SIAM J. SCI. COMPUT.
Vol. 35, No. 1, pp. A294–A326

TETRAHEDRALIZATION OF ISOSURFACES WITH GUARANTEED-QUALITY BY EDGE REARRANGEMENT (TIGER)*

SHAWN W. WALKER†

Abstract. We present a method for generating three-dimensional (3-D) unstructured tetrahedral meshes of solids whose boundary is a smooth surface. The method uses a background grid (body-centered-cubic (BCC) lattice) from which to build the final conforming 3-D mesh. The algorithm is fast and robust and provides *useful* guaranteed dihedral angle bounds for the output tetrahedra. The dihedral angles are bounded between 8.5° and 164.2°. If the lattice spacing is smaller than the "local feature size," then the dihedral angles are between 11.4° and 157.6° (cf. Labelle and Shewchuk [SIGGRAPH '07, ACM, New York, 2007]). The method is simple to implement and performs *no* extra refinement of the background grid. The most complicated mesh transformations are 4-4 edge flips. Moreover, the only parameter in the method is the BCC lattice spacing. If the surface has bounded curvature and if the background grid is sufficiently fine, then the boundary of the output mesh is guaranteed to be a geometrically and topologically accurate approximation of the solid surface. Applications of the method are in free boundary flows, modeling deformations, shape optimization, and anything that requires dynamic meshing, such as virtual surgery.

Key words. mesh generation, three dimensions, tetrahedra, dihedral angles, octahedra, isosurface, level set, front-tracking

AMS subject classifications. 65N50, 65D18, 68U05

DOI. 10.1137/120866075

1. Introduction.

Mesh generation is a classic problem in computer graphics, geometric modeling, and scientific computing. Meshes are used for representing and rendering surfaces, mechanical design, and enabling physical simulations. The finite element method (FEM) [8, 9, 17, 30] is a popular choice for computing numerical solutions of continuum level partial differential equations (PDEs) that model various physical phenomena, such as elastic deformations and free surface fluid flows. In this instance, the mesh provides a decomposition of the physical domain into elementary shapes (such as tetrahedra) through which a FEM can be built.

Using a FEM leads to finite dimensional representations (matrices) of the PDE model that must be solved to obtain the physical solution (e.g., deformation field). The accuracy of the FEM and condition number of the related matrices depend on the number of mesh elements as well as the mesh element "qualities" [3, 5, 34, 55]. More specifically, for tetrahedra, the dihedral angles must not be close to 0° and 180°. Naturally, these are desirable properties for any finite element mesh generator.

This paper develops a fast generation method for tetrahedral volume meshes. It is directly inspired by the method in [36] but is simpler to implement and the dihedral angle bounds are better if the mesh size is sufficiently small. Our method is applicable in any dynamic meshing situation, such as multiphase flows, mechanical deformations, or whenever a front-tracking approach is needed. It is numerically robust and offers theoretical guarantees on the dihedral angles that are actually useful, which is crucial to avoid "user intervention." Our method does not accommodate extra "surface

*Submitted to the journal's Methods and Algorithms for Scientific Computing section February 14, 2012; accepted for publication (in revised form) September 26, 2012; published electronically January 15, 2013. This work was supported by NSF grant DMS-1115636.
http://www.siam.org/journals/sisc/35-1/86607.html
†Department of Mathematics and Cneter for Computation and Technology, Louisiana State University, Baton Rouge, LA 70803-4918 (walker@math.lsu.edu).

constraints," such as edge or corner constraints; it applies only to three-dimensional (3-D) solids with smooth boundaries.

There has been a significant amount of work on tetrahedral mesh generation. Two notable free software programs are TetGen [56] and NetGen [53]. Some early work on Delaunay and octree methods can be found in [65, 25, 44, 4, 16, 6, 21] and more recently in [39, 37, 15, 14]. Some methods use *sliver exudation* to remove degenerate elements from the mesh [15, 20, 49].

Other methods take an optimization viewpoint [53, 24, 32, 41, 40, 23], while others [12, 13, 2] use a variational form to minimize the interpolation error by local remeshing. Some methods use specific tilings of 3-D space [62, 22] or marching cubes [38] or marching tetrahedra [43]. Another option for mesh generation uses implicit (level set) functions to create conforming meshes [46, 47, 45, 10, 50, 51, 63, 35, 36] as well as adaptive methods to create meshes adapted to the local feature size [7, 28, 33]. Some of these methods also include mesh smoothing operations (see [18, 48, 54, 64, 29] for more smoothing methods). An excellent detailed review of current meshing technology can be found in [35, 36].

Because of the wide variety of mesh generation methods available, it is worthwhile to ask what is *not* available. The following list describes several desirable properties for any mesh generator:

- Tetrahedralize a solid region defined by a continuous and *Lipschitz* closed surface (e.g., piecewise smooth with corners).
- Generate the mesh quickly, i.e., have an $O(n)$ algorithm, where n is some reasonable quantity such as the number of output vertices.
- Have theoretical guarantees on the "quality" of the output tetrahedra that are *practical* (e.g., minimum dihedral angles $\gg 1^\circ$; maximum dihedral angles $\ll 180^\circ$).
- Handle internal boundary constraints*.
- Not be complicated to implement*.
- Be parallelizable*.

To the best of the author's knowledge, no method exists that satisfies the above items simultaneously, even if the "starred" items are omitted. Indeed, most dihedral angle guarantees are less than 1°, which is not useful in practice. See [35, 36] for more information on current dihedral angle guarantees. For these reasons, mesh generation is a major bottleneck in industrial design and simulation, especially for dynamic meshing.

Although our method does not satisfy all items in the above list either, it does achieve the following:

- Tetrahedralizes an interior solid region defined by a continuous closed surface, with *bounded* curvature (i.e., a $C^{1,1}$ surface).
- Is a practically fast algorithm (see section 4.3).
- Has guarantees on the dihedral angles of the output tetrahedra:

no restrictions:	restriction: $c \le R_{\mathrm{m}}/1.1$
minimum dihedral angle $> 8.54^\circ$,	minimum dihedral angle $> 11.47^\circ$,
maximum dihedral angle $< 164.18^\circ$,	maximum dihedral angle $< 157.59^\circ$,

  where c is the lattice spacing (background mesh size) and R_{m} is (essentially) the local feature size of the solid's surface (see Theorem 2).
- Not complicated to implement.
- Can mostly be parallelized.

The algorithm is called "Tetrahedralization of Isosurfaces, with Guaranteed-quality, by Edge Rearrangement" or TIGER. In section 2, we note some basic definitions and material. Section 3 gives a high-level description of the main algorithm, while section 4 describes Version 1 of the algorithm (simplest version). Section 5 summarizes the meshing guarantees of the method and the dihedral angle bounds for Version 1. Section 6 presents a minor modification (Version 2) that leads to better dihedral angles provided the background mesh is sufficiently fine. We present meshing results in section 7 and conclude with a discussion in section 8 on parallelization and heuristics one can use to improve the meshes created by our method. Appendix A describes our technique for proving the dihedral angle bounds.

2. Preliminaries. The method uses a background mesh from which to construct an output mesh that conforms to an object's boundary. The idea is to deform the background mesh (i.e., move the vertices) such that there is a set of mesh edges that conform to the object's surface. But a mesh is inherently anisotropic, meaning the mesh edges will *not* be favorably aligned with the surface in all instances. Therefore, the method makes simple local topological transformations of the background mesh (i.e., 4-4 edge-flips [31]) in order to ensure the edges are favorably aligned.

2.1. Background mesh. Let $L_{\mathrm{A}} := \mathbb{Z}^3$ be a cubic lattice, i.e., the set of points in $\mathbb{R}^3$ whose coordinates are integers. We denote the (uniformly) scaled cubic lattice by $L_{\mathrm{A}}(c)$, where all three dimensions are scaled by $c > 0$ (c is the lattice spacing). Similarly, let $L_{\mathrm{B}} := L_{\mathrm{A}} + \left(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right)$, i.e., each point is shifted by the vector $\left(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right)$; we can also scale it: $L_{\mathrm{B}}(c)$.

Consider the (scaled) body-centered-cubic (BCC) lattice:

$$\mathrm{BCC}(c) := L_{\mathrm{A}}(c) \cup L_{\mathrm{B}}(c). \tag{1}$$

Let $\mathcal{T}$ be the Delaunay triangulation of the point set $\mathrm{BCC}(c)$ [58]. For mesh notation, let the set of vertices of $\mathcal{T}$ (lattice points) be denoted by $\mathcal{V}$ and edges of $\mathcal{T}$ denoted by $\mathcal{E}$ (see Figure 1). The edges decompose into two types of edges, long and short, each type having the exact same (Euclidean) length. For the unscaled BCC lattice, the lengths are $L_{\mathrm{lg}} := 1$ and $L_{\mathrm{st}} := \sqrt{3}/2$. Naturally, the *scaled* BCC lattice has long and short edge lengths cL_{lg} and cL_{st} (respectively).

A crucial part of the method uses an octahedral view of the BCC mesh, which has recently been considered for generating surface meshes [11, 61]. In the BCC context, an *octahedron* is a set of four tetrahedra that all share a common longest edge (Figure 2). The common longest edge of an octahedron is called the *spine* of the octahedron. Basic considerations lead to the following proposition.

PROPOSITION 1. *The following properties are true for the BCC tetrahedral mesh* $\mathcal{T}$*.*

1. *Every edge of the* L_{A} *lattice is the spine of an octahedron* $O \subset \mathcal{T}$*; likewise for* L_{B}*.*
2. *The set of BCC tetrahedra* $\mathcal{T}$ *can be decomposed into a disjoint union of octahedra whose spines are edges in* L_{A}*. A different decomposition is given if* L_{B} *is used.*
3. *Every* $T \in \mathcal{T}$*, belongs to two distinct (but overlapping) octahedra, i.e., there exist octahedra* O_1*,* O_2 *such that* $O_1 \cap O_2 = T$ *(see Figure* 2 (b)*).*

Proof. 1. Let E be a long edge of L_{A}, and let $\{Q_1, Q_2, Q_3, Q_4\}$ be the four cubes in L_{A} that all contain E as an edge (see Figure 1). The octahedron, whose spine is E, is defined by the two end point vertices of E and the four L_{B} vertices that lie at the center of the cubes.

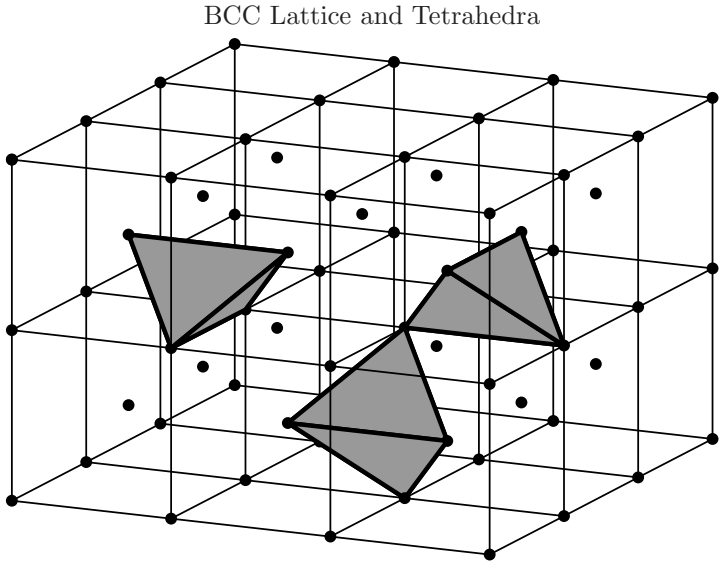
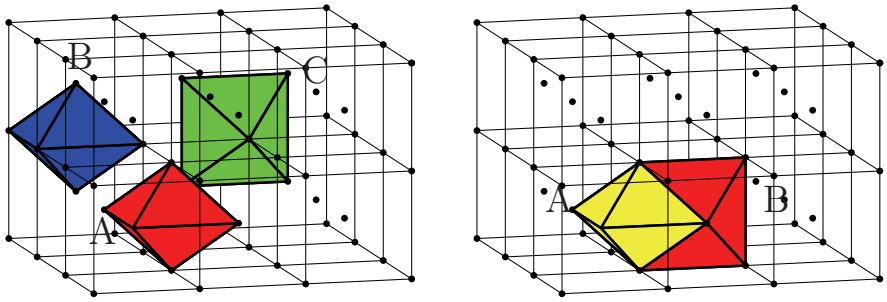


FIG. 1. *Highlighted tetrahedra in a BCC lattice. The tetrahedra are generated by the Delaunay triangulation of the point set* BCC(c) *for any* $c > 0$. *Each tetrahedron has two long edges (equal length): one belonging to lattice* L_{A}, *the other to* L_{B}. *The remaining four short edges (equal length) connect* L_{A} *vertices to* L_{B} *vertices (i.e., they bridge the two lattices). All of the tetrahedra are self-similar, differing only by rotations and translations.*

2. Given two edges E_1, E_2 in L_{A}, one can show that the interior of their corresponding octahedra do not overlap. Moreover, given any point p in $\mathbb{R}^3$, it is obvious how to find the enclosing cube Q (assumed to be a closed set) in L_{A}. Next, find the closest edge E of Q and let O be the octahedron whose spine is E. One can verify that p is contained in the closure of O.

3. Let $T \in \mathcal{T}$. Clearly, T has two long edges E_1, E_2. Let O_1, O_2 be the corresponding octahedra. It is straightforward to check that $O_1 \cap O_2 = T$. ∎



(a) Distinct octahedra. (b) Overlapping octahedra.

FIG. 2. *Examples of octahedra contained in the BCC tetrahedral mesh. In* (a), *octahedra A and C belong to the same lattice* L_{A}; *the spine of octahedron B belongs to lattice* L_{B}. *Moreover, the spine of C is orthogonal to the spines of A and B. In* (b), *the spines of A and B belong to lattices* L_{A} *and* L_{B} *(respectively), and the intersection of A and B is a single tetrahedron (not shown).*

2.2. Physical domain. Consider a bounded domain $\Omega \subset \mathbb{R}^3$. Let $\phi : \mathbb{R}^3 \to \mathbb{R}$ be a continuous level set function such that $\phi > 0$ inside Ω and $\phi < 0$ outside Ω; thus, $\Gamma \equiv \partial\Omega = \{\mathbf{x} : \phi(\mathbf{x}) = 0\}$. We want to decompose $\mathbb{R}^3$, using the BCC background grid, into a mesh of tetrahedra that approximates Ω well. For now, we require only that Γ be continuous (i.e., a C^0 surface). Note: the output mesh from our algorithm will not (in general) respect any sharp edges/corners of Γ.

2.3. Intersections with Γ. We define a *cut point* to be a point $\mathbf{c} \in \mathbb{R}^3$ such that $\mathbf{c} \in E$, for some edge E in $\mathcal{E}$, and $\phi(\mathbf{c}) = 0$. If an edge $E \in \mathcal{E}$ has end points v_1, v_2 such that $\text{sgn}(\phi(v_1)) \neq \text{sgn}(\phi(v_2))$, then there exists at least one cut point on E and we call E a *cut edge*. Note that we never have to consider the situation where $\text{sgn}(\phi(v)) = 0$ because we assume the following definition of the *sign* function:

$$\begin{aligned} \text{sgn}(s) &= +1, \qquad s \geq 0, \\ \text{sgn}(s) &= -1, \qquad s < 0. \end{aligned} \tag{2}$$

Hence, if $\phi(v) = 0$ (i.e., v is a point on the surface), then $\text{sgn}(\phi(v)) = 1$. If a tetrahedron $T \in \mathcal{T}$ contains a cut edge, then we call it a *cut tetrahedron*.

Remark 1. The following fact is taken from the marching tetrahedra algorithm [43] (which is a variant of the marching cubes algorithm [38]).

- For any $T \in \mathcal{T}$, only the following cases are possible: (a) all the vertices have the same sign (i.e., level set values are either all positive or negative), (b) one vertex has a different sign than the other three, (c) two vertices are negative and two vertices are positive. The last two cases correspond to a cut tetrahedron (See Figure 3). Therefore, a cut tetrahedron must have either three or four cut edges (of its six edges), and their arrangement must match the pattern in Figure 3.

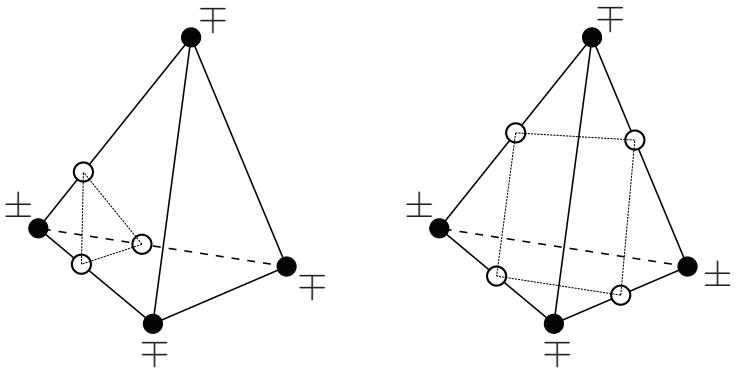


FIG. 3. *The two classic examples of marching tetrahedra* [43]. *The signs of the level set function are given at each vertex. If a tetrahedron is cut, then it must have one of these two configurations (after accounting for symmetry). Of course, there are also the cases (not shown) when the level set function is zero at a vertex; because of the sign convention* (2)*, these are special cases of the two cases shown here.*

Let $\mathcal{E}_c \subset \mathcal{E}$ be the set of all cut edges. Furthermore, associate a *single* cut point $\mathbf{c}_E \in E$ with each $E \in \mathcal{E}_c$. If a cut edge has more than one cut point, then choose any one of the cut points to associate with the cut edge. Moreover, let $\mathcal{T}_c \subset \mathcal{T}$ be the set of all cut tetrahedra. So $\mathcal{T}_c$ consists of a "shell" of tetrahedra near the surface Γ. It does not necessarily enclose Γ because one can construct examples where a tetrahedron has all positive vertices, but pairs of cut points lie along its edges.

2.4. Distance to cut points. We define a special "distance" function. Let $v \in \mathcal{V}$ be given and let $E \in \mathcal{E}$ be an edge that has v as an end point. Then, setting $\mathbf{x}_v$ to be the position coordinate of v and assuming E is a cut edge, with associated cut point $\mathbf{c}_E$, define

$$d_{\text{cut}}(v; E) := \frac{|\mathbf{x}_v - \mathbf{c}_E|}{|E|}, \tag{3}$$

where $|E|$ is the length of E, and the norm $|\cdot|$ is the (Euclidean) length of a vector. If E *does not* contain a cut point, then $d_{\text{cut}}(v; E) := +\infty$. Essentially, (3) is a normalized distance measure. Note that $d_{\text{cut}}(v; E)$ is not defined if E does not contain v as an end point.

Let $St(v; \mathcal{E}) = \{E \in \mathcal{E} : v \text{ is an end point of } E\}$, i.e., the star of edges emanating from v; sometimes we abbreviate by just writing $St(v)$. The following minimization is an important part of the method:

$$d_{\min}(v) := \min_{E \in St(v)} d_{\text{cut}}(v; E). \tag{4}$$

The following proposition is basic.

PROPOSITION 2. *Let $E \in \mathcal{E}_c$ and let v_1, v_2 be the end point vertices of E. Then*

$$\min\left(d_{\text{cut}}(v_1; E), d_{\text{cut}}(v_2; E)\right) \leq 0.5. \tag{5}$$

2.5. Manifold vertices. An important part of the algorithm is to identify a subset of mesh vertices that approximates the surface Γ. We refer to this subset as the set of *manifold vertices*; i.e., this set of vertices will lie on the boundary of the output mesh. We also allow manifold vertices to modify the "state" of adjoining cut edges. In other words, any cut edge where at least one end point is a manifold vertex is considered to be inactive or *suppressed*. Otherwise, it is *active*.

Manifold vertices are chosen to prevent an interior vertex from being connected to an exterior vertex through a single mesh edge. Note that manifold vertices are *not* considered to be interior or exterior to Γ because eventually their coordinate positions will be moved to the surface. Hence, the only candidate vertices for manifold status are the end points of cut edges.

It is possible that the manifold vertex labeling could lead to a tetrahedron that has all four vertices labeled manifold. We call these tetrahedra *ambiguous* because it is not clear whether they are inside or outside Γ. We also call an octahedron ambiguous if all four tetrahedra in the octahedron are ambiguous. The algorithm handles ambiguous tetrahedra by either deleting them, or using heuristics (section 8.3). When all manifold vertices are moved to the surface and all topological transformations are done, the interior and exterior tetrahedra can be identified (step 8 of Algorithm 1).

3. Main algorithm. Given a bounded domain Ω, Algorithm 1 outputs a mesh of the *interior* of Ω (where ϕ is positive). The main idea of the algorithm is to label vertices as manifold until there are no more active cut short edges. All remaining active long edges are accounted for by simple topological transformations of the background BCC mesh. The main procedure is separated into several subalgorithms that are described in subsequent sections.

4. Algorithm: Version 1. Version 1 of our algorithm is the simplest (i.e., we ignore step 7 in Algorithm 1). Note: comments within all algorithm descriptions are denoted with "//."

Algorithm 1 Mesh Generation

1. Find all cut edges $\mathcal{E}_c$ and their associated cut points; i.e., for every $E \in \mathcal{E}_c$ there is a distinct $\mathbf{c}_E \in E$.
2. Initialize the **active** set of short and long cut edges $\mathcal{E}_{\text{st}}^+$, $\mathcal{E}_{\text{lg}}^+$ (respectively) such that $\mathcal{E}_{\text{st}}^+ \cup \mathcal{E}_{\text{lg}}^+ = \mathcal{E}_c$.
3. Execute *Initial Vertex Labeling*, Algorithm 2. This returns $\mathcal{E}_{\text{st}}^+$ as empty and a set of labeled manifold vertices along with their associated destination points. $\mathcal{E}_{\text{lg}}^+$ is also modified.
4. Execute *Back-Labeling*, Algorithm 3. This *changes* the set of labeled manifold vertices such that no manifold vertex moves to a destination point along an edge toward another manifold vertex. Moreover, it ensures $\mathcal{E}_{\text{st}}^+$ is *still empty*, but modifies the set $\mathcal{E}_{\text{lg}}^+$.
5. Move each manifold vertex to its destination point. This generates new vertex coordinates $\mathcal{V}_{\text{new}}$.
6. Execute *Active Long Edge-Flips*, Algorithm 4. This generates a new triangulation $\mathcal{T}_{\text{new}}$.
7. Additional Processing Step. If Version 2 is used, then Execute *Additional Edge-Flips*, Algorithm 5. This further modifies the triangulation $\mathcal{T}_{\text{new}}$.
8. For each $T \in (\mathcal{T}_{\text{new}}, \mathcal{V}_{\text{new}})$, label T as an interior (exterior) tetrahedron if ϕ is positive (negative) on all of T's nonmanifold vertices. If a tetrahedron is ambiguous, then mark it as an exterior tetrahedron.

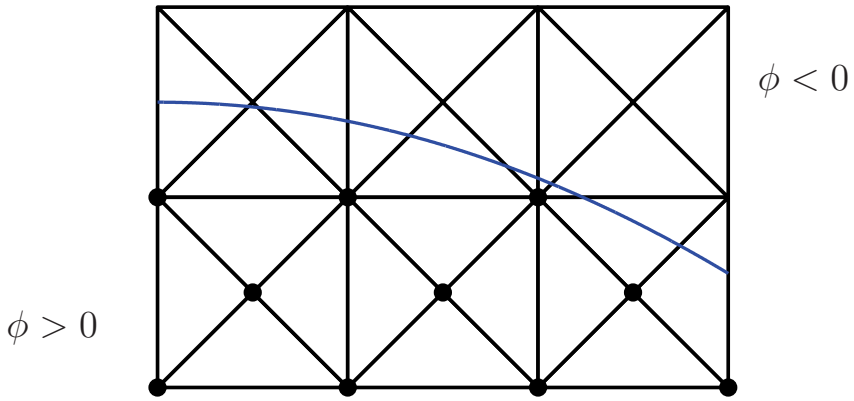
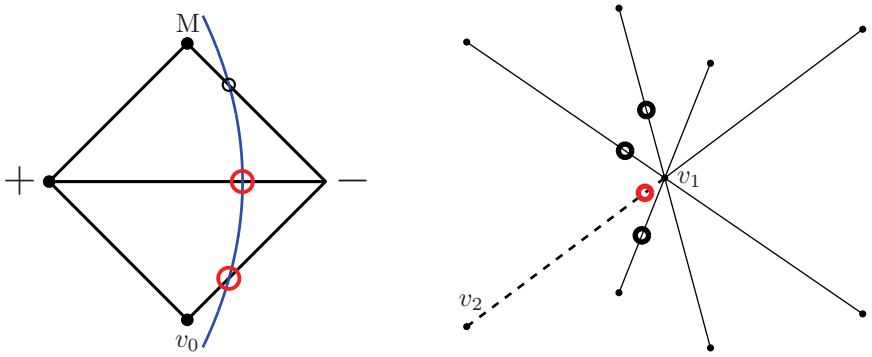


FIG. 4. *Example of step* 1 *of Algorithm* 1 *(in two dimensions). The blue curve is the zero level set of* ϕ *(i.e., the surface). The bold points of the mesh are interior vertices; exterior vertices are not bold. A cut edge of the mesh has one vertex as interior and the other vertex as exterior.*

4.1. Identify interior vertices and cut edges. Step 1 identifies which vertices of the BCC mesh are inside the object by direct interpolation of the level set function ϕ using the sign convention (2). See Figure 4 for an illustration. Step 2 initializes data structures to keep track of which cut edges are active and suppressed while the algorithm runs.

4.2. Initial vertex labeling. In step 3, we must choose a subset of mesh vertices to be manifold vertices so that we can apply straightforward processing to move the manifold vertices in order to ensure the mesh conforms to Γ. A simple choice is to take



(a) A manifold labeling example (2-D). (b) Computing d_{cut} (3-D).

FIG. 5. *Examples of labeling manifold vertices.* (a) *shows a two-dimensional (2-D) mesh during the labeling procedure in Algorithm* 2. *The blue curve is the level set surface* $\{\phi = 0\}$. *One vertex has been labeled manifold (M) which causes the adjoining cut edge to be suppressed (note small black circle). An active* short *cut edge still remains (note lower large red circle) with an interior end point denoted* v_0. *Algorithm* 2 *will label* v_0 *manifold, because it is closer to an active cut point than the* − *end point, and suppress the lower short cut edge. The remaining active* long *cut edge (horizontal edge) is handled separately by an edge-flip (see section* 4.4 *in the* 3*-D case).* (b) *shows eight short edges of the* BCC *mesh that adjoin vertex* v_1, *where four of the edges are active, with cut points denoted by circles. The current active short cut edge is denoted by a dashed line segment. We compute* d_1, d_2 *by finding* $St(v_1; \mathcal{E}^+_{\text{st}})$, $St(v_2; \mathcal{E}^+_{\text{st}})$ *and computing the minimization in* (6). *Since* $d_1 < d_2$, *we label* v_1 *manifold. Furthermore, we assign the closest cut point (red circle on dashed line segment) as the destination point of* v_1. *We then suppress* all *active (long and short) edges adjoining* v_1.

the inner (or outer) free boundary of the cut tetrahedra $\mathcal{T}_c$. However, this can be a bad choice because it could lead to highly squashed tetrahedra in the general case. This would require employing mesh smoothing and nontrivial topological transformations for nodes away from the manifold in order to maintain a decent element quality. Unfortunately, this requires more computation, and it would make obtaining bounds on the dihedral angles impossible.

Alternatively, Algorithm 2 provides a simple, inexpensive way to choose the manifold vertices and where to place them on the surface Γ. See Figure 5 for an illustration.

Algorithm 2 Initial Vertex Labeling

while $\mathcal{E}^+_{\text{st}} \neq \emptyset$ **do**

Choose an active cut edge E in $\mathcal{E}^+_{\text{st}}$.

Let v_1, v_2 be the end point vertices of E; compute

$$E_i = \arg \min_{E \in St(v_i; \mathcal{E}^+_{\text{st}})} d_{\text{cut}}(v_i; E), \quad d_i = d_{\text{cut}}(v_i; E_i) \quad \text{for } i = 1, 2. \tag{6}$$

If $d_i < d_j$, then label v_i a *manifold vertex* and choose $\mathbf{c}_{E_i}$ to be its *destination* point, where $i = 1$, $j = 2$ or $i = 2$, $j = 1$. If $d_1 = d_2 = \frac{1}{2}$, then choose the point that adjoins the most active cut edges. If they are the same, then choose v_1.

Remove all active edges from $\mathcal{E}^+_{\text{st}}$ and $\mathcal{E}^+_{\text{lg}}$ that adjoin the newly chosen manifold vertex, i.e., suppress them.

end while

Basic considerations yield the following result.

PROPOSITION 3. *After step* 3 *or step* 4 *of Algorithm* 1 *has completed, every manifold vertex has its own* unique *destination cut point that it will move to. Furthermore, the largest distance of all manifold vertices from their destination cut points is bounded by half the corresponding short edge length* cL_{st} *(recall Proposition* 2*), where* c *is the lattice spacing. Moreover, each manifold BCC vertex is constrained to move along* eight distinct directions*, which are defined by the unit vectors:*

$$\mathbf{d}_i = (\pm 1, \pm 1, \pm 1)/\sqrt{3} \quad \textit{for} \quad i = 1, 2, \ldots, 8. \tag{7}$$

4.3. Back-labeling procedure. After the initial labeling process has finished, some manifold vertices may move along short edges (of the BCC mesh) toward other manifold vertices (see Figure 6). These situations can lead to flattened tetrahedra, so they must be eliminated. Algorithm 3 achieves this by "back-labeling."

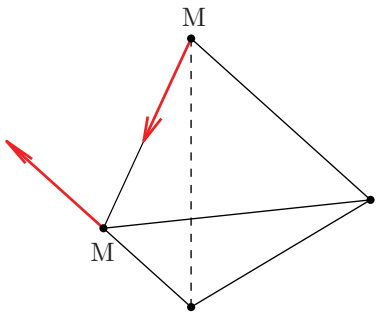


FIG. 6. *Tetrahedron with bad initial manifold labeling. This is a situation where the initial labeling procedure results in a manifold vertex (M) moving (along red arrow) to a destination point along an edge toward another manifold vertex. Back-labeling removes these situations.*

In order to better describe the algorithm, define the following subsets of vertices of $\mathcal{V}$:

(8)

$$\begin{aligned}
M_{\text{A}} &= \{v \in L_{\text{A}} : v \text{ is manifold}\}, \qquad M_{\text{B}} = \{v \in L_{\text{B}} : v \text{ is manifold}\},\\
M_{\text{A}\to\text{B}} &= \{v \in M_{\text{A}} : (v,z) = E, \text{ where } v \text{ moves along } E \text{ toward } z, \text{ and } z \in M_{\text{B}}\},\\
M_{\text{B}\to\text{A}} &= \{v \in M_{\text{B}} : (v,z) = E, \text{ where } v \text{ moves along } E \text{ toward } z, \text{ and } z \in M_{\text{A}}\},\\
M_{\text{A}\to 0} &= M_{\text{A}} \setminus M_{\text{A}\to\text{B}}, \qquad M_{\text{B}\to 0} = M_{\text{B}} \setminus M_{\text{B}\to\text{A}}.
\end{aligned}$$

Clearly, we have the following disjoint unions: $M_{\text{A}} = M_{\text{A}\to 0} \cup M_{\text{A}\to\text{B}}$, $M_{\text{B}} = M_{\text{B}\to 0} \cup M_{\text{B}\to\text{A}}$.

The result of Algorithm 3 is to modify the labeling so that $M_{\text{A}\to\text{B}} = M_{\text{B}\to\text{A}} = \emptyset$, i.e., that no manifold vertex moves along an edge toward another manifold vertex (see Figure 7). The following lemma verifies this.

LEMMA 1. *Suppose we have an initial labeling; i.e., steps* 1–3 *of Algorithm* 1 *have executed. Then the back-labeling procedure (Algorithm* 3*) is guaranteed to terminate with* $M_{\text{A}\to\text{B}} = M_{\text{B}\to\text{A}} = \emptyset$.

Proof. Note that since Ω is bounded, we need only to consider a finite subset of $(\mathcal{T}, \mathcal{V})$. Suppose $M_{\text{A}\to\text{B}} \neq \emptyset$. Then one pass through the *first* while loop of the back-labeling procedure chooses a $v \in M_{\text{A}\to\text{B}}$, removes v from $M_{\text{A}\to\text{B}}$, and results in one of the following three possible outcomes:

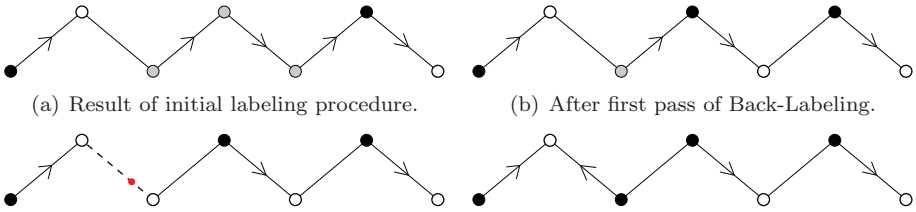
Algorithm 3 Back-Labeling Procedure

// Ensure the set $M_{\mathrm{A}\to\mathrm{B}} = \emptyset$.
while $M_{\mathrm{A}\to\mathrm{B}} \neq \emptyset$ **do**
  Choose any $v \in M_{\mathrm{A}\to\mathrm{B}}$ and remove v from $M_{\mathrm{A}\to\mathrm{B}} \subset M_{\mathrm{A}}$, i.e., unlabel it; this *may* reactivate some cut short and cut long edges.
  Recompute the active cut (short) edge list $\mathcal{E}^+_{\mathrm{st}}$, and modify the active cut (long) edge list $\mathcal{E}^+_{\mathrm{lg}}$.
  if $\mathcal{E}^+_{\mathrm{st}} \neq \emptyset$ **then**
    Compute the *candidate* manifold vertices for each $E \in \mathcal{E}^+_{\mathrm{st}}$ (independently) using (6), but do not suppress any cut edges yet.
    // Note that since E is active, the end point opposite to v *cannot* be a manifold vertex.
    Let v_E be the candidate manifold vertex for each $E \in \mathcal{E}^+_{\mathrm{st}}$.
    if $v = v_E$ for some $E \in \mathcal{E}^+_{\mathrm{st}}$ **then**
      Choose v to be manifold, and append v to $M_{\mathrm{A}\to 0}$.
      // $\#(M_{\mathrm{A}})$ remains the same; $\#(M_{\mathrm{A}\to\mathrm{B}})$ decreases by 1.
      // The sets $M_{\mathrm{B}\to 0}$, $M_{\mathrm{B}\to\mathrm{A}}$ remain unchanged.
    else
      // PROPAGATION
      Make all $\{v_E\}$ manifold and append to $M_{\mathrm{B}\to 0}$. This may cause some vertices to move from $M_{\mathrm{A}\to 0}$ to $M_{\mathrm{A}\to\mathrm{B}}$.
      // $\#(M_{\mathrm{A}})$ decreases by 1; $\#(M_{\mathrm{A}\to\mathrm{B}})$ *could* increase by (at most) $\#(\{v_E\})-1$.
      // The set $M_{\mathrm{B}\to 0}$ increases in size by $\#(\{v_E\})$; the set $M_{\mathrm{B}\to\mathrm{A}}$ *may* lose some elements, but it definitely will not gain any. This is because there may have been L_{B} manifold vertices moving toward v.
    end if
    // Note: for either case, all cut short edges become suppressed, i.e., $\mathcal{E}^+_{\mathrm{st}} = \emptyset$, and $\mathcal{E}^+_{\mathrm{lg}}$ is modified.
  else
    // $\#(M_{\mathrm{A}})$ decreases by 1; $\#(M_{\mathrm{A}\to\mathrm{B}})$ decreases by 1.
    // The set $M_{\mathrm{B}\to\mathrm{A}}$ *may* lose some elements, but it definitely will not gain any.
  end if
end while
// Ensure the set $M_{\mathrm{B}\to\mathrm{A}} = \emptyset$. The procedure is analogous to the previous one.

- v becomes a nonmanifold vertex, and no other vertices in $\mathcal{V}$ are set to manifold status; or
- v is appended to $M_{\mathrm{A}\to 0}$, and no other vertices in $\mathcal{V}$ are set to manifold status; or
- v becomes nonmanifold, but a new set of vertices $\{v_i\}$ is added to $M_{\mathrm{B}\to 0}$, which move toward v. This *may* cause a set of vertices $\{\tilde{v}_i\} \subset M_{\mathrm{A}\to 0}$ to be removed from $M_{\mathrm{A}\to 0}$ but appended to $M_{\mathrm{A}\to\mathrm{B}}$.

Let $\#(S)$ denote the number of elements in any set S. Let $a_0 = \#(M_{\mathrm{A}})$ before the back-labeling procedure runs, and define $a_k = \#(M^k_{\mathrm{A}})$, where M^k_{A} is the state of the set M_{A} after the kth pass. Clearly, from the definition of Algorithm 3, $\{a_k\}$ is a monotonically decreasing sequence of nonnegative integers.

Suppose the procedure does not terminate; i.e., the condition of the *while loop* is



(a) Result of initial labeling procedure. (b) After first pass of Back-Labeling.

(c) *During* second pass of Back-Labeling. (d) After second pass of Back-Labeling.

FIG. 7. *2-D illustration of Back-Labeling Procedure (Algorithm* 3*). Manifold vertices are denoted by a black or shaded circle; unlabeled vertices are white circles. A subset of short edges of the background mesh are shown with edge arrows indicating how the "tail" manifold vertex moves.* (a) *shows the result of the initial labeling procedure (Algorithm* 2*) where three manifold vertices (shaded circles) move toward other manifold vertices.* (b) *shows the "labeling state" after the first pass of the while loop in Algorithm* 3*. An offending manifold vertex was* unlabeled *without reactivating a cut edge. However, there still remains one manifold vertex moving toward another manifold vertex.* (c) *shows the "labeling state"* during *the second pass. The offending vertex is unlabeled which reactivates a cut edge (denoted by dashed edge). The red dot denotes the location of a cut point on the reactivated edge.* (d) *After the second pass completes, the result is that the vertex is relabeled manifold but directed along a different edge, i.e., toward a different cut point. The final state has no manifold vertices moving toward other manifold vertices; see Lemma* 1*.*

always true. Then $\{a_k\}$ is an infinite sequence, and there exists an $N > 0$ such that $a_k = p$ for all $k > N$ for some nonnegative integer p. But if a_k remains *constant* for $k > N$, then $\#(M^{k+1}_{\mathrm{A}\to\mathrm{B}}) = \#(M^{k}_{\mathrm{A}\to\mathrm{B}}) - 1$ for all $k > N$. Thus, there is a finite $J > N$ such that $\#(M^{J}_{\mathrm{A}\to\mathrm{B}}) = 0$, so $M^{J}_{\mathrm{A}\to\mathrm{B}} = \emptyset$. But this contradicts the condition of the while loop always being true. Hence, the first part of Algorithm 3 must terminate.

Next, consider the *second* while loop, and note that no new elements were added to the set $M_{\mathrm{B}\to\mathrm{A}}$ during the first loop. By the same argument, we have that the second loop also terminates. Furthermore, the set $M_{\mathrm{A}\to\mathrm{B}}$ is unaffected by the second loop because it is empty when the second loop begins and no new elements can be added to it (similar to the previous case for the set $M_{\mathrm{B}\to\mathrm{A}}$). Therefore, we obtain the assertion. ∎

Remark 2. Important property of the back-labeling procedure: once it completes, it is *not* possible to remove any of the manifold vertices from manifold status without reactivating at least one cut short edge (i.e., the one that the manifold vertex is moving along).

For implementation purposes, it is not necessary to keep track of the sets $M_{\mathrm{A}\to 0}$, $M_{\mathrm{B}\to 0}$, $M_{\mathrm{A}\to\mathrm{B}}$, $M_{\mathrm{B}\to\mathrm{A}}$. These were introduced only for the proof of Lemma 1. One can also interleave modifying $M_{\mathrm{A}\to\mathrm{B}}$ and $M_{\mathrm{B}\to\mathrm{A}}$; it is not necessary to modify L_{A} vertices first followed by L_{B} vertices. Note that an "ordered warping" (ordered movement) approach was an option in [36], except they could not guarantee it would terminate. This seems to be an advantage of our method, which considers *only* the short cut edges when labeling vertices.

Remark 3 (parallelization issue). The back-labeling procedure presents some difficulties for a parallel implementation, since a chain of dependencies may need to be resolved in order to guarantee that $M_{\mathrm{A}\to\mathrm{B}} = M_{\mathrm{B}\to\mathrm{A}} = \emptyset$ (recall the "// PROPAGATION" line). From our experience, the back-labeling execution time is not significant (sometimes it does not even execute). Usually, the initial vertex labeling gives an adequate labeling with only a few vertices that require modification without any far-reaching propagation effects. The main purpose of back-labeling is to eliminate cases

that could lead to flattened tetrahedra. It also has the advantage of reducing the number of ambiguous tetrahedra.

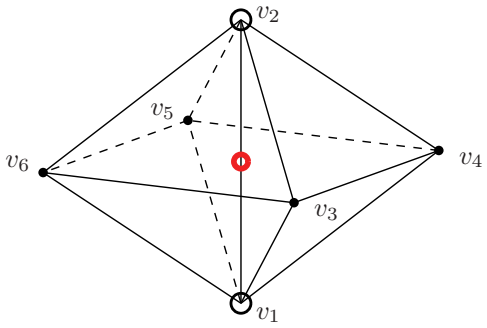


FIG. 8. *Remaining active cut long edge, which is the spine of an octahedron. The vertices* v_1, v_2 *are* not *labeled manifold, and edge* (v_1, v_2) *is cut at the middle, while* v_3, v_4, v_5, v_6 *are all labeled manifold. This is the only possible configuration for an active cut long edge (see Proposition* 4*).*

4.4. Flipping active long edges. In general, after all cut short edges of the mesh are suppressed, there will be some remaining cut *long* edges that are still active (see Figure 8). The following proposition says there is only one possible configuration for an active cut long edge.

PROPOSITION 4. *Suppose all cut short edges of the mesh are suppressed, meaning that every cut short edge has one of its end points labeled manifold. Suppose there is a cut* long *edge* E *with neither end point labeled manifold (i.e., it is active). Consider the octahedron* O *whose spine is* E*, and assume the vertices are numbered as shown in Figure* 8*. Then the vertices* $\{v_3, v_4, v_5, v_6\}$ *must all be labeled manifold.*

Proof. Suppose that one of the vertices $\{v_3, v_4, v_5, v_6\}$ is not labeled manifold (say v_3). Since neither v_1 or v_2 are labeled manifold, then the short edges (v_2, v_3) and (v_1, v_3) must *not* be cut (by hypothesis). However, all four tetrahedra of O must be cut (because the spine E is cut), and there are only two possible cut edge configurations for each tetrahedron (recall Figure 3). Therefore, it is not possible for E to be cut and *both* (v_2, v_3) and (v_1, v_3) to not be cut. This can be seen by accounting for all possible cut configurations.

So either (v_2, v_3) or (v_1, v_3) must be cut. But v_1, v_2, and v_3 are not manifold, which means there is a cut short edge with neither end point labeled manifold. But this contradicts the fact that all cut short edges are suppressed. Hence, v_3 must be labeled manifold. The same argument holds for v_4, v_5, and v_6. ∎

Remark 4. The remaining active (cut) long edges must be suppressed in order to have a valid output mesh. One option is to label one of the end points of the active long edge manifold. For example, label v_2 in Figure 8 as manifold. But this requires v_2 to move toward the cut point on the spine of the octahedron or toward one of the manifold vertices on the outer ring of the octahedron. Both choices lead to bad dihedral angle bounds. It is more advantageous to instead *flip* the spine of the octahedron.

The above considerations suggest that we *delete* the cut long edge (i.e., (v_1, v_2) in Figure 8) and insert a new edge: either (v_3, v_5) or (v_4, v_6). The tetrahedral con-

nectivity for each octahedral "slice" is as follows:

$$
\begin{array}{lll}
\text{slice S } (v_1\text{-}v_2 \text{ edge}): & \text{slice A } (v_3\text{-}v_5 \text{ edge}): \quad \text{or} & \text{slice B } (v_4\text{-}v_6 \text{ edge}): \\
T_1 = [v_1, v_3, v_2, v_6] \quad \Rightarrow & \widetilde{T}_1 = [v_3, v_4, v_5, v_2] & \widetilde{T}_1 = [v_6, v_3, v_4, v_2] \\
T_2 = [v_1, v_4, v_2, v_3] \quad \Rightarrow & \widetilde{T}_2 = [v_3, v_5, v_6, v_2] & \widetilde{T}_2 = [v_6, v_4, v_5, v_2] \\
T_3 = [v_1, v_5, v_2, v_4] \quad \Rightarrow & \widetilde{T}_3 = [v_3, v_5, v_4, v_1] & \widetilde{T}_3 = [v_6, v_4, v_3, v_1] \\
T_4 = [v_1, v_6, v_2, v_5] \quad \Rightarrow & \widetilde{T}_4 = [v_3, v_6, v_5, v_1] & \widetilde{T}_4 = [v_6, v_5, v_4, v_1]
\end{array} \tag{9}
$$

where slice S is the original (standard) tetrahedral configuration of the octahedron, and slices A and B are the two options (see Figure 9).

Algorithm 4 Flip Active Long Edges

Recall that $\mathcal{E}_{\text{lg}}^{+}$ is the set of (remaining) active cut long edges.
for $E \in \mathcal{E}_{\text{lg}}^{+}$ **do**
Get the octahedral cell that has E as its spine (i.e., get the four tetrahedra $\{T_1, T_2, T_3, T_4\}$ that have E as an edge).
Put the six vertices $\{v_i\}_{i=1}^{6}$ of the octahedron into a well-defined order (see Figure 8).
// We want to replace $\{T_i\}_{i=1}^{4}$ by $\{\widetilde{T}_i\}_{i=1}^{4}$ using either slice A or slice B; see equation (9).
// Note: for either slice, each $\widetilde{T}_i$ has exactly one nonmanifold vertex (v_1 or v_2). Therefore, $\{\widetilde{T}_1, \widetilde{T}_2\}$ is inside the surface Γ and $\{\widetilde{T}_3, \widetilde{T}_4\}$ is outside, or vice versa; see section 4.5 for more details.
The slice choice is made based on the *interior* tetrahedra. Without loss of generality, denote the interior tetrahedra as $\{\widetilde{T}_1^A, \widetilde{T}_2^A\}$ for slice A and $\{\widetilde{T}_1^B, \widetilde{T}_2^B\}$ for slice B.
Let $\theta_{\min}^A$, $\theta_{\max}^A$ be the min and max dihedral angles for $\{\widetilde{T}_1^A, \widetilde{T}_2^A\}$ and $\theta_{\min}^B$, $\theta_{\max}^B$ be similarly defined for slice B.
// apply slice choice policy.
if $\theta_{\min}^A > 11.47^\circ$ and $\theta_{\min}^B > 11.47^\circ$ **then**
Choose slice A if $\theta_{\max}^A < \theta_{\max}^B$; otherwise, choose slice B. // minimize the maximum dihedral angle.
else
Choose slice A if $\theta_{\min}^A > \theta_{\min}^B$; otherwise, choose slice B. // maximize the minimum dihedral angle.
end if
end for

Local edge-flips within octahedra have the following advantages:

- They do not cause any *propagation* problems, i.e., where one has to continue modifying the mesh away from the initial edge-flip region. The operation is completely contained within the octahedral cell.
- By Proposition 4, the edge-flip introduces a new edge that is guaranteed to *not* be an active cut edge.
- It is still possible to compute a priori bounds on the dihedral angles.

Algorithm 4 implements an edge-flip policy for the remaining active cut long edges.

Remark 5. The "slice" choice policy in Algorithm 4 is important to guarantee good dihedral angles for the *interior* mesh. We consider only the interior tetrahedra

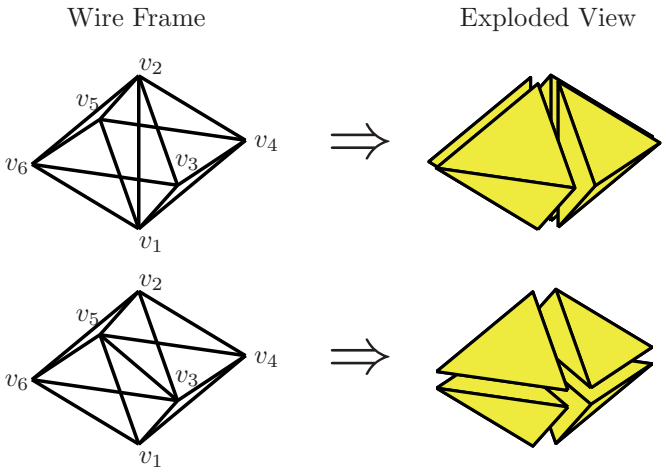


FIG. 9. *Edge-flip operation. Top part of figure is the original slice S. Bottom part shows slice A (see* (9)*). slice B is analogous to slice A.*

of each slice because it is not possible to ensure good angles for all four tetrahedra if the four outer vertices of the octahedron move in extreme ways.

Different criteria can be used to choose the slice. One choice is to always maximize the minimum dihedral angle. But our dihedral angle bound calculations (see Appendix A) found that it is not possible to do better than 11.47° for the minimum dihedral angle. Moreover, the maximum dihedral angle can degrade to $\approx 166°$ if we are concerned only with the minimum dihedral angle. So our policy acts to first ensure that the minimum dihedral angle has *sufficient* quality, followed by ensuring a good maximum dihedral angle.

We conclude with a result stating that the active long edge-flips in Algorithm 4 can be performed independently.

PROPOSITION 5. *Suppose all cut short edges of the mesh are suppressed, and suppose there are two distinct active cut long edges E_1, E_2. Then both edges can be flipped with Algorithm* 4*, and the connectivity of the resulting mesh is guaranteed to be consistent (i.e., the mesh is conforming and there are no overlapping, tangled elements).*

Proof. Let O_1 and O_2 be the octahedra associated with E_1 and E_2 (respectively). Suppose that O_1 and O_2 overlap (nonempty intersection). Then, by Proposition 1, part 3, O_1 must contain E_2 as an outer long edge and O_2 must contain E_1 as an outer long edge (recall Figure 2 (b) and Figure 8). But by Proposition 4, it means that E_1, E_2 have both end points labeled manifold, implying that E_1 and E_2 are not active. Ergo, O_1 and O_2 cannot overlap.

Therefore, since the long edge-flip within each octahedron does not affect the mesh connectivity outside each octahedron, it is clear that the new mesh connectivity is consistent. □

4.5. Choosing inside and outside tetrahedra. The last step of Algorithm 1 marks which tetrahedra in the mesh are inside or outside the surface. It is clear that if the level set function ϕ is positive at all nonmanifold vertices of a tetrahedron T, then T should be marked as being inside Γ. If a tetrahedron T has all four vertices labeled manifold, then T is ambiguous and it is not obvious where it belongs. The

simplest strategy is to consider all ambiguous tetrahedra to be outside the surface, i.e., just delete them. The final output mesh $(\mathcal{T}_{\text{out}}, \mathcal{V}_{\text{out}})$ consists of only the interior tetrahedra and their referenced vertices.

5. Meshing guarantees. Despite the simplicity of Algorithm 1, we are able to achieve the following guarantees on the output mesh:

- The tetrahedra of the *interior* mesh have good dihedral angles.
- The boundary of the output mesh is close to the surface Γ.
- If Γ is $C^{1,1}$ (i.e., has bounded curvature) and the BCC grid has sufficient resolution, then the boundary of the output mesh is homeomorphic to Γ.

Proving the above items is done by techniques similar to those in [36]. We outline them in the following sections.

5.1. Dihedral angles for Version 1. If Algorithm 1 is used to mesh an arbitrary continuous surface Γ, but without step 7, we have the following bounds on the dihedral angles for the interior tetrahedra (see Appendix A):

$$\text{minimum dihedral angle} > 8.54^\circ, \qquad \text{maximum dihedral angle} < 164.18^\circ. \tag{10}$$

Of course, any edges/corners of Γ will not be respected by the output mesh (i.e., there will not be a set of mesh edge segments that conform to a surface edge). These angle bounds are directly comparable to the bounds obtained by the method described in [36]. However, we highlight the following differences:

- For one-sided meshing, the bounds in [36] are better than in (10) (by roughly 2° to 3° depending on the parameters of their method; see [36] for more details).
- The method in [36] requires extra refinement of the mesh in order to conform to the surface. In particular, they require the use of special stencils (8 nontrivial stencils after accounting for symmetries) which they use to subdivide the BCC mesh. This requires a matching procedure to choose the correct stencil and the use of a parity rule [36].

Thus, our method makes a small trade-off in the angle qualities but gains in simplicity. The most complicated aspect is the edge-flips, which are inexpensive to compute and straightforward to implement because the structure of the BCC mesh is known a priori. Our angle bounds are obtained by a computer-assisted proof (see Appendix A).

5.2. Geometric and topological accuracy. Obviously, the generated mesh should be a faithful representation of the shape described by the surface Γ. It is clear from the algorithm that every vertex on the boundary of the output mesh lies on Γ. Moreover, the algorithm never connects an interior vertex (i.e., positive level set sign) with an exterior vertex (negative sign), so the output mesh respects Γ.

The method in [36] also achieves this, and the authors prove a geometric and topological accuracy result provided the background mesh is sufficiently fine. Indeed, their statements are general in that they apply to any background mesh method that satisfies the statements in the previous paragraph; ergo, they directly apply to our method as well. For convenience of the reader, the following two theorems taken from [36] apply to our method.

THEOREM 1 (one-sided Hausdorff bound). *Suppose Algorithm* 1 *meshes a continuous level set function ϕ. (It does not matter if ambiguous tetrahedra are included in the output mesh.) For any point p in space, if p lies in an output tetrahedron but $\phi(p) < 0$ (implying that p should lie outside the mesh), or if p does not lie in an output tetrahedron but $\phi(p) > 0$, then p is within a distance no greater than $\omega = \sqrt{7/8}$ from*

the isosurface $\{\phi = 0\}$*, i.e.,* Γ*. (The number* ω *applies for the unscaled* BCC *lattice. If the* BCC *lattice is scaled by* c*, the number is* ωc*.)*

Proof. The same proof in [36, 35] applies here as well despite the edge-flip in our method. This is because you can go back to the reference BCC grid to compute the worst-case distance that p can be from Γ. ∎

Remark 6. Theorem 1 says that if the background lattice is scaled by a factor c, then the greatest distance between a mesh boundary point and its nearest point on Γ converges to zero as $c \to 0$.

THEOREM 2 (topologically accurate). *Suppose Algorithm* 1 *meshes a continuous level set function* ϕ *whose zero surface* Γ *is* $C^{1,1}$ [1, 19]*. Assume the background BCC grid is scaled by* c*. Let* $R_{\mathrm{m}} > 0$ *be the minimum distance from a point on* Γ *to a point on the medial axis of* Γ [57]*. (Thus,* R_{m} *is a lower bound on the radius of curvature of* Γ*.) If* $R_{\mathrm{m}} > \omega c$*, with* ω *defined as in Theorem* 1*, then every point on* Γ *is within a distance* ωc *from the mesh boundary. Moreover, if* c/R_{m} *is sufficiently small, then the boundary of the mesh is homeomorphic to* Γ*, and there is a continuous deformation of space that carries* Γ *to the mesh boundary (i.e., there is an ambient isotopy from the identity map on* Γ *to the homeomorphism that maps* Γ *to the mesh boundary).*

Proof. See [36] for the proof and [35] for more details. ∎

6. Algorithm: Version 2.

6.1. One bad case. When computing the dihedral angle bounds, it was found that there is one case that prevents our algorithm from outright beating the method in [36] in terms of dihedral angle bounds for one-sided meshing. This case, and its mirror image, are shown in Figure 10. More specifically, the movement directions of the three manifold vertices in Figure 10 are given by the unit vectors

$$\begin{aligned} &\text{Case 1:} \quad (1,-1,1)/\sqrt{3}, \quad (-1,-1,1)/\sqrt{3}, \quad (-1,1,1)/\sqrt{3}, \\ &\text{Case 2:} \quad (1,1,1)/\sqrt{3}, \quad (-1,-1,1)/\sqrt{3}, \quad (-1,1,1)/\sqrt{3}. \end{aligned}$$

The two cases in Figure 10 are not unreasonable, since they can arise from a flat surface, parallel to the y-z plane, cutting through the mesh at the "right" x-intercept.

However, ignoring this one case would yield the angle bounds for a single tetrahedron with three vertices labeled manifold (see Appendix A.3.1),

$$\text{(11)} \quad \text{minimum dihedral angle} > 13.26^\circ, \qquad \text{maximum dihedral angle} < 157.59^\circ,$$

which are, of course, much better than (10).

6.2. Another edge-flip. The case in Figure 10 can be eliminated by an appropriate long edge-flip (similar to Figure 9). Consider the edge-flip policy depicted in Figure 11; the details are given in Algorithm 5.

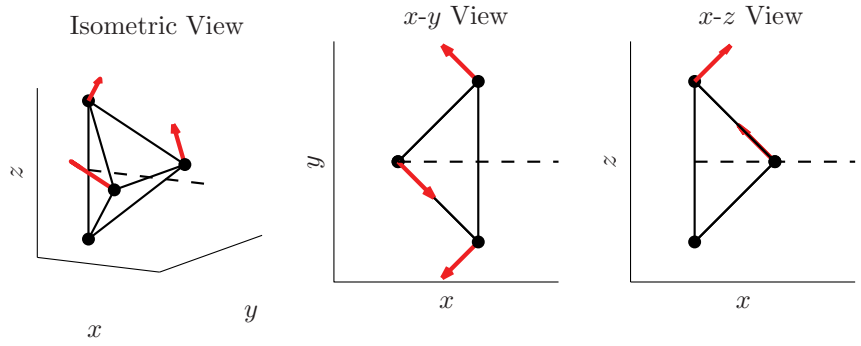
The edge-flip operation shown in Figure 11 (for both mirror image cases) yields the following dihedral angle bounds for the whole octahedron (see Appendix A.3.1):

$$\text{(12)} \quad \text{minimum dihedral angle} > 18.53^\circ, \qquad \text{maximum dihedral angle} < 150.01^\circ.$$

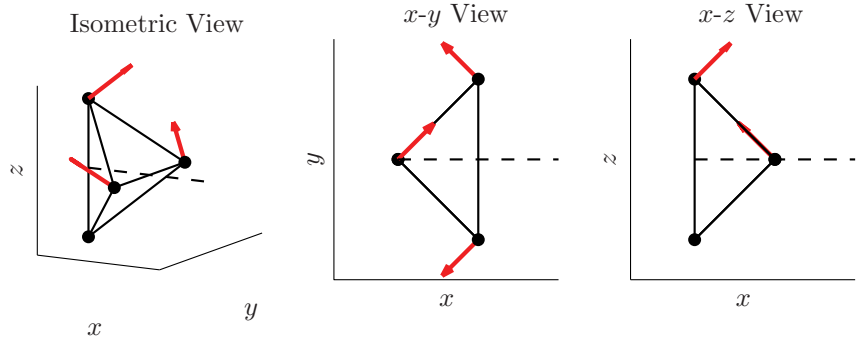
Note: these bounds are for when the manifold vertices meet the extreme movement criteria given in Algorithm 5.

Of course, it is important that this additional edge-flip not interfere with itself or the active long edge-flip discussed in section 4.4. We now verify this.

PROPOSITION 6. *Suppose all cut short edges of the mesh are suppressed, and suppose there are two distinct long edges* E_1*,* E_2 *with only one end point labeled manifold.*



(a) Case 1.



(b) Case 2.

FIG. 10. *Two cases that give the worst angles. Cases* 1 *and* 2 *(which are mirror images of each other) consist of three vertices labeled manifold and are moving along (short) edges that are highlighted by the red arrows. Recall that there are eight directions along which manifold vertices can move* (7)*. Because of our algorithm, the vertices cannot move more than half the short edge length. From our computer-assisted proof, it turns out that when the three vertices move the maximum amount, the angle bounds in* (10) *are achieved. Hence, they are sharp estimates because this can occur for a flat surface.*

Let O_1*,* O_2 *be the octahedral cells associated with* E_1 *and* E_2 *(respectively). Assume that* O_1*,* O_2 *match the case depicted in Figure* 11*; i.e., each has three manifold vertices moving in the directions shown. Then both edges can be flipped with Algorithm* 5*, and the connectivity of the resulting mesh is guaranteed to be consistent (i.e., the mesh is conforming and there are no overlapping, tangled elements).*

Proof. We proceed similar to the proof of Proposition 5 in that we must show that O_1 cannot contain E_2 and O_2 cannot contain E_1. For convenience, let us adopt the vertex labeling shown in Figure 11. If O_1 did contain E_2, then $E_2 = (v_3, v_6)$ or $E_2 = (v_4, v_5)$ (because E_2 has one manifold end point). But this would imply that O_2 does not match the case depicted in Figure 11; i.e., E_2 would not have its manifold vertex moving "upward." In other words, if E_1 is a long edge to be flipped by Algorithm 5, then the surrounding long edges (v_3, v_6), (v_4, v_5) will *not* be flipped by Algorithm 5. Therefore, O_1 cannot contain E_2 and O_2 cannot contain E_1, so the algorithm cannot interfere with itself. ∎

We also have the following proposition.

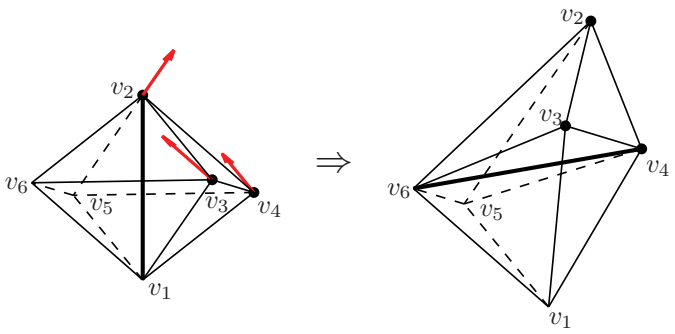


FIG. 11. *Additional edge-flip policy. Diagram shows an octahedron with three manifold vertices labeled by bold black dots and moving along the red arrows (corresponds to Figure* 10 (a)*); the movement directions are relative to the octahedron's spine orientation. If the manifold vertices move by a large amount, then the tetrahedron with three manifold vertices can have bad angles (recall Figure* 10*). In this case, we flip the spine of the octahedron (bold line segment) using the same operation depicted in Figure* 9*. This produces tetrahedra with better dihedral angles. Note: the same policy holds for the mirror image case (Figure* 10 (b)*).*

Algorithm 5 Additional Edge-Flip Policy

Let $\mathcal{E}_{\text{flip}}$ be the set of long edges that have one end point labeled manifold.
for $E \in \mathcal{E}_{\text{flip}}$ **do**
  Find the octahedron O associated with E, i.e., the set of tetrahedra $O = \{T_i\}_{i=1}^4$ that share E as a common edge.
  Order (number) the vertices of O as depicted in Figure 11.
  if O has only 3 manifold vertices that match the labeling configuration shown in Figure 11 **then**
    Let $\mathbf{d}_i = \tilde{\mathbf{x}}_i - \mathbf{x}_i$, where $\mathbf{x}_i$, $\tilde{\mathbf{x}}_i$ are the coordinates of v_i and its destination point (respectively) for $i = 2, 3, 4$.
    if $\{\mathbf{d}_2, \mathbf{d}_3, \mathbf{d}_4\}$ match the directions shown in Figure 10 (either case) **then**
      Define $d_i = |\mathbf{d}_i| / (cL_{\text{st}})$, for $i = 2, 3, 4$.
      if $d_2 \geq 0.3$ AND $d_3 \geq 0.3$ AND $d_4 \geq 0.3$ **then**
        Let $\{\widetilde{T}_i^A\}_{i=1}^4$ be the alternate set of tetrahedra given by the set of six vertices in O and the slice A operation depicted in Figure 9 and (9). Note: Slice B could be used (it does not matter).
        Replace O by $\{\widetilde{T}_i^A\}_{i=1}^4$.
      end if
    end if
  end if
end for

PROPOSITION 7. *Both edge-flip policies, Algorithms* 4 *and* 5*, cannot interfere with each other.*

Proof. Obviously, if there is an active cut long edge to be flipped, then none of its surrounding long edges can have *only* one end point labeled manifold by Proposition 4.

Alternatively, if there is a long edge that is the spine of an octahedron with the configuration shown in Figure 11, then none of the surrounding long edges can be active cut long edges. Note that the edge (v_5, v_6) cannot be a cut long edge, because then Proposition 4 would imply that v_1 is labeled manifold, which is a contradiction. So neither edge-flip policy can affect the other. ∎

Remark 7. We execute Algorithm 5 immediately after Algorithm 4. In fact, both algorithms can be executed *completely* in parallel, since no communication is necessary between edge-flips. The movement criteria, i.e., $d_2, d_3, d_4 \geq 0.3$, are *not* numerically sensitive. For example, one could use 0.31 and still obtain the same overall dihedral angle bounds in (15).

6.2.1. Sufficient resolution eliminates conflicting case. The edge-flip policy in Algorithm 5 cannot always eliminate the two cases shown in Figure 10. For example, v_5 in Figure 11 could also be labeled manifold and be moving in such a way that the long edge (v_5, v_6) may *also* need to be flipped. Obviously, we cannot apply Algorithm 5 to both long edges without ruining the output mesh connectivity. Therefore, if this "conflicting" situation arises, the safest policy is to not flip either.

However, it is worthwhile to consider when (and if) this situation can indeed occur. To this end, let R_{m} be the minimum distance from a point on Γ to a point on the *medial axis* of Γ [57] (i.e., shape skeleton of Γ). Next, choose any four points on Γ that are not coplanar, and find the sphere that intersects all four points. The radius of the sphere provides an upper bound on R_{m}; if the four points are coplanar, then take the radius to be $+\infty$. This essentially follows from the definition of medial axis and is related to the concept of global radius of curvature [27, 26] which can be extended to surfaces.

The following remark summarizes when this conflicting situation can happen.

Remark 8. Suppose we apply the mesh generation Algorithm 1 with a scaled BCC lattice (with spacing c) and there is a long edge E with one end point labeled manifold, and E is the spine of an octahedron with vertices ordered as shown in Figure 11. In particular, assume that $\{v_2, v_3, v_4, v_5\}$ are labeled manifold and that $\{v_2, v_3, v_4\}$ are moving along the directions described in Figure 10 (relative to the spine orientation E). In addition, assume that the vertices $\{v_2, v_3, v_4\}$ move more than $0.3cL_{\mathrm{st}}$, where c is the lattice spacing and $L_{\mathrm{st}} = \sqrt{3}/2$ (i.e., $\{v_2, v_3, v_4\}$ satisfy the criteria of Algorithm 5 to execute an edge-flip). Then, no matter which valid short edge direction v_5 moves along, R_{m} (defined earlier) satisfies the estimate

$$R_{\mathrm{m}} < 1.1c. \tag{13}$$

This result is a straightforward application of a computer-assisted proof, similar to our approach for estimating the dihedral angle bounds (see Appendix A.3.2).

Thus, the "conflicting" situation can be avoided if the lattice spacing satisfies

$$c \leq R_{\mathrm{m}}/1.1, \tag{14}$$

which is not an unreasonable assumption if we want a decent approximation of Γ.

6.2.2. Last resort edge-flip. If (14) is not satisfied, and if a fourth vertex is manifold, we can still *try* to flip, as long as that fourth vertex does not move more than $0.3cL_{\mathrm{st}}$. That way we know we do not need to flip the long edge (v_5, v_6). This is a straightforward modification of Algorithm 5. In this case, a choice must be made between not flipping and using one of the alternate slices (A or B). The same decision policy used in Algorithm 4 can be used here.

6.3. Improved angle bounds for Version 2. All the properties that were satisfied for Version 1 (see section 5) also hold for Version 2, except the dihedral angle bounds are better (see Appendix A.4):

$$\text{minimum dihedral angle} > 11.47^\circ, \qquad \text{maximum dihedral angle} < 157.59^\circ, \tag{15}$$

provided condition (14) is satisfied. These bounds are uniformly better than the bounds given in [36].

7. Numerical results. We apply Algorithm 1 (Version 2) to a variety of shapes in the following sections. Step 1 was implemented in MATLAB. Steps 2–8 were implemented in C++ by a MATLAB-MEX file for efficiency. The BCC lattice is built by first defining a standard (scaled) *cubic* lattice, with N points per unit length along each dimension, followed by including the centroid of each cube as additional vertices. Next, the triangulation connectivity data is straightforwardly defined and we store all long and short edges separately. Then we store the local neighboring tetrahedra of all long edges (i.e., the octahedra).

Several figures show results of Algorithm 1 (Version 2). The minimum and maximum dihedral angles for each example are listed in the figure.

7.1. Level set input data. For these examples, the input data is a scalar function $\phi(x, y, z)$ (i.e., a level set function). For the simple shapes, ϕ is implemented analytically. The Bimba shape is represented by a 3-D Cartesian grid with level set values defined at the grid points; thus, ϕ is computed by interpolation of the grid data. Timings are given in Table 1.

TABLE 1

Statistics for the level set input cases in Figures 12 *and* 13 *are given: number of cubic lattice points used (along one coordinate direction), number of output tetrahedra, number of surface triangles, and computational running times (in seconds) for each step of Algorithm* 1 *(Version* 2*). The hardware used was a Lenovo laptop with Intel Core i*7*-*2640*M CPU @* 2.80 *GHz processor,* 8 *GB RAM, and Windows* 7 *(*64*-bit). Newton's method was used for computing cut points in step* 1 *for the first three examples. Bisection was used for step* 1 *in the Bimba example, which is why the running time is longer. Note: these running times do not include the initial setup time of the background BCC mesh.*

Shapes	Pts	Tetra.	Triang.	Step 1	Step 2	Step 3	Step 4	Steps 5–8	Total
Plane	31	163,210	12,682	0.044	0.011	< 0.001	< 0.001	0.011	< 0.068
Sphere	31	2,904	552	0.046	0.010	< 0.001	< 0.001	0.010	< 0.068
Torus	31	8,012	1,640	0.082	0.011	0.001	< 0.001	0.011	< 0.106
Bimba	51	112,495	8,670	0.299	0.040	0.003	0.002	0.055	0.399

7.1.1. Simple shapes. For the three cases in Figure 12, we restrict the scaled BCC lattice to the unit cube, using 31 *cubic lattice points* along each of the coordinate directions x, y, z. The level set functions for each shape are defined analytically. The computational running time for each shape is given in Table 1. The same background cube mesh was used for each of the shapes in Figure 12.

7.1.2. Bimba sculpture. Similar setup as in the previous section, except the scaled BCC lattice (restricted to the unit cube) uses 51 *cubic lattice points* along each of the coordinate directions x, y, z (see Figure 13). The level set function $\phi(x, y, z)$ was defined by linear interpolation of samples on a $100\times100\times100$ Cartesian grid. The running times are given in Table 1. Step 1 was implemented in MATLAB using *bisection* to compute the cut points instead of Newton's method. Thus, step 1 takes a lot longer here than for the simple shapes. Recall that Newton's method converges *quadratically*, whereas bisection converges only linearly. However, Newton's method requires derivative information [59, 60].

7.2. Surface mesh inputs. For these examples (Figures 14–19), the input data is a closed manifold (watertight) triangular surface mesh. Hence, we use bisection to compute the cut points, which requires that we "query" the geometry to determine

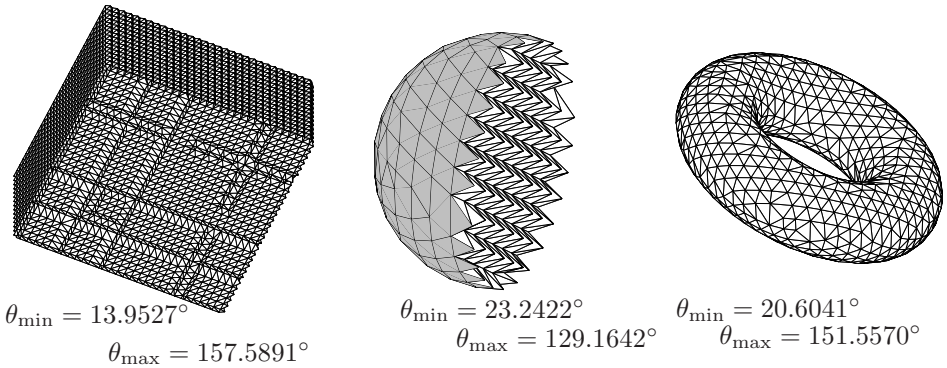


FIG. 12. *Meshing results for simple shapes (left to right: plane, sphere, torus). All three cases were meshed with a background grid given by the unit cube meshed by BCC tetrahedra (*31 *cubic lattice points along each coordinate). The plane case realizes the upper bound on the maximum dihedral angle. The sphere of radius* 0.13 *is shown with a cutaway view to illustrate the interior tetrahedra. The torus with cross-sectional radius* 0.08 *and "sweeping" radius* 0.2 *(e.g., surface of revolution) is rotated* 25° *with respect to the x-y plane of the background cube before meshing. The minimum and maximum dihedral angles are given.*

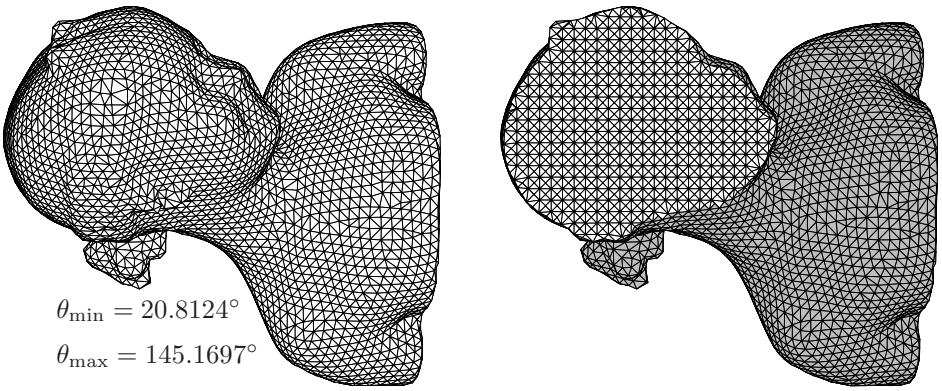


FIG. 13. *Meshing results for a sculpture "Bimba" (level set data courtesy of Xin Li at Louisiana State University). Mesh was generated from a unit cube of BCC tetrahedra (*51 *cubic lattice points along each coordinate). The surface mesh is shown on the left, with a cutaway view on the right to illustrate the interior tetrahedra. The minimum and maximum dihedral angles are given.*

whether or not a point is inside the surface. The querying operation was done in MATLAB using an M-file `inpolyhedron`, by Sven Holcombe, available from the MATLAB File Exchange. This causes the running time of step 1 to be rather long. It is certainly possible to further improve the efficiency of step 1, which is the subject of future work. The method in [36] had the same issue. Again, the scaled BCC lattice is restricted to the unit cube. Timings for all examples are given in Table 2.

Most of the input meshes were obtained from the Aim@Shape website:

TABLE 2

Statistics for the surface mesh input cases in Figures 14–19 *are given. The same format as in Table* 1 *is used, as is the same hardware. Bisection (and a geometry query) was used for step* 1 *in all examples.*

Shapes	Pts	Tetra.	Triang.	Step 1	Step 2	Step 3	Step 4	Steps 5–8	Total
Bumpy Torus	31	59,432	8,502	14.885	0.013	0.003	< 0.001	0.021	< 14.923
Genus 3	51	118,030	13,466	20.451	0.041	0.005	0.001	0.060	20.558
Hand	61	138,592	12,368	28.541	0.069	0.005	0.002	0.091	28.708
Skull	81	648,464	68,182	87.778	0.160	0.026	0.009	0.292	88.265
Stanford Bunny	61	326,242	17,932	36.785	0.073	0.006	0.004	0.097	36.965
Stanford Dragon	41	32,854	5,930	14.149	0.031	< 0.001	< 0.001	0.031	< 14.213

`http://shapes.aimatshape.net`. However, all surface meshes shown in the figures are the result of Algorithm 1; i.e., the surface mesh shown is the boundary of the output tetrahedral mesh produced by Algorithm 1, Version 2.

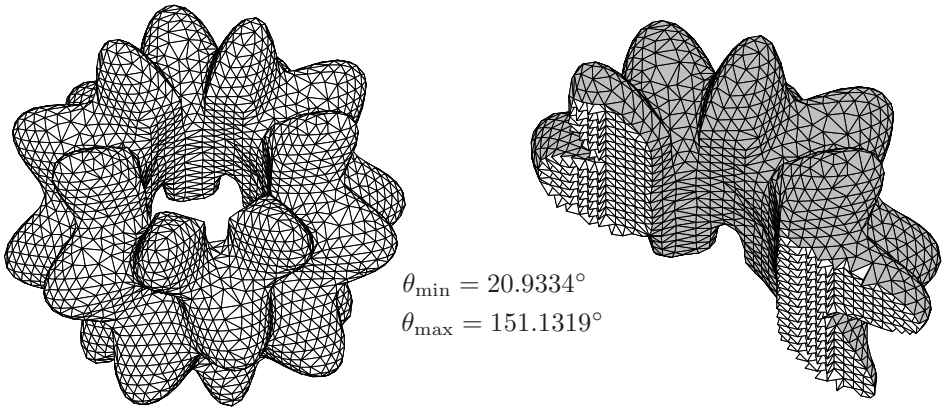


FIG. 14. *Meshing results for a Bumpy Torus (surface mesh data from the Aim@Shape website). Mesh was generated from a unit cube of BCC tetrahedra (*31 *cubic lattice points along each coordinate). Same format as Figure* 13.

We compare our Stanford Dragon meshing results (see last row of Table 2 and Figure 19) to the results obtained by the meshing method in [36]. Labelle and Shewchuk's Stanford Dragon mesh had 32,853 tetrahedra (very close to our result), with minimum and maximum dihedral angles of 14.9° and 157.5°. The total computation time took 24.5 seconds, of which 0.172 seconds were for mesh generation; the rest of the time was spent doing bisection and geometry queries to determine if a point is inside or outside the surface Γ (i.e., step 1). The hardware they used was a Mac Pro with a 2.66 GHz Intel Xeon processor. For our Stanford Dragon, the output mesh had 32,854 tetrahedra, with a minimum and maximum dihedral angle of 15.9° and 152.0°. The total computation time took 14.213 seconds, of which 0.064 seconds were for mesh generation (i.e., steps 2–8). Hence, both methods have comparable computing times. Note the discrepancy in the different hardware used when comparing the computing times.

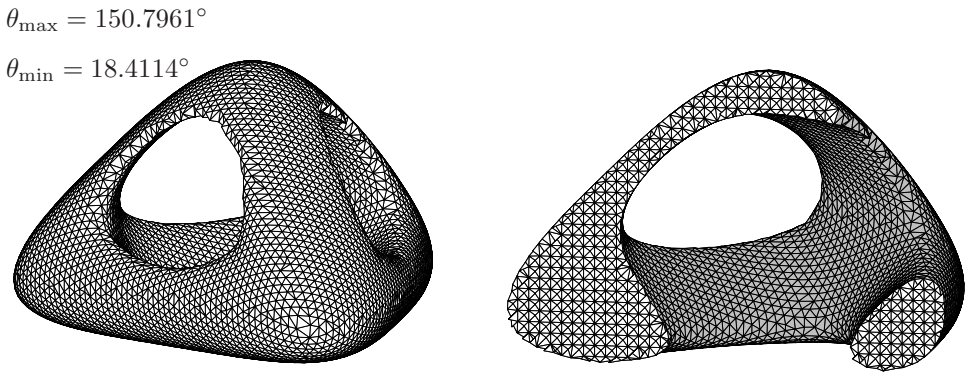


FIG. 15. *Meshing results for a Genus* 3 *shape (surface mesh data from the Aim@Shape website). Mesh was generated from a unit cube of BCC tetrahedra (*51 *cubic lattice points along each coordinate). Same format as Figure* 13*. The surface mesh is slightly jagged in the high curvature regions.*

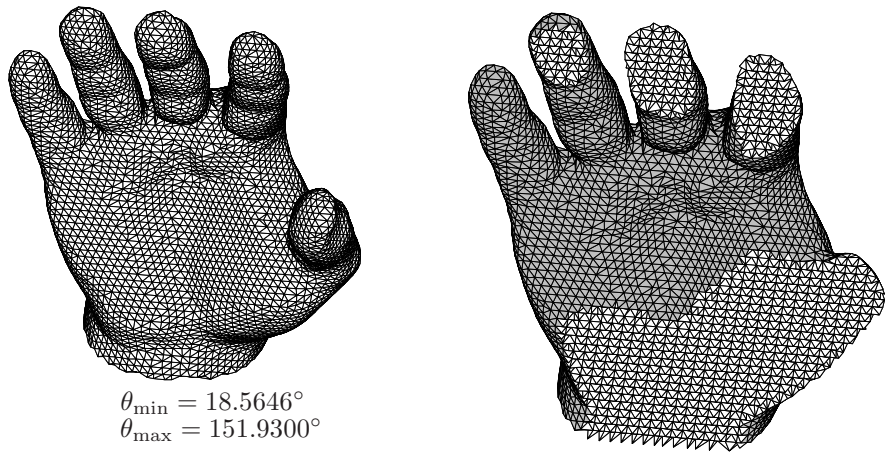


FIG. 16. *Meshing results for a hand shape (surface mesh data from the Aim@Shape website). Mesh was generated from a unit cube of BCC tetrahedra (*61 *cubic lattice points along each coordinate). Same format as Figure* 13*.*

8. Conclusion.

8.1. Adapted meshes. We presented a method for generating quasi-uniform tetrahedral meshes of isosurfaces. A *graded* mesh can be created by using an octree decomposition of the volume containing the isosurface; this technique was proposed by Labelle and Shewchuk [36]. Their idea was to have uniformly sized cubes covering the surface but then have successively larger cubes in the interior using an octree. Meshing the graded (interior) cubes was achieved through the use of stencils. Then

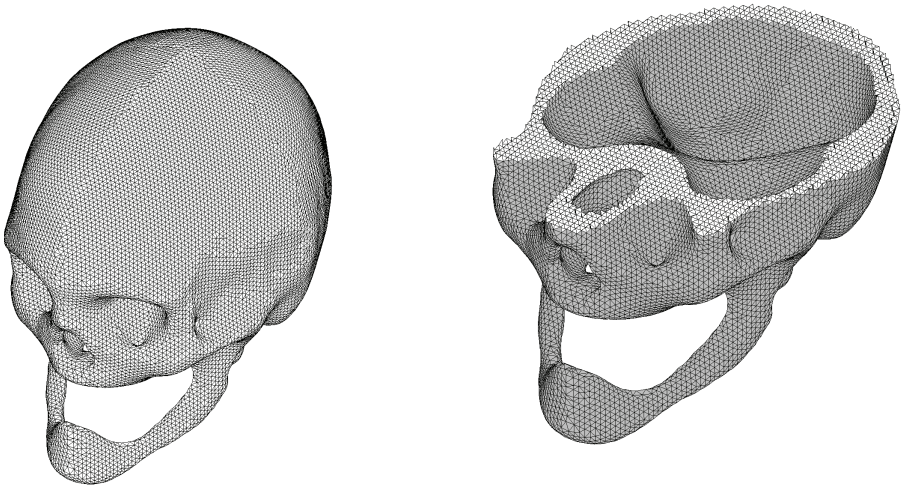


FIG. 17. *Meshing results for a skull with brain cavity (surface mesh data from the Aim@Shape website). Mesh was generated from a unit cube of BCC tetrahedra (*81 *cubic lattice points along each coordinate). Same format as Figure* 13. *Dihedral angles: minimum* 17.8194°, *maximum* 152.4381°.

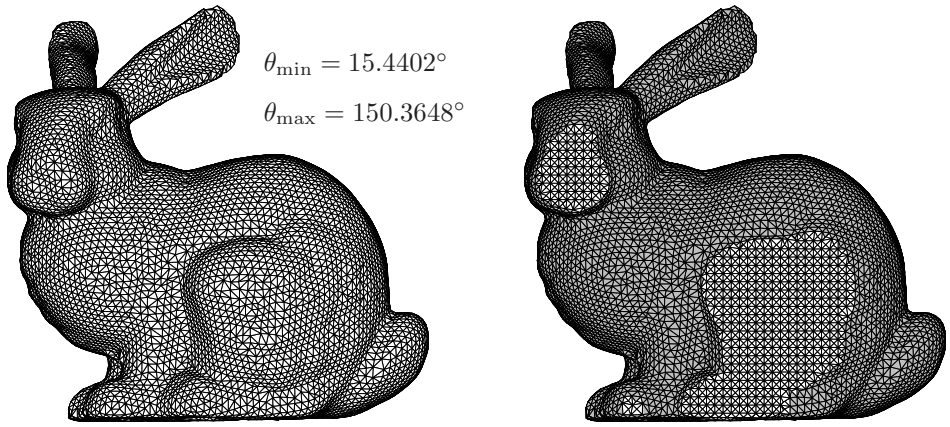


FIG. 18. *Meshing results for the Stanford Bunny (surface mesh data courtesy of Xin Li at Louisiana State University). Mesh was generated from a unit cube of BCC tetrahedra (*61 *cubic lattice points along each coordinate). Same format as Figure* 13.

they applied their meshing algorithm to the uniform part of the background grid that covers the surface. One can use the same approach for our method, i.e., use the Labelle and Shewchuk octree decomposition to generate the background grid, followed by the TIGER algorithm applied to the uniform region of the background grid that covers the surface.

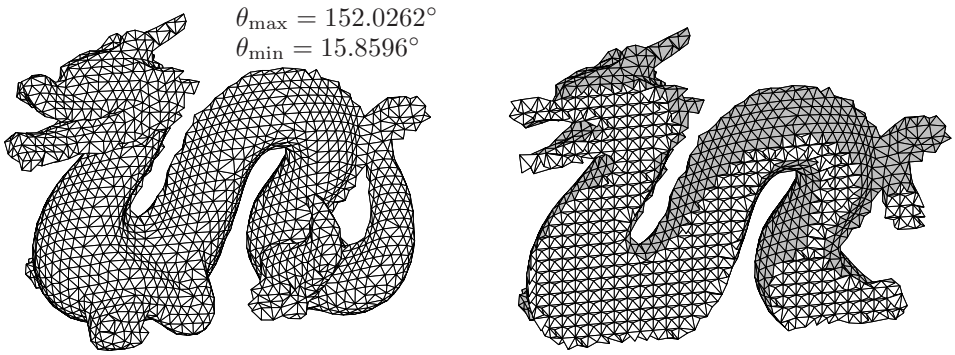


FIG. 19. *Meshing results for the Stanford Dragon (surface mesh data courtesy of Xin Li at Louisiana State University). Mesh was generated from a unit cube of BCC tetrahedra (*41 *cubic lattice points along each coordinate). Same format as Figure* 13.

Grading the mesh *on the surface*, while maintaining useful dihedral angle bounds, appears to be difficult. It is not clear how to achieve this with the TIGER algorithm. The same issue exists in the Labelle and Shewchuk method [36].

8.2. Parallelization. Every step of Algorithm 1 is parallelizable, except for the back-labeling procedure, step 4. In all numerical experiments performed, the back-labeling never propagated very far from the initial set of manifold vertices that moved toward other manifold vertices. However, one can construct pathological surfaces that could cause every end point of a cut edge to be visited by the back-labeling procedure. But these cases tend to be very sensitive to the position of the BCC mesh relative to the surface. Hence, one can try displacing the background grid slightly and rerunning the algorithm if the back-labeling takes too long.

8.3. Some heuristic approaches. Algorithm 1 is guaranteed only to mesh the interior or exterior of Γ but not both. However, in practice, both sides are usually adequate if the lattice spacing is sufficiently fine. It is still possible to have ambiguous tetrahedra, even for a flat surface. In this case, the ambiguous tetrahedra deform into slivers with zero volume. For one-sided meshing, ambiguous tetrahedra can be deleted. If both sides must be meshed consistently, then a bisection/refinement procedure could be used to remove the sliver [42]. As a last resort, try rerunning the algorithm with the surface shifted by a fraction of the lattice spacing. This may induce a different labeling and mesh topology that may not have ambiguous elements.

For ambiguous tetrahedra that have sufficient quality, there is still the issue of identifying whether they are inside or outside the surface. Various rules have been devised to classify these tetrahedra [36], the simplest being to evaluate ϕ at the centroid and let the sign determine whether it is in or out. One could introduce extra refinement to "break" the ambiguity, though this adds complexity to the implementation. Taking advantage of the octahedral view of the BCC mesh might be advantageous here.

The meshes generated by our algorithm tend to have good topological connectivity, so applying straightforward mesh smoothing techniques may be useful for improving the dihedral angles even further.

Appendix A. Computing guarantees on dihedral angles. We determine the dihedral angle bounds by direct computation over a finite number of cases. For example, a single BCC tetrahedron may have three of its four vertices labeled manifold, each of which may move along a short edge of the BCC mesh. There are a finite number of directions that each vertex can move, so there are only a finite number of cases to check. For each case, each vertex's position is represented by a *single parameter* (i.e., its distance along the short edge); call it α_i for the ith vertex. Thus, there is a 3-D parameter space that must be searched [36].

When searching the parameter space, we compute the six dihedral angles of the tetrahedron by the formula

$$\theta_k := \arccos(-f_k), \quad 1 \le k \le 6, \qquad f_k := \mathbf{N}_{k_1} \cdot \mathbf{N}_{k_2}, \tag{16}$$

where $\mathbf{N}_{k_1}$, $\mathbf{N}_{k_2}$ are the unit normal vectors of the adjoining faces of edge k of the tetrahedron, where $1 \le k_1, k_2 \le 4$. Note that the function $\arccos(-s)$ is a monotonically increasing function of s, so minimizing (or maximizing) θ_k is equivalent to minimizing or maximizing f_k. Note that $\mathbf{N}_i = \mathbf{a}_i \times \mathbf{b}_i / |\mathbf{a}_i \times \mathbf{b}_i|$, where $\mathbf{a}_i, \mathbf{b}_i$ are vectors defining the edges of face i, and that $|\mathbf{a}_i \times \mathbf{b}_i|$ is twice the area of face i. Clearly, f_k is a function of the four vertex positions of the tetrahedron. If no three vertices are co-linear (i.e., no tetrahedral face has zero area), then clearly f_k is differentiable with respect to the vertex positions. In particular, f_k is differentiable with respect to the three *parameters* $(\alpha_1, \alpha_2, \alpha_3)$ (in fact, it is C^∞).

Furthermore, if the area of each tetrahedral face is bounded below by a strictly positive constant, then one can obtain a uniform explicit bound on $|\nabla f_k|$ (uniformly with respect to the parameters), i.e.,

$$c_0 := \max_{1 \le k \le 6} \left[\max_{j=1,2,3} |(\partial_{\alpha_j} f_k)(\alpha_1, \alpha_2, \alpha_3)| \right]. \tag{17}$$

Now suppose we sample the 3-D parameter space by a uniform Cartesian grid. Essentially, we are interpolating a smooth function via trilinear interpolation. Ergo, we have the standard error estimate [9, 52]

$$\| f_k - \mathcal{I}_h f_k \|_{L^\infty(Q)} \le 3 c_0 h, \tag{18}$$

where Q is the parameter space "cube," $\mathcal{I}_h$ is the interpolation operator, h is the Cartesian grid size, and the "3" comes from summing over the three parameters. Therefore, assuming c_0 is known, one can compute the *true global minimum* of f_k (over Q) to within whatever accuracy is desired by choosing a small enough h. This is exactly how we computed the dihedral angle bounds. Furthermore, all calculations were done for an unscaled (unit) BCC lattice, because uniform scaling does not affect the angles. The following sections give details on each of the tetrahedral (or octahedral) configurations that were computed. All angle bounds are accurate to two decimal places (in degrees) and are *strictly* correct as written.

Remark 9. We actually estimate the *minimum* of all six dihedral angles of a tetrahedron, i.e.,

$$f_{\min} = \min_{1 \le k \le 6} f_k, \tag{19}$$

which is not differentiable but is Lipschitz with constant given by the largest derivative in absolute value over all f_k for $1 \le k \le 6$; this follows by a standard argument. So

we have an error estimate similar to (18) by basic Sobolev interpolation theory [9]. The same holds when computing the maximum dihedral angle over a tetrahedron.

Remark 10. Estimating c_0 requires computing the sensitivity of $\mathbf{N}_i$ to perturbations of the parameters, which critically depends on the area of the tetrahedral face i. In other words, one must find a *lower* bound on the area of each tetrahedral face over the entire parameter range. We did this using the same technique suggested above, i.e., we sample the face area function fine enough, then take a conservative (uniform) lower bound on the face area that is guaranteed to be correct. For all of the cases we investigate, the tetrahedral face area never comes close to being degenerate. Ergo, we obtain a correct, albeit conservative, estimate of c_0.

With the estimate in hand, we compute conservative estimates of the global minimum and maximum of f_k over the parameter range. Thus, we can compute (correctly) a lower bound on the minimum dihedral angle and an upper bound on the maximum dihedral angle using (16). The accuracy of these bounds can be tightened by simply using a smaller mesh spacing h.

Of course, this is an expensive calculation. But it is not part of the mesh generation method, so it has no impact on the algorithm's efficiency. Moreover, this calculation can be trivially parallelized so it is tractable. Indeed, one can even use an adaptive octree approach to make it more efficient.

A.1. Single tetrahedron. A single tetrahedron can have (at most) three of its four vertices labeled manifold. Ambiguous tetrahedra (all four vertices manifold) are *not* considered here because they are not included in the interior mesh; recall step 8 of Algorithm 1. In other words, we *guarantee only the interior mesh which does not include any ambiguous tetrahedra*.

More specifically, let $\{v_1, v_2, v_3, v_4\}$ be the vertices of a tetrahedron in the BCC mesh, and assume that v_1 does not move. Thus, we must search the parameter range:

$$\alpha_1 = 0, \qquad 0.0 \leq \alpha_i \leq 0.5 \quad \text{for } i = 2, 3, 4, \tag{20}$$

where α_i is the distance that v_i moves (relative to the initial vertex position in the BCC mesh) along a short edge normalized by $L_{\text{st}} = \sqrt{3}/2$.

Since there are only eight short edges adjoining each vertex, there are $8^3 = 512$ cases to evaluate. We rule out some of the cases where a vertex moves along a short edge toward another vertex that also moves a nonzero amount. This is because the back-labeling procedure eliminates those cases (see section 4.3). For this configuration, the worst-case dihedral angle bounds over all the valid cases are

$$\text{minimum dihedral angle} > 8.54^\circ, \qquad \text{maximum dihedral angle} < 164.18^\circ. \tag{21}$$

If the two cases in Figure 10 are *ignored*, then the angle bounds become

$$\text{minimum dihedral angle} > 13.26^\circ, \qquad \text{maximum dihedral angle} < 157.59^\circ. \tag{22}$$

A.2. Active cut long edge case.

A.2.1. Parameter ranges. An octahedron whose spine is an active cut long edge must have the four outer ring vertices labeled manifold (see Figure 8). Since there are eight short edges adjoining each vertex, there are $8^4 = 4096$ cases to evaluate. However, many of these cases are redundant because of rotational symmetry about the spine (rigid rotations do not change angles); thus, we eliminate the repeat cases. The parameter ranges are

$$\alpha_1, \alpha_2 = 0, \qquad 0.0 \leq \alpha_i \leq 0.5 \quad \text{for } i = 3, 4, 5, 6, \tag{23}$$

where α_i is the normalized distance that v_i moves along a short edge (see Figure 8).

When computing the maximum and minimum dihedral angles (for a fixed set of parameter choices), we apply the policy described in Algorithm 4. This means we compute the angles for only two of the four tetrahedra (i.e., $\{\widetilde{T}_1, \widetilde{T}_2\}$) while accounting for the "best" slice choice. The next section describes how we verified the dihedral angle bounds for the active cut long edge configuration.

A.2.2. Angle bounds.

LEMMA 2. *Consider the edge-flip policy in Algorithm* 4*, and suppose* $\{\widetilde{T}_1^A, \widetilde{T}_2^A\}$ *and* $\{\widetilde{T}_1^B, \widetilde{T}_2^B\}$ *are the interior tetrahedra for slices A and B. Then it is always guaranteed that at least one of the slices satisfies the dihedral angle bounds*

$$\text{(24)} \qquad \textit{minimum dihedral angle} > 11.47^\circ, \qquad \textit{maximum dihedral angle} < 157.49^\circ$$

over the entire parameter range given in (23) *and over all cases.*

Proof. Recall (16) and let $f^A_{\min}$, $f^B_{\min}$ be the minimum of f_k, for $1 \le k \le 6$, of slices A, B, respectively, and analogously define $f^A_{\max}$, $f^B_{\max}$. Define the dot product bounds by

$$\text{(25)} \qquad \eta_{\min} = -\cos(11.47\pi/180), \qquad \eta_{\max} = -\cos(157.49\pi/180).$$

Thus, proving (24) is equivalent to proving the following statement:

$$\text{(26)} \qquad (f^A_{\min} > \eta_{\min}) \text{ and } (f^A_{\max} < \eta_{\max}) \quad \text{or} \quad (f^B_{\min} > \eta_{\min}) \text{ and } (f^B_{\max} < \eta_{\max}).$$

We encode this relation into the function

$$\rho := (f^A_{\min} - \eta_{\min})^+ \cdot (\eta_{\max} - f^A_{\max})^+ + (f^B_{\min} - \eta_{\min})^+ \cdot (\eta_{\max} - f^B_{\max})^+,$$

where $s^+ := \max(s, 0)$. Note: ρ is a Lipschitz function with a bounded derivative that is easily estimated in terms of (17). Hence, proving (26) is equivalent to showing that ρ is bounded below (uniformly) by a positive constant. In other words, we must compute the global minimum of ρ for each case and verify that it is positive. We do this by the same type of computer-assisted proof as was described earlier, which requires estimating $\partial_{\alpha_j}\rho$ and using a sufficiently small mesh spacing h (recall Remark 10). Thus, we are able to verify that the angle bounds in (24) are strictly correct as written. ∎

Remark 11. We verify the result of Lemma 2 for the other two tetrahedra (i.e., $\{\widetilde{T}_3, \widetilde{T}_4\}$), because the interior of Γ could be on either side of Γ depending on the sign of ϕ.

For an arbitrary surface, we cannot guarantee good dihedral angles **and** a consistent/conforming mesh on *both* sides of the surface simultaneously for the active cut long edge case. This is because different slices may be used on either side of the surface due to the edge-flip policy in Algorithm 4. If the *same* slice were used on *both* sides, then the worst case dihedral angles become 0° and 180°! Clearly not desirable. For the same reason, we cannot guarantee meshing a domain such that the mesh conforms to an open surface *contained inside* the domain (e.g., a mesh that conforms to an internal "crack"). However, see the discussion in section 8.3 for potential remedies in practice.

A.3. Additional edge-flip case. Consider the octahedron shown in Figure 11 (as well as its mirror image). Hence, there are *two* cases to check with three parameters. The angle bounds we compute in this configuration are valid whether one always chooses slice A or slice B when executing the edge-flip.

A.3.1. Angle bounds. Recall the edge-flip policy described in Algorithm 5. So there are only three manifold vertices that move (out of six total). Hence, the parameter ranges are

$$\alpha_1, \alpha_5, \alpha_6 = 0, \qquad 0.3 \leq \alpha_i \leq 0.5 \quad \text{for } i = 2, 3, 4, \tag{27}$$

where α_i is the normalized distance that v_i moves along a short edge (see Figure 11). For this configuration, the dihedral angle bounds over the two cases are

$$\text{minimum dihedral angle} > 18.53^\circ, \qquad \text{maximum dihedral angle} < 150.01^\circ, \tag{28}$$

using either slice A or slice B.

If we use Algorithm 5, we must also consider the case when we *do not* flip; i.e., perhaps v_2, v_3, or v_4 do not move far enough. This implies that we reconsider the single tetrahedron case in Appendix A.1 except with the following three sets of parameter ranges:

$$\begin{aligned}
&(1) \quad 0.0 \leq \alpha_2 \leq 0.3, \quad 0.0 \leq \alpha_3 \leq 0.5, \quad 0.0 \leq \alpha_4 \leq 0.5, \\
&(2) \quad 0.0 \leq \alpha_2 \leq 0.5, \quad 0.0 \leq \alpha_3 \leq 0.3, \quad 0.0 \leq \alpha_4 \leq 0.5, \\
&(3) \quad 0.0 \leq \alpha_2 \leq 0.5, \quad 0.0 \leq \alpha_3 \leq 0.5, \quad 0.0 \leq \alpha_4 \leq 0.3;
\end{aligned}$$

i.e., the edge-flip policy in Algorithm 5 is *not* triggered. For this configuration, the dihedral angle bounds over the two cases are

$$\text{minimum dihedral angle} > 13.39^\circ, \qquad \text{maximum dihedral angle} < 156.68^\circ. \tag{29}$$

A.3.2. Allowable conditions for additional edge-flip. The additional edge-flip is not guaranteed to be allowable if there is an additional fourth vertex labeled in the octahedron shown in Figure 11 (see sections 6.2.1 and 6.2.2). It turns out that if the lattice spacing c of the BCC background grid is not sufficiently small relative to R_{m} (the minimum distance to the medial axis of Γ), then this "conflict" situation can indeed happen. Therefore, we would like to estimate how large c must be for this to occur.

We estimate via a numerical approximation. Suppose we are in the situation described by Remark 8. We have four parameters (the distances that the four manifold vertices move along their short edges) to be searched. For a fixed set of parameters, we have a set of four vertex positions $\{\tilde{v}_2, \tilde{v}_3, \tilde{v}_4, \tilde{v}_5\}$. We then find the sphere that intersects these vertices and compute its radius R_0. As noted earlier, this radius satisfies $R_0 \geq R_{\mathrm{m}}$. We then sweep the parameter space

$$0.3 \leq \alpha_i \leq 0.5 \quad \text{for } i = 2, 3, 4, 5,$$

where α_i refers to the normalized distance that v_i moves along its edge to its destination $\tilde{v}_i$. This search is performed for each possible set of movement directions. There are two directions along which v_2 can move: one direction for v_3 and one direction for v_4 (we are assuming $\{v_2, v_3, v_4\}$ satisfy the edge-flip criteria of Algorithm 5). And v_5 could move along seven distinct short edges (note it cannot move toward v_2 because of back-labeling). Hence, we sweep for 14 different cases.

Finding the sphere that intersects four points can be numerically *sensitive* if the four points are close to being coplanar. However, because of the choices of the parameters here, the four points are *never* close to being coplanar. This is easily

monitored in our code by checking the value of a 4×4 determinant. Essentially, this means that R_0 is differentiable with respect to the four parameters. We derive an explicit bound on the gradient of R_0 (with respect to α_i) and use that to rigorously estimate the error of our calculation. Basically, this is the same approach described earlier.

For this configuration, using the unscaled BCC lattice, we found $R_{\mathrm{m}} < 1.1$ uniformly over all 14 cases; this estimate is strictly correct as written. If the lattice is scaled by c, then a standard argument gives

$$R_{\mathrm{m}} < 1.1c,$$

which means that if the lattice spacing is too large, then the conflict situation *might* occur. Therefore, the additional edge-flip policy can *always* be done so long as

$$c \leq R_{\mathrm{m}}/1.1.$$

A.4. Final dihedral angle bounds.

THEOREM 3. *Version* 1 *of Algorithm* 1 *has dihedral angle bounds given by*

$$\textit{minimum dihedral angle} > 8.54^\circ, \qquad \textit{maximum dihedral angle} < 164.18^\circ.$$

For version 2 *(includes step* 7 *of Algorithm* 1*), let* c *be the* BCC *lattice spacing and* R_{m} *be defined as in Theorem* 2*. If* $c \leq R_{\mathrm{m}}/1.1$*, then the dihedral angles satisfy*

$$\textit{minimum dihedral angle} > 11.47^\circ, \qquad \textit{maximum dihedral angle} < 157.59^\circ.$$

Proof. For version 1, the angle bounds are given by (21) since those are clearly the most conservative. These bounds hold for any continuous surface, though the mesh that is generated will not respect geometric edges/corners.

For version 2, if we assume the condition $c \leq R_{\mathrm{m}}/1.1$, then (21) is no longer relevant because that case is removed by the additional edge-flip, Algorithm 5. Ergo, we must contend with (22) and (24); the bounds in (28) and (29) are of course more generous (so can be ignored). Therefore, we obtain the assertion. ∎

Note added in proof. The TIGER algorithm in implemented in the software package FELICITY, which is available at `www.mathworks.com/matlabcentral/fileexchange/31141-felicity`.

Acknowledgments. The author thanks Xin Li (Louisiana State University) for providing the Bimba, Stanford Bunny, and Stanford Dragon data to test the algorithm. The author also acknowledges `shapes.aimatshape.net`, the Aim@Shape website, for the other surface meshes.

REFERENCES

[1] R. A. ADAMS AND J. J. F. FOURNIER, *Sobolev Spaces*, 2nd ed., Pure Appl. Math. (Amsterdam) 140, Elsevier, Amsterdam, 2003.

[2] P. ALLIEZ, D. COHEN-STEINER, M. YVINEC, AND M. DESBRUN, *Variational tetrahedral meshing*, ACM Trans. Graph., 24 (2005), pp. 617–625.

[3] I. BABUŠKA AND A. K. AZIZ, *On the angle condition in the finite element method*, SIAM J. Numer. Anal., 13 (1976), pp. 214–226.

[4] B. S. BAKER, E. GROSSE, AND C. S. RAFFERTY, *Nonobtuse triangulation of polygons*, Discrete Comput. Geom., 3 (1988), pp. 147–168.

[5] R. E. BANK AND L. R. SCOTT, *On the conditioning of finite element equations with highly refined meshes*, SIAM J. Numer. Anal., 26 (1989), pp. 1383–1394.

[6] M. Bern, D. Eppstein, and J. Gilbert, *Provably good mesh generation*, J. Comput. System Sci., 48 (1994), pp. 384–409.
[7] J. Bey, *Tetrahedral grid refinement*, Computing, 55 (1995), pp. 355–378.
[8] D. Braess, *Finite Elements: Theory, Fast Solvers, and Applications in Solid Mechanics*, 2nd ed., Cambridge University Press, Cambridge, UK, 2001.
[9] S. C. Brenner and L. R. Scott, *The Mathematical Theory of Finite Element Methods*, 2nd ed., Springer, New York, 2002.
[10] R. Bridson, J. Teran, N. Molino, and R. Fedkiw, *Adaptive physics based tetrahedral mesh generation using level sets*, Engineering with Computers, 21 (2005), pp. 2–18.
[11] H. Carr, T. Theussl, and T. Möller, *Isosurfaces on optimal regular samples*, in Proceedings of the Symposium on Data Visualisation 2003, VISSYM '03, Aire-la-Ville, Switzerland, 2003, pp. 39–48.
[12] L. Chen, *Mesh smoothing schemes based on optimal Delaunay triangulations*, in 13th International Meshing Roundtable, Sandia National Laboratories, 2004, pp. 109–120.
[13] L. Chen and J. Xu, *Optimal Delaunay triangulations*, J. Comput. Math., 22 (2004), pp. 299–308.
[14] S.-W. Cheng and T. K. Dey, *Quality meshing with weighted Delaunay refinement*, SIAM J. Comput., 33 (2004), pp. 69–93.
[15] S.-W. Cheng, T. K. Dey, H. Edelsbrunner, M. A. Facello, and S.-H. Teng, *Sliver exudation*, in Proceedings of the Fifteenth Annual Symposium on Computational Geometry, SCG '99, ACM, New York, 1999, pp. 1–13.
[16] L. P. Chew, *Guaranteed-quality Delaunay meshing in* 3*d (short version)*, in Proceedings of the Thirteenth Annual Symposium on Computational Geometry, SCG '97, ACM, New York, 1997, pp. 391–393.
[17] P. G. Ciarlet, *The Finite Element Method for Elliptic Problems*, Classics Appl. Math. 40, SIAM, Philadelphia, PA, 2002.
[18] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr, *Implicit fairing of irregular meshes using diffusion and curvature flow*, in SIGGRAPH '99: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, ACM Press/Addison-Wesley, New York, 1999, pp. 317–324.
[19] M. P. do Carmo, *Differential Geometry of Curves and Surfaces*, Prentice Hall, Upper Saddle River, NJ, 1976.
[20] H. Edelsbrunner and D. Guoy, *An experimental study of sliver exudation*, Engineering with Computers, 18 (2002), pp. 229–240.
[21] H. Edelsbrunner, *Geometry and Topology for Mesh Generation*, Cambridge Monogr. Appl. Comput. Math., Cambridge University Press, Cambridge, UK, 2001.
[22] D. Eppstein, J. M. Sullivan, and A. Üngör, *Tiling space and slabs with acute tetrahedra*, Comput. Geom., 27 (2004), pp. 237–255.
[23] J. M. Escobar, G. Montero, R. Montenegro, and E. Rodríguez, *An algebraic method for smoothing surface triangulations on a local parametric space*, Internat. J. Numer. Methods Engrg., 66 (2006), pp. 740–760.
[24] L. A. Freitag and C. Ollivier-Gooch, *Tetrahedral mesh improvement using swapping and smoothing*, Internat. J. Numer. Methods Engrg., 40 (1997), pp. 3979–4002.
[25] A. Fuchs, *Automatic grid generation with almost regular Delaunay tetrahedra*, in Proceedings, 7th International Meshing Roundtable, Sandia National Laboratories, 1998, pp. 133–148.
[26] O. Gonzalez, J. H. Maddocks, F. Schuricht, and H. von der Mosel, *Global curvature and self-contact of nonlinearly elastic curves and rods*, Calculus of Variations, 14 (2002), pp. 29–68.
[27] O. Gonzalez and J. H. Maddocks, *Global curvature, thickness and the ideal shapes of knots*, Proc. Nat. Acad. Sci. USA, 96 (1999), pp. 4769–4773.
[28] R. Grosso, C. Lürig, and T. Ertl, *The multilevel finite element method for adaptive mesh optimization and visualization of volume data*, in VIS '97: Proceedings of the 8th Conference on Visualization, IEEE Computer Society Press, Washington, DC, 1997, pp. 387–394.
[29] H. Huang and U. Ascher, *Surface mesh smoothing, regularization, and feature detection*, SIAM J. Sci. Comput., 31 (2008), pp. 74–93.
[30] T. J. R. Hughes, *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*, Dover Publications, Mineola, NY, 2000.
[31] B. M. Klingner and J. R. Shewchuk, *Aggressive tetrahedral mesh improvement*, in Proceedings of the 16th International Meshing Roundtable, Seattle, WA, 2007, pp. 3–23.
[32] P. M. Knupp, *Algebraic mesh quality metrics*, SIAM J. Sci. Comput., 23 (2001), pp. 193–218.
[33] A. P. Kuprat and D. R. Einstein, *An anisotropic scale-invariant unstructured mesh generator suitable for volumetric imaging data*, J. Comput. Phys., 228 (2009), pp. 619–640.

[34] M. Křížek, *On the maximum angle condition for linear tetrahedral elements*, SIAM J. Numer. Anal., 29 (1992), pp. 513–520.
[35] F. Labelle, *Tetrahedral Mesh Generation with Good Dihedral Angles Using Point Lattices*, Ph.D. thesis, University of California, Berkeley, 2007.
[36] F. Labelle and J. R. Shewchuk, *Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles*, in SIGGRAPH '07, ACM, New York, 2007.
[37] X.-Y. Li and S.-H. Teng, *Generating well-shaped Delaunay meshed in* 3*d*, in Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '01, SIAM, Philadelphia, 2001, pp. 28–37.
[38] W. E. Lorensen and H. E. Cline, *Marching cubes: A high resolution* 3*d surface construction algorithm*, SIGGRAPH Comput. Graph., 21 (1987), pp. 163–169.
[39] S. A. Mitchell and S. A. Vavasis, *Quality mesh generation in higher dimensions*, SIAM J. Comput., 29 (2000), pp. 1334–1370.
[40] R. Montenegro, J. M. Cascón, J. M. Escobar, E. Rodríguez, and G. Montero, *An automatic strategy for adaptive tetrahedral mesh generation*, Appl. Numer. Math., 59 (2009), pp. 2203–2217.
[41] R. Montenegro, G. Montero, J. M. Escobar, and E. Rodríguez, *Efficient strategies for adaptive* 3*-d mesh generation over complex orography*, Neural Parallel Sci. Comput., 10 (2002), pp. 57–76.
[42] K. R. Moyle and Y. Ventikos, *Local remeshing for large amplitude grid deformations*, J. Comput. Phys., 227 (2008), pp. 2781–2793.
[43] H. Müller and M. Wehle, *Visualization of implicit surfaces using adaptive tetrahedrizations*, in DAGSTUHL '97: Proceedings of the Conference on Scientific Visualization, IEEE Computer Society Press, Washington, DC, 1997, p. 243.
[44] D. J. Naylor, *Filling space with tetrahedra*, Internat. J. Numer. Methods Engrg., 44 (1999), pp. 1383–1395.
[45] P. Ning and J. Bloomenthal, *An evaluation of implicit surface tilers*, IEEE Comput. Graph. Appl., 13 (1993), pp. 33–41.
[46] Y. Ohtake, A. Belyaev, and A. Pasko, *Dynamic meshes for accurate polygonization of implicit surfaces with sharp features*, in SMI '01: Proceedings of the International Conference on Shape Modeling & Applications, IEEE Computer Society, Washington, DC, 2001, p. 74.
[47] Y. Ohtake and A. G. Belyaev, *Dual/primal mesh optimization for polygonized implicit surfaces*, in SMA '02: Proceedings of the Seventh ACM Symposium on Solid Modeling and Applications, ACM, New York, 2002, pp. 171–178.
[48] Y. Ohtake, A. G. Belyaev, and I. A. Bogaevski, *Polyhedral surface smoothing with simultaneous mesh regularization*, in GMP '00: Proceedings of Geometric Modeling and Processing, IEEE Computer Society Press, Washington, DC, 2000, p. 229.
[49] S. Oudot, L. Rineau, and M. Yvinec, *Meshing volumes bounded by smooth surfaces*, in Proceedings of the 14th International Meshing Roundtable, San Diego, CA, 2005, pp. 203–219.
[50] P.-O. Persson, *Mesh size functions for implicit geometries and PDE-based gradient limiting*, Engineering with Computers, 22 (2006), pp. 95–109.
[51] P.-O. Persson and G. Strang, *A simple mesh generator in MATLAB*, SIAM Rev., 46 (2004), pp. 329–345.
[52] A. Ralston and P. Rabinowitz, *A First Course in Numerical Analysis*, 2nd ed., Dover, Mineola, NY, 2001.
[53] J. Schöberl, *Netgen an advancing front* 2*d/*3*d-mesh generator based on abstract rules*, Comput. Vis. Sci., 1 (1997), pp. 41–52.
[54] V. Shapiro and I. Tsukanov, *Implicit functions with guaranteed differential properties*, in SMA '99: Proceedings of the Fifth ACM Symposium on Solid Modeling and Applications, ACM, New York, 1999, pp. 258–269.
[55] J. R. Shewchuk, *What Is a Good Linear Finite Element? Interpolation, Conditioning, Anisotropy, and Quality Measures*, http://www.cs.cmu.edu/~jrs/jrspapers.html (2002).
[56] H. Si, *Tetgen: A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator, v*1.4.3 *User's Manual*, Technical report, Research Group: Numerical Mathematics and Scientific Computing, Weierstrass Institute for Applied Analysis and Stochastics (WIAS), Berlin, 2011.
[57] K. Siddiqi and S. M. Pizer, *Medial Representations*, Computational Imaging 37, Springer, New York, 2008.
[58] D. M. Y. Sommerville, *Space-filling tetrahedra in Euclidean space*, Proc. Edinb. Math. Soc., 41 (1923), pp. 49–57.
[59] G. W. Stewart, *Afternotes on Numerical Analysis*, SIAM, Philadelphia, 1996.

[60] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis*, 3rd ed., Texts Appl. Math. 12, Springer, New York, 2002.
[61] J. C. Torres, F. Soler, F. Velasco, A. León, and G. Arroyo, *Marching octahedra*, in Congreso Español de Informática Gráfica, 2009, pp. 179–186.
[62] A. Üngör, *Tiling 3d Euclidean space with acute tetrahedra*, in Canadian Conference on Computational Geometry, 2001.
[63] Z. J. Wood, P. Schröder, D. Breen, and M. Desbrun, *Semi-regular mesh extraction from volumes*, in VIS '00: Proceedings of the Conference on Visualization '00, IEEE Computer Society Press, Los Alamitos, CA, 2000, pp. 275–282.
[64] Y. Yao, C. S. Koh, and D. Xie, *Robust mesh regeneration based on structural deformation analysis for 3D shape optimization of electromagnetic devices*, in ICEMS 2003, Sixth International Conference on Electrical Machines and Systems, Vol. 2, 2003, pp. 732–735.
[65] M. A. Yerry and M. S. Shephard, *Automatic three-dimensional mesh generation by the modified-octree technique*, Internat. J. Numer. Methods Engrg., 20 (1984), pp. 1965–1990.