Tetrahedral Mesh Generation with Good Dihedral Angles Using Point Lattices

by

François Labelle

B.Sc. (McGill University) 1997
M.Sc. (McGill University) 2000

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Jonathan Shewchuk, Chair
Professor James F. O'Brien
Professor Panayiotis Papadopoulos

Fall 2007

The dissertation of François Labelle is approved:

Chair _____ Date _____

_____ Date _____

_____ Date _____

University of California, Berkeley

Fall 2007

Abstract

Tetrahedral Mesh Generation with Good Dihedral Angles
Using Point Lattices
by
François Labelle
Doctor of Philosophy
in
Computer Science
University of California, Berkeley
Professor Jonathan Shewchuk, Chair

Three-dimensional meshes are frequently used to perform physical simulations in science and engineering. This involves decomposing a domain into a mesh of small elements, usually tetrahedra or hexahedra. The elements must be of good quality; in particular there should be no plane or dihedral angle close to 0 or 180 degrees. Automatically creating such meshes for complicated domains is a challenging problem, especially guaranteeing good dihedral angles, a goal that has eluded researchers for nearly two decades. By using point lattices, notably the body centered cubic lattice, we develop two tetrahedral mesh generation algorithms that, for the first time, come with meaningful guarantees on the quality of the elements.

For domains bounded by an isosurface, we generate a tetrahedral mesh whose dihedral angles are bounded between 10.7 and 164.8 degrees, or (with a change in parameters) between 8.9 and 158.8 degrees. The algorithm is numerically robust and easy to implement because it generates tetrahedra from a small set of precomputed stencils. The algorithm is so fast that it can be invoked at each time step of a simulation, possibly in real time for small meshes. The tetrahedra are uniformly sized on the boundary, but in the interior it is possible to make them progressively larger. This combination of features makes the algorithm well suited for dynamic fluid simulation. 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.

For polyhedral domains, Delaunay refinement is a common mesh generation technique that produces guaranteed-quality tetrahedra with one exception: sliver-shaped tetrahedra that can have dihedral angles arbitrarily close to 0 and 180 degrees. We show how slivers away from the boundary can be avoided by inserting lattice points while maintaining the Delaunay property of the mesh. It is possible to control the size of tetrahedra, this time both in the interior and on the boundary, and a user may input a set of points which

must be vertices of the output mesh. The resulting dihedral angles are guaranteed to be between 30 and 135 degrees, except near the boundary.

Most angle bounds are obtained by recursive bisection of the space of possible tetrahedron configurations together with interval arithmetic. They are guaranteed by computer-assisted proofs.

Chair _____ Date _____

Acknowledgments

I would like to thank my advisor Jonathan Shewchuk for his guidance and support, and the Berkeley Graphics group for feedback. I also thank Nuttapong Chentanez for providing isosurface code and geometric models, and Carlo Séquin for the *Whirled White Web* model.

This work was supported in part by the National Science Foundation under Awards CCR-0204377, CCF-0430065 and CCF-0635381, and in part by an Alfred P. Sloan Research Fellowship.

Contents

Chapter 1

Introduction

Meshes of small elements are often used to simulate physical phenomena numerically on a computer. In engineering, numerical simulations can help design and test planes, bridges or components before they are built. This can decrease costs and increase development speed where prototypes were once needed. In science, the same techniques can be used for a wide range of problems like modeling tides or star interiors. Due to the success of these methods, more difficult problems are tackled, for example in biology where geometry tends to be complicated and curved, hence harder to mesh. Figure 1.1 shows example applications.

In computer graphics, the traditional thinking is that only the surface of an object is visible so only surface meshes are necessary. However, volume meshes are used increasingly as researchers incorporate physical simulations in graphics to automatically animate liquids and gasses, or objects that bend, crack or break. As computing performance improves, some of these simulations are starting to appear in computer games to make the virtual environment more realistic.

The *finite element method* (FEM) is the most common way of solving a partial differential equation (PDE) over a meshed domain. The theory for elliptic problems is well understood and enjoys some nice results; for example a piecewise linear approximation is sufficient to solve an elliptic PDE of order 2 (such as heat diffusion or elasticity), and in general a piecewise approximation of degree m is sufficient to solve an elliptic PDE of order 2m. See the book by Johnson [48] for an introduction to the FEM and the mathematics behind it.

The mesh can follow the material as it moves (Lagrangian formulation), or stay fixed in space as the material flows through it (Eulerian formulation). Those two possibilities are normally used for solids and fluids, respectively, but the reverse is technically possible. There is a third possibility: the mesh is allowed to move but doesn't necessarily have to track the material. This extremely flexible formulation is called *Arbitrary Lagrangian*

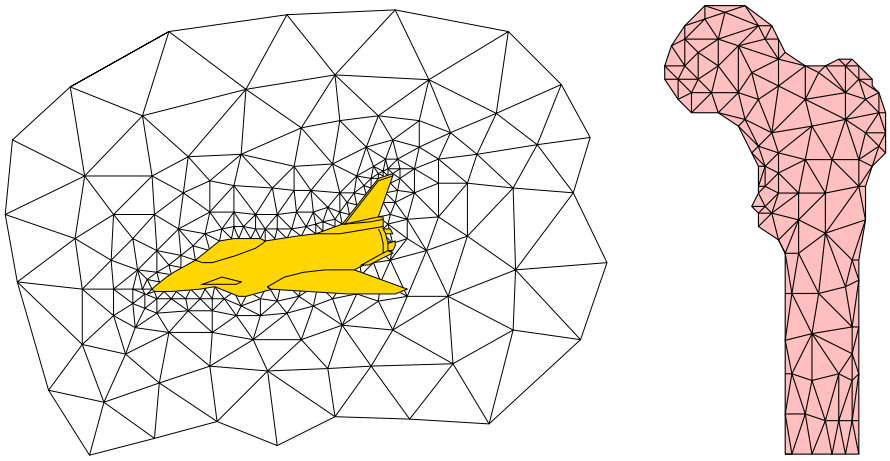

Figure 1.1: Part of an exterior mesh used for computational fluid dynamics and part of a mesh of a femur used for stress analysis.

Eulerian (ALE); see for example the survey of Donea et al. [30].

In very difficult applications, some people have developed methods that do not require a mesh, called "meshfree methods". One commonly cited early example is *smoothed particle hydrodynamics*, introduced in 1977 by Lucy [60] and Gingold and Monaghan [44]. Meshfree methods have some drawbacks: they are computationally expensive and boundary conditions are harder to impose. Ironically, a mesh is often used at one step of the method, for example to perform the numerical integration of the special basis functions.

1.1 The Mesh Generation Problem

Given a description of the domain geometry, the goal is to produce a mesh of elements that fill the domain. Sometimes we are also given a *sizing function* defined over the domain to control element sizes.

The mesher doesn't usually depend on other details of the application (for example which differential equation is going to be solved with the mesh). This means that the same mesh generation code can be used for many different applications.

For more background on mesh generation, see the survey by Bern and Plassmann [10].

1.1.1 Domain Representation

Domain geometry can be represented in many different ways which we classify as explicit or implicit.

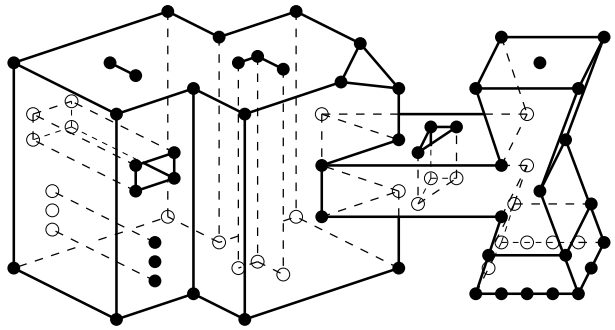

Figure 1.2: A piecewise linear complex can be the input to a mesh generator (J. Shewchuk).

Explicit Representation

In an *explicit* representation, we are given a list of vertices, segments and facets bounding the domain. A common example is a *piecewise linear complex* [62] which can be used to represent polyhedral domains (internal vertices, segments and facets are also allowed). Self-intersection is not allowed unless the intersection is explicitly resolved with vertices, segments and facets. See Figure 1.2.

When meshing such a domain, input vertices must be part of the output mesh. Input segments and facets must appear unchanged or as a union of subsegments or subfacets.

Extension to curved segments and facets is possible [72, 15]. Sharp features are explicitly listed and must be accurately represented by the mesh. This is similar to boundary representation (B-rep) in computer-aided design.

Implicit Representation

In an *implicit* representation, the domain is defined as the volume bounded by a given isosurface. An isosurface is a surface of constant value $\{p : f(p) = c\}$ where $f : \mathbf{R}^3 \rightarrow \mathbf{R}$ is a scalar field called a *cut function* that the mesher can interrogate. See Figure 1.3. Minimally, the domain can be defined implicitly by a black box that tells if a point is "inside" or "outside" without returning any numerical value.

This is a convenient representation in many cases. For example when simulating a moving liquid, an inside/outside query at a point p for the current time step can be answered by moving p backward in time using the estimated velocity field and querying the previous time step—a method called *semi-Lagrangian advection* [7]. By contrast, maintaining an explicit representation of a liquid surface would be considerably harder.

Isosurfaces are produced by several algorithms for surface reconstruction [88, 67, 77]. Even for geometric models that do not use isosurfaces, it is usually possible to compute

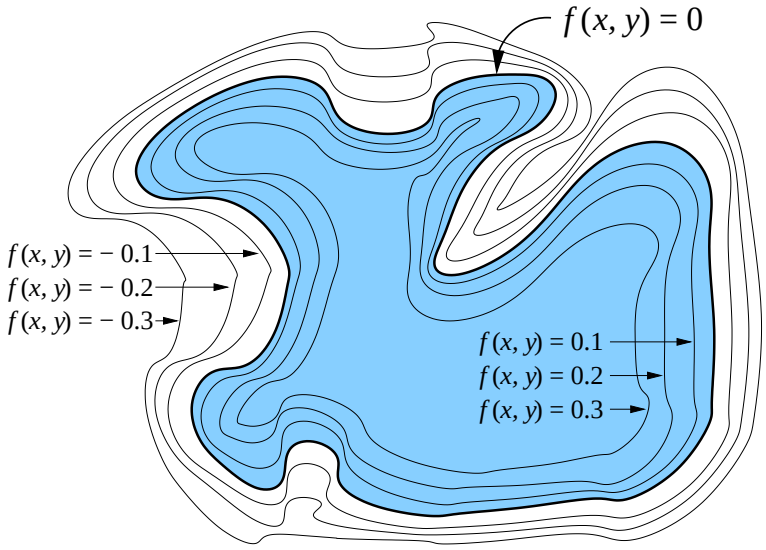

Figure 1.3: A two-dimensional example showing 7 isosurfaces. The zero-surface is the isosurface where the function f is zero. The inside and outside regions correspond to positive and negative values of f, respectively.

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 [76, 68]. An algorithm of Bærentzen and Aanæs [5] for watertight triangular surface meshes computes just the sign, which suffices for our algorithm.

It is theoretically possible to represent sharp features with an implicit representation, but if the mesher is unaware of this, corners and edges of the domain could be rounded off. There has been some work in detecting sharp features in implicit surfaces [49], but we feel that this is an exercise in recovering information that should not have been lost. If there are important sharp features then it is better to represent them explicitly.

1.2 Mesh Quality

The quality of a mesh depends on the quality of its elements, which in turn depends on the application. In most applications, tetrahedra that are close to regular are favored, while tetrahedra that are close to degenerate should be avoided [26, 38], see Figure 1.4. In this section we explain the main reasons for this.

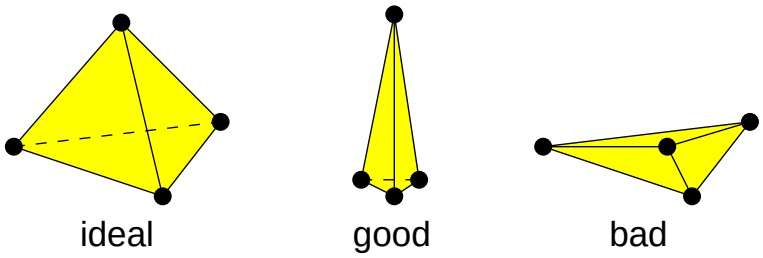

Figure 1.4: The shape of a tetrahedron is important for numerical methods.

Often, a mesh is used to approximate a function over a domain. For example, given a tetrahedral mesh and values at its vertices (or a triangular mesh in two dimensions), we can linearly interpolate the vertex values over each individual tetrahedron (or triangle). This creates a piecewise linear approximation over the meshed domain.

In what follows, $f : \mathbf{R}^3 \to \mathbf{R}$ is a function with a bound on curvature of $2c_t$ inside a tetrahedron t (i.e. a bound of $2c_t$ on the second derivative at a point of t in any direction). Given the value of f at the 4 vertices of t, let g be the approximation obtained by linear interpolation.

Interpolation Error

A bound on the piecewise interpolation error in the tetrahedron t can be given in terms of $l_{\max}$, the maximum edge length of t. A tighter bound is possible in terms of r_{mc}, the radius of the *minimum containment* sphere of t, i.e. the smallest sphere that encloses t. The bounds [80] are

$$|f - g| \leq c_t r_{\mathrm{mc}}^2 \leq c_t \frac{3}{8} l_{\max}^2.$$

In short, the shape of an element doesn't affect the interpolation error, only its size does.

Gradient Error

In many applications, including the finite element method, the approximation g must also give accurate derivatives. Bounds on the gradient error $\|\nabla f - \nabla g\|_2$ are more complicated and given by Shewchuk [80]. The conclusion is that plane and dihedral angles near 0° don't cause an error in gradient, but plane and dihedral angles near 180° do. See Figure 1.5 for an example with two large dihedral angles.

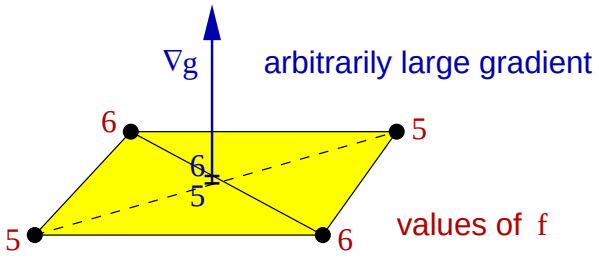

Figure 1.5: Although the function f is close to constant, linear interpolation of f over this tetrahedron gives a function g with a large gradient.

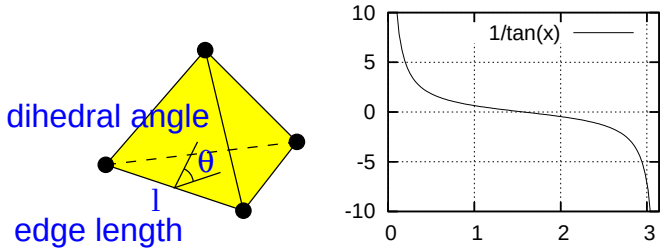

Figure 1.6: If a dihedral angle is close to $0°$ or $180°$, a stiffness matrix entry of the form $\pm\frac{l}{6}\cot\theta$ can be arbitrarily large because of the cotangent.

Stiffness Matrix Conditioning

In the finite element method, a partial differential equation is converted into a large, sparse linear system to be solved. For example, when solving Poisson's equation with linear elements, each edge of each tetrahedron contributes four matrix entries of the form $\pm\frac{l}{6}\cot\theta$ where l is the edge length and θ is the dihedral angle at that edge (see Figure 1.6). Because the expression contains a cotangent, dihedral angles near $0°$ or near $180°$ can cause large matrix entries which lead to poor matrix conditioning.

Anisotropy

The considerations so far assumed that the phenomenon to be modeled is *isotropic*, i.e. the same in all directions. In some applications, a PDE or its solution can be *anisotropic*, the opposite. An example is fluid flow close to a boundary, which tends to be uniform along the boundary but varies dramatically in the orthogonal direction. In such cases, skinny elements that are correctly aligned are desirable [73]. We do not attempt to create anisotropic meshes in this thesis.

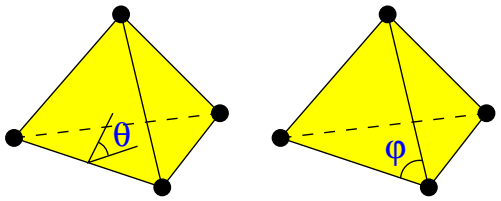

Figure 1.7: A tetrahedron has 6 dihedral angles (one per edge of the tetrahedron) and 12 plane angles (3 on each of its 4 triangular faces). One angle of each type is illustrated. The dihedral angle at an edge is the angle between the two triangular faces incident on it.

1.2.1 Quality Measures

Many measures of tetrahedron quality have been proposed to express the requirements imposed by numerical methods. A possible quality measure, the *radius ratio*, is given by the inscribed sphere radius divided by the circumscribed sphere radius. Most of the proposed measures are asymptotically equivalent in the sense that a bound on one implies a bound on the others [57]. In this thesis, we focus on dihedral angles and plane angles (see Figure 1.7) because they directly appear in error bound expressions for tetrahedra and triangles, and they are easy to interpret. Since a single bad element can potentially ruin a whole simulation, it is desirable to guarantee the quality of every element of the mesh.

Covering Efficiency

Good quality elements with a large volume are preferable over those with a small volume because fewer of them will be needed to cover the mesh.

Given a fixed element budget for the mesh and a quality measure that favors small elements, a good mesh optimizer will tend to create "fat" elements even if the volume of the element is not explicitly part of the quality measure. In some cases, it might be necessary to include the volume of the element in the quality measure to obtain reasonable covering efficiency.

1.2.2 Size-Optimality

It is normally much easier to refine a mesh (add vertices) than to simplify a mesh (remove vertices) while maintaining mesh quality. For this reason, mesh generators are often compared based on their ability to create a good quality mesh with as few elements as

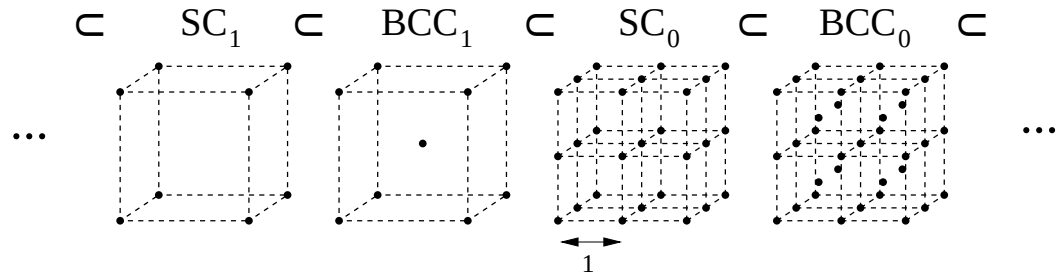

Figure 1.8: The point sets SC_k and BCC_k for $k \in \mathbf{Z}$ form an infinite sequence of nested lattices.

possible, knowing that if more elements are needed then this won't be a problem. In the following definition we compare a mesh with the "optimal" mesh.

Definition 1 (Size-Optimality) *A mesh that satisfies some quality bound is said to be* size optimal *if its number of elements is at most a constant factor larger than the number of elements in any other mesh of the same domain that satisfies the same quality bound. The constant may be a function of the quality bound, but cannot depend on the mesh.*

1.3 Point Lattices

Point lattices are common in mineralogy where they represent the structure of many crystals. These regular point sets lead to simple grids that can be used directly in some numerical methods, but more interestingly they can be used as a backbone for more sophisticated meshing algorithms. They are the special ingredient of this thesis.

Below we define two families of lattices that are used in this thesis. The first one simply corresponds to the vertices of a regular grid of cubes. For conciseness, we define addition and scalar multiplication of point sets as follows:

$$\begin{aligned} A + B &= \{a + b : a \in A \text{ and } b \in B\}, \\ cA &= \{ca : a \in A\}. \end{aligned}$$

Definition 2 (Simple Cubic Lattice) *The simple cubic lattice* SC_0 *and its scaling* SC_k *by powers of two are defined as*

$$\begin{aligned} \mathrm{SC}_0 &= \mathbf{Z}^3, \\ \mathrm{SC}_k &= 2^k \mathrm{SC}_0 \text{ for } k \in \mathbf{Z}. \end{aligned}$$

The next lattice can be obtained from the simple cubic lattice by adding new vertices at the center of each cube.

Definition 3 (Body-Centered Cubic Lattice) *The body-centered cubic lattice* BCC_0 *and its scaling* BCC_k *by powers of two are defined as*

$$\begin{aligned} \mathrm{BCC}_0 &= \{(0,0,0),(\tfrac{1}{2},\tfrac{1}{2},\tfrac{1}{2})\} + \mathbf{Z}^3, \\ \mathrm{BCC}_k &= 2^k \mathrm{BCC}_0 \text{ for } k \in \mathbf{Z}. \end{aligned}$$

The following lattice is mentioned for comparison only. It is known to describe a sphere packing of maximum density, but is not used in this thesis.

Definition 4 (Face-Centered Cubic Lattice) *The face-centered cubic lattice* FCC_0 *and its scaling* FCC_k *by powers of two are defined as*

$$\begin{aligned} \mathrm{FCC}_0 &= \{(0,0,0),(\tfrac{1}{2},\tfrac{1}{2},0),(\tfrac{1}{2},0,\tfrac{1}{2}),(0,\tfrac{1}{2},\tfrac{1}{2})\} + \mathbf{Z}^3, \\ \mathrm{FCC}_k &= 2^k \mathrm{FCC}_0 \text{ for } k \in \mathbf{Z}. \end{aligned}$$

The sets above are called *point lattices* because they are discrete subgroups of Euclidean space under vector addition of point coordinates. Other regular point patterns, such as the centers of spheres in a hexagonal close packing, do not share this property. None of our results depend on this subgroup property, so non-lattice point sets can be used in future work if the lattices that we have chosen turn out to have limitations.

Proposition 1 (Nesting of Lattices)

$$\mathrm{BCC}_{k+1} \subset \mathrm{SC}_k \subset \mathrm{BCC}_k \text{ for } k \in \mathbf{Z}.$$

The nesting of lattices is illustrated in Figure 1.8. A lattice L_1 is said to be *finer* than a lattice L_2 if $L_1 \supset L_2$. L_1 is said to be *coarser* than L_2 if $L_1 \subset L_2$.

1.4 Summary of Results

In Chapter 3, we show how a background grid of lattice points can be used to generate a high-quality mesh for bodies whose boundaries are smooth surfaces. The method is surprisingly quick and easy, and it offers numerical robustness and tetrahedron quality, which is particularly reassuring for simulations that need to generate new meshes frequently, perhaps even at frame rates.

The 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, it 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.

Besides high-quality tetrahedra, the algorithm offers three other guarantees, described in Section 3.4. 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 Corollary 5.) 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 8.)

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.

In Chapter 4, we show how point lattices can be used in a different type of mesh generation algorithm where a mesh is iteratively constructed by inserting one vertex at a time, and the domain to be meshed can have sharp features. The algorithm can create a mesh from scratch, or fix a pre-existing mesh by adding more vertices. It guarantees dihedral angles between 30° and 135° away from the boundary. In comparison, previous bounds on dihedral angles were microscopic. Unfortunately, the algorithm offers no guarantee close to be boundary, where sliver-shaped tetrahedra can exist.

The algorithm 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". The algorithm comes with a bound on the sizes of the features it creates, and can provably grade from small to large tetrahedra.

The method is an extension of Delaunay refinement, a common mesh generation technique. This is appealing because the changes can be easily added to a pre-existing Delaunay refinement mesher.

Chapter 2

Previous Work

Tetrahedral mesh generation has an extensive history in both engineering and computer science, so we review here mainly methods that share similarities with ours or have theoretical guarantees. For more information on mesh generation methods, see the survey by Bern and Plassmann [10], Bern and Eppstein [8], or Owen [70].

2.1 Background Grids

A *background grid* is an invisible, structured grid that is used to guide mesh generation. In this section we also review algorithms that produce merely *surface meshes* in three dimensions, because they contain ideas that we borrow or extend to generate a full interior mesh in Chapter 3.

2.1.1 Surface Meshes

Marching Cubes

The *Marching Cubes* algorithm of Lorensen and Cline [59] triangulates an isosurface (but not its interior). It computes the cut function f at the vertices of the simple cubic lattice, 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 15 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.

Chernyaev [22] proposed an improved version with 33 stencils that are selected based on the topology of the isosurface defined by the trilinear interpolation of values at cube

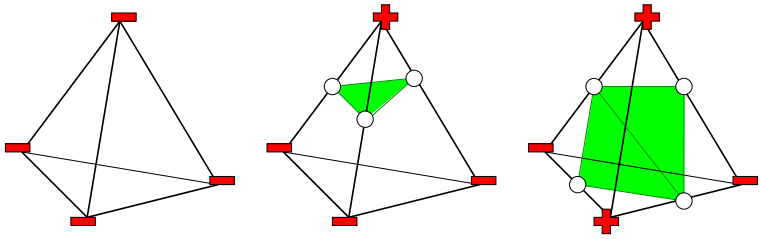

Figure 2.1: In Marching Tetrahedra, there are only 3 stencils up to symmetry, including the case where no surface triangle is produced.

vertices. However, surface mesh quality remains poor. Tzeng [83] sketched a strategy to improve the quality of output triangles, but the description given is not precise and the analysis not rigorous. Attali et al. [4] combine Marching Cubes with a surface simplification heuristic to improve mesh quality without having to store the intermediate poor-quality surface mesh.

Marching Tetrahedra

Marching Tetrahedra is an obvious extension (or simplification) of Marching Cubes where the background grid is composed of tetrahedra. The tetrahedral background grid can be obtained by decomposing each cube of a cubic grid into tetrahedra, as done by Bloomenthal [13] in his implementation. The tetrahedral background grid can have a different structure, or could even be totally unstructured. Carr et al. [18] compare the results of Marching Tetrahedra (on several background grids) with Marching Cubes. The main advantage of Marching Tetrahedra is that there are only 3 stencils to consider, illustrated in Figure 2.1. Treece et al. [82] use the body-centered cubic lattice and propose a method to improve the quality of output triangles, but the worst case is not analyzed.

2.1.2 Volume Meshes

Sharp Domains

Field and Smith [39] propose a method to obtain *graded* meshes based on the body-centered cubic lattice; however their algorithm description is informal and no bounds on the dihedral angles are provided. Fuchs [43] uses the BCC lattice to create a Delaunay mesh that is mostly regular.

In two dimensions, the early grid-based algorithm of Baker, Grosse, and Rafferty [6] is notable, guaranteeing that the angles of the triangles of the mesh are between 13° and 90° (excepted input angles, which could be less than 13° and can't be eliminated).

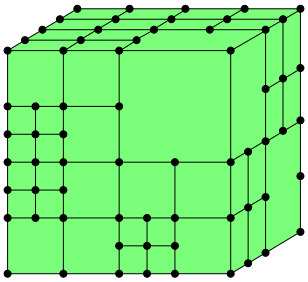

Figure 2.2: In an octree, an initial axis-aligned cube is recursively cut into eight smaller cubes as needed. The octree is said to be balanced if the the sizes of adjacent cubes differ by a factor of at most two.

Smooth Domains

There are several prior algorithms for filling smooth surfaces with tetrahedra. Molino et al. [65] begin with a BCC grid, then grade the mesh by using the red-green mesh refinement of Bey [11] 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 a one-pass stencil-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.

2.2 Quadtree and Octree

Quadtrees and octrees are ways of adaptively refining space in two and three dimensions (respectively); see Figure 2.2. In the context of mesh generation, their advantage over a simple uniform grid is to provide a way to control and adapt the sizes of elements. Quadtrees and octrees were used for meshing in the pioneering work of Yerry and Shephard [86, 87].

Guarantees

In two dimensions, the quadtree-based triangular meshing algorithm of Bern, Eppstein, and Gilbert [9] comes with a small guarantee on angles. The domain is first enclosed in an axis-aligned square which is recursively subdivided in a quadtree fashion until each leaf square intersects the domain in a simple way. The quadtree is further refined so that it is balanced, and is warped to conform to input segments. The guarantee is derived by looking at the worst possible angle that can be created by the process.

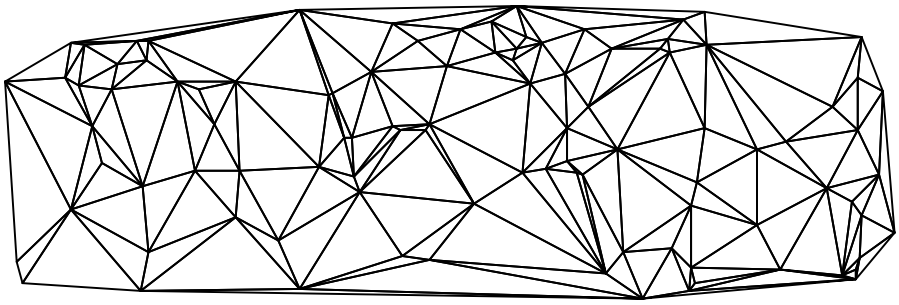

Figure 2.3: The Delaunay triangulation of 100 points randomly chosen inside a rectangle.

The idea is extended to three dimensions by Mitchell and Vavasis [63], who give an octree-based algorithm that creates meshes with a very small guarantee on radius ratios. Unlike the two-dimensional version, the meshes obtained are not very satisfactory because they contain many elements and many small angles. The algorithm is extended to higher dimensions by the same authors [64].

2.3 Delaunay

The *Delaunay triangulation* of a point set has the property that the circumcircle of every triangle is empty; or in three dimensions, the circumsphere of every tetrahedron is empty. It is named after Boris Delaunay [29] who introduced it. In two dimensions, Lawson [51] proves that the Delaunay triangulation maximizes the minimum angle over all possible triangulation of the point set, a property highly relevant to mesh quality. Figure 2.3 shows a Delaunay triangulation.

In mesh generation by Delaunay refinement, a Delaunay triangulation is maintained at all times during the meshing process, and is used to guide the insertion of new vertices. The Delaunay triangulation and Delaunay refinement are described in more details in Section 4.1. Delaunay refinement algorithms are intimately linked to the following geometric ratio.

Definition 5 (Radius-Edge Ratio) *The* radius-edge ratio *of a simplex (triangle, tetrahedron, or higher dimensional analogues) is the ratio of the simplex's circumradius to the length of its shortest edge [61].*

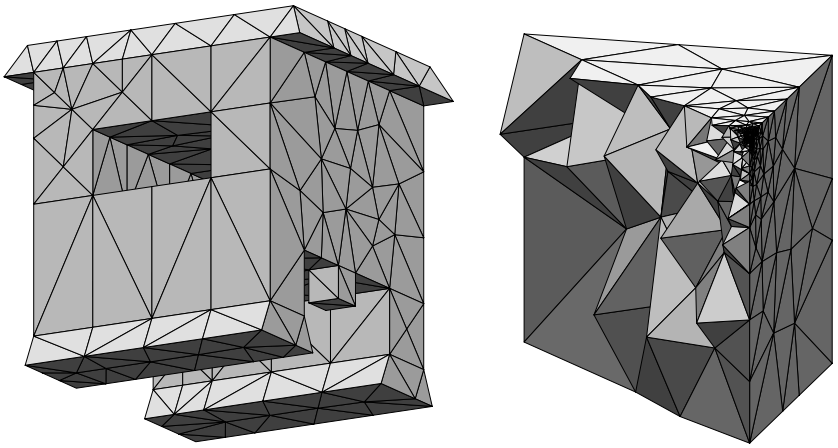

Figure 2.4: Meshes obtained by three-dimensional Delaunay refinement. The mesh on the right is a cutaway view of a graded mesh. (J. Shewchuk)

Sharp Domains

In two dimensions, Frey [42] suggests eliminating poor quality triangles by inserting their circumcenters. Based on this idea, the Delaunay refinement algorithm of Chew [24] produces uniform meshes with a radius-edge ratio of at most 1, which implies triangle angles between 30° and 120° (Chew assumes some conditions on the domain boundary). Given a domain with no acute angle, Ruppert's algorithm [74] produces *graded* meshes with a radius-edge ratio of at most $\sqrt{2}$, which translates into triangle angles between 20.7° and 138.6°.

Shewchuk [78] extends Delaunay refinement to three dimensions (see Figure 2.4 for example results). For domains with no acute angles, his algorithm can guarantee radius-edge ratios of at most 2. Unfortunately, this guarantee does not rule out the *sliver tetrahedron*, which can have dihedral angles arbitrarily close to 0° and 180°; see Figure 2.5(a). In a mesh wherein all the tetrahedra have good (small) radius-edge ratios, the problem of obtaining well-shaped tetrahedra is reduced to eliminating slivers, and an algorithm with a guarantee on radius-edge ratios should be complemented with a guarantee on the smallest dihedral angle. See Figure 2.5(b,c,d) for examples of tetrahedra that demonstrate that bounds on dihedral angles alone may not be sufficient to rule out some types of degenerate tetrahedra.

Since three-dimensional Delaunay refinement apparently comes so close to providing well-shaped elements, it is natural to ask whether slivers can be eliminated with a small modification or in a post-processing step. Chew [25] prevents the creation of slivers by doing a random search for a suitable new vertex in a ball around the circumcenter

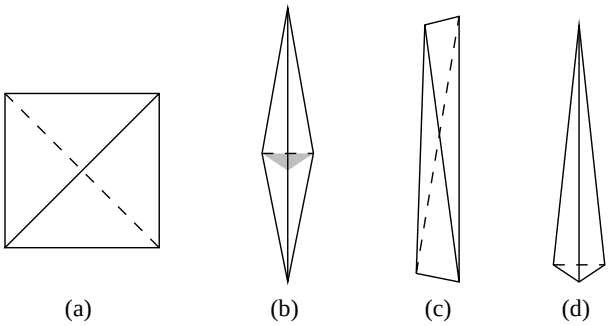

Figure 2.5: (a) Sliver tetrahedron: a good radius-edge ratio does not rule out poor dihedral angles (the four vertices are nearly coplanar). (b) Spear tetrahedron: a lower bound ($60°$) on dihedral angles does not rule out a very large dihedral angle. (c) Splinter tetrahedron: an upper bound ($90°$) on dihedral angles does not rule out very small dihedral angles. (d) Needle tetrahedron: bounds on both small ($60°$) and large ($90°$) dihedral angles do not rule out an arbitrary large radius-edge ratio.

of each bad quality tetrahedron. Cheng et al. [21] show that if the radius-edge ratios of the tetrahedra are bounded, then slivers can be eliminated by switching to a *weighted* Delaunay tetrahedralization and selecting the weights of the vertices in such a way that all slivers disappear, a process they call *sliver exudation*. Edelsbrunner et al. [33] show that smoothing (moving the vertices) can be used instead of weights. For these last two results, the authors meshed a periodic space to avoid having to deal with the domain boundary. This is a reasonable simplification in order to make progress on a very hard problem, which we take advantage of in Chapter 4. Nevertheless, handling of the boundary is important for applications, and has been accomplished by Li and Teng [53] by improving upon the algorithm sketched by Chew [25], and by Cheng and Dey [20] who extended the work of Cheng et al. [21]. The relationships between these results are illustrated in Figure 2.6.

The weakness of all these results is that the actual dihedral angle bounds, while positive, are too minuscule to be worth computing explicitly: they are probably less than 10^{-6} degrees. However, experiments by Edelsbrunner and Guoy [32] show that the sliver exudation algorithm of Cheng et al. [21] can eliminate most slivers with dihedral angles below $5°$ in practice.

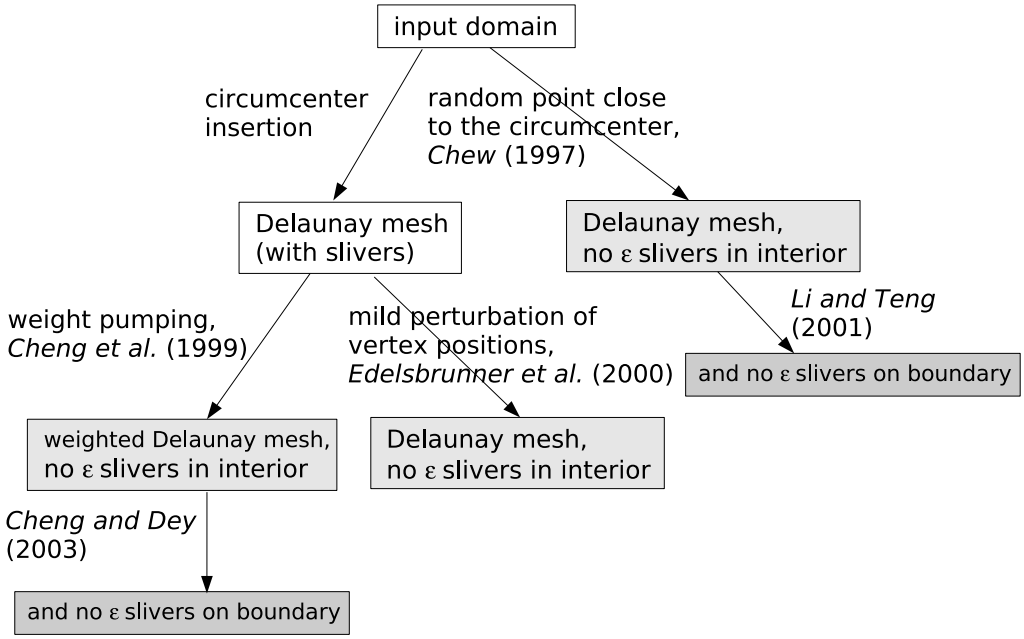

Figure 2.6: Previous theoretical work on sliver removal. The guarantees rule out microscopic angles only (ϵ slivers).

Smooth Domains

The Delaunay refinement algorithm of Oudot, Rineau, and Yvinec [69] guarantees radius-edge ratios of at most $4+\epsilon$ for arbitrary $\epsilon > 0$. It relies on sliver exudation [21] to remove poor tetrahedra from meshes. The algorithm is an extension of the earlier of work of Boissonnat and Oudot [14] which produced a surface mesh only. In both versions, surface sampling is adapted to the distance to the medial axis d_M, which leads to theoretical guarantees on topological equivalence, Hausdorff distance, normal approximation, and size-optimality. Quantitatively, points with inter-distance as small as $0.03d_M$ might need to appear in the mesh in order to guarantee topology.

Variational tetrahedral meshing, by Alliez, Cohen-Steiner, Yvinec, and Desbrun [1], is at the core a mesh improvement procedure where connectivity and vertex positions are optimized to minimize an energy function. The energy function itself has nice theoretical properties, but there is no theoretical result on the quality of a mesh that minimizes this energy. To obtain graded meshes, the energy must be modified to incorporate a sizing field. The authors turn the mesh improvement procedure into a meshing algorithm by presenting a way to create the initial mesh needed to start the optimization procedure.

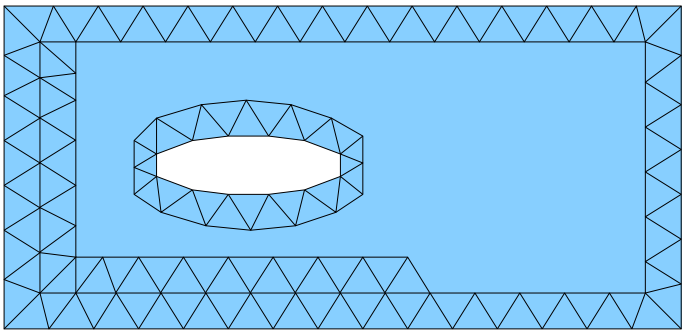

Figure 2.7: The advancing front method generates elements starting from the boundary. In this example, the second layer of elements is progressing, and we can foresee the complications that will occur later when fronts collide.

2.4 Other Mesh Generation Techniques

This section briefly covers other techniques which are popular or practical, although unrelated to this thesis.

Advancing Front

In the *advancing front method* [58], the domain boundary is first segmented (or triangulated if the domain is three dimensional). Elements are generated starting from the boundary, usually layer by layer, until the whole domain is meshed (See Figure 2.7). This method tends to create good quality elements close to the boundary where new vertices can be created at near optimal positions. The situation is much more difficult toward the end of the process where many fronts collide at the same time, possibly with a mismatch in element sizes. The method is popular in aerodynamics where mesh quality close to the boundary is important. There are no mathematical guarantees on mesh quality.

Biting

The *biting method* introduced by Li et al. [56] is a method related to advancing front and circle packing. The domain is iteratively "eaten" by disks until it is completely consumed. The mesh is obtained from a Delaunay triangulation of the disk centers. An advantage of the biting method is that it provides guarantees on triangle quality and size-optimality of the mesh (Section 1.2.2). The method has been extended to anisotropic meshes with biting ellipses [54], and to three dimensions with biting spheres [55]. The biting sphere

algorithm suffers from the presence of slivers, just like three-dimensional Delaunay refinement.

Mesh Improvement

Mesh improvement is technically not a mesh generation technique, but it can be used to improve meshes that were obtained by other methods, potentially turning a bad quality mesh into a good one.

Two aspects of a mesh can be improved: its geometry and its topology. Geometrical improvement, often simply called *smoothing*, consists in optimizing the position of mesh vertices without changing the mesh connectivity. An early and simple example is *Laplacian smoothing* [46], where a vertex is moved to the center of mass of its neighbors (unless doing this would create an inverted element). Typically, a few passes of Laplacian smoothing can be performed. Superior methods include nonsmooth optimization [41] and variational approaches [19].

Topological improvement consists in making local operations that change the connectivity of the mesh. One strategy is to consider a large set of topological transformations and perform those that improve the degree of vertices [17] or any other quality measure. The best results are obtained by combining smoothing and topological transformations. Freitag and Ollivier-Gooch [41] achieve better results through optimization-based smoothing and topological transformations than Edelsbrunner and Guoy with sliver exudation, but dihedral angles less than 1° sometimes survive, and in many examples dihedral angles under 10° survive. Alliez et al. [1] claim good results by combining variational smoothing and Delaunay tetrahedralization, but it is not clear that they can consistently avoid boundary slivers.

Chapter 3

Tetrahedral Meshing Inside an Isosurface

In this chapter, we describe an algorithm that we call *isosurface stuffing* that can create a tetrahedral mesh inside an isosurface.

We assume that the input is a continuous *cut function* $f : \mathbf{R}^3 \rightarrow \mathbf{R}$ 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.

Because our algorithm 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 numerical procedure more delicate than using iterated bisection to find a zero of a function of one variable. A *streaming* implementation is possible, i.e. one that accepts the values of f in any order and outputs tetrahedra as they are ready. If the input order is reasonable then huge models can be processed with little memory [47].

A second version of isosurface stuffing (Section 3.5) creates meshes whose interior tetrahedra are *graded*—they grade from largest at the core to smallest at the surface, as Figure 3.1 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

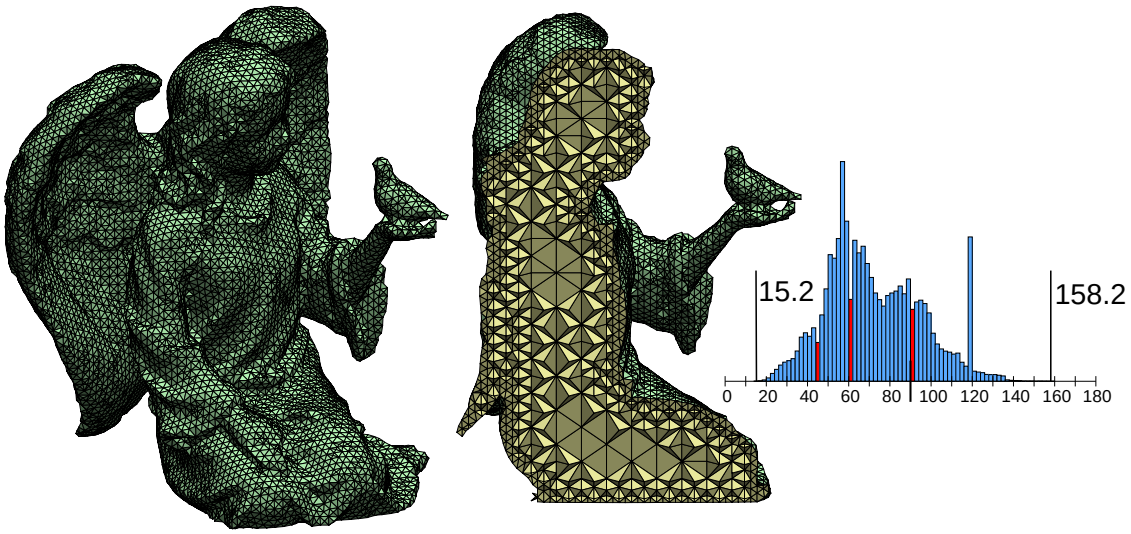

Figure 3.1: A 134,400-tetrahedron mesh produced by isosurface stuffing, with a cutaway view. At the right is a histogram of tetrahedron dihedral angles in $2°$ intervals; multiply the heights of the red bars by 20. (Angles of $45°$, $60°$, and $90°$ occur with high frequency.) The extreme dihedral angles 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.

are unable to guarantee dihedral angles better than $1.66°$ degrees for the worst tetrahedra, so we do not report details. (But see Section 3.6 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.

3.1 Uniform Tetrahedra

The isosurface stuffing algorithm uses the *body centered cubic (BCC) lattice* as a space-tiling background grid to guide the creation of a mesh.

The Delaunay triangulation of these points, illustrated in Figure 3.2, is a tetrahedral

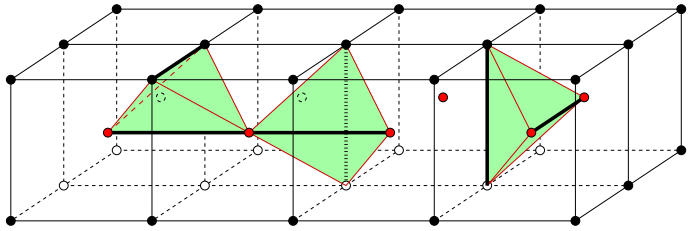

Figure 3.2: The body centered cubic (BCC) lattice is composed of two staggered cubical grids of vertices. The three tetrahedra illustrated here (which are identical) and copies of them tile space.

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 [81]. 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,

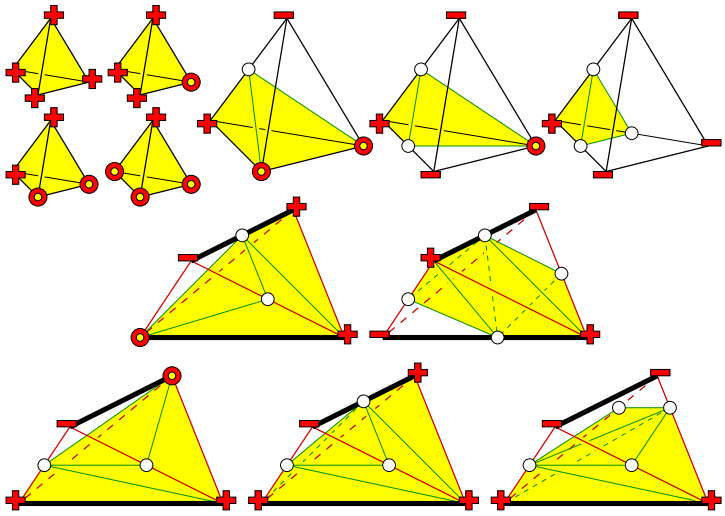

Figure 3.3: Stencils for isosurface stuffing. Vertices of the BCC grid tetrahedra are labeled with their signs ($+$, $-$, 0). Cut points are white, and output tetrahedra are yellow. The seven stencils in the top row apply in all rotations and reflections, and their edges can be matched arbitrarily with 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.

fill the tetrahedron (which might be warped) with a stencil of 1–3 precomputed tetrahedra. Output these tetrahedra. Figure 3.3 depicts the stencils. The choice of stencil depends on the signs of the four vertices of the BCC grid tetrahedron.

These steps are illustrated in Figure 3.4 and are described in more details in the following sections.

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.

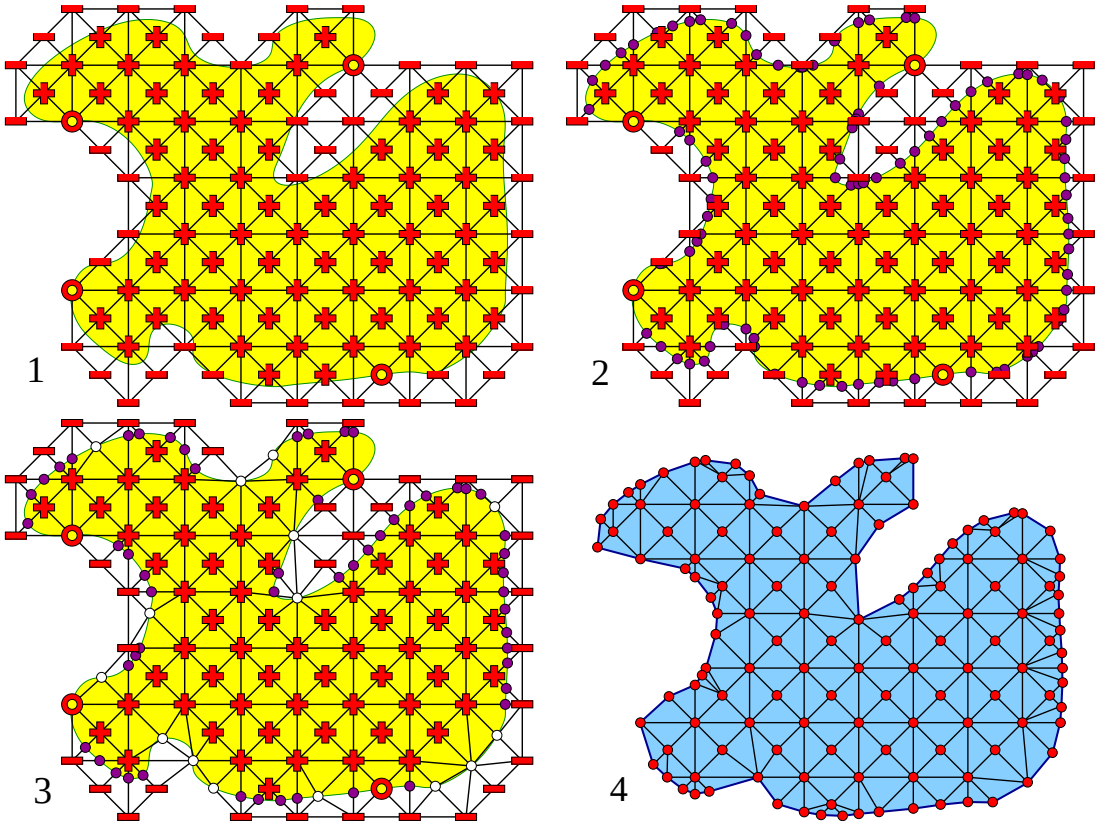

Figure 3.4: A two-dimensional illustration of the steps of isosurface stuffing: (1) Choose a subset of the BCC lattice and store the value of the cut function at each lattice vertex. (2) Compute cut points where edges cross the isosurface. (3) Warp the lattice points that are violated, shown as hollow vertices. (4) Triangulate the warped background mesh using stencils.

3.1.1 Creating the Background Grid

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

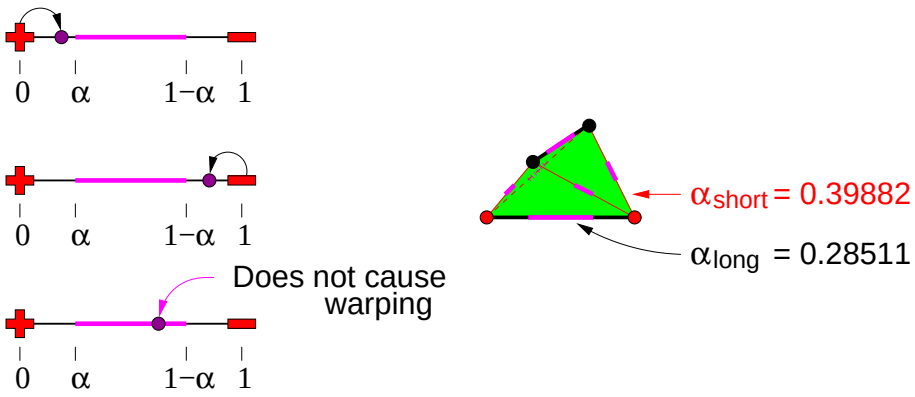

Figure 3.5: On the left: three parts of an edge where a cut point can fall. The first two cases trigger a lattice point to warp, while the third does not. The lengths of the parts are determined by a parameter α, which can vary depending on the edge. On the right: the α parameters from the second line of Table 3.1 are illustrated.

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.)

3.1.2 Computing Cut Points

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

3.1.3 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. The warping criterion is illustrated in Figure 3.5 (left), and

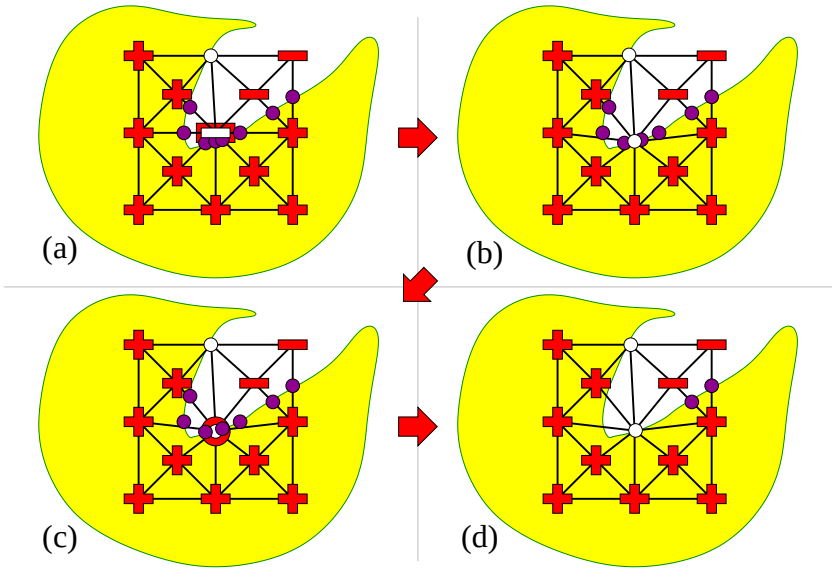

Figure 3.6: Warping lattice points: (a) The lattice point at the center is violated because a cut point lies too close to it. (b) The lattice point is snapped to the position of the cut point. (c) Because the new position lies on the isosurface, the lattice point is reassigned a value of zero. (d) The cut points that were adjoining are removed.

the warping process in Figure 3.6.

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 3.1, and one is illustrated in Figure 3.5 (right). 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*. Angle bounds are given in Table 3.2, they are derived with a computer-assisted proof, discussed in Section 3.3.

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 3.1 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

	α_{long}	α_{short}	what is optimized	safe?
1	0.26649	0.36918	maximum dihedral angle	unsafe
2	0.28511	0.39882	minimum dihedral angle	unsafe
3	0.24999	0.40173	maximum dihedral angle	safe
4	0.24999	0.41189	minimum dihedral angle	safe
5	0.24999	0.42978	. . . with ordered warping	safe
6	0.21509	0.35900	max dihedral, double-sided	safe
7	0.22383	0.39700	min dihedral, double-sided	safe
8	0.22385	0.40501	. . . with ordered warping	safe
9	0.23926	0.27376	max exposed plane angle	safe
10	0.23463	0.29505	. . . with ordered warping	safe
11	0.36378	0.33951	min exposed plane angle	unsafe
12	0.24999	0.35464	min exposed plane angle	safe
13	0.23573	0.5	. . . with ordered warping	safe

Table 3.1: 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. 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 3.1.3. The bottom five rows are of interest mainly for surface meshing; see Section 3.6.

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 dependencies 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.

	minimum dihedral angle	maximum dihedral angle	minimum plane angle	maximum plane angle	minimum exposed plane	maximum exposed plane
1	8.9716	*__158.7403__	11.9072	150.9944	12.0162	147.6786
2	*__10.7843__	164.7373	9.0454	154.9845	9.0454	154.9845
3	9.0551	__160.5331__	8.7614	155.7053	8.7614	155.7053
4	__9.3171__	161.6432	7.7810	158.2252	7.7810	158.2252
5	__9.7766__	163.5685	10.5695	149.7137	15.1645	138.1929
6	6.4917	__164.1013__	8.8535	157.8278	13.0689	145.1886
7	__7.6872__	168.0481	9.2237	155.0594	9.2237	154.5340
8	__7.8653__	168.0572	9.5400	154.6644	14.4726	135.7164
9	5.3440	163.8969	6.2646	158.2960	11.8387	*__124.9195__
10	5.8017	162.1673	7.2694	158.0368	12.1108	*__124.0867__
11	n/a	n/a	10.4741	†149.6794	*__15.1285__	*149.5205
12	7.8390	160.5447	10.4213	153.7863	__13.5241__	144.1259
13	7.4904	169.1465	9.2685	†145.4921	__16.4299__	144.9032

Table 3.2: For the choices of α_{long} and α_{short} of the previous table, these 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. Asterisks and daggers are explained in Section 3.5.3. All angle bounds have been computer-verified to be strictly correct as written and tight to within 0.0001°.

3.1.4 Triangulating the Background Grid

The fourth step generates the output tetrahedra specified by the stencils depicted in Figure 3.3. 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 3.3 obey this rule.

The Parity 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 $\mathbf{Z}^3$ (the black points in Figure 3.2), 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 3.3, 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 $\mathbf{Z}^3 + (\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$ (the red points in Figure 3.2), 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 3.3 after looking it up.

Although isosurface stuffing could be implemented so that it numerically measures angles and uses them to choose stencils, we think much of the algorithm's charm is its ability to ensure quality with an absolute minimum of geometric computation.

Quadruple-Zero Tetrahedra

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 3.1 (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 3.3) 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. [65] 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 3.2, 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 3.1 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 3.3. 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 Section 3.4.2.) 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.)

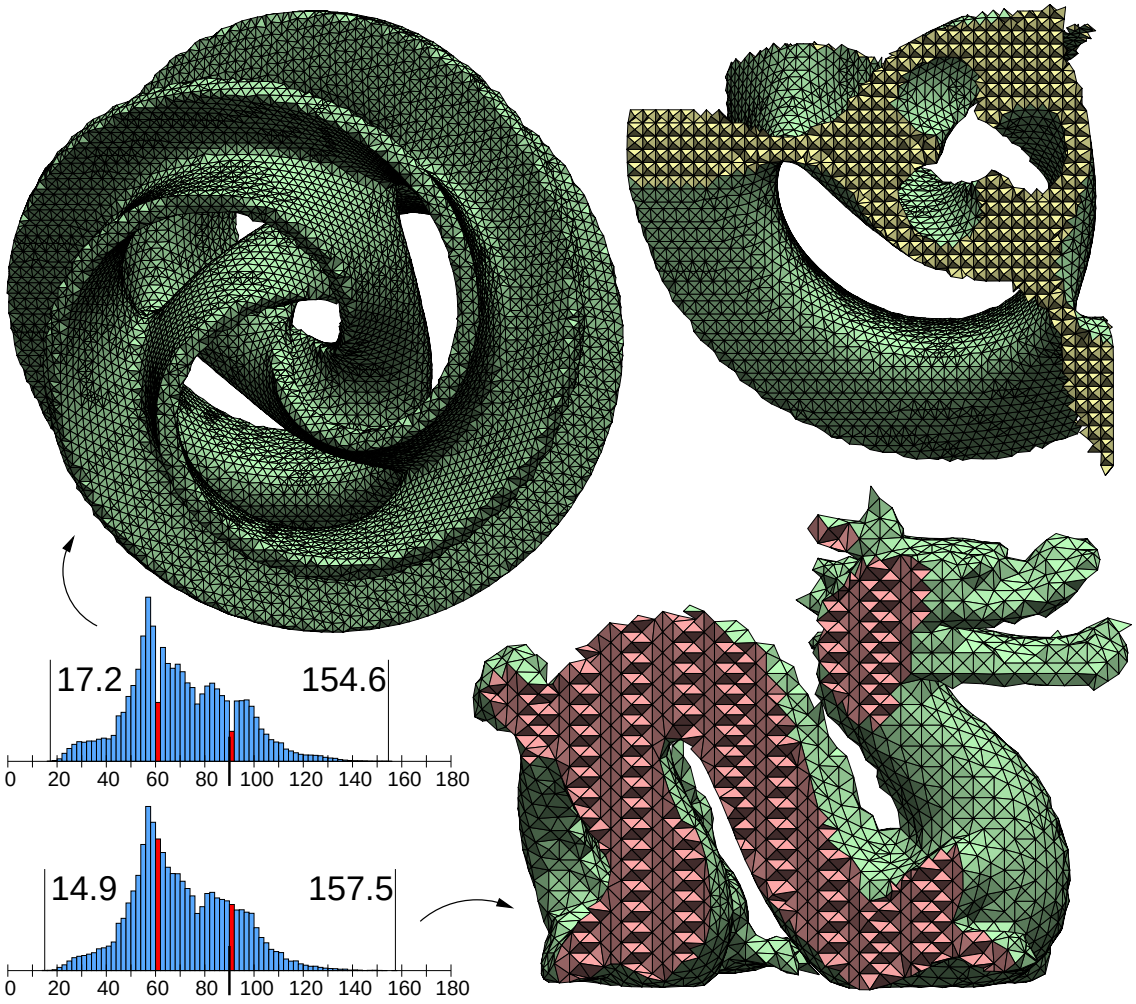

Figure 3.7: Meshes of uniformly sized tetrahedra produced by isosurface stuffing with $\alpha_{\text{long}} = 0.28511$ and $\alpha_{\text{short}} = 0.39882$. *Whirled White Web* is by courtesy of Carlo Séquin. Histograms tabulate the dihedral angles in 2° intervals; multiply the heights of the red bars by 20.

3.2 Mesh Examples

Figure 3.7 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.

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, Jonathan Shewchuk's 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.

3.3 Computer-Assisted Proofs of Angle Bounds

Isosurface stuffing guarantees that the tetrahedra it produces have good angles.

Theorem 1 *The bounds in Table 3.2 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, 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 3.8. 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.

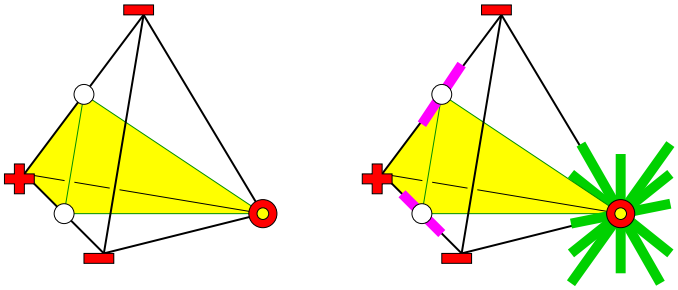

Figure 3.8: On the left, a tetrahedron from one of the stencils. On the right, the possible positions where the tetrahedron vertices can lie: vertices labeled $+$ or $-$ are fixed, cut points lie on the central part of an edge, and vertices labeled zero can be warped and lie on an asterisk which involves seven separate cases.

- 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; see Figure 3.8. 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.

Figure 3.9 illustrates the most general case, where we ask for the minimum and maximum possible dihedral angle in a tetrahedron where all four vertices lie on different segments. Call a_0 and a_1 the endpoints of the segment where vertex a lies, and similarly for the three other vertices. Then, the positions of a, b, c, and d can be parametrized as

$$\begin{aligned} a &= a_0 + \lambda_1(a_1 - a_0) \\ b &= b_0 + \lambda_2(b_1 - b_0) \\ c &= c_0 + \lambda_3(c_1 - c_0) \\ d &= d_0 + \lambda_4(d_1 - d_0) \end{aligned}$$

where $\lambda_1, \lambda_2, \lambda_3, \lambda_4 \in [0, 1]$. The dihedral angle at edge ab is some function

$$f(\lambda_1, \lambda_2, \lambda_3, \lambda_4),$$

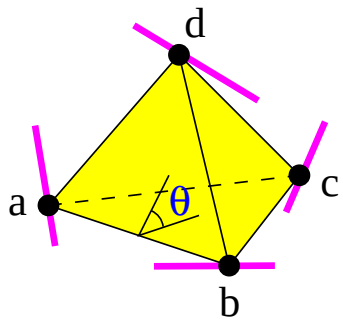

Figure 3.9: What is the minimum (or maximum) possible dihedral angle at edge ab if all four vertices of the $abcd$ tetrahedron lie on given segments?

and we are asking for the extrema of that function over $[0,1]^4$. To be able to answer that question, we first develop some machinery.

3.3.1 Extrema of a Function of n Variables

Let $f(\lambda_1, \ldots, \lambda_n)$ be a function of n variables for which the global minimum and maximum over $[0,1]^n$ are sought. We consider several cases.

1. If f is a polynomial of arbitrary degree, but whose degree is at most 1 in each of $\lambda_1, \ldots, \lambda_n$, such as $f = \lambda_1 + \lambda_1\lambda_2 + \lambda_1\lambda_2\lambda_3$, then the minimum and maximum can be found by evaluating the 2^n corners of the box $[0,1]^n$.

2. If f is a polynomial of degree 2, such as $f = \lambda_1^2 + \lambda_2$, then the extrema could be one of the 2^n corners, or somewhere along a boundary edge, a boundary face, or a boundary entity of some higher dimension, or inside the n-dimensional volume. We can find local extrema along axis-aligned lines, planes and other subspaces by computing partial derivatives (which are of degree 1), setting them to 0, and solving the linear equation or system. Solutions that fall outside of $[0,1]^n$ are excluded. By comparing the function f at every local extremum found, the global extrema are obtained.

3. If f is the ratio of two polynomials of degree 1, such as $f = (\lambda_1+\lambda_2)/(\lambda_1-\lambda_2+1)$, then partial derivatives will be rational functions with numerators of degree 1. Since only the numerator matters when setting a ratio to 0, the local extrema can be found by solving linear systems as before, and the global extrema are obtained. This works for any f such that poles in $[0,1]^n$ can be ruled out, and the partial derivatives written as a ratio where the numerator is a polynomial of degree 1.

4. If f is a polynomial of degree 3 or higher, such as $f = \lambda_1^2\lambda_2$, then the partial derivatives are of degree 2 or higher, and the local extrema can be found by solving non-linear systems in up to n variables. If f is non-polynomial, then the partial derivatives are non-linear also.

While solving non-linear systems is possible in practice, remember that the goal is not only to compute extrema but to *prove* them. Even when there are few variables, detecting and computing the intersection of curves and surfaces is numerically sensitive, and code that must do this with absolute correctness is unavoidably complicated and hard to verify [36].

We conclude that cases 1-3 above represent the most complicated functions of n variables for which the extrema over the region $[0,1]^n$ can be easily and provably found.

3.3.2 Extrema of Elementary Geometric Computations

Assume the general case where the four vertices of a tetrahedron $abcd$ lie on segments a_0a_1, b_0b_1, c_0c_1, d_0d_1, leading to the parametrization

$$\begin{aligned} a &= a_0 + \lambda_1(a_1 - a_0) \\ b &= b_0 + \lambda_2(b_1 - b_0) \\ c &= c_0 + \lambda_3(c_1 - c_0) \\ d &= d_0 + \lambda_4(d_1 - d_0) \end{aligned}$$

where $\lambda_1, \lambda_2, \lambda_3, \lambda_4 \in [0,1]$.

Let $u = b - a$, $v = c - a$, and $w = d - a$. The components of these vectors are polynomials of degree 1 in $\lambda_1, \cdots, \lambda_4$, so dot products (such as $u \cdot v$) and the components of cross products are polynomials of degree 2 and their extrema can be found by the method described in the previous section.

An interesting case is that of the scalar triple product

$$[u, v, w] = u \cdot (v \times w) = v \cdot (w \times u) = w \cdot (u \times v).$$

It has degree 3 in $\lambda_1 \ldots \lambda_4$, but it is a special function (equal to 6 times the signed volume of the tetrahedron $abcd$) given by the determinants

$$[u, v, w] = \begin{vmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{vmatrix} = \begin{vmatrix} 1 & a_x & a_y & a_z \\ 1 & b_x & b_y & b_z \\ 1 & c_x & c_y & c_z \\ 1 & d_x & d_y & d_z \end{vmatrix}.$$

From this last expression, it is clear that this function has degree 1 in each of $\lambda_1, \ldots, \lambda_4$ when the three other variables are fixed. This means that extrema of the scalar triple

product $[u, v, w]$ can be found by comparing the values at the 16 combinations of segment endpoints.

In summary, the extrema of simple geometric expressions in u, v, w such as dot products, components of cross products, and the scalar triple product $[u, v, w]$ can be found easily. The extrema of a more complicated geometric expression cannot, but if we consider the minimum and maximum of an expression to be an interval, we can use interval arithmetic to compute conservative bounds on the minimum and maximum of complicated expressions. To do this efficiently, it is important to express a complicated expression using as few of the simple geometric expressions as possible.

3.3.3 Dihedral Angles

Suppose a tetrahedron has vertices a, b, c, and d. Let $u = b - a$, $v = c - a$, and $w = d - a$. Then normals to the triangular faces abc and abd are given by $u \times v$ and $u \times w$, respectively. The dihedral angle θ at edge ab is the angle between these two normals, and can be derived using the dot product formula

$$\cos\theta = \frac{(u \times v) \cdot (u \times w)}{\|u \times v\| \, \|u \times w\|}.$$

There are other formulas for the dihedral angle, such as

$$\sin\theta = \frac{[u, v, w] \, \|u\|}{\|u \times v\| \, \|u \times w\|}, \quad \tan\theta = \frac{[u, v, w] \, \|u\|}{(u \times v) \cdot (u \times w)}.$$

Remember that a cross product is expensive, with three components in disguise. They can be eliminated using the "$\epsilon - \delta$ rule",

$$(p \times q) \cdot (r \times s) = (p \cdot r)(q \cdot s) - (p \cdot s)(q \cdot r).$$

By applying it to the tangent equation, we derive our final formula for the dihedral angle, written so that every expression in parentheses corresponds to a "simple expression" of the previous section, whose extrema can be computed directly. Because $\tan\theta$ has a discontinuity at $90°$, it is important to turn it into an inverse cotangent formula:

$$\theta = \cot^{-1} \frac{\sqrt{(u \cdot u)}(v \cdot w) - (u \cdot v)(u \cdot w)/\sqrt{(u \cdot u)}}{[u, v, w]}. \tag{3.1}$$

In our problem, each vertex of the tetrahedron $abcd$ is constrained to lie on a segment (different for each vertex). Suppose that the four vertices cannot be coplanar under that constraint. (This can be verified by computing the minimum and maximum of the scalar

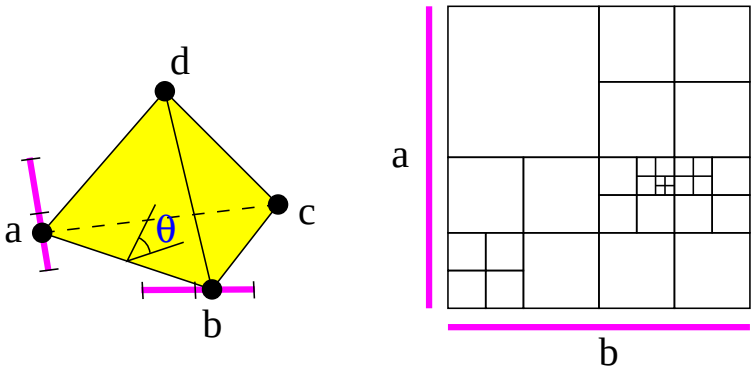

Figure 3.10: To find the dihedral angle extrema, recursively subdivide the parameter space, bound angles in each range with interval arithmetic, and stop when the bound accuracy is 0.0001°

triple product $[u, v, w]$, and making sure that they have the same sign.) Then it is easy to see 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 using (3.1) 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. Figure 3.10 illustrates this.

By this means, we have verified all the dihedral angle bounds in Table 3.2. The cases that limit our bound on the smallest dihedral angle to 10.7843° degrees appear in Figure 3.11.

3.3.4 Plane Angles

Using the same notation as in the previous section, formulas for the plane angle $\varphi = \angle bac$ (illustrated in Figure 3.12) are

$$\cos\varphi = \frac{u \cdot v}{\|u\|\,\|v\|}, \quad \sin\varphi = \frac{\|u \times v\|}{\|u\|\,\|v\|}, \quad \tan\varphi = \frac{\|u \times v\|}{u \cdot v}.$$

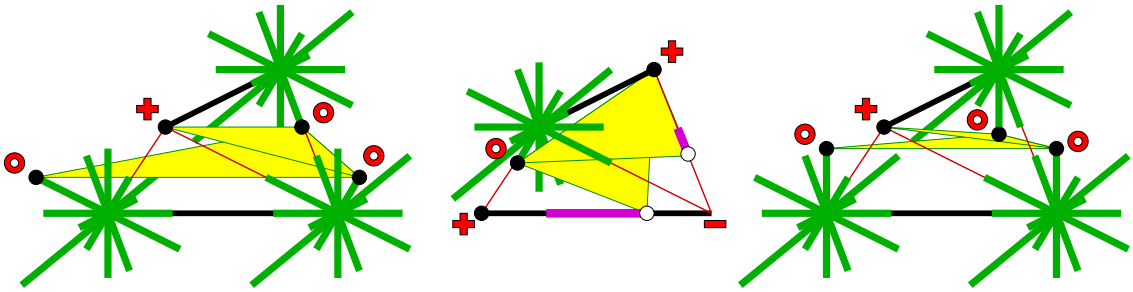

Figure 3.11: Some of the limiting cases in which a dihedral angle of $10.7843°$ arises (where the two yellow triangles meet). Warped background vertices must lie on their green asterisks. Cut points must lie on the magenta segments.

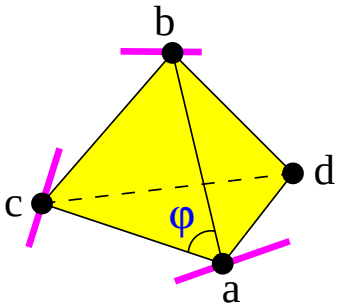

Figure 3.12: What is the minimum (or maximum) possible plane angle φ if all four vertices of the $abcd$ tetrahedron lie on given segments? Obviously φ doesn't depend on the position of d, but it is not possible to argue that angle extrema must be achieved at the endpoints of segments at a, b, or c. The parameter space is thus three-dimensional.

By expanding the cross product in the tangent formula, we obtain our final expression for the plane angle. Again, every expression in parentheses is a "simple expression", and we transform the formula to use a cotangent:

$$\varphi = \cot^{-1} \frac{(u \cdot v)}{\sqrt{(u_x v_y - u_y v_x)^2 + (u_y v_z - u_z v_y)^2 + (u_z v_x - u_x v_z)^2}}. \tag{3.2}$$

To verify plane angle bounds, we subdivide a three-dimensional parameter space with an octree, and use (3.2) with interval arithmetic. By this means, we have verified all the plane angle bounds in Table 3.2. This concludes the overview of the proof of Theorem 1.

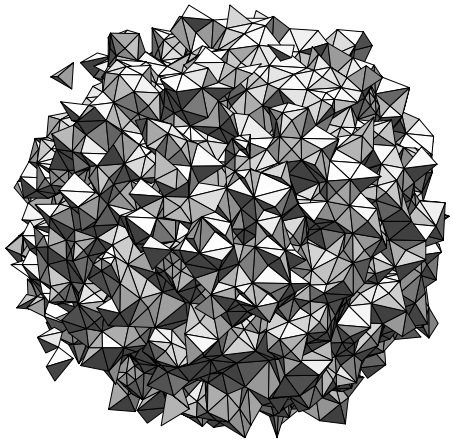

Figure 3.13: Isosurface triangulation torture. We mesh a random cut "function" with a spherical probability distribution. Despite having nonsense for input, the algorithm created a 16,240-tetrahedron mesh with all its dihedral angles between $13.6°$ and $147.5°$.

3.4 Approximation Guarantees

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. In this section, we give some approximation results that hold for an arbitrary isosurface, and some stronger results that hold if the isosurface has bounded curvature.

3.4.1 Arbitrary Isosurfaces

An interesting feature of our algorithm is that it can be run on an arbitrary isosurface. This means that, in principle, we can run it with a cut function that returns a random value in response to any probe; see Figure 3.13.

Hausdorff Distance

The one-sided Hausdorff distance between two subsets A and B in any metric space is defined to be

$$H_*(A, B) = \sup_{a \in A} d(a, B),$$

where $d(a, B)$ is the distance from the point a to the set B, given by $\inf_{b \in B} d(a, b)$. A consequence is that every point in A is at most at distance $H_*(A, B)$ from B. Another

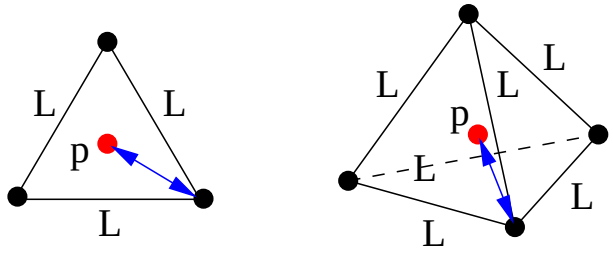

Figure 3.14: A point p lies as far as possible from the vertices of a triangle with maximum edge length L. In the worst case, the distance is $L/\sqrt{3}$. For a tetrahedron, the distance is $\sqrt{3/8}L$, and in general for a simplex of dimension d, the distance is $\sqrt{d/(2d+2)}L$.

way to think of it is that if we were to increase the thickness of B by brushing it with a sphere of radius $H_*(A, B)$ centered at points of B, then A would be completely included in it.

The expression is not symmetric in A and B. The full (symmetrized) Hausdorff distance is

$$H(A, B) = \max\{H_*(A, B), H_*(B, A)\}.$$

One-Sided Hausdorff Distance Between Surfaces

We start with a very simple general result, and then apply it to isosurface stuffing.

Theorem 2 *If a surface S is approximated by triangles whose vertices lie exactly on S, and there is an upper bound L on triangle edge lengths, then the one-sided Hausdorff distance between the approximation T and the surface S satisfies $H_*(T, S) \leq L/\sqrt{3}$.*

PROOF: Given a point p on the triangulated surface T, p lies in some triangle $\triangle abc$. Since L is an upper bound on edge lengths, p is at distance at most $L/\sqrt{3}$ from a vertex of the triangle (the worst case is depicted in Figure 3.14, left). Since vertices of the triangle are assumed to lie exactly on the surface S, $d(p, S) \leq L/\sqrt{3}$. Since p was arbitrary, the result follows. □

It is clear from the description of isosurface stuffing that every vertex on the boundary of the mesh is a cut point or is labeled zero, and therefore lies on the isosurface. For a mesh produced by our algorithm with a unit BCC background grid,

$$L = \max\{1 + 2\alpha_{\text{long}}, \frac{\sqrt{3}}{2}(1 + 2\alpha_{\text{short}})\}.$$

A straightforward application of Theorem 2 gives the following result.

Corollary 3 *For a mesh produced by isosurface stuffing with a unit* BCC *background grid, every point on the mesh boundary is within a distance* ω_{s} *from the zero-surface, where*

$$\omega_{\mathrm{s}} = L/\sqrt{3} = \max\{\frac{1+2\alpha_{\mathrm{long}}}{\sqrt{3}}, \frac{1}{2} + \alpha_{\mathrm{short}}\}.$$

For example, if $\alpha_{\mathrm{long}} = 0.28511$ and $\alpha_{\mathrm{short}} = 0.39882$—the parameters from the second line in Table 3.1, then $L = 1.57022$ and $\omega_{\mathrm{s}} = 0.90657$.

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.

The bound applies for the unscaled BCC lattice. If the BCC lattice is scaled by a factor c, then the bound is scaled by c also. As $c \to 0$, the greatest distance between a mesh boundary point and its nearest neighbor on the zero-surface converges to zero.

One-Sided Hausdorff Distance Between Volumes

Since Hausdorff distance can apply to arbitrary subsets, it can apply to volumes too. The results are similar:

Theorem 4 *Suppose a volume* V *is approximated by a mesh* A *of tetrahedra whose vertices are correctly labeled by the volume, and are labeled* 0 *(boundary) or* $+$ *(inside). If there is an upper bound* L *on tetrahedron edge lengths, then the one-sided Hausdorff distance between the approximation* A *and the volume* V *satisfies* $H_*(A, V) \leq \sqrt{3/8}L$.

PROOF: Given a point p in the approximation mesh A, p lies in some tetrahedron $abcd$. Since L is an upper bound on edge lengths, p is at distance at most $\sqrt{3/8}L$ from a vertex of the tetrahedron (the worst case is depicted in Figure 3.14, right). Since vertices of the tetrahedron are labeled 0 or $+$, they are part of the volume V, so $d(p, V) \leq \sqrt{3/8}L$. Since p was arbitrary, the result follows. □

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 and Theorem 4 applies with the same bound on edge length as before. Also if we assume that isosurface stuffing is used to generate compatible meshes on both sides on the isosurface, then the theorem and Hausdorff bounds apply between the exterior mesh and the closure of the complement volume $\mathbf{R}^3 \backslash V$. These considerations lead to the following result.

Corollary 5 *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*

$$\omega_{\mathrm{v}} = \sqrt{\frac{3}{8}}L = \sqrt{\frac{3}{8}}\max\{1 + 2\alpha_{\mathrm{long}}, \frac{\sqrt{3}}{2}(1 + 2\alpha_{\mathrm{short}})\}.$$

from the isosurface.

If $\alpha_{\mathrm{long}} = 0.28511$ and $\alpha_{\mathrm{short}} = 0.39882$, then $L = 1.57022$ as before and $H_*(A, V) \leq 0.96156$. By taking into account the special geometry of the BCC tetrahedron, a tighter bound is possible:

$$\omega_{\mathrm{v}} = \max\{\sqrt{\alpha_{\mathrm{long}}^2 + \alpha_{\mathrm{long}} + 5/16}, \frac{1}{2}\sqrt{3\alpha_{\mathrm{short}}^2 + 3\alpha_{\mathrm{short}} + 5/4}\}.$$

This bound also gives a tighter value of ω_{s} for surfaces, but an even tighter ω_{s} could be derived by analyzing the geometry of a BCC tetrahedron's face. The bound applied to the example α values above gives $H_*(A, V) \leq 0.85494$.

3.4.2 Isosurfaces with Bounded Curvature

Bounds will be given in terms of d_M, the minimum distance between the isosurface S and its medial axis M.

Medial Axis

The medial axis of a surface S is defined to be the set of points in space that have at least two closest points on S. The medial axis is illustrated in Figure 3.15.

Write $d_M(p) = d(p, M)$ for $p \in S$, and let $d_M = d(S, M)$ be the minimum value that $d_M(p)$ can attain—a constant which depends on the surface. An isosurface with bounded curvature will have $d_M > 0$, but a surface with a corner or an edge will have $d_M = 0$.

An important property of d_M is that for any point $p \in S$, a sphere tangent to S at p (on either side) is empty. Furthermore, if S is the isosurface of some cut function, then the cut function is all positive inside one of the spheres, and all negative inside the other. Also note that $1/d_M$ is an upper bound on the curvature of S, but it isn't necessarily the maximum curvature because d_M might be determined where different parts of the surface come close to each other.

Assuming that d_M is bounded away from zero, the Delaunay-based surface approximation algorithm of Boissonnat and Oudot [14] and its extension [69] enjoy many surface

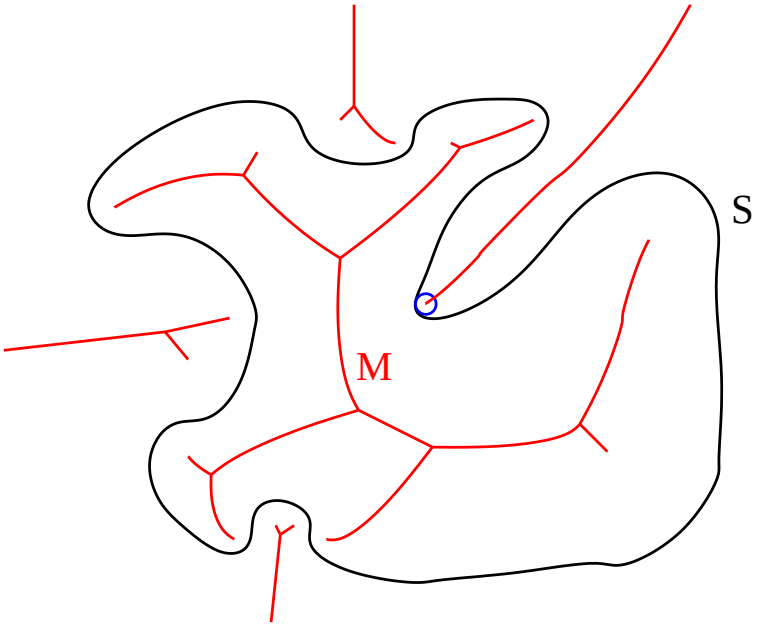

Figure 3.15: A two-dimensional representation of the medial axis M of a surface S. Also, a tangent sphere of radius d_M at the position where the minimum distance between the surface and the medial axis is attained.

approximation guarantees. Their guarantees follow from results by Amenta and Bern [2], which themselves depend on results by Edelsbrunner and Shah [34]. Unfortunately, all of this machinery is intimately linked with Delaunay properties that don't necessarily hold in our case. Fortunately, equivalent results can be proved directly.

Normal Approximation

Below is a general result, followed by its application to isosurface stuffing.

Theorem 6 *Let S be a surface and let $\triangle abc$ be a triangle whose vertices lie exactly on S. Assume that $|ab|, |ac| \leq L$, $\sin(\angle bac) \geq s$, and $d_M(a) \geq r$. Then the angle θ between the normal of $\triangle abc$ and the normal of S at a satisfies*

$$\sin\theta \leq \frac{L}{rs}.$$

PROOF: For ease of notation, translate and rotate the geometry so that $a = (0,0,0)$ and the plane tangent to S at a is the xy-plane (hence the normal is the z-axis). The fact that $b, c \in S$ and $d_M(a) \geq r$ implies that b and c lie on or outside spheres of radius r centered

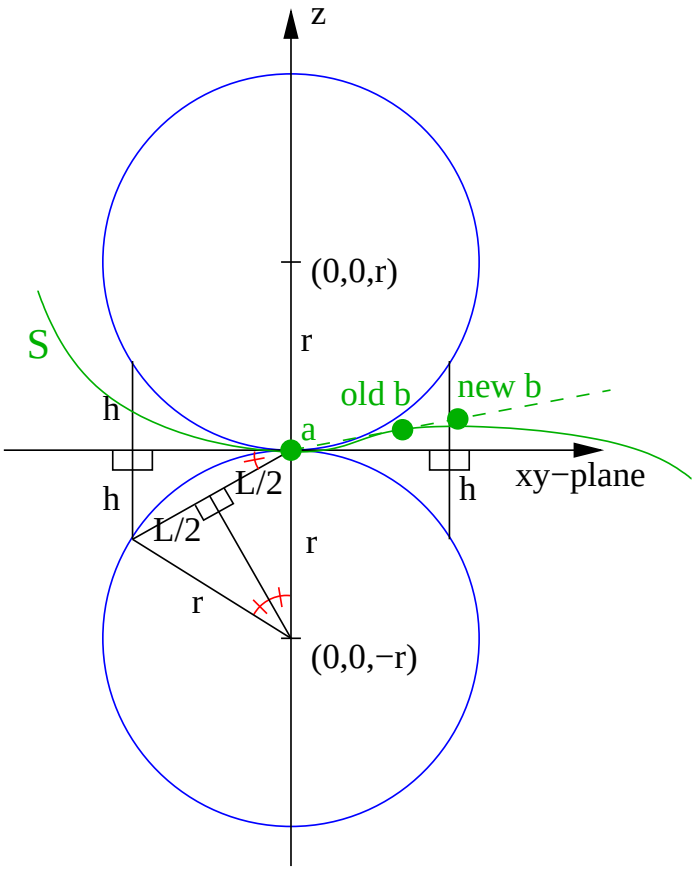

Figure 3.16: The vertices a and b are part of a triangle $\triangle abc$ whose vertices lie on a surface S which avoids two open spheres of radii r tangent at a. The vertex b is moved along the ray ab so that $|ab| = L$. The distance between b and the plane $z = 0$ is at most h, where $h = L^2/(2r)$ by geometry with similar triangles.

at $(0, 0, r)$ and $(0, 0, -r)$. Since this is the only property of b and c that will be used, move b and c along the rays $\overrightarrow{ab}$ and $\overrightarrow{ac}$ (respectively) so that $|ab| = |ac| = L$. The property still holds and the angle $\angle bac$ didn't change. Note also that the maximum distance between b and c and the plane $z = 0$ is $h = L^2/(2r)$ (see Figure 3.16).

If the triangle $\triangle abc$ is completely horizontal, then $\theta = 0$ and we are done. Otherwise, there exists a direction of maximal slope on the triangle, given by the projection of the z-direction onto the triangle plane. Draw a line on the triangle $\triangle abc$ in the direction of maximal slope so that the line goes through a vertex of the triangle and crosses the opposite edge. Call q the intersection point on the opposite edge.

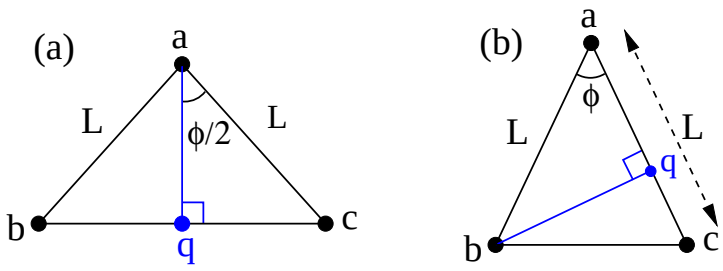

Figure 3.17: (a) A line goes through vertex a, exiting at q on the opposite edge. The shortest possible distance between a and q is $L\cos(\phi/2)$. (b) A line goes through vertex b, exiting at q on the opposite edge. The shortest possible distance between b and q is $L\sin\phi$.

- In case the line goes through vertex a, the maximum height difference between a and q is h. Let $\phi = \angle bac$. The minimum distance between a and q is

$$\begin{aligned} L\cos(\phi/2) &\geq L\cos(\phi/2)\sin(\phi/2) \\ &= L(\sin\phi)/2 \\ &\geq Ls/2 \end{aligned}$$

  See Figure 3.17(a). This implies that $\sin\theta \leq h/(Ls/2) = 2h/(Ls)$.

- In case the line goes through vertex b or c (without loss of generality assume b), the maximum height difference between b and q is $2h$. The minimum distance between b and q is $L\sin\phi \geq Ls$; see Figure 3.17(b). This implies that $\sin\theta \leq 2h/(Ls)$.

In both cases we have $\sin\theta \leq 2h/(Ls)$. Plugging in the expression $h = L^2/(2r)$ we get the result, $\sin\theta \leq L/(rs)$. □

For a mesh produced by our algorithm with a unit BCC background grid with parameters $\alpha_{\text{long}} = 0.28511$ and $\alpha_{\text{short}} = 0.39882$, the maximum edge length is $L = 1.57022$, the minimum sine of an exposed plane angle is $s = 0.15721$ (from Table 3.2), therefore the angle θ between the normal of $\triangle abc$ and the normal of S at a satisfies $\sin\theta \leq 9.9881/r$.

Note that if $L/(rs)$ is small, the bound given by Theorem 6 implies that the angle θ is close to $0°$ *or* close to $180°$. The $180°$ case represents an inversion of the normal direction and cannot be excluded based on the preconditions of Theorem 6 only.

Two-Sided Hausdorff Distance

Theorem 7 *Suppose isosurface stuffing uses a background BCC grid scaled by c to mesh a continuous cut function f whose zero-surface S is a smooth 2-manifold. Let $d_M > 0$*

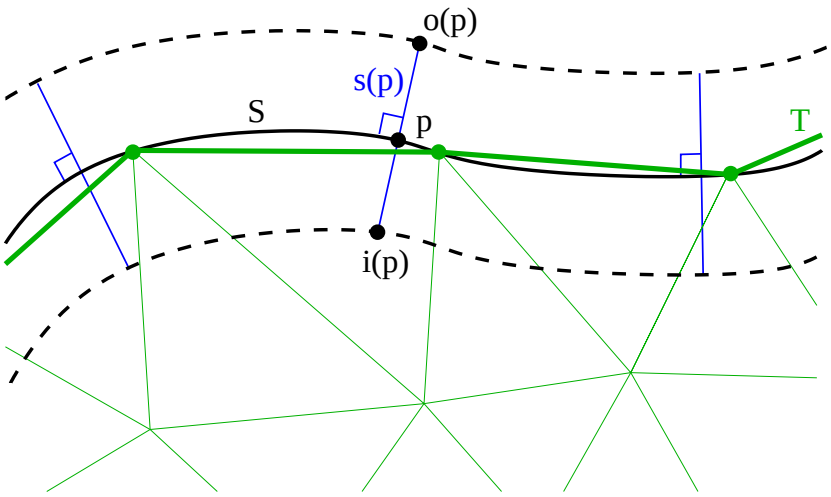

Figure 3.18: Three isotopy segments are shown, including the isotopy segment $s(p)$ corresponding to point p of the surface S. The union of all the isotopy segments forms an envelope around the surface, and the approximate surface T must lie inside the envelope.

be the shortest distance from a point on the zero-surface to a point on the medial axis of the zero-surface. If $d_M > \omega_v c$, with ω_v defined as in Corollary 5, then every point on the zero-surface is within a distance of $\omega_v c$ from the mesh boundary T.

PROOF: Suppose $d_M > \omega_v c$. For each point p on the zero-surface, let $i(p)$ be the point found by moving a distance infinitesimally greater than $\omega_v c$ 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. See Figure 3.18.

By Corollary 5, 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, we obtain the two-sided Hausdorff distance bound $H(T, S) \leq \omega_v c$ when $d_M > \omega_v c$. □

Topology

Theorem 8 *Assume the preconditions of Theorem 7. 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. We use the same isotopy segment construction as in the proof of Theorem 7. Our approach is similar to that of Kolluri [50]. We construct a continuous map $m : \mathbf{R}^3 \to \mathbf{R}^3$ 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 Corollary 3, every point on the mesh boundary is within a distance of $\omega_s c$ 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.

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 re-enters 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 6, or Theorem 5 of Amenta, Choi, Dey, and Leekha [3]). 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 [2]). We omit details. □

3.5 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 3.1 and 3.19. By reducing the number of tetrahedra in the mesh, we reduce the finite element

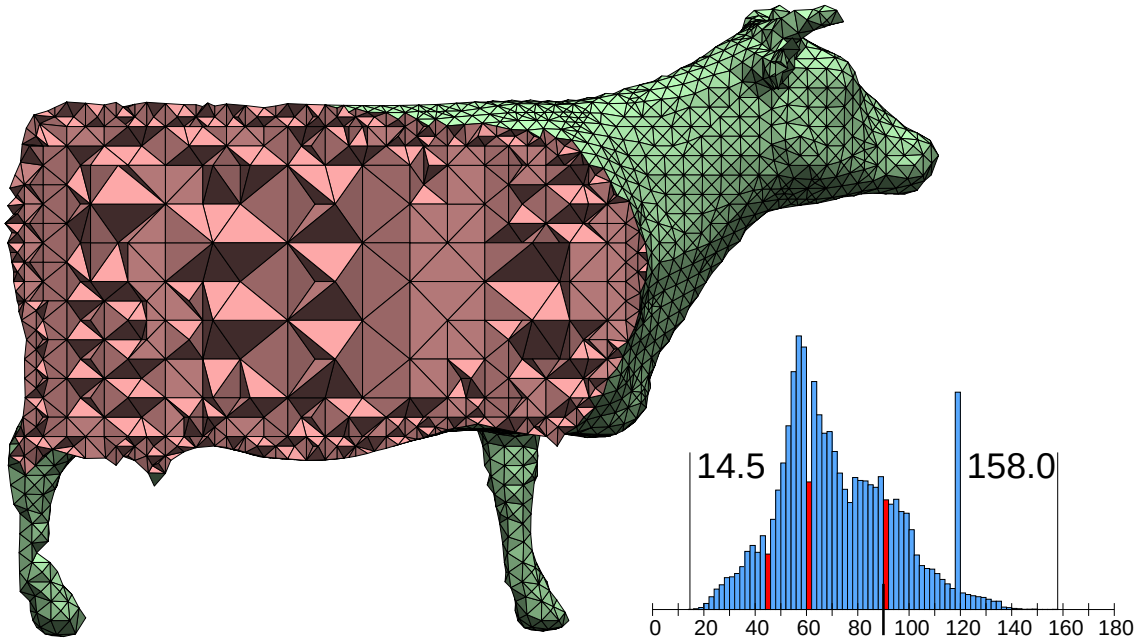

Figure 3.19: Cutaway view of a 42,053-tetrahedron mesh whose elements are uniformly fine on the surface but grade to coarse in the interior. Produced with $\alpha_{\text{long}} = 0.28511$ and $\alpha_{\text{short}} = 0.39882$. A histogram of dihedral angles in 2° intervals appears at lower right; multiply 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.

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 3.6.)

3.5.1 Four Kinds of Tetrahedra

We replace the BCC grid with a graded background grid composed of four kinds of tetrahedra, illustrated in Figure 3.20. 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. 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-

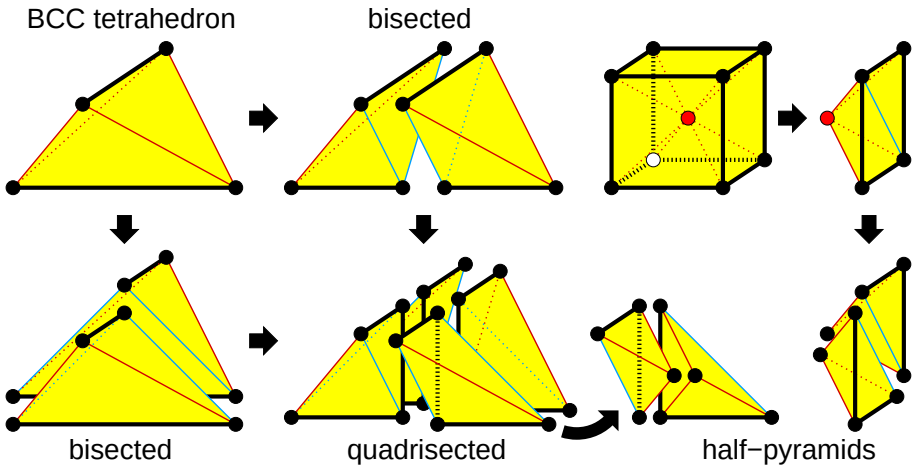

Figure 3.20: Graded background grids consist of BCC tetrahedra, bisected BCC tetrahedra, quadrisected BCC tetrahedra, and half-pyramids.

tetrahedron kind	dihedral angles	radius-edge ratio
BCC	$60°, 60°, 60°, 60°, 90°, 90°$	≈ 0.645
bisected	$45°, 60°, 60°, 90°, 90°, 90°$	≈ 1.118
quadrisected	$45°, 45°, 60°, 90°, 90°, 90°$	≈ 0.866
half-pyramid	$45°, 45°, 60°, 60°, 90°, 120°$	≈ 0.866

Table 3.3: Quality statistics on the four kinds of tetrahedra used to grade the interior of the mesh.

pyramids. Half-pyramids can also be obtained by bisecting the red edge of a quadrisected BCC tetrahedron. The dihedral angles and radius-edge ratios of these tetrahedra are listed in Table 3.3.

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).

3.5.2 Non-Standard Octree

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.

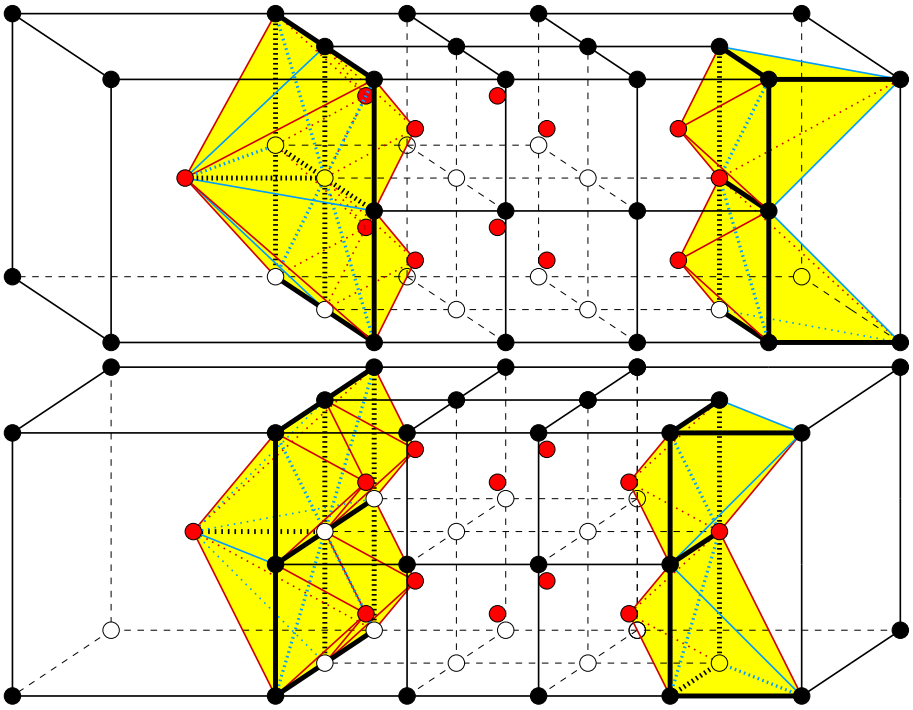

Figure 3.21: Background tetrahedra used to bridge two levels of the octree, viewed from two different angles. Cube centers are red.

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 3.21 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.

3.5.3 Conditions Close to the Isosurface

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 3.2 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 3.3, to show that these tetrahedra will not be warped enough to violate the angle bounds in Table 3.2, 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 3.22(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 [13] 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 3.2, 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

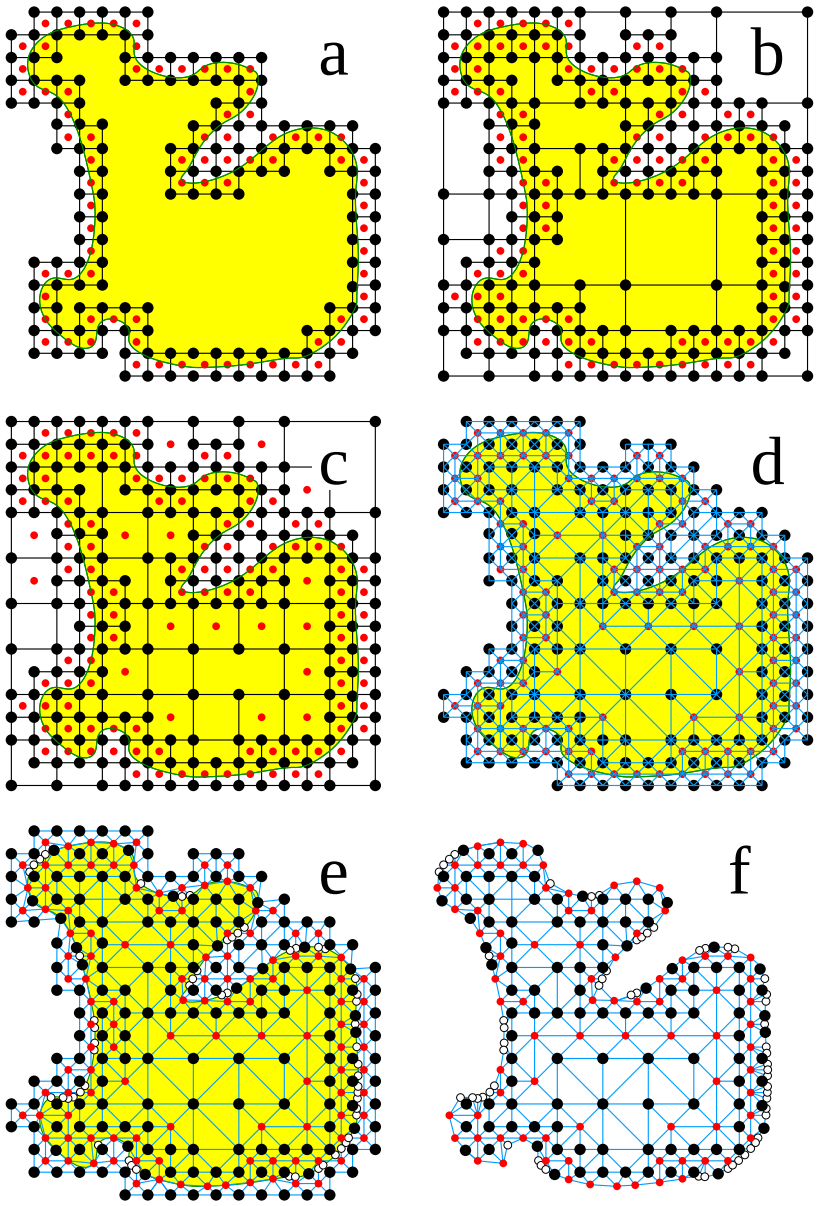

Figure 3.22: A two-dimensional illustration of the (three-dimensional) algorithm. (a) Cells intersecting the isosurface. (b) After enforcing the Continuation Condition and building an octree. (c) After enforcing the Weak Balance Condition. (d) The background grid. (e) After introducing cut points and warping the vertices. (f) The final mesh.

has all three vertices warped. This step is unnecessary to achieve the angle bounds not marked by asterisks.

3.5.4 Interior Grading

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 3.22(b).

We cannot bridge directly between two octants whose lengths differ by more than a factor of two while maintaining high element quality, so the next step is to impose the following *Weak Balance Condition*, illustrated in Figure 3.22(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 3.23 converts a balanced octree to a background grid, as illustrated in Figure 3.22(d).

3.5.5 Mesh Generation

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 3.1 and illustrated in Figure 3.22(e). (The Continuation Condition ensures that blue edges are never cut.) We use the stencils in Figure 3.3 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 3.22(f) for a two-dimensional analog.

3.6 Discussion

Surface Meshing

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 3.2. The upper bounds of less than 125° on plane angles

```
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 3.23: 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.

are noteworthy. They compete with the 120° guarantee of Chew's algorithm [24], at a fraction of the effort and running time.

For the α parameters listed as "safe" in Table 3.1, 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 $\mathbf{Z}^3 + (\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$, but not the endpoint on the cubical lattice $\mathbf{Z}^3$. 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.

Surface Grading

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 3.24 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°.

Conclusion

Nevertheless, we achieve 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

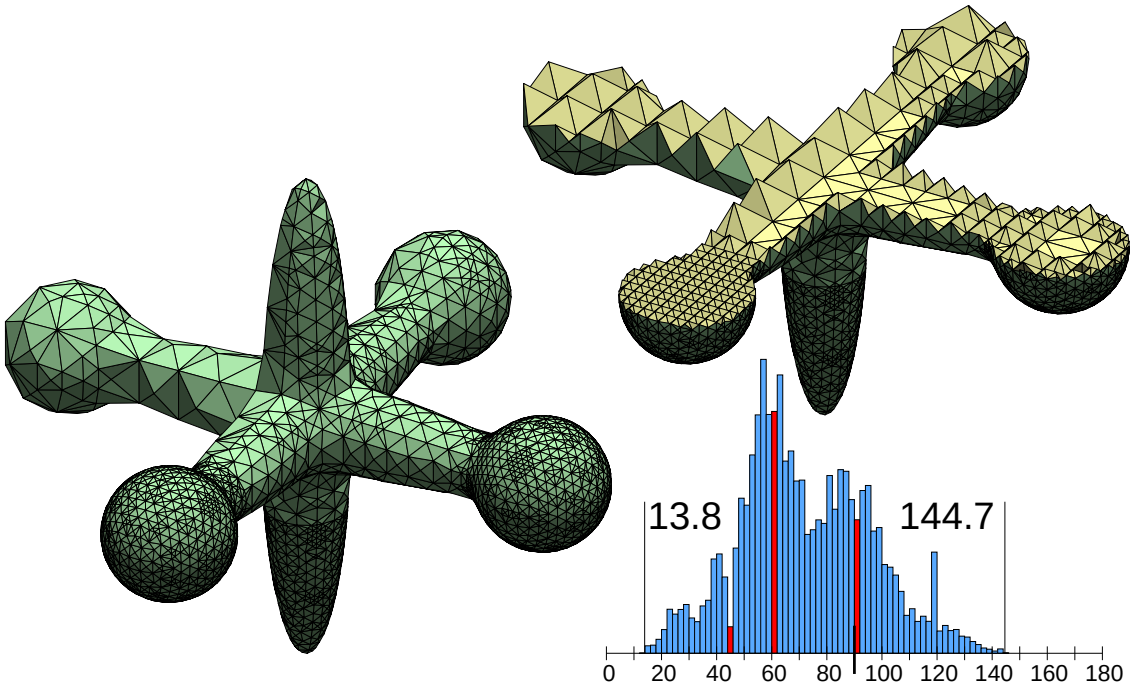

Figure 3.24: A 22,728-tetrahedron mesh, and a cutaway view thereof, generated in 2,859 milliseconds by a variant of isosurface stuffing that allows tetrahedra to grade both on and inside the surface. Dihedral angles vary from $13.8°$ to $144.7°$. Plane angles on the surface vary from $10.9°$ to $138.5°$. These results are typical, but much worse angles can occur.

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.

For isosurfaces with bounded curvature, it would be interesting to improve the bound on Hausdorff distance from $O(c)$ to $O(c^2)$ in Theorem 7, and to derive an explicit condition on c/d_M that guarantees isotopy in Theorem 8.

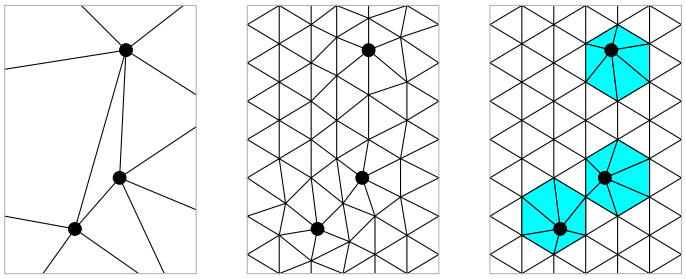

Figure 4.1: Two-dimensional equivalents of two possible strategies to get rid of slivers by refinement. (a) The Delaunay triangulation of the input contains a poorly shaped element. (b) Use a distorted fine grid. (c) Create small shells around input vertices and use a regular mesh outside.

Chapter 4

Delaunay Refinement Without Slivers

In this chapter, we present a Delaunay refinement algorithm that we call *lattice refinement* that can generate a mesh without slivers away from the boundary. New vertices are carefully chosen from lattices described in Section 1.3. The minimum dihedral angle guarantee of 30° is spectacularly good.

Inserting new vertices to get rid of slivers is a powerful tool which can be abused: a first solution is to lay down a very fine regular grid, and to slightly perturb it to match the input vertices. A second solution is to find a way to wrap each input vertex individually with a small shell of lattice vertices, and to fill the outside of the shells in some regular way. See Figure 4.1. The problem with these two solutions is that they require a lot of new vertices. The algorithm that we propose has the second solution as its worst case, but is likely to be much better in practice for two reasons.

1. A new vertex is inserted only in response to a bad-quality tetrahedron. If the Delaunay tetrahedralization of the input is already a good quality mesh, then it won't be changed by the algorithm. If the input has only a few bad tetrahedra, it is likely that a few vertex insertions will suffice to eliminate them because good angles can also be obtained by chance, especially if the quality requirements are relaxed to, say, a 15° minimum dihedral angle.

2. Although one can start with standard Delaunay refinement to bound the radius-edge ratios and then use our algorithm to eliminate the remaining slivers, it is a much better idea to use our algorithm to do both at the same time. In lattice refinement, vertices with arbitrary coordinates are the enemy; they should not be needlessly generated.

In Section 4.4, we present a way to incrementally generate a good quality mesh made out of lattice vertices only. The tetrahedra generated by our algorithm can grade from small to large and are guaranteed to have dihedral angles in the interval $[30^\circ, 135^\circ]$ and radius-edge ratios at most 1.119. Using this technique as part of a Delaunay refinement algorithm guarantees good-quality tetrahedra away from the boundary (and internal features of the domain), where the mesh is locally composed of lattice vertices only.

In Section 4.5, we add the ability to handle the simplest of internal features: input vertices with arbitrary coordinates. This makes our result directly comparable to previous results [25, 21]. This also slightly broadens the possible applications to, for example, the simulation of a number of point-like heat sources in three dimensions, where each source should be a vertex of the mesh. The dihedral angle guarantee stays the same, $[30^\circ, 135^\circ]$, and the bound on radius-edge ratio slightly worsens to 1.368. The algorithm can still grade from small to large tetrahedra, although the constant in the grading guarantee is weaker.

4.1 Delaunay Refinement

In this section, we briefly review Delaunay refinement, a popular mesh generation technique that is based on the Delaunay triangulation.

As we mentioned in Section 2.3, the Delaunay triangulation of a vertex set obeys the empty circumcircle property: for every triangle in the triangulation, its circumcircle contains no vertex in its interior. In three dimensions, the property is that every tetrahedron has an empty circumsphere. For more details and other properties, see the chapter by de Berg et al. [28].

A Delaunay refinement algorithm first computes the Delaunay triangulation of the input vertices (in two or three dimensions). If this gives a good quality mesh of the

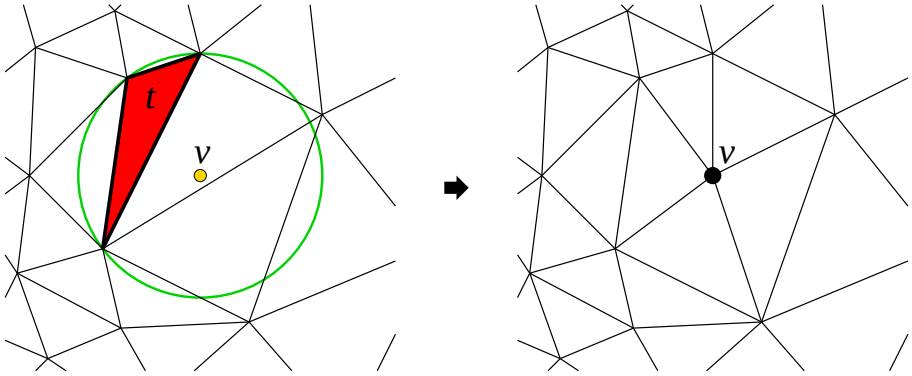

Figure 4.2: In Delaunay refinement, poorly-shaped triangles are eliminated by inserting their circumcenters (J. Shewchuk).

domain then we are done. Most likely, the triangulation contains a poor quality element, or doesn't even conform to the domain boundary, in which case a new vertex is inserted to correct the problem and the Delaunay triangulation is updated. The steps are repeated until a good quality mesh is obtained.

By poor quality we mean a radius-edge ratio greater than some bound B, in which case the circumcenter of the poor quality simplex is inserted to eliminate the simplex (see Figure 4.2).

The domain boundary makes this simple procedure more complicated. If the domain boundary is missing from the mesh, it must be recovered by inserting vertices. During refinement, new vertices cannot be inserted too close to the boundary or outside of the domain; instead a vertex is inserted somewhere on the boundary. Domains with small angles are another issue that is important to deal with. Using a constrained Delaunay triangulation [23, 79] can help. We defer to the literature for details on boundary handling [78, 71], as the issue is unimportant in this chapter.

It is important to guarantee that the refinement procedure terminates. A proof of termination can be given—it depends on the constant B and on details of the domain boundary handling.

4.1.1 Delaunay Triangulation Algorithms

In two dimensions, the Delaunay triangulation of a set of n points contains $\Theta(n)$ triangles and can be computed in optimal $O(n \log n)$ time by a divide-and-conquer algorithm [52, 45, 31], or by Fortune's sweep-line algorithm [40]. In three dimensions, a Delaunay tetrahedralization can have anywhere between $\Omega(n)$ and $O(n^2)$ tetrahedra, which means that the worst case running time must be at least $O(n^2)$.

Incremental Insertion

For the purpose of mesh generation, the ability to construct a Delaunay triangulation from scratch is not as useful as the ability to *update* a Delaunay triangulation when a vertex is inserted. Incremental insertion can be done with edge flips in two dimensions, as shown by Lawson [51], or by retriangulating a cavity by an algorithm of Bowyer [16] and Watson [85] that works in arbitrary dimensions.

Because the effect of inserting one vertex is usually a local operation, the running time of one insertion is usually $O(1)$, although in the worst case this could be $O(n)$, giving a $O(n^2)$ incremental Delaunay triangulation algorithm in two dimensions. Clarkson and Shor [27] randomize the order in which vertices are inserted. The randomization means that inserting a vertex now requires the ability to locate a point in the mesh based on coordinates. By using a smart point location algorithm, the expected running time of one insertion is $O(\log n)$, leading to an expected running time of $O(n \log n)$ to compute the full triangulation, and $O(n^{\lceil d/2 \rceil})$ in $d \geq 3$ dimensions. In mesh generation, we cannot randomize the order in which vertices are inserted so a randomized analysis is not possible. In practice, insertions take $O(1)$ time on average during refinement, and mesh generation takes $O(n)$ time where n is the size of the final mesh.

4.1.2 Data Structures

While a simple list of triangles might be sufficient to render a mesh, it is not a good format to manipulate a mesh. To allow fast local operations, the mesh must be represented in computer memory in a way that makes it easy to look up neighboring elements. In the straightforward implementation, the structure for a simplex contains pointers to the neighboring simplices, in addition to pointers to its vertices. This requires 6 pointers per triangle in two dimensions, and 8 pointers per tetrahedron in three dimensions.

Guibas and Stolfi [45] proposed a quad-edge representation in two dimensions. Some advantages are that it can represent planar subdivisions, not just triangulations, and one can access the dual subdivision without any work. Unfortunately, it is an expensive way to represent a triangulation, requiring 12 pointers per triangle.

More recently, Blandford et al. [12] proposed representing two and three dimensional triangulations in a special, compressed way that requires about 3 times less memory than the straightforward representation. Neighbors can still be found rapidly, and the method has the practical advantage that mesh operations can be performed by deleting and adding simplices without having to update any pointer.

4.2 Simplified Mesh Generation

Ideally one would like to be able to mesh any *piecewise linear complex* (PLC) with high-quality tetrahedra, even when the boundary is complicated. This chapter doesn't treat the boundary. Provably eliminating slivers is a very hard problem that has seen little progress, so even a partial solution is significant.

Here are a few concrete ways to interpret boundary-less mesh generation, the first two being mathematically precise.

Periodic space [21, 33]. The algorithm is used to mesh the periodic space $[0, 1)^3$, where space wraps around at the boundary like in the game *Asteroids*. Assuming that the space is initialized with enough vertices that inserting a new vertex won't create a tetrahedron that uses that new vertex twice, the periodic space behaves locally like $\mathbf{R}^3$.

Superset mesh. Given a domain $\Omega \subset \mathbf{R}^3$ with boundary, one can ask for a good quality mesh that does not necessarily respect the boundary but is a superset of Ω, sometimes called a *simulation envelope* [77]. This can be accomplished by considering only the quality of tetrahedra that intersect Ω and allowing the insertion of new vertices outside of Ω. When the algorithm terminates, delete the tetrahedra that do not intersect Ω. The result is a superset mesh of Ω with good quality tetrahedra only.

Good quality except at the boundary. The algorithm is used as part of an existing solution to mesh PLCs. If the lattice refinement algorithm is about to insert a vertex that is too close to the boundary or even outside the domain, abort the insertion and do what the PLC algorithm would do instead.

These are illustrated in Figure 4.3. In this chapter the results and proofs are written with the notation of the superset mesh problem.

4.2.1 Sizing Function and Local Feature Size

Definition 6 (Sizing Function) *The* sizing function *is a scalar field $s : \Omega \to (0, \infty]$ used to prescribe the approximate size of elements at each point of the domain.*

We will use the sizing function as follows: the final mesh should be such that a closed ball of radius $s(p)$ centered at p is non-empty, for all $p \in \Omega$. Because the mesh is Delaunay, this directly implies an upper bound of $s(p)$ for the circumradius of a tetrahedron, where p is its circumcenter.

Since the simplified setting does not allow input edges or faces, we use a definition of local feature size adapted to the case of an input vertex set with a sizing function. (See Ruppert [74] for the original definition.)

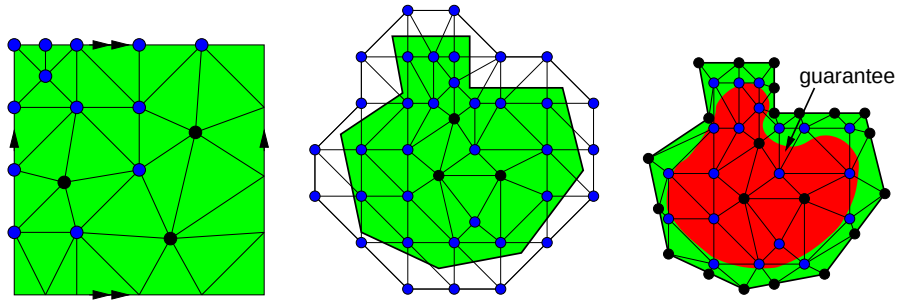

Figure 4.3: Three problems that can be solved by a mesh generation algorithm that doesn't handle the boundary.

Definition 7 (Local Feature Size) *Let p be any point in $\mathbf{R}^3$. The local feature size due to input vertices, $\mathrm{lfs_i}(p)$, is the distance between p and its second-closest input vertex. The local feature size due to the sizing function is $\mathrm{lfs_s}(p) = \inf\{s(a) + \mathrm{dist}(p,a) : a \in \Omega\}$. The combined local feature size is $\mathrm{lfs}(p) = \min(\mathrm{lfs_i}(p), \mathrm{lfs_s}(p))$.*

By considering the case $a = p$ in the definition of $\mathrm{lfs_s}(p)$ we see that $\mathrm{lfs_s}(p) \leq s(p)$ for $p \in \Omega$. Note that $\mathrm{lfs_s}(p)$ can be interpreted as a "1-Lipschitzation" of the sizing function $s(p)$.

Lemma 9 *For any two points p and q,*

1. $\mathrm{lfs_i}(q) \leq \mathrm{lfs_i}(p) + \mathrm{dist}(p,q)$,
2. $\mathrm{lfs_s}(q) \leq \mathrm{lfs_s}(p) + \mathrm{dist}(p,q)$,
3. $\mathrm{lfs}(q) \leq \mathrm{lfs}(p) + \mathrm{dist}(p,q)$;

i.e. the functions are 1-Lipschitz.

PROOF: The proof is adapted from Ruppert [74].

1. By the definition of $\mathrm{lfs_i}(p)$, there are two input vertices v and w at distance at most $\mathrm{lfs_i}(p)$ from p. By the triangle inequality, v and w are at distance at most $\mathrm{lfs_i}(p) + \mathrm{dist}(p,q)$ from q. Therefore $\mathrm{lfs_i}(q) \leq \mathrm{lfs_i}(p) + \mathrm{dist}(p,q)$.

2.

$$\begin{aligned}
\mathrm{lfs}_{\mathrm{s}}(q) &= \inf\{s(a) + \mathrm{dist}(q,a) : a \in \Omega\} \\
&\leq \inf\{s(a) + \mathrm{dist}(q,p) + \mathrm{dist}(p,a) : a \in \Omega\} \\
&= \inf\{s(a) + \mathrm{dist}(p,a) : a \in \Omega\} + \mathrm{dist}(p,q) \\
&= \mathrm{lfs}_{\mathrm{s}}(p) + \mathrm{dist}(p,q).
\end{aligned}$$

3. Follows by taking the minimum of both sides of inequalities (i) and (ii).

□

4.3 Uniform Tetrahedra

Creating a superset mesh composed of uniform tetrahedra is easy because we only have to consider an infinite tiling of space with tetrahedra and retain those that intersect the domain. Sommerville [81] asked which tetrahedra can fill space with congruent copies and found four such space-filling tetrahedra. More space-filling tetrahedra exist, but they do not meet face-to-face, which makes them unsuitable for a finite element mesh. See the survey of Senechal [75] on space-filling tetrahedra.

Combining tetrahedra with different shapes gives more flexibility. Field [37] proposes filling the interior of a domain with a tetrahedral mesh constructed from a complicated icosahedral assembly. Naylor [66] compares the suitability for numerical methods of five ways to fill space with tetrahedra, including Field's assembly. Based on five conditioning measures, he concludes that the best choice was simply given by one of Sommerville's space-filling tetrahedra. This assembly corresponds to the BCC grid of Section 3.1, and is the Delaunay tetrahedralization of the body-centered cubic lattice. It gives a mesh with dihedral angles of $60°$ and $90°$, and radius-edge ratios of $\sqrt{15}/6$ (≈ 0.645).

Eppstein, Sullivan and Üngör [35] show that it is possible to tile space with acute tetrahedra—tetrahedra with dihedral angles all strictly less than $90°$. They give several constructions, one even achieving dihedral angles between $60°$ and $74.21°$, and radius-edge ratios at most ≈ 0.711. This way of filling space is more complicated than the BCC grid, but its quality is comparable and possibly better, depending on the application.

4.4 Graded Tetrahedra

In this section, we show the lattices defined in Section 1.3 can be used in concert to generate tetrahedra that can grade from small to large.

An algorithm to generate graded meshes of guaranteed quality that honor a sizing function s (but not input vertices) can be devised easily: one can construct a balanced octree and take its Delaunay tetrahedralization. By analyzing the 2^6 possible types of cells in a balanced octree, we find that the dihedral angles are in the interval $[19.471°, 135°]$ and radius-edge ratios at most 1.347. The dihedral angle bounds can be improved to $[45°, 120°]$ by using templates of transition tetrahedra, as shown in Section 3.5. The radius-edge ratios are at most $\sqrt{5}/2$ (< 1.119).

The technique presented in this section is simple and generates tetrahedra with dihedral angles in the interval $[30°, 135°]$ and radius-edge ratios of at most $\sqrt{5}/2$. It is formulated as an incremental insertion method to make it easy to combine with standard Delaunay refinement.

Given any full-rank lattice $L \subset \mathbf{R}^3$, let $e(L)$ be the minimum distance between two distinct points in L. Let $r(L)$ be the radius of the largest empty open ball in $\mathbf{R}^3 \backslash L$.

Lemma 10 *Any ball of radius $r(L)$ contains at least one point of L in its interior, or at least 4 on its boundary.*

PROOF: Let B be an open ball of radius $r(L)$ with no point of L in its interior. By definition of $r(L)$, B must be a largest possible empty open ball. Since B is of maximum radius, there must be at least 4 points of L on its boundary to prevent the radius from being increased further. The result follows. □

Proposition 2

$$\begin{aligned} e(\mathrm{SC}_k) &= 2^k, & e(\mathrm{BCC}_k) &= 2^k\sqrt{3}/2, \\ r(\mathrm{SC}_k) &= 2^k\sqrt{3}/2, & r(\mathrm{BCC}_k) &= 2^k\sqrt{5}/4. \end{aligned}$$

PROOF: The first two are clear. For the last two, note that maximal open balls occur at Voronoi vertices of the lattices. For example, $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$ is a Voronoi vertex of SC_0 with a maximal empty ball of radius $\sqrt{3}/2$, and $(\frac{1}{2}, \frac{1}{4}, 0)$ is a Voronoi vertex of BCC_0 with a maximal empty ball of radius $\sqrt{5}/4$. Due to the symmetry of these lattices, all other maximal balls are of equal sizes. □

The method depends crucially on the following two lemmas, which assert that a tetrahedron with vertices in a lattice and a small circumradius either has good dihedral angles, or its circumsphere contains a lattice point (that the algorithm can insert to eliminate the tetrahedron).

Lemma 11 *Let t be a tetrahedron with vertices in SC_k and a circumradius of $r(\mathrm{BCC}_{k+1})$ or less. Then the dihedral angles of t are all at least $30°$ and at most $135°$.*

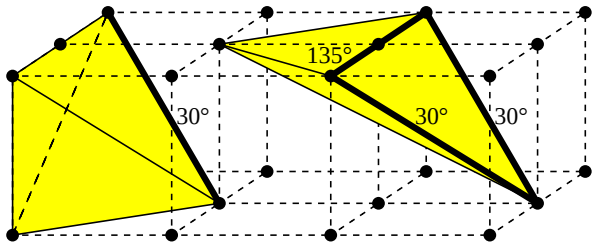

Figure 4.4: Two worst-case tetrahedra for Lemma 11. Each tetrahedron has vertices in SC_0 and a circumradius of $\sqrt{5}/2$ or less. They are the only such tetrahedra to achieve at least one of the angle bounds of Lemma 11.

PROOF: By rescaling, it suffices to show that a tetrahedron with vertices in SC_0, a dihedral angle of less than 30° or of more than 135°, and a circumradius of $\sqrt{5}/2$ or less doesn't exist. Because of the bound on circumradius, there is a finite number of possible tetrahedra to test. The verification was carried out by a computer. Figure 4.4 shows the tetrahedra that achieve the extreme angles. □

Lemma 12 *Let t be a tetrahedron with vertices in BCC_k and a circumradius of $r(\mathrm{SC}_k)$ or less. If t has a dihedral angle less than $\arccos(\sqrt{6}/3)$ ($\approx 35.264^\circ$) or larger than $\arccos(-\sqrt{3}/3)$ ($\approx 125.264^\circ$), then there exists a point of SC_k inside the circumsphere of t.*

PROOF: By rescaling, it suffices to show that a tetrahedron with vertices in BCC_0, a dihedral angle of less than $\arccos(\sqrt{6}/3)$ or of more than $\arccos(-\sqrt{3}/3)$, and a circumradius of $\sqrt{3}/2$ or less must always contain a point of SC_0 inside its circumsphere. Because of the bound on circumradius, there is a finite number of possible tetrahedra to test. The verification was carried out by a computer. □

In fact, there is a finite number of tetrahedra with vertices in SC_k or BCC_k that do not contain a vertex from a coarser lattice inside their circumspheres. They are listed in Table 4.1, and the first two tetrahedra from the table are illustrated in Figure 4.4. By negation, every lattice tetrahedron *not* in the table contains a vertex from a coarser lattice inside its circumsphere, which we can insert to refine the mesh and eliminate the tetrahedron. Thus, by refinement, it is possible to construct a graded mesh composed of tetrahedra appearing in Table 4.1 only. (The proof of grading appears in Section 4.4.2.)

minimum dihedral angle	maximum dihedral angle	radius-edge ratio	can refine from same lattice	coordinates of an example tetrahedron (SC_0 or BCC_1)
SC_k				
30.0000	120.0000	1.1180	yes	(0,0,0), (0,0,1), (0,2,0), (1,1,1)
30.0000	135.0000	0.7906	yes	(0,0,1), (0,1,2), (0,2,1), (1,1,0)
35.2644	125.2644	0.8660	no	(0,0,0), (0,0,1), (0,1,0), (1,1,1)
35.2644	90.0000	1.1180	yes	(0,0,0), (0,0,1), (0,2,1), (1,1,1)
45.0000	90.0000	0.8660	no	(0,0,0), (0,0,1), (0,1,1), (1,1,1)
45.0000	90.0000	1.1180	yes	(0,0,0), (0,0,2), (0,1,1), (1,1,1)
48.1897	90.0000	1.1180	yes	(0,0,0), (0,1,1), (1,0,2), (1,1,1)
54.7356	109.4712	0.7071	yes	(0,0,1), (0,1,0), (0,1,2), (1,1,1)
54.7356	90.0000	0.8660	no	(0,0,1), (0,1,0), (0,1,1), (1,1,1)
60.0000	90.0000	0.6455	yes	(0,0,1), (0,2,1), (1,1,0), (1,1,2)
70.5288	70.5288	0.6124	no	(0,0,1), (0,1,0), (1,0,0), (1,1,1)
BCC_k				
35.2644	125.2644	0.8660	yes	(0,0,0), (0,0,2), (0,2,0), (2,2,2)
45.0000	120.0000	0.8660	yes	(0,0,2), (0,2,2), (1,1,1), (2,2,2)
45.0000	90.0000	0.8660	yes	(0,0,0), (0,0,2), (0,2,2), (2,2,2)
54.7356	90.0000	0.8660	yes	(0,0,2), (0,2,0), (0,2,2), (2,2,2)
60.0000	90.0000	0.6455	no	(0,2,2), (1,1,1), (1,1,3), (2,2,2)
70.5288	70.5288	0.6124	yes	(0,0,2), (0,2,0), (2,0,0), (2,2,2)

Table 4.1: Tetrahedra that do not contain a vertex of a coarser lattice inside their circumsphere. The tetrahedra under the heading "SC_k" relate to Lemma 11, and the tetrahedra under the heading "BCC_k" relate to Lemma 12. The fourth column indicates if the tetrahedron's circumsphere contains a vertex of the *same* lattice, implying that these tetrahedra can be refined, but doing so may lead to a uniform mesh (not graded).

4.4.1 Algorithm

Algorithm 1 generates a mesh by incrementally inserting lattice vertices while maintaining a Delaunay tetrahedralization of these lattice vertices. It uses Algorithm 2 to refine empty balls that are too large. When a vertex is created it is assigned a *label*, either SC_k or BCC_k for some k, and is tagged with a *type number*. The labels are needed in the implementation, but the type numbers are not (they are only used in the analysis). Each vertex belongs to the lattice of its label, but the label is not necessarily the coarsest lattice

Algorithm 1: Simple graded meshing
Input:

- A domain $\Omega \subset \mathbf{R}^3$.
- A sizing function $s : \Omega \to (0, \infty]$.

Repeat the enforcement steps below in any order until none apply. Maintain a Delaunay tetrahedralization as new vertices are inserted. When finished, remove all tetrahedra whose interiors do not intersect Ω.

Size enforcement: If there exists a point $p \in \Omega$ such that a closed ball B of radius $s(p)$ centered at p is empty, then **call Algorithm 2** to refine the empty ball B with a safety factor $f = 0$.

Quality enforcement: If the Delaunay tetrahedralization contains a tetrahedron t with circumradius r and shortest edge length e whose interior intersects Ω and which has a dihedral angle smaller than $30°$ or larger than $135°$, or a radius-edge ratio r/e larger than $\beta = \sqrt{5}/2$, do:

Among the 4 vertices of t, find the vertex v with the finest lattice label, according to the nesting of lattices.

Case 1: The label of v is SC_k.

By assumption, $r > e\beta \geq e(\mathrm{SC}_k)\beta = r(\mathrm{BCC}_{k+1})$, or t has a dihedral angle that is less than $30°$ or greater than $135°$. By Lemma 11, $r > r(\mathrm{BCC}_{k+1})$. Let B be a closed ball of radius $r(\mathrm{BCC}_{k+1})$ tangent at v inside the circumsphere of t (see Figure 4.5, left). By Lemma 10 there exists a point w of BCC_{k+1} in $B\backslash\{v\}$. **Insert** w **with label** BCC_{k+1} **and type 1**.

Case 2: The label of v is BCC_k.

By assumption, $r > e\beta \geq e(\mathrm{BCC}_k)\beta > r(\mathrm{SC}_k)$, or t has a dihedral angle that is less than $30°$ or greater than $135°$. If $r > r(\mathrm{SC}_k)$, then let B be a closed ball of radius $r(\mathrm{SC}_k)$ tangent at v inside the circumsphere of t (see Figure 4.5, right). By Lemma 10 there exists a point w of SC_k in $B\backslash\{v\}$. Else by Lemma 12 there exists a point w of SC_k inside the circumsphere of t. **Insert** w **with label** SC_k **and type 2**.

that contains the vertex.

The algorithm inserts vertices for one of two reasons: the sizing function, or a bad-

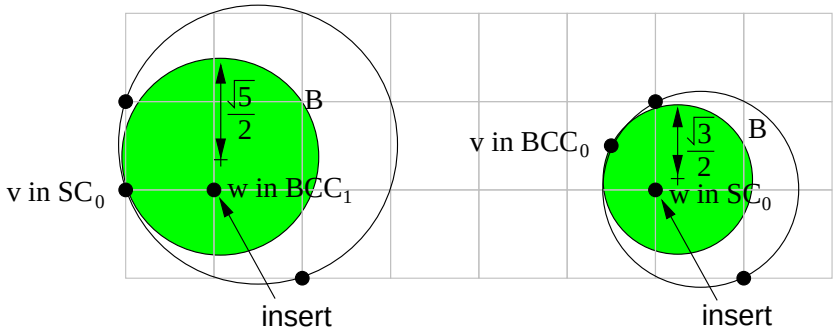

Figure 4.5: Two-dimensional examples of the geometric constructions in cases 1 and 2 of the quality-enforcement step of Algorithm 1. The new vertex w is chosen from a lattice that is one level coarser than the label of v. Furthermore, w is chosen from an inner ball B tangent at v in order to be close to v. This strategy generates a mesh that grades quickly.

Algorithm 2: Refine empty ball
Input:

- An empty ball B (open or closed) with center c and radius $r > 0$.
- A safety factor $f \geq 0$ (a nonzero value will be used by Algorithms 3 and 6).

One of the following two cases holds because these two families of intervals cover every possible $r > 0$.

Case 1: $r \in (2^j(\sqrt{3}/2 + f), 2^j(\sqrt{5}/2 + f)]$ for some $j \in \mathbf{Z}$.

By Lemma 10 there exists a point w of SC_j at distance at most $2^j\sqrt{3}/2$ from c. Insert w with label SC_j into the Delaunay tetrahedralization.

Case 2: $r \in (2^j(\sqrt{5}/2 + f), 2^j(\sqrt{3} + 2f)]$ for some $j \in \mathbf{Z}$.

By Lemma 10 there exists a point w of BCC_{j+1} at distance at most $2^j\sqrt{5}/2$ from c. Insert w with label BCC_{j+1} into the Delaunay tetrahedralization.

Because B is empty, the new vertex w is at a distance greater than $2^j f$ from any previous vertex, where the label of w is SC_j or BCC_{j+1}.

quality tetrahedron. In the latter case, the inserted vertex always comes from a lattice that is strictly coarser than the finest lattice label of the tetrahedron vertices. This allows a proof of good grading. The refinement strategy is closer to Üngör's off-centers [84] than

to the circumcenter method.

Because the algorithm is building a superset mesh, any empty half-space that is tangent to a vertex, an edge, or a triangle of the tetrahedralization can be considered a degenerate circumsphere with missing vertices at infinity. Vertices at infinity are considered infinitely coarse in the nesting of lattices. Obviously, the tangent vertices can only be on the convex hull of the tetrahedralization. If such a half-space intersects Ω, then the corresponding fictive tetrahedron with 1, 2 or 3 vertices from the mesh and the half-space as its circumsphere qualifies for quality enforcement in Algorithm 1. During that step, the vertex v with the finest lattice label is from the mesh. The circumradius r is considered to be infinite. The circumcenter does not exist but it is not used. B is a closed ball tangent at v inside the half-space. The other steps are unaffected.

As written, the algorithm starts with no vertices at all, so it must first perform a size enforcement step. When there is at least one vertex, the empty half-space rule can apply and lead to quality enforcement steps.

4.4.2 Analysis

A mesh generation algorithm has *good grading* if the sizes of the elements can vary from small to large over a short distance. Since the work of Ruppert [74], a grading guarantee is usually a proof of a linear relationship between the nearest neighbor distance of a vertex v of the final mesh and its local feature size $\mathrm{lfs}(v)$.

As a first step we show that there exist positive constants a and b such that

$$\mathrm{lfs}(v) \leq \begin{cases} 2^k a & \text{if } v \text{ has label } \mathrm{SC}_k, \\ 2^k b & \text{if } v \text{ has label } \mathrm{BCC}_k. \end{cases} \tag{4.1}$$

We show by induction that these bounds are maintained by the algorithm. The constants a and b are derived at the end of this section. Once we have these bounds, the following theorem shows that we obtain good grading.

Theorem 13 *If a mesh consists of only* SC *or* BCC *lattice vertices, and if there exist positive constants a and b such that (4.1) holds, then for any vertex v of the mesh, the distance to its nearest neighbor is at least*

$$\min(\tfrac{1}{1+a}, \tfrac{1}{1+2b/\sqrt{3}})\mathrm{lfs}(v).$$

PROOF: (Adapted from Ruppert [74]). Let v be any vertex of the mesh. Let w be its nearest neighbor.

If the label of v is as fine or finer than the label of w we argue as follows.

- In case v has label SC_k, $\mathrm{dist}(v,w) \geq e(\mathrm{SC}_k) = 2^k$ and $\mathrm{lfs}(v) \leq 2^k a$, so
$$\mathrm{dist}(v,w) \geq \tfrac{1}{a}\mathrm{lfs}(v).$$
- In case v has label BCC_k, $\mathrm{dist}(v,w) \geq e(\mathrm{BCC}_k) = 2^k\sqrt{3}/2$ and $\mathrm{lfs}(v) \leq 2^k b$, so
$$\mathrm{dist}(v,w) \geq \tfrac{\sqrt{3}}{2b}\mathrm{lfs}(v).$$

In either case,
$$\mathrm{dist}(v,w) \geq \min(\tfrac{1}{a}, \tfrac{\sqrt{3}}{2b})\mathrm{lfs}(v).$$

Else (w is finer than v), we use Lemma 9 and apply the bound above to w.

$$\begin{aligned}\mathrm{lfs}(v) &\leq \mathrm{lfs}(w) + \mathrm{dist}(v,w)\\ &\leq \mathrm{dist}(v,w)/\min(\tfrac{1}{a}, \tfrac{\sqrt{3}}{2b}) + \mathrm{dist}(v,w)\\ &= \max(1+a, 1+\tfrac{2b}{\sqrt{3}})\mathrm{dist}(v,w).\end{aligned}$$

So $\mathrm{dist}(v,w) \geq \min(\frac{1}{1+a}, \frac{1}{1+2b/\sqrt{3}})\mathrm{lfs}(v)$. □

We perform a separate analysis for each type of inserted vertex w in Algorithm 1.

Insertion due to size

In the size enforcement step of Algorithm 1, $p \in \Omega$ is a point such that a closed ball B of radius $s(p)$ centered at p is empty. Let w be the new vertex. By Lemma 9,

$$\mathrm{lfs}_{\mathrm{s}}(w) \leq \mathrm{lfs}_{\mathrm{s}}(p) + \mathrm{dist}(p,w) \leq s(p) + \mathrm{dist}(p,w).$$

- If w was given label SC_j by Algorithm 2, then
$$s(p) + \mathrm{dist}(p,w) \leq 2^j(\sqrt{5}/2 + f) + 2^j\sqrt{3}/2$$
(here $\underline{f = 0}$, but a different value will be used in Section 4.5.2). We require that
$$\underline{\sqrt{5}/2 + f + \sqrt{3}/2 \leq a}$$
so that $\mathrm{lfs}_{\mathrm{s}}(w) \leq 2^j a$ and the insertion preserves (4.1).
- Else (w has label BCC_{j+1}),
$$s(p) + \mathrm{dist}(p,w) \leq 2^j(\sqrt{3} + 2f) + 2^j\sqrt{5}/2.$$
We require that
$$\underline{\sqrt{3} + 2f + \sqrt{5}/2 \leq 2b}$$
so that $\mathrm{lfs}_{\mathrm{s}}(w) \leq 2^{j+1} b$ and the insertion preserves (4.1).

Type 1 insertion

The new vertex w has label BCC_{k+1} at a distance at most $2r(\mathrm{BCC}_{k+1}) = 2^k\sqrt{5}$ from v with label SC_k. So

$$\mathrm{lfs}(w) \leq \mathrm{lfs}(v) + \mathrm{dist}(v, w) \leq 2^k a + 2^k\sqrt{5}.$$

We require that

$$\underline{a + \sqrt{5} \leq 2b},$$

so that by induction $\mathrm{lfs}(w) \leq 2^{k+1}b$.

Type 2 insertion

The new vertex w has label SC_k at a distance at most $2r(\mathrm{SC}_k) = 2^k\sqrt{3}$ from v with label BCC_k. So

$$\mathrm{lfs}(w) \leq \mathrm{lfs}(v) + \mathrm{dist}(v, w) \leq 2^k b + 2^k\sqrt{3}.$$

We require that

$$\underline{b + \sqrt{3} \leq a},$$

so that by induction $\mathrm{lfs}(w) \leq 2^k a$.

Guarantee

The requirements (underlined in the text) can be satisfied by setting

$$\begin{aligned} a &= 2\sqrt{3} + \sqrt{5}, \text{ and} \\ b &= \sqrt{3} + \sqrt{5}. \end{aligned}$$

By applying Theorem 13 we deduce that during the course of the algorithm, the distance between any vertex v and its nearest neighbor is at least

$$\mathrm{lfs}(v)/6.701.$$

Assuming there is a positive lower bound on the sizing function s, this result gives a positive lower bound on the distance between any pair of vertices. This in turn implies termination of the algorithm, because Ω has finite volume. (A slightly bigger volume must be considered in the case of a superset mesh).

4.5 Adding Vertex Constraints

In this section we extend lattice refinement to allow input vertices with arbitrary coordinates in Ω. The algorithm inserts lattice vertices, but never "too close" to an input vertex. Each lattice point has an associated *forbidden region* around it. No lattice point is ever inserted that has an input vertex in its forbidden region.

Definition 8 (Forbidden Region) *Let $p \in \mathbf{R}^3$ and $k \in \mathbf{Z}$. The forbidden region $R(p,k)$ is an axis-aligned cube with side $2^k(2+\sqrt{6})/2$ centered at p. Mathematically,*

$$R(p,k) = \{q : \|q-p\|_\infty \leq 2^k(2+\sqrt{6})/4\}.$$

Also define

$$\rho = \sqrt{3}(2+\sqrt{6})/4 \tag{4.2}$$

to be the safety radius constant, which is the distance from p to the furthest point of $R(p,0)$.

The lattice points that the algorithm inserts are called *refinement vertices* to distinguish them from input vertices. The algorithm maintains the invariant that if a refinement vertex v has label SC_k or BCC_{k+1}, then $R(v,k)$ contains no input vertex.

Theorem 14 *For any $p \in \mathbf{R}^3$ and any $k \in \mathbf{Z}$, let $S = \{p\} \cup (\mathrm{SC}_k \backslash R(p,k))$. Then every Delaunay tetrahedron in S has dihedral angles in the interval $[30^\circ, 127.903^\circ]$, and a radius-edge ratio of at most 1.368.*

PROOF: By rescaling, it suffices to consider the case $k=0$. Because the forbidden region is an axis-aligned cube with half-side $\sigma = (2+\sqrt{6})/4$ (≈ 1.112), if the coordinate of p along some axis falls in the set $(\sigma, 3-\sigma) + \mathbf{Z}$, then two points of SC_0 will be covered by the forbidden region $R(p,0)$ along that axis. If the coordinate falls in the complement $[3-\sigma, 1+\sigma] + \mathbf{Z}$, then three points of SC_0 will be covered along that axis. The total number of points of SC_0 that are removed by the set difference with $R(p,0)$ is therefore 8, 12, 18 or 27, depending on the three coordinates of p.

By invoking translational symmetry, we can assume that the coordinate of p along some axis falls in the interval $(\sigma, 1+\sigma]$. We consider two separate cases: either the coordinate falls in $(\sigma, 3-\sigma)$, or it falls in $[3-\sigma, 1+\sigma]$. In the first case, the interval is mirror symmetric about $(\sigma + (3-\sigma))/2 = \frac{3}{2}$ and the integers covered by the forbidden region are 1 and 2, also symmetric about $\frac{3}{2}$. So without loss of generality we can assume that the coordinate of p falls on the left half of the interval, i.e. $I = (\sigma, \frac{3}{2}]$. The second case is similar: the symmetry axis is $((3-\sigma)+(1+\sigma))/2 = 2$, the covered integers are 1,2 and 3, so without loss of generality we assume that the coordinate of p falls in the left

case	min dihedral	max dihedral	max ratio	achieved at $p =$
1	$\approx 31.962°$	$\approx 125.264°$(*)	≈ 1.198	(σ, σ, σ)
2	$\approx 30.129°$	$\approx 125.264°$(*)	≈ 1.267	$(\sigma, \sigma, 3-\sigma)$
3	$30°$	$< 127.903°$	≈ 1.329	$(\sigma, 3-\sigma, 3-\sigma)$
4	$\approx 34.785°$	$\approx 125.264°$(*)	< 1.368	$(3-\sigma, 3-\sigma, 3-\sigma)$

Table 4.2: Four cases in the proof of Theorem 14. In all cases the dihedral angles are in the interval $[30°, 127.903°)$ and the radius-edge ratios are less than 1.368. The bounds marked (*) are intrinsic to the simple cubic lattice and are achieved independently of the position of p. (The Delaunay tetrahedralization of a simple cube may contain a dihedral angle of $\arccos(-\sqrt{3}/3) \approx 125.264°$.)

half $J = [3-\sigma, 2]$. There are further symmetries because the three axes can be permuted, e.g. the case $I \times J \times I$ is equivalent to $I \times I \times J$. The possible cases for p are then reduced to these four:

1. $p \in I^3$ and $S = \{p\} \cup (\mathrm{SC}_0 \backslash \{1,2\}^3)$
2. $p \in I^2 \times J$ and $S = \{p\} \cup (\mathrm{SC}_0 \backslash \{1,2\}^2 \times \{1,2,3\})$
3. $p \in I \times J^2$ and $S = \{p\} \cup (\mathrm{SC}_0 \backslash \{1,2\} \times \{1,2,3\}^2)$
4. $p \in J^3$ and $S = \{p\} \cup (\mathrm{SC}_0 \backslash \{1,2,3\}^3)$

The hole in SC_0 created by the set difference with $R(p, 0)$ is a box with side either 3 or 4 in each axis direction (see Figure 4.6).

In each case the lattice vertices forming the box around p can be naturally classified as corner, edge or face vertices. We claim that in a Delaunay tetrahedralization of S, p always connects exactly to the convex hull of the box face vertices only. This is because the box corner and box edge vertices are isolated from p by Delaunay tetrahedra. The tetrahedra whose circumspheres come closest to containing p occur in case 1. An example is a tetrahedron with circumsphere center $(\frac{1}{2}, \frac{1}{2}, \frac{3}{2})$ and radius $\sqrt{3}/2$. The circumsphere goes through the point $(\sigma, \sigma, \frac{3}{2})$ which is avoided by $p \in (\sigma, \frac{3}{2}]^3$. This is why we chose this value of σ.

The angle bounds are more difficult to prove formally than in Lemmas 11 and 12 because the coordinates of point p can vary continuously in a range. Nonetheless the dihedral angles are smooth functions of the position of p (there is no change in connectivity as p moves) and the extrema are easily found. See Table 4.2 for a summary of the bounds in each case. Figure 4.7 shows tetrahedra that achieve the lower bounds. □

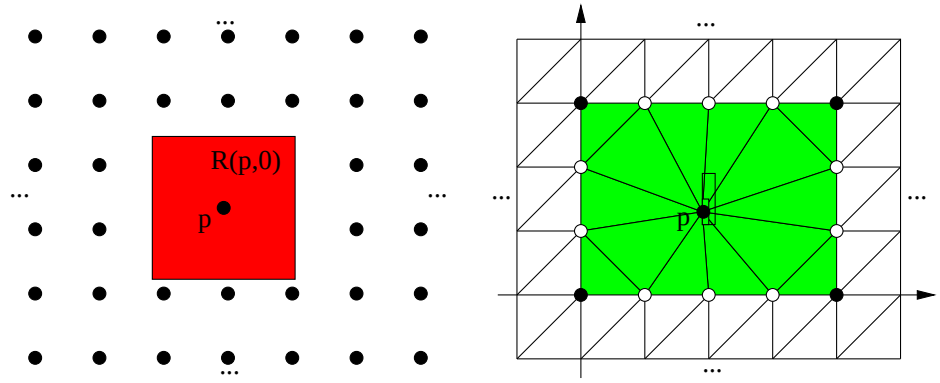

Figure 4.6: A two-dimensional equivalent of the set $S = \{p\} \cup (\mathrm{SC}_0 \backslash R(p,0))$ and its Delaunay tetrahedralization in the proof of Theorem 14. In this example, p lies in the rectangular frame $[3-\sigma, 1+\sigma] \times (\sigma, 3-\sigma)$ shown at the center of the right-hand figure, so the hole in $\mathrm{SC}_0 \backslash R(p,0)$ is a 4×3 rectangle. By mirror symmetry, we assume that p lies in the smaller sub-rectangle $[3-\sigma, 2] \times (\sigma, \frac{3}{2}] = J \times I$. Hollow circles are the box face vertices.

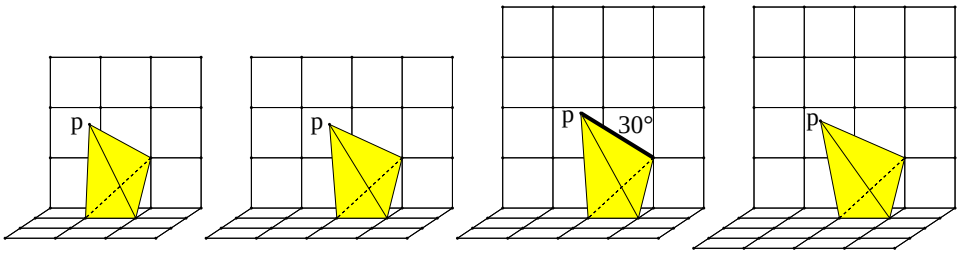

Figure 4.7: Tetrahedra achieving a minimum dihedral angle in each case of Theorem 14. Only two sides of the box around the input vertex p are shown. The minimum dihedral angles and coordinates of p appear in Table 4.2. (The coordinate system is different for this figure.)

Corollary 15 *Let $p \in \mathbf{R}^3$ and $k \in \mathbf{Z}$. Let t be a tetrahedron with vertices in the set $S = \{p\} \cup (\mathrm{SC}_k \backslash R(p,k))$. Suppose that p is not inside the circumsphere of t. If t has a dihedral angle smaller than $30°$ or larger than $127.903°$, or a radius-edge ratio larger than 1.368, then there exists a point q of $\mathrm{SC}_k \backslash R(p,k)$ inside the circumsphere of t.*

PROOF: By Theorem 14, t cannot be Delaunay in S. Therefore there exists a point q in S inside the circumsphere of t. Since we assumed that p is not inside the circumsphere of t, $q \neq p$. So $q \in \mathrm{SC}_k \backslash R(p,k)$. □

Algorithm 3: Meshing with input vertices
Input:

- A domain $\Omega \subset \mathbf{R}^3$.
- A finite number of input vertices with arbitrary coordinates in Ω.
- A sizing function $s : \Omega \to (0, \infty]$.

Compute a Delaunay tetrahedralization of the input vertices. Repeat the enforcement steps below in any order until none apply. Maintain a Delaunay tetrahedralization as new vertices are inserted. When finished, remove all tetrahedra whose interiors do not intersect Ω.

Size enforcement: If there exists a point $p \in \Omega$ such that a closed ball B of radius $s(p)$ centered at p is empty, then **call Algorithm 2** to refine the empty ball B with a safety factor $f = \sqrt{3}(2 + \sqrt{6})/4$ (ρ of Definition 8). Assign **type 0** to the new vertex.

Quality enforcement: If the Delaunay tetrahedralization contains a tetrahedron t whose interior intersects Ω and which has a dihedral angle smaller than 30° or larger than 135°, or a radius-edge ratio larger than $\beta = 1.368$, then **call Algorithm 4** if t has at least 1 refinement vertex, or **Algorithm 6** if t has at least 2 input vertices. (If both cases apply, then choose either algorithm.)

4.5.1 Algorithm

Algorithm 3 is an extension of Algorithm 1 of Section 4.4.1. It uses Algorithms 2, 4, 5 and 6 to handle specific geometric cases. At places these algorithms make queries of the form *"Is there an input vertex in the region $R(w, k)$?"* This kind of query can be answered efficiently by using the Delaunay tetrahedralization as a search structure.

In Algorithm 4, an attempt is made to insert a new vertex from a strictly coarser lattice, to obtain good mesh grading. The attempt is aborted if the candidate for insertion is "too close" to an input vertex.

In Algorithm 5, there is an input vertex nearby and the algorithm has permission to insert a new vertex from a lattice that is just as fine as the finest vertex of the bad-quality tetrahedron, or even one level finer (inserting a vertex from SC_k when the finest tetrahedron vertex label is BCC_{k+1}). Corollary 15 allows an analysis of this case.

In Algorithm 6, there are two input vertices nearby. Corollary 15 cannot be used because it can only deal with one nearby input vertex at a time. We have no other choice

Algorithm 4: Refine tetrahedron (Case 1)
Input:

- A tetrahedron t of bad quality (dihedral angle smaller than 30° or larger than 135°, or a radius-edge ratio larger than $\beta = 1.368$) with circumcenter c, circumradius r, and at least 1 refinement vertex.

Let v be the refinement vertex of t with the finest label, according to the nesting of lattices.

1.1 If the label of v is SC_k for some k.

1.1.1 If $r > r(\mathrm{BCC}_{k+1})$, then let B be a closed ball of radius $r(\mathrm{BCC}_{k+1})$ tangent at v inside the circumsphere of t. By Lemma 10 there exists a point w of BCC_{k+1} in $B\backslash\{v\}$. Note that $\mathrm{dist}(w, v) \leq 2r(\mathrm{BCC}_{k+1})$. w is a candidate for insertion.

1.1.2 Else if t has at least one input vertex u_1, then **call Algorithm 5**. Return. Note that $\mathrm{dist}(u_1, v) \leq 2r(\mathrm{BCC}_{k+1})$ and $\mathrm{dist}(u_1, c) = r$.

1.1.3 Else (t has 4 refinement vertices), the radius-edge ratio of t is at most $r(\mathrm{BCC}_{k+1})/e(\mathrm{SC}_k) = \sqrt{5}/2 < \beta$, so t must have a dihedral angle smaller than 30° or larger than 135°. This is impossible by Lemma 11, so this subcase never happens.

If the forbidden region $R(w, k)$ of the candidate w doesn't contain any input vertex, then **insert** w **with label** BCC_{k+1} **and type 1.1**. Else, let u_1 be an input vertex in the forbidden region. Note that $\mathrm{dist}(u_1, v) \leq 2r(\mathrm{BCC}_{k+1}) + 2^k\rho$ and $\mathrm{dist}(u_1, c) \leq r + 2^k\rho < (1 + 2\rho/\sqrt{5})r$ because $r > r(\mathrm{BCC}_{k+1}) = 2^k\sqrt{5}/2$. **Call Algorithm 5**.

1.2 Else (the label of v is BCC_{k+1} for some k).

1.2.1 If $r > r(\mathrm{SC}_{k+1})$, then let B be a closed ball of radius $r(\mathrm{SC}_{k+1})$ tangent at v inside the circumsphere of t. By Lemma 10 there exists a point w of SC_{k+1} in $B\backslash\{v\}$. w is a candidate for insertion.

1.2.2 Else if t has at least one input vertex u_1, then **call Algorithm 5**. Return. Note that $\mathrm{dist}(u_1, v) \leq 2r(\mathrm{SC}_{k+1})$ and $\mathrm{dist}(u_1, c) = r$.

1.2.3 Else (t has 4 refinement vertices), the radius-edge ratio of t is at most $r(\mathrm{SC}_{k+1})/e(\mathrm{BCC}_{k+1}) = 1 < \beta$, so t must have a dihedral angle smaller than 30° or larger than 135°. By Lemma 12 there exists a point w of SC_{k+1} inside the circumsphere of t. w is a candidate for insertion.

If the forbidden region $R(w, k+1)$ of the candidate w doesn't contain any input vertex, then **insert** w **with label** SC_{k+1} **and type 1.2**. Else, let u_1 be an input point in the forbidden region. Note that $\mathrm{dist}(u_1, v) \leq 2r(\mathrm{SC}_{k+1}) + 2^{k+1}\rho$ and $\mathrm{dist}(u_1, c) \leq r + 2^{k+1}\rho \leq (1 + 4\rho/\sqrt{3})r$ because $r \geq e(\mathrm{BCC}_{k+1})/2 = 2^k\sqrt{3}/2$. **Call Algorithm 5**.

Algorithm 5: Refine tetrahedron (Case 2)
Input:

- A tetrahedron t of bad quality (dihedral angle smaller than 30° or larger than 135°, or a radius-edge ratio larger than $\beta = 1.368$) with circumcenter c, circumradius r, and at least 1 refinement vertex. Let v be the finest refinement vertex of t, with label SC_k or BCC_{k+1}.
- An input vertex u_1 (possibly a vertex of t) such that $\mathrm{dist}(u_1, v) \leq 2^k(2\sqrt{3} + 2\rho)$ and $\mathrm{dist}(u_1, c) \leq (1 + 4\rho/\sqrt{3})r$.

2.1 If $r > r(\mathrm{SC}_k) + 2^k\rho/2$, then let B be a closed ball of radius $r(\mathrm{SC}_k) + 2^k\rho/2$ tangent at v inside the circumsphere of t. Let B' be a closed ball of radius $r(\mathrm{SC}_k)$ lying in B as far as possible from u_1. By Lemma 10 there exists a point w of SC_k in $B'\backslash\{v\}$. By construction, the forbidden region $R(w, k)$ doesn't contain u_1, and $\mathrm{dist}(w, v) \leq 2r(\mathrm{SC}_k) + 2^k\rho$. w is a candidate for insertion.

2.2 Else if t has no input vertex except possibly u_1, by Corollary 15 there exists a point w of SC_k inside the circumsphere of t such that $w \notin R(u_1, k)$. This implies that $u_1 \notin R(w, k)$. Note that $\mathrm{dist}(w, v) < 2r(\mathrm{SC}_k) + 2^k\rho$. w is a candidate for insertion.

2.3 Else (t has an input vertex $u_2 \neq u_1$), note that $\mathrm{dist}(u_2, c) = r$. **Call Algorithm 6**. Return.

If the forbidden region of $R(w, k)$ doesn't contain *any* input vertex, then **insert** w **with label** SC_k **and type 2**. Else let u_2 be an input vertex in the forbidden region. Note that $\mathrm{dist}(u_2, c) \leq r + 2^k\rho \leq (1 + 2\rho)r$ because $r \geq e(\mathrm{SC}_k)/2 = 2^k/2$. **Call Algorithm 6**.

Algorithm 6: Refine tetrahedron (case 3)
Input:

- A tetrahedron t with circumcenter c and circumradius r.
- Two input vertices u_1 and u_2 that are vertices of t or are at distance at most $(1 + 4\rho/\sqrt{3})r$ from c.

Call Algorithm 2 to refine the empty circumsphere of t with a safety factor ρ. Assign **type 3** to the new vertex.

than to refine the mesh with a lattice vertex and try to make the local lattice resolution smaller than the distance between the two input vertices.

4.5.2 Analysis

The analysis is structured in the same way as in Section 4.4.2. We find positive constants a and b such that (4.1) holds.

We perform a separate analysis for each type of the inserted vertex w. The first three analyses are identical to what was done in Section 4.4.2 (except for type 1.2 where k is defined to be one less).

Insertion due to size

As in Section 4.4.2, we require that

$$\underline{\sqrt{5}/2 + f + \sqrt{3}/2 \leq a} \text{ and } \underline{\sqrt{3} + 2f + \sqrt{5}/2 \leq 2b}$$

so that the insertion preserves (4.1), where $\underline{f = \rho}$ this time.

Type 1.1 insertion

As in Section 4.4.2 (for a Type 1 insertion), we require that

$$\underline{a + \sqrt{5} \leq 2b},$$

so that by induction $\mathrm{lfs}(w) \leq 2^{k+1}b$.

Type 1.2 insertion

The new vertex w has label SC_{k+1} at a distance at most $2r(\mathrm{SC}_{k+1}) = 2^{k+1}\sqrt{3}$ from v with label BCC_{k+1}. So

$$\mathrm{lfs}(w) \leq \mathrm{lfs}(v) + \mathrm{dist}(v, w) \leq 2^{k+1}b + 2^{k+1}\sqrt{3}.$$

We require that

$$\underline{b + \sqrt{3} \leq a},$$

so that by induction $\mathrm{lfs}(w) \leq 2^{k+1}a$.

Type 3 insertion

By assumption, the two input vertices are at distance at most $(1 + 4\rho/\sqrt{3})r$ from c. By Lemma 9,

$$\mathrm{lfs_i}(w) \leq \mathrm{lfs_i}(c) + \mathrm{dist}(c, w) \leq (1 + 4\rho/\sqrt{3})r + \mathrm{dist}(c, w).$$

- If w was given label SC_j by Algorithm 2, then

$$\mathrm{lfs_i}(w) \leq (1 + 4\rho/\sqrt{3})2^j(\sqrt{5}/2 + \rho) + 2^j\sqrt{3}/2.$$

  We require that

$$\underline{(1 + 4\rho/\sqrt{3})(\sqrt{5}/2 + \rho) + \sqrt{3}/2 \leq a}$$

  so that $\mathrm{lfs_i}(w) \leq 2^j a$.

- Else (w has label BCC_{j+1}),

$$\mathrm{lfs_i}(w) \leq (1 + 4\rho/\sqrt{3})2^j(\sqrt{3} + 2\rho) + 2^j\sqrt{5}/2.$$

  We require that

$$\underline{(1 + 4\rho/\sqrt{3})(\sqrt{3} + 2\rho) + \sqrt{5}/2 \leq 2b}$$

  so that $\mathrm{lfs_i}(w) \leq 2^{j+1} b$.

Type 2 insertion

This case is the most difficult because in the neighborhood of an input vertex, new lattice vertices are not necessarily coarser than their neighbors. Given a vertex of type 2, the idea is to construct a sequence of lattice vertices. The sequence is designed to consist of type 1.1 and type 2 vertices because type 2 vertices do not come from a coarser lattice and they can undo the coarsening progress made by type 1.1 vertices. The sequence is used to discover a nearby vertex of type 0, 1.2 or 3, or to discover a second nearby input vertex. Figure 4.8 illustrates all possible sequences.

Each new vertex w produced by the algorithm will be assigned a *parent* $p(w)$ and a *nearby input vertex* $n(w)$, depending on the type of inserted vertex w.

For type 1.1,	let $p(w) = v$ and $n(w) = n(v)$.
For type 2,	let $p(w) = v$ and $n(w) = u_1$.
For type 0, 1.2, or 3,	let $p(w) = \mathrm{null}$ and $n(w) = \mathrm{null}$.

Lemma 16 *If $p(w) \neq \mathrm{null}$ where w has label SC_k or BCC_{k+1} then w is of type 1.1 or 2 and*

$$\mathrm{dist}(p(w), w) \leq \begin{cases} 2^k\sqrt{5} & \textit{if } w \textit{ is of type 1.1}, \\ 2^k(\sqrt{3} + \rho) & \textit{if } w \textit{ is of type 2}, \\ 2^k(\sqrt{3} + \rho) & \textit{in general}. \end{cases}$$

PROOF: The bounds appear in Algorithms 4 and 5.

For type 1.1, $\mathrm{dist}(p(w), w) \leq 2r(\mathrm{BCC}_{k+1}) = 2^k\sqrt{5}$.

For type 2, $\mathrm{dist}(p(w), w) \leq 2r(\mathrm{SC}_k) + 2^k\rho = 2^k(\sqrt{3} + \rho)$. □

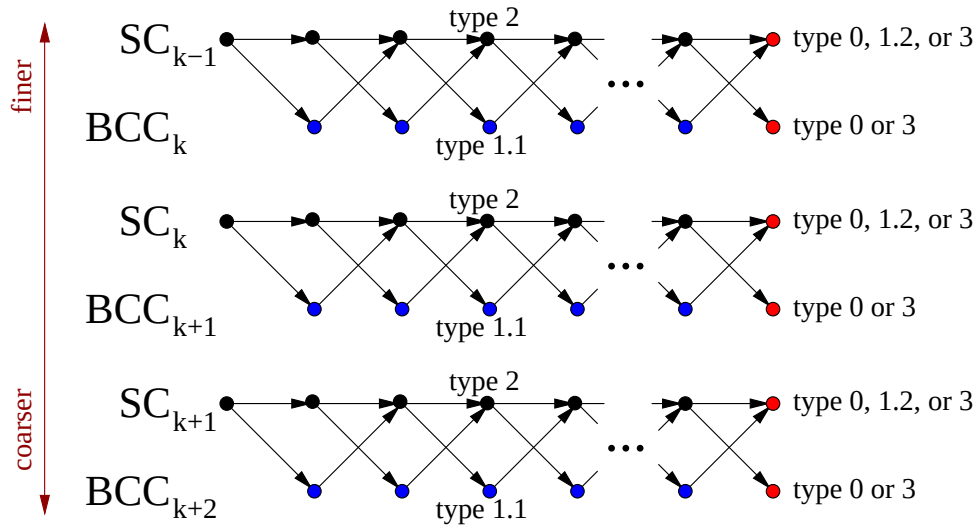

Figure 4.8: This diagram shows all possible parent relationships in the analysis of a Type 2 insertion. Nodes represent vertices and arrows point to all possible parents. A chain of parents correspond to one path in this graph, starting from one of the leftmost vertices (of type 2), and ending at one of the red vertices (or anywhere before). The vertical position of a node indicates its lattice label from the nesting of lattices. We see that a chain can consist of a sequence of type 1.1 or type 2 vertices of arbitrary length from the same two lattices, and must end if a vertex of type 0, 1.2 or 3 is encountered.

Lemma 17 *If* $n(w) \neq \text{null}$ *where* w *has label* SC_k *or* BCC_{k+1} *then* w *is of type 1.1 or 2 and*

$$\text{dist}(n(w), p(w)) \leq \begin{cases} 2^k(3\sqrt{3} + 3\rho) & \textit{if } w \textit{ is of type 1.1}, \\ 2^k(2\sqrt{3} + 2\rho) & \textit{if } w \textit{ is of type 2}, \\ 2^k(3\sqrt{3} + 3\rho) & \textit{in general}. \end{cases}$$

PROOF: For type 2, the bound comes from the input conditions of Algorithm 5.

For type 1.1, let $v = p(w)$ which has label SC_k. Note that $n(v) = n(w) \neq \text{null}$. v must be of type 2 because this is the only type of vertex with label SC and $n(v) \neq \text{null}$. We use the triangle inequality

$$\text{dist}(n(w), p(w)) \leq \text{dist}(n(v), p(v)) + \text{dist}(p(v), v)$$

where $v = p(w)$ and $n(w) = n(v)$. v has label SC_k, so

$$\text{dist}(n(v), p(v)) \leq 2^k(2\sqrt{3} + 2\rho)$$

from the consideration above for type 2, and

$$\operatorname{dist}(p(v), v) \leq 2^k(\sqrt{3} + \rho)$$

from Lemma 16. By combining these inequalities the result follows. □

Lemma 18 *If* $n(w) \neq \text{null}$ *where* w *has label* SC_k *or* BCC_{k+1} *then* w *is of type 1.1 or 2 and*

$$\operatorname{dist}(n(w), w) \leq \begin{cases} 2^k(3\sqrt{3} + 3\rho + \sqrt{5}) & \textit{if } w \textit{ is of type 1.1}, \\ 2^k(3\sqrt{3} + 3\rho) & \textit{if } w \textit{ is of type 2}, \\ 2^k(3\sqrt{3} + 3\rho + \sqrt{5}) & \textit{in general}. \end{cases}$$

PROOF: The bound for each particular type follows by applying the triangle inequality to the corresponding bounds in Lemmas 16 and 17. The bound *in general* is simply the maximum. □

Lemma 19 *If a vertex* w *is of type 0, 1.2, or 3, with label* SC_k *or* BCC_{k+1}*, then* $\operatorname{lfs}(w) \leq 2^k \max(b + \sqrt{3}, \alpha)$*, where*

$$\alpha = (1 + 4\rho/\sqrt{3})(\sqrt{3} + 2\rho) + \sqrt{5}/2. \tag{4.3}$$

PROOF: From the previous analyses we have the following, depending on the type of w.

For type 0, $\operatorname{lfs}_{\text{s}}(w) \leq 2^k(\sqrt{5}/2 + \rho + \sqrt{3}/2)$ or $\operatorname{lfs}_{\text{s}}(w) \leq 2^k(\sqrt{3} + 2\rho + \sqrt{5}/2)$ from Section 4.5.2.

For type 1.2, $\operatorname{lfs}(w) \leq 2^k(b + \sqrt{3})$ from Section 4.5.2 with k replacing $k + 1$.

For type 3, $\operatorname{lfs}_{\text{i}}(w) \leq 2^k((1 + 4\rho/\sqrt{3})(\sqrt{5}/2 + 2\rho) + \sqrt{3}/2)$ or $\operatorname{lfs}_{\text{i}}(w) \leq 2^k((1 + 4\rho/\sqrt{3})(\sqrt{3} + 2\rho) + \sqrt{5}/2)$ from Section 4.5.2.

The result follows by considering the maximum over all possibilities. □

Now, starting from a vertex w of type 2, follow the chain of parents as follows.

- Initialize $w' := w$.
- While $p(w')$ is of type 1.1 or 2, and $n(p(w')) = n(w)$, do $w' := p(w')$.

Note that by construction, every vertex of the chain from w to w' has label SC_k or BCC_{k+1} with the *same* k. $n(w') = n(w)$ because otherwise the *while* loop would have stopped on the previous iteration. Furthermore, w' must be of type 2 because if w' is of type 1.1 with $n(w') = n(w) \neq \text{null}$, then $p(w')$ is of type 2 with $n(p(w')) = n(w)$ and the *while* loop would have continued.

$\operatorname{dist}(p(w'), n(w')) \leq 2^k(2\sqrt{3} + 2\rho)$ since w' is of type 2.

$\mathrm{dist}(n(w'), w) \leq 2^k(3\sqrt{3} + 3\rho)$ since $n(w') = n(w)$ and w is of type 2.
By the triangle inequality,

$$\mathrm{dist}(p(w'), w) \leq 2^k(5\sqrt{3} + 5\rho).$$

The rest of the analysis depends on which condition made the *while* loop stop.

- If $p(w')$ is of type 0, 1.2, or 3:

  By Lemma 19,

  $$\mathrm{lfs}(p(w')) \leq 2^k \max(b + \sqrt{3}, \alpha).$$

  By Lemma 9,

  $$\mathrm{lfs}(w) \leq \mathrm{lfs}(p(w')) + \mathrm{dist}(p(w'), w) \leq 2^k \max(b + \sqrt{3}, \alpha) + 2^k(5\sqrt{3} + 5\rho).$$

  We require that

  $$\underline{\max(b + \sqrt{3}, \alpha) + 5\sqrt{3} + 5\rho \leq a}$$

  so that by induction $\mathrm{lfs}(w) \leq 2^k a$.

- Else if $n(p(w')) = \mathrm{null}$:

  $p(w')$ must be of type 1.1 and $p(p(w'))$ of type 0, 1.2 or 3. By Lemma 19,

  $$\mathrm{lfs}(p(p(w'))) \leq 2^k \max(b + \sqrt{3}, \alpha).$$

  By Lemma 16,

  $$\mathrm{dist}(p(p(w')), p(w')) \leq 2^k\sqrt{5}$$

  since $p(w')$ is of type 1.1. By Lemma 9 and the triangle inequality,

  $$\begin{aligned} \mathrm{lfs}(w) &\leq \mathrm{lfs}(p(p(w'))) + \mathrm{dist}(p(p(w')), p(w')) + \mathrm{dist}(p(w'), w) \\ &\leq 2^k \max(b + \sqrt{3}, \alpha) + 2^k\sqrt{5} + 2^k(5\sqrt{3} + 5\rho). \end{aligned}$$

  We require that

  $$\underline{\max(b + \sqrt{3}, \alpha) + \sqrt{5} + 5\sqrt{3} + 5\rho \leq a}$$

  so that by induction $\mathrm{lfs}(w) \leq 2^k a$.

- Else ($n(p(w')) \neq n(w)$ and $n(p(w')) \neq \mathrm{null}$):

  By Lemma 18,

  $$\mathrm{dist}(n(p(w')), p(w')) \leq 2^k(3\sqrt{3} + 3\rho + \sqrt{5}).$$

By the triangle inequality,

$$\begin{aligned} \mathrm{dist}(n(p(w')), w) &\leq \mathrm{dist}(n(p(w')), p(w')) + \mathrm{dist}(p(w'), w) \\ &\leq 2^k(3\sqrt{3} + 3\rho + \sqrt{5}) + 2^k(5\sqrt{3} + 5\rho). \end{aligned}$$

On the other hand,

$$\mathrm{dist}(n(w), w) \leq 3\sqrt{3} + 3\rho$$

since w is of type 2. $n(p(w')) \neq n(w)$, which implies that

$$\begin{aligned} \mathrm{lfs_i}(w) &\leq \max(\mathrm{dist}(n(p(w')), w), \mathrm{dist}(n(w), w)) \\ &\leq 2^k(8\sqrt{3} + 8\rho + \sqrt{5}). \end{aligned}$$

We require that

$$\underline{8\sqrt{3} + 8\rho + \sqrt{5} \leq a}$$

so that $\mathrm{lfs}(w) \leq 2^k a$.

Guarantee

The requirements (underlined in the text) can be satisfied by setting

$$\begin{aligned} a &= \alpha + \sqrt{5} + 5\sqrt{3} + 5\rho, \text{ and} \\ b &= (a + \sqrt{5})/2. \end{aligned}$$

Theorem 13 does not apply directly because of the input vertices (which have no labels). We need to add two cases to the proof of Theorem 13:

- If v and w are both input vertices, then we directly have $\mathrm{dist}(v, w) \geq \mathrm{lfs}(v)$.
- If v is a refinement vertex and w is an input vertex, then:
  - In case v has label SC_k: $\mathrm{dist}(v, w) \geq 2^k\sigma$, where $\sigma = (2 + \sqrt{6})/4$.
  - In case v has label BCC_k: $\mathrm{dist}(v, w) \geq 2^k\sigma/2$.

The bound given by Theorem 13 becomes

$$\min(\tfrac{1}{1+a}, \tfrac{1}{1+2b/\sigma})\mathrm{lfs}(v).$$

By applying this, we deduce that during the course of the algorithm the distance between any vertex v and its nearest neighbor is at least

$$\mathrm{lfs}(v)/53.086.$$

This implies that the algorithm terminates, as argued in Section 4.4.2.

4.6 Discussion

The use of two types of lattices is complicated but appears to be necessary to obtain a lower bound of 30° on dihedral angles. For a system that would give good grading based uniquely on one of the simple, body-centered, or face-centered cubic lattices we could only obtain these respective approximate lower bounds: 17.023°, 14.312°, and 14.197°.

Lattice refinement tends to create coplanar and cospherical vertices so the Delaunay code must be robust. Fortunately, lattice point coordinates are rational numbers with a power-of-two denominator, so in reasonable cases they are exactly represented by floating-point numbers.

Because of the guarantee on tetrahedron quality and the bound on element size in terms of local feature size, when the domain is convex it should be possible to prove size-optimality (Section 1.2.2). This doesn't hold for non-convex domains where two input vertices can be very close geometrically, but quite far apart when the distance is computed as a geodesic inside the domain. In this case our algorithm will refine the mesh around the two input vertices while the optimal mesh doesn't need to. The same issue affects conforming Delaunay mesh generation algorithms.

The time complexity of the algorithm depends on the particularities of mesh operations like point location and incremental insertion, so we leave the analysis as an open problem.

It would be interesting to perform experiments with Algorithm 3 to see how much refinement is done in practice compared to standard Delaunay refinement, and when trying to eliminate slivers from an already refined mesh.

We hope that this work is just a first step toward a lattice refinement algorithm for domains with boundary constraints. As a step in that direction, we showed in Section 4.5 how internal input vertices can be handled. We believe that planar constraints can also be handled without much difficulty. Sharp features like segments and corners are more challenging because when a segment is split, there is only one degree of freedom for the position of the split point. Of course, the dihedral angle guarantees don't have to stay as good as 30° and 135°.

Bibliography

[1] Pierre Alliez, David Cohen-Steiner, Mariette Yvinec, and Mathieu Desbrun. Variational Tetrahedral Meshing. *ACM Transactions on Graphics*, 24(3):617–625, 2005. Special issue on Proceedings of SIGGRAPH 2005.

[2] Nina Amenta and Marshall Bern. Surface Reconstruction by Voronoi Filtering. *Discrete & Computational Geometry*, 22(4):481–504, December 1999.

[3] Nina Amenta, Sunghee Choi, Tamal Krishna Dey, and N. Leekha. A Simple Algorithm for Homeomorphic Surface Reconstruction. *International Journal of Computational Geometry and Applications*, 12(1–2):125–141, 2002.

[4] Dominique Attali, David Cohen-Steiner, and Herbert Edelsbrunner. Extraction and Simplification of Iso-surfaces in Tandem. In *Symposium on Geometry Processing 2005*, pages 139–148, Vienna, Austria, July 2005. Eurographics Association.

[5] J. Andreas Bærentzen and Henrik Aanæs. Generating Signed Distance Fields from Triangle Meshes. Technical Report IMM-TR-2002-21, Informatics and Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark, 2002.

[6] Brenda S. Baker, Eric Grosse, and Conor S. Rafferty. Nonobtuse Triangulation of Polygons. *Discrete and Computational Geometry*, 3(2):147–168, 1988.

[7] Adam W. Bargteil, Tolga G. Goktekin, James F. O'Brien, and John A. Strain. A Semi-Lagrangian Contouring Method for Fluid Simulation. *ACM Transactions on Graphics*, 25(1):19–38, January 2006.

[8] Marshall Bern and David Eppstein. Mesh Generation and Optimal Triangulation. In Ding-Zhu Du and Frank Hwang, editors, *Computing in Euclidean Geometry*, volume 1 of *Lecture Notes Series on Computing*, pages 23–90. World Scientific, Singapore, 1992.

[9] Marshall Bern, David Eppstein, and John R. Gilbert. Provably Good Mesh Generation. *Journal of Computer and System Sciences*, 48(3):384–409, June 1994.

[10] Marshall Bern and Paul Plassmann. Mesh Generation. In Jörg Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, chapter 6. Elsevier Science, 2000.

[11] Jürgen Bey. Tetrahedral Grid Refinement. *Computing*, 55:355–378, 1995.

[12] Daniel K. Blandford, Guy E. Blelloch, David E. Cardoze, and Clemens Kadow. Compact Representations of Simplicial Meshes in Two and Three Dimensions. *International Journal of Computational Geometry and Applications*, 15(1):3–24, February 2005.

[13] Jules Bloomenthal. An Implicit Surface Polygonizer. In *Graphics Gems IV*, chapter IV.8, pages 324–349. Academic Press, 1994.

[14] Jean-Daniel Boissonnat and Steve Oudot. Provably Good Surface Sampling and Approximation. In *Symposium on Geometry Processing*, pages 9–18. Eurographics Association, June 2003.

[15] Sergey N. Borovikov, Igor A. Kryukov, and Igor E. Ivanov. An Approach for Delaunay Tetrahedralization of Bodies with Curved Boundaries. In *Fourteenth International Meshing Roundtable*, pages 221–238, San Diego, California, September 2005. Sandia National Laboratories.

[16] Adrian Bowyer. Computing Dirichlet Tessellations. *Computer Journal*, 24(2):162–166, 1981.

[17] Scott A. Canann, S. N. Muthukrishnan, and R. K. Phillips. Topological Refinement Procedures for Triangular Finite Element Meshes. *Engineering with Computers*, 12(3 & 4):243–255, 1996.

[18] Hamish Carr, Torsten Möller, and Jack Snoeyink. Artifacts Caused by Simplicial Subdivision. *IEEE Transactions on Visualization and Computer Graphics*, 12(2):231–242, 2006.

[19] Long Chen. Mesh Smoothing Schemes Based on Optimal Delaunay Triangulations. In *Proceedings of the Thirteenth International Meshing Roundtable*, pages 109–120, Williamsburg, Virginia, September 2004.

[20] Siu-Wing Cheng and Tamal K. Dey. Quality Meshing with Weighted Delaunay Refinement. *SIAM Journal on Computing*, 33(1):69–93, 2003.

[21] Siu-Wing Cheng, Tamal Krishna Dey, Herbert Edelsbrunner, Michael A. Facello, and Shang-Hua Teng. Sliver Exudation. *Journal of the ACM*, 47(5):883–904, September 2000.

[22] Evgeni V. Chernyaev. Marching Cubes 33: Construction of Topologically Correct Isosurfaces. Technical Report CERN-CN-95-17, European Organization for Nuclear Research, Geneva, Switzerland, 1995.

[23] L. Paul Chew. Constrained Delaunay Triangulations. *Algorithmica*, 4(1):97–108, 1989.

[24] L. Paul Chew. Guaranteed-Quality Triangular Meshes. Technical Report TR-89-983, Department of Computer Science, Cornell University, 1989.

[25] L. Paul Chew. Guaranteed-Quality Delaunay Meshing in 3D. In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*, pages 391–393, June 1997.

[26] Philippe G. Ciarlet. *The Finite Element Method for Elliptic Problems*. North-Holland, Amsterdam, 1978.

[27] Kenneth L. Clarkson and Peter W. Shor. Applications of Random Sampling in Computational Geometry, II. *Discrete & Computational Geometry*, 4(1):387–421, 1989.

[28] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. Delaunay Triangulations. In *Computational Geometry: Algorithms and Applications*, chapter 9. Springer-Verlag, Berlin, 1997.

[29] Boris Nikolaevich Delaunay. Sur la Sphère Vide. *Izvestia Akademia Nauk SSSR, VII Seria, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7:793–800, 1934.

[30] J. Donea, Antonio Huerta, J.-Ph. Ponthot, and A. Rodríguez-Ferran. Arbitrary Lagrangian-Eulerian Methods. In *Encyclopedia of Computational Mechanics, Volume 1*, chapter 14. John Wiley, 2004.

[31] Rex A. Dwyer. A Faster Divide-and-Conquer Algorithm for Constructing Delaunay Triangulations. *Algorithmica*, 2(2):137–151, 1987.

[32] Herbert Edelsbrunner and Damrong Guoy. An Experimental Study of Sliver Exudation. In *Tenth International Meshing Roundtable*, pages 307–316, Newport Beach, California, October 2001.

[33] Herbert Edelsbrunner, Xiang-Yang Li, Gary Miller, Andreas Stathopoulos, Dafna Talmor, Shang-Hua Teng, Alper Üngör, and Noel Walkington. Smoothing and Cleaning Up Slivers. In *Proceedings of the 32nd Annual Symposium on the Theory of Computing*, pages 273–278, Portland, Oregon, May 2000. Association for Computing Machinery.

[34] Herbert Edelsbrunner and Nimish R. Shah. Triangulating Topological Spaces. In *Proceedings of the Tenth Annual Symposium on Computational Geometry*, pages 285–292, 1994.

[35] David Eppstein, John M. Sullivan, and Alper Üngör. Tiling Space and Slabs with Acute Tetrahedra. *Computational Geometry: Theory and Applications*, 27(3):237–255, March 2004.

[36] EXACUS. Libraries for efficient and exact algorithms for curves and surfaces. http://www.mpi-inf.mpg.de/projects/EXACUS/.

[37] David A. Field. Implementing Watson's Algorithm in Three Dimensions. In *Proceedings of the Second Annual Symposium on Computational Geometry*, pages 246–259, Yorktown Heights, New York, June 1986. Association for Computing Machinery.

[38] David A. Field. Qualitative Measures for Initial Meshes. *International Journal for Numerical Methods in Engineering*, 47:887–906, 2000.

[39] David A. Field and Warren D. Smith. Graded Tetrahedral Finite Element Meshes. *International Journal for Numerical Methods in Engineering*, 31(3):413–425, March 1991.

[40] Steven Fortune. A Sweepline Algorithm for Voronoi Diagrams. *Algorithmica*, 2(2):153–174, 1987.

[41] Lori A. Freitag and Carl Ollivier-Gooch. Tetrahedral Mesh Improvement Using Swapping and Smoothing. *International Journal for Numerical Methods in Engineering*, 40(21):3979–4002, November 1997.

[42] William H. Frey. Selective Refinement: A New Strategy for Automatic Node Placement in Graded Triangular Meshes. *International Journal for Numerical Methods in Engineering*, 24(11):2183–2200, November 1987.

[43] Alexander Fuchs. Automatic Grid Generation with Almost Regular Delaunay Tetrahedra. In *Seventh International Meshing Roundtable*, pages 133–148, October 1998.

[44] R.A. Gingold and J.J. Monaghan. Smoothed Particle Hydrodynamics: Theory and Application to Non-Spherical Stars. *Monthly Notices of the Royal Astronomical Society*, 181:375–389, 1977.

[45] Leonidas J. Guibas and Jorge Stolfi. Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams. *ACM Transactions on Graphics*, 4(2):74–123, April 1985.

[46] L. R. Hermann. Laplacian-Isoparametric Grid Generation Scheme. *Journal of the Engineering Mechanics Division of the American Society of Civil Engineers*, 102:749–756, October 1976.

[47] Martin Isenburg and Peter Lindstrom. Streaming meshes. In *Visualization 2005*, pages 231–238, Minneapolis, Minnesota, October 2005. IEEE.

[48] Claes Johnson. *Numerical Solution of Partial Differential Equations by the Finite Element Method*. Cambridge University Press, Cambridge, 1987.

[49] Leif P. Kobbelt, Mario Botsch, Ulrich Schwanecke, and Hans-Peter Seidel. Feature Sensitive Surface Extraction from Volume Data. In *Computer Graphics (SIGGRAPH 2001 Proceedings)*, pages 57–66, 2001.

[50] Ravikrishna Kolluri. Provably Good Moving Least Squares. Submitted to *ACM Transactions on Algorithms*, 2007.

[51] Charles L. Lawson. Software for C^1 Surface Interpolation. In John R. Rice, editor, *Mathematical Software III*, pages 161–194. Academic Press, New York, 1977.

[52] Der-Tsai Lee and Bruce J. Schachter. Two Algorithms for Constructing a Delaunay Triangulation. *International Journal of Computer and Information Sciences*, 9(3):219–242, 1980.

[53] Xiang-Yang Li and Shang-Hua Teng. Generating Well-Shaped Delaunay Meshes in 3D. In *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms*, pages 28–37, January 2001.

[54] Xiang-Yang Li, Shang-Hua Teng, and Alper Üngör. Biting Ellipses to Generate Anisotropic Mesh. In *Eighth International Meshing Roundtable*, pages 97–108, South Lake Tahoe, California, October 1999.

[55] Xiang-Yang Li, Shang-Hua Teng, and Alper Üngör. Biting Spheres in 3d. In *Eighth International Meshing Roundtable*, pages 85–95, South Lake Tahoe, California, October 1999.

[56] Xiang-Yang Li, Shang-Hua Teng, and Alper Üngör. Biting: Advancing Front Meets Sphere Packing. *International Journal for Numerical Methods in Engineering*, 49(1):61–81, September 2000.

[57] Anwei Liu and Barry Joe. Relationship between Tetrahedron Shape Measures. *BIT*, 34:268–287, 1994.

[58] R. Löhner and P. Parikh. Generation of Three-Dimensional Unstructured Grids by the Advancing Front Method. *International Journal of Numerical Methods in Fluids*, 8(10):1135–1149, 1988.

[59] William E. Lorensen and Harvey E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *Computer Graphics (SIGGRAPH '87 Proceedings)*, pages 163–170, Anaheim, California, July 1987.

[60] L. B. Lucy. A Numerical Approach to Testing the Fission Hypothesis. *The Astronomical Journal*, 82(12):1013–1924, December 1977.

[61] Gary L. Miller, Dafna Talmor, Shang-Hua Teng, and Noel Walkington. A Delaunay Based Numerical Method for Three Dimensions: Generation, Formulation, and Partition. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing*, pages 683–692, Las Vegas, Nevada, May 1995.

[62] Gary L. Miller, Dafna Talmor, Shang-Hua Teng, Noel Walkington, and Han Wang. Control Volume Meshes Using Sphere Packing: Generation, Refinement and Coarsening. In *Fifth International Meshing Roundtable*, pages 47–61, Pittsburgh, Pennsylvania, October 1996.

[63] Scott A. Mitchell and Stephen A. Vavasis. Quality Mesh Generation in Three Dimensions. In *Proceedings of the Eighth Annual Symposium on Computational Geometry*, pages 212–221, 1992.

[64] Scott A. Mitchell and Stephen A. Vavasis. Quality Mesh Generation in Higher Dimensions. *SIAM Journal on Computing*, 29(4):1334–1370, 2000.

[65] Neil Molino, Robert Bridson, Joseph Teran, and Ronald Fedkiw. A Crystalline, Red Green Strategy for Meshing Highly Deformable Objects with Tetrahedra. In *Twelfth International Meshing Roundtable*, pages 103–114, Santa Fe, New Mexico, September 2003.

[66] David J. Naylor. Filling Space with Tetrahedra. *International Journal for Numerical Methods in Engineering*, 44(10):1383–1395, April 1999.

[67] Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel. Multi-Level Partition of Unity Implicits. *ACM Transactions on Graphics*, 22(3):463–470, July 2003. Special issue on Proceedings of SIGGRAPH 2003.

[68] Stanley Osher and Ronald Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer-Verlag, New York, 2002.

[69] Steve Oudot, Laurent Rineau, and Mariette Yvinec. Meshing Volumes Bounded by Smooth Surfaces. In *Proceedings of the 14th International Meshing Roundtable*, pages 203–219, 2005.

[70] Steven J. Owen. A Survey of Unstructured Mesh Generation Technology. In *Proceedings of the Seventh International Meshing Roundtable*, pages 239–267, Dearborn, Michigan, October 1998.

[71] Steven E. Pav and Noel J. Walkington. Robust Three Dimensional Delaunay Refinement. In *Thirteenth International Meshing Roundtable*, pages 145–156, Williamsburg, Virginia, September 2004. Sandia National Laboratories.

[72] Steven E. Pav and Noel J. Walkington. Delaunay Refinement by Corner Lopping. In *Fourteenth International Meshing Roundtable*, pages 165–182, San Diego, California, September 2005. Sandia National Laboratories.

[73] Shmuel Rippa. Long and Thin Triangles Can Be Good for Linear Interpolation. *SIAM Journal on Numerical Analysis*, 29(1):257–270, February 1992.

[74] Jim Ruppert. A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation. *Journal of Algorithms*, 18(3):548–585, May 1995.

[75] Marjorie Senechal. Which Tetrahedra Fill Space? *Mathematics Magazine*, 54(5):227–243, 1981.

[76] James A. Sethian. A Fast Marching Level Set Method for Monotonically Advancing Fronts. *Proceedings of the National Academy of Sciences*, 93(4):1591–1595, February 1996.

[77] Chen Shen, James F. O'Brien, and Jonathan R. Shewchuk. Interpolating and Approximating Implicit Surfaces from Polygon Soup. *ACM Transactions on Graphics*, 23(3):896–904, August 2004. Special issue on Proceedings of SIGGRAPH 2004.

[78] Jonathan Richard Shewchuk. Tetrahedral Mesh Generation by Delaunay Refinement. In *Proceedings of the Fourteenth Annual Symposium on Computational Geometry*, pages 86–95, Minneapolis, Minnesota, June 1998. Association for Computing Machinery.

[79] Jonathan Richard Shewchuk. Constrained Delaunay Tetrahedralizations and Provably Good Boundary Recovery. In *Eleventh International Meshing Roundtable*, pages 193–204, Ithaca, New York, September 2002. Sandia National Laboratories.

[80] Jonathan Richard Shewchuk. What Is a Good Linear Element? Interpolation, Conditioning, and Quality Measures. In *Eleventh International Meshing Roundtable*, pages 115–126, September 2002.

[81] D. M. Y. Sommerville. Space-Filling Tetrahedra in Euclidean Space. *Proceedings of the Edinburgh Mathematical Society*, 41:49–57, 1923.

[82] Graham M. Treece, Richard W. Prager, and Andrew H. Gee. Regularised Marching Tetrahedra: Improved Iso-Surface Extraction. *Computers & Graphics*, 23(4):583–598, 1999.

[83] LeeAnn Tzeng. Warping Cubes: Better Triangles from Marching Cubes. In *20th European Workshop on Computational Geometry*, Seville, Spain, March 2004.

[84] Alper Üngör. Off-Centers: A New Type of Steiner Points for Computing Size-Optimal Guaranteed-Quality Delaunay Triangulations. In *Latin American Theoretical Informatics*, pages 152–161, Buenos Aires, Argentina, April 2004.

[85] David F. Watson. Computing the n-dimensional Delaunay Tessellation with Application to Voronoi Polytopes. *Computer Journal*, 24(2):167–172, 1981.

[86] Mark A. Yerry and Mark S. Shephard. A Modified Quadtree Approach to Finite Element Mesh Generation. *IEEE Computer Graphics and Applications*, 3:39–46, January/February 1983.

[87] Mark A. Yerry and Mark S. Shephard. Automatic Three-Dimensional Mesh Generation by the Modified-Octree Technique. *International Journal for Numerical Methods in Engineering*, 20(11):1965–1990, November 1984.

[88] Hong-Kai Zhao, Stanley Osher, and Ronald Fedkiw. Fast Surface Reconstruction Using the Level Set Method. In *Workshop on Variational and Level Set Methods*, pages 194–202, July 2001.