Isosurface Stuffing: Fast Tetrahedral Meshes with Good Dihedral Angles

François Labelle Jonathan Richard Shewchuk

University of California at Berkeley

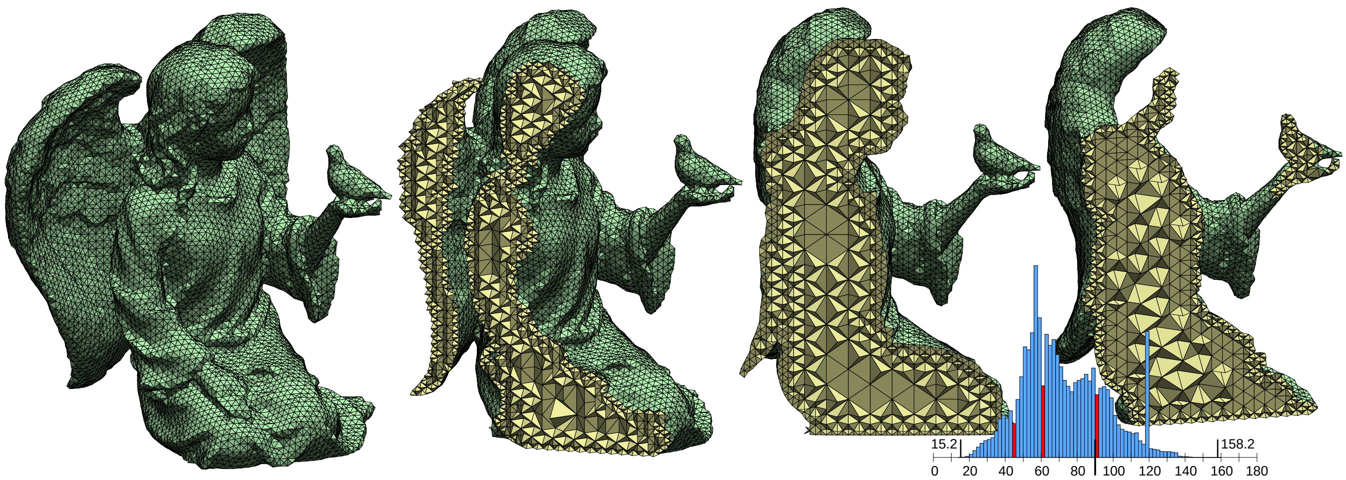

Figure 1: A 134,400-tetrahedron mesh produced by isosurface stuffing, with cutaway views. At the lower right is a histogram of tetrahedron dihedral angles in 2° intervals; multiply the heights of the red bars by 20. (Angles of 45°, 60°, and 90° occur with high frequency.) The extreme dihedral angles are 15.2° and 158.2°. This mesh took 55 seconds to generate on a Mac Pro with a 2.66 GHz Intel Xeon processor, but the mesh generation time was only 642 milliseconds; nearly all the time was spent in the isosurface evaluation code.

Abstract

The *isosurface stuffing* algorithm fills an isosurface with a uniformly sized tetrahedral mesh whose dihedral angles are bounded between 10.7° and 164.8°, or (with a change in parameters) between 8.9° and 158.8°. The algorithm is whip fast, numerically robust, and easy to implement because, like Marching Cubes, it generates tetrahedra from a small set of precomputed stencils. A variant of the algorithm creates a mesh with internal grading: on the boundary, where high resolution is generally desired, the elements are fine and uniformly sized, and in the interior they may be coarser and vary in size. This combination of features makes isosurface stuffing a powerful tool for dynamic fluid simulation, large-deformation mechanics, and applications that require interactive remeshing or use objects defined by smooth implicit surfaces. It is the first algorithm that rigorously guarantees the suitability of tetrahedra for finite element methods in domains whose shapes are substantially more challenging than boxes. Our angle bounds are guaranteed by a computer-assisted proof. If the isosurface is a smooth 2-manifold with bounded curvature, and the tetrahedra are sufficiently small, then the boundary of the mesh is guaranteed to be a geometrically and topologically accurate approximation of the isosurface.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms

Keywords: isosurface, tetrahedral mesh generation, dihedral angle

1 Introduction

Finite element methods are the most popular techniques for numerical simulation of the partial differential equations governing physical phenomena such as fluid flow, mechanical deformation, and diffusion. Most objects worth simulating have complicated shapes. To make them amenable to analysis, modelers decompose them into simple shapes called *elements*, which commonly are tetrahedra. The success of the finite element method depends on the shapes of these tetrahedra—large dihedral angles cause large interpolation errors and discretization errors, robbing the numerical simulation of its accuracy [Jamet 1976; Křížek 1992; Shewchuk 2002], and small dihedral angles render the stiffness matrices associated with the finite element method fatally ill-conditioned [Bank and Scott 1989; Shewchuk 2002].

Mesh generation is a famously difficult problem, in part because of the severe difficulty of forcing a mesh to conform to objects' sharp creases and corners without sacrificing the quality of the tetrahedra. Moreover, conventional mesh generators depend on Delaunay triangulations, iterative algorithms, or numerically sensitive geometric calculations, all of which are costly and tend to preclude remeshing at interactive speeds.

In this paper, we show that generating a high-quality finite element mesh is surprisingly quick and easy for bodies whose boundaries are smooth surfaces—or for circumstances where the modeler is willing to round off the edges a bit (for instance, in videogames). Our mesher is well suited for simulations that dynamically track a smooth liquid boundary, or for modeling elasticity in organic structures such as the tissues of the body. Besides speed, it offers sure numerical robustness and tetrahedron quality, which is particularly reassuring for simulations that need to generate new meshes frequently, perhaps even at frame rates.

Our algorithm is not heuristic; it absolutely guarantees that all the dihedral angles of all the tetrahedra it generates are between 10.78° and 164.74°. To our knowledge, ours is the first tetrahedral mesh generation algorithm of any sort that both offers meaningful bounds on dihedral angles and conforms to the boundaries of geometric domains with complicated shapes. Significant provable bounds on dihedral angles (1° or over) are virtually unheard of outside of space-filling or slab-filling tetrahedralizations.

We assume that the input is a continuous *cut function* $f : \mathbb{R}^3 \mapsto \mathbb{R}$ that implicitly represents the geometric domain to be stuffed with tetrahedra, namely the point set $\{p : f(p) \geq 0\}$. Points where f is negative are outside the domain, and usually should not be meshed—though our algorithm offers the option to create compatible meshes on both sides of the boundary, with a somewhat weaker angle guarantee.

Besides high-quality tetrahedra, isosurface stuffing offers three other guarantees, described in Section 4.2. First, every vertex on the boundary of the mesh lies on the *zero-surface* $\{p : f(p) = 0\}$, presuming that the client can answer a query requesting a zero-surface point that intersects a specified line segment. Second, any point p sufficiently far from the zero-surface is correctly classified, in the sense that it is inside the mesh if $f(p)$ is positive, and outside the mesh if $f(p)$ is negative. (Our notion of "sufficiently far" scales with the tetrahedron size. See Theorem 2.) The only precondition for these two guarantees to hold is that f be continuous. The third guarantee is that if the zero-surface is a smooth 2-manifold with bounded curvature, and if the tetrahedra are sufficiently small, then the boundary of the mesh is homeomorphic to the zero-surface. (We also guarantee ambient isotopy. See Theorem 3.)

These guarantees imply that the triangles on the boundary of the mesh collectively form an accurate approximation of the boundary of the domain. This is important because the boundary is often where the most interesting physics occurs, and is the part of the domain that is most frequently rendered.

Isosurfaces are popular geometric representations in computer graphics. They arise naturally in modeling with implicit surfaces, and are produced by several algorithms for surface reconstruction [Zhao et al. 2001; Ohtake et al. 2003; Shen et al. 2004]. Even for geometric models that do not use isosurfaces, it is usually possible to compute a suitable cut function f by approximating the *signed distance function*, which is the distance from a point p to the boundary of the domain, using a negative distance for points outside the domain. Signed distance functions can be approximated from geometric models or voxel data by fast marching level set methods [Sethian 1996; Osher and Fedkiw 2002]. An algorithm of Bærentzen and Aanæs [2002] for watertight triangular surface meshes computes just the sign, which suffices for our algorithm.

Isosurface stuffing borrows ideas from the popular *Marching Cubes* algorithm [Lorensen and Cline 1987], which triangulates an isosurface (but not its interior). Marching Cubes computes f at the vertices of a cubical grid, and processes the domain cube by cube. When Marching Cubes processes a cube, it outputs triangles that approximate the intersection of the isosurface with that cube. The cubes themselves are not part of the output; they form an invisible *background grid*. The output triangles are generated from a small table of precomputed *stencils*. The vertices of the output triangles depend solely on the values of f at the eight vertices of the cube, and the choice of stencil depends solely on the signs of f at those eight vertices.

Because our algorithm also uses stencils to generate tetrahedra, it is blazingly fast compared to traditional mesh generation algorithms based on Delaunay triangulations or advancing front methods. It is also much easier to implement—we coded our prototype mesher for uniformly sized tetrahedra in two days. Furthermore, our algorithm is numerically bulletproof, as its correctness does not rely on numerically sensitive geometric predicates or any iterative numerical procedure more delicate than using iterated bisection to find a zero of a function of one variable.

A second version of isosurface stuffing creates meshes whose interior tetrahedra are *graded*—they grade from largest at the core to smallest at the surface, as Figure 1 illustrates. This option reduces the amount of computation the finite element method expends on the domain interior while maintaining high resolution near the surface, where modeling errors are most visible. Our algorithm uses a nonstandard octree to create a tetrahedral background grid.

Although our technique can also be used to generate fully graded meshes (whose surface tetrahedra vary in size too), most of whose tetrahedra have excellent quality, we are unable to guarantee dihedral angles better than 1.66° degrees for the worst tetrahedra, so we do not report details here. (But see Section 6 for an example.)

A drawback of our approach is that it does not preserve sharp edges or corners. Guaranteed-quality mesh generation that tightly fits sharp features (without rounding them off) has challenged researchers for over two decades, and will continue to do so, because these constraints impose fundamental difficulties that arguably could never be accommodated by any approach as simple as the method we describe here. For our smooth target domains, however, we make guaranteed-quality meshing easy.

2 Related Work

Tetrahedral mesh generation has an extensive history in both engineering and computer science, so we review here only methods that share similarities with ours or have theoretical guarantees. Octree algorithms were pioneered by Yerry and Shephard [1984]. Fuchs [1998] and Naylor [1999] proposed using the body centered cubic (BCC) lattice and observed the high quality of its tetrahedra. Our algorithms also use octrees and the BCC lattice.

The first provably good mesh generation algorithms—guaranteeing some bounds on the angles of the triangles in a two-dimensional mesh—were the grid-based algorithm of Baker, Grosse, and Rafferty [1988], the Delaunay algorithm of Chew [1989], and the quadtree algorithm of Bern, Eppstein, and Gilbert [1994]. (We adopt the idea of warping a background grid from Bern et al.) Analogous three-dimensional algorithms followed, but it is difficult to guarantee that a mesh's dihedral angles will not be arbitrarily close to 0° or 180°. High-quality space-filling or slab-filling tetrahedralizations are known [Eppstein et al. 2004], but the need to conform to domain boundaries makes meshing substantially harder.

There are nearly a dozen algorithms that provide a theoretical guarantee on the smallest dihedral angle of any tetrahedron, using octrees [Mitchell and Vavasis 2000] or Delaunay triangulations [Chew 1997; Cheng et al. 2000; Li and Teng 2001; Cheng and Dey 2002]. Unfortunately, these algorithms share the characteristic that their provable bounds on dihedral angles are so small—less than 0.1° and probably less than a millionth of one degree for most of them—that the authors don't bother to explicitly derive the numerical bounds.

Delaunay-based meshing algorithms often work well in practice, but the quality of their tetrahedra can be poor at the domain boundary, and virtually none guarantee a meaningful dihedral angle bound (e.g. 1°). The single exception is an algorithm by Labelle [2006], also based on the BCC lattice, that creates a graded mesh that encloses (but does not conform to) the domain boundary, with all dihedral angles between 30° and 135°. That algorithm only accommodates two kinds of constraints: a user may input a set of points which must be vertices of the output mesh, and the tetrahedron sizes cannot exceed a user-specified "sizing function."

Most tetrahedral meshing algorithms take for granted that the input geometry has sharp corners and edges that the output mesh must represent precisely; this constraint introduces tremendous complications that become moot when the boundaries are smooth isosurfaces. Implementing a robust tetrahedral mesh generator for classical input models is a year-long task, whereas our algorithms took us days to code.

There are several prior algorithms for filling smooth surfaces with tetrahedra. Molino, Bridson, Teran, and Fedkiw [2003] begin with a BCC grid, then grade the mesh by using red-green mesh refinement [Bey 1995] to locally adapt tetrahedron sizes as desired. Next, they use an iterative optimization procedure to deform the tetrahedra so that they conform to the boundary. This iterative method is necessarily more expensive than our one-pass stencil-

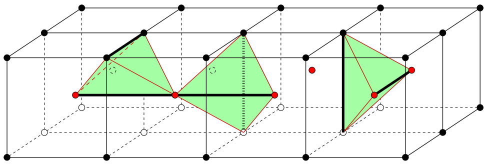

Figure 2: The body centered cubic (BCC) lattice is composed of two staggered cubical grids of vertices. The three tetrahedra illustrated here (which are identical) and copies of them tile space.

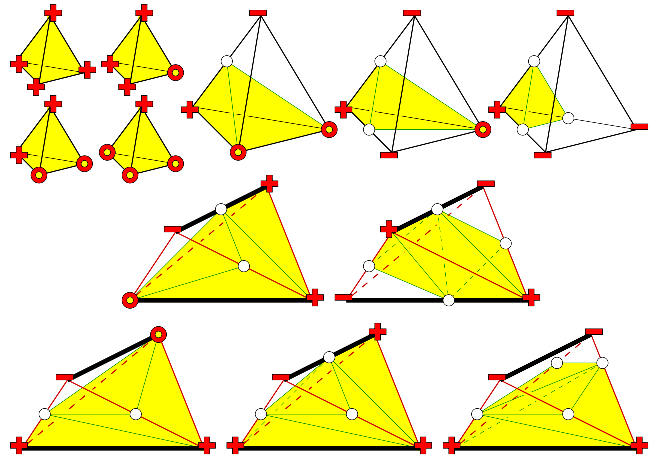

Figure 3: Stencils for isosurface stuffing. Vertices of the BCC grid tetrahedra are labeled with their signs (+, −, 0). Cut points are white, and output tetrahedra are yellow. The seven stencils in the top row apply in all rotations and reflections, and their edges can be matched arbitrarily with the long and short edges of the BCC grid. For the remaining five stencils, the long edges of the BCC grid are depicted as thick and black; the short edges are red. For the three stencils in the bottom row (wherein the bottom long edge has both endpoints positive), the Parity Rule applies and may require a stencil to be reflected. The bottom five stencils apply in all rotations and reflections (left to right or front to back) that observe the Parity Rule and correctly match the edge colors.

based approach. Molino et al. obtain dihedral angles between 13° and 156° for the meshes they use to illustrate their algorithm, but they offer no guarantees.

The Delaunay refinement algorithm of Oudot, Rineau, and Yvinec [2005] relies on a technique called *sliver exudation* [Cheng et al. 2000] to remove poor tetrahedra from meshes and achieve good dihedral angles. Edelsbrunner and Guoy [2001] demonstrate that sliver exudation usually removes most of the bad tetrahedra from a Delaunay mesh, but rarely all; tetrahedra with dihedral angles less than 1° sometimes survive, and in most of their examples a few dihedral angles less than 5° survive. Alliez, Cohen-Steiner, Yvinec, and Desbrun [2005] claim (without mathematical guarantees) that their variational meshing algorithm produces no poor tetrahedra in practice. Neither Oudot et al. nor Alliez et al. specify the dihedral angles they achieve, so we cannot compare ourselves with them on that basis.

Freitag and Ollivier-Gooch [1997] achieve results better than Edelsbrunner and Guoy through optimization-based smoothing and topological transformations, but again dihedral angles less than 1° sometimes survive, and in many examples dihedral angles under 10° survive.

3 Isosurface Stuffing: Uniform Tetrahedra

The isosurface stuffing algorithm, like the Marching Cubes algorithm, employs a space-tiling background grid to guide the creation of a mesh. The *body centered cubic (BCC) lattice*, which describes the structure of many crystals, is the union of two point grids,

$$\mathrm{BCC} = \mathbb{Z}^3 \cup \left(\mathbb{Z}^3 + (\tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2})\right),$$

where $\mathbb{Z}^3$ is the grid of points with integer coordinates, and $\mathbb{Z}^3 + (\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$ is a copy of that grid shifted so that each point falls at the center of a cube of the original grid.

The Delaunay triangulation of these points, illustrated in Figure 2, is a tetrahedral mesh that we call the *BCC grid*. The BCC grid is composed of identical tetrahedra that are of excellent quality, having edge lengths 1 and $\sqrt{3}/2$, and dihedral angles 60° and 90°. This space-filling tetrahedron was noted by Sommerville [1923]. The fact that all the BCC grid tetrahedra are identical simplifies both implementing our algorithm and proving its correctness.

We fill a zero-surface with uniformly sized tetrahedra in four steps. All but the third step are borrowed (with changes) from Marching Cubes.

1. Choose a subset P of the BCC lattice. P should include every lattice point where the cut function f is nonnegative, and every lattice point connected by an edge of the BCC grid to a lattice point where f is positive. Compute and store the value of f at each lattice point in P.
2. For each edge of the BCC grid with both endpoints in P, if one endpoint is positive (meaning "inside") and one is negative (meaning "outside"), then compute or approximate a *cut point* where the edge crosses the zero-surface.
3. For each lattice point $q \in P$, check for the presence of cut points on the fourteen grid edges that adjoin q. If one of these cut points c is too close to q, we say that c *violates* q. If any cut point violates q, *warp* the grid by moving q to a cut point that violates q. (We usually choose the nearest violating cut point, but our guarantees do not depend on it. Technically, q is no longer a lattice point, but we still call it one.) The effect is to snap q onto the zero-surface. Change q's value to zero. Discard all cut points on the edges adjoining q, because those edges no longer have both a positive endpoint and a negative endpoint. Because we process every lattice point in P *sequentially* in this manner, no cut point adjoining q can subsequently cause another lattice point to move.
4. For each BCC grid tetrahedron that has at least one vertex with a positive value, fill the tetrahedron (which might be warped) with a stencil of 1–3 precomputed tetrahedra. Output these tetrahedra. Figure 3 depicts the stencils. The choice of stencil depends on the signs of the four vertices of the BCC grid tetrahedron.

We distinguish between three kinds of points and two kinds of tetrahedra. *Cut points* (where BCC grid edges cross the zero-surface) and *lattice points* may or may not become *output vertices*. Likewise, some of the *output tetrahedra* that comprise the final mesh are distinct from the *BCC tetrahedra* of the background grid.

3.1 Steps 1 and 2: Lattice and Cut Points

For a general continuous cut function f, the first step is technically impossible, because there is an infinite number of lattice points to test. Every isosurface-processing algorithm faces the problem that it is difficult to find all the components of $f(p) = 0$—and it is generally impossible if f is a black box that can only be evaluated at individual points. A practical way to find the points in P is to begin with several "seed" points known to be in the domain, then find the rest by depth-first search on the edges of the BCC grid. This method may fail to find the entire domain if the lattice is not fine enough to resolve the narrower portions of the domain, or if the domain has a connected component that does not contain a seed point. For ease of programming, our prototype mesher evaluates f at every lattice point in a user-specified bounding box, but this is costly when the volume of the bounding box is much greater than the domain volume. (Our timings reflect that.)

For the second step, we assume that the geometric modeler that

Guaranteed bounds To optimize the ...	minimum dihedral angle	maximum dihedral angle	minimum plane angle	maximum plane angle	minimum exposed plane	maximum exposed plane	use these parameters α_{long}	α_{short}
maximum dihedral angle, unsafe	8.9716	***158.7403**	11.9072	150.9944	12.0162	147.6786	0.26649	0.36918
minimum dihedral angle, unsafe	***10.7843**	164.7373	9.0454	154.9845	9.0454	154.9845	0.28511	0.39882
maximum dihedral angle, safe	9.0551	**160.5331**	8.7614	155.7053	8.7614	155.7053	0.24999	0.40173
minimum dihedral angle, safe	**9.3171**	161.6432	7.7810	158.2252	7.7810	158.2252	0.24999	0.41189
... with ordered warping	**9.7766**	163.5685	10.5695	149.7137	15.1645	138.1929	0.24999	0.42978
max dihedral, double-sided, safe	6.4917	**164.1013**	8.8535	157.8278	13.0689	145.1886	0.21509	0.35900
min dihedral, double-sided, safe	**7.6872**	168.0481	9.2237	155.0594	9.2237	154.5340	0.22383	0.39700
... with ordered warping	**7.8653**	168.0572	9.5400	154.6644	14.4726	135.7164	0.22385	0.40501
max exposed plane angle, safe	5.3440	163.8969	6.2646	158.2960	11.8387	***124.9195**	0.23926	0.27376
... with ordered warping	5.8017	162.1673	7.2694	158.0368	12.1108	***124.0867**	0.23463	0.29505
min exposed plane angle, unsafe	n/a	n/a	10.4741	†149.6794	***15.1285**	*149.5205	0.36378	0.33951
min exposed plane angle, safe	7.8390	160.5447	10.4213	153.7863	**13.5241**	144.1259	0.24999	0.35464
... with ordered warping	7.4904	169.1465	9.2685	†145.4921	**16.4299**	144.9032	0.23573	0.5

Table 1: Choices of α_{long} and α_{short} that optimize the minimum or maximum dihedral angles, or the minimum or maximum plane angles of triangles exposed on the boundary of the mesh. Columns list the extremal dihedral angles, plane angles of triangular faces (including triangles in the mesh interior), and plane angles of triangular faces exposed on the boundary. Rows marked "safe" indicate values for which the tetrahedra are guaranteed not to overlap each other, even if the background grid is not fine enough to resolve the surface correctly. Rows marked "double-sided" are for guaranteeing good quality when meshing both sides of an isosurface with compatible tetrahedra. *Ordered warping* is described in Section 3.2. Asterisks and daggers are explained in Section 5. The bottom five rows are of interest mainly for surface meshing; see Section 6. All angle bounds have been computer-verified to be strictly correct as written and tight to within 0.0001°.

defines the cut function f can answer a query asking for a point where a line segment intersects the zero-surface. Our prototype implementation does this by iterative bisection, which can approximate the cut point to arbitrary accuracy, even for a black box function f. If f is expensive to evaluate, one could estimate the cut point by linear interpolation along the edge, at the cost of losing all the guarantees about geometric and topological fidelity, and retaining only the angle guarantee.

3.2 Step 3: Warping the Background Grid

The third step uses a simple rule to decide if a lattice point is violated. If a cut point c lies on a grid edge e, and the distance between c and an endpoint v of e is less than α times the length of e, then c violates v; so v must be snapped to the isosurface, assigned a value of zero, and purged of adjoining cut points—*unless* the other endpoint of e gets snapped first, eliminating c.

BCC grid edges come in two lengths, and we use a different value of α for each, chosen by experimentation. Several options are summarized in Table 1. In the table, α_{long} is the coefficient for the longer, axis-aligned edges, which we call the *black edges*; and α_{short} is the coefficient for the shorter, diagonal edges, which we call the *red edges*. The angle bounds are derived with a computer-assisted proof, discussed in Section 4.1.

The order in which we process and warp the lattice points affects the final mesh, but it does not affect most of our guarantees. The exceptions are the four rows of Table 1 wherein an angle bound is improved by *ordered warping*, in which we use the following algorithm to ensure that a lattice point never warps along an edge toward a neighboring vertex that will also be warped.

while some negative lattice point q^- is violated by a cut point on
    an edge adjoining an *unviolated* positive lattice point
  Warp q^- to a violating cut point on such an edge
while some positive lattice point q^+ is violated
  Warp q^+ to a violating cut point

When this algorithm terminates, no violated lattice points survive, because if a negative lattice point is still violated when the first loop ends, the cut points that violate it are discarded when the second loop warps the violated positive lattice points. The disadvantage of this ordering is that it makes a parallel or streaming implementation difficult, because the dependences of the first loop can cascade long distances. When a negative lattice point warps, the cut points that adjoin it disappear, which may cause a formerly violated positive lattice point to become unviolated, thereby forcing a different negative lattice point to warp toward it, and so on *ad infinitum*. It is sometimes better to settle for a slightly weaker angle bound so that the lattice points can warp in an arbitrary order.

3.3 Step 4: Triangulating the Background Grid

The fourth step generates the output tetrahedra specified by the stencils depicted in Figure 3. We store the stencils in a table, indexed by the signs of the vertex values (positive, negative, or zero). Some stencils generate two or three output tetrahedra, to respect surviving cut points on the grid edges. Symmetry reduces the number of distinct cases from 81 to the 12 illustrated. In accounting for symmetry, note that black edges are not always interchangeable with red ones—some stencils offer better quality than others in particular circumstances, and not all stencils meet compatibly face-to-face.

Some cases admit more than one possible stencil because the isosurface truncates some BCC grid triangles, creating quadrilateral faces, each of which we bisect into two triangles. Each stencil's tetrahedra are determined by the choice of diagonal used to bisect each quadrilateral. To choose diagonals, we use two disambiguation rules, designed to produce high-quality output tetrahedra.

Observe that every BCC grid triangle has one black edge and two red ones, so each quadrilateral has either a whole black edge or a truncated one. We bisect a quadrilateral with a truncated black edge by choosing the diagonal that adjoins the cut point where the black edge was truncated. The stencils in Figure 3 obey this rule.

If a quadrilateral face has a whole black edge (and two truncated red edges), we break symmetry by using the following *Parity Rule* to choose a diagonal. Let a and b be the endpoints of the black edge, and let c and d be the cut points where the red edges are truncated, labeled so the quadrilateral's diagonals are ac and bd. Because of the geometry of the BCC lattice, either a has an even number of coordinates that are greater than c's corresponding coordinates and b has an odd number of coordinates greater than d's coordinates, or vice versa. If a and b lie on the cubical lattice $\mathbb{Z}^3$ (the black points in Figure 2), we choose ac if a has an odd number of coordinates greater than c's coordinates; we choose bd if the number is even. This rule allows us to use the bottom right stencil in Figure 3, which has better quality than alternatives. Observe that the stencil has not one but two of these quadrilateral faces, front and back, and the two corresponding diagonals do not share an endpoint.

If a and b lie on the cubical lattice $\mathbb{Z}^3 + (\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$ (the red points in Figure 2), we reverse the rule and choose ac if a has an even number of coordinates greater than c's coordinates. This reversal makes it possible to mesh both sides of an isosurface compatibly with the same stencil. To implement the Parity Rule, we occasionally have to reflect one of the stencils in the bottom row of Figure 3 after looking it up.

Every BCC tetrahedron with no negative (outside) vertex becomes an output tetrahedron, *except* perhaps a BCC tetrahedron with all four vertices labeled zero. Such a *quadruple-zero tetrahedron* is ambiguous; it is not clear whether to treat it as if it is inside or outside the domain. Because all four vertices of this tetrahedron are warped, the most aggressive choices for the α parameters in Table 1 (those labeled "unsafe") may cause it to be *inverted* (turned inside-out, with negative signed volume)—even if the isosurface is nearly flat. Parameters are marked "safe" if our computer-assisted proof code (Section 4.1) guarantees that no BCC tetrahedron can become inverted. Inverted BCC tetrahedra do not necessarily hurt the mesh or imply that the lattice is insufficiently fine to resolve the surface. In rare cases, though, they might cause a few output tetrahedra to have mutually intersecting interiors. This danger is avoided if the zero-surface is a smooth manifold with bounded curvature and the BCC grid is sufficiently fine to resolve it.

We offer four options for handling quadruple-zero tetrahedra. The simplest is to discard them all. In some applications this is mandatory; Molino et al. [2003] observe that for modeling large mechanical deformations, a tetrahedron with all four vertices on the boundary (or an edge that extends through the mesh interior but has both vertices on the boundary) is easily crushed and can ruin a simulation.

For applications that can tolerate tetrahedra with all four vertices on the boundary, we observe that we can often improve a mesh's surface fidelity by heuristically retaining some of the quadruple-zero tetrahedra. We discard the quadruple-zero tetrahedra that are inverted or whose dihedral angles are poor, and we argue that these tetrahedra are too flat to have much effect on the surface fidelity. Of the nicely shaped survivors, we choose to retain a tetrahedron if all four of its faces adjoin output tetrahedra (not of the quadruple-zero kind), and to discard a tetrahedron if none of its faces does. This heuristic prevents spurious "bubbles" from appearing in the mesh. For the remaining tetrahedra, the decision is made by an evaluation of the cut function f at each tetrahedron's centroid. This heuristic tends to reduce divots on a poorly-resolved surface.

Both these options guarantee the angle bounds in Table 1, by discarding quadruple-zero tetrahedra that fail to meet them.

A third option is to change the warping parameters so that it is safe to output every quadruple-zero BCC tetrahedron, at the cost of weakening the dihedral angle bounds. The options labeled "double-sided" in Table 1 achieve this; the bounds given in those rows of the table include BCC tetrahedra with all four vertices warped (whereas the other dihedral angles in the table do not take them into account). These options make it possible to mesh both sides of an isosurface with compatible tetrahedra. To mesh the exterior of a domain, simply swap the $+$ and $-$ signs in Figure 3. Again, heuristics can classify each quadruple-zero tetrahedron as being inside or outside the domain.

A fourth option is to observe that if the isosurface is a smooth manifold with bounded curvature, and the BCC grid is sufficiently fine, then the boundary of the mesh will be a geometrically and topologically accurate approximation of the zero-surface. (See Theorems 2 and 3 in Section 4.2.) Any good-quality quadruple-zero BCC tetrahedron is a sign that the lattice does not adequately resolve the surface, and that one might start over with a finer lattice. (But remember, a *poor*-quality quadruple-zero BCC tetrahedron is *not* a sign of insufficient resolution; just discard it.)

3.4 Mesh Examples

Figure 4 depicts two meshes whose dihedral angles lie between 14° and 158°. More than half the dihedral angles in each mesh are 60° or 90°. (Note the red bars, which represent twenty times more angles than blue bars of the same height.) It took 25.2 seconds to generate the 131,259-tetrahedron *Whirled White Web* mesh on a Mac Pro with a 2.66 GHz Intel Xeon processor, of which 644 milliseconds were mesh generation time (the rest being used to evaluate the cut function f). The 32,853-tetrahedron Stanford dragon mesh took 24.5 seconds, of which 172 milliseconds were for mesh generation.

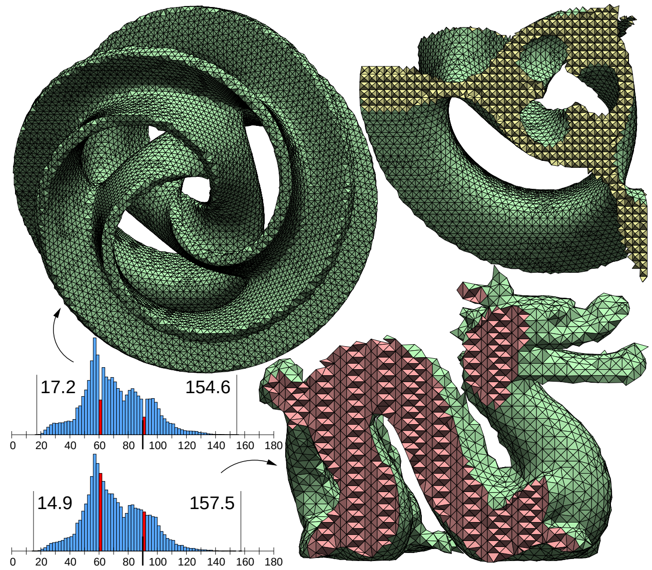

Figure 4: Meshes of uniformly sized tetrahedra produced by isosurface stuffing with $\alpha_{\text{long}} = 0.28511$ and $\alpha_{\text{short}} = 0.39882$. *Whirled White Web* is by courtesy of Carlo Séquin. Histograms tabulate the dihedral angles in 2° intervals; multiply the heights of the red bars by 20.

Our mesh generation timings are misleadingly slow, because our prototype implementation evaluates the cut function at every lattice point in a large box, and typically inspects twenty empty BCC tetrahedra for every BCC tetrahedron that intersects the domain. A more efficient implementation would never stray far from the domain. To get a sense of how much faster this would be, we performed a comparison of our prototype implementation against Pyramid, our fast Delaunay-based meshing code. We used a dense domain, intersecting about half the BCC grid tetrahedra, for which evaluating the cut function took only 20% of the running time. Our implementation generates about 510 tetrahedra per millisecond, whereas the Delaunay mesher generates about 157 tetrahedra per millisecond. This discrepancy in running time occurs because isosurface stuffing does far fewer numerical calculations and requires less complicated data structures.

4 Guarantees

Isosurface stuffing offers three mathematical guarantees. First, the tetrahedra it produces have good angles. Second, the boundary of the mesh it produces is close to the zero-surface. Third, if the zero-surface is a smooth manifold with bounded curvature and the BCC grid is fine enough, then the mesh boundary is homeomorphic to the zero-surface. We suggest proofs of these facts here.

4.1 Dihedral and Plane Angles

Theorem 1. *The bounds in Table 1 on the angles produced by isosurface stuffing are correct as written (i.e., lower bounds are rounded down; upper bounds are rounded up). They are tight to within a margin of* 0.0001°*—we can exhibit cut functions that cause these angles to appear.* ■

Our angle guarantees were obtained through a computer-assisted proof. There is only a finite number of stencils to test; but there is an infinite number of locations where a cut point might be placed,

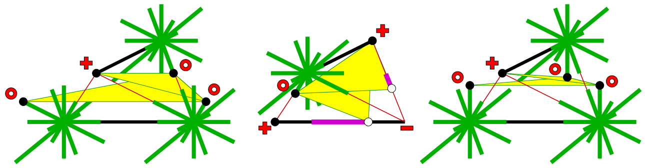

Figure 5: Some of the limiting cases in which a dihedral angle of 10.7843° arises (where the two yellow triangles meet). Warped background vertices must lie on their green asterisks. Cut points must lie on the magenta segments.

or destinations a lattice point might be warped to. Although a proof by hand might be possible through a (horrendous) case analysis, we verified the angle bounds by writing a program that breaks the space of possible tetrahedron configurations into a finite number of subspaces that can be verified by interval arithmetic.

The analysis begins with the observation that each edge of the BCC grid has a central part (from a fraction of α to $1-\alpha$ of its length) where a cut point triggers no warping, and two peripheral parts where a cut must trigger warping. The *asterisk* of a grid vertex is the union of the peripheral parts adjacent to the vertex, as illustrated in Figure 5. We depend upon the following facts.

- A warped vertex lies on its asterisk, and its value (in choosing a stencil) is zero.
- Stencil vertices labeled $+$ or $-$ are not warped.
- A cut point cannot lie on an edge adjoining a warped vertex.
- Two vertices that share an edge of the BCC grid cannot both warp toward each other along that edge. (The first one to warp eliminates the cut point between them.)
- If ordered warping is used, a vertex cannot warp along an edge whose other endpoint also warps.

To divide the configuration space into cases, we consider each tetrahedron in each stencil. Each cut point's location is described by a single parameter (its position along the segment). Each asterisk is composed of seven segments, so a vertex labeled zero is represented by seven separate cases. In particular, the quadruple-zero tetrahedron requires the enumeration of 7^4 cases. In each case, the position of each vertex is fixed or described by one parameter.

Suppose a tetrahedron has vertices a, b, c, and d, each of which is constrained to lie on a segment (different for each vertex), and suppose that the four vertices cannot be coplanar under that constraint. Then it is easy to prove that the dihedral angle at edge ab can be minimized (or maximized) with c and d lying at endpoints of their respective segments. (For intuition, imagine opening an infinitely large door that is constrained to intersect a line segment floating in space.) Therefore, we only need to consider cases wherein each of c and d lies at one of the two endpoints of the segment it is constrained to lie on. We reduce every case to cases that have at most two continuously varying parameters.

Our program verifies the dihedral angle bounds by subdividing this two-dimensional parameter space with a quadtree, and estimating the worst-case angles for each quadrant by interval arithmetic. When an interval does not prove our conjectured bound to within a specified tolerance, the program subdivides the quadrant into smaller quadrants, and tries again on those. To verify plane angle bounds, we subdivide a three-dimensional parameter space with an octree. By this means, we have verified all the angle bounds in Table 1 as stated in Theorem 1. The cases that limit our bound on the smallest dihedral angle to 10.7843° degrees appear in Figure 5.

4.2 Geometric and Topological Fidelity

A mesh generation algorithm needs more than good elements; it also needs to produce a mesh that is a reasonable facsimile of the domain it is supposed to represent. It is clear from the algorithm that every vertex on the boundary of the mesh is a cut point or is labeled zero, and therefore lies on the isosurface. By inspection of the stencils, we see that isosurface stuffing never connects a vertex inside the isosurface (labeled $+$) to one outside (labeled $-$), so the mesh respects the isosurface.

Furthermore, the boundary of a mesh produced by isosurface stuffing approximates the zero-surface by a one-sided Hausdorff bound: every point on the mesh boundary is close to the zero-surface. If the background grid is a BCC lattice scaled by a factor c, then as $c \to 0$, the greatest distance between a mesh boundary point and its nearest neighbor on the zero-surface converges to zero. The following theorem confirms this, and is more general.

Theorem 2. *Suppose isosurface stuffing meshes a continuous cut function f. (It does not matter which quadruple-zero tetrahedra become output tetrahedra.) For any point p in space, if p lies in an output tetrahedron but $f(p) < 0$ (implying that p should lie outside the mesh), or if p does not lie in a output tetrahedron but $f(p) > 0$, then p is within a distance no greater than $\omega = \max\{\sqrt{\alpha_{\text{long}}^2 + \alpha_{\text{long}} + 5/16}, \frac{1}{2}\sqrt{3\alpha_{\text{short}}^2 + 3\alpha_{\text{short}} + 5/4}\}$ from the isosurface. (The number ω applies for the unscaled BCC lattice. If the BCC lattice is scaled by c, the number is ωc.)*

Proof. Let t be the tetrahedron that contains p, or (if p is outside the mesh) the tetrahedron that would contain p if the domain exterior were meshed as well. All four vertices of t have different signs than p (opposite or zero), so the distance from p to the isosurface cannot exceed the distance from p to the nearest vertex of t. The latter quantity is maximized when p lies at the centroid of a BCC tetrahedron, all four of that tetrahedron's vertices are labeled zero, and all four vertices are warped as far from p as possible. ∎

The Hausdorff distance discussed above is one-sided because, if we can only access the cut function f by pointwise probing, it is impossible to guarantee that every point on the zero-surface is close to the mesh boundary. In general, an isosurface can have extremely tiny components that are unlikely to be found by pointwise probing. However, if the isosurface is a smooth manifold with bounded curvature and the BCC grid is sufficiently fine, then we can guarantee that our Hausdorff distance bound is two-sided and that the mesh is topologically accurate.

Theorem 3. *Suppose isosurface stuffing uses a background BCC grid scaled by c to mesh a continuous cut function f whose zero-surface is a smooth 2-manifold. Let $d_M > 0$ be the shortest distance from a point on the zero-surface to a point on the medial axis of the zero-surface. (Thus $1/d_M$ is an upper bound on the curvature of the zero-surface.) If $d_M > \omega c$, with ω defined as in Theorem 2, then every point on the zero-surface is within a distance of ωc from the mesh boundary. Moreover, if c/d_M is sufficiently small, then the boundary of the mesh is homeomorphic to the zero-surface, and there is a continuous deformation of space that carries the zero-surface to the mesh boundary. (I.e., there is an ambient isotopy from the identity map on the zero-surface to the homeomorphism that maps the zero-surface to the mesh boundary.)*

Proof sketch. Suppose $d_M > \omega c$. For each point p on the zero-surface, let $i(p)$ be the point found by moving a distance infinitesimally greater than ωc from p along the inward-facing normal to the zero-surface, and let $o(p)$ be the point found by moving the same distance outward. The *isotopy segment* $s(p)$ is the segment with endpoints $i(p)$ and $o(p)$; it is perpendicular to the zero-surface at p. Imagine an isotopy segment for each point on the surface. None of these isotopy segments intersect the medial axis or each other, and their union forms an envelope around the zero-surface.

We construct a continuous map $m : \mathbb{R}^3 \mapsto \mathbb{R}^3$ that maps each isotopy segment to itself, and maps each point p on the zero-surface to the point where its isotopy segment $s(p)$ intersects the boundary of the mesh. Our goal is to show that each isotopy segment intersects the mesh boundary in one and only one point. (Every point on the mesh boundary intersects exactly one isotopy segment, because by Theorem 2, every point on the mesh boundary is within a distance

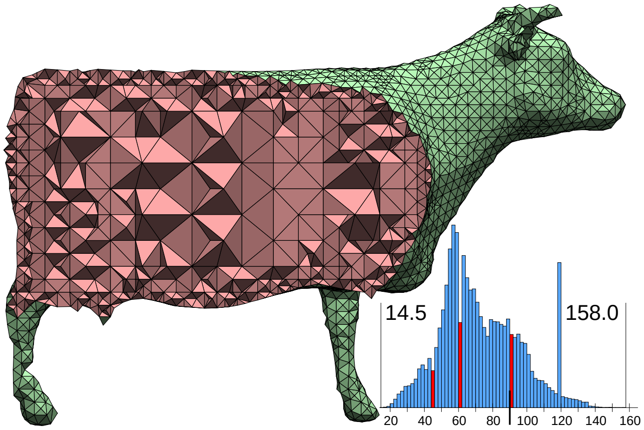

Figure 6: Cutaway view of a 42,053-tetrahedron mesh whose elements are uniformly fine on the surface but grade to coarse in the interior. Produced with $\alpha_{\text{long}} = 0.28511$ and $\alpha_{\text{short}} = 0.39882$. A histogram of dihedral angles in 2° intervals appears at lower right; multiply the heights of the red bars by 20. This mesh took 4.9 seconds on a Mac Pro with a 2.66 GHz Intel Xeon processor, of which 403 milliseconds were mesh generation time.

of ωc from the zero-surface.) It follows that m induces a homeomorphism between the zero-surface and the mesh boundary, and we have an ambient isotopy by linearly interpolating between the identity map and m.

By Theorem 2, for any point p on the zero-surface, $i(p)$ lies in a tetrahedron and $o(p)$ does not, so $s(p)$ intersects at least one point on the mesh boundary. Thus, the bound on Hausdorff distance implied by Theorem 2 is two-sided when $d_M > \omega c$.

The hard part is showing that each isotopy segment intersects *only* one point—loosely speaking, that the mesh boundary does not have wrinkles or extraneous components. Suppose for the sake of contradiction that an isotopy segment $s(p)$ intersects several points on the mesh boundary. Then on a "walk" from $i(p)$ to $o(p)$ one reenters the mesh at least once, implying that some boundary triangle t faces the "wrong" way relative to $s(p)$.

Because t is a boundary face, all three of its vertices lie on the zero-surface, and it is a face of a high-quality output tetrahedron h. Let v be the vertex of t opposite t's longest edge. The two balls of radius d_M tangent to the zero-surface at v have interiors that do not intersect the zero-surface, so no vertex of t lies inside them. The size of t is proportional to the BCC grid size c, so if c/d_M is sufficiently small, the balls constrain t to be nearly parallel to the tangent plane (see Theorem 5 of Amenta, Choi, Dey, and Leekha [2002]). Because h has good quality, its fourth vertex must lie inside one of the two balls, so the vertex is labeled $+$ (rather than 0) and the ball lies entirely within the domain (as its interior does not intersect the isosurface). Thus the isotopy segment $s(v)$, which is collinear with the ball centers, is correctly oriented relative to t (i.e., $i(v)$ is on the same side of t as h). So are all the isotopy segments that intersect t, because the curvature of the zero-surface is bounded (see Lemma 3 of Amenta and Bern [1999]). We omit details. ■

5 Graded Interior Tetrahedra

For many applications in rendering and engineering, the need for accuracy is greatest near the surface of the domain. This section addresses the goal of creating a graded mesh that has uniformly fine (small) elements on its boundary, where accuracy is most crucial, but increasingly coarse elements deeper in its interior, as illustrated in Figures 1 and 6. By reducing the number of tetrahedra in the mesh, we reduce the finite element method's computation time. (Ideally, we would like to allow element sizes to grade on the boundary too, but we have not been able to achieve satisfying dihedral angle bounds. See Section 6.)

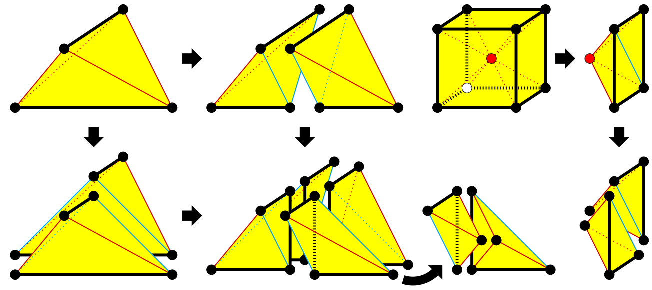

Figure 7: Graded background grids consist of BCC tetrahedra, bisected BCC tetrahedra, quadrisected BCC tetrahedra, and half-pyramids.

We replace the BCC grid with a graded background grid composed of four kinds of tetrahedra, illustrated in Figure 7. In addition to the BCC tetrahedron, we use a bisected BCC tetrahedron, created by splitting a BCC tetrahedron at the midpoint of one of its long edges, and a quadrisected BCC tetrahedron, created by splitting a bisected BCC tetrahedron along the surviving long edge. These tetrahedra are nearly as well shaped as the BCC tetrahedron, having dihedral angles of 45°, 60°, and 90°. The fourth tetrahedron kind is a half-pyramid. A cube can be divided into six pyramids—one for each face of the cube—with their apices meeting at the center of the cube, as illustrated. Each pyramid can be bisected by a diagonal into two half-pyramids, which have dihedral angles of 45°, 60°, 90°, and 120°. Half-pyramids can also be obtained by bisecting the red edge of a quadrisected BCC tetrahedron.

In addition to the black and red edges of the BCC grid, bisection and quadrisection also introduce a new kind of diagonal edge we call *blue edges*. Bisection and quadrisection also split black edges into shorter black edges, and quadrisection creates a new black edge (so colored because it is axis-aligned).

We use an octree to help create a graded tetrahedral background grid using these four kinds of tetrahedra. The vertices of the background grid will be corners and centers of the octants (cubes) in the octree. As in the BCC grid, one tetrahedron can span two octants. If the octree were refined to the same depth everywhere, the background grid would be composed of BCC tetrahedra, except at its boundary. However, we try to refine the octree as little as possible, to minimize the number of tetrahedra.

Our octree is not quite the usual one. In a classical octree, when an octant is refined, it is divided into eight octants of half the length—its *children*. For better grading, our octree is adjusted so that an octant can be refined without creating all eight children—rather, we can choose to create any subset of an octant's eight possible children, and thus target refinement more precisely.

Figure 8 shows how to bridge between BCC grids whose edge lengths differ by a factor of two. Not only can our four tetrahedron kinds bridge between an octant and an adjoining octant twice the size; they can bridge between an octant and its parent (unlike with most octree meshing algorithms). On the coarse side, we use quadrisected BCC tetrahedra. On the fine side, we use half-pyramids. (Bisected BCC tetrahedra are also needed, because some octants have extra vertices at the midpoints of some of their edges.)

A useful intuition is to observe that, given a BCC grid, we can bisect any arbitrary subset of black (long) edges independently, yielding a mesh of our first three tetrahedron kinds. This fact is germane in the transition regions, where smaller tetrahedra force some larger ones to be bisected.

The main idea of our meshing algorithm is to ensure that only BCC tetrahedra will intersect the isosurface, so most of the angle bounds in Table 1 apply to our graded meshes as well as our uniform ones. Tetrahedra of the other three kinds might still have their vertices warped, however. It is a straightforward extension of our computer-assisted proof, described in Section 4.1, to show that these tetrahedra will not be warped enough to violate the angle

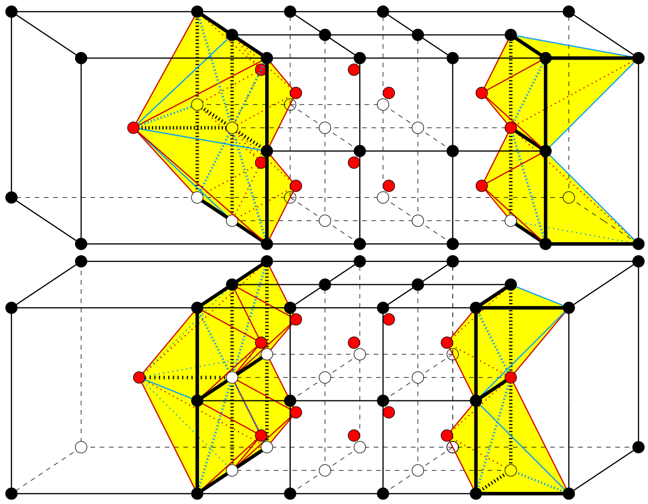

Figure 8: Background tetrahedra used to bridge two levels of the octree, viewed from two different angles. Cube centers are red.

bounds in Table 1, except for the two bounds marked by daggers, which deteriorate to 158.1918° and 147.0470°, respectively.

To start the algorithm, a user selects an approximate tetrahedron size by specifying the width w of the leaf octants of the octree. Conceptually, the algorithm partitions space into an infinite grid of cubes having width w. Each cube has nine *probe points*: its center and its eight vertices. The *sign* of a probe point is the sign of the cut function f at that point. Ideally, we wish to find every cube that the isosurface passes through. Practically, we search for every cube that has at least one nonnegative probe point and one nonpositive probe point, as illustrated in Figure 9(a). (This includes every cube with a zero probe point.) If the isosurface is connected, and the grid is fine enough to resolve it accurately, and we can find one such cube, then we can find the others by depth-first search through the space of cubes (using a hash table to store the cubes, keyed on their coordinates). This is a standard technique, sometimes called *continuation*; see Bloomenthal [1994] for details.

These cubes will be among the leaves of the octree. To ensure that only BCC tetrahedra will intersect the isosurface, we gather additional cubes according to a *Continuation Condition*.

> If a leaf octant o has a square face s with at least one nonpositive vertex and one nonnegative vertex (a zero vertex counts as both), then we must create a leaf octant (the same size as o) adjoining the other side of s. Moreover, if a leaf octant o has a corner vertex v whose sign is opposite the sign of o's center point, or if either sign is zero, then we must create the three leaf octants incident on v that share a square face with o.

Next, to obtain the angle guarantees marked by asterisks in Table 1, we sometimes must create a few additional leaf octants to prevent half-pyramids from becoming overly deformed by warping. We determine which lattice points are violated by cut points. If a leaf octant has a violated center point and a face s with two opposite violated corners, we create a leaf octant adjoining the other side of s. The goal is to ensure that the kind of triangle shared by two half-pyramids, having two red edges and one blue edge, never has all three vertices warped. This step is unnecessary to achieve the angle bounds not marked by asterisks.

Next, we create an octree whose octants are the leaf cubes and their ancestors. We keep the octree as sparse as possible by taking advantage of the fact that an octant can have a child without having eight children. See Figure 9(b).

We cannot bridge directly between two octants whose lengths differ by more than a factor of two while maintaining high element

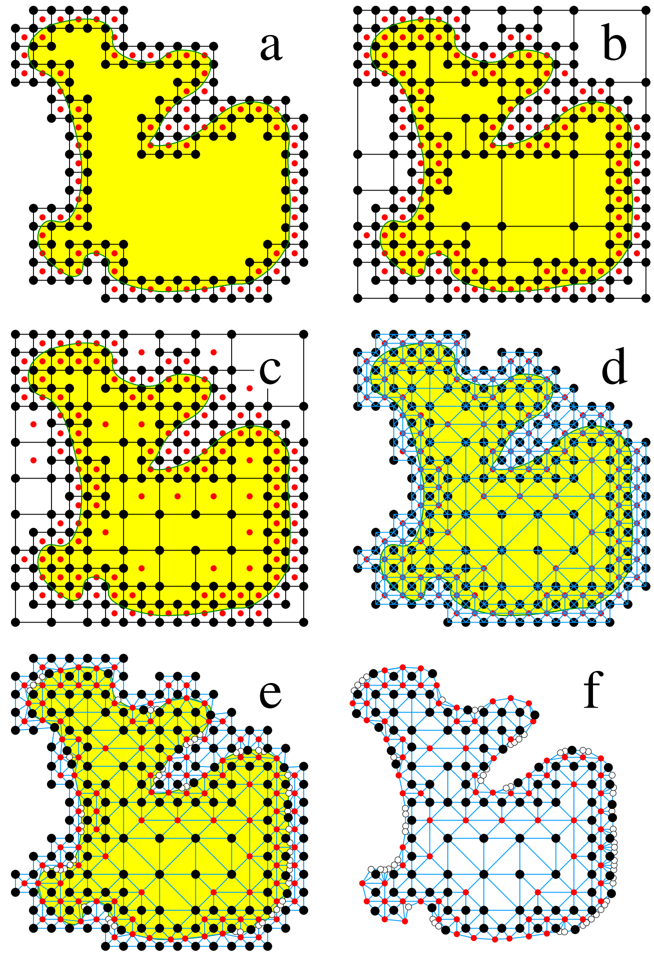

Figure 9: A two-dimensional illustration of the (three-dimensional) algorithm. (a) Cells intersecting the isosurface. (b) After enforcing the Continuation Condition and building an octree. (c) After enforcing the Weak Balance Condition. (d) The background grid. (e) After introducing cut points and warping the vertices. (f) The final mesh.

quality, so the next step is to impose the following *Weak Balance Condition*, illustrated in Figure 9(c).

> If an octant o intersects an edge e of the octree for which the length of e is strictly less than half o's width, then we must create every child of o that intersects the interior of e. (Note that e could be on the boundary or in the interior of o—in the latter case, it would be an edge of a grandchild of o.)

The algorithm in Figure 10 converts a balanced octree to a background grid, as illustrated in Figure 9(d).

Once we have a background grid, our algorithm for constructing an internally graded mesh is almost identical to the uniform meshing algorithm. We compute the value of the cut function f at every vertex of the background grid. (Most of these values were already computed to enforce the Continuation Condition.) We compute cut points and warp the background grid as described in Section 3 and illustrated in Figure 9(e). (The Continuation Condition ensures that blue edges are never cut.) We use the stencils in Figure 3 to create output tetrahedra from the BCC tetrahedra in the background grid. Every background tetrahedron of the other three kinds becomes an output tetrahedron if it has at least one positive vertex. See Figure 9(f) for a two-dimensional analog.

Figure 11 illustrates the use of isosurface stuffing in a liquid

```
for each octant o that is a leaf or has a nonnegative center point
  c ⇐ the center vertex of o.
  for each square face s of o
    if there is no vertex at the center of s
      if s is shared with another octant o′ the same size as o
        c′ ⇐ the center vertex of o′.
        for each edge e of the square s
          if there is no vertex at the midpoint of e
            Create the BCC tetrahedron conv(e ∪ {c, c′}).
          else
            m ⇐ the midpoint vertex of e.
            for each endpoint p of e
              Create the bisected BCC tet conv({p, m, c, c′}).
      else {s is shared with a larger octant or the boundary }
        Create two half-pyramids filling the pyramid conv(s ∪ {c}).
                  The diagonal bisecting the pyramid must
                  adjoin a corner or center vertex of c's parent.
    else {there is a vertex at the center of the square s}
      d ⇐ the center vertex of s.
      for each edge e of the square s
        if there is no vertex at the midpoint of e
          Create the bisected BCC tetrahedron conv(e ∪ {d, c}).
        else
          m ⇐ the midpoint vertex of e.
          for each endpoint p of e
            if o has no child with vertex p
              Create the quadrisected BCC tet conv({p, m, d, c}).
            { else do nothing; o has a child that will take care of
                  tetrahedralizing the corner of o near p. }
```

Figure 10: Algorithm for creating a background grid from our weakly balanced octree. Note that the algorithm as written here creates any background tetrahedron that spans two octants twice; an implementation should take care to avoid this duplication.

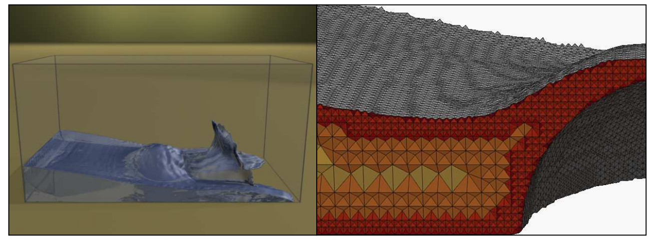

Figure 11: A frame from an animation of flowing liquid, and a fragment of the mesh used to create it. Courtesy of Nuttapong Chentanez, Bryan Feldman, and James O'Brien.

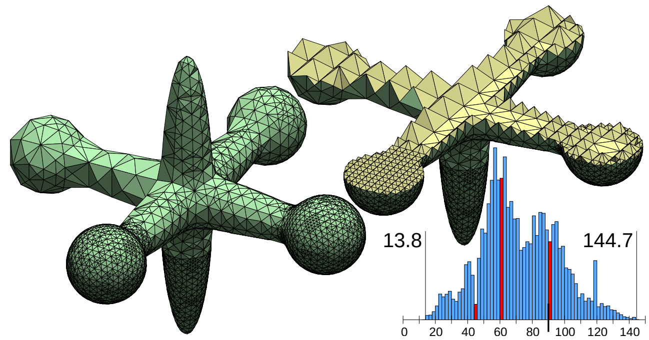

Figure 12: A 22,728-tetrahedron mesh, and a cutaway view thereof, generated in 2,859 milliseconds by a variant of isosurface stuffing that allows tetrahedra to grade both on and inside the surface. Dihedral angles vary from 13.8° to 144.7°. Plane angles on the surface vary from 10.9° to 138.5°. These results are typical, but much worse angles can occur.

simulation video by the Berkeley Computer Animation and Modeling group. They tell us they prefer it to their previous algorithm, the variational mesher of Alliez et al. [2005], because it is faster, it never creates poor tetrahedra near the boundary, its meshes are more coherent between time steps, and they can exploit the regularity of the meshes for fast point location and element stiffness matrix reuse.

6 Conclusions

Isosurface stuffing doubles as a guaranteed-quality, watertight isosurface triangulation algorithm almost as simple and fast as Marching Cubes: simply output the triangles on the boundary of the mesh generated by isosurface stuffing, and enjoy the exposed plane angle bounds listed in Table 1. The upper bounds of less than 125° on plane angles are noteworthy. They compete with the 120° guarantee of Chew's algorithm [1989], at a fraction of the effort and running time.

For the α parameters listed as "safe" in Table 1, the background BCC tetrahedra cannot become degenerate, so the surface mesh generated by the algorithm cannot have self-intersections. The 15.1285° bound in the table is unsafe; a BCC lattice tetrahedron with four warped vertices could become inverted, and could potentially cause a few of the output triangles to have intersecting interiors (though we have not seen it in practice) if the grid is insufficiently fine or if the zero-surface is not a smooth manifold with bounded curvature. To obtain the most aggressive bound on the smallest angle (16.4299°, with ordered warping), we use $\alpha_{\text{short}} = 0.5$, which allows a BCC tetrahedron to become arbitrarily close to degenerate. To prevent it from becoming perfectly flat, we adopt the convention that a cut point precisely at the midpoint of a red edge violates the endpoint on the cubical lattice $\mathbb{Z}^3 + (\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$, but not the endpoint on the cubical lattice $\mathbb{Z}^3$. The choice $\alpha_{\text{short}} = 0.5$ means that no cut point ever survives on a red edge, so most of the stencils are never used. This simplifies the algorithm, and also makes the 16.4299° bound possible.

An alluring goal that we have not been able to achieve is a tetrahedral meshing algorithm that permits grading of both surface and interior tetrahedra and has a strong bound on the dihedral angles. This is not to say that we have no algorithm. With our techniques, we can construct a background grid that is graded on the domain boundary as well as in the interior, and we can apply our cutting and warping technique to it. We have experimented with different stencils for the four kinds of background tetrahedron, and found some good choices. The majority of the tetrahedra produced this way are good in practice, as Figure 12 shows. However, we cannot make guarantees on dihedral angles better than 1.66° or 174.72°. We see three main obstacles: the half-pyramid background tetrahedron can be severely distorted by warping (because of its 120° dihedral angle); smaller tetrahedra can have their vertices warped a long distance by larger neighbors; and although we can find good stencils for all four kinds of background tetrahedron, we cannot get them to agree on their shared diagonals without sacrificing quality. We are optimistic that a solution to this hard problem is tantalizingly within reach, but it might require a more clever graded background grid, one that somehow avoids dihedral angles much larger than 90°.

Nevertheless, this paper achieves a goal that has eluded researchers for nearly two decades: a mesh generation algorithm for complicated shapes that offers theoretical guarantees on dihedral angles strong enough to be meaningful to practitioners. The shortcomings of isosurface stuffing—its tendency to round off sharp corners and edges, and the reduction of guaranteed quality if the surface tetrahedra are not of uniform size—are balanced by its simplicity and raw speed. The combination of three features—speed, guaranteed quality, and numerical robustness—makes isosurface stuffing the first mesh generation algorithm suitable for robust remeshing in physically-based animation at interactive rates.

Acknowledgments

We thank Nuttapong Chentanez for providing isosurface code and geometric models; Carlo Séquin for *Whirled White Web*; and Nuttapong, Bryan Feldman, and James O'Brien for their animation still. This work was supported in part by the National Science Foundation under Awards CCF-0430065 and CCF-0635381, and in part by an Alfred P. Sloan Research Fellowship.

References

ALLIEZ, P., COHEN-STEINER, D., YVINEC, M., AND DESBRUN, M. 2005. Variational Tetrahedral Meshing. *ACM Transactions on Graphics 24*, 3, 617–625. Special issue on Proceedings of SIGGRAPH 2005.

AMENTA, N., AND BERN, M. 1999. Surface Reconstruction by Voronoi Filtering. *Discrete & Computational Geometry 22*, 4 (Dec.), 481–504.

AMENTA, N., CHOI, S., DEY, T. K., AND LEEKHA, N. 2002. A Simple Algorithm for Homeomorphic Surface Reconstruction. *International Journal of Computational Geometry and Applications 12*, 1–2, 125–141.

BÆRENTZEN, J. A., AND AANÆS, H. 2002. Generating Signed Distance Fields from Triangle Meshes. Tech. Rep. IMM-TR-2002-21, Informatics and Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark.

BAKER, B. S., GROSSE, E., AND RAFFERTY, C. S. 1988. Nonobtuse Triangulation of Polygons. *Discrete and Computational Geometry 3*, 2, 147–168.

BANK, R. E., AND SCOTT, L. R. 1989. On the Conditioning of Finite Element Equations with Highly Refined Meshes. *SIAM Journal on Numerical Analysis 26*, 6 (Dec.), 1383–1394.

BERN, M., EPPSTEIN, D., AND GILBERT, J. R. 1994. Provably Good Mesh Generation. *Journal of Computer and System Sciences 48*, 3 (June), 384–409.

BEY, J. 1995. Tetrahedral Grid Refinement. *Computing 55*, 355–378.

BLOOMENTHAL, J. 1994. An Implicit Surface Polygonizer. In *Graphics Gems IV*. Academic Press, ch. IV.8, 324–349.

CHENG, S.-W., AND DEY, T. K. 2002. Quality Meshing with Weighted Delaunay Refinement. In *Proceedings of the Thirteenth Annual Symposium on Discrete Algorithms*, 137–146.

CHENG, S.-W., DEY, T. K., EDELSBRUNNER, H., FACELLO, M. A., AND TENG, S.-H. 2000. Sliver Exudation. *Journal of the ACM 47*, 5 (Sept.), 883–904.

CHEW, L. P. 1989. Guaranteed-Quality Triangular Meshes. Tech. Rep. TR-89-983, Department of Computer Science, Cornell University.

CHEW, L. P. 1997. Guaranteed-Quality Delaunay Meshing in 3D. In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*, 391–393.

EDELSBRUNNER, H., AND GUOY, D. 2001. An Experimental Study of Sliver Exudation. In *Tenth International Meshing Roundtable*, 307–316.

EPPSTEIN, D., SULLIVAN, J. M., AND ÜNGÖR, A. 2004. Tiling Space and Slabs with Acute Tetrahedra. *Computational Geometry: Theory and Applications 27*, 3 (Mar.), 237–255.

FREITAG, L. A., AND OLLIVIER-GOOCH, C. 1997. Tetrahedral Mesh Improvement Using Swapping and Smoothing. *International Journal for Numerical Methods in Engineering 40*, 21 (Nov.), 3979–4002.

FUCHS, A. 1998. Automatic Grid Generation with Almost Regular Delaunay Tetrahedra. In *Seventh International Meshing Roundtable*, 133–148.

JAMET, P. 1976. Estimations d'Erreur pour des Élements Finis Droits Presque Dégénérés. *RAIRO Analyse Numérique 10*, 43–61.

KŘÍŽEK, M. 1992. On the Maximum Angle Condition for Linear Tetrahedral Elements. *SIAM Journal on Numerical Analysis 29*, 2 (Apr.), 513–520.

LABELLE, F. 2006. Sliver Removal by Lattice Refinement. In *Proceedings of the Twenty-Second Annual Symposium on Computational Geometry*, 347–356.

LI, X.-Y., AND TENG, S.-H. 2001. Generating Well-Shaped Delaunay Meshes in 3D. In *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms*, 28–37.

LORENSEN, W. E., AND CLINE, H. E. 1987. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *Computer Graphics (SIGGRAPH '87 Proceedings)*, 163–170.

MITCHELL, S. A., AND VAVASIS, S. A. 2000. Quality Mesh Generation in Higher Dimensions. *SIAM Journal on Computing 29*, 4, 1334–1370.

MOLINO, N., BRIDSON, R., TERAN, J., AND FEDKIW, R. 2003. A Crystalline, Red Green Strategy for Meshing Highly Deformable Objects with Tetrahedra. In *Twelfth International Meshing Roundtable*, 103–114.

NAYLOR, D. J. 1999. Filling Space with Tetrahedra. *International Journal for Numerical Methods in Engineering 44*, 10 (Apr.), 1383–1395.

OHTAKE, Y., BELYAEV, A., ALEXA, M., TURK, G., AND SEIDEL, H.-P. 2003. Multi-Level Partition of Unity Implicits. *ACM Transactions on Graphics 22*, 3 (July), 463–470. Special issue on Proceedings of SIGGRAPH 2003.

OSHER, S., AND FEDKIW, R. 2002. *Level Set Methods and Dynamic Implicit Surfaces*. Springer-Verlag, New York.

OUDOT, S., RINEAU, L., AND YVINEC, M. 2005. Meshing Volumes Bounded by Smooth Surfaces. In *Proceedings of the 14th International Meshing Roundtable*, 203–219.

SETHIAN, J. A. 1996. A Fast Marching Level Set Method for Monotonically Advancing Fronts. *Proceedings of the National Academy of Sciences 93*, 4 (Feb.), 1591–1595.

SHEN, C., O'BRIEN, J. F., AND SHEWCHUK, J. R. 2004. Interpolating and Approximating Implicit Surfaces from Polygon Soup. *ACM Transactions on Graphics 23*, 3 (Aug.), 896–904. Special issue on Proceedings of SIGGRAPH 2004.

SHEWCHUK, J. R. 2002. What Is a Good Linear Element? Interpolation, Conditioning, and Quality Measures. In *Eleventh International Meshing Roundtable*, 115–126.

SOMMERVILLE, D. M. Y. 1923. Space-Filling Tetrahedra in Euclidean Space. *Proceedings of the Edinburgh Mathematical Society 41*, 49–57.

YERRY, M. A., AND SHEPHARD, M. S. 1984. Automatic Three-Dimensional Mesh Generation by the Modified-Octree Technique. *International Journal for Numerical Methods in Engineering 20*, 11 (Nov.), 1965–1990.

ZHAO, H.-K., OSHER, S., AND FEDKIW, R. 2001. Fast Surface Reconstruction Using the Level Set Method. In *Workshop on Variational and Level Set Methods*, 194–202.