpolating all the mesh boundary vertices given in order. A curve can be described as having C^n continuity, n being the measure of smoothness. Consider the segments on either side of a point on a curve: (1) C^0: The segments touch at the joint point; (2) C^1: First derivatives are continuous at the joint point; (3) C^2: First and second derivatives are continuous at the joint point.

A smooth C^1 piecewise cubic curve is composed of pieces of different cubic curves glued together, and it has a first derivative everywhere and the derivative is continuous. A Bézier path is C^1 smooth provided that two Bézier curves share a common tangent direction at the joint point. The basic idea is to calculate control points around each endpoint so that they lie in a straight line with the endpoint. However, curved segments would not flow smoothly together when quadratic Bézier form (three control points) is used. Instead, we need to go one order higher to the cubic Bézier form (four control points) so we can build 'S' shaped segments. We find these control points by translating the segments formed by the lines between the previous endpoint and the next endpoint such that these segments become the tangents of the curves at the endpoints. We scale these segments to control the curvature. An example is illustrated in Figure 17a. For the curve between P_1 and P_2, we need C_2 and C_3. On segment P_0P_2, find a point Q_1 such that$|P_0Q_1|/|Q_1P_2|= |P_0P_1|/|P_1P_2|$. Translate segment P_0P_2 so that point Q_1 lies on point P_1, and scale the length of translated segment P_0P_2, then the new position of point P_2 is the position of control point C_2. Similarly, the position of control point C_3 can be found by translating segment P_1P_3 such that point Q_2 lies on point P_2.



(a) A smooth C^1 path (b) An A-frame (c) A smooth C^2 curve

FIG. 17: An illustration of the construction of the C^1 and C^2 smooth curves. The black points show the mesh vertices, the red points show the control points, and the green points show the B-spline points.

The cubic Bézier form provides enough degrees of freedom to construct a cubic spline curve that satisfies C^2 smoothness requirement. Since the curvature of a point on a curve is a function with respect to the first and second derivative of this point, and if the first and the second derivative are continuous, then the curvature at this point is continuous. We prefer C^2 smooth curve to C^1 smooth curve because the boundary of the biomedical objects usually have continuous curvatures. If two Bézier curves with control points P_0, P_1, P_2, S and S, Q_1, Q_2, Q_3 touch at point S, both their first and second derivatives match at S if and only if their control polygons fit an A-frame, which is a structure in which P_2 is the midpoint of $\overline{AP_1}$, Q_1 is the midpoint of $\overline{AQ_2}$ and S is the midpoint of $\overline{P_2Q_1}$ as Figure 17b shows. To fit the A-frame in the set of cubic curves, one easy approach is to use B-spline as an intermediate step. In Figure 17c, the junction points S (shown in black) are mesh vertices that are classified on the boundary of the linear mesh. If the B-spline points B (the apexes of the A-frames, shown in green) are known, the control points (shown in red) can be calculated by computing the one third and two thirds positions between the connection of every two adjacent B-spline points. The B-spline points B and the junction points S satisfy a relationship:

$$6S_i = B_{i-1} + 4B_i + B_{i+1}.$$

By solving for a linear system of equations, the coordinates of B-spline points can be obtained.

4.2 ELEMENT VALIDITY

The following two subsections present two methods to verify the validity of the high-order mesh elements. In the first method, the element validity is evaluated by the lower bound of the Jacobians. The Jacobians of an element can be written as a fourth-order Bézier triangle whose convex hull provides a lower bound of the Jacobians. By recursively split the Bézier triangle, a tight lower bound can be obtained to offer an accurate validity check. In the second method, we proved that if the control net of a cubic Bézier triangle does not contain inverted control triangles, then the cubic Bézier triangle has strictly positive Jacobians. The second method offers a faster way to verify the element validity.

4.2.1 VALIDITY VERIFICATION BY THE LOWER BOUND OF THE JACOBIAN

The invalid elements are usually caused by curving only the boundary mesh edges while the interior mesh edges remain straight. Some of the curvilinear triangular patches may have tangled edges. Thus, it is necessary to verify the validity and to eliminate all the invalid elements by curving interior mesh edges as a post-processing step once the curved mesh has been constructed. One approach to verify the positiveness of the *Jacobian* is sampling the *Jacobian* at discrete locations such as at Gaussian points. A more precise way is to calculate the tight lower bound for the *Jacobian*.

Since the Bézier basis is selected to represent the elements, the tight lower bound can be calculated with the help of its special properties such as the convex hull property and subdivision property [48]. We can write the Jacobian matrix J of a cubic Bézier triangle as:

$$\begin{aligned} J &= \begin{bmatrix} \frac{\partial \mathcal{T}_x^3}{\partial u} & \frac{\partial \mathcal{T}_x^3}{\partial v} & \frac{\partial \mathcal{T}_x^3}{\partial w} \\ \frac{\partial \mathcal{T}_y^3}{\partial u} & \frac{\partial \mathcal{T}_y^3}{\partial v} & \frac{\partial \mathcal{T}_y^3}{\partial w} \end{bmatrix} \begin{bmatrix} \frac{\partial u}{\partial \hat{x}} & \frac{\partial u}{\partial \hat{y}} \\ \frac{\partial v}{\partial \hat{x}} & \frac{\partial v}{\partial \hat{y}} \\ \frac{\partial w}{\partial \hat{x}} & \frac{\partial w}{\partial \hat{y}} \end{bmatrix} \\ &= \begin{bmatrix} \frac{\partial \mathcal{T}_x^3}{\partial u} & \frac{\partial \mathcal{T}_x^3}{\partial v} & \frac{\partial \mathcal{T}_x^3}{\partial w} \\ \frac{\partial \mathcal{T}_y^3}{\partial u} & \frac{\partial \mathcal{T}_y^3}{\partial v} & \frac{\partial \mathcal{T}_y^3}{\partial w} \end{bmatrix} \begin{bmatrix} -1 & -1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \frac{\partial \mathcal{T}_x^3}{\partial v} - \frac{\partial \mathcal{T}_x^3}{\partial u} & \frac{\partial \mathcal{T}_x^3}{\partial w} - \frac{\partial \mathcal{T}_x^3}{\partial u} \\ \frac{\partial \mathcal{T}_y^3}{\partial v} - \frac{\partial \mathcal{T}_y^3}{\partial u} & \frac{\partial \mathcal{T}_y^3}{\partial w} - \frac{\partial \mathcal{T}_y^3}{\partial u} \end{bmatrix}, \end{aligned}$$

and the determinant of J can be represented as:

$$det(J) = (\frac{\partial \mathcal{T}^3}{\partial v} - \frac{\partial \mathcal{T}^3}{\partial u}) \times (\frac{\partial \mathcal{T}^3}{\partial w} - \frac{\partial \mathcal{T}^3}{\partial u}) \cdot \mathbf{n},$$

where $\mathbf{n}$ is the vector $(0, 0, 1)$. Because the derivative of a cubic Bézier polynomial is a quadratic Bézier polynomial, and the product of two quadratic Bézier polynomials is a fourth order Bézier polynomial, the *Jacobian* is a fourth order Bézier polynomial with fifteen control points. Notice that this *Jacobian* function is a scalar-valued function, and the control points are scalar values. So in the following the scalar-valued control points are named control values.

$$\mathcal{T}^4(u, v, w) = \sum_{i+j+k=4} B^4_{ijk}(u, v, w) P_{ijk},$$

where P_{ijk} is one of the fifteen control values, $B^4_{ijk}(u, v, w) = \frac{4!}{i!j!k!} u^i v^j w^k$, $u \in [0, 1]$, $v \in [0, 1]$ and $w \in [0, 1]$ are the barycentric coordinates and $u+v+w = 1$. Therefore, the fifteen control values of the *Jacobian* can be represented by the control points of the cubic Bézier triangle. Because

$$\frac{\partial \mathcal{T}}{\partial v} - \frac{\partial \mathcal{T}}{\partial u} = 3u^2 a_1 + 3v^2 b_1 + 3w^2 c_1 + 6uwd_1 + 6uve_1 + 6vwf_1$$

and

$$\frac{\partial \mathcal{T}}{\partial w} - \frac{\partial \mathcal{T}}{\partial u} = 3u^2 a_2 + 3v^2 b_2 + 3w^2 c_2 + 6uwd_2 + 6uve_2 + 6vwf_2,$$

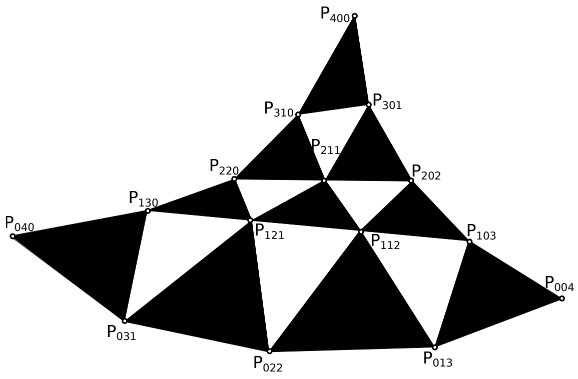
where $a_1 = P_{210} - P_{300}$, $b_1 = P_{030} - P_{120}$, $c_1 = P_{012} - P_{102}$, $d_1 = P_{111} - P_{201}$, $e_1 = P_{120} - P_{210}$, $f_1 = P_{021} - P_{111}$, $a_2 = P_{201} - P_{300}$, $b_2 = P_{021} - P_{120}$, $c_2 = P_{003} - P_{102}$, $d_2 = P_{102} - P_{201}$, $e_2 = P_{111} - P_{210}$, $f_2 = P_{012} - P_{111}$, the fifteen control values can be calculated. They are listed in Table 1.

TABLE 1: Fifteen control values for $det(J)$ of a cubic triangle

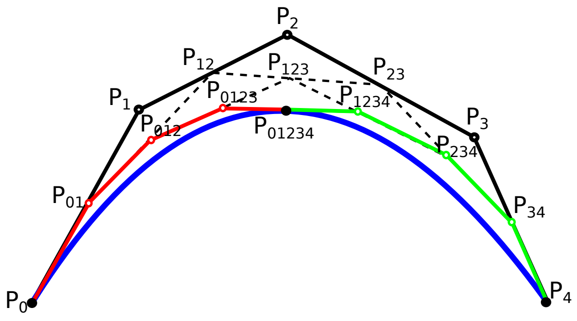
P_{ijk}	Control value
P_{400}	$9(a_1 \times a_2 \cdot \mathbf{n})$
P_{040}	$9(b_1 \times b_2 \cdot \mathbf{n})$
P_{004}	$9(c_1 \times c_2 \cdot \mathbf{n})$
P_{220}	$\frac{3}{2}(a_1 \times b_2 \cdot \mathbf{n} + b_1 \times a_2 \cdot \mathbf{n} + 4e_1 \times e_2 \cdot \mathbf{n})$
P_{202}	$\frac{3}{2}(a_1 \times c_2 \cdot \mathbf{n} + c_1 \times a_2 \cdot \mathbf{n} + 4d_1 \times d_2 \cdot \mathbf{n})$
P_{022}	$\frac{3}{2}(b_1 \times c_2 \cdot \mathbf{n} + c_1 \times b_2 \cdot \mathbf{n} + 4f_1 \times f_2 \cdot \mathbf{n})$
P_{301}	$\frac{9}{2}(a_1 \times d_2 \cdot \mathbf{n} + d_1 \times a_2 \cdot \mathbf{n})$
P_{310}	$\frac{9}{2}(a_1 \times e_2 \cdot \mathbf{n} + e_1 \times a_2 \cdot \mathbf{n})$
P_{130}	$\frac{9}{2}(b_1 \times e_2 \cdot \mathbf{n} + e_1 \times b_2 \cdot \mathbf{n})$
P_{031}	$\frac{9}{2}(b_1 \times f_2 \cdot \mathbf{n} + f_1 \times b_2 \cdot \mathbf{n})$
P_{103}	$\frac{9}{2}(c_1 \times d_2 \cdot \mathbf{n} + d_1 \times c_2 \cdot \mathbf{n})$
P_{013}	$\frac{9}{2}(c_1 \times f_2 \cdot \mathbf{n} + f_1 \times c_2 \cdot \mathbf{n})$
P_{211}	$\frac{3}{2}(a_1 \times f_2 \cdot \mathbf{n} + f_1 \times a_2 \cdot \mathbf{n} + 2d_1 \times e_2 \cdot \mathbf{n} + 2e_1 \times d_2 \cdot \mathbf{n})$
P_{121}	$\frac{3}{2}(b_1 \times d_2 \cdot \mathbf{n} + d_1 \times b_2 \cdot \mathbf{n} + 2e_1 \times f_2 \cdot \mathbf{n} + 2f_1 \times e_2 \cdot \mathbf{n})$
P_{112}	$\frac{3}{2}(c_1 \times e_2 \cdot \mathbf{n} + e_1 \times c_2 \cdot \mathbf{n} + 2d_1 \times f_2 \cdot \mathbf{n} + 2f_1 \times d_2 \cdot \mathbf{n})$

Due to the convex hull property, the fourth order Bézier triangle is completely contained in its convex hull formed by the control values, thus, the minimum of the convex hull is the lower bound of the *Jacobian*. If the lower bound is positive, then the element is valid. However, if the lower bound is non-positive, it does not necessarily mean that the element is invalid. Since it is only a sufficient condition, sometimes it is overly conservative. In the cases that the lower bound is not tight, the minimum value of the *Jacobian* could be positive whereas the calculated lower bound is non-positive.

To further confirm the answer, we obtain the tighter bound by refining the convex hull using the Bézier subdivision algorithm. The *Jacobian* of the element is a fourth order Bézier triangle defined by fifteen control values. The boundary of the fourth order Bézier triangle is three fourth order Bézier curves. The control values of the three boundary Bézier curves are the control values of the Bézier triangle located on the boundary. For illustration purpose, we use vector-valued function to represent scalar-valued function in Figure 18. In Figure 18a, $P_{ijk}, i + j + k = 4$ are the control values of the fourth order Bézier triangle. The control values of the left boundary Bézier curve is $P_{400}, P_{310}, P_{220}, P_{130}$ and P_{040}, of the right boundary Bézier curve is $P_{400}, P_{301}, P_{202}, P_{103}$ and P_{004}, and of the bottom boundary Bézier curve is $P_{040}, P_{031}, P_{022}, P_{013}$ and P_{004}. The convex hull of the fourth order Bézier triangle is a subset of the union of the convex hulls of the three boundary Bézier curves. Thus, we refine the convex hull of the fourth order Bézier triangle by refining the three convex hulls of its boundary curves. The minimum of the convex hull of the fourth order Bézier triangle is the minimum of the convex hulls of the three boundary Bézier curves. Therefore, we verify the positiveness of the *Jacobian* by verifying the positiveness of the three fourth order Bézier curves.



The control net of a fourth order Bézier triangle



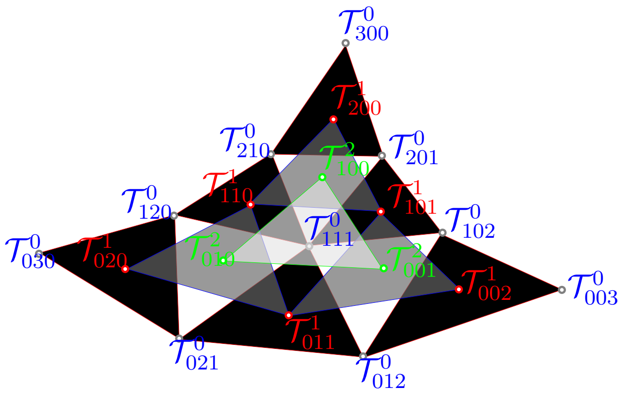
The recursive subdivision algorithm on the fourth order Bézier curve

FIG. 18: An illustration of the positiveness verification on *Jacobian*. The newly created control polygons for the fourth order Bézier curve are shown in red and green.

Algorithm 5 summarizes the positiveness verification on a fourth order Bézier curve. Function MINIMUM() returns the minimum value of the five control values. Function MIDDLE() returns the middle value of the two input control values. Figure 18b illustrates the recursive subdivision algorithm for the fourth order Bézier curve. If the five control values are positive, all the values on the Bézier curve are positive because of the convex hull property. Notice that two of the control values are on the ends of the curve. If at least one of the control values is non-positive, we first verify that if the non-positive control values is on the ends of the curve. If it is, the element is invalid because there is at least one non-positive value on the *Jacobian* of this element. If both of the ends of the curve are positive, the algorithm splits the curve into two curves and verifies the positiveness on the two sub-curves. The Bézier subdivision algorithm recursively splits the curve and verifies the positiveness until either all the control values of the sub-curves are positive, or found at least one non-positive value on the curve. So the algorithm always terminates.

Algorithm 5: The positiveness verification on a fourth-order Bézier curve

```
1  Algorithm: Subdivision($P_0$, $P_1$, $P_2$, $P_3$, $P_4$)
   Input  : $P_0$, $P_1$, $P_2$, $P_3$, $P_4$ are the five control points of a fourth-order Bézier
            curve.
   Output: A boolean value.
2  if MINIMUM($P_0$, $P_1$, $P_2$, $P_3$, $P_4$) $> 0$ then
3      return true
4  end
5  if $P_0 \leq 0 \vee P_4 \leq 0$ then
6      return false
7  end
8  $P_{01}$ = MIDDLE($P_0$, $P_1$)
9  $P_{12}$ = MIDDLE($P_1$, $P_2$)
10 $P_{23}$ = MIDDLE($P_2$, $P_3$)
11 $P_{34}$ = MIDDLE($P_3$, $P_4$)
12 $P_{012}$ = MIDDLE($P_{01}$, $P_{12}$)
13 $P_{123}$ = MIDDLE($P_{12}$, $P_{23}$)
14 $P_{234}$ = MIDDLE($P_{23}$, $P_{34}$)
15 $P_{0123}$ = MIDDLE($P_{012}$, $P_{123}$)
16 $P_{1234}$ = MIDDLE($P_{123}$, $P_{234}$)
17 $P_{01234}$ = MIDDLE($P_{0123}$, $P_{1234}$)
18 returnBÉZIERCURVESUBDIVISION($P_0$, $P_{01}$, $P_{012}$, $P_{0123}$, $P_{01234}$) $\wedge$
     BÉZIERCURVESUBDIVISION($P_{01234}$, $P_{1234}$, $P_{234}$, $P_{34}$, $P_4$)
```



$$\begin{aligned}
\mathcal{T}^1_{200} &= u\mathcal{T}^0_{300} + v\mathcal{T}^0_{210} + w\mathcal{T}^0_{201},\\
\mathcal{T}^1_{110} &= u\mathcal{T}^0_{210} + v\mathcal{T}^0_{120} + w\mathcal{T}^0_{111},\\
\mathcal{T}^1_{101} &= u\mathcal{T}^0_{201} + v\mathcal{T}^0_{111} + w\mathcal{T}^0_{102},\\
\mathcal{T}^1_{020} &= u\mathcal{T}^0_{120} + v\mathcal{T}^0_{030} + w\mathcal{T}^0_{021},\\
\mathcal{T}^1_{011} &= u\mathcal{T}^0_{111} + v\mathcal{T}^0_{021} + w\mathcal{T}^0_{012},\\
\mathcal{T}^1_{002} &= u\mathcal{T}^0_{102} + v\mathcal{T}^0_{012} + w\mathcal{T}^0_{003},\\
\mathcal{T}^2_{100} &= u\mathcal{T}^1_{200} + v\mathcal{T}^1_{110} + w\mathcal{T}^1_{101},\\
\mathcal{T}^2_{010} &= u\mathcal{T}^1_{110} + v\mathcal{T}^1_{020} + w\mathcal{T}^1_{011},\\
\mathcal{T}^2_{001} &= u\mathcal{T}^1_{101} + v\mathcal{T}^1_{011} + w\mathcal{T}^1_{002},\\
\mathcal{T}^3_{000} &= u\mathcal{T}^2_{100} + v\mathcal{T}^2_{010} + w\mathcal{T}^2_{001}.
\end{aligned}$$

FIG. 19: An illustration of the de Casteljau Algorithm for a cubic Bézier triangle. The control points $\mathcal{T}^0_{\mathbf{i}}$ form a control net, and the black triangles $\triangle\mathcal{T}^0_{300}\mathcal{T}^0_{210}\mathcal{T}^0_{201}$, $\triangle\mathcal{T}^0_{210}\mathcal{T}^0_{120}\mathcal{T}^0_{111}$, $\triangle\mathcal{T}^0_{201}\mathcal{T}^0_{111}\mathcal{T}^0_{102}$, $\triangle\mathcal{T}^0_{120}\mathcal{T}^0_{030}\mathcal{T}^0_{021}$, $\triangle\mathcal{T}^0_{111}\mathcal{T}^0_{021}\mathcal{T}^0_{012}$ and $\triangle\mathcal{T}^0_{102}\mathcal{T}^0_{012}\mathcal{T}^0_{003}$ are called control triangles.

4.2.2 VALIDITY VERIFICATION BY THE CONTROL NET OF THE ELEMENT

Denote $A(\mathbf{p},\mathbf{q},\mathbf{r})$ the signed area of the triangle $\triangle\mathbf{pqr}$,

$$2A(\mathbf{p},\mathbf{q},\mathbf{r}) = \begin{vmatrix} 1 & 1 & 1 \\ p_x & q_x & r_x \\ p_y & q_y & r_y \end{vmatrix},$$

and $\mathbf{p} = (p_x, p_y)$, $\mathbf{q} = (q_x, q_y)$, $\mathbf{r} = (r_x, r_y)$. A triangle is considered not inverted if its vertices are labeled counterclockwise, meaning that the signed area of the triangle is positive.

Theorem 4.2.1. *A cubic Bézier triangle has strictly positive Jacobian if the control net of the cubic Bézier triangle is not twisted, meaning that all the control triangles (black triangles in Fig. 19 composed by control points) in the control net are not inverted.*

Proof. The proof mainly depends on the de Casteljau Algorithm for a cubic Bézier triangle [48] (illustrated in Fig. 19). Denote $\mathbf{e1} = (1,0,0)$, $\mathbf{e2} = (0,1,0)$, $\mathbf{e3} = (0,0,1)$. Given a set of control points $\mathcal{T}^0_{\mathbf{i}} \in \mathcal{R}^2$ and barycentric coordinates $\mathbf{u} =$

(u, v, w), set

$$\mathcal{T}_{\mathbf{i}}^{r}(\mathbf{u}) = u\mathcal{T}_{\mathbf{i+e1}}^{r-1}(\mathbf{u}) + v\mathcal{T}_{\mathbf{i+e2}}^{r-1}(\mathbf{u}) + w\mathcal{T}_{\mathbf{i+e3}}^{r-1}(\mathbf{u}), \quad r = 1, ..., n, \quad |\mathbf{i}| = n - r,$$

then $\mathcal{T}_{\mathbf{0}}^{n}(\mathbf{u})$ is the point with parameter $\mathbf{u}$ on the n-th order Bézier triangle $\mathcal{T}^n$.

We first prove that the Jacobian has the same sign as the signed area of the triangle $\triangle\mathcal{T}_{100}^2\mathcal{T}_{010}^2\mathcal{T}_{001}^2$ in Fig. 19, then we observe that if there is no inverted control triangle, then the triangle $\triangle\mathcal{T}_{100}^2\mathcal{T}_{010}^2\mathcal{T}_{001}^2$ has positive area.

The Jacobian can be written as:

$$|J| = \begin{vmatrix} \frac{\partial\mathcal{T}_x^3}{\partial\hat{x}} & \frac{\partial\mathcal{T}_x^3}{\partial\hat{y}} \\ \frac{\partial\mathcal{T}_y^3}{\partial\hat{x}} & \frac{\partial\mathcal{T}_y^3}{\partial\hat{y}} \end{vmatrix} = \frac{1}{3}\begin{vmatrix} 3 & 0 & 0 \\ \frac{\partial\mathcal{T}_x^3}{\partial u} + \frac{\partial\mathcal{T}_x^3}{\partial v} + \frac{\partial\mathcal{T}_x^3}{\partial w} & \frac{\partial\mathcal{T}_x^3}{\partial\hat{x}} & \frac{\partial\mathcal{T}_x^3}{\partial\hat{y}} \\ \frac{\partial\mathcal{T}_y^3}{\partial u} + \frac{\partial\mathcal{T}_y^3}{\partial v} + \frac{\partial\mathcal{T}_y^3}{\partial w} & \frac{\partial\mathcal{T}_y^3}{\partial\hat{x}} & \frac{\partial\mathcal{T}_y^3}{\partial\hat{y}} \end{vmatrix}.$$

Because $u = 1 - \hat{x} - \hat{y}$, $v = \hat{x}$ and $w = \hat{y}$, we have $\frac{\partial u}{\partial\hat{x}} + \frac{\partial v}{\partial\hat{x}} + \frac{\partial w}{\partial\hat{x}} = 0$, $\frac{\partial u}{\partial\hat{y}} + \frac{\partial v}{\partial\hat{y}} + \frac{\partial w}{\partial\hat{y}} = 0$ and

$$\begin{vmatrix} 1 & \frac{\partial u}{\partial\hat{x}} & \frac{\partial u}{\partial\hat{y}} \\ 1 & \frac{\partial v}{\partial\hat{x}} & \frac{\partial v}{\partial\hat{y}} \\ 1 & \frac{\partial w}{\partial\hat{x}} & \frac{\partial w}{\partial\hat{y}} \end{vmatrix} = 3,$$

thus

$$|J| = \frac{1}{3}\begin{vmatrix} 1 & 1 & 1 \\ \frac{\partial\mathcal{T}_x^3}{\partial u} & \frac{\partial\mathcal{T}_x^3}{\partial v} & \frac{\partial\mathcal{T}_x^3}{\partial w} \\ \frac{\partial\mathcal{T}_y^3}{\partial u} & \frac{\partial\mathcal{T}_y^3}{\partial v} & \frac{\partial\mathcal{T}_y^3}{\partial w} \end{vmatrix}\begin{vmatrix} 1 & \frac{\partial u}{\partial\hat{x}} & \frac{\partial u}{\partial\hat{y}} \\ 1 & \frac{\partial v}{\partial\hat{x}} & \frac{\partial v}{\partial\hat{y}} \\ 1 & \frac{\partial w}{\partial\hat{x}} & \frac{\partial w}{\partial\hat{y}} \end{vmatrix} = \begin{vmatrix} 1 & 1 & 1 \\ \frac{\partial\mathcal{T}_x^3}{\partial u} & \frac{\partial\mathcal{T}_x^3}{\partial v} & \frac{\partial\mathcal{T}_x^3}{\partial w} \\ \frac{\partial\mathcal{T}_y^3}{\partial u} & \frac{\partial\mathcal{T}_y^3}{\partial v} & \frac{\partial\mathcal{T}_y^3}{\partial w} \end{vmatrix}.$$

Because $\frac{\partial\mathcal{T}_{200}^1}{\partial u}$, $\frac{\partial\mathcal{T}_{110}^1}{\partial u}$ and $\frac{\partial\mathcal{T}_{101}^1}{\partial u}$ can be computed as: (refer to Fig. 19)

$$\begin{aligned}
\frac{\partial\mathcal{T}_{200}^1}{\partial u} &= \frac{\partial(u\mathcal{T}_{300}^0 + v\mathcal{T}_{210}^0 + w\mathcal{T}_{201}^0)}{\partial u} = \mathcal{T}_{300}^0, \\
\frac{\partial\mathcal{T}_{110}^1}{\partial u} &= \frac{\partial(u\mathcal{T}_{210}^0 + v\mathcal{T}_{120}^0 + w\mathcal{T}_{111}^0)}{\partial u} = \mathcal{T}_{210}^0, \\
\frac{\partial\mathcal{T}_{101}^1}{\partial u} &= \frac{\partial(u\mathcal{T}_{201}^0 + v\mathcal{T}_{111}^0 + w\mathcal{T}_{102}^0)}{\partial u} = \mathcal{T}_{201}^0,
\end{aligned}$$

then

$$\begin{aligned}
\frac{\partial\mathcal{T}_{100}^2}{\partial u} &= \frac{\partial(u\mathcal{T}_{200}^1 + v\mathcal{T}_{110}^1 + w\mathcal{T}_{101}^1)}{\partial u} \\
&= \mathcal{T}_{200}^1 + u\frac{\partial\mathcal{T}_{200}^1}{\partial u} + v\frac{\partial\mathcal{T}_{110}^1}{\partial u} + w\frac{\partial\mathcal{T}_{101}^1}{\partial u} \\
&= \mathcal{T}_{200}^1 + u\mathcal{T}_{300}^0 + v\mathcal{T}_{210}^0 + w\mathcal{T}_{201}^0 \\
&= \mathcal{T}_{200}^1 + \mathcal{T}_{200}^1 \\
&= 2\mathcal{T}_{200}^1.
\end{aligned}$$

And $\frac{\partial \mathcal{T}^2_{010}}{\partial u}$, $\frac{\partial \mathcal{T}^2_{001}}{\partial u}$ can be derived similarly:

$$\frac{\partial \mathcal{T}^2_{010}}{\partial u} = 2\mathcal{T}^1_{110},$$

$$\frac{\partial \mathcal{T}^2_{001}}{\partial u} = 2\mathcal{T}^1_{101}.$$

Thus,

$$\begin{aligned}
\frac{\partial \mathcal{T}^3}{\partial u} &= \frac{\partial(u\mathcal{T}^2_{100} + v\mathcal{T}^2_{010} + w\mathcal{T}^2_{001})}{\partial u} \\
&= \frac{\partial(u\mathcal{T}^2_{100})}{\partial u} + \frac{\partial(v\mathcal{T}^2_{010})}{\partial u} + \frac{\partial(w\mathcal{T}^2_{001})}{\partial u} \\
&= \mathcal{T}^2_{100} + u\frac{\partial \mathcal{T}^2_{100}}{\partial u} + v\frac{\partial \mathcal{T}^2_{010}}{\partial u} + w\frac{\partial \mathcal{T}^2_{001}}{\partial u} \\
&= \mathcal{T}^2_{100} + u2\mathcal{T}^1_{200} + v2\mathcal{T}^1_{110} + w2\mathcal{T}^1_{101} \\
&= 3\mathcal{T}^2_{100}.
\end{aligned}$$

Similarly, we derive the following equations:

$$\frac{\partial \mathcal{T}^3}{\partial v} = 3\mathcal{T}^2_{010},$$

$$\frac{\partial \mathcal{T}^3}{\partial w} = 3\mathcal{T}^2_{001}.$$

Therefore, the Jacobian can be computed as

$$\begin{aligned}
|J| &= \begin{vmatrix} 1 & 1 & 1 \\ \frac{\partial \mathcal{T}^3_x}{\partial u} & \frac{\partial \mathcal{T}^3_x}{\partial v} & \frac{\partial \mathcal{T}^3_x}{\partial w} \\ \frac{\partial \mathcal{T}^3_y}{\partial u} & \frac{\partial \mathcal{T}^3_y}{\partial v} & \frac{\partial \mathcal{T}^3_y}{\partial w} \end{vmatrix} \\
&= \begin{vmatrix} 1 & 1 & 1 \\ 3\mathcal{T}^2_{100x} & 3\mathcal{T}^2_{010x} & 3\mathcal{T}^2_{001x} \\ 3\mathcal{T}^2_{100y} & 3\mathcal{T}^2_{010y} & 3\mathcal{T}^2_{001y} \end{vmatrix} \\
&= 9\begin{vmatrix} 1 & 1 & 1 \\ \mathcal{T}^2_{100x} & \mathcal{T}^2_{010x} & \mathcal{T}^2_{001x} \\ \mathcal{T}^2_{100y} & \mathcal{T}^2_{010y} & \mathcal{T}^2_{001y} \end{vmatrix} \\
&= 18A(\mathcal{T}^2_{100}, \mathcal{T}^2_{010}, \mathcal{T}^2_{001}).
\end{aligned}$$

So the Jacobian has the same sign as the signed area of the triangle $\triangle\mathcal{T}^2_{100}\mathcal{T}^2_{010}\mathcal{T}^2_{001}$.

Because of the structure of the control net, when the vertices of all the triangles in the control net are in the counterclockwise order, the vertices of all the triangles in the second layer control net (formed by vertices $\mathcal{T}_{\mathbf{i}}^{1}$) are also in the counterclockwise order, so do the triangles in the third layer control net (formed by triangle $\triangle\mathcal{T}_{100}^{2}\mathcal{T}_{010}^{2}\mathcal{T}_{001}^{2}$). Thus, the intermediate triangle $\triangle\mathcal{T}_{100}^{2}\mathcal{T}_{010}^{2}\mathcal{T}_{001}^{2}$ has positive area, and therefore the Jacobian is positive. $\square$

4.3 MESH UNTANGLING

It is usually not enough to curve only the mesh boundary because some control points may be located such that invalid elements occur. In such case, edges in the interior of the mesh should also be curved to eliminate the invalidity or to improve the curved element quality.

We move the control points of the interior mesh edges using a finite element method [1]. The geometry of the domain to be meshed is represented as an elastic solid. For each linear mesh edge, the two points which are located in the one third and two thirds ratio of each edge are computed. These points together with the mesh vertices are the original positions of the control points of the edges before deformation. These points form the control nets of the linear mesh elements. The control nets together as a whole is the undeformed geometry (shown in Figure 20b). The external loads are the displacements of the control points (red points in Figure 20c) of the smooth curved boundary edges. The control nets are deformed such that when the control points of the boundary edges of the linear mesh moved to the corresponding control points of the curved boundary edge, the new positions of the control points of the interior mesh edges are determined by solving for the equilibrium configuration of an elasticity problem. Figure 20 illustrates these steps.

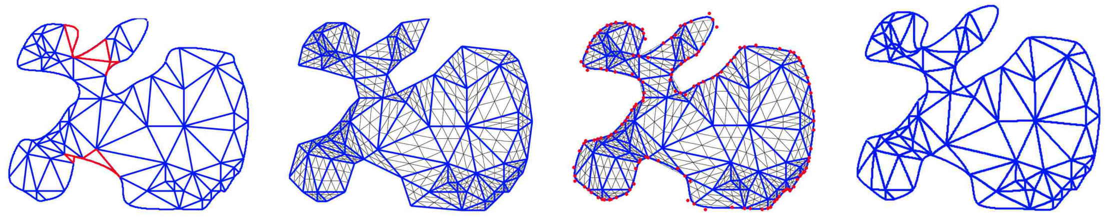
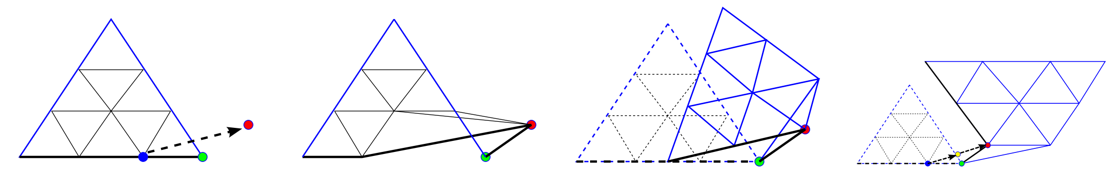

(a) Invalid mesh (b) The control nets (c) The external loads (d) The final mesh

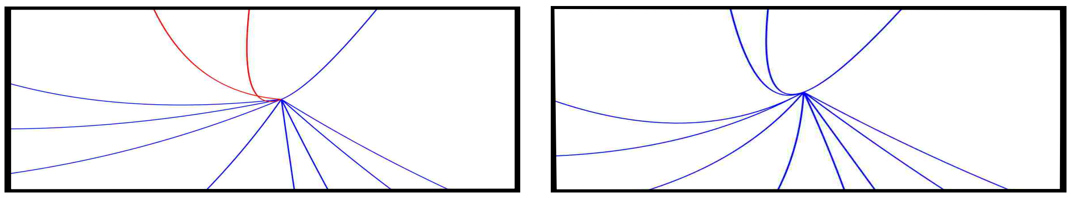
FIG. 20: An illustration of eliminating the element invalidity or improving the element quality.



(a) The mesh composed of one element (b) The invalid mesh with twisted control net (c) The one-step FE method was applied, but the control net is still twisted (d) The iterative FE method successfully corrected the twisted control net

FIG. 21: An illustration of the iterative finite element method.

In some cases, the one step finite element method can handle this problem successfully. However, in the case that the curvature of the boundary edge is very large, the interior edges may not be able to be curved enough to correct the invalidity. The iterative finite element method successfully solves this problem. Figure 21 illustrates the iterative FE method. In this example there is only one element in the mesh, the black border line represents the mesh boundary, the blue point represents one control point of the linear boundary edge. The red point represents the corresponding control point of the curved boundary edge. The green point is one of the mesh vertices on the mesh boundary, thus it has to maintain its position. The control net is invalid because there exists an inverted triangle. When one step FE method was applied, the blue point was directly moved to the red point. After solving for the equilibrium configuration, the control net is still twisted. However, when the yellow point was considered as the intermediate displacement, the blue point was first moved to the yellow point, then moved to the red point, the two iteration FE method successfully corrected the twisted control net.



(a) Part of final mesh after one-step FE method (b) Part of final mesh after eight iterations of iterative FE method

FIG. 22: A comparison of the result of one-step FE method and the result of the iterative FE method. Red edges high light the tangled element.

The iterative FE method executes the validity check before each round. When it is reported that an invalid element exists, the procedure divides the segments formed by the control points of the linear boundary edges and the corresponding control points of the curved edges. The procedure takes the endpoints of the subsegments one by one as the intermediate external loadings, and takes the solution of the current external loadings as the undeformed geometry of the next external loadings. The algorithm terminates when all the invalid elements are corrected. Figure 22 shows an example of the comparison of the result of one-step FE method and the result of the iterative FE method.

CHAPTER 5

EXPERIMENTAL RESULTS

5.1 THREE-DIMENSIONAL LINEAR MESHING RESULTS

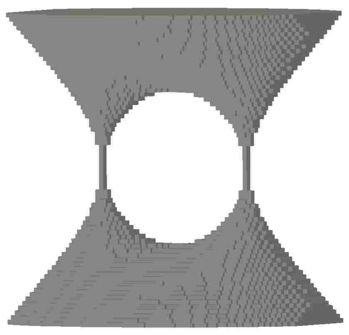
We apply the proposed octree refinement and decimation algorithm (ORD) to both synthetic and real medical data in the following sections. All the experiments were conducted on a 64-bit machine equipped with two 3.06 GHz 6-Core Intel Xeon CPU and 64 GB main memory. The algorithm was implemented in C++, in both two and three dimensions.

Our algorithm deals with segmented images in which each voxel is associated with a label in accordance with the material the voxel belongs to. The evaluation of our algorithm includes the visualization of the final meshes, run time, and the number of tetrahedra, decided by the input geometry. In the following, the first subsection evaluates on a three-dimensional geometry (Cassini surface). The final mesh with a fixed size and other parameters is visualized, and is compared with the final mesh generated by LD algorithm with the same size and parameters. This way, we gain insight into the result in terms of the topological correctness. We also evaluate the run time with respect to different sizes of the Cassini surface and fixed quality and fidelity parameters, and evaluate the number of tetrahedra with respect to a fixed size of the Cassini surface and varied quality and fidelity parameters.

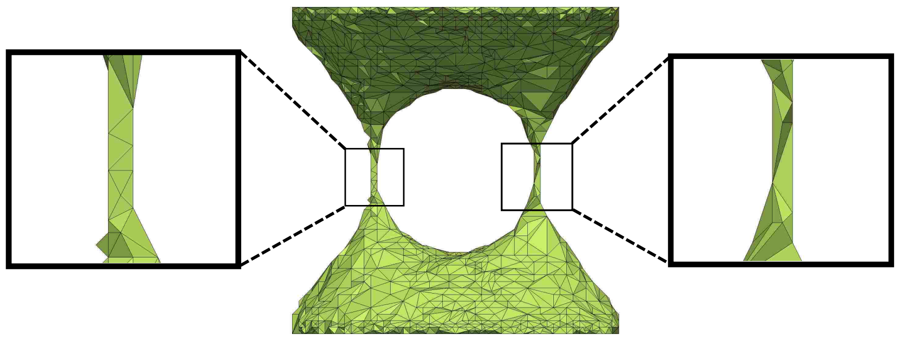
The second subsection evaluates on several real-world medical images: a *head neck* atlas [49], a *knee* atlas [50], an *abdomen* atlas [51] and an *abdomen background* atlas [51]. Each of these atlases is a segmented medical image whose domain is a multi-domain where each subdomain corresponds to a specific tissue. We visualize the final meshes, and evaluate the run time and the number of tetrahedra with respect to varied quality and fidelity parameters.

For the 3D visualization of the final meshes, we used ParaView [52], an open-source visualization software. In this section the barred symbols $\bar{H}$, $\bar{h}$, and $\bar{\theta}$ stand for the bounds that are guaranteed by the algorithm, while the regular symbols H, h, and θ represent the values measured from the output meshes.

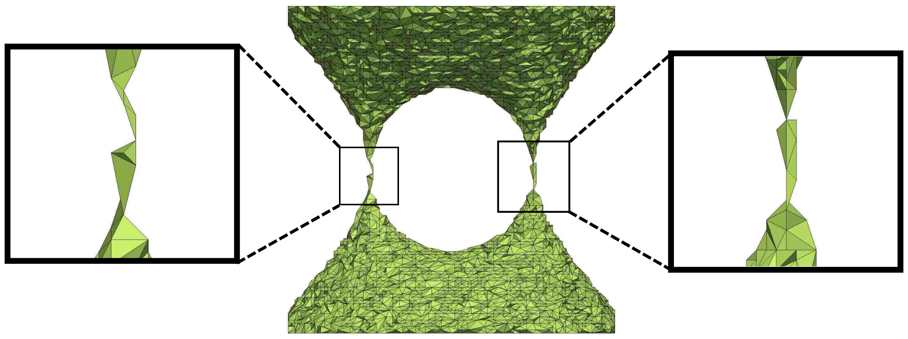
5.1.1 SYNTHETIC BENCHMARK—THREE-DIMENSIONAL CASSINI SURFACE



(a) Cassini data set



(b) Final mesh of ORD



(c) Final mesh of LD

FIG. 23: The comparison of the ORD mesh and the LD mesh on *Cassini* for $\bar{h}(I, M) = 2$, $\bar{h}(M, I) = 2$ and $\bar{\theta} = 19.47°$ on topology.

Figure 23 shows the output meshes of the proposed ORD algorithm and the LD algorithm for *Cassini* of a fixed size of 100 voxels in each dimension. We set the symmetric Hausdorff distance 2 voxels and the minimum dihedral angle bound

19.47°. Figure 23c shows the final mesh of the LD algorithm. From the input data set, the original topology is "face-to-face" connect, while the part of LD mesh in the roomed-in view does not maintain the topology of the original image, although it maintains the tissue connectivity. The final mesh of the ORD algorithm (shown in 23b) shows "face-to-face" connect in the part of the mesh in the zoomed-in view, therefore maintains the topology of the original image.

TABLE 2: The comparison of final number of tetrahedra for ORD and LD

$\bar{h}(I,M)$	$\bar{h}(M,I)$	Resulting number of tetrahedra			
		$\bar{\theta} = 5°$		$\bar{\theta} = 10°$	
		ORD	LD	ORD	LD
0	0	1,804,020	2,759,879	1,985,280	3,086,398
0	1	1,772,499	2,251,579	1,990,606	2,534,690
0	2	19,294	1,164,428	50,621	1,423,068
1	0	2,536,685	2,737,787	2,877,747	3,058,038
1	1	1,769,593	2,107,505	1,990,916	2,351,398
1	2	17,004	718,675	48,846	915,237
2	0	2,031,825	2,769,166	2,232,203	3,044,782
2	1	1,747,493	2,047,444	1,925,185	2,272,424
2	2	16,826	512,485	43,744	629,291

$\bar{h}(I,M)$	$\bar{h}(M,I)$	Resulting number of tetrahedra			
		$\bar{\theta} = 15°$		$\bar{\theta} = 19.47°$	
		ORD	LD	ORD	LD
0	0	2,198,280	3,505,180	2,799,217	5,415,193
0	1	2,338,183	2,990,277	3,419,832	4,756,283
0	2	198,320	1,940,147	866,719	4,659,695
1	0	3,263,521	3,533,180	4,253,085	5,270,459
1	1	2,329,039	2,766,013	3,414,341	4,435,609
1	2	193,271	1,368,816	856,591	3,946,446
2	0	2,454,476	3,514,740	3,137,021	5,500,846
2	1	2,186,278	2,685,738	3,029,784	4,675,325
2	2	193,687	1,034,359	855,199	3,700,563

Table 2 shows the comparison of the output mesh size of the ORD algorithm and LD algorithm for a Cassini surface of a fixed diameter of 400 voxels. In these tests, we fixed the size of the Cassini surface and the dihedral angle bound, and vary each of the two one-sided Hausdorff distance bound parameters independently. The different interesting values of the dihedral angle bound were set to 5°, 10°, 15°, and 19.47° spread through the range of its feasible values (0° to 19.47°). As we can see from Table 2, in every case, the size of the output mesh generated by the ORD algorithm is smaller than the size of the output mesh generated by the LD algorithm.

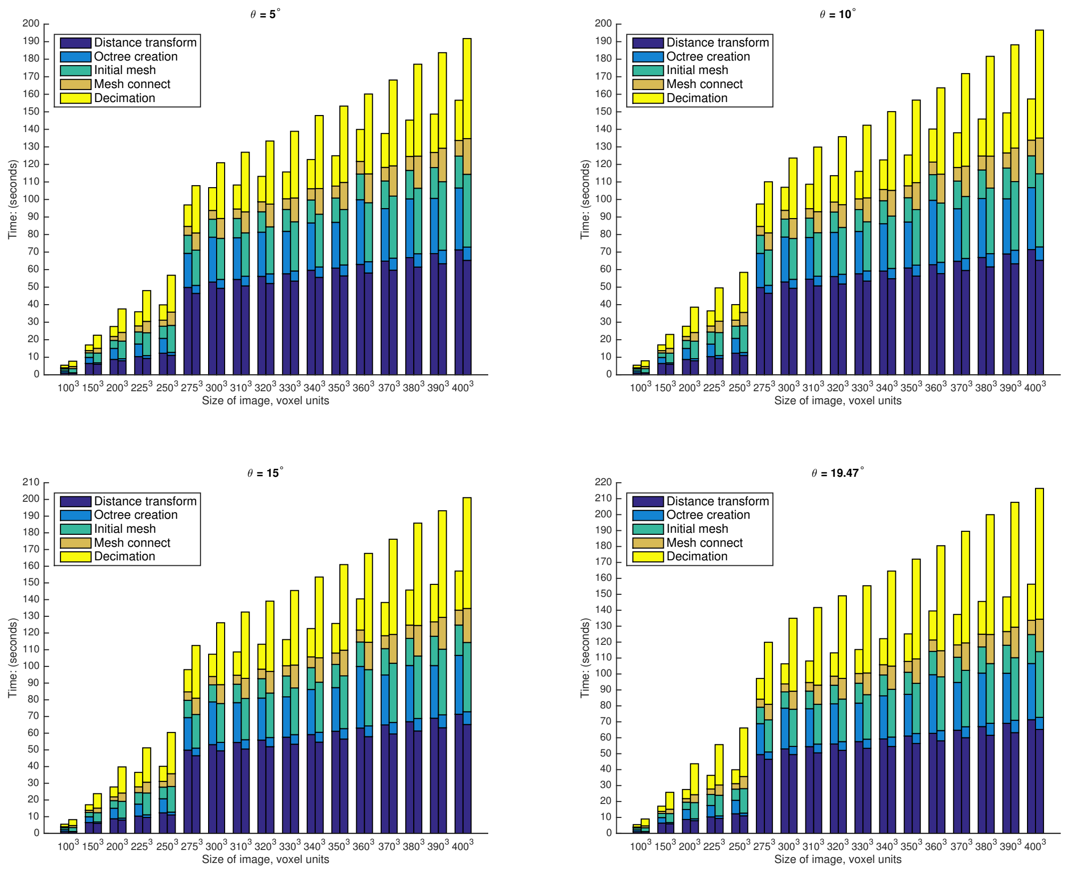


FIG. 24: A breakdown of the total run time of ORD and LD into the major computational parts, as the diameter of the *Cassini* varies from 100 to 400 voxels. $\bar{h}(I, M) = 2$, $\bar{h}(M, I) = 2$. The left bar shows the ORD time and the right bar shows the LD time.

The reason is that LD refines the octree to the deepest level so that the initial mesh has a large number of elements. But ORD refines the octree to a smaller depth so that the initial mesh contains a lower number of elements. The output mesh sizes also decrease proportional to the quality and fidelity parameters, since the weaker constraints can be satisfied with a smaller number of tetrahedra: the output mesh size is high when $\bar{h}(I, M)$ and(or) $\bar{h}(M, I)$ is low, and decreases as $\bar{h}(I, M)$ and(or) $\bar{h}(M, I)$ increases. Similarly, the final number of tetrahedra decreases as the dihedral angle bound decreases.

In time measurements we exclude the time taken by input and output and by the process of initializing the data structure representing the initial image. Figure 24 shows a breakdown of the total running time into the main computational components, as the diameter of the *Cassini* grows from 100 to 400 voxels, and $\bar{h}(I, M) = 2$, $\bar{h}(M, I) = 2$. These parts are the computation of the distance transform, the construction of the octree, the construction of the initial mesh that fills the leaves of the octree, the finding of the connectivity among the tetrahedra of the initial mesh, and the decimation. We conclude that all components represented in the figure grow approximately linear with respect to the total number of voxels in the image, while the sharp jump between the diameter values of 250 and 275 corresponds to the increase of the octree size which can only take values of powers of two. For all configurations the ORD algorithm is less expensive than the LD algorithm. The octree construction of the ORD algorithm is more expensive than the LD algorithm, since to generate a much coarser mesh, extra computations have to spend on the simple manifold condition and fidelity check. However, the time was saved on the initial mesh construction, connection, and decimation, because the number of elements in the initial ORD mesh is much lower than the number of elements in the initial LD mesh.

5.1.2 MULTI-TISSUE THREE-DIMENSIONAL MEDICAL IMAGES

For the following examples, the input of the algorithm is segmented medical images with multiple labels. Table 3 shows the information about the input images. Before meshing the images, we resampled them with voxels of equal side length corresponding to the smallest original units to obtain equally spaced images that were used for meshing. Figure 25, 26, 27 and 28 show the final meshes produced on these input images and the corresponding cut views.

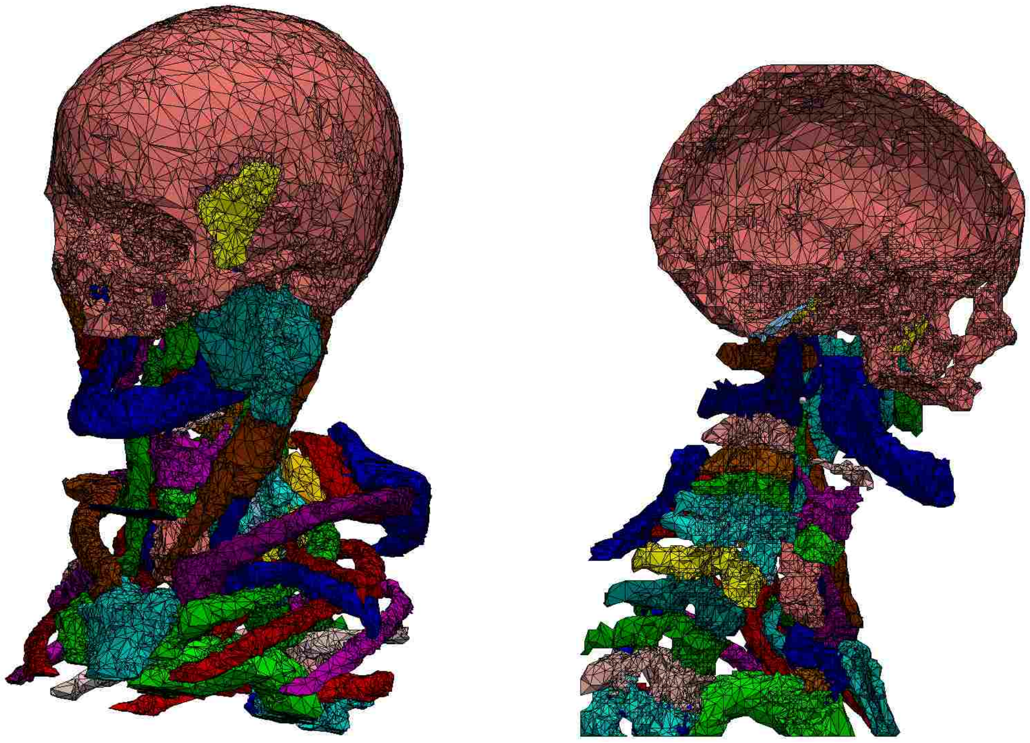
TABLE 3: Information about the multi-material medical images

Image ID	Image name	Resolution	Spacing (mm^3)	Tissues
1	*head neck*	$255 \times 255 \times 229$	$0.97 \times 0.97 \times 1.40$	60
2	*knee*	$512 \times 512 \times 119$	$0.27 \times 0.27 \times 1.00$	49
3	*abdomen*	$512 \times 512 \times 219$	$0.96 \times 0.96 \times 2.40$	22
4	*abdomen background*	$512 \times 512 \times 219$	$0.96 \times 0.96 \times 2.40$	23

In Table 4 we show the comparison of the output mesh size of the ORD algorithm and LD algorithm for the four input images. We fixed the two-sided Hausdorff distance bound parameters, and vary the dihedral angle bound to the value 5°, 10°, 15°, and 19.47° spread through the range of its feasible values (0° to 19.47°). For each fixed value of parameter $\bar{h}(I, M)$ and $\bar{h}(M, I)$, the first row shows the number of tetrahedra before decimation, and the rest rows show the final number of tetrahedra after decimation with different values of parameter θ. As we can see from Table 4, the output mesh size is low when either or both $\bar{h}(I, M)$ $\bar{h}(M, I)$ is high for all configurations. Also, the final number of tetrahedra decreases as the dihedral angle bound decreases. The tetrahedra generated by ORD before decimation is much fewer than the tetrahedra generated by LD before decimation. When $\bar{H}(I, M) = 0$ and $\bar{H}(I, M) = 1$, the sizes of ORD meshes are slightly smaller than the sizes of LD meshes. However, when $\bar{H}(I, M) = 2$, the ORD algorithm generated a significant smaller number of tetrahedra compared to the number of tetrahedra generated by the LD algorithm. We conclude that the ORD algorithm performs better in terms of the number of tetrahedra than LD algorithm even though it maintains the topology.

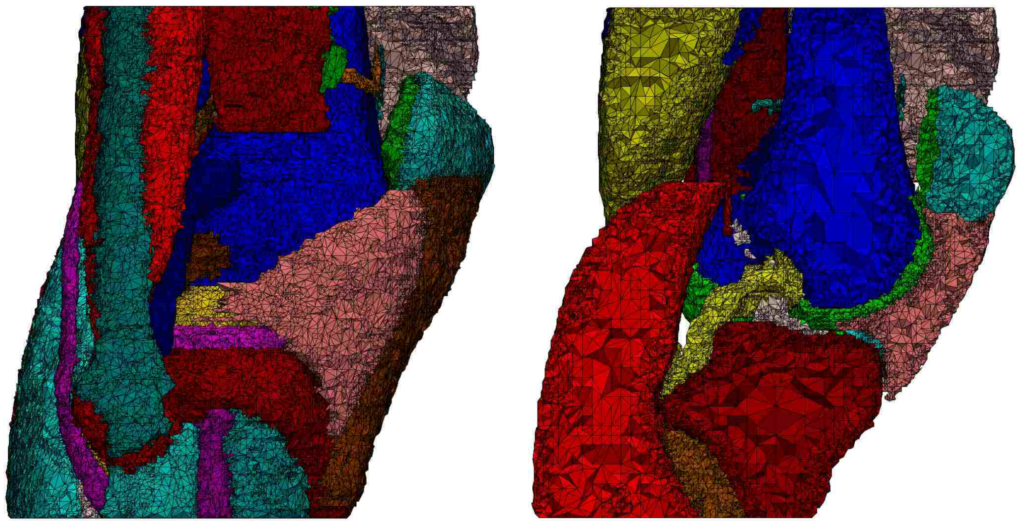
Figure 29 shows breakdowns of the total running time into the main computational components as the symmetric Hausdorff distance bound set to 2 voxels and the angle bound varies among 5°, 10°, 15° and 19.47° on the four input images. For image *head neck*, the performance of LD algorithm is the same (when the angle bound is 19.47°) or slightly better than the ORD algorithm, while in all the other cases, the ORD algorithm performs better than LD algorithm. The reason is that for a small data set where the surface-to-volume ratio is high, most computations are spent on constructing a coarse mesh in the octree construction step. But for large data sets, once the coarse surface is constructed, it grades quickly towards the center.

Figure 30 and Figure 31 gives insight how the run time distributed in the octree construction and the decimation respectively. These two figures show how we arranged the sequences of pass-fail condition evaluations such that the least expensive conditions and those most likely to fail are evaluated first. For example, in the octree construction, the simple manifold condition is conducted firstly and the two-sided Hausdorff distance evaluation is conducted lastly because the evaluation of the simple manifold condition on a single leaf is not expensive. Similarly in the decimation, the quality evaluation is conducted firstly and the topology evaluation is conducted finally because the evaluation of the simple manifold condition on multiple leaves is more expensive.



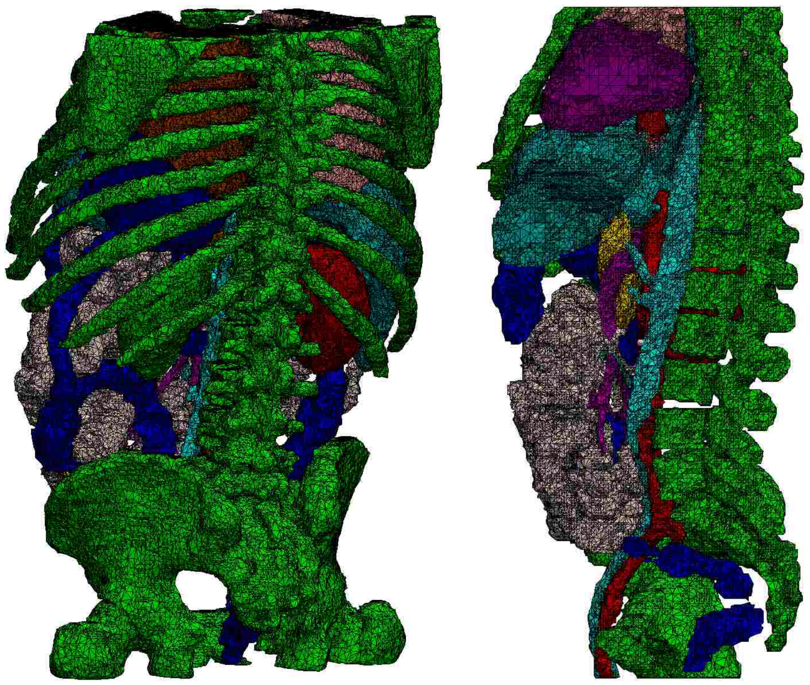
(a) Final mesh (b) A cut view

FIG. 25: The ORD mesh and a cut view of the *head neck* for $\bar{h}(I, M) = 2$, $\bar{h}(M, I) = 2$ and $\bar{\theta} = 15°$.



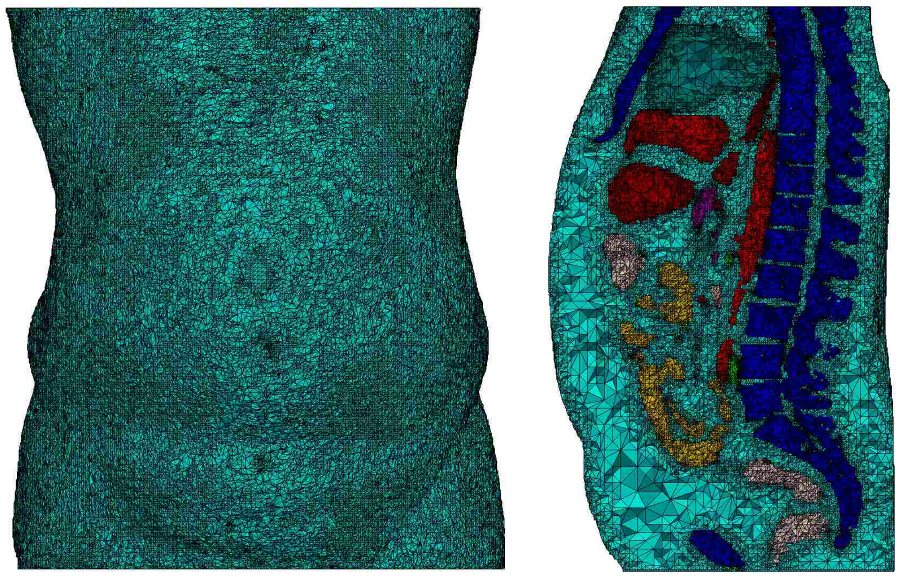
(a) Final mesh (b) A cut view

FIG. 26: The ORD mesh and a cut view of the *knee* for $\bar{h}(I, M) = 2$, $\bar{h}(M, I) = 2$ and $\bar{\theta} = 15°$.



(a) Final mesh (b) A cut view

FIG. 27: The ORD mesh and a cut view of the *abdomen* for $\bar{h}(I, M) = 2$, $\bar{h}(M, I) = 2$ and $\bar{\theta} = 19.47°$.



(a) Final mesh (b) A cut view

FIG. 28: The ORD mesh and a cut view of the *abdomen background* for $\bar{h}(I, M) = 2$, $\bar{h}(M, I) = 2$ and $\bar{\theta} = 19.47°$.

TABLE 4: The comparison of final number of tetrahedra for ORD and LD

$\bar{h}(I,M)=0$, $\bar{h}(M,I)=0$				
Input	*head neck*		*knee*	
Algorithm	ORD	LD	ORD	LD
Before decimation	6,327,256	7,594,020	35,320,498	46,373,356
After decimation with $\bar{\theta}=19.47°$	3,426,932	3,989,203	15,444,410	20,030,904
After decimation with $\bar{\theta}=15.00°$	2,981,503	3,143,790	12,244,059	13,844,888
After decimation with $\bar{\theta}=10.00°$	2,810,071	2,924,632	11,094,344	12,412,983
After decimation with $\bar{\theta}=5.00°$	2,651,199	2,701,699	10,100,664	11,147,366

$\bar{h}(I,M)=0$, $\bar{h}(M,I)=0$				
Input	*abdomen*		*abdomen background*	
Algorithm	ORD	LD	ORD	LD
Before decimation	32,237,816	42,202,662	78,093,018	104,106,742
After decimation with $\bar{\theta}=19.47°$	14,250,189	18,163,216	33,103,866	43,542,841
After decimation with $\bar{\theta}=15.00°$	11,374,746	12,656,733	25,878,728	29,372,760
After decimation with $\bar{\theta}=10.00°$	10,328,705	11,329,731	23,334,857	26,138,425
After decimation with $\bar{\theta}=5.00°$	9,424,694	10,146,908	21,171,900	23,372,206

$\bar{h}(I,M)=1$, $\bar{h}(M,I)=1$				
Input	*head neck*		*knee*	
Algorithm	ORD	LD	ORD	LD
Before decimation	6,103,367	7,579,516	32,888,166	46,544,124
After decimation with $\bar{\theta}=19.47°$	3,260,361	3,886,950	14,068,574	19,032,469
After decimation with $\bar{\theta}=15.00°$	2,817,508	3,024,995	11,080,778	12,588,781
After decimation with $\bar{\theta}=10.00°$	2,652,812	2,806,558	10,082,613	11,160,507
After decimation with $\bar{\theta}=5.00°$	2,513,153	2,588,679	9,243,276	9,935,281

$\bar{h}(I,M)=1$, $\bar{h}(M,I)=1$				
Input	*abdomen*		*abdomen background*	
Algorithm	ORD	LD	ORD	LD
Before decimation	30,530,656	42,344,304	75,573,274	104,024,688
After decimation with $\bar{\theta}=19.47°$	13,253,140	17,608,762	31,410,834	41,943,017
After decimation with $\bar{\theta}=15.00°$	10,533,610	11,994,099	24,126,057	27,470,652
After decimation with $\bar{\theta}=10.00°$	9,607,409	10,646,685	21,706,906	24,187,838
After decimation with $\bar{\theta}=5.00°$	8,823,564	9,488,867	19,751,996	21,567,190

$\bar{h}(I,M)=2$, $\bar{h}(M,I)=2$				
Input	*head neck*		*knee*	
Algorithm	ORD	LD	ORD	LD
Before decimation	4,472,537	8,046,988	24,144,753	56,415,318
After decimation with $\bar{\theta}=19.47°$	1,013,863	1,992,313	5,677,360	14,198,097
After decimation with $\bar{\theta}=15.00°$	475,250	654,756	2,135,102	3,837,660
After decimation with $\bar{\theta}=10.00°$	302,740	456,699	1,121,325	2,335,844
After decimation with $\bar{\theta}=5.00°$	227,256	367,618	740,320	1,881,087

$\bar{h}(I,M)=2$, $\bar{h}(M,I)=2$				
Input	*abdomen*		*abdomen background*	
Algorithm	ORD	LD	ORD	LD
Before decimation	19,927,216	51,284,042	52,484,086	128,702,656
After decimation with $\bar{\theta}=19.47°$	4,189,035	12,434,028	12,961,174	35,150,634
After decimation with $\bar{\theta}=15.00°$	1,699,631	3,426,170	4,796,130	8,890,392
After decimation with $\bar{\theta}=10.00°$	980,373	2,081,046	2,298,430	5,057,907
After decimation with $\bar{\theta}=5.00°$	683,145	1,644,226	1,398,060	4,062,051

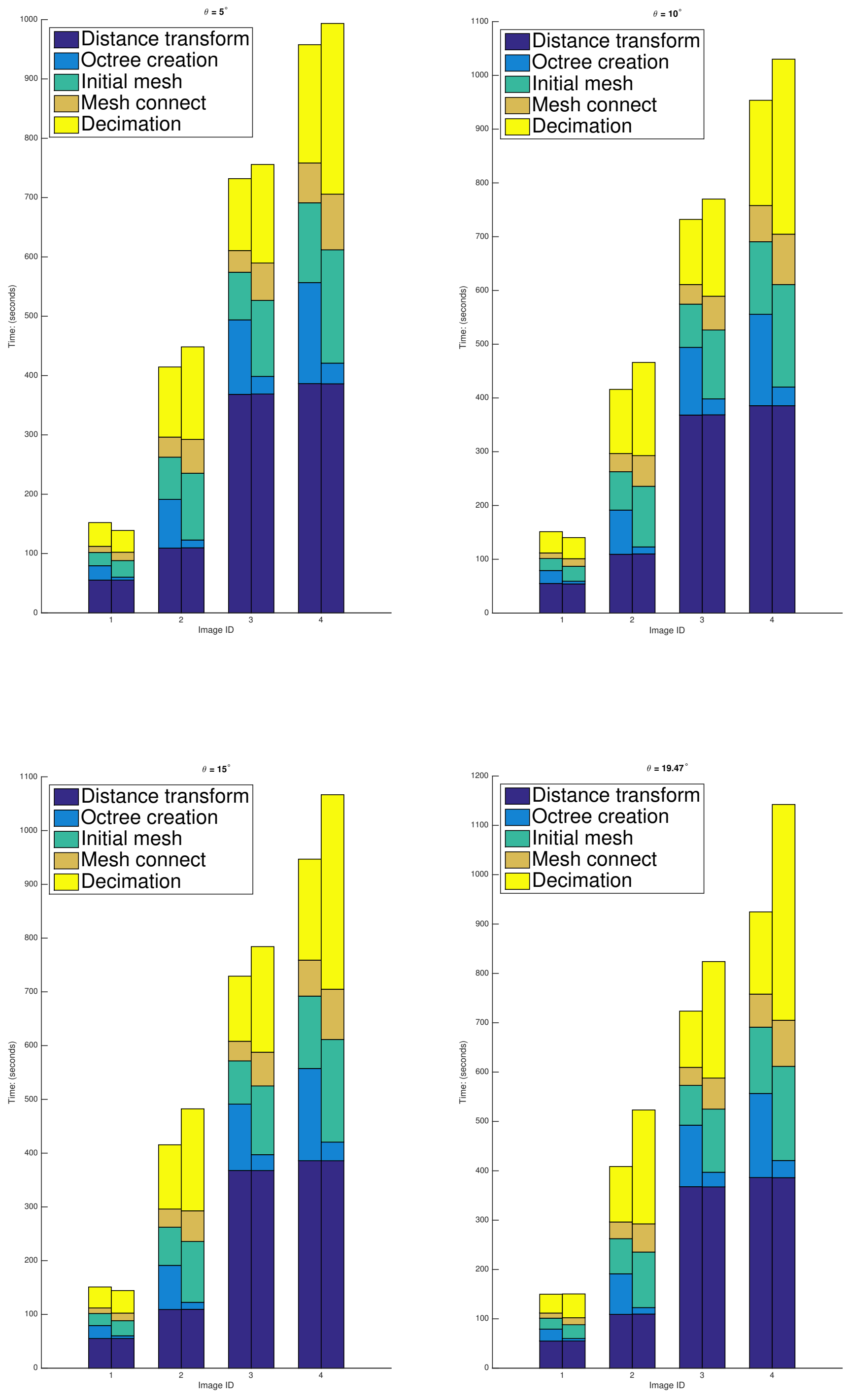


FIG. 29: Comparison of the breakdowns of the total running time into the main computational components for ORD and LD on the four input data sets for $\bar{h}(I, M) = 2$, $\bar{h}(M, I) = 2$ and $\bar{\theta}$ varies among 5°, 10°, 15° and 19.47°. The left bar shows the ORD time and the right bar shows the LD time.

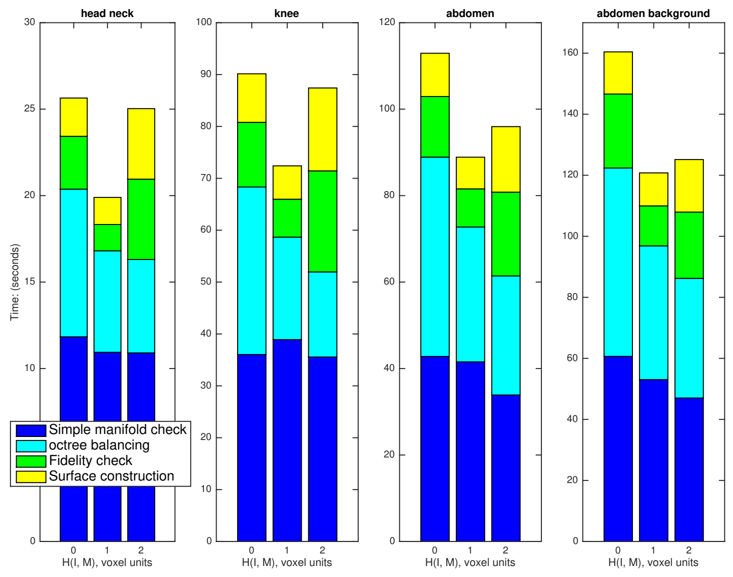


FIG. 30: The breakdowns of the octree creation time into four computational components for ORD on the four input data sets. $\bar{H}(I, M)$ varies from 0 to 2.

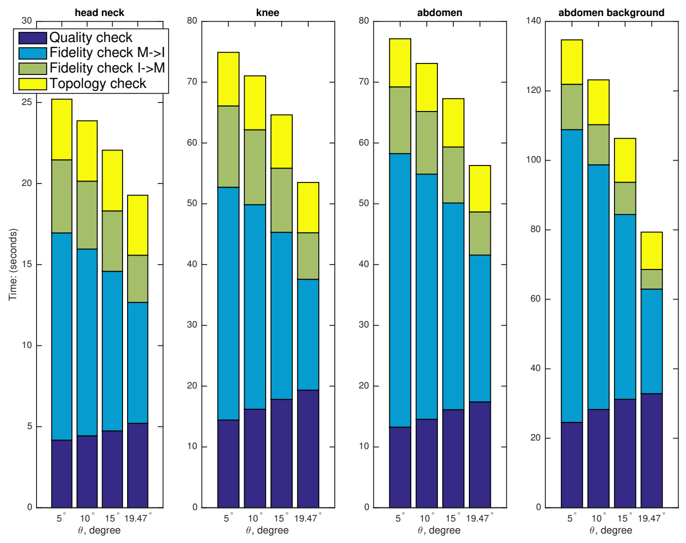


FIG. 31: The breakdowns of the decimation time into four computational components for ORD on the four input data sets for $\bar{h}(I, M) = 2$ and $\bar{h}(M, I) = 2$ and $\bar{\theta}$ varies among 5°, 10°, 15° and 19.47°.

A histogram of all dihedral angles is shown for each example in Figure 32. Empty bars show zero frequency. From Figure 32, all the dihedral angles are distributed between 15° and 165°, and the three peaks occur at 45°, 90° and 135°.

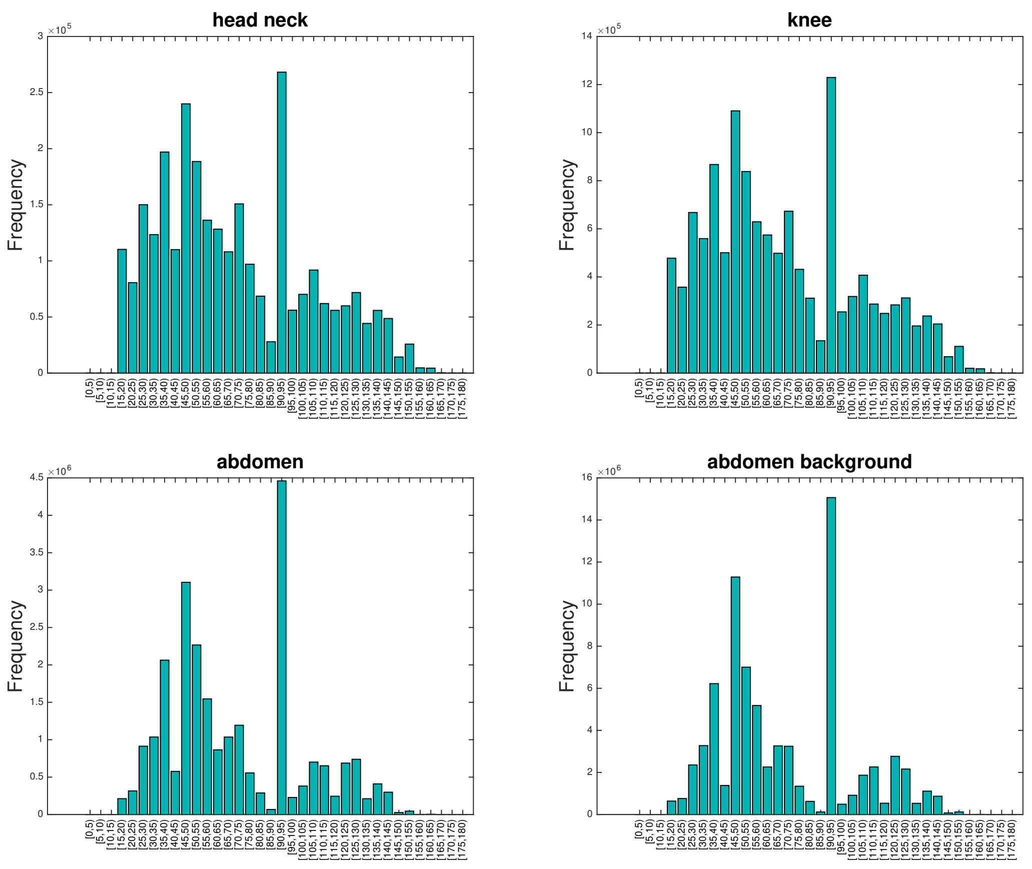


FIG. 32: A histogram of all dihedral angles for each of the output meshes in Figure 25, 26, 27 and 28. Empty bars show zero frequency.

We also conducted an experiment using an open source mesh generator Computational Geometry Algorithms Library (CGAL) [53] and compare the performance with the performance of the ORD algorithm. Table 5, 6 and 7 present our experimental evaluation of the I2M conversion functionality *make_mesh_3* offered by CGAL and by ORD. The parameters of function *make_mesh_3* are described below:

1. *Input domain*: 3D labeled images *head neck*, *knee*, *abdomen* and *abdomen background* without re-sampling.
2. *facet_angle*: A lower bound for the angle (in degree) of surface facets; we set it to an ignored value.
3. *facet_size*: An upper bound for the radii of surface Delaunay balls; it controls the

size of surface facets; we set it to an ignored value.
4. *facet_distance*: An upper bound for the distance between the circumcenter of a surface facet and the center of a surface Delaunay ball of this facet; it controls the approximation error of boundary and subdivision surfaces; we varied this parameter as shown in the table.
5. *facet_topology*: it controls the set of topological constraints which have to be verified by each surface facet; by default, each vertex of a surface facet has to be located on a surface patch, on a curve segment, or on a corner; it can also be set to check whether the three vertices of a surface facet belongs to the same surface patch; we set it to an ignored value.
6. *cell_radius_edge_ratio*: an upper bound for the ratio between the circumradius of a mesh tetrahedron and its shortest edge; we used 2.0.
7. *cell_size*: an upper bound on the circumradii of the mesh tetrahedra; we set it to an ignored value.
8. *lloyd(), odt(), perturb(), exude()*: they control which optimization processes are performed; we used four combinations specified in the table.

In Table 5, 6 and 7, we vary the value of parameter *facet_distance*, and show $h(M, I)$ and $h(I, M)$. They are measured by resampled voxel unit. We also list the final number of tetrahedra produced by CGAL, the minimum dihedral angle (measured by degree) and the total running time (measured by seconds). In some cases, when the value of parameter *facet_distance* increased, the value of $h(M, I)$ increased, however, there is no obvious relationship between *facet_distance* and $h(M, I)$. Furthermore, the values of $h(I, M)$ in all the cases are unreasonably large. We conclude that CGAL can only approximate one-sided Hausdorff distance, while the ORD algorithm can always guarantee that the Hausdorff distance bound is two-sided. CGAL improves mesh properties such as the dihedral angles using various combinations of optimization algorithms, however, it could only obtain the best minimum dihedral angles $5°$. The number of tetrahedra and processing time of CGAL varies significantly, from much lower than that of the ORD algorithm to order of magnitude higher, depending on the selection of mesh optimization algorithms.

TABLE 5: The comparison of final number of tetrahedra and run time for ORD and CGAL on *head neck*

head neck											
Opt.		*no_lloyd*(), *no_odt*(), *perturb*(), *exude*()					*lloyd*(), *no_odt*(), *perturb*(), *exude*()				
Algorithm	*facet_dist.*	$h(M,I)$	$h(I,M)$	dih. angle	# of tets	Total time	$h(M,I)$	$h(I,M)$	dih. angle	# of tets	Total time
CGAL	0.1	3	58	2.07	11,379,822	724.76	5	58	2.00	10,936,756	6190.44
ORD	N/A	3	58	2.07	770,600	224.13	5	58	2.00	768,883	224.79
CGAL	0.2	2	58	1.71	2,059,641	116.20	3	58	2.26	2,018,163	821.98
ORD	N/A	2	58	1.71	772,620	225.50	3	58	2.26	772,834	224.83
CGAL	0.3	2	58	2.43	984,846	55.34	2	58	2.50	970,894	353.20
ORD	N/A	2	58	2.43	777,995	224.90	2	58	2.50	778,166	226.52
CGAL	0.4	2	58	3.03	558,328	32.29	2	58	3.11	551,638	191.66
ORD	N/A	2	58	3.03	782,480	225.54	2	58	3.11	781,708	226.20
Opt.		*no_lloyd*(), *odt*(), *perturb*(), *exude*()					*lloyd*(), *odt*(), *perturb*(), *exude*()				
Algorithm	*facet_dist.*	$h(M,I)$	$h(I,M)$	dih. angle	# of tets	Total time	$h(M,I)$	$h(I,M)$	dih. angle	# of tets	Total time
CGAL	0.1	3	58	1.01	11,131,809	2693.19	3	58	1.17	10,938,526	6708.72
ORD	N/A	3	58	1.01	765,806	226.82	3	58	1.17	766,055	224.97
CGAL	0.2	2	58	1.07	2,048,964	414.66300	2	58	3.02	2,017,204	872.25
ORD	N/A	2	58	1.07	769,420	227.33	2	58	3.02	782,426	227.41
CGAL	0.3	2	58	0.15	985,924	189.429000	2	58	0.64	970,824	399.63
ORD	N/A	2	58	0.15	766,283	226.77	2	58	0.64	767,831	226.91
CGAL	0.4	3	58	1.40	560,335	104.13300	3	59	3.06	552,930	226.41
ORD	N/A	3	58	1.40	767,869	225.77	3	59	3.06	778,644	225.41

TABLE 6: The comparison of final number of tetrahedra and run time for ORD and CGAL on *knee*

knee											
Opt.		*no_lloyd*(), *no_odt*(), *perturb*(), *exude*()					*lloyd*(), *no_odt*(), *perturb*(), *exude*()				
Algorithm	*facet_dist.*	$h(M,I)$	$h(I,M)$	dih. angle	# of tets	Total time	$h(M,I)$	$h(I,M)$	dih. angle	# of tets	Total time
CGAL	0.1	3	29	1.50	3,077,237	139.47	3	29	1.62	3,010,499	1194.83
ORD	N/A	3	29	1.50	1,885,707	775.24	3	29	1.62	1,935,538	744.58
CGAL	0.2	3	30	2.52	732,553	29.33	4	30	3.03	723,109	253.02
ORD	N/A	3	30	2.52	1,935,538	744.58	4	30	3.03	1,935,538	744.58
CGAL	0.3	4	30	3.33	345,335	15.75	5	30	3.00	342,144	119.78
ORD	N/A	4	30	3.33	1,961,041	745.32	5	30	3.00	1,961,041	745.32
CGAL	0.4	4	30	3.40	203,997	8.13	6	30	2.36	202,755	63.55
ORD	N/A	4	30	3.40	1,963,608	743.67	6	30	2.36	1,963,608	743.67
Opt.		*no_lloyd*(), *odt*(), *perturb*(), *exude*()					*lloyd*(), *odt*(), *perturb*(), *exude*()				
Algorithm	*facet_dist.*	$h(M,I)$	$h(I,M)$	dih. angle	# of tets	Total time	$h(M,I)$	$h(I,M)$	dih. angle	# of tets	Total time
CGAL	0.1	6	29	0.59	3,052,342	545.65	7	29	1.01	3,023,807	1252.05
ORD	N/A	6	29	0.59	1,961,041	745.32	7	29	1.01	1,870,359	781.80
CGAL	0.2	6	30	0.79	732,978	119.23	5	30	1.96	729,632	264.77
ORD	N/A	6	30	0.79	1,935,538	744.58	5	30	1.96	1,910,120	746.03
CGAL	0.3	6	30	1.29	347,603	52.88	5	31	2.30	347,184	122.07
ORD	N/A	6	30	1.29	1,961,041	745.32	5	31	2.30	1,904,132	716.69
CGAL	0.4	8	30	1.02	205,790	31.62	5	30	2.81	206,187	73.00
ORD	N/A	8	30	1.02	1,963,608	743.67	5	30	2.81	1,933,393	746.69

TABLE 7: The comparison of final number of tetrahedra and run time for ORD and CGAL on *abdomen*

abdomen											
Opt.		*no_lloyd*(), *no_odt*(), *perturb*(), *exude*()					*lloyd*(), *no_odt*(), *perturb*(), *exude*()				
Algorithm	*facet_dist.*	$h(M,I)$	$h(I,M)$	dih. angle	# of tets	Total time	$h(M,I)$	$h(I,M)$	dih. angle	# of tets	Total time
CGAL	0.2	3	23	1.49	10,349,057	532.06	6	18	1.61	9,979,982	4558.32
ORD	N/A	3	23	1.49	1,165,177	1024.49	6	18	1.61	1,160,524	1233.29
CGAL	0.4	3	23	2.26	2,042,684	95.80	4	23	1.74	2,000,988	762.00
ORD	N/A	3	23	2.26	1,180,896	1024.59	4	23	1.74	1,163,455	1022.03
CGAL	0.6	4	23	2.35	862,668	39.23	4	24	2.66	849,445	299.09
ORD	N/A	4	23	2.35	1,176,734	1021.54	4	24	2.66	1,193,001	1185.63
CGAL	0.8	4	23	3.05	487,277	27.99	4	24	3.20	481,107	163.09
ORD	N/A	4	23	3.05	1,201,106	1022.16	4	24	3.20	1,207,731	1184.07
Opt.		*no_lloyd*(), *odt*(), *perturb*(), *exude*()					*lloyd*(), *odt*(), *perturb*(), *exude*()				
Algorithm	*facet_dist.*	$h(M,I)$	$h(I,M)$	dih. angle	# of tets	Total time	$h(M,I)$	$h(I,M)$	dih. angle	# of tets	Total time
CGAL	0.2	5	24	1.01	10,178,967	2132.82	5	18	2.01	9,985,320	5079.16
ORD	N/A	5	24	1.01	1,149,725	1179.20	5	18	2.01	1,173,067	1233.10
CGAL	0.4	7	18	0.86	2,031,462	379.13	5	18	2.17	2,002,825	800.18
ORD	N/A	7	18	0.86	1,144,908	1231.44	5	18	2.17	1,172,660	1229.75
CGAL	0.6	6	19	2.04	862,156	152.53	6	19	2.07	851,196	354.28
ORD	N/A	6	19	2.04	1,164,331	1025.44	6	19	2.07	1,170,672	1026.11
CGAL	0.8	8	19	3.05	487,879	85.16	8	23	5.01	481,937	178.30
ORD	N/A	8	19	3.05	1,195,433	1033.67	8	23	5.01	1,257,769	1022.35

5.2 TWO-DIMENSIONAL CURVILINEAR MESHING RESULTS

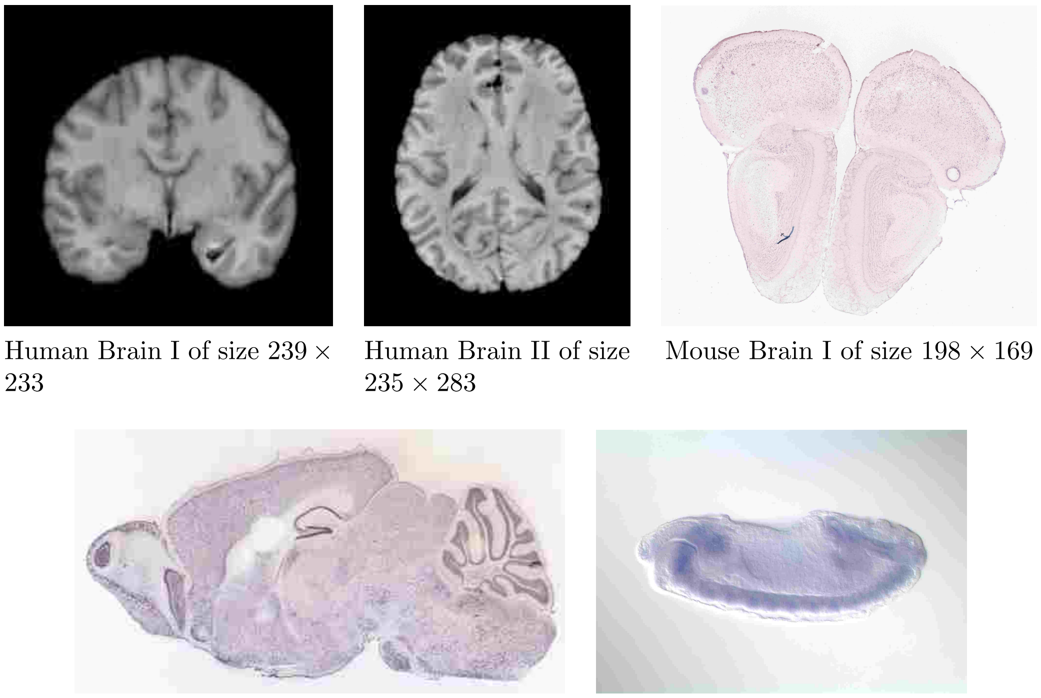
We apply the algorithm to real medical data in the following sections. All the experiments were conducted on a 64 bit machine equipped with two 3.06 GHz 6-Core Intel Xeon CPU and 64 GB main memory. The procedure for mesh untangling and quality improvement was implemented in MATLAB. All the other steps were implemented in C++ for efficiency.

5.2.1 ORIGINAL INPUT IMAGES

The input data to our algorithm is a two-dimensional image. In the following mesh examples, we meshed two slices of the mouse brain image [54], two slices of the human brain image [54], and the fly embryo image [55]. The original images are listed in Fig. 33.

5.2.2 CONSTRUCTION OF LINEAR MESHES

We show the linear mesh results for the original images with different requirements in Figure 34. The fidelity tolerance was specified by 4 pixels for the first slice of the



Human Brain I of size 239 × 233

Human Brain II of size 235 × 283

Mouse Brain I of size 198 × 169

Mouse Brain II of size 460 × 247

Fly Embryo of size 182 × 130

FIG. 33: The original images. Each pixel has side lengths of 1 unit in both x, y directions.

human brain image, and 3 pixels for the second slice of the human brain image; we set 3 pixels for the first slice of the mouse brain image, and 6 pixels for the second slice of the mouse brain image; for the fly embryo image, the fidelity tolerance was specified by 2 pixels. For all the linear mesh results, the mesh vertices that are classified on the mesh boundary were required to be located on the boundary between the background and the tissue of the image. This requirement results in different angle bounds for the linear mesh results: the minimum angle bound of the first slice of the human brain image is 3.2°, of the second slice is 5.4°; the bound of the first slice of the mouse brain image is 3.6°, of the second slice is 2.8°; of the fly embryo image, the bound is 2.8°. The minimum angle bound is an important measure to the quality of the linear mesh (the higher the better), and it also directly contributes to the quality of the curvilinear mesh. For the curved meshes, the quality can not be measured just simply by calculating the planar angles, however, it can be measured by *scaled Jacobian* [44]. The lower minimum angle bound for the linear mesh could lead to worse *scaled Jacobian* after curving the linear mesh boundary to a smooth closed

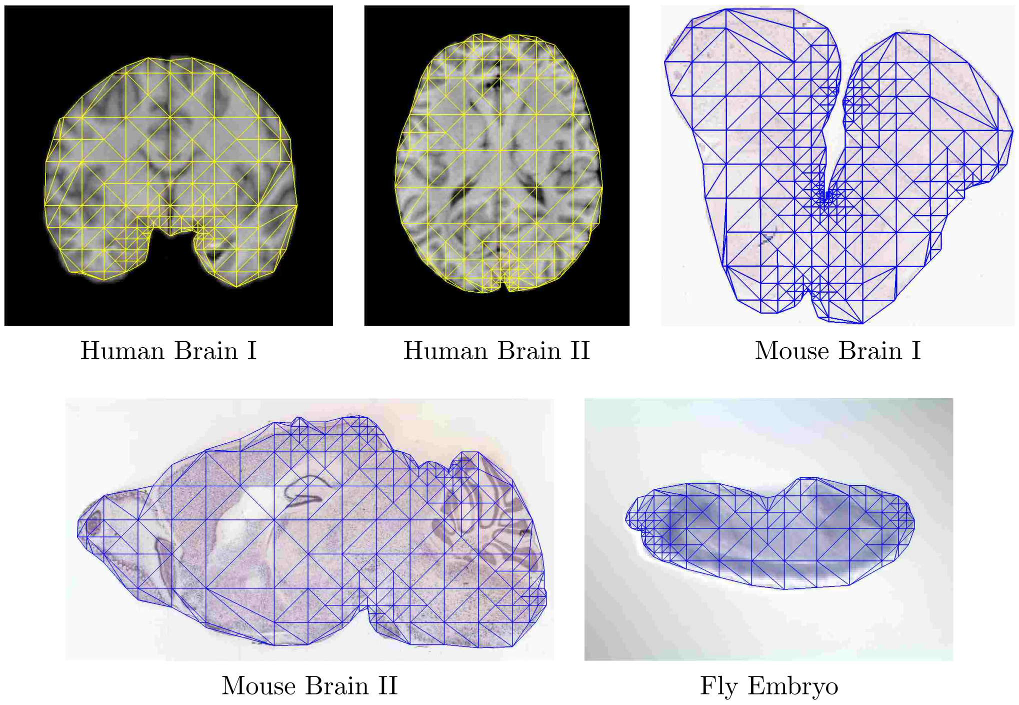


FIG. 34: The linear meshes for the input original images.

path, however, the *scaled Jacobian* can be improved by the iterative FE method. For all the linear mesh results, the elements were not coarsened.

5.2.3 CONSTRUCTION OF SMOOTH CURVED BOUNDARIES AND THE ACCURACY EVALUATION

For each of the above linear meshes, we show the linear mesh boundaries and the curved boundaries with both C^1 and C^2 smoothness requirements. In Figure 35, from left to right for each image, the boundaries are linear boundaries, C^1 boundaries and C^2 boundaries.

The accuracy was specified by the number of misclassified pixels that composed of background pixels that are inside the mesh and tissue pixels that are outside the mesh. The accuracy of the linear mesh results and the corresponding curvilinear meshes with C^1 and C^2 smoothness requirements are listed in Table 8.

For each of the original image with two linear meshing results and their corresponding C^1 and C^2 smooth boundaries, we list the number of background pixels inside the mesh, the number of tissue pixels outside the mesh, the total number of misclassified pixels, the percentage for misclassified pixels out of all pixels and the improved accuracy in percentage for both C^1 and C^2 smooth boundaries compared to the linear mesh boundary. Compare the improved accuracy in percentage in Table 8, both C^1 and C^2 smooth boundaries improved the accuracy of the representation. The improved accuracy also relates to the size of the dataset, usually the larger the image, the more improvement its curvilinear mesh obtained. However, if the linear mesh is a very close representation of the image object, after smoothing the mesh boundary, the accuracy can not improve much. Compare the improved accuracy of the meshes that have C^1 smooth boundaries with those of the meshes that have C^2 smooth boundaries, the C^2 smooth boundaries usually have higher accuracy than the C^1 smooth boundaries, but the differences are not large. We chose the results that have better accuracy to construct the final valid high quality meshes.

TABLE 8: Accuracy of the mesh boundaries

Image	Boundary	# background pixels inside mesh	# tissue pixels outside mesh	# misclassified pixels	Percentage for misclassified pixels (%)	Improved accuracy (%)
Human Brain I	Linear	70	376	446	0.800	N/A
	C^2	138	229	307	0.659	31.166
	C^1	111	268	319	0.681	28.475
Human Brain II	Linear	147	314	461	0.693	N/A
	C^2	215	216	431	0.648	6.508
	C^1	206	201	407	0.648	11.714
Mouse Brain I	Linear	73	308	381	1.139	N/A
	C^2	128	166	294	0.879	22.835
	C^1	95	205	300	0.897	21.260
Mouse Brain II	Linear	293	1073	1366	1.202	N/A
	C^2	243	691	934	0.822	31.625
	C^1	260	687	947	0.834	30.673
Fly Embryo	Linear	39	101	140	0.592	N/A
	C^2	59	83	120	0.507	14.286
	C^1	52	89	123	0.520	12.143

5.2.4 FINAL MESHES AND THE QUALITY EVALUATION

When the linear mesh boundaries were curved to closed smooth paths, and the interior mesh edges remained straight, the invalid elements were created. The number of invalid elements for Human Brain I is 3, for Human Brain II is 1. There are 4 invalid elements for Mouse Brain I, and 9 for Mouse Brain II. The invalid elements are shown in red in Figure 37a, Figure 38a, Figure 39a and Figure 40a. The iterative FE method was applied to the invalid meshes. After 5, 5, 6 and 25 iterations, all the invalid elements were eliminated for these invalid meshes. For Fly Embryo, there is no invalid element (Figure 41a). We executed 10 iterations to improve the quality of the elements. The final meshes are shown in Figure 37b, Figure 38b, Figure 39b, Figure 40b, and Figure 41b.

The quality of the curvilinear meshes was also improved by the iterative FE method. The measure *scaled Jacobian* is defined by:

$$I = \frac{min|J|}{max|J|},$$

where $|J|$ is the *Jacobian* of the mapping from the reference coordinates to the physical coordinates. For a straight-sided element, since its *Jacobian* is a constant, $I = 1$; for a curved element, $I \leq 1$. When the curved element is invalid, I is negative; when it gets degenerated, I approaches to 0. From Figure 36, the iterative FE method produced more elements with larger *scaled Jacobian*, thus the poorly shaped elements were improved significantly.

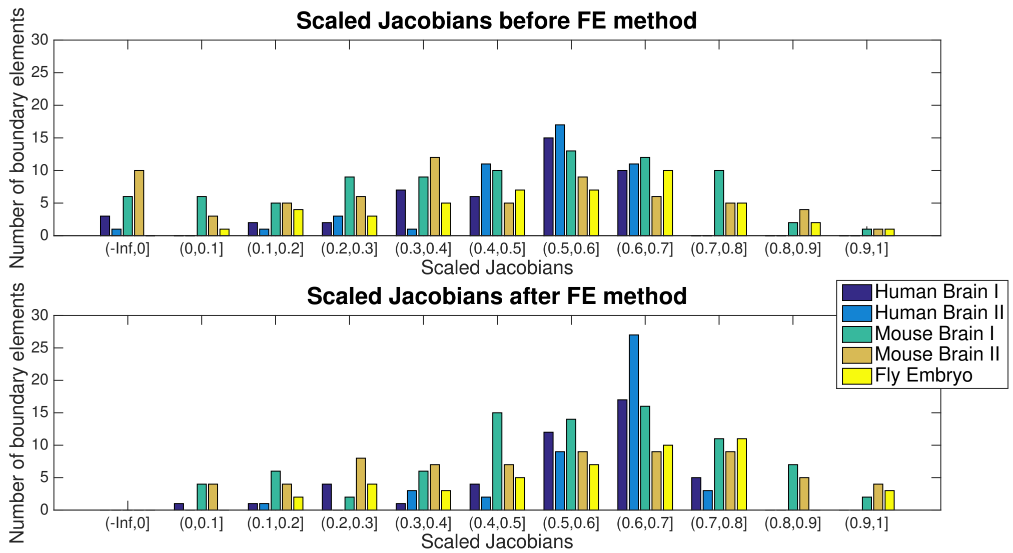
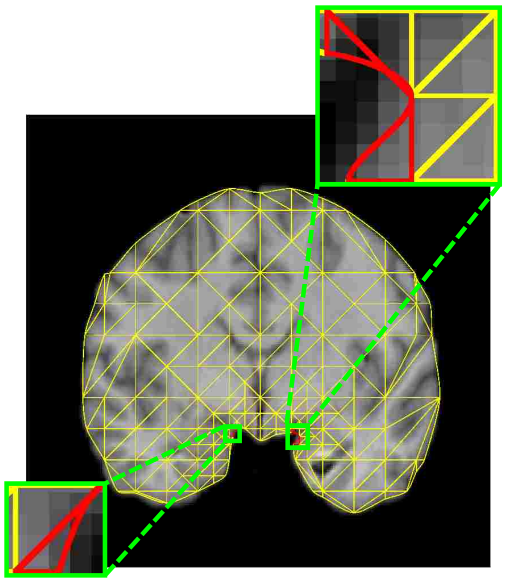


FIG. 36: The comparison of the *scaled Jacobian*.


(a) Invalid mesh with red invalid elements

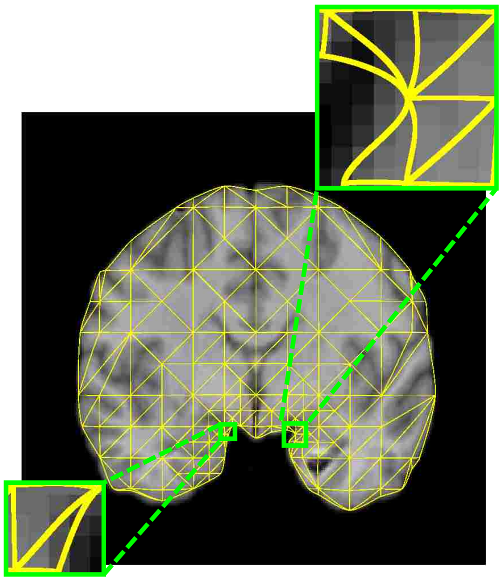
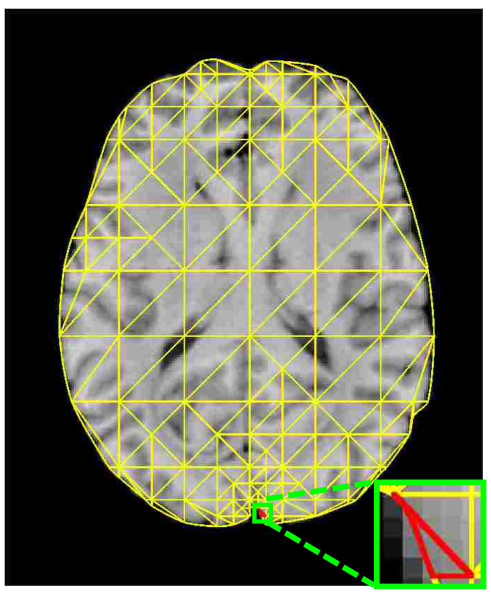

(b) Valid final curvilinear mesh with quality improvement

FIG. 37: Invalid mesh and corresponding corrected mesh for Human Brain I.


(a) Invalid mesh with a red invalid element

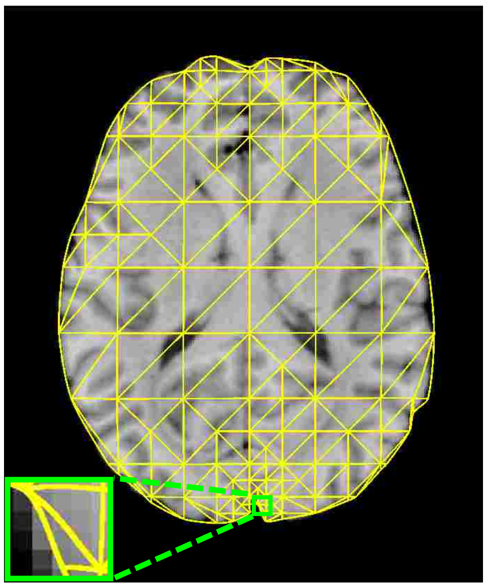
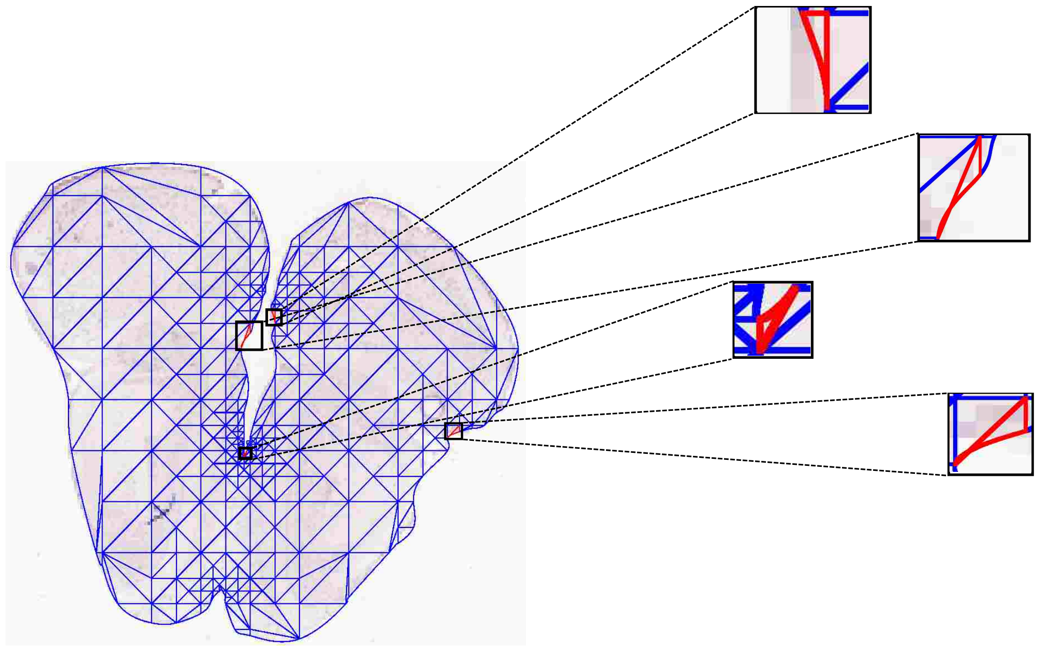
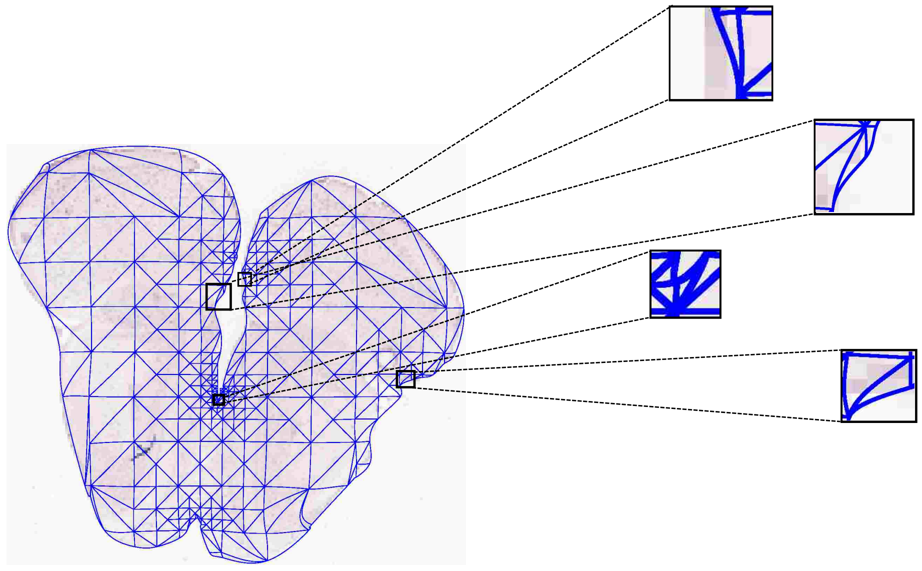

(b) Valid final curvilinear mesh with quality improvement

FIG. 38: Invalid mesh and corresponding corrected mesh for Human Brain II.

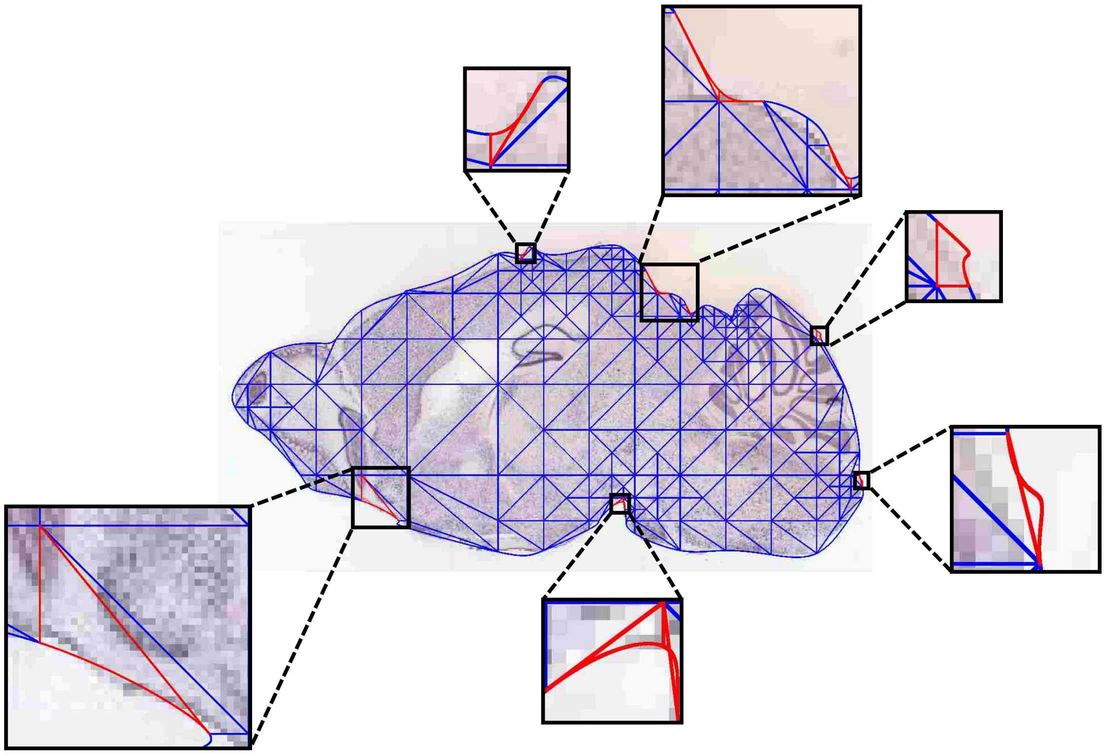


(a) Invalid mesh with red invalid elements

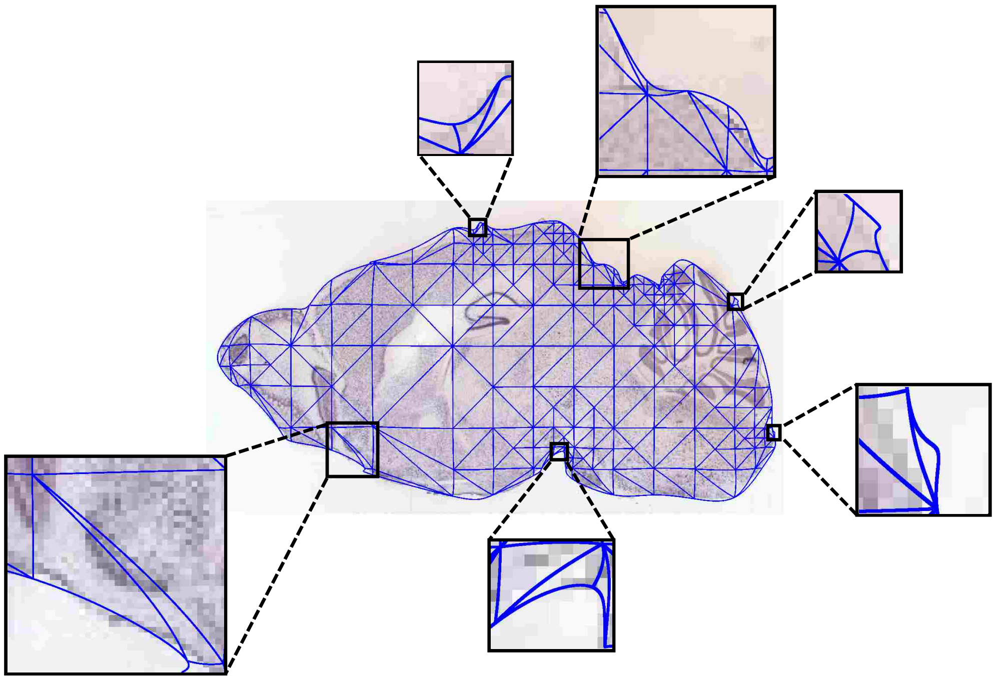


(b) Valid final curvilinear mesh with quality improvement

FIG. 39: Invalid mesh and corresponding corrected mesh for Mouse Brain I.



(a) Invalid mesh with red invalid elements



(b) Valid final curvilinear mesh with quality improvement

FIG. 40: Invalid mesh and corresponding corrected mesh for Mouse Brain II.

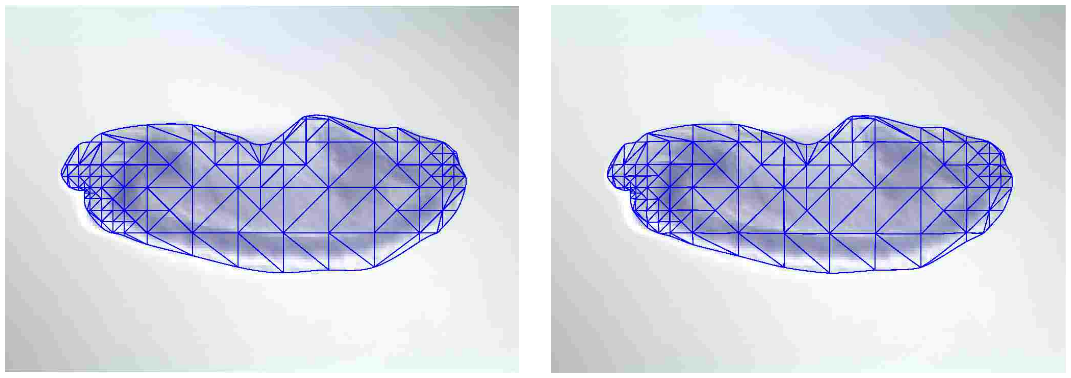

(a) Bad quality curvilinear mesh (b) Improved quality curvilinear mesh

FIG. 41: Bad quality curvilinear mesh and corresponding mesh with quality improvement for Fly Embryo.

The algorithm can also construct curved meshes with coarsened elements inside that have fewer elements. Figure 42 shows the coarsened curvilinear meshes for the five original images.

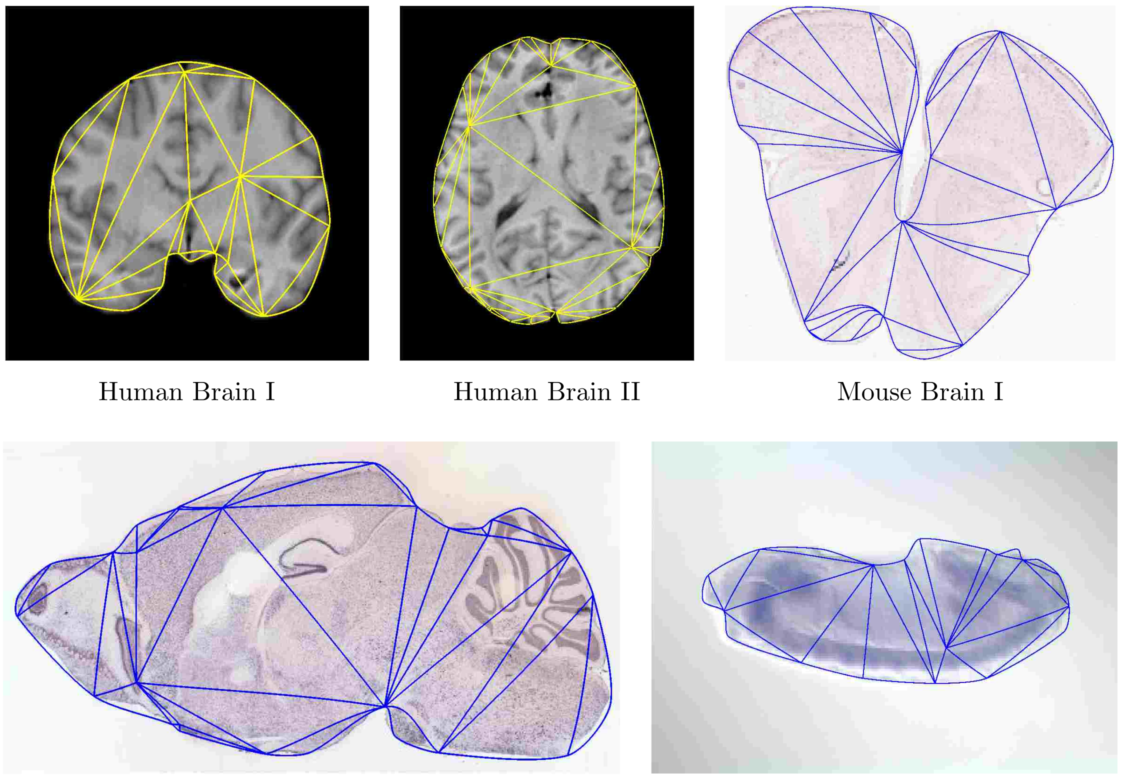


FIG. 42: Curvilinear meshes with coarsened elements for the original images.

5.2.5 PERFORMANCE

In Table 9, we list the total number of elements inside the mesh, the number of invalid elements, the iterations needed to improve the quality of the mesh, the run time of the linear mesh, the time spent on FE method and the total run time. The high-order mesh generator is slower, and most of the time was spent on the FEM iterations. The run time is not only decided by the number of elements inside the mesh, but also determined by how many iterations it needs, because when there are highly distorted invalid elements, more iterations are needed to correct them.

TABLE 9: Run time (s) for the ten examples

Image	# elements inside the mesh	# invalid elements	# iterations of FEM	# run time of linear mesh (s)	run time of FEM (s)	total run time (s)
Human Brain I (fine)	251	3	5	0.239	23.890	24.785
Human Brain II (fine)	235	1	5	0.262	23.212	25.691
Mouse Brain I (fine)	528	4	6	0.169	88.457	89.703
Mouse Brain II (fine)	373	9	25	0.467	221.336	224.917
Fly Embryo (fine)	213	0	10	0.111	42.373	44.357
Human Brain I (coarse)	27	3	12	0.238	16.860	19.079
Human Brain II (coarse)	39	1	80	0.266	40.500	42.391
Mouse Brain I (coarse)	39	4	10	0.164	10.199	13.122
Mouse Brain II (coarse)	37	5	16	0.518	12.298	15.445
Fly Embryo (coarse)	21	3	8	0.119	10.685	14.366

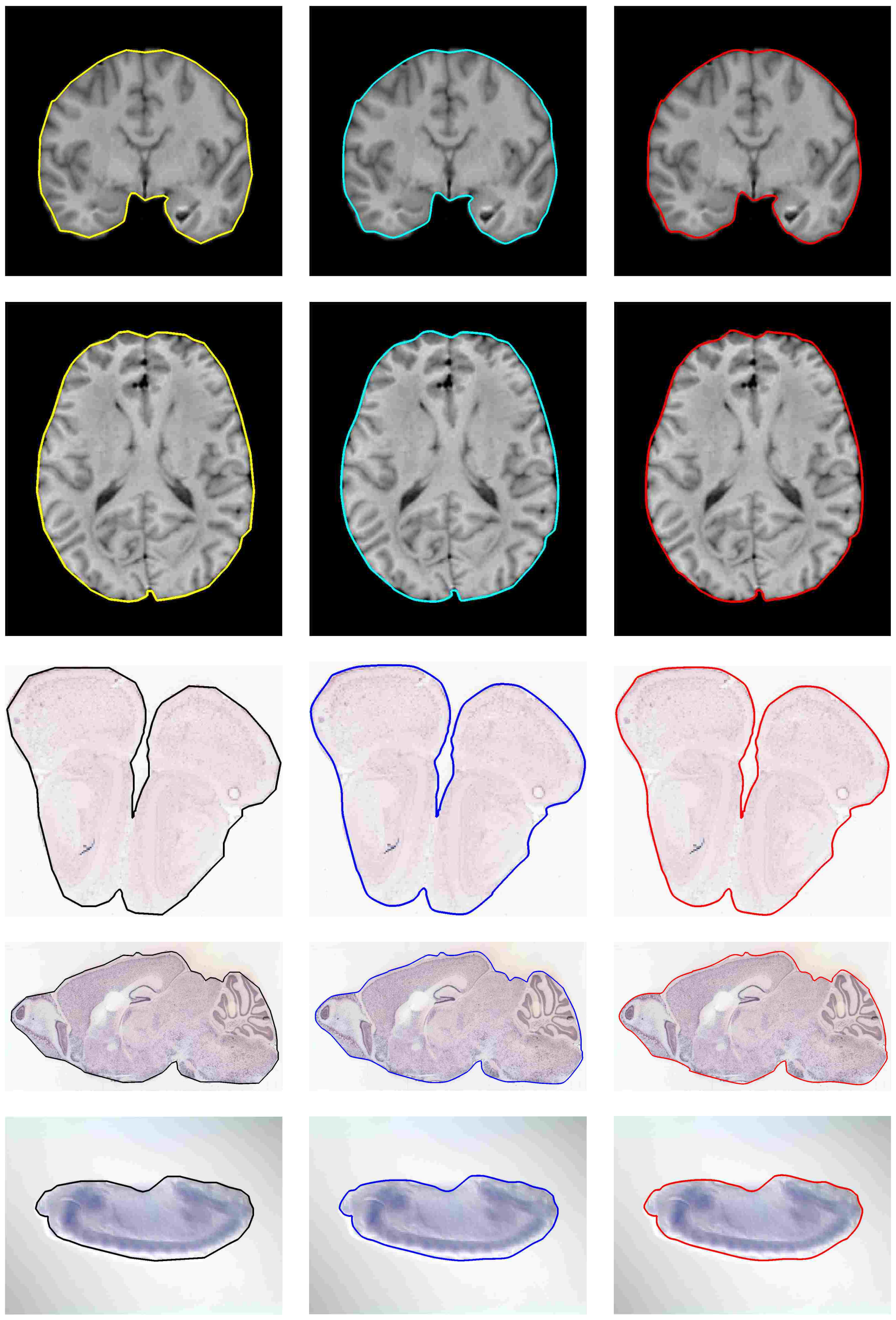


FIG. 35: The linear mesh boundaries and curved mesh boundaries with C^1 and C^2 smoothness requirements.

CHAPTER 6

CONCLUSIONS

6.1 SUMMARY AND EXTENSIONS

Two mesh generation algorithms have been developed:

- Automatic construction of two- and three-dimensional unstructured linear meshes of multi-material images characterized by (i) guaranteed dihedral angle bound for the output tetrahedra, (ii) guaranteed bounds on two-sided Hausdorff distance between the boundaries of the mesh and the boundaries of the materials, (iii) the mesh boundary is proved to be homeomorphic to the object surface, and (iv) a smaller number of mesh elements than other algorithms run with similar parameters.

- Automatic construction of two-dimensional high-order curvilinear meshes to represent images of objects with smooth boundaries. This technique allows for a transformation of straight-sided meshes to curvilinear meshes with C^1 or C^2 smooth boundaries while keeping all elements valid and with good quality as measured by their Jacobians.

The first algorithm creates a coarse mesh with relatively few elements with two reasons: it generates a tetrahedral mesh based on an octrees which offers a graded space subdivision structure; it provides a postprocessing step that decimates the mesh into a much fewer number of elements. It is suitable for a number of applications that impose varying requirements on the size of the elements due to the parameters of the algorithm that allow for precise numerical bounds. Compared to similar algorithms such as LD method, it offers a smoother transition in element size thus achieves better grading. It also offers a topological guarantee, i.e. there is a global homeomorphism between the boundary of the mesh and the boundary of the image. This guarantee comes up with the single manifold condition, a condition we defined that can provide a local homeomorphism in each boundary octree leaf. The time measurements show that for four complex medical atlas images the performance is compatible with or even faster than LD method. We also compare our implementation with a state-of-the-art Delaunay code CGAL. CGAL could only obtain the best minimum dihedral angles 5°, however, the best minimum dihedral angles bound of the proposed algorithm is 19.47°. When we set similar input parameters as CGAL, the performance has a significant improvement in terms of runtime and number of tetrahedra.

The second algorithm takes the output of the first algorithm as the input. The transformation of linear mesh boundaries to C^1 or C^2 smooth boundaries offers higher accuracy of the representation compared to the corresponding linear mesh. By using the Bézier basis, the Jacobians which measure the validity of the high-order mesh element can be written as a fourth-order Bézier triangle. The split of the fourth-order Bézier triangle provides a tight lower-bound of the Jacobians, thus offers an accurate validity check compared to sampling Jacobian at discrete locations. The experimental results show that the iterative finite element method is able to correct all the invalid elements in the high-order mesh as measured by their Jacobians.

The proposed approach can be extended to the three-dimensional curvilinear meshing problem. As the first step a three-dimensional linear mesh is generated, then a transformation of straight-sided meshes to curvilinear meshes is performed. Similarly, the invalid elements can be detected by their Jacobians, and finally are corrected using the equilibrium configuration of an elasticity problem.

6.2 PUBLICATIONS

The following is a list of publications related to this dissertation:

- Jing Xu and Andrey Chernikov, Tetrahedralization of Multi-material Images with Quality Fidelity and Topological Guarantees, unpublished manuscript in preparation.

- Jing Xu and Andrey Chernikov, Construction of Discrete Descriptions of Biological Shapes through Curvilinear Image Meshing, International Journal of Bioinformatics Research and Applications, vol. 15(3), pages 272-295. 2019.

- Jing Xu and Andrey Chernikov, Homeomorphic Tetrahedralization of Multi-material Images with Quality and Fidelity Guarantees, 26th International Meshing Roundtable, pages 40-52. Barcelona, Spain. September 2017.

- Jing Xu and Andrey Chernikov, Automatic curvilinear mesh generation with smooth boundary driven by guaranteed validity and fidelity, 23rd International Meshing Roundtable, pages 200-212. London, UK, October 2014.

- Jing Xu and Andrey Chernikov, Curvilinear triangular discretization of biomedical images with smooth boundaries, 11th International Symposium on Bioinformatics Research and Applications, pages 343-354. Norfolk, VA, July 2015.

- Andrey Chernikov and Jing Xu, A computer-assisted proof of correctness of a marching cubes algorithm, 22nd International Meshing Roundtable, pages 505-523. Orlando, FL, October 2013.

- Jing Xu and Andrey Chernikov, Tetrahedralization of multi-material images with quality and Hausdorff distance guarantees, 25th International Meshing Roundtable, Washington, D.C., USA, September 27-30, 2016. 5-page research note.

- Jing Xu and Andrey Chernikov, A sufficient condition of validity for cubic Bézier triangles, 24th International Meshing Roundtable, Austin, TX, October 2015. 5-page research note.

- Jing Xu and Andrey Chernikov, A guaranteed quality boundary graded triangular meshing algorithm backed by a computer-assisted proof, 22nd International Meshing Roundtable, Orlando, FL, October 2013. 5-page research note.

- Jing Xu and Andrey Chernikov, Fidelity and quality improvement of curvilinear image meshing on medical images, Modeling, Simulation, and Visualization Student Capstone Conference, pages 159-166. Suffolk, VA, April 2016.

- Jing Xu and Andrey Chernikov, Automatic curvilinear quality mesh generation with smooth mesh boundaries of medical images, Modeling, Simulation, and Visualization Student Capstone Conference, pages 205-212. Suffolk, VA, April 2015. Best paper and best presentation awards in the Medical Simulation track.

- Jing Xu and Andrey Chernikov, Quality meshing of 2D images with guarantees derived by a computer-assisted proof, Modeling, Simulation, and Visualization Student Capstone Conference, pages 119-126. Suffolk, VA, April 2014.

- Jing Xu and Andrey Chernikov, A parallel marching cubes algorithm for extracting isosurfaces from medical images, Modeling, Simulation, and Visualization Student Capstone Conference, pages 88-93. Suffolk, VA, April 2013.

BIBLIOGRAPHY

[1] O. C. Zienkiewicz, R. L. Taylor, J. Z. Zhu, The Finite Element Method: Its Basis and Fundamentals, 6th edition, Oxford: Butterworth-Heinemann, 2005.

[2] W. Zhang, D. Feng, R. Li, A. Chernikov, N. Chrisochoides, C. Osgood, C. Konikoff, S. Newfeld, S. Kumar, S. Ji, A mesh generation and machine learning framework for Drosophila gene expression pattern image analysis, BMC Bioinformatics 14:372.

[3] W. Zhang, R. Li, D. Feng, A. Chernikov, N. Chrisochoides, C. Osgood, S. Ji, Evolutionary soft co-clustering: formulations, algorithms, and applications, Data Mining and Knowledge Discovery (2014) 1–27.

[4] E. Frise, A. S. Hammonds, S. E. Celniker, Systematic image-driven analysis of the spatial drosophila embryonic expression landscape, Molecular systems biology 6 (1).

[5] M. Jagalur, C. Pal, E. Learned-Miller, R. T. Zoeller, D. Kulp, Analyzing in situ gene expression in the mouse brain with image registration, feature extraction and block clustering, BMC bioinformatics 8 (Suppl 10) (2007) S5.

[6] S.-W. Cheng, T. K. Dey, J. Shewchuk, Delaunay mesh generation, CRC Press, 2012.

[7] P.-L. George, H. Borouchaki, Delauney triangulation and meshing : application to finite elements, Hermes Science Publications, Paris, 1998.

[8] D. L. Marcum, F. Alauzet, Unstructured mesh generation using advancing layers and metric-based transition for viscous flowfields, in: 21st AIAA Computational Fluid Dynamics Conference, 2013.

[9] J. Schöberl, Netgen an advancing front 2d/3d-mesh generator based on abstract rules, Computing and Visualization in Science 1 (1) (1997) 41–52.

[10] Y. Ito, A. Shih, A. Erukala, B. Soni, A. Chernikov, N. Chrisochoides, K. Nakahashi, Parallel unstructured mesh generation by an advancing front method, Mathematics and Computers in Simulation 75 (2007) 200–209.

[11] D. J. Mavriplis, An advancing front delaunay triangulation algorithm designed for robustness, Journal of Computational Physics 117 (1) (1995) 90–101.

[12] W. E. Lorensen, H. E. Cline, Marching cubes: A high resolution 3d surface construction algorithm, COMPUTER GRAPHICS 21 (4) (1987) 163–169.

[13] S. A. Mitchell, S. A. Vavasis, Quality mesh generation in higher dimensions, SIAM Journal on Computing 29 (4) (2000) 1334–1370.

[14] F. Labelle, J. R. Shewchuk, Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles, ACM Transactions on Graphics 26 (3) (2007) 57.1–57.10, special issue on Proceedings of SIGGRAPH 2007.

[15] S. W. Walker, Tetrahedralization of isosurfaces with guaranteed-quality by edge rearrangement (tiger), SIAM Journal on Scientific Computing 35 (1) (2013) A294–A326.

[16] A. Chernikov, N. Chrisochoides, Multitissue tetrahedral image-to-mesh conversion with guaranteed quality and fidelity, SIAM Journal on Scientific Computing 33 (2011) 3491–3508.

[17] S.-W. Cheng, T. K. Dey, E. A. Ramos, T. Ray, Sampling and meshing a surface with guaranteed topology and geometry, SIAM Journal on Computing 37 (4) (2007) 1199–1227.

[18] J.-D. Boissonnat, D. Cohen-Steiner, G. Vegter, Isotopic implicit surface meshing, Discrete & Computational Geometry 39 (1) (2008) 138–157.

[19] N. Amenta, S. Choi, T. K. Dey, N. Leekha, A simple algorithm for homeomorphic surface reconstruction, in: Proceedings of the Sixteenth Annual Symposium on Computational Geometry, SCG '00, 2000, pp. 213–222.

[20] N. Amenta, T. J. Peters, A. C. Russell, Computational topology: ambient isotopic approximation of 2-manifolds, Theoretical Computer Science 305 (1) (2003) 3 – 15.

[21] S. Plantinga, G. Vegter, Isotopic meshing of implicit surfaces, The Visual Computer 23 (1) (2007) 45–58.

[22] A. Chernikov, P. Foteinos, Y. Liu, M. Audette, A. Enquobahrie, N. Chrisochoides, Tetrahedral image-to-mesh conversion approaches for surgery simulation and navigation, in: Y. J. Zhang (Ed.), Image-Based Geometric Modeling and Mesh Generation, Lecture Notes in Computational Vision and Biomechanics, Vol. 3, Springer, 2013, pp. 69–84.

[23] J. F. Thompson, B. K. Soni, N. P. Weatherill (Eds.), Handbook of grid generation, CRC Press, Boca Raton, London, New York, 1999.

[24] L. P. Chew, Guaranteed-quality delaunay meshing in 3d (short version), in: Proceedings of the Thirteenth Annual Symposium on Computational Geometry, SCG '97, 1997, pp. 391–393.

[25] L. Paul Chew, Constrained delaunay triangulations, Algorithmica 4 (1) (1989) 97–108.

[26] X.-Y. Li, S.-H. Teng, Generating well-shaped delaunay meshed in 3d, in: Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '01, 2001, pp. 28–37.

[27] S.-W. Cheng, T. K. Dey, H. Edelsbrunner, M. A. Facello, S.-H. Teng, Silver exudation, J. ACM 47 (5) (2000) 883–904.

[28] Conforming delaunay triangulations in 3d, Computational Geometry 28 (2) (2004) 217 – 233.

[29] S.-W. Cheng, T. K. Dey, Quality meshing with weighted delaunay refinement, SIAM Journal on Computing 33 (1) (2003) 69–93.

[30] H. Edelsbrunner, D. Guoy, An experimental study of sliver exudation, Engineering with Computers 18 (3) (2002) 229–240.

[31] S. Oudot, L. Rineau, M. Yvinec, Meshing Volumes Bounded by Smooth Surfaces, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 203–219.

[32] D. Boltcheva, M. Yvinec, J.-D. Boissonnat, Mesh Generation from 3D Multi-material Images, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 283–290.

[33] J. P. Pons, F. Ségonne, J. D. Boissonnat, L. Rineau, M. Yvinec, R. Keriven, High-Quality Consistent Meshing of Multi-label Datasets, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 198–210.

[34] B. M. Klingner, J. R. Shewchuk, Aggressive Tetrahedral Mesh Improvement, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 3–23.

[35] J.-D. Boissonnat, S. Y. Oudot, Provably good sampling and meshing of surfaces, Graphical Models 67 (2005) 405–451.

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

[37] P. Foteinos, A. Chernikov, N. Chrisochoides, Guaranteed quality tetrahedral Delaunay meshing for medical images, Computational Geometry Theory and Applications 47 (2014) 539–562.

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

[39] J. Bronson, J. A. Levine, R. Whitaker, Lattice cleaving: A multimaterial tetrahedral meshing algorithm with guarantees, IEEE Transactions on Visualization and Computer Graphics 20 (2) (2014) 223–237.

[40] X. Liang, Y. Zhang, An octree-based dual contouring method for triangular and tetrahedral mesh generation with guaranteed angle range, Engineering with Computers 30 (2) (2014) 211–222.

[41] W. J. Schroeder, J. A. Zarge, W. E. Lorensen, Decimation of triangle meshes, in: Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '92, ACM, New York, NY, USA, 1992, pp. 65–70.

[42] J. Yan, P. Shi, D. Zhang, Mesh simplification with hierarchical shape analysis and iterative edge contraction, IEEE Transactions on Visualization and Computer Graphics 10 (2) (2004) 142–151.

[43] S. Sherwin, J. Peiro, Mesh generation in curvilinear domains using high-order elements, Int. J. Numer 00 (2000) 1–6.

[44] P.-O. Persson, J. Peraire, Curved Mesh Generation and Mesh Refinement using Lagrangian Solid Mechanics, in: Proceedings of the 47th AIAA Aerospace Sciences Meeting and Exhibit, Orlando, FL, 2009.

[45] X. juan Luo, M. S. Shephard, R. M. O'Bara, R. Nastasia, M. W. Beall, Automatic p-version mesh generation for curved domains, Engineering with Computers 20 (2004) 273–285.

[46] P.L.George, H.Borouchaki, Construction of tetrahedral meshes of degree two, Int. J. Numer. Mesh. Engng 90 (2012) 1156–1182.

[47] C. R. Maurer, Jr., R. Qi, V. Raghavan, A linear time algorithm for computing exact euclidean distance transforms of binary images in arbitrary dimensions, IEEE Trans. Pattern Anal. Mach. Intell. 25 (2) (2003) 265–270.

[48] G. Farin, Curves and Surfaces for Computer-Aided Geometric Design, Academic Press, 1997.

[49] M. Jakab, R. Kikinis, Head and neck atlas.
URL `https://spl.harvard.edu/software-and-data-sets`

[50] R. J.A., J. M., K. R, Spl knee atlas (SPL 2015 Sep).
URL `https://spl.harvard.edu/software-and-data-sets`

[51] Ircad Laparoscopic Center, http://www.ircad.fr/softwares/3Dircadb (2013).

[52] J. Ahrens, B. Geveci, C. Law, Paraview: An end-user tool for large data visualization.

[53] CGAL, Computational Geometry Algorithms Library, http://www.cgal.org.

[54] P. Allen, Allen brain atlas, `http://www.brain-map.org` (2014).
URL `http://www.brain-map.org`

[55] Berkeley drosophila genome project, `http://www.fruitfly.org/` (2014).
URL `http://www.fruitfly.org/`

VITA

Jing Xu
Department of Computer Science
Old Dominion University
Norfolk, VA 23529

Jing Xu is a Ph.D student in the Department of Computer Science at Old Dominion University, USA. She received her B.S. degree in Network Engineering from Information Engineering University, China, and her M.S. degree in Computer Science from the Soochow University, China. Her research interests are medical and bio-material image analysis and quality mesh generation.