Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Engineering 00 (2017) 000–000

Procedia Engineering

www.elsevier.com/locate/procedia

26th International Meshing Roundtable, IMR26, 18-21 September 2017, Barcelona, Spain

Homeomorphic Tetrahedralization of Multi-material Images with Quality and Fidelity Guarantees

Jing Xu, Andrey N. Chernikov*

Old Dominion University, Department of Computer Science, 4700 Elkhorn Ave, Norfolk, VA, 23508, USA

Abstract

We present a novel algorithm for generating three-dimensional unstructured tetrahedral meshes of multi-material images. The algorithm produces meshes with high quality since it provides a guaranteed dihedral angle bound of up to 19.47° for the output tetrahedra. In addition, it allows for user-specified guaranteed bounds on the two-sided Hausdorff distance between the boundaries of the mesh and the boundaries of the materials. Moreover, the mesh boundary is proved to be homeomorphic to the object surface. The algorithm is fast and robust, it produces a sufficiently small number of mesh elements that comply with these guarantees, as compared to other software. The theory and effectiveness of our method are illustrated with the experimental evaluation on synthetic and real medical data.

Keywords: tetrahedralization; guaranteed quality; fidelity; multi-material; homeomorphism

1. Introduction

With medical data sets, one can generate conforming quality meshes of the spatially realistic domains that help producing 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, so it is usually segmented into multiple regions (materials). 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.

There are three conceptually different approaches to triangular and tetrahedral mesh generation: Delaunay refinement methods, advancing front methods, and adaptive space-tree methods. Delaunay refinement methods [5,8] 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. Advancing front methods [9,14,20] build the mesh in layers starting by dividing the boundaries of the mesh into edges (in two dimensions) or triangular faces (in three dimensions) and

*Corresponding author.
E-mail address: jxu@cs.odu.edu, achernik@cs.odu.edu

work toward the center of the region being meshed. Some of the advancing front methods [16] combine the Delaunay triangulation and advancing front ideas, however, no theoretical guarantees are usually available. Adaptive space-tree methods [6,11,17] build graded meshes from adaptive space subdivision (e.g., quadtree in two dimensions, standard octree or octree of body-centric cubic lattice in three dimensions). Sometimes theoretical guarantees on the quality in terms of dihedral angle are provided. This work falls into the third category.

The generation of geometric discretizations from segmented multi-material images presents a number of challenges. In particular, a mesh should meet constraints on both the shape and the size of its elements, and must conform at the material boundaries. In addition, the algorithm must handle arbitrary topology. In this paper we present an algorithm for constructing tetrahedral volume meshes that satisfy the following criteria:

1. Elements with arbitrarily small angles which cause the stiffness matrix in FE analysis to be ill-conditioned do not appear in the mesh. Specificly, we guarantee that all dihedral angles are above a user-specified lower bound which can be set to any value up to 19.47°.
2. The mesh offers a guaranteed bound on fidelity to 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.
3. The mesh is able to offer a faithful topology of the materials. In other words, mesh boundary is guaranteed to be homeomorphic to the object surface. We discuss the concept of homeomorphism, and give a sufficient condition for the approximation to offer homeomorphism.
4. The mesh contains a small number of elements that comply with the three guarantees above.
5. The implementation is as fast as other image-to-mesh conversion codes.

There has been a significant amount of work on guaranteed quality mesh construction.

Isosurface Stuffing [11] 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 paper is able to achieve a minimum dihedral angle bound of 19.47°.

Expanding on the ideas from Isosurface Stuffing [11], J.Bronson et al. designed a software called Cleaver [4] 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 X. and Zhang Y. [12] proposed an octree-based dual contouring algorithm with guaranteed mesh quality for closed smooth surface. The algorithm first generates an adaptive 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 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. [7] present a Delaunay meshing algorithm with several mathematical guarantees. They proved that the tetrahedra in the output mesh have radius-edge ratio less than 1.93. The two-sided Hausdorff distance between the object surface and mesh boundary is bounded by a user-specified parameter. Using a strategy called *ϵ-Sample* [2,3], the mesh boundary is proved to be ambient isotopic to the object surface. However, the method only supplies the circumradius-to-shortest-edge ratio bound. Even if this ratio is very low, the absence of almost flat tetrahedra called slivers is not guaranteed.

To guarantee a faithful topology of the materials, the generation of a watertight surface mesh that is homeomorphic to the object surface is our initial step. A number of previous approaches deal with the extraction of isosurfaces from scalar fields or implicit surfaces. SnapMC [19] is an isosurface extraction algorithm using an extended Marching Cubes lookup table and a warping technique. The lookup table has $3^8 = 6561$ entries, based on three possible cases of each cube vertex: with its scalar value greater than the isovalue, less than the isovalue and exactly equal to the isovalue. However, generating a lookup table with 6561 cases is labor intensive and error prone. Instead, our lookup table generates cutting edges on each of the cube face, so it has only $3^4 = 81$ entries totally, therefore avoided the unnecessary complexity. Moreover, the warping technique adopted by SnapMC can change isosurface topology, such as eliminating small components and merging vertices. In contrast, our method always maintains the original topology.

Plantinga S. and Vegter G. [18] presented an algorithm that generates an isotopic piecewise linear approximation of implicit curves and surfaces. The guarantee of topological correctness is achieved by splitting the octree cells such that the cells are parameterizable using interval arithmetic. Unlike our algorithm, the cases where the surface passes through vertices of the grid were handled by considering the function value to be strictly positive. However, the mesh quality and geometric accuracy were not considered in this method. Our algorithm allows for any user-specified bounds on two-sided Hausdorff distance, and the mesh quality is also guaranteed.

Kazhdan M. et al. [10] designed an algorithm for generating a watertight isosurface from an octree without extra refinement of the octree nodes or modification of their values. The algorithm uses a set of binary edge-tree derived from the octree's topology to generate iso-edges on node faces such that the iso-edges of a node are closed loops. Using the minimal area triangulation, the method triangulates iso-polygons of octree nodes without self-intersections. The method targets at correctness and computational efficiency as well as memory overhead, mesh quality and representation accuracy are not considered. Our algorithm targets at a good quality volume mesh with faithful topology and geometric accuracy, so we made a trade off between number of octree nodes and a good volume mesh.

This work builds upon the Lattice Decimation (LD) method [6]. 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 splits the octree 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, so the authors designed a post-processing decimation step using vertex removal operation [21,22] while at all times maintaining the required fidelity and quality bounds. However, the decimation step is a greedy algorithm which was not designed for smooth transition in element size. In fact, it can produce clusters of small elements surrounded by much larger elements. In this work, we refine the octree to a lower level so that we avoid tiny elements, therefore, the issue can be mitigated. Moreover, although LD maintains the material connectivity, it does not guarantee topological fidelity. We designed a new technique based on the enforcement of a single manifold condition that solves this problem.

In the presented algorithm, we approximate the boundary of the materials with a set of triangular patches in octree leaves using a pre-defined intersection edge look-up table. The triangular patches all together form a waterproof surface mesh which is proven to be homeomorphic to the boundaries of the materials. We achieve the two-sided Hausdorff distance bound by constructing a sequence of such surface meshes until the fidelity condition in each leaf is satisfied. The octree leaves are filled with high-quality elements from a pre-defined triangle look-up table. 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. Although the proposed approach requires more computational time to construct the initial coarse mesh, 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), since the initial coarse mesh saves computational time during decimation.

2. Definitions and preliminaries

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.

A quadtree (octree) subdivision is called satisfying the *2-to-1 rule* if any two neighboring squares (cubes) differ at most by a factor of two in size.

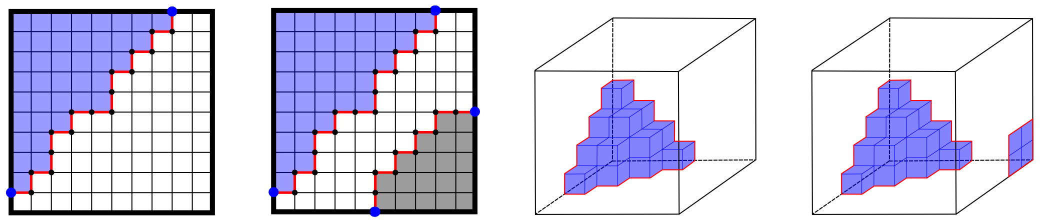
A mapping $\mu : X \rightarrow Y$ defines a *homeomorphism* between two compact Euclidean subspaces X and Y if μ is continuous, one-to-one and onto. The inverse function μ^{-1} is also continuous.

A *manifold* is a topological space that is locally Euclidean. Each point of an n-dimensional manifold has a neighborhood that is homeomorphic to the open unit ball in $\mathcal{R}^n$. A *manifold with boundary* is a manifold with an edge. The boundary of an n-manifold with boundary is an $(n-1)$-manifold.

Before we define the single manifold condition, we define the boundary leaf, image boundary, the image boundary edge, the image boundary face and the image boundary vertex first. We call a leaf of an octree or a quadtree a *proper boundary leaf* if it contains more than one material; a *boundary leaf* is either a proper boundary leaf, or 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. *Image boundary* is the union of voxel faces that separate voxels of different colors. It includes the interior boundary and exterior boundary. The exterior boundary separates the materials from the background, while the interior boundary separates different materials. An *image boundary edge* is an edge of a voxel upon which a voxel of a different color is incident. This definition applies for both two- and three-dimensional cases. In the three-dimensional case, an *image boundary face* is a face of a voxel upon which a voxel of a different color is incident. The end points of image boundary edges and the corner points of image boundary faces are called *image boundary vertices*. The single manifold condition below applies to the boundary leaves. The construction of the waterproof surface mesh which is homeomorphic to the boundary of the materials relies on this condition.

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

Specifically, in the two dimensional case, the image boundary edges form a 1-manifold with boundary, a chain composed by image boundary edges. On this chain, every image boundary vertex has two neighbors, except the first and the last ones, which have only one neighbor. The closed cycle is not included, because every image boundary vertex on the cycle has two neighbors. Figure 1a is an illustration of the two-dimensional single manifold condition on a quadtree leaf. It shows a boundary quadtree leaf that contains two materials (white and blue). It respects the single manifold condition, because there is only one chain such that the two image boundary vertices at the end of the chain have only one neighbor and all the other image boundary vertices have two neighbors. Figure 1b shows a boundary quadtree leaf that contains three materials (white, blue and gray). It violates the single manifold condition, because there are two chains with four image boundary vertices at the end of the chains that have only one neighbor.


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

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

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

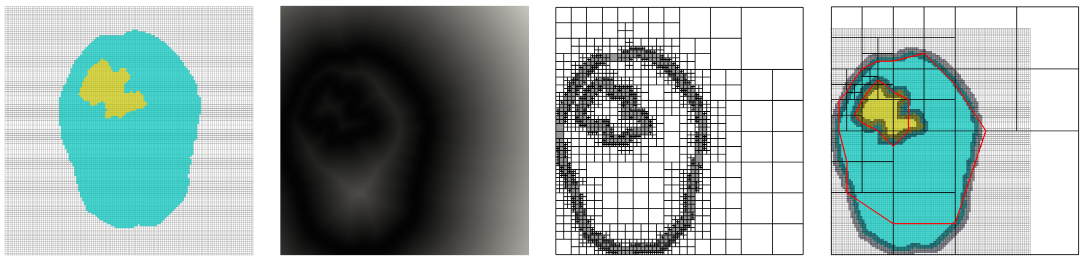
(d) A boundary octree leaf violates the single manifold condition

Fig. 1: An illustration of the two- and three-dimensional single manifold condition on a quadtree/octree leaf. In (a) and (b), 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. In (c) and (d), 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 degree two image boundary edges, and the red edges show the degree one image boundary edges.

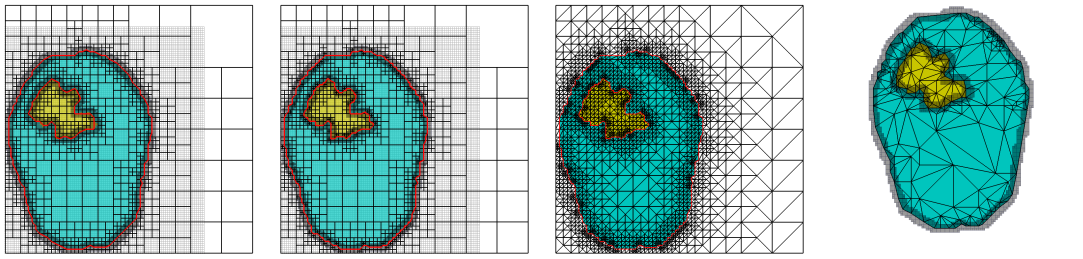
In the three dimensional case, the *degree of the image boundary edge* is defined by the number of boundary faces that are incident upon the edge. In a boundary octree leaf, the image boundary faces form a 2-manifold with boundary, a sheet on which the degree one image boundary edges form a cycle and all the other image boundary edges are degree two edges. A closed surface is not included, because every image boundary edge on the closed surface is a degree two edge. Figure 1c is an illustration of the three-dimensional single manifold condition on an octree leaf. It shows a boundary octree leaf which respects the single manifold condition. In this leaf, the image boundary faces form a sheet on which the degree one image boundary edges form a cycle and all the other image boundary edges are degree two edges. Figure 1d shows a boundary octree leaf which violates the single manifold condition. In this leaf, the image boundary faces form two sheets with two cycles that are composed of degree one image boundary edges.

3. 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 19.47°. The algorithm outputs a quality mesh which satisfies these bounds.



(a) The input two-dimensional image of size 128×128 (b) Euclidean distance transform (c) The leaves of the quadtree that are within the fidelity bound are marked (d) The first iteration of constructing the surface mesh



(e) The second iteration of constructing the surface mesh (f) The final iteration of constructing the surface mesh (g) The original fine mesh, 2678 triangles inside the object, 4408 triangles total (h) The decimated mesh, 378 triangles inside the object, the outside triangles are removed

Fig. 2: An illustration of the main steps performed by our algorithm.

The main steps performed by our algorithm are illustrated in Figure 2. For explanation purposes, in Figure 2a we show a bounded domain $\Omega \subset \mathcal{R}^2$ represented by a two-dimensional image. The size of the image is 128×128 pixels. It defines a slice of a simplified human brain (shown in cyan) with a tumor (shown in yellow) against a white background. The input dihedral angle bound is set to 19.47°, and the fidelity bounds are both set to two voxels. Figure 2b shows the Euclidean distance transform of the input image. Figure 2c shows the fidelity zone, i.e. the leaves of the quadtree that are within the fidelity bound. In Figure 2d, we show the quadtree with leaves refined to meet the single manifold condition and the 2-to-1 rule. The surface mesh was extracted from a pre-defined intersection edge look-up table (shown in section 3.1) and displayed by the red segments. The shaded region shows the fidelity zone. Since the surface mesh does not meet the fidelity condition, the octree needs to be split in the next iteration. Figure 2e shows the leaves whose two-sided Hausdorff distance of either side is larger than the fidelity bound in Figure 2d having been split into four children and then the quadtree having been refined once more to meet the single manifold condition and the 2-to-1 rule. The red segments show the surface mesh, which still does not satisfy the fidelity condition. In Figure 2f, the quadtree leaves were refined to meet the single manifold condition and the 2-to-1 rule, and the surface mesh respects the fidelity condition. The red segments show the surface mesh. It is the final mesh boundary. Figure 2g shows the fine mesh constructed from a pre-defined two-dimensional triangle look-up table (shown in section 3.2). Figure 2h shows the decimated mesh which maintains the quality, the topology of the underlying object and the fidelity bound. We elaborate each step in the following sections.

3.1. *Construction of the octree*

The algorithm first constructs an octree that completely encloses all the materials (except for the background voxels) from the bitmap. The boundaries between the octree leaves correspond exactly to the boundaries between the voxels. Besides that, 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. Initially, the algorithm splits the octree until the leaves satisfy the single manifold condition, and the sizes of the leaves respect the 2-to-1 ratio. Then it generates a waterproof surface mesh which is homeomorphic to the the boundaries of the materials from a pre-defined intersection edge look-up table. It computes the two-sided Hausdorff distance between the boundaries of the mesh and the boundaries of the materials in each leaf. It splits the leaf into 8 children if the Hausdorff distance of either side is larger than the user specified fidelity bounds. If at least one of the leaves was split due to a violation of a fidelity condition, the current surface mesh is discarded. The algorithm iteratively checks the single manifold condition and the 2-to-1 rule, and generates waterproof surface meshes until the two-sided Hausdorff distance respects the user specified fidelity bounds.

3.1.1. *Waterproof surface mesh*

To generate the waterproof surface mesh, we use an approach reminiscent of the Marching Cubes algorithm [13]. There are three main differences between the classic MC and our algorithm: first, MC processes on a rectilinear grid with uniform size, while our algorithm processes on a balanced octree. Second, MC evaluates the boundary distance function f at the vertices of each cube to two values, positive or negative. In contrast, our algorithm evaluates the vertices of a node to three values: positive, negative and zero. For vertices lying inside of materials, a vertex could be inside of one material but be outside of another. For the ease of assigning values, after we construct the octree root, we presplit the octree until each leaf contains at most two materials. A vertex of a leaf evaluates to positive if the vertex is located in the material whose label is larger between the two materials, to negative if the vertex is located in the material whose label is smaller and to zero if the vertex is located exactly on the boundary between the materials. Third, MC approximates the intersection of the isosurface with that cube using triangles from a look-up table defined based on a cube. However, for our algorithm, the templates on cubes would be cumbersome, because there would be 3^8 cases to be analyzed in the uniform background grid case or even much more for the graded case. Instead, we designed an intersection edge look-up table on squares for each cube face, so there are totally only 3^4 entries in the table. For each of the proper boundary leaves, the intersection edge look-up table generates intersection edges on the cube faces, and the triangular patches are generated by connecting those intersection edges with the center of the cube.

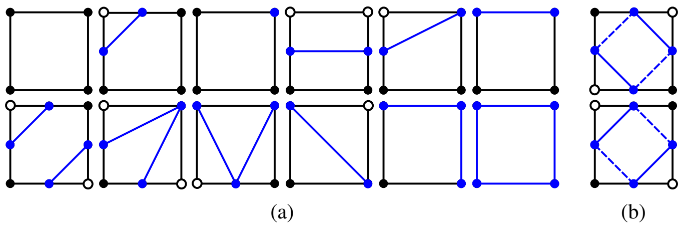
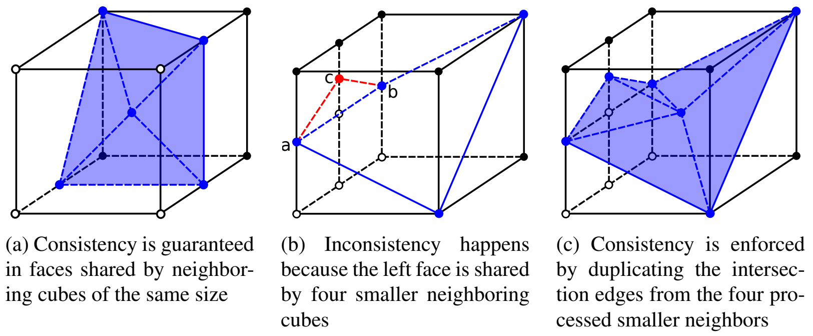


Fig. 3: Black circles show the positive vertices, white circles show the negative vertices, and blue circles show the zero vertices. Blue segments show the intersection edges created by the algorithm. (a) All possible stencils for creating intersection vertices and intersection edges by grouping cases with the opposite relations to the vertex value in all corners into one case and also grouping rotationally symmetric cases. (b) Solid edges correspond to the set of rules analyzed in this paper, while dashed edges show another feasible set of rules.

The idea of designing the table is to connect all the zero valued vertices. Those zero valued vertices include the vertices of the cube located exactly on the boundary of the materials, and the vertices that are generated by calculating the central point along an edge of the cube whose two ends have the opposite values. We list all possible stencils in Figure 3a by grouping cases with the opposite relations to the vertex value in all corners into one case and also grouping rotationally symmetric cases. Figure 3b shows the rules for the case of the second row first column in Figure 3a. Solid edges correspond to one set of rules, while dashed edges show another feasible set of rules. If the two different set of rules are used on a common face shared by two cubes, the sets of triangles created in both cubes form a hole in the surface at the shared face. To avoid this ambiguity, the set of rules the solid edges that correspond to is always used in this paper. For the boundary leaves that contain only one material, the intersection edge look-up table is not used. We generate triangular patches that are same as the leaf face if every voxel that is incident upon this face is adjacent to a voxel of a different color.

Figure 4a shows an octree leaf whose faces are shared by neighboring cubes of the same size. The triangular patches are generated by the intersection edge look-up table. In the case of a face of the octree leaf being shared by four smaller neighboring cubes, the consistency is not always guaranteed. For example, in Figure 4b we show a leaf whose left face is shared by four smaller neighboring cubes. From the leaf side of the view, the intersection edge generated for the left face of the leaf should be edge $\overline{ab}$ (shown as a blue dashed segment). However, from the four smaller neighbors' side of the view, the intersection edges generated for the same face should be edge $\overline{ac}$ and edge $\overline{cb}$ (shown in red dashed segments). The inconsistency in the shared face of neighboring leaves leads to a hole in the surface mesh.

As Kazhdan M. et al. [10] pointed out, 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 in the mesh must be shared by exactly two polygons. We process the octree leaves in the order of their size, starting with the smallest. We duplicate the intersection edges from the processed neighbors (including the neighbors of same size and of smaller size), and generate new intersection edges for the faces that are shared by the neighboring cubes which have not been processed. Then we check if the intersection edges (unique) from the six cube faces form a cycle. If all the intersection edges do not form a cycle (including the case when the set of intersection edges is empty), the triangulation on this leaf needs to be discarded and the leaf needs to be split. We use the fan-like triangular patches to approximate the image boundary in each proper boundary leaf. In Figure 4c, the consistency is considered by duplicating the intersection edges from the four processed smaller neighbors. Since the intersection edges from the six cube faces form a cycle, also the duplication of the intersection edges make them shared twice, the surface mesh of our algorithm is guaranteed to be watertight.



(a) Consistency is guaranteed in faces shared by neighboring cubes of the same size

(b) Inconsistency happens because the left face is shared by four smaller neighboring cubes

(c) Consistency is enforced by duplicating the intersection edges from the four processed smaller neighbors

Fig. 4: An illustration of generating triangular patches to approximate the image boundary in a leaf. Blue circles show the zero vertices, blue edges on cube faces show the intersection edges and blue triangles show the triangular patches.

3.1.2. *Two-sided Hausdorff distance*

The mesh has to provide a close 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. For image boundary I and mesh boundary M, The two-sided distance is:

$$H(I, M) = \max\{h(I, M), h(M, I)\},$$

where

$$h(I, M) = \max_{i \in I} \min_{m \in M} d(i, m), \qquad h(M, I) = \max_{m \in M} \min_{i \in I} d(m, i).$$

The zero two-sided Hausdorff distance bound means that the boundaries of each extracted material in the tetrahedral mesh is aligned with the voxel boundaries. To measure the Hausdorff distance from the surface mesh to the boundaries of the image, we construct the second octree (in the implementation, we make the two octrees one with common ancestor for efficiency and storage purpose). We compute the Euclidean distance transform [15] (EDT) of the extended image (same size as the octree), and split the octree until the distances of EDT 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. We mark the leaves that are within the input fidelity tolerance. If one of the triangular facets of the surface mesh in the leaf intersects at least one of the leaves marked as outside the fidelity tolerance, the fidelity condition is violated and the leaf is split. To measure the other side, we compute the shortest distance 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, we assign each of the image boundary vertices to its closest triangular patch for later use in the mesh decimation step.

3.2. Filling in the octree

We fill the octree leaves with high quality tetrahedra using a pre-defined triangle look-up table. The template triangles of the triangle look-up table should respect the intersection edges of the intersection edge look-up table, which means that the intersection edges should be the edges of the template triangles. We list the stencils that correspond to the stencils of the intersection edge look-up table in Figure 5.

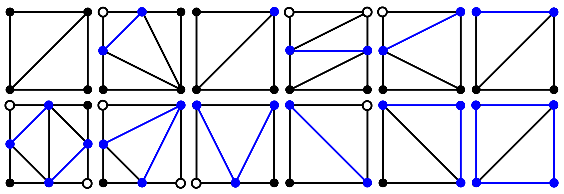


Fig. 5: The stencils of the triangle look-up table that correspond to the stencils of the intersection edge look-up table for creating triangles on cube faces. The blue segments show the intersection edges. The template triangles respect the intersection edges.

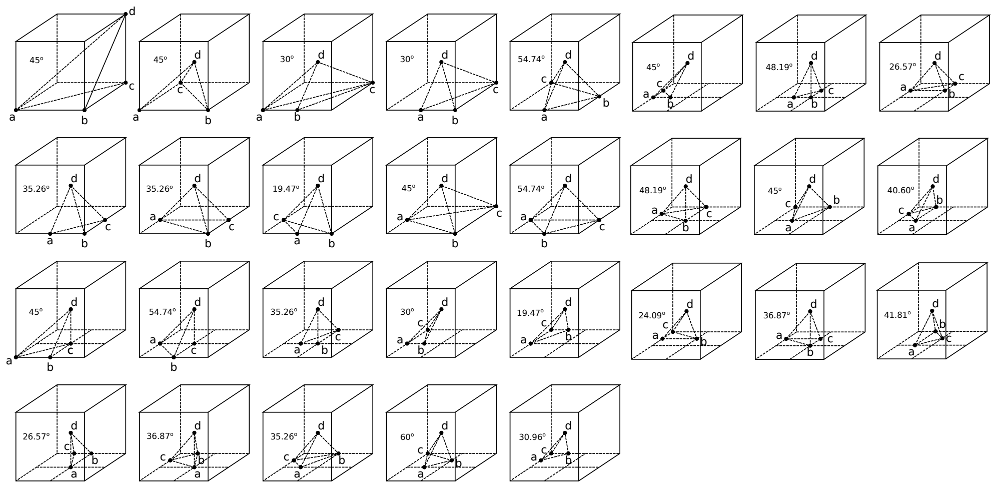


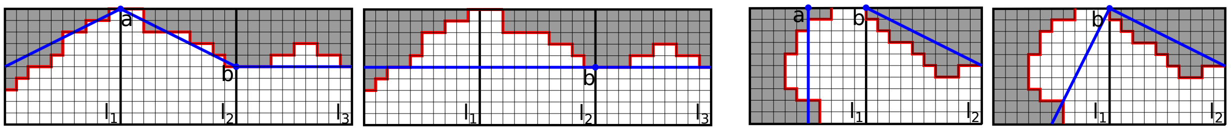
Fig. 6: All possible shapes of the initial tetrahedra (abcd) filling in a cubic leaf of the octree, up to symmetry. The smallest dihedral angles are listed on the figure. Therefore, 19.47° is the lower bound for the dihedral angle in the initial three-dimensional tetrahedralization of the octree.

Theorem 3.1. *All dihedral angles are above a user-specified lower bound which can be set to any value up to* 19.47°.

Proof. The key to prove theorem 1 is to prove that the bound on the dihedral angles before mesh decimation is 19.47°. Since our mesh vertices have finite number of locations in the octree leaf (either on the corners or on the quarters of the leaf edges, except the center of the cube), we obtained a minimum dihedral angle of 19.47° by analyzing all possible resulting leaf triangulations; see Figure 6. So the user-specified lower bound can be set to any value up to 19.47°. □

3.3. Mesh decimation

Similar to the LD algorithm [6], the vertex removal operation is used to coarsen the mesh. The detailed operation can be consulted in the paper [6], here we only discuss the merging conditions. A vertex can not 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 facets has at least one-sided Hausdorff distance larger than the input fidelity bound, the merge is discarded. 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 triangular facets 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 triangular facet we maintain a cumulative list of the image boundary vertices. The image boundary vertices were first assigned to the triangles of the surface mesh during the fidelity check of the octree construction. Each image boundary vertex was assigned to the closest triangle. After each merge, the image boundary vertices of the boundary triangular patches that are incident upon the merged vertex are reassigned to the closest newly created boundary triangular patches. 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 triangular patches than the fidelity tolerance, the merge is discarded. 3. The topological equivalence requirement,



(a) a can merge to b (b) a can not merge to b

Fig. 7: An illustration of the topological equivalence requirement during decimation. For each figure, the left one shows before the merge and the right one shows after the merge. The red segments represent the image boundaries, and the blue segments represent the mesh boundaries. a is the merged vertex, b is its destination.

i.e., if the homeomorphism is not maintained during an operation of merge, the merge is discarded. We apply the following rules to maintain the original structure of the inter-material boundaries: (1) Boundary vertices only merge to boundary vertices. (2) For each boundary vertex we also maintain a cumulative list of the octree leaves it belongs to. The merge happens if the union of the set of octree leaves of the merged vertex and the set of the octree leaves of the destination respect the single manifold condition, i.e., the image boundary edges (or faces) of the union form only one $(n-1)$-manifold with boundary. Specifically, suppose boundary vertex a merges to boundary vertex b. L_a denotes the set of octree leaves that contains a, and L_b denotes the set of octree leaves that contains b. Notice that $L_a \cap L_b$ may not be empty. If $L_a \cup L_b$ satisfy the single manifold condition, the merge operation can be executed. We show two examples in Figure 7 for an illustration. In Figure 7a, $L_a = \{l_1, l_2\}$, $L_b = \{l_2, l_3\}$, $L_a \cup L_b = \{l_1, l_2, l_3\}$. Because the image boundary edges in $L_a \cup L_b$ form a manifold with boundary, a can merge to b. In Figure 7b, $L_a = \{l_1\}$, $L_b = \{l_1, l_2\}$, $L_a \cup L_b = \{l_1, l_2\}$. Because the image boundary edges in $L_a \cup L_b$ form two manifolds with boundary, a cannot merge to b.

Theorem 3.2. *The boundary of the mesh produced by the proposed algorithm is homeomorphic to the boundary of the input image.*

Proof. We prove the theorem by induction. Basis step: There is a homeomorphism between the image boundary and the surface mesh before decimation. Inductive step: For the inductive hypothesis we assume that after the k-th decimating operation, the boundary of the mesh is homeomorphic to the boundary of the image. Under this assumption, we show that after the $(k+1)$-th decimating operation, the boundary of the mesh is homeomorphic to the boundary of the image.

We prove the basis step. Due to the single manifold condition, the intersection of the image boundary with each of the boundary leaves forms an $(n-1)$-manifold with boundary, n being the dimension. We approximate the $(n-1)$-manifold with boundary in the leaf using the fan-like triangular patches. The fan-like triangular patches also form a

$(n-1)$-manifold with boundary by the way we construct it. So there is a continuous deformation between the image boundary and the triangular patches in the leaf. Thus, there exists a continuous mapping between the two manifolds with boundary such that the mapping is a bijection, continuous, and the inverse function of the mapping is continuous. Therefore, in every leaf, there is a homeomorphism between the boundary of the mesh and the boundary of the image. Then we have a global homeomorphism between the boundary of the mesh and the boundary of the image.

We prove the inductive step. Assume that after the k-th decimating operation, the boundary of the mesh is homeomorphic to the boundary of the image. This step also depends on the single manifold condition, but instead of checking on only one leaf, the condition is verified on several leaves. So if the image boundary edges in these leaves satisfy the single manifold condition, there is a continuous mapping between the image boundary and mesh boundary. Thus, after the $(k+1)$-th decimating operation, the boundary of the mesh is homeomorphic to the boundary of the image. This completes the inductive step. □

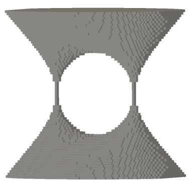
4. Experimental results

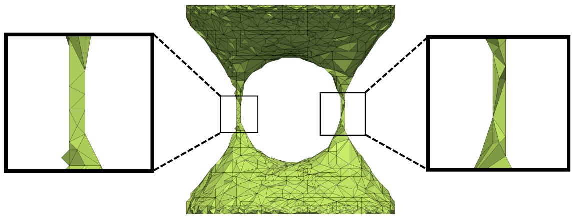
We evaluated the proposed Homeomorphic Octree Refinement and Decimation algorithm (HORD) on both synthetic and real medical data. All the experiments were conducted on a 64 bit machine equipped with two 3.06 GHz 6-Core Intel Xeon CPUs and 64 GB main memory. The algorithm was implemented in C++. We used integer arithmetic, avoided expensive mathematical functions such as trigonometric, square root, and floating point division, and customized memory allocations to improve the performance. Besides, we minimized computation through out the whole implementation, for instance, each leaf was checked only once for the simple manifold condition, no matter how many iterations it has been through; only the leaves whose neighbors have split need to be checked the second time in the octree balancing after the first iteration. Memory overhead also was given extra concern: first, we avoided redundant storage. For example, we implemented the two octrees one with common ancestor. Second, we stored pointers instead of objects. For example, each boundary vertex maintained only a list of pointers of octree leaves that it belongs to. Third, we freed memory as soon as possible. For example, after each iteration, we deleted points and surface triangles if they were of no use, and deleted the distance transform right after we constructed the fidelity zone. The simple manifold condition does not induce extra refinement of the octree, because in most of the cases one needs to fulfill other conditions such as small Hausdorff distance bounds. Nevertheless, our algorithm can run on a desktop even with a very large three-dimensional image. Our algorithm does not deal with noisy data, because our simple manifold condition maintains every manifold in the original topology. For the 3D visualization of the final meshes, we used ParaView [1], 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.

Figure 8 shows the output meshes of our implementation and the LD algorithm on *Cassini* of a fixed size of 100 × 100 × 100 voxels. Each voxel has side lengths of 1 unit in x, y, and z directions. We set the symmetric Hausdorff distance bound to 2 voxels and the minimum dihedral angle bound to 19.47°. It is clear that the presented HORD algorithm maintains better topology than LD.

The size of the *knee* atlas is 512 × 512 × 119 voxels and the size of the *abdomen* atlas is 512 × 512 × 219 voxels. In *knee* each voxel has side lengths of 0.27, 0.27, and 1 units in x, y, and z directions. In *abdomen* each voxel has side lengths of 0.96, 0.96, and 2.4 units in x, y, and z directions. Before meshing the atlases, we resampled them with voxels of equal side length corresponding to the smallest units so that we can obtain equally spaced images, the final meshes produced on them and their corresponding cut views. A histogram of all dihedral angles is shown for each example.

In Table 1 we show the comparison of the output mesh size and the total run time of the HORD and LD for the *knee* atlas and the *abdomen* atlas. We fixed the two-sided Hausdorff distance bound parameters, and vary the dihedral angle bound using values of 10°, 15°, and 19.47°. As we can see from Table 1, the output mesh size is low when $\bar{H}(I, M)$ is high, and the final mesh sizes decrease as the dihedral angle bounds decrease. When $\bar{H}(I, M) = 0$ and $\bar{H}(I, M) = 1$, the sizes of HORD meshes are slightly smaller than the sizes of LD meshes. However, when $\bar{H}(I, M) = 2$, the HORD algorithm generated a significantly smaller number of tetrahedra compared to the number of tetrahedra generated by the LD algorithm. We conclude that the HORD algorithm performs better in terms of the number of tetrahedra than LD algorithm even though it maintains the topology. When $\bar{H}(I, M) = 0$ and $\bar{H}(I, M) = 1$, the processing time of the


(a) Original image


(b) Final mesh of HORD

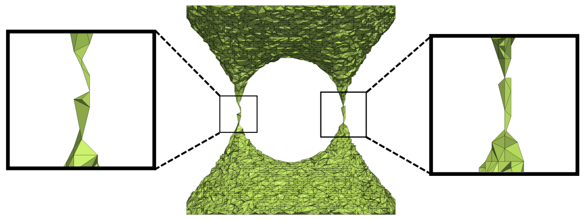
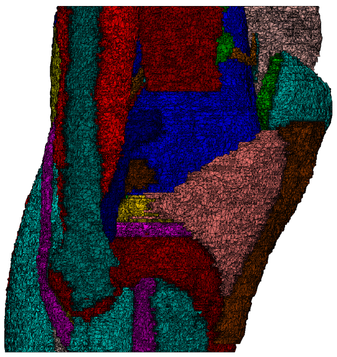
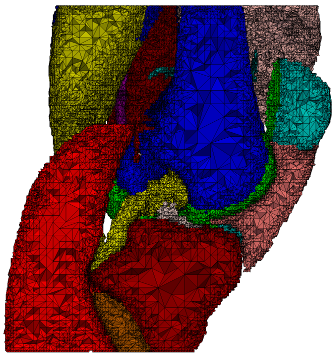
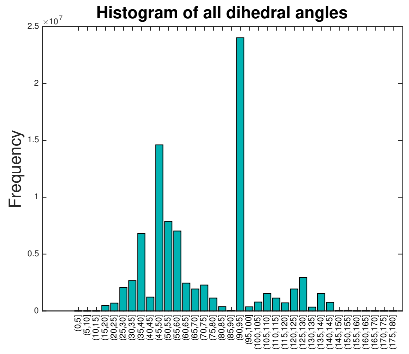

(c) Final mesh of LD

Fig. 8: The comparison of the HORD mesh and the LD mesh on *Cassini* for $\bar{h}(I, M) = 2$, $\bar{h}(M, I) = 2$ and $\bar{\theta} = 19.47°$.


(a) Final mesh


(b) A cut view of the final mesh

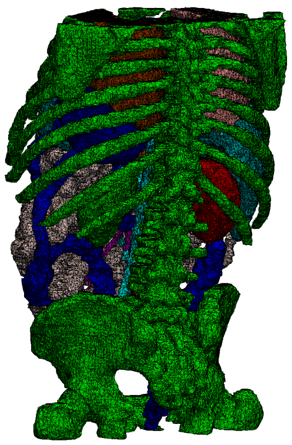


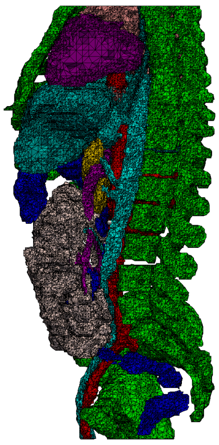
(c) A histogram of all dihedral angles. Empty bars show zero frequency

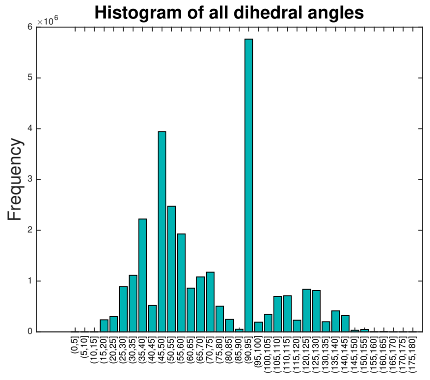
Fig. 9: The HORD mesh and its cut view on *knee* for $\bar{h}(I, M) = 2$, $\bar{h}(M, I) = 2$ and $\bar{\theta} = 19.47°$.

HORD algorithm is 10% to 20% slower than the LD algorithm, however, when $\bar{H}(I, M) = 2$, in most of the cases the HORD algorithm is slightly faster than the LD algorithm.

We also conducted experiments using open source mesh generator Computational Geometry Algorithms Library (CGAL) and compared the performance with the performance of our HORD algorithm. Table 2 presents the experimental evaluation of the I2M conversion functionality *make_mesh_3* offered by CGAL. We varied the value of parameter *facet_distance*, and computed the two-sided Hausdorff distance $h(M, I)$ and $h(I, M)$ from the final meshes generated by CGAL. $h(M, I)$ and $h(I, M)$ are measured by resampled voxel unit as we did in our algorithm so that the two algorithms has the same measure of Hausdorff distances. We also list the final number of tetrahedra, the minimum dihedral angle (measured in degrees) and the total running time (measured in seconds) by CGAL. From this table, there is no obvious relationship between *facet_distance* and $h(M, I)$. Further more, 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 proposed HORD algorithm can always guarantee that the Hausdorff distance bound is two-sided. CGAL improves mesh properties such as the dihedral angles using a combination of optimization algorithms (with parameters *lloyd*() and *odt*()), however, we could only obtain the best minimum dihedral angles 2.52°. In contrast, our best minimum dihedral angle bound is 19.47°.


(a) Final mesh


(b) A cut view of the final mesh



(c) A histogram of all dihedral angles. Empty bars show zero frequency

Fig. 10: The HORD mesh and its cut view on *abdomen* for $\bar{h}(I, M) = 2$, $\bar{h}(M, I) = 2$ and $\bar{\theta} = 19.47°$.

Table 1: The comparison of the final number of tetrahedra for HORD and LD

Hausdorff distance	$\bar{h}(I,M) = 0, \bar{h}(M,I) = 0$							
Input	*knee atlas*				*abdominal atlas*			
Algorithm	HORD		LD		HORD		LD	
Performance	# of tets	time(s)	# of tets	time(s)	# of tets	time(s)	# of tets	time(s)
After decimation with $\bar{\theta} = 19.47°$	14,641,738	473.26	20,030,904	400.13	13,495,336	824.71	18,163,216	698.93
After decimation with $\bar{\theta} = 15.00°$	11,617,559	478.50	13,844,888	378.72	10,796,997	821.24	12,656,733	682.76
After decimation with $\bar{\theta} = 10.00°$	10,553,530	469.40	12,412,983	376.39	9,839,441	819.88	11,329,731	692.24
Hausdorff distance	$\bar{h}(I,M) = 1, \bar{h}(M,I) = 1$							
Input	*knee atlas*				*abdominal atlas*			
Algorithm	HORD		LD		HORD		LD	
Performance	# of tets	time(s)	# of tets	time(s)	# of tets	time(s)	# of tets	time(s)
After decimation with $\bar{\theta} = 19.47°$	14,054,944	525.18	19,032,469	409.37	13,211,305	865.54	17,608,762	709.79
After decimation with $\bar{\theta} = 15.00°$	11,238,719	545.28	12,588,781	399.01	10,652,612	876.93	11,994,099	698.89
After decimation with $\bar{\theta} = 10.00°$	10,306,800	555.46	11,160,507	400.12	9,784,550	886.57	10,646,685	699.83
Hausdorff distance	$\bar{h}(I,M) = 2, \bar{h}(M,I) = 2$							
Input	*knee atlas*				*abdominal atlas*			
Algorithm	HORD		LD		HORD		LD	
Performance	# of tets	time(s)	# of tets	time(s)	# of tets	time(s)	# of tets	time(s)
After decimation with $\bar{\theta} = 19.47°$	6,549,841	384.55	14,198,097	442.37	5,509,483	678.69	12,434,028	747.85
After decimation with $\bar{\theta} = 15.00°$	2,100,759	390.65	3,837,660	399.02	1,593,965	689.39	3,426,170	700.86
After decimation with $\bar{\theta} = 10.00°$	1,136,779	395.92	2,335,844	378.64	856,455	688.93	2,081,046	691.58

Table 2: The final number of tetrahedra and run time of CGAL on *knee* and *abdomen*

Opt.		*no_lloyd(), no_odt(), perturb(), exude()*					*lloyd(), odt(), perturb(), exude()*				
knee											
Algorithm	*facet_dist.*	$h(M,I)$	$h(I,M)$	θ	# of tets	Total time	$h(M,I)$	$h(I,M)$	θ	# of tets	Total time
CGAL	0.1	3	29	1.50	3,077,237	139.47	7	29	1.01	3,023,807	1252.05
CGAL	0.15	3	29	2.52	1,320,892	57.60	3	29	0.45	1,309,545	504.63
CGAL	0.2	3	30	2.52	732,553	29.33	5	30	1.96	729,632	264.77
abdomen											
Algorithm	*facet_dist.*	$h(M,I)$	$h(I,M)$	θ	# of tets	Total time	$h(M,I)$	$h(I,M)$	θ	# of tets	Total time
CGAL	0.2	3	23	1.49	10,349,057	532.06	5	18	2.01	9,985,320	5079.16
CGAL	0.3	2	23	1.50	3,797,349	180.81	4	18	1.16	3,707,828	1587.65
CGAL	0.4	3	23	2.26	2,042,684	95.80	5	18	2.17	2,002,825	800.18

5. Conclusion

We presented a novel approach for automatic construction of two- and three-dimensional unstructured 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 small number of mesh elements. Our algorithm creates coarse meshes with relatively few large elements. The applications of the algorithm include mechanical modeling for finite element simulation, computational medicine and computational biology such as medical imaging, image registration, surgical simulation, and image-guided intervention. Despite multiple performance optimizations in the presented implementation, there is still space to improve the run time. There is also some space to improve the size of the mesh. For example, the topological requirement in the decimation is only a sufficient condition. Our future work is to further improve the performance in terms of the runtime and element size.

6. Acknowledgments

This work was supported (in part) by the Modeling and Simulation Graduate Research Fellowship Program at the Old Dominion University and NSF grant CCF-1439079.

References

[1] James Ahrens, Berk Geveci, and Charles Law. Paraview: An end-user tool for large data visualization. 2005.

[2] N. Amenta, S. Choi, T. K. Dey, and N. Leekha. A simple algorithm for homeomorphic surface reconstruction. In *Proceedings of the Sixteenth Annual Symposium on Computational Geometry*, SCG '00, pages 213–222, 2000.

[3] Nina Amenta, Thomas J. Peters, and Alexander C. Russell. Computational topology: ambient isotopic approximation of 2-manifolds. *Theoretical Computer Science*, 305(1):3 – 15, 2003.

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

[5] Siu-Wing Cheng, Tamal K Dey, and Jonathan Shewchuk. *Delaunay mesh generation*. CRC Press, 2012.

[6] Andrey Chernikov and Nikos Chrisochoides. Multitissue tetrahedral image-to-mesh conversion with guaranteed quality and fidelity. *SIAM Journal on Scientific Computing*, 33:3491–3508, 2011.

[7] Panagiotis Foteinos, Andrey Chernikov, and Nikos Chrisochoides. Guaranteed quality tetrahedral Delaunay meshing for medical images. *Computational Geometry Theory and Applications*, 47:539–562, 2014.

[8] Paul-Louis George and Houman Borouchaki. *Delauney triangulation and meshing: application to finite elements*. Hermes Science Publications, Paris, 1998.

[9] Yasushi Ito, Alan Shih, Anil Erukala, Bharat Soni, Andrey Chernikov, Nikos Chrisochoides, and Kazuhiro Nakahashi. Parallel unstructured mesh generation by an advancing front method. *Mathematics and Computers in Simulation*, 75:200–209, September 2007.

[10] Michael Kazhdan, Allison Klein, Ketan Dalal, and Hugues Hoppe. Unconstrained isosurface extraction on arbitrary octrees. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, SGP '07, pages 125–133, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[11] François Labelle and Jonathan Richard Shewchuk. Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles. *ACM Transactions on Graphics*, 26(3):57.1–57.10, July 2007. Special issue on Proceedings of SIGGRAPH 2007.

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

[13] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *COMPUTER GRAPHICS*, 21(4):163–169, 1987.

[14] David L Marcum and Frédéric Alauzet. Unstructured mesh generation using advancing layers and metric-based transition for viscous flowfields. In *21st AIAA Computational Fluid Dynamics Conference*, 2013.

[15] Calvin R. Maurer, Jr., Rensheng Qi, and Vijay 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):265–270, February 2003.

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

[17] Scott A. Mitchell and Stephen A. Vavasis. Quality mesh generation in higher dimensions. *SIAM Journal on Computing*, 29(4):1334–1370, 2000.

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

[19] S. Raman and R. Wenger. Quality isosurface mesh generation using an extended marching cubes lookup table. *Computer Graphics Forum*, 27(3):791–798, 2008.

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

[21] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '92, pages 65–70, New York, NY, USA, 1992. ACM.

[22] Jingqi Yan, Pengfei Shi, and David Zhang. Mesh simplification with hierarchical shape analysis and iterative edge contraction. *IEEE Transactions on Visualization and Computer Graphics*, 10(2):142–151, March 2004.