2011

Multitissue Tetrahedral Image-to-Mesh Conversion with Guaranteed Quality and Fidelity

Andrey N. Chernikov
Old Dominion University

Nikos P. Chrisochoides
Old Dominion University

Original Publication Citation

Chernikov, A. N., & Chrisochoides, N. P. (2011). Multitissue tetrahedral image-to-mesh conversion with guaranteed quality and fidelity. *SIAM Journal on Scientific Computing*, *33*(6), 3491-3508. doi:10.1137/100815256

SIAM J. SCI. COMPUT.
Vol. 33, No. 6, pp. 3491–3508

MULTITISSUE TETRAHEDRAL IMAGE-TO-MESH CONVERSION WITH GUARANTEED QUALITY AND FIDELITY*

ANDREY N. CHERNIKOV† AND NIKOS P. CHRISOCHOIDES†

Abstract. We present a novel algorithm for tetrahedral image-to-mesh conversion which allows for guaranteed bounds on the smallest dihedral angle and on the distance between the boundaries of the mesh and the boundaries of the tissues. The algorithm produces a small number of mesh elements that comply with these bounds. We also describe and evaluate our implementation of the proposed algorithm that is compatible in performance with a state-of-the art Delaunay code, but in addition solves the small dihedral angle problem.

Key words. image-to-mesh conversion, guaranteed quality, fidelity, multitissue

AMS subject classifications. 65D18, 68U20, 68W05, 92C10, 92C50

DOI. 10.1137/100815256

1. Introduction.

The problem of unstructured image-to-mesh conversion (I2M) is the following. Given an image as a collection of voxels, such that each voxel is assigned a label of a single tissue or of the background, construct a tetrahedral mesh that overlays the tissues and conforms to their boundaries. In this paper we present an algorithm for constructing meshes that are suitable for real-time finite element (FE) analysis, i.e., they satisfy the following requirements:

1. Elements do not have arbitrarily small angles which lead to poor conditioning of the stiffness matrix in FE analysis for biomechanics applications. In particular, we guarantee that all dihedral angles are above a user-specified lower bound which can be set to any value up to 35.26°. In contrast, guaranteed-quality Delaunay methods only satisfy a bound on circumradius-to-shortest-edge ratio which in three dimensions does not imply a bound on dihedral angles.
2. The mesh offers a reasonably close representation (fidelity) of the underlying tissues. Since the image is already an approximation (up to a pixel granularity) of a continuous physical object, even a strict matching of the mesh to individual pixel's boundaries will not lead to a mesh which is completely faithful to the boundaries of the object. Moreover, this approach will produce a large number of elements that will slow down the solver. Instead, our solution is to expose parameters that allow for a trade-off between the fidelity and the final number of elements with the goal of improving the end-to-end execution time of the FE analysis codes.
3. The number of tetrahedra in the mesh is as small as possible provided the two requirements above are satisfied. This requirement is based on the cost of assembling and solving a sparse system of linear equations in the FE method, which directly depends on the number of tetrahedra [20, 23]. We achieve this goal by developing a specialized mesh decimation procedure.

*Submitted to the journal's Methods and Algorithms for Scientific Computing section November 17, 2010; accepted for publication (in revised form) September 6, 2011; published electronically December 15, 2011. This work was supported in part by NSF grants CCF-1139864, CCF-1136538, and CSI-1136536, as well as by the John Simon Guggenheim Foundation and the Richard T. Cheng Endowment.
http://www.siam.org/journals/sisc/33-6/81525.html
†Department of Computer Science, Old Dominion University, Norfolk, VA 23529 (achernik@cs.odu.edu, nikos@cs.odu.edu).

4. The mesh can be constructed within tight real-time time constraints enforced by clinical applications. Below we describe our efficient implementation which is close in performance and even faster than a state-of-the art Delaunay code.

There is a large body of work on constructing guaranteed quality meshes for computer aided design (CAD) models. The specificity of CAD-oriented approaches is that the meshes have to match exactly to the boundaries of the models. The most widely used guaranteed-quality CAD-oriented approach is based on Delaunay refinement; see [6] and the references therein. However, the problem with Delaunay refinement in three dimensions is that it allows only for a bound on circumradius-to-shortest-edge ratio of tetrahedra, which does not help to improve the dihedral angles. As a result, almost flat tetrahedra called slivers can survive. There are a number of postprocessing techniques to eliminate slivers [2, 3, 4, 9, 14, 21]. While some have been shown to produce very good dihedral angles in practice, we are not aware of an implementation that can *guarantee* significant (1° and above) dihedral angle bounds.

Labelle and Shewchuk [8] described a guaranteed-quality tetrahedral meshing algorithm for general surfaces. They offer a one-sided fidelity guarantee (from the mesh to the model) in terms of the Hausdorff distance and, provided the surface is sufficiently smooth, also the guarantee in the other direction (from the model to the mesh). Their algorithm first constructs an octree that covers the model, then fills the octree leaves with high-quality template elements, and finally warps the mesh vertices onto the model surface, or inserts vertices on the surface, and locally modifies the mesh. Using interval arithmetic, they prove that new elements have dihedral angles above a certain threshold. However, images are not smooth surfaces, and to the best of our knowledge, this technique has not been extended to mesh images. One approach could be to interpolate or approximate the boundary pixels by a smooth surface, but it would be complicated by the need to control the maximum approximation (interpolation) error. On the other hand, an I2M solution can benefit from the fact that images provide more information on their structure than general surfaces. For example, in our proposed I2M algorithm we do not have to struggle with the problem of quadruple-zero tetrahedra, which complicates the solution in [8]. Quadruple-zero tetrahedra are those that have all four vertices on the surface, and it is not clear if they should be classified as interior or exterior.

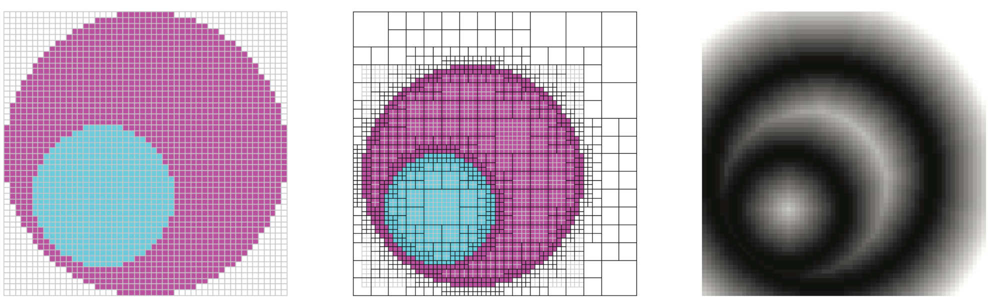
There are also heuristic solutions to the I2M problem, some of them developed in our group [5, 10], that fall into two categories: (1) first coarsen the boundary of the image, and then apply CAD-based algorithms to construct the final mesh; (2) construct the mesh which covers the image, and then warp some of the mesh vertices onto the image surface. The first approach tries to address the fidelity and then the quality requirements, while the second approach does it in reverse order. Unfortunately, neither of these approaches can guarantee the quality of elements in terms of dihedral angles. Both face the same underlying difficulty which consists in separating the steps that attempt to satisfy the quality and the fidelity requirements. As a result, the output of one step does not produce an optimal input for the other step.

The solution we propose in this paper is to simultaneously satisfy the quality and the fidelity requirements. We achieve this goal by constructing an initial fine mesh with very high quality and fidelity. The construction of this mesh is feasible due to the specific structure of the input, which is a collection of cubic blocks corresponding to the voxels of the image. This initial mesh, however, has a large number of elements because it is a one-fits-all solution with respect to the angle and fidelity parameters, for the given image, since it satisfies the highest dihedral angle and fidelity bounds.

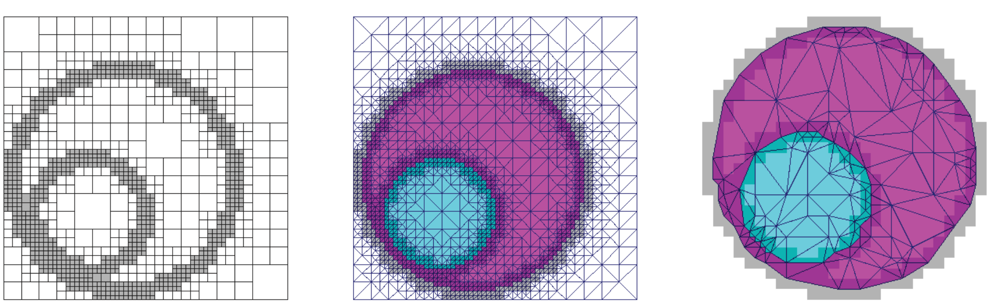
Therefore, we implement a postprocessing decimation step that coarsens the mesh to a much lower number of elements while at all times maintaining the required fidelity and quality bounds. Mesh coarsening using vertex removal operation, which we use in our algorithm, has been employed in various formulations in a large number of works for a variety of optimization problems; see, e.g., [7, 11, 13, 15, 22] and the references therein. The proposed approach may appear to require excessive amounts of computational time and storage. However, we demonstrate that with a carefully optimized implementation it can be used to mesh three-dimensional images of practically significant sizes even on a regular desktop workstation. Furthermore, our time measurements show that for two complex medical atlas images (brain and abdominal) it is 28% to 42% faster than a state-of-the art Delaunay software.

The rest of the paper is organized as follows. In section 2 we describe the proposed algorithm in detail. In section 3 we present the implementation details along with the experimental evaluation. Section 4 concludes the paper.

2. Algorithm. The proposed algorithm works for both two- and three-dimensional images. For explanation purposes, in Figure 1 we show a simple two-dimensional example of an image being converted into a triangular mesh. The size of this image



(a) **(b)** **(c)**



(d) **(e)** **(f)**

FIG. 1. *An illustration of the main steps performed by our I2M algorithm.* (a) *The input two-dimensional image of size* 50×50*. It shows two circles, displayed with cyan and magenta, against the white background. The angle bound is set to* $20°$*, and the fidelity bounds are both set to two voxels.* (b) *The quadtree with leaves refined to meet the bounds on triangle quality and fidelity.* (c) *Euclidean distance transform.* (d) *The leaves of the quadtree that are within the fidelity bound are marked.* (e) *The original fine mesh,* 2076 *triangles inside the circles,* 3534 *triangles total.* (f) *The decimated mesh,* 263 *triangles inside the circles, the outside triangles are removed. The intertissue boundaries are within the marked leaves and therefore within the requested tolerance.*

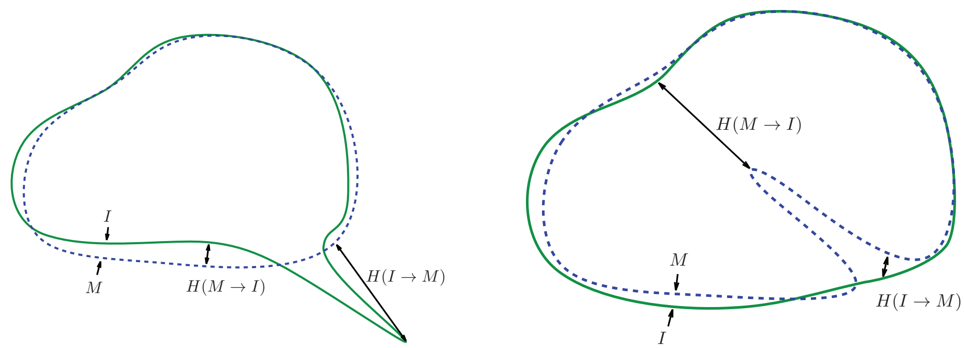


FIG. 2. *An illustration of the Hausdorff distance. Left:* $H(I \leftrightarrow M) = H(I \to M)$. *Right:* $H(I \leftrightarrow M) = H(M \to I)$.

is 50×50 voxels. It defines two circular objects, which could represent tissues or materials, one within another, shown with different colors (cyan and magenta) against white background.

The mesh has to provide a faithful representation of the underlying tissues, i.e., each element needs to be marked with the physical properties of a unique type of tissue. To measure the distance between the boundaries of the two regions (the image of a tissue and the corresponding submesh), we use the Hausdorff distance. It can be specified as either a two-sided distance or a one-sided distance. For tissue boundary I and mesh boundary M, the one-sided distance from I to M is given by

$$H(I \to M) = \max_{i \in I} \min_{m \in M} d(i, m),$$

where $d(\cdot, \cdot)$ is the regular Euclidean distance. The one-sided distance from M to I is given similarly by

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

Note that $H(I \to M)$ is generally not equal to $H(M \to I)$. The two-sided distance is symmetric:

$$H(I \leftrightarrow M) = \max\{H(I \to M), H(M \to I)\};$$

see Figure 2.

2.1. Input. The input to our algorithm is a two- or a three-dimensional bitmap; see Figure 1(a). Each voxel of the bitmap corresponds to a separate material or tissue, as indicated by a single label (color) assigned to this voxel. The user also supplies the desired angle lower bound and fidelity bounds. We will use starred letters θ^* and H^* to denote the bounds on the angle and the Hausdorff distance, respectively.

2.2. Construction of the octree. We construct an octree (in three dimensions) or a quadtree (in two dimensions) that satisfies the following properties (see Figure 1(b)):

1. The octree (equivalently, its root node) completely encloses all the tissues from the image, except possibly for the background voxels that can be ignored.

2. There is extra space, equal to or greater than the maximum of the fidelity parameters, between the tissues and the exterior boundaries of the octree.

3. The boundaries between the leaves correspond exactly to the boundaries between the voxels. This is possible by using integer coordinates corresponding to voxel indices.

4. No leaf contains voxels from multiple tissues. The nodes of the tree are split recursively until all the leaves satisfy this condition.

5. The sizes of the octree leaves respect the 2-to-1 rule, i.e., two adjacent leaves must differ in depth by no more than one.

2.3. Computation of the distance transform. A distance transform of an image is an assignment to every voxel of a distance to the nearest feature of the image. In our case, the features are the boundaries between the tissues, and the distance is measured in the usual Euclidean metric. See, e.g., Figure 1(c), where darker shades correspond to the voxels that are closer to tissue boundaries. We implemented the Euclidean distance transform (EDT) algorithm described by Maurer, Qi, and Raghavan [12]. We chose this algorithm for two reasons: (1) its linear time complexity with respect to the number of voxels, and (2) it is formulated to work in an arbitrary dimension. We run the EDT computation on 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.

2.4. Labeling of octree leaves. For each leaf of the octree, we find the maximum distance to the intertissue boundaries, using the EDT values of the voxels enclosed by this leaf. In Figure 1(d) we marked the leaves that are within the tolerance (two voxels in this example) in transparent gray.

2.5. Filling in the octree. We process the leaves in the order of their size, starting with the smallest; see Figure 1(e) for a two-dimensional example. The procedure is recursive on dimension: to *triangulate* an n-dimensional *face* of the leaf, first *triangulate* all of its $(n-1)$-dimensional sub-*faces*. If at least one of the $(n-1)$-dimensional sub-*faces* is split by a midpoint, introduce the midpoint of the n-dimensional *face* and connect to the elements of the $(n-1)$-dimensional *triangulation* of the sub-*face* to construct the n-dimensional *triangulation* of the *face*. If none of the sub-*faces* was split, use the diagonals of the *face*.

This procedure is equivalent to using a finite number of predefined canonic leaf triangulations, with the benefits of reducing manual labor and applicability in an arbitrary dimension. For all possible resulting leaf triangulations we obtain a minimum planar angle of 45° in two dimensions or a minimum dihedral angle of 35.26° in three dimensions; see Figures 3 and 4 for an illustration. Hence, these are the bounds that the algorithm can guarantee.

Once all octree leaves are filled with tetrahedra, we finish the construction of the mesh data structure by identifying face-adjacent tetrahedra, in order to facilitate the decimation procedure.

2.6. Mesh decimation. We say vertex $\mathbf{u}$ is *merged* to vertex $\mathbf{v}$ if vertex $\mathbf{u}$ and edge $\mathbf{uv}$ are removed from the mesh, such that all tetrahedra (triangles) incident upon edge $\mathbf{uv}$ are also removed from the mesh and the remaining edges that were incident upon $\mathbf{u}$ now become incident upon $\mathbf{v}$. See Figure 5 for an illustration.

Our decimation algorithm is shown in Figure 6. We maintain a queue Q of mesh vertices that are candidates for merging. The algorithm removes from and adds vertices to Q until it becomes empty. Note that after the initialization a vertex can be added on the queue only as a result of a merge of an adjacent vertex. Therefore, when none of the vertices in Q passes the check for a merge, Q will become empty and

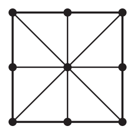


FIG. 3. *The union of all possible edges of a triangulation obtained using our algorithm for a two-dimensional square corresponding to a leaf of the octree. The edges meet at angles greater than or equal to* $45°$. *Therefore, this is the lower bound for the planar angle in the initial two-dimensional triangulation of the quadtree.*

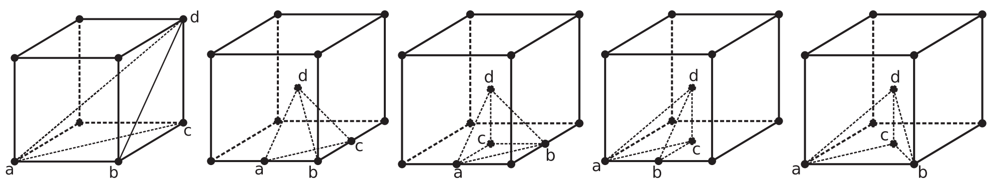


FIG. 4. *All possible shapes of the initial tetrahedra* (**abcd**) *filling a cubic leaf of the octree, up to symmetry. The smallest dihedral angles are (left to right):* $45°$, $35.26°$, $54.74°$, $45°$, *and* $45°$. *Therefore,* $35.26°$ *is the lower bound for the dihedral angle in the initial three-dimensional tetrahedralization of the octree.*

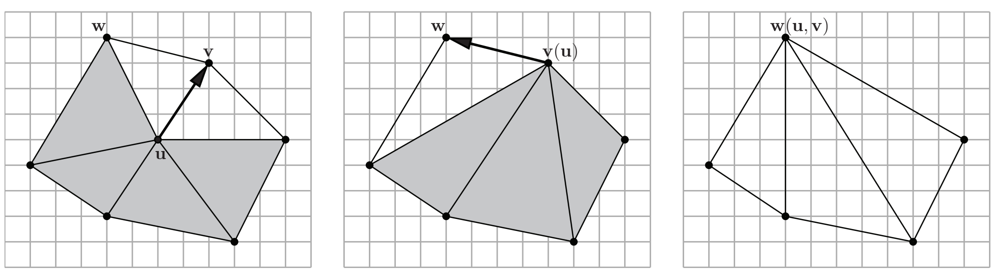


FIG. 5. *An illustration of the vertex merge operation. Left: Vertex* **u** *is evaluated for merging to vertex* **v**. *The shaded triangles need to be checked for the effect of changing their shape. Center: Vertex* **u** *is merged to vertex* **v**. *The list of vertices in brackets shows merge history. Vertex* **v** *is evaluated for merging to vertex* **w**. *Right: Vertex* **v** *is merged to vertex* **w**.

the decimation procedure will terminate. Suppose n is the total number of vertices in the original mesh. Every vertex is added to Q once in the beginning. Afterwards, a vertex is added to Q only if one of its vertex neighbors was merged. If m is the total number of merges performed (obviously $m < n$), then the total number of evaluations is bounded from above by $n+cm$, where c is the maximum number of vertex neighbors for each vertex. For two dimensions, it is easy to see that c is constant: since at all times during the run of the algorithm the minimum planar angle is bounded by a constant threshold θ^*, the degree of each vertex is bounded by $360/\theta^*$. In three dimensions, to our knowledge, there is no such clear relationship. The algorithm starts with a semi-regular filling of the octree in which the degree of each vertex is a small number, and then during decimation the maximum vertex degree can increase. Our experiments show, however, that it does not increase dramatically. For the three examples presented in the paper with millions of vertices, the maximum observed vertex degree for the brain atlas is 187, for the abdominal atlas is 326, and for the ball is 420.

DECIMATION($\mathcal{M}$, $\mathcal{O}$, θ^*, $H^*(I \to M)$, $H^*(M \to I)$)
Input: $\mathcal{M}$ is the initial mesh
    $\mathcal{O}$ is the octree
    θ^* is the lower bound on the minimum angle bound
    $H^*(I \to M)$ and $H^*(M \to I)$ are the upper bounds
    on one-sided Hausdorff distances
Output: Decimated mesh $\mathcal{M}$ that respects angle and fidelity bounds
Initialize Q to the set of all vertices in $\mathcal{M}$
while $Q \neq \emptyset$
    Pick $\mathbf{v}_i \in Q$
    $Q \longleftarrow Q \setminus \{\mathbf{v}_i\}$
    Find $A = \{\mathbf{v}_j\}$ the set of vertices adjacent to $\mathbf{v}_i$
    for each $\mathbf{v}_j \in A$
        Find $T = \{t_k\}$ the set of tetrahedra incident
            upon $\mathbf{v}_i$ and not incident upon $\mathbf{v}_j$
        for each $t_k \in T$
            Replace $\mathbf{v}_i$ with $\mathbf{v}_j$ in t_k
        endfor
        if (CHECK4QUALITY(T, θ^*) $\wedge$
                CHECK4FIDELITY(T, $\mathcal{O}$, $H^*(I \to M)$, $H^*(M \to I)$) $\wedge$
                CHECK4CONNECTIVITY(T, $\mathcal{M}$))
            Merge $\mathbf{v}_i$ to $\mathbf{v}_j$, update $\mathcal{M}$
            $Q \longleftarrow Q \cup A$
            break
        endif
        for each $t_k \in T$
            Replace $\mathbf{v}_j$ with $\mathbf{v}_i$ in t_k
        endfor
    endfor
endwhile
return $\mathcal{M}$

FIG. 6. *A high-level description of the decimation algorithm. The actual implementation is slightly different and more elaborate to support efficient data structures and to minimize computation; for more details see section* 3.

2.6.1. Maintaining element quality. The function CHECK4QUALITY(T, θ^*) returns *true* if and only if all elements on the list T are not inverted and have all angles (planar in two dimensions or dihedral in three dimensions) above the bound θ^*. Therefore, the merge is not accepted if at least one newly created angle is smaller than θ^*.

2.6.2. Maintaining fidelity to boundaries. This check, represented by the function CHECK4FIDELITY(T, $\mathcal{O}$, $H^*(I \to M)$, $H^*(M \to I)$) consists of two parts, for each of the one-sided Hausdorff distances. To evaluate the distance from the boundary of the submesh to the boundary of the corresponding tissue, for each of the boundary faces (edges in two dimensions or triangles in three dimensions) of elements in T, we recursively check for the intersection with the octree nodes. If at least one of the faces intersects at least one of the nodes marked as outside the fidelity tolerance, the merge is discarded. To evaluate the distance from the boundary of each tissue to the boundary of the corresponding submesh, for each vertex we maintain a cumulative list of the boundary vertices that were merged to it. If at least one of the boundary vertices, as a result of a sequence of merges, is further away from its original location than the corresponding fidelity tolerance, the merge is discarded.

2.6.3. Maintaining tissue connectivity. The geometric constructions used in our algorithm are assigned colors based on their location with respect to the tissues on the bitmap:

1. Each leaf of the octree (quadtree) derives the color from the block of voxels that it encloses. Remember that the nodes are split recursively until they enclose voxels of a single color; in the limit case a leaf encloses a single voxel.
2. Each tetrahedron in three dimensions (or triangle in two dimensions) derives its color from the octree (quadtree) leaf that is used to tetrahedralize; it keeps the original color even after it changes shape due to vertex merge. As a result, all tetrahedra (triangles) are always correctly classified with respect to the underlying tissues, including the quadruple-zero (triple-zero) ones.
3. Each vertex derives its color from the block of incident voxels (eight in three dimensions or four in two dimensions); if the block of voxels has multiple colors, the vertex is considered *boundary*.

The following rules help us maintain the original structure of the intertissue boundaries: (1) boundary vertices cannot merge to nonboundary vertices, (2) a vertex cannot merge to a nonboundary vertex of a different color, and (3) a boundary vertex can merge to another boundary vertex only along a boundary edge—this helps to prevent the case when a vertex from one boundary merges to another boundary along a nonboundary edge, and thus the merge connects the parts of the boundaries that were not originally connected.

3. Implementation and evaluation. We implemented the proposed lattice decimation (LD) algorithm in C++, in both two and three dimensions. The following implementation decisions have significantly improved the performance:

1. Most of the computation is performed in integer arithmetic. This is possible because vertex coordinates are integers; they are indices with respect to the matrix of voxels. The only floating point computation is involved in the comparison of cosines of angles since long integer arithmetic could overflow. In addition, the lengths of the integer variables correspond to the range of values of each specific arithmetic operation, such that very long integers are used only when necessary to avoid overflow. For example, if variable x is represented with b bits, then x^2 requires $2b$ bits, while x^4 requires $4b$ bits; using $4b$ bits for x^2 would be excessive.
2. All expensive mathematical functions, such as trigonometric, square root, etc., including floating point division, are avoided in the computationally critical parts. Instead, computation is performed on squares, cosines, and other functions of the original values.
3. We wrote customized memory allocation functions, such that objects that are created in large numbers but occupy little memory each (vertices, tetrahedra, nodes of the tree) are allocated in contiguous memory buffers. This improvement decreases memory fragmentation and allocation overheads.
4. We arranged the sequences of complex pass-fail condition evaluations such that the least expensive conditions and those most likely to fail are evaluated first, while the most expensive ones are evaluated last.

Clearly, the performance of our algorithm in terms of the running time, as well as the number and the size of tetrahedra, depends on the input geometry. Therefore, our approach to the evaluation is based on two experimental setups. The first setup uses a very simple three-dimensional geometry (sphere) and evaluates the performance with respect to different sizes of the sphere. This way, we gain insight into the performance for a controlled range of domain geometries with varied ratio

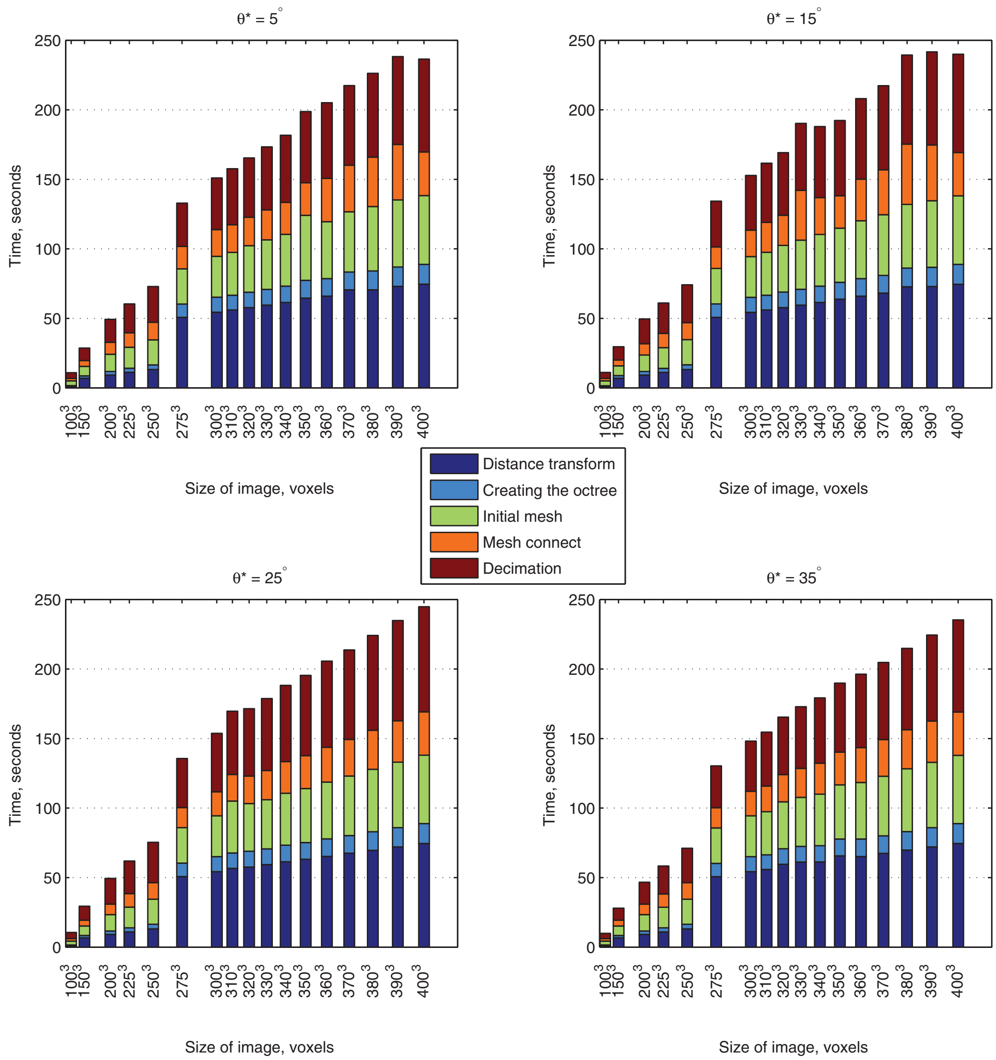


FIG. 7. *A breakdown of the total LD time into the major computational parts, as the diameter of the sphere varies from* 100 *to* 400 *voxels.* $H^*(I \leftrightarrow M) = 0$.

of $\max_{\mathbf{p}\in\Omega} \mathrm{lfs}\,(\mathbf{p}) \,/\, \min_{\mathbf{p}\in\Omega} \mathrm{lfs}\,(\mathbf{p})$. The local feature size function $\mathrm{lfs}\,(\mathbf{p})$ for a given point $\mathbf{p}$ is equal to the radius of the smallest ball centered at $\mathbf{p}$ that intersects two nonincident elements of domain Ω. For images, these elements are vertices, edges, and faces from the tissue boundary. For a sphere, this ratio is simply equal to its radius and therefore is easy to vary.

The second setup uses two complex real-world medical images: an abdominal atlas [18] and a brain atlas [19]. The atlases come with a segmentation, such that each voxel is assigned a label which corresponds to one of 75 abdominal and 149 brain tissues. All tests were performed on a desktop with Intel Core i7 CPU with 2.80 GHz and 8 GB of main memory.

3.1. Synthetic benchmark—three-dimensional sphere. Figure 7 shows a breakdown of the total time into the major computational parts, as the diameter of the sphere grows from 100 to 400 voxels. 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 (which is expensive because involves a search through the adjacent leaves), and the decimation. Here and in all other time measurements we exclude the time taken by input and output and by the process of initializing the data structure representing the initial image. 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.

Table 1 shows the output mesh size of our implementation for a sphere of a fixed diameter of 400 voxels. In these tests, having fixed the dihedral angle bound and the diameter of the sphere, we vary each of the two one-sided Hausdorff distance bound parameters independently. We present four columns, for different interesting values of the dihedral angle bound (5°, 15°, 25°, and 35°) spread through the range of its feasible values (0° to 35.26°). As we can see from Table 1, for all configurations the output mesh size is high when either one or both of the H^* parameters is low ($H^*(M \to I)$ has higher influence than $H^*(I \to M)$ due to implementation-specific details) and decreases as the H^* bounds decrease. Indeed, the weaker constraints can be satisfied with a smaller number of tetrahedra. The same argument explains why the final number of tetrahedra grows as the dihedral angle bound increases. The total running time in these tests does not change significantly with the variation of the fidelity bounds and is close to 250 seconds. The actually obtained smallest dihedral angles in all experiments are between θ^* and $\theta^* + 0.3°$.

3.2. Three-dimensional medical images. The size of the abdominal atlas is $256 \times 256 \times 113$ voxels and the size of the brain atlas is $256 \times 256 \times 159$ voxels. In both cases each voxel has side lengths of 0.9375, 0.9375, and 1.5000 units in x, y, and z directions, respectively. Before meshing the atlases, we resampled them with voxels of equal side length corresponding to the original 0.9375 units. As a result, in both cases we obtained equally spaced images that were used for meshing. Figures 8, 9, 10 and 11 show the atlas images and corresponding three-dimensional meshes.

In Table 2 we list the final number of tetrahedra, the smallest dihedral angle, and the total running time for both images, as we vary the H^* and θ^* parameters. To obtain a point of reference for these numbers, we conducted a separate experiment using a state-of-the art open source tetrahedral mesh generator Tetgen [16]. Tetgen is designed to work with piecewise linear complexes (PLCs) and not images. Therefore, to make it process the same tissue geometries, we extracted the voxel faces corresponding to the boundaries between different tissues and between the tissues and the surrounding space and saved them in the PLC format files that we passed to Tetgen. The main difference between the two methods is that Tetgen does not provide any guarantees on the dihedral angle (since it is designed to improve only circumradius-to-shortest-edge ratio of tetrahedra for general PLCs), and its empirical smallest dihedral angle will generally be different for other input geometries. As can be expected, the meshes produced by Tetgen had low smallest dihedral angles, around 5°. At the same time, our LD algorithm can provide guaranteed smallest dihedral angles up to 35.26° for all input images.

For all our time measurements we excluded all data preprocessing, such as image resampling, surface extraction, and input and output. We see that our LD implementation is faster than Tetgen by a significant margin for both atlas images. Figures 12 and 13 show breakdowns of the total LD time into the main computational compo-

Table 1
The size of the LD final mesh for the image of a sphere of diameter 400 *voxels, depending on the one-sided Hausdorff bounds varied independently, and on the angle bound.*

$H^*(I \to M)$	$H^*(M \to I)$	Resulting number of tetrahedra			
		$\theta^* = 5°$	$\theta^* = 15°$	$\theta^* = 25°$	$\theta^* = 35°$
0	0	2,676,905	3,533,827	6,228,998	7,568,401
0	1	2,628,051	3,486,279	6,192,725	7,585,208
0	2	1,051,537	1,828,910	5,228,905	6,810,152
0	4	1,049,190	1,831,595	5,241,114	6,816,286
0	6	1,052,819	1,833,990	5,239,028	6,810,705
0	8	1,050,645	1,830,927	5,239,671	6,812,804
0	10	1,049,490	1,828,948	5,235,814	6,805,851
1	0	2,674,541	3,536,514	6,231,690	7,585,208
1	1	2,619,272	3,483,032	6,180,726	7,585,208
1	2	615,495	1,294,315	4,832,255	6,702,274
1	4	622,144	1,296,503	4,845,919	6,708,335
1	6	620,531	1,286,649	4,853,066	6,700,642
1	8	621,753	1,288,004	4,832,317	6,707,849
1	10	618,635	1,287,916	4,833,412	6,695,580
2	0	2,678,337	3,528,875	6,219,659	7,567,694
2	1	2,622,046	3,471,997	6,169,755	7,567,694
2	2	420,742	954,058	4,761,521	6,689,753
2	4	420,287	963,392	4,767,670	6,694,121
2	6	418,914	956,169	4,767,004	6,687,824
2	8	418,605	967,353	4,761,786	6,693,867
2	10	415,096	957,492	4,764,773	6,682,444
4	0	2,680,017	3,529,907	6,227,939	7,572,235
4	1	2,610,273	3,470,074	6,181,200	7,572,235
4	2	206,744	669,844	4,761,246	6,693,880
4	4	205,792	669,727	4,761,246	6,693,880
4	6	201,715	674,707	4,761,532	6,687,545
4	8	197,087	664,810	4,757,938	6,693,670
4	10	198,760	662,854	4,765,483	6,682,099
6	0	2,673,759	3,528,173	6,222,394	7,567,112
6	1	2,618,425	3,470,097	6,173,119	7,567,112
6	2	108,961	627,153	4,761,532	6,687,545
6	4	108,667	627,153	4,761,532	6,687,545
6	6	108,667	627,153	4,761,532	6,687,545
6	8	111,692	616,427	4,757,938	6,693,670
6	10	109,520	600,885	4,765,479	6,682,099
8	0	2,673,161	3,527,634	6,221,837	7,569,203
8	1	2,612,149	3,467,740	6,174,740	7,569,203
8	2	76,523	604,821	4,757,938	6,693,670
8	4	76,108	604,795	4,757,938	6,693,670
8	6	76,108	604,795	4,757,938	6,693,670
8	8	76,108	604,795	4,757,938	6,693,670
8	10	78,542	594,901	4,765,479	6,682,099
10	0	2,671,255	3,534,080	6,216,296	7,567,005
10	1	2,613,405	3,466,948	6,167,959	7,567,005
10	2	58,510	593,862	4,765,479	6,682,099
10	4	56,975	593,519	4,765,479	6,682,099
10	6	56,975	593,519	4,765,479	6,682,099
10	8	56,975	593,519	4,765,479	6,682,099
10	10	56,975	593,519	4,765,479	6,682,099

nents as the symmetric Hausdorff distance bound changes from 0 to 2 voxels. Similar to the case of the sphere, we see little change both in the running time and in its distribution with the variation of H^* and θ^* parameters.

As far as the number of tetrahedra, the difference between Tetgen and LD is insignificant, although in both cases in favor of LD, for the bound of $H^*(I \leftrightarrow M) = 0$

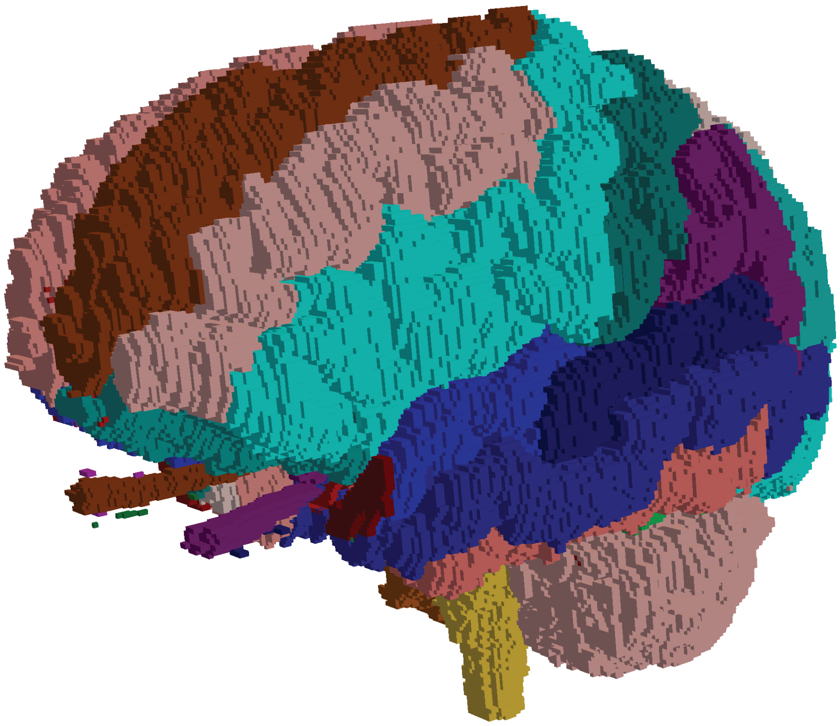


FIG. 8. *Three-dimensional image of the brain atlas.*

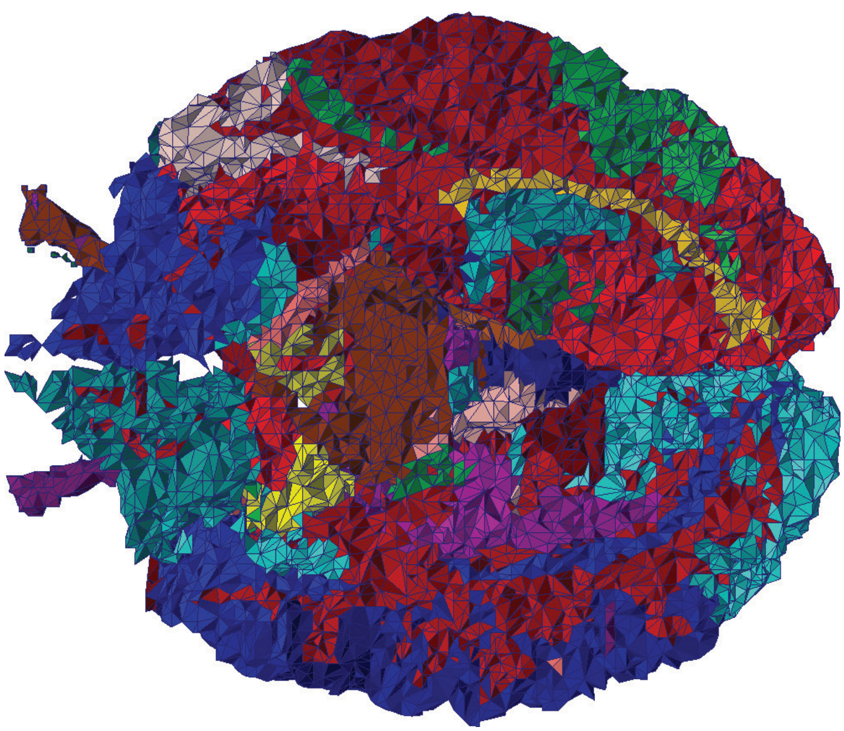


FIG. 9. *A slice through the LD mesh of the brain atlas for* $\theta^* = 15^\circ$ *and* $H^*(I \leftrightarrow M) = 2$.

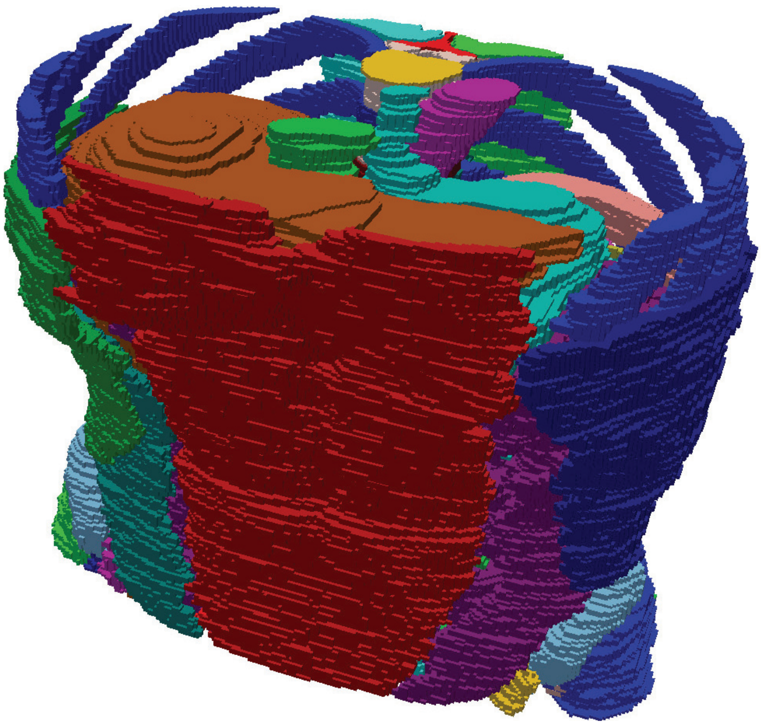


FIG. 10. *Three-dimensional image of the abdominal atlas.*

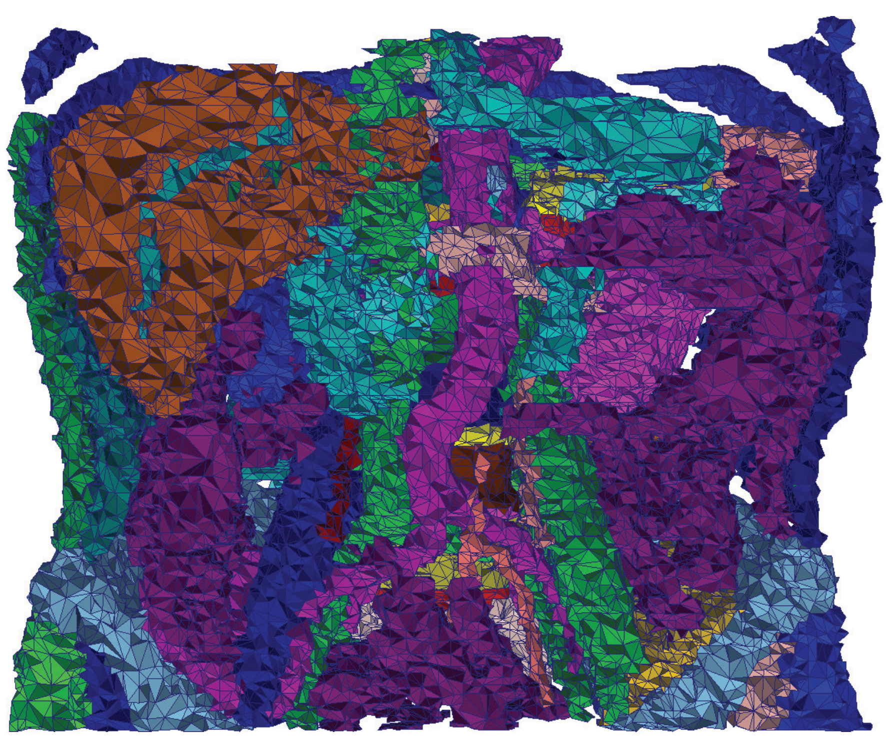


FIG. 11. *A slice through the LD mesh of the abdominal atlas for* $\theta^* = 15^\circ$ *and* $H^*(I \leftrightarrow M) = 2$.

TABLE 2

Experimental evaluation of Tetgen and LD. AA is abdominal atlas and BA is brain atlas. Angles are measured in degrees, and time is measured in seconds.

Code	Input bounds		Resulting mesh statistics				Total time	
	$H^*(I \leftrightarrow M)$	θ^*	Number of tetrahedra		Smallest dih. angle			
			AA	BA	AA	BA	AA	BA
Tetgen	0 (implicit)	n/a	3,501,569	3,398,654	4.725	5.002	169	128
LD	0	5	3,332,477	3,267,276	5.002	5.003	122	74
		15	3,932,131	3,698,787	15.002	15.002	120	74
		25	6,768,472	6,266,442	25.066	25.066	120	73
		35	7,806,292	6,773,951	35.264	35.264	110	68
	1	5	3,134,565	3,126,393	5.000	5.005	134	82
		15	3,714,781	3,555,290	15.002	15.002	128	82
		25	6,415,049	6,082,604	25.066	25.066	127	75
		35	7,625,360	6,657,475	35.264	35.264	115	68
	2	5	557,242	521,952	5.000	5.002	111	71
		15	924,642	874,764	15.000	15.000	116	77
		25	4,506,340	5,137,417	25.061	25.066	135	83
		35	6,658,700	6,318,544	35.097	35.264	119	71

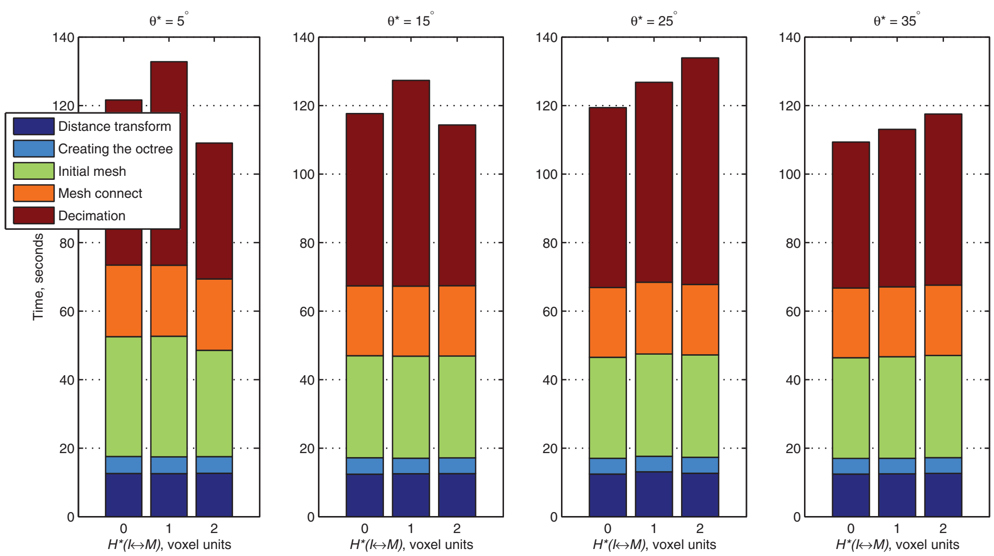


FIG. 12. *A breakdown of the total LD time for the abdominal atlas, for varied θ^* and $H^*(I \leftrightarrow M)$.*

which allows for a comparison with respect to the same fidelity, and $\theta^* = 5°$, which is close to the empirical Tetgen angles. In Figure 14 we show the final number of tetrahedra for both atlases, as we vary $H^*(I \leftrightarrow M)$ and θ^*.

Table 3 presents our experimental evaluation of the I2M conversion functionality offered by the Computational Geometry Algorithms Library (CGAL) [1]. We used function `make_mesh_3` with the following parameters:

- `domain` is the brain or abdominal atlas image without resampling.
- `facet_angle` is a lower bound on the planar angle of boundary faces; we set it to an ignored value.
- `facet_size` is an upper bound on the radii of the surface Delaunay balls; we set it to an ignored value.

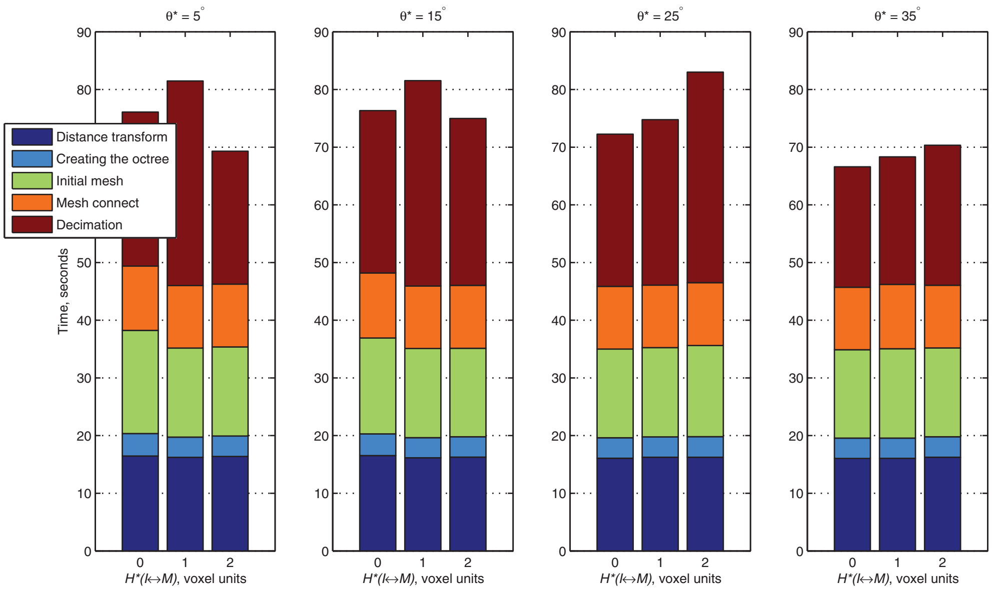


FIG. 13. *A breakdown of the total LD time for the brain atlas, for varied θ^* and $H^*(I \leftrightarrow M)$.*

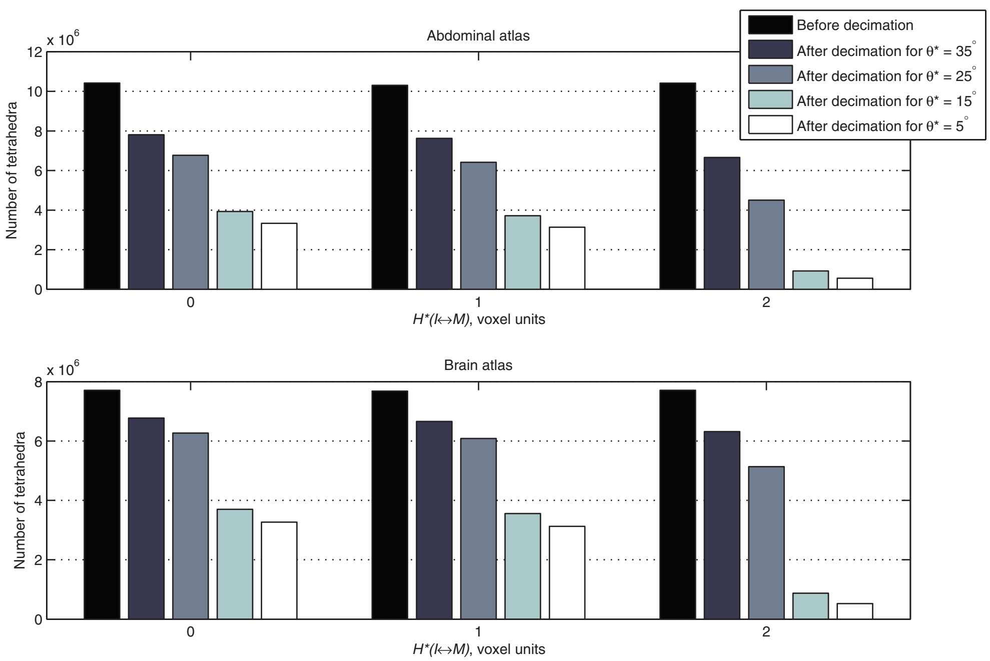


FIG. 14. *Final number of tetrahedra using LD, for varied θ^* and $H^*(I \leftrightarrow M)$.*

- `facet_distance` is an upper bound on the distance between the circumcenters of surface facets and the centers of the corresponding surface Delaunay balls; we varied this parameter as shown in the table.
- `cell_radius_edge_ratio` is an upper bound on radius-edge ratio of tetrahedra; we used 2.0.

TABLE 3

Experimental evaluation of the I2M functionality offered by CGAL. AA stands for abdominal atlas and BA stands for brain atlas. Angles are measured in degrees, time is measured in seconds.

`facet_distance`	Resulting mesh statistics: # of tetrahedra AA	# of tetrahedra BA	Smallest dih. angle AA	Smallest dih. angle BA	# of subdoms. AA	# of subdoms. BA	Total time AA	Total time BA
`no_lloyd(), no_odt(), perturb(), exude()`								
0.5	659,104	751,640	2.833	2.365	74	145	65.5	50.4
1.0	174,080	209,437	3.355	2.246	74	143	18.0	16.5
2.0	44,108	51,772	4.504	3.586	74	138	3.9	3.8
`no_lloyd(), odt(), perturb(), exude()`								
0.5	659,641	753,978	1.012	1.185	74	145	342.9	251.7
1.0	173,554	209,457	0.815	0.812	74	144	103.4	82.2
2.0	43,197	51,603	1.588	2.943	73	141	81.5	60.7
`lloyd(), no_odt(), perturb(), exude()`								
0.5	643,371	745,864	1.277	2.941	74	144	2533.7	1017.4
1.0	171,117	207,582	2.983	1.211	74	143	513.8	179.9
2.0	42,903	52,688	0.090	1.182	74	140	124.0	73.4
`lloyd(), odt(), perturb(), exude()`								
0.5	651,023	756,038	1.695	0.371	74	144	3571.6	956.0
1.0	173,512	212,914	2.009	3.388	74	143	468.8	202.7
2.0	44,357	53,192	2.223	2.799	74	138	138.4	111.8

- `facet_size` is an upper bound on the circumradii of the mesh tetrahedra; we set it to an ignored value.
- `lloyd(), odt(), perturb(), exude()` with the corresponding `no_` prefixes are available mesh optimization functions; the default usage is `no_lloyd(), no_odt(), perturb(), exude()`. We used four combinations specified in the table with all the default arguments.

In addition to the quantities measured in the previous experiments, we queried `subdomain_index` for each tetrahedron of the resulting mesh. Then we counted the total number of unique subdomain indexes and compared with the number of unique voxel labels in the image (75 for the abdominal atlas and 149 for the brain atlas).

Similar to Tetgen, mesh generation in CGAL consists of two phases, the construction of the initial mesh and its improvement as a postprocessing step. The first phase in both Tetgen and CGAL uses a variation of the Delaunay refinement approach which guarantees only a bound on the radius-edge ratio. The second phase, optimization, improves other mesh properties such as the dihedral angles. Using various combinations of optimization algorithms implemented in CGAL, we could not obtain minimum dihedral angles of 5° or more. The final number of tetrahedra produced by CGAL is significantly smaller than produced by LD, however, at the expense of occasionally not representing some of the tissues from the image. CGAL's processing time varies significantly, from much lower than that of LD to order of magnitude higher, depending on the selection of mesh optimization algorithms.

4. Summary. We presented a novel guaranteed quality and fidelity image-to-mesh conversion algorithm and its efficient sequential implementation. The algorithm preserves not only external boundaries, but also the boundaries between multiple tissues, which makes the resulting meshes suitable for FE simulations of multitissue regions with different physical tissue properties.

The algorithm and the implementation we presented are sequential. Our future work includes the development of the corresponding parallel algorithm and the code to increase the processing speed and the size of the images that can be handled. One stage of the algorithm, the distance transform, has already been parallelized [17]. However, according to Amdahl's law, to achieve good speedup, we need to parallelize the other stages as well. We also plan to address the smoothness of mesh boundaries in order to improve the accuracy of such simulations as blood flow.

Acknowledgments. We thank the anonymous reviewers for detailed comments which helped us improve the manuscript.

REFERENCES

[1] *Computational Geometry Algorithms Library*, http://www.cgal.org.

[2] D. Boltcheva, M. Yvinec, and J.-D. Boissonnat, *Mesh generation from 3d multi-material images*, in Proceedings of the 12th International Conference on Medical Image Computing and Computer-Assisted Intervention: Part II, Springer-Verlag, Berlin, 2009, pp. 283–290.

[3] S.-W. Cheng, T. K. Dey, H. Edelsbrunner, M. A. Facello, and S.-H. Teng, *Sliver exudation*, J. ACM, 47 (2000), pp. 883–904.

[4] L. P. Chew, *Guaranteed-quality Delaunay meshing in 3D*, in Proceedings of the 13th ACM Symposium on Computational Geometry, Nice, France, 1997, pp. 391–393.

[5] A. Fedorov and N. Chrisochoides, *Tetrahedral mesh generation for non-rigid registration of brain MRI: Analysis of the requirements and evaluation of solutions*, in Proceedings of the 17th International Meshing Roundtable, Pittsburgh, PA, 2008, Springer-Verlag, Berlin, pp. 55–72.

[6] P.-L. George and H. Borouchaki, *Delaunay Triangulation and Meshing. Application to Finite Elements*, Hermes, Paris, 1998.

[7] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, *Mesh optimization*, in Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques, New York, 1993, ACM, pp. 19–26.

[8] F. Labelle and J. R. Shewchuk, *Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles*, ACM Transactions on Graphics, 26 (2007), pp. 57.1–57.10.

[9] X.-Y. Li and S.-H. Teng, *Generating well-shaped Delaunay meshes in 3D*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, Washington, DC, 2001, pp. 28–37.

[10] Y. Liu, P. Foteinos, A. Chernikov, and N. Chrisochoides, *Multi-tissue mesh generation for brain images*, in Proceedings of the 19th International Meshing Roundtable, Chattanooga, TN, Springer-Verlag, Berlin, 2011, pp. 367–384.

[11] D. P. Luebke, *A developer's survey of polygonal simplification algorithms*, IEEE Comput. Graph. Appl., 21 (2001), pp. 24–35.

[12] C. Maurer, R. Qi, and V. Raghavan, *A linear time algorithm for computing exact Euclidean distance transforms of binary images in arbitrary dimensions*, IEEE Trans. Pattern Anal. Machine Intel., 25 (2003), pp. 265–270.

[13] R. Pajarola and J. Rossignac, *Compressed progressive meshes*, IEEE Trans. Visualization Comput. Graphics, 6 (2000), pp. 79–93.

[14] J.-P. Pons, F. Ségonne, J.-D. Boissonnat, L. Rineau, M. Yvinec, and R. Keriven, *High-quality consistent meshing of multi-label datasets*, in Proceedings of the 20th International Conference on Information Processing in Medical Imaging, 2007, Springer-Verlag, Berlin, pp. 198–210.

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

[16] H. Si, *Tetgen Version* 1.4.3, http://tetgen.berlios.de.

[17] R. Staubs, A. Fedorov, L. Linardakis, B. Dunton, and N. Chrisochoides, *Parallel n-dimensional exact signed Euclidean distance transform*, Insight J., 2006.

[18] I. Talos, M. Jakab, and R. Kikinis, *SPL Abdominal Atlas* 2010, http://www.spl.harvard.edu/publications/item/view/1918 (Oct. 2010).

[19] I. Talos, M. Jakab, R. Kikinis, and M. Shenton, *SPL-PNL Brain Atlas*, http://www.spl.harvard.edu/publications/item/view/1265 (March 2008).

[20] J. F. Thompson, B. K. Soni, and N. P. Weatherill, *Handbook of Grid Generation*, CRC Press, Boca Raton, FL, 1998.

[21] J. Tournois, R. Srinivasan, and P. Alliez, *Perturbing slivers in 3D Delaunay meshes*, in Proceedings of the 18th International Meshing Roundtable, B. W. Clark, ed., Springer-Verlag, Berlin, 2009, pp. 157–173.

[22] J. Yan, P. Shi, and D. Zhang, *Mesh simplification with hierarchical shape analysis and iterative edge contraction*, IEEE Trans. Visualization Comput. Graphics, 10 (2004), pp. 142–151.

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