Improving Tetrahedral Meshes

Bryan Matthew Klingner

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2008-145
http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-145.html

November 25, 2008

Improving Tetrahedral Meshes

by

Bryan Matthew Klingner

B.S. (University of Texas, Austin) 2003
B.A. (University of Texas, Austin) 2003
M.S. (University of California, Berkeley) 2006

A dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Engineering-Electrical Engineering and Computer Sciences

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor Jonathan Shewchuk, Chair
Professor James O'Brien
Professor Sara McMains

Fall 2008

The dissertation of Bryan Matthew Klingner is approved.

Chair	Date
	Date
	Date

University of California, Berkeley

Fall 2008

Improving Tetrahedral Meshes

by

Bryan Matthew Klingner

Abstract

Improving Tetrahedral Meshes

by

Bryan Matthew Klingner

Doctor of Philosophy in Engineering-Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Jonathan Shewchuk, Chair

I present a tetrahedral mesh improvement method that creates meshes whose worst tetrahedra have a level of quality substantially better than those produced by any previous method for tetrahedral mesh generation or "mesh clean-up." Mesh optimization methods often get stuck in bad local optima (poor-quality meshes) because their repertoire of mesh transformations is weak. In contrast, my mesh improvement software employs a broader palette of operations than any other. Alongside the best traditional topological and smoothing operations, I introduce a topological transformation that inserts a new vertex, as well as methods for smoothing vertices on the boundary of the mesh. My implementation routinely improves meshes so that all the dihedral angles lie between 34 and 131 degrees. It also allows a user to locally control the sizes and grading of the tetrahedra, and to generate anisotropic meshes with local control of the orientations and eccentricities of the tetrahedra. With the same operations, I develop a dynamic mesh improvement method for simulations of deforming materials that updates a mesh at each timestep to maintain the quality of its tetrahedra. The dynamic mesher strikes a balance between maintaining high element quality and minimizing the error introduced through artificial diffusion.

Professor Jonathan Shewchuk
Dissertation Committee Chair

Contents

Chapter 1

Introduction

The finite element and finite volume methods are invaluable tools for solving complex engineering problems in structural analysis, fluid dynamics, electromagnetism, and many other areas. These tools rely on a *mesh*, a discretization of space into simple geometric pieces that makes numerical solution possible. Tetrahedral meshes are a popular choice for discretization of three-dimensional domains, and can be generated using advancing front, Delaunay, or octree methods. Not every mesh is suitable for numerical computation, however. Poorly-shaped tetrahedra in a mesh can introduce numerical errors and increase the time needed to find a solution. Hence, there is a market for mesh improvement tools that can improve the quality of the tetrahedra in an existing mesh. The *quality* of a tetrahedron is a number that estimates its good or bad effects on interpolation error, discretization error, and stiffness matrix conditioning.

The overall quality of a mesh is largely dictated by its worst elements. Although existing mesh improvement techniques are usually effective at improving the average element quality in a mesh, they are often hamstrung by their inability to improve the worst ones. In this dissertation, I demonstrate a combination of mesh improvement operations that consistently pushes the worst elements in a tetrahedral mesh to levels of quality not attained by any previous technique.

I explore two styles of mesh improvement: *static* and *dynamic*. Mesh generation algorithms often yield meshes that are unusable in practice because of poor quality tetrahedra. In static mesh improvement, I try to improve a single mesh to as high a quality as possible, changing as much

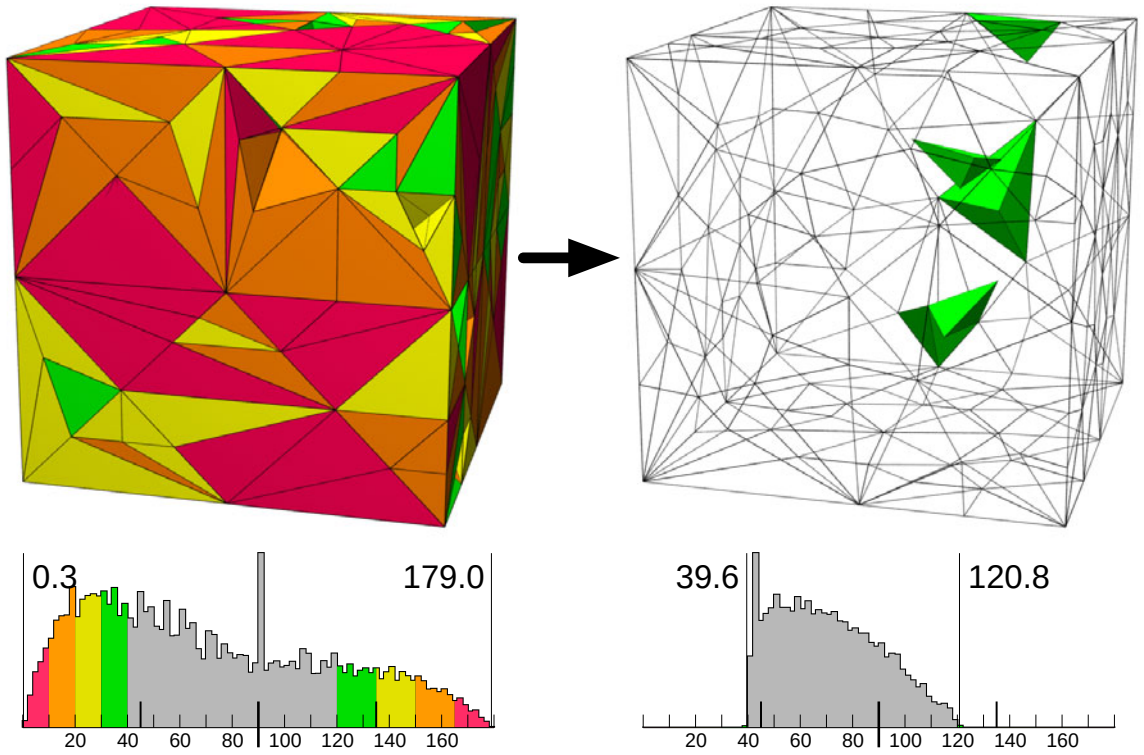


Figure 1.1. An example of tetrahedral mesh improvement using my static improvement schedule. At left is an input mesh and a histogram of its dihedral angles. On the right is the mesh after improvement. Tetrahedra are colored according to their worst dihedral angle as marked on the histograms below. Tetrahedra with no dihedral angle smaller than 40° nor larger than 120° are not rendered. The histograms plot the number of dihedral angles in each 2° interval.

of the mesh and expending as much computation time as is necessary to rescue a mesh and make it usable. Figure 1.1 shows a mesh that improves from exceptionally poor to exceptionally high quality.

In dynamic mesh improvement, a mesh deforms as dictated by a physical simulation or other process, hurting the quality of its tetrahedra. I try to repair only the tetrahedra that have fallen below a minimum quality, while changing as little of the mesh as possible.

Most operations for mesh improvement can be classified into one of two categories. *Smoothing* is the act of moving one or more mesh vertices to improve the quality of the elements adjoining them. Smoothing does not change the connectivity (topology) of the mesh. *Topological transformations* are operations that remove elements from a mesh and replace them with a different set of elements occupying the same space, changing the topological structure of the mesh in the process.

Smoothing lies largely in the domain of numerical optimization, and topological transformations in the domain of combinatorial optimization. The two techniques are most effective when used in concert.

Smoothing and topological transformations are usually used as operations in a *hill-climbing* method for optimizing a mesh. An *objective function* maps each possible mesh to a numerical value (or sequence of values) that describes the quality of the mesh. A hill-climbing method considers performing an operation at a specific site in the mesh. If the quality of the changed mesh will be greater than that of the original mesh, the operation is performed; then the hill-climbing method searches for another operation that will improve the new mesh. Operations that do not improve the value of the objective function are not performed. Thus, the final mesh cannot be worse than the input mesh. Hill climbing stops when no operation can achieve further improvement (the mesh is locally optimal), or when further optimization promises too little gain for too much expenditure of time.

The work of Freitag and Ollivier-Gooch [23] is the best example to date of such a hill-climbing method for tetrahedral mesh optimization, and is the inspiration for my work on mesh improvement. In it, the authors combine optimization-based smoothing with several topological transformations. They report the performance of several schedules for performing these operations on a variety of meshes, show that their best schedule eliminates most poorly shaped tetrahedra, and offer empirical recommendations about what makes some schedules better than others. I follow in the footsteps of this research, seeking yet better results through an expanded set of operations.

Delaunay mesh generation algorithms achieve good results by inserting new vertices into a Delaunay triangulation [24], often boosting the smallest dihedral angle to 19° or more [45]. But no tetrahedral "mesh clean-up" paper I know of uses transformations that add new vertices to the mesh—a strange omission. No doubt this lack stems partly from the desire not to increase the number of tetrahedra in a mesh.

I show in Chapters 3, 4, and 5 that the addition of a new mesh improvement operation that inserts a vertex makes it possible to achieve levels of mesh quality that, to the best of my knowledge, are unprecedented. My implementation usually improves meshes so that no dihedral angle is smaller

than 34° or larger than 131°. It sometimes achieves extreme angles bounded between 41° and 117°. No previous software I know of for tetrahedral mesh generation or mesh improvement achieves angles of even 22° or 155° with much consistency.

As a combinatorial optimization problem, mesh improvement is not well behaved. The *search space* is the set of all possible meshes of a fixed geometric domain. A transformation (including smoothing) is an operation that transforms one mesh of the domain to another. These operations give the search space structure by dictating what meshes are immediately reachable from another mesh. The *objective function*, which maps the search space to quality scores, has many local optima (meshes whose quality cannot be improved by any single transformation at hand), and it is unlikely that any practical mesh improvement algorithm will ever find the global optimum—the best possible mesh of the domain—or even come close.

Luckily, finding a global optimum is unnecessary. For many domains, there are many mesh configurations in which all the tetrahedra are excellent; all a successful algorithm needs to do is locate one of them. To this end, I have assembled a broader set of operations than any before employed, in the hope that this will "smooth out" the search space of meshes enough that hill climbing will never get stuck in a bad local minimum. In addition to optimization-based vertex smoothing, 2-3 and 3-2 flips, and edge removal (the operations used by Freitag and Ollivier-Gooch), I have added the following:

- A topological transformation that inserts a new vertex, usually into a bad tetrahedron. Sometimes it deletes vertices as well. (Section 3.4.)
- A topological transformation that collapses an edge, removing one of the edge's endpoints. (Section 3.3.)
- Smoothing of vertices constrained to lie on or near the boundary of the mesh. (Section 3.2.3.)
- Edge removal for boundary edges.
- The *multi-face removal* operation of de Cougny and Shephard [18]. (Section 3.1.2.)
- Compound operations that combine several successive operations in the hope of getting over

a valley in the objective function and finding a better peak. If unsuccessful, these operations are rolled back. (Section 3.4.)

Some of these additions are motivated by the observation that the most difficult tetrahedra to improve are nearly always found adjacent to the mesh boundary. By smoothing boundary vertices, inserting new vertices on the boundary, and permitting flips on the boundary (none of which Freitag and Ollivier-Gooch implemented), I can repair a bad surface triangulation while breaking down recalcitrant boundary tetrahedra, leaving behind better tetrahedra. I have implemented and tested mesh improvement schedules that use these operations, and I investigate the consequences of turning different operations on or off.

The mesh improvement process I describe is agnostic about the size of a mesh's tetrahedra, and can cause significant changes to tetrahedron sizes and the number of tetrahedra in a mesh. Some applications would be better served by a schedule that removes bad tetrahedra without changing the size of the mesh. Some applications may need tetrahedra of an entirely different size.

A user may want to *refine* a mesh, reducing the average edge length, or *coarsen* it, increasing the average edge length. They may want a *graded* mesh that has smaller tetrahedra in some areas and larger tetrahedra in others. In Chapter 6, I describe a method for *size control* of tetrahedra that matches the sizes of tetrahedra in a mesh with the demands of the user. Size control uses some of the same operations as mesh improvement (vertex insertion and edge contraction), but it aims to optimize the edge lengths of tetrahedra, not their quality. After size control is complete, the mesh improvement algorithm restores the quality of the tetrahedra but disallows transformations that create edges of unacceptable lengths. It thereby maintains the specified edge lengths.

In many applications, a function being approximated over a mesh has much greater curvature in one direction than another. Small equilateral tetrahedra may yield accurate approximations, but equal accuracy can be achieved with many fewer *anisotropic* tetrahedra. Anisotropic tetrahedra are elongated and oriented according to the curvature of the function. They provide more resolution along the directions that need it, without the high tetrahedron count of isotropic tetrahedra.

I describe a way to use my mesh improvement algorithm to create anisotropic meshes in Chapter 7. Users specify the anisotropy they desire with a *scaling tensor field*. This tensor provides a

mapping between the physical space of the mesh and an isotropic space where the quality of tetrahedra can be measured by standard isotropic quality measures. Mesh improvement operations adapt the shape of the tetrahedra to conform to the anisotropy specified by the scaling tensor field, yielding an anisotropic mesh.

The most ambitious application of mesh improvement methods is in dynamic mesh generation. Meshes used in *Lagrangian* simulations of physical processes, where vertices of the mesh move according to the physical laws governing the material they represent, change shape with time. For example, a simulated elastic material must compress and stretch realistically, altering the shape of a mesh and its elements drastically. If the quality of the elements degrades too much due to this deformation, the simulation can become inaccurate or crash.

One way of maintaining tetrahedron quality in deforming meshes is to regenerate the entire mesh from scratch whenever the tetrahedra become too deformed. Unfortunately, this method quickly accumulates large numerical errors because of the need to reinterpolate physical properties such as velocity and strain from the old mesh to the new mesh. This loss of accuracy through repeated reinterpolation is sometimes called *artificial diffusion*.

Most of this reinterpolation can be prevented by remeshing only the bad parts of the mesh. I propose a dynamic mesh improvement schedule that does just this, striving to maintain a minimum tetrahedron quality while altering as little of the mesh as possible. In Chapters 8 and 9, I show that by integrating dynamic meshing into a simulation, I can achieve better results and make artifacts caused by artificial diffusion disappear.

Chapter 2

The Quality of Tetrahedra and Meshes

Tetrahedral meshes are used to form piecewise polynomial (usually linear) approximations of functions. The *quality* of a tetrahedron denotes how suitable it is for a particular approximation. Many factors contribute to the quality of a tetrahedron, but of all its properties, the *dihedral angles* formed between each pair of faces are particularly important. Figure 2.1 illustrates a dihedral angle between two triangular faces. In the finite element method, large dihedral angles (near 180°) cause large interpolation errors, which hurt the accuracy of a simulation [28; 35; 47], and small dihedral angles have a disastrous affect on stiffness matrix conditioning [4; 47]. A clear and complete explanation of the costs of extreme dihedral angles in finite element meshes can be found in Shewchuk's *What is a Good Linear Finite Element* [47]. Examples of some tetrahedra with extreme dihedral an-

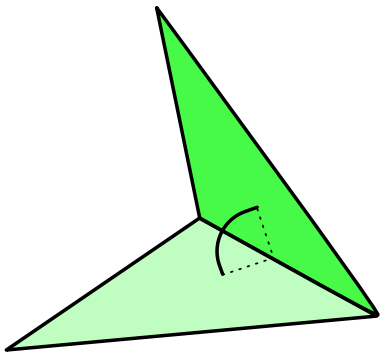


Figure 2.1. A *dihedral angle*, the angle formed between two faces of a tetrahedron. Each tetrahedron has six dihedral angles, one for each edge.

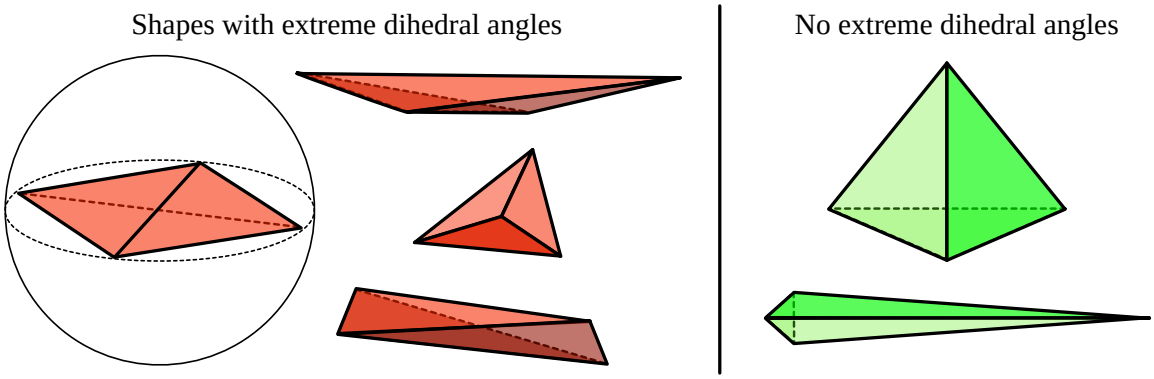


Figure 2.2. Left: tetrahedra with extreme dihedral angles (approaching 0° or 180°). Right: tetrahedra with no extreme dihedral angles including the *spire* (bottom), which can have an arbitrarily small solid angle at its tip.

gles are shown in the left half of Figure 2.2. Examples of tetrahedra with no small or large dihedral angles are shown on the right. There are some contexts, such as aerodynamics, where anisotropic tetrahedra with extreme angles are necessary and desirable. In Chapter 7, I describe how anisotropic tetrahedra can be evaluated in a manner similar to how isotropic tetrahedra are evaluated.

2.1 Quality measures

Because of the wide range of applications that employ tetrahedral meshes, it is impossible to capture with just one number the suitability of an element in all the situations where it could be used. Nevertheless, mesh improvement programs are expected to work well with meshes destined for many different applications. Therefore, most mesh improvement programs encapsulate the quality of a tetrahedron t as a single numerical *quality measure* $q(t)$, allowing users to choose q from among several functions. Many such quality measures are available [21; 47]. All of the mesh improvement operations I use can accommodate almost every measure in the literature. For uniformity of implementation, I standardize each measure so that a larger value of $q(t)$ indicates a better tetrahedron, with $q(t)$ positive if t has the correct topological orientation, zero if t is degenerate (having zero volume and all four vertices coplanar), and negative if t is *inverted*, meaning that there is a wrinkle in the fabric of the mesh. I assume that no input tetrahedra are inverted, and guarantee that no mesh improvement operation will invert one.

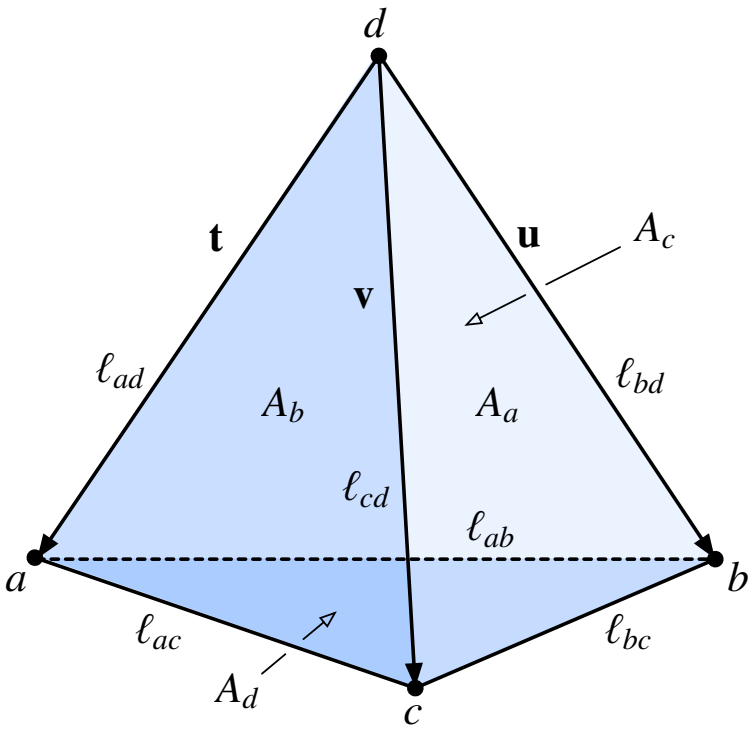


Figure 2.3. A tetrahedron. ℓ is edge length, A is face area, and $\mathbf{t}$, $\mathbf{u}$, and $\mathbf{v}$ are edge vectors. Formulae for computing the labeled quantities are listed in Table 2.2.

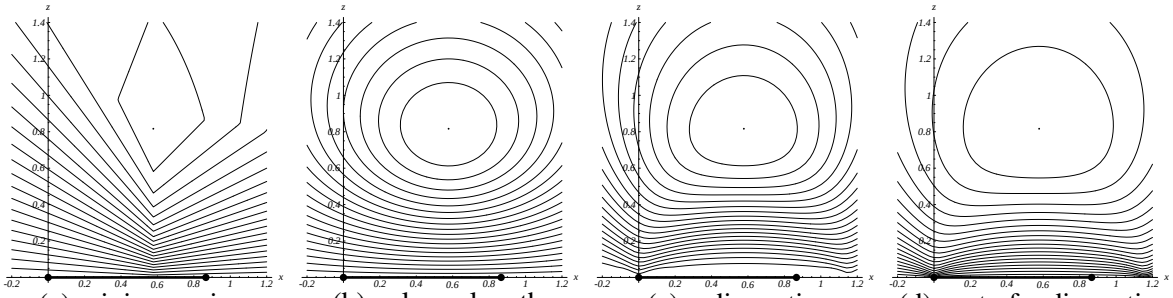
Besides describing the quality of a tetrahedron, quality measures serve as objective functions for mesh optimization. If a quality measure does a good job of evaluating a tetrahedron's shape, but behaves poorly when used for numerical optimization of the mesh, its usefulness for mesh improvement is compromised. I seek measures that do a good job of evaluating an element's fitness and also possess well-behaved, easy-to-compute gradients for numerical optimization.

From the vast pool of tetrahedron quality measures, I selected four to test with my implementation. All four measures take on their maximum values for an equilateral tetrahedron. Descriptions of the measures follow in Sections 2.1.1–2.1.4. Table 2.1 lists formulae for computing each measure and its gradient. (The gradients are needed for vertex smoothing; see Section 3.2.) Figure 2.3 illustrates the quantities used to compute the measures and Table 2.2 lists formulae, compiled by Shewchuk [48], for robustly computing their values. In Section 5.1, I empirically compare the performance of each quality measure as an objective function for mesh optimization.

Each of the measures includes the *signed volume* V of the tetrahedron in its numerator. Let a, b, c, and d be the vertices of a tetrahedron, and define the vectors $\mathbf{t} = a - d$, $\mathbf{u} = b - d$, and

Table 2.1. Formulae for tetrahedron quality measures and their gradients. Quantities are labeled in Figure 2.3 and formulae for their computation are given in Table 2.2. Where numbers are used as vertex subscripts, 1, 2, 3, and 4 correspond to a, b, c, and d, respectively.

measure	formula	formula for gradient
minimum sine	$\frac{3V}{2} \min_{1 \le k < l \le 4} \frac{\ell_{kl}}{A_k A_l}$	
	$\frac{3}{2} \min_{1 \le k < l \le 4} \frac{A_k A_l (\nabla V \ell_{kl} + V \nabla \ell_{kl}) - V \ell_{kl} (\nabla A_k A_l + A_k \nabla A_l)}{A_k^2 A_l^2}$	
volume-length	$\frac{6}{\sqrt{2}} \frac{V}{\ell_{\text{rms}}^3}$	$\frac{6}{\sqrt{2}} \left(\frac{\nabla V}{\ell_{\text{rms}}^3} - \frac{3V \nabla \ell_{\text{rms}}}{\ell_{\text{rms}}^4} \right)$
square root of radius ratio	$6\sqrt{3} \frac{V}{\sqrt{Z(A_a + A_b + A_c + A_d)}}$	$6\sqrt{3} \frac{\nabla V}{\sqrt{Z(A_a + A_b + A_c + A_d)}}$
	$-6\sqrt{3} V \frac{Z(\nabla A_a + \nabla A_b + \nabla A_c) + \nabla Z (A_a + A_b + A_c + A_d)}{2 [Z(A_a + A_b + A_c + A_d)]^{3/2}}$	



(a) minimum sine (b) volume-length (c) radius ratio (d) root of radius ratio

Figure 2.4. Isocontour plots of quality functions for the $y = 0$ cross-section of a tetrahedron with three vertices fixed at $(0, 0, 0)$, $(\sqrt{3}/2, 1/2, 0)$, $(\sqrt{3}/2, -1/2, 0)$ and the fourth vertex allowed to vary freely. Adapted from Shewchuk [48]. The minimum sine measure (a) is the only function which is nonsmooth within a single tetrahedron. The volume-length ratio (b) displays good gradient behavior, even for degenerate tetrahedra. The radius ratio (c) has a zero gradient for all degenerate tetrahedra; this can be remedied by instead using its square root (d).

Table 2.2. Formulae from Shewchuk [48] for quantities needed to compute quality measures and their gradients with respect to a vertex d being smoothed. Let $\mathbf{t} = a - d$, $\mathbf{u} = b - d$, and $\mathbf{v} = c - d$. See Figure 2.3 for a visual depiction of most of these quantities. V is the signed volume of a tetrahedron, ℓ_{rms} is its root-mean-squared edge length, and Z is a subexpression used to compute its circumradius.

$f(d)$	formula	$\nabla f(d)$
ℓ_{ad}	$\lvert\mathbf{t}\rvert$	$-\dfrac{1}{\ell_{ad}}\mathbf{t}$
ℓ_{bd}	$\lvert\mathbf{u}\rvert$	$-\dfrac{1}{\ell_{bd}}\mathbf{u}$
ℓ_{cd}	$\lvert\mathbf{v}\rvert$	$-\dfrac{1}{\ell_{cd}}\mathbf{v}$
ℓ_{ab}	$\lvert a - b\rvert$	0
ℓ_{bc}	$\lvert b - c\rvert$	0
ℓ_{ac}	$\lvert c - a\rvert$	0
ℓ_{rms}	$\sqrt{\dfrac{1}{6}\left(\ell_{ad}^2 + \ell_{bd}^2 + \ell_{cd}^2 + \ell_{ab}^2 + \ell_{bc}^2 + \ell_{ac}^2\right)}$	$-\dfrac{1}{6\ell_{\text{rms}}}(\mathbf{t} + \mathbf{u} + \mathbf{v})$
A_a	$\dfrac{\lvert\mathbf{u} \times \mathbf{v}\rvert}{2}$	$\dfrac{1}{4A_a}[(\mathbf{v} \cdot (\mathbf{u} - \mathbf{v}))\mathbf{u} - (\mathbf{u} \cdot (\mathbf{u} - \mathbf{v}))\mathbf{v}]$
A_b	$\dfrac{\lvert\mathbf{v} \times \mathbf{t}\rvert}{2}$	$\dfrac{1}{4A_b}[(\mathbf{t} \cdot (\mathbf{v} - \mathbf{t}))\mathbf{v} - (\mathbf{v} \cdot (\mathbf{v} - \mathbf{t}))\mathbf{t}]$
A_c	$\dfrac{\lvert\mathbf{t} \times \mathbf{u}\rvert}{2}$	$\dfrac{1}{4A_c}[(\mathbf{u} \cdot (\mathbf{t} - \mathbf{u}))\mathbf{t} - (\mathbf{t} \cdot (\mathbf{t} - \mathbf{u}))\mathbf{u}]$
A_d	$\dfrac{\lvert(\mathbf{u} - \mathbf{v}) \times (\mathbf{t} - \mathbf{v})\rvert}{2} = \dfrac{\lvert(b - c) \times (a - c)\rvert}{2}$	0
V	$\dfrac{\det[\mathbf{t}\ \mathbf{u}\ \mathbf{v}]}{6}$	$\dfrac{1}{6}(\mathbf{u} - \mathbf{v}) \times (\mathbf{t} - \mathbf{v}) = \dfrac{1}{6}(b - c) \times (a - c)$
Z	$\left\lvert\lvert\mathbf{t}\rvert^2\mathbf{u} \times \mathbf{v} + \lvert\mathbf{u}\rvert^2\mathbf{v} \times \mathbf{t} + \lvert\mathbf{v}\rvert^2\mathbf{t} \times \mathbf{u}\right\rvert$	
	$(2/Z)\left[(\lvert\mathbf{u}\rvert^2\mathbf{v} \cdot \mathbf{t} - \lvert\mathbf{v}\rvert^2\mathbf{t} \cdot \mathbf{u})(\lvert\mathbf{u}\rvert^2 - \lvert\mathbf{v}\rvert^2) - (\lvert\mathbf{u}\rvert^2\lvert\mathbf{v}\rvert^2 + \lvert\mathbf{t}\rvert^2\mathbf{u} \cdot \mathbf{v})\lvert\mathbf{u} - \mathbf{v}\rvert^2\right]\mathbf{t} +$ $(2/Z)\left[(\lvert\mathbf{v}\rvert^2\mathbf{t} \cdot \mathbf{u} - \lvert\mathbf{t}\rvert^2\mathbf{u} \cdot \mathbf{v})(\lvert\mathbf{v}\rvert^2 - \lvert\mathbf{t}\rvert^2) - (\lvert\mathbf{v}\rvert^2\lvert\mathbf{t}\rvert^2 + \lvert\mathbf{u}\rvert^2\mathbf{v} \cdot \mathbf{t})\lvert\mathbf{v} - \mathbf{t}\rvert^2\right]\mathbf{u} +$ $(2/Z)\left[(\lvert\mathbf{t}\rvert^2\mathbf{u} \cdot \mathbf{v} - \lvert\mathbf{u}\rvert^2\mathbf{v} \cdot \mathbf{t})(\lvert\mathbf{t}\rvert^2 - \lvert\mathbf{u}\rvert^2) - (\lvert\mathbf{t}\rvert^2\lvert\mathbf{u}\rvert^2 + \lvert\mathbf{v}\rvert^2\mathbf{t} \cdot \mathbf{u})\lvert\mathbf{t} - \mathbf{u}\rvert^2\right]\mathbf{v}$	

$\mathbf{v} = c - d$. The signed volume of the tetrahedron is the determinant of the 3×3 matrix $[\mathbf{t}\ \mathbf{u}\ \mathbf{v}]/\mathbf{6}$. This volume is positive if the vertices are oriented as depicted in Figure 2.3, zero if the vertices are coplanar, and negative if the vertices are oriented as in the figure's mirror image. (The orientation of a tetrahedron is the three-dimensional analog of listing the vertices of a triangle in clockwise versus counterclockwise order.) The sign of a tetrahedron's volume is inherited by all its quality measures, and causes the quality measures to smoothly become negative if one of the vertices is smoothed so that the tetrahedron becomes inverted.

2.1.1 Minimum sine measure

The *minimum sine measure* of a tetrahedron is the minimum among the sines of its six dihedral angles. Because the sines of 0° and 180° are both zero, this measure penalizes both small angles and large, and supports both matrix conditioning and discretization accuracy. The minimum sine measure reaches a maximum value of $\arcsin(2\sqrt{2}/3)$ for an equilateral tetrahedron. Contrary to what you might expect, no trigonometric calculations are needed to compute it; see Table 2.1.

As an objective function for mesh optimization, Freitag and Ollivier-Gooch [23] find the minimum sine measure to be the most effective measure they consider. Nevertheless, it has some minor potential drawbacks. It requires computation of all six dihedral angles' sines, making it and its gradient somewhat costly to compute. It is a *nonsmooth* function of the vertex positions (see Figure 2.4a), because as the vertices move, the identity of the smallest sine (and the gradient of the measure) can change abruptly. Fortunately, our smoothing algorithm (Section 3.2) can cope with this nonsmoothness. In contrast to many popular quality measures, the minimum sine measure does not harshly penalize spire tetrahedra (see lower right of Figure 2.2). Spire tetrahedra may be bad or harmless, depending on the application. See Section 2.2 for further discussion.

2.1.2 Biased minimum sine measure

Large dihedral angles are arguably more harmful than small dihedral angles, because they hurt the accuracy (rather than the speed) of the finite element method. Fortunately, I have found that in practice, large dihedral angles can be attacked more aggressively than small ones, without much

sacrifice in improving the latter. The *biased minimum sine measure* is like the minimum sine measure, but if a dihedral angle is obtuse, I multiply its sine by 0.7 (before choosing the minimum), thereby penalizing obtuse angles more than acute angles.

2.1.3 Volume-length measure

The *volume-length measure*, suggested by Parthasarathy, Graichen, and Hathaway [40] and denoted V/ℓ_{rms}^3, is the signed volume of a tetrahedron divided by the cube of its root-mean-squared edge length. Intuitively, it prefers "fat" tetrahedra. Tetrahedra with extreme dihedral angles have little volume relative to their edge length, and so are penalized. I multiply the measure by $6\sqrt{2}$ so that the highest quality is one, the measure of an equilateral tetrahedron.

As an objective function for numerical optimization, the volume-length measure is the most attractive of the four measures I consider. It is easy and fast to compute, as is its gradient. It is a smooth function of its vertex positions, with well-behaved gradients and nearly spherical isocontours (see Figure 2.4b). It punishes spire tetrahedra (see Section 2.2) much more severely than the minimum sine measure. For a detailed discussion of its relative strengths, see Section 5.1.

2.1.4 Radius ratio (and its square root)

The *radius ratio*, suggested by Cavendish, Field, and Frey [11], is the radius of a tetrahedron's inscribed sphere divided by the radius of its circumscribing sphere. Like the volume-length measure, it intuitively prefers "fat" tetrahedra. Like the volume-length measure, the radius ratio penalizes spire tetrahedra. I multiply the measure by 3 so that the highest quality is one, the measure of an equilateral tetrahedron.

This measure is inexplicably popular in the mesh generation literature, but it has nothing to recommend it over the volume-length measure. It is costly to compute, and its gradient is especially so. The radius ratio is a smooth function of its vertex positions, but its gradient is badly behaved in a way that tends to thwart optimization-based vertex smoothing (described in Section 3.2). For one thing, its gradient (with respect to the vertex positions) is zero for a degenerate tetrahedron. (Observe the open space below the bottommost isocontour in Figure 2.4c.) I fix this problem by

using the square root of the radius ratio as a quality measure (Figure 2.4d), instead of the radius ratio itself.

The square root of the radius ratio is better as an objective function, but its gradient is still unreliable because the circumradius of a tetrahedron is unstable (and can be very large) when the four vertices are nearly cocircular. Occasionally, the gradient may point optimization-based vertex smoothing in a counterproductive direction. I find that in practice, the volume-length objective function outperforms the root radius ratio objective function even for optimizing the radius ratio. So I will revisit it only once—in Section 5.1 where I demonstrate this fact.

2.2 Spire tetrahedra

Spire tetrahedra are skinny, needle-shaped tetrahedra that may have excellent dihedral angles but also arbitrarily small solid angles at their tips. An example of a spire tetrahedron is shown at the lower right of Figure 2.2. Quality measures that consider only dihedral angles—such as the minimum sine and biased minimum sine measures—penalize spire tetrahedra only lightly, whereas the volume-length measure and the radius ratio penalize them severely. The choice of quality measure depends in part on whether it is important to eliminate spires, which are harmless to some applications and harmful to others.

The argument in favor of permitting spires is that, because they have good dihedral angles, they do not cause discretization errors or harm matrix conditioning [47]. Moreover, a spire is indispensable at the tip of a needle-shaped domain.

A mild argument against spires is that they are less efficient than equilateral tetrahedra at providing the most accuracy for the fewest elements. The number of tetrahedra that fill a domain is inversely proportional to their volumes, and the accuracy of an element for interpolation is inversely proportional to the length of its longest edge. A spire can have arbitrarily little volume relative to its longest edge. In practice, spires are a minority in meshes optimized with the minimum sine objective function, so only a little efficiency is lost.

Spires are substantially more harmful in time-dependent simulations with explicit time integra-

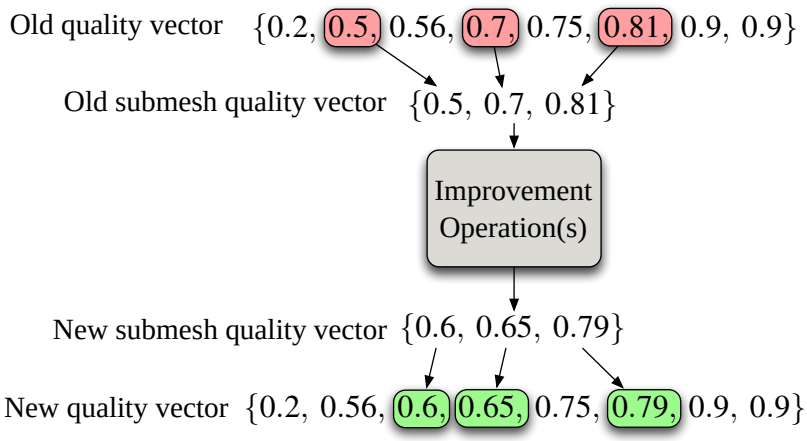


Figure 2.5. A global quality vector is lexicographically improved by a change to a subset of its elements. Because any lexicographic improvement to a subset of element qualities constitutes a global improvement, we need never compute the global quality vector.

tion. The maximum timestep that the simulation can take without sacrificing the stability of the integrator is proportional to the length of the shortest edge, as dictated by the Courant–Friehrichs–Lewy condition [15].

To users who believe it is important not to have spires in their meshes, I recommend the volume-length measure. Users who are tolerant of spires should also consider the minimum sine and biased minimum sine measures.

2.3 The quality of the whole mesh

Recall that most mesh improvement algorithms use hill-climbing optimization, in which an objective function maps each mesh to a numerical value. Quality measures are numerical evaluations of individual tetrahedra, but how do we evaluate a whole mesh?

My objective function for mesh quality is driven by the observation that the *worst* elements are overwhelmingly influential on the usefulness of a mesh. A single bad tetrahedron can ruin a simulation: one large dihedral angle can induce an arbitrarily large, incorrect strain in the simulation of a mechanical system.

How can we construct a measure of the quality of a mesh that emphasizes the worst elements?

The solution I adopt is to optimize a mesh's *quality vector*: a vector listing the quality of each tetrahedron, ordered from worst to best. Two meshes' quality vectors are compared *lexicographically*: the first elements are compared, then the second, and so forth, much like comparing words alphabetically. For example, a quality vector of {0.1, 0.5, 0.5} is worse than a quality vector of {0.2, 0.2, 0.2}, which is itself worse than {0.2, 0.2, 0.3}. An improvement in the second-worst tetrahedron improves the overall objective function even if the worst tetrahedron is not changed. A nice property of the quality vector is that if an operation replaces a subset of a mesh's tetrahedra with new ones, I only need to compare the quality vectors of the submeshes constituting the changed tetrahedra (before and after the operation). If the submesh improves, the quality vector of the whole mesh improves. Figure 2.5 illustrates an example in which a local transformation replaces one submesh with another having a better quality vector, thereby improving the whole mesh.

Chapter 3

Mesh Improvement Operations

Here I describe the mesh improvement operations that form the core of my mesh improvement algorithm. Simultaneously, I survey much of the existing work in tetrahedral mesh improvement. Designing mesh improvement operations is tricky. The task of improving unstructured tetrahedral meshes offers a buffet of computationally intractable problems, both combinatorial and numerical. I have chosen a set of operations that can be efficiently implemented, in an effort to discover which are the most effective in practice.

I divide the operations into three categories: vertex smoothing, topological operations that preserve mesh vertices, and topological operations that change the number of vertices in the mesh—namely, edge contraction and a new form of vertex insertion. In practice, edge contraction and vertex insertion are more successful when bundled together with other improvement operations, so I describe both simple and *composite* forms of these operations.

3.1 Topological operations

Topological operations change a mesh by removing some elements and replacing them with new elements that occupy exactly the same space. Topological operations naturally rely on combinatorial optimization to find the best new configuration, because the set of topological configurations is discrete.

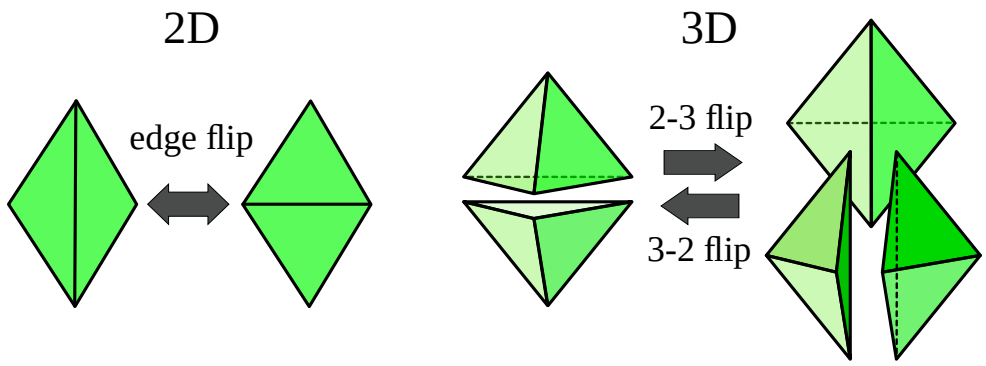


Figure 3.1. The two-dimensional edge flip and its three-dimensional analogues, the 2-3 and 3-2 flips.

Topological transformations are usually *local*, meaning that only a small number of elements are changed, removed, or introduced by a single operation. The simplest topological transformations on tetrahedral meshes are the 2-3 and 3-2 flips, which are the natural extension to three dimensions of the familiar edge flip in two-dimensional triangulations. The numbers denote the number of tetrahedra removed and created, respectively. The 2-3 flip takes two tetrahedra that meet at a face and deletes them, replacing them with three tetrahedra meeting at a new edge. The 3-2 flip does the opposite, removing three tetrahedra that meet at an edge and replacing them with two. The two operations are illustrated in Figure 3.1.

Both of these operations generalize to larger sets of tetrahedra. *Multi-face removal* extends the 2-3 flip to remove more than a single face, whereas *edge removal* extends the 3-2 flip to replace an edge that is shared by any number of tetrahedra.

The 2-2 flip is a special flip performed on two adjacent tetrahedra with boundary faces. If boundary faces of the adjacent tetrahedra are coplanar, than an interior face between them can be deleted and replaced with a crossing face, as depicted in Figure 3.2. It is a special case of both edge removal and face removal.

3.1.1 Edge removal

Proposed by Brière de l'Isle and George [8], *edge removal* is a topological transformation that removes a single edge from the mesh, along with all the tetrahedra that include it. (The name

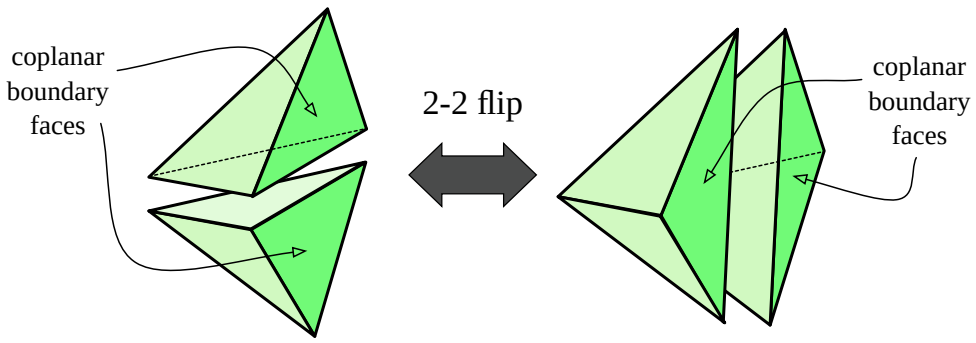


Figure 3.2. The 2-2 flip, which flips a face common to two tetrahedra that lie on the boundary.

is slightly misleading, because edge removal can create new edges while removing the old one. Freitag and Ollivier-Gooch refer to edge removal as "edge swapping," but I use the earlier name.) The simplest edge removal is the 3-2 flip, but it also includes other transformations that remove edges shared by any number of tetrahedra. In general, edge removal replaces m tetrahedra with $2m-4$ ($2m-2$ if the edge is on the mesh boundary); Figure 3.3 illustrates replacing seven tetrahedra with ten. De Cougny and Shephard [18] and Freitag and Ollivier-Gooch [23] have shown dramatic evidence for its effectiveness, especially in combination with other mesh improvement operations.

Let ab be an edge in the interior of the mesh with vertices a and b. Let I be the set of tetrahedra that include ab. If ab lies on the boundary of the mesh, and the tetrahedra in I do not subtend an angle of exactly 180° around ab, then ab cannot be removed without changing the shape of the domain; but if the angle is 180° (or 360°, meaning ab is in the mesh interior, as in Figure 3.3), it might be possible to remove ab. Each tetrahedron in I has an edge opposite ab. Let R be the set of these edges. (R is known as the *link* of ab.) R forms a (non-planar) polygon in three-dimensional space, as illustrated. An edge removal transformation constructs a triangulation T of R, and creates a set of new tetrahedra $J = \bigcup_{t \in T} \{\text{conv}(\{a\} \cup t), \text{conv}(\{b\} \cup t)\}$ (where $\text{conv}(S)$ is the convex hull of the point set S) which replace the tetrahedra in I.

The chief algorithmic problem is to find the triangulation T of R that maximizes the quality of the worst tetrahedron in J. Freitag and Ollivier-Gooch explicitly enumerate all triangulations of rings of up to seven vertices and choose the best among all of them. They do not consider larger rings, which they correctly claim are less likely to yield improvement. In contrast, I solve this

problem with a dynamic programming algorithm of Klincsek [31], which was invented long before anyone studied edge removal and solves a general class of problems in optimal triangulation. The algorithm runs in $O(m^3)$ time, but m is never large enough for its speed to be an impairment. I find experimentally that edge removal rarely succeeds when m is larger than 10, and that it is most often successful when $m = 4$.

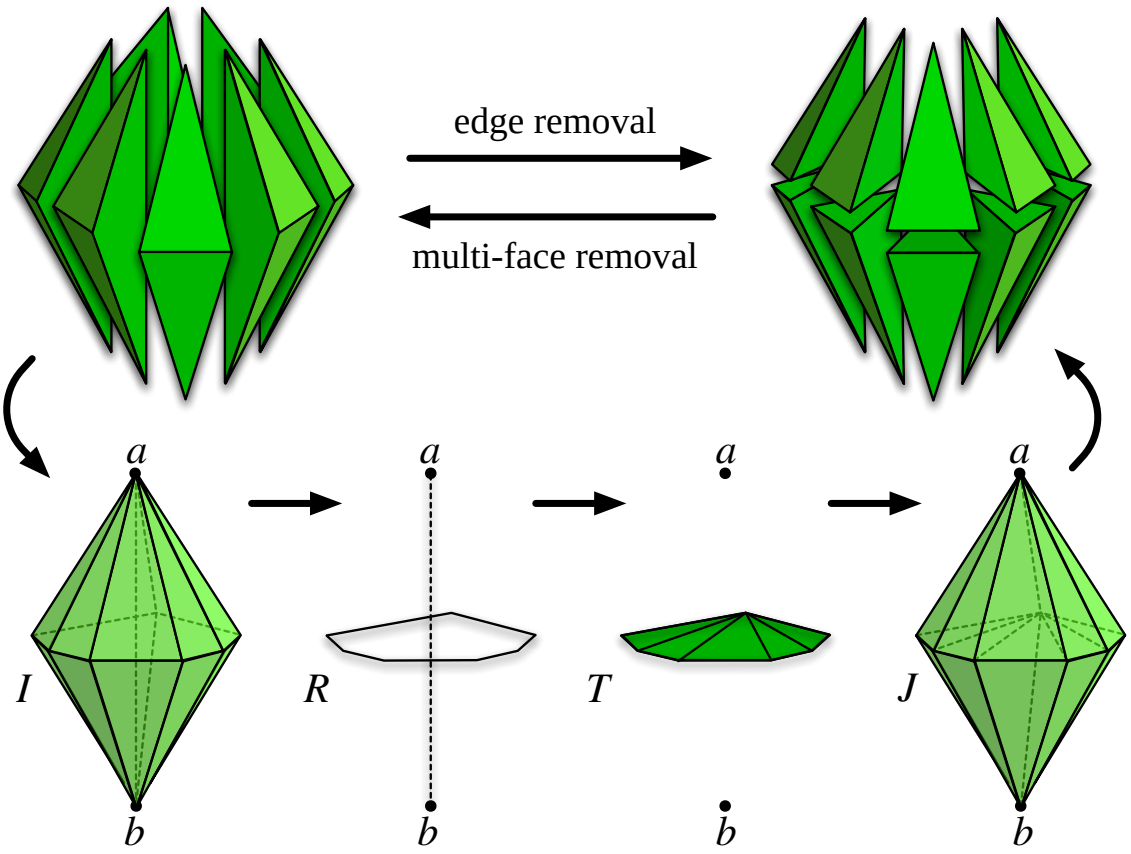


Figure 3.3. Edge removal and its inverse, multi-face removal. Figure adapted from Shewchuk [46].

3.1.2 Multi-face removal

Multi-face removal is the inverse of edge removal, and its simplest form is the 2-3 flip. An m-face removal replaces $2m$ tetrahedra with $m + 2$. It has been neglected in the literature; as far as I know, it has appeared only in an unpublished manuscript of de Cougny and Shephard [18], who present evidence that multi-face removal is effective for mesh improvement, though I find that it offers only marginal improvement if edge removal has already been implemented (see Section 5.3).

Multi-face removal, like edge removal, revolves around two chosen vertices a and b. Given a mesh, say that a triangular face f is *sandwiched* between a and b if the two tetrahedra that include f are $\text{conv}(\{a\} \cup f)$ and $\text{conv}(\{b\} \cup f)$. For example, in Figure 3.3, the faces of T are sandwiched between a and b in the mesh J. An m-face removal operation singles out m of those sandwiched faces, and replaces the tetrahedra that adjoin them, as illustrated. (An m-face removal actually removes $3m - 2$ faces, but only m of them are sandwiched between a and b.)

My implementation uses multi-face removal by singling out a particular internal face f it would like to remove. Let a and b be the apex vertices of the two tetrahedra adjoining f. The optimal multi-face removal operation does not necessarily remove *all* the faces sandwiched between a and b. I use the algorithm of Shewchuk [46] to find the optimal multi-face removal operation for f (and to determine whether *any* multi-face removal operation can remove f without creating inverted tetrahedra), in time linear in the number of sandwiched faces. I perform the optimal operation only if the worst new tetrahedron has better quality than the worst deleted tetrahedron.

3.2 Vertex smoothing

Smoothing moves vertices without changing the connectivity of the mesh. The most famous smoothing technique is *Laplacian smoothing*, which moves a vertex to the average of its connected neighbors [27]. Figure 3.4 shows Laplacian smoothing in two dimensions. Typically, Laplacian smoothing is applied to each mesh vertex in sequence, and several passes of smoothing are done, where each "pass" moves every vertex once. Laplacian smoothing is popular and somewhat effective for triangular meshes, but for tetrahedral meshes it is much less reliable and often produces poor tetrahedra.

Better smoothing algorithms are based on numerical optimization [41; 9]. Early algorithms define a smooth objective function that summarizes the quality of a group of elements (e.g. the sum of squares of the qualities of all the tetrahedra adjoining a vertex), and use a numerical optimization algorithm such as steepest descent or Newton's method to move a vertex to the optimal location. Freitag, Jones, and Plassman [22] propose a more sophisticated *nonsmooth* optimization algorithm, which makes it possible to optimize the worst tetrahedron in a group—for instance, to maximize

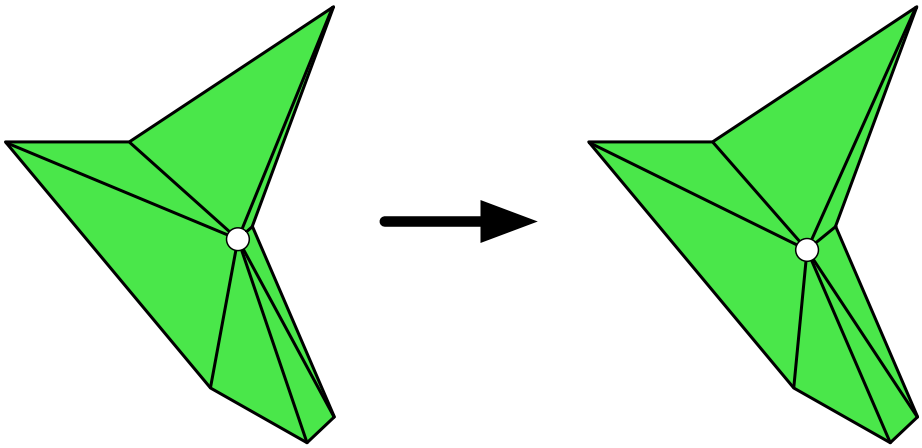


Figure 3.4. An example of Laplacian smoothing of the white vertex. The vertex is moved to the average of its neighbors.

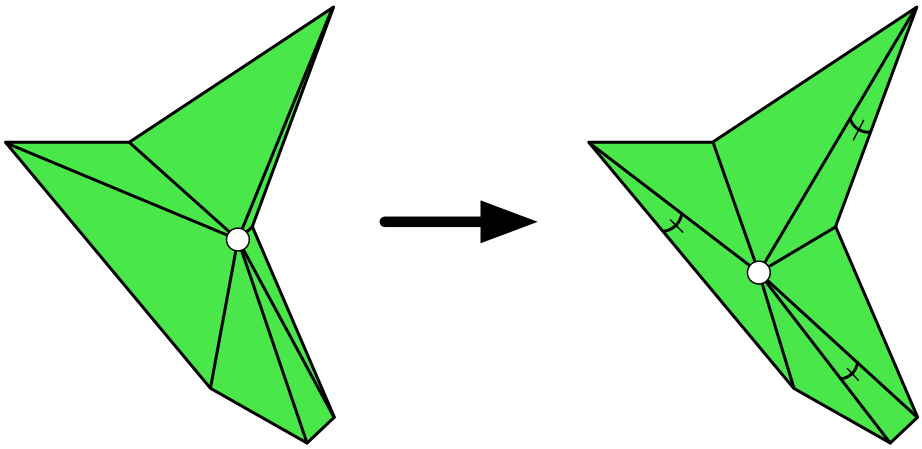


Figure 3.5. An example of nonsmooth optimization-based smoothing of the white vertex. The objective is to maximize the minimum angle among all incident triangles. The three marked angles are equal and are smaller than the other angles depicted. When smoothing is finished, any movement of the vertex would cause some minimum angle to become smaller. Figure adapted from Shewchuk [48].

the minimum dihedral angle among the tetrahedra that share a specified vertex. A nonsmooth optimization algorithm is needed because the objective function—the minimum quality among several tetrahedra—is *not* a smooth function of the vertex coordinates; the gradient of this function is discontinuous wherever the identity of the worst tetrahedron in the group changes. Freitag and Ollivier-Gooch [23] had great success with this algorithm, and I use it (in my own implementation) essentially unchanged. An example of the result of nonsmooth optimization-based smoothing is shown in Figure 3.5.

3.2.1 Computing the search direction

A strategy for nonsmooth optimization-based smoothing is to select a search direction **d** to move the vertex, find the optimal position for the vertex constrained along **d**, then compute a new search direction and iterate until convergence. Computing this search direction—the direction that the vertex will move in an effort to improve the quality of the mesh—is perhaps the most interesting step in Freitag, Jones, and Plassman's smoothing algorithm. I describe it here unchanged.

In nonsmooth optimization of the position of a vertex v, there is not a single smooth objective function $f(v)$, but a collection of them—one or more for each element incident to v. For example, there might be one function for each tetrahedron, or one function for each dihedral angle in each tetrahedron. Only the functions that are worst for the current vertex position are relevant in selecting the search direction.

Let $f_1(v)$, $f_2(v)$, ..., $f_m(v)$ be a set of smooth quality functions that vary with the position of vertex v. The objective function that we maximize is $f(v) = \min_i f_i(v)$. Let the *active set* A be the set of quality functions that are minimal for a position of v, i.e.

$$A = \{f_i : f_i(v) = f(v)\}. \tag{3.1}$$

A can contain as few as one function, but generally contains two or more if v's position is optimal. In practice, it is rare for v to fall at a point where two or more quality functions attain exactly the same value, so I place a function $f_i(v)$ in the active set if it is within 3% of $f(v)$.

Each iteration of smoothing tries to improve all of the quality functions in the active set, but each function has its own gradient $\nabla f_i(v)$ that wants to pull the vertex v in a different direction

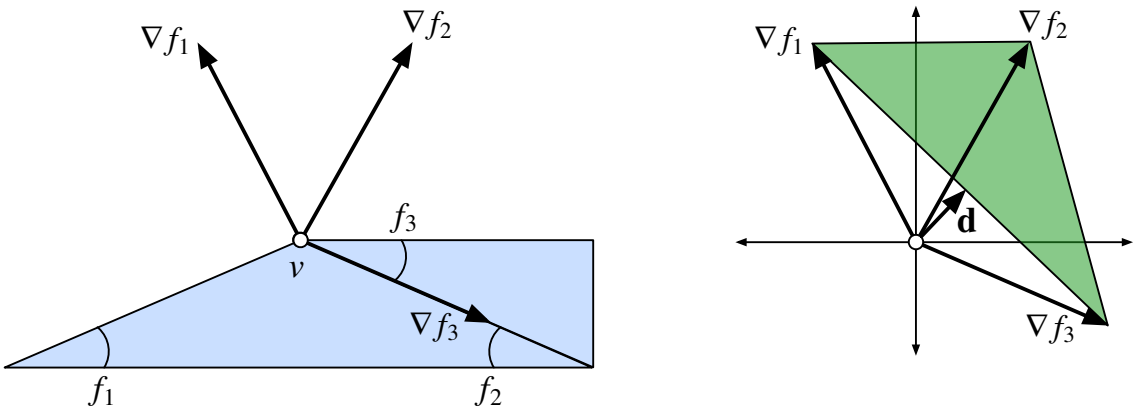


Figure 3.6. A two-dimensional example of search direction computation. Here, the active set $A = \{f_1, f_2, f_3\}$. The search direction $\mathbf{d}$ is the point closest to the origin on the convex hull of the gradients of the functions in the active set. Figure adapted from Shewchuk [48].

than the others. Figure 3.6 illustrates a two-dimensional example in which the objective function is the smallest angle. At left are two triangles incident to a vertex v, which is being smoothed. The active set is $A = \{f_1, f_2, f_3\}$, because all three angles are tied for the smallest. The smoothing algorithm must therefore consider each of their gradients when selecting a search direction $\mathbf{d}$. We want to choose the direction that best improves the slowest-improving function in the active set. This direction is the point nearest the origin on the convex hull of the three gradients ($\mathbf{d}$ in Figure 3.6). When this convex hull contains the origin, there is nowhere the vertex can move that will improve all of the quality functions, and the smoothing algorithm terminates.

Formulae for the quality functions f_i and their gradients appear in Table 2.1.

3.2.2 Nonsmooth line search

A *line search* algorithm determines how far to move the vertex in the search direction. When the objective function is smooth, line search ends where the derivative of the function is zero. When the objective function is nonsmooth, the line search will likely end where the active set changes, and hence where the derivative of the objective function is undefined. Therefore, root-finding line search methods are ineffective. Instead, Freitag, Jones, and Plassman suggest approximating each function as a line and jumping to the point where intersections of these lines would cause the active set to change. Their line search iterates this procedure until the steps it takes are too small.

3.2.3 Smoothing boundary vertices

Most mesh improvement software, including that of Freitag and Ollivier-Gooch, regards the boundary configuration of the mesh as "untouchable": vertices on the boundary cannot be smoothed, and the connectivity between them cannot be changed. This constraint is real in some circumstances—for example, when the boundary must be matched face-for-face with another mesh—but in many applications it is unnecessarily limiting.

If a boundary contains flat regions or straight ridges (as is often the case with mechanical models), vertices within these regions are free to move, subject to constraints, without altering the domain shape at all. Alternatively, if a curved model is discretized with a linear tetrahedral mesh, then the mesh boundary shape necessarily is erroneous, as it is only a piecewise linear approximation of the true boundary. If we can improve the mesh by moving boundary vertices without significantly increasing the surface error, then why not do so? A bad surface triangulation makes a good volume triangulation impossible. Considerations like these motivate me to permit the movement of boundary vertices in my implementation of vertex smoothing.

A user of my mesh improvement software can choose between two methods of boundary vertex smoothing (or neither). In the first, *constrained vertex smoothing*, I smooth vertices that lie in flat areas or on ridges without changing the domain shape. In the second, *quadric smoothing*, I permit all surface vertices to move, but they are encouraged to move along the (perhaps unknown, perhaps curved) original surface, and discouraged from making noticeable changes to the shape of the domain. I control domain shape error by balancing tetrahedron quality against a quadric error measured at each vertex.

3.2.4 Constrained vertex smoothing

Constrained vertex smoothing is best for models that are composed mostly of flat surfaces. I identify three types of boundary vertex on such models. (Examples of each type of vertex are shown in Figure 3.7.)

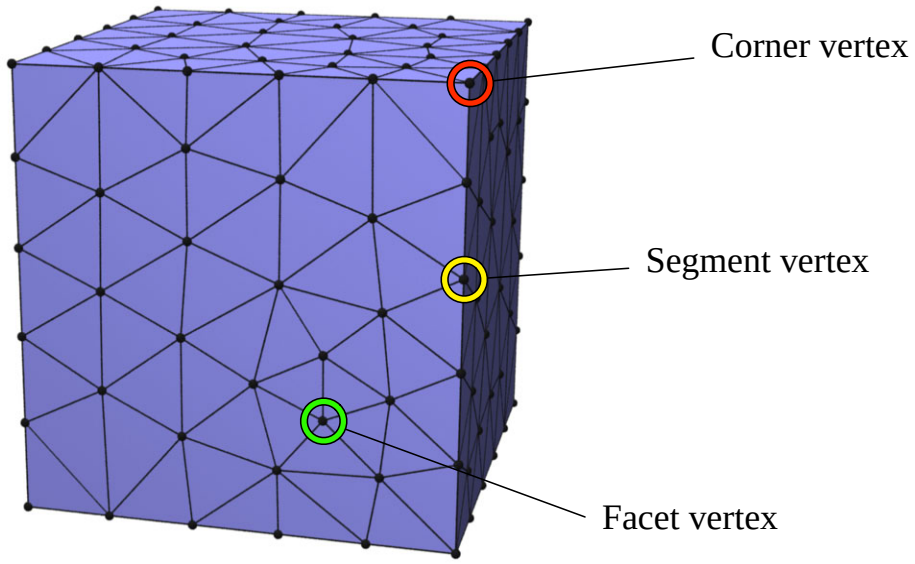


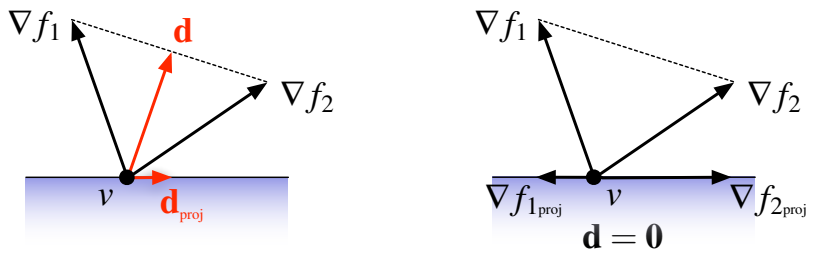
Figure 3.7. In constrained vertex smoothing, each boundary vertex has one of three types.

- *Facet* vertices lie in a flat region of the boundary. All the boundary faces adjoining a facet vertex are in a common plane. Facet vertices are free to move on this plane without changing the domain shape.

- *Segment* vertices lie on a straight ridge of the boundary. The faces incident to a segment vertex are divided into two sets, with all faces in a set coplanar. Segment vertices can move along the line that is the intersection of the two common planes.

- *Corner* vertices lie on a corner of the boundary. At least three faces incident to a corner vertex lie in distinct planes. Corner vertices cannot move without changing the domain shape.

The optimization-based smoothing algorithm must be modified to accommodate constrained smoothing of facet and segment vertices. In particular, the search direction must be constrained to a vertex's segment or facet. One obvious way to accomplish this is to project the search direction, computed as described in Section 3.2.1, onto the constraining facet or segment, but this projection will not produce the correct result. Consider the segment vertex v with active set $A = \{f_1, f_2\}$ shown in Figure 3.8(a). If the search direction is computed first and then projected onto the constraining segment, the projected search direction $\mathbf{d}_{\text{proj}}$ will improve f_2 but worsen f_1. The right way to compute the search direction is to first project the gradients ∇f_1 and ∇f_2 onto the constraining



(a) wrong (b) right

Figure 3.8. The wrong (a) and right (b) way to compute the search direction **d** for a constrained vertex. In (a), projecting **d** onto the constraining segment yields a search direction that improves only f_2 while worsening f_1. In (b), first projecting the gradients ∇f_1 and ∇f_2 onto the segment yields the correct result: no search direction on the segment can improve both functions. Figure adapted from Shewchuk [46].

segment, and then compute **d** as the point nearest the origin in the convex hull of the projected gradients. In Figure 3.8(b), that point is the origin.

3.2.5 Quadric smoothing

Constrained vertex smoothing works well for models that are mostly flat, but what about models that are discretizations of curved surfaces? Ideally, we would smooth vertices along the surfaces that the mesh approximates. Unfortunately, a mesh improvement program rarely has access to the original surface representation (e.g., splines, constructive solids) that was used to generate the mesh. Quadric smoothing is a attempt to smooth vertices while respecting an unknown domain shape.

After implementing quadric smoothing, I discovered a fringe benefit of giving boundary vertices the flexibility to move. Often, a small movement on the surface enables topological improvements of boundary tetrahedra that would not otherwise have been possible, which in turn permit the perturbed surface vertices to move back near their original locations, while yielding a mesh of much higher quality than I could obtain when these vertices were immobile.

In 1997, Garland and Heckbert proposed a measure of shape error in surface meshes they dub the *quadric error* [26; 25]. The quadric error assigns an error to a moved vertex based on how far that vertex has moved from the planes induced by the original triangular faces that adjoined it. More precisely, let P be the set of planes that are the affine hulls of the boundary faces incident to a vertex

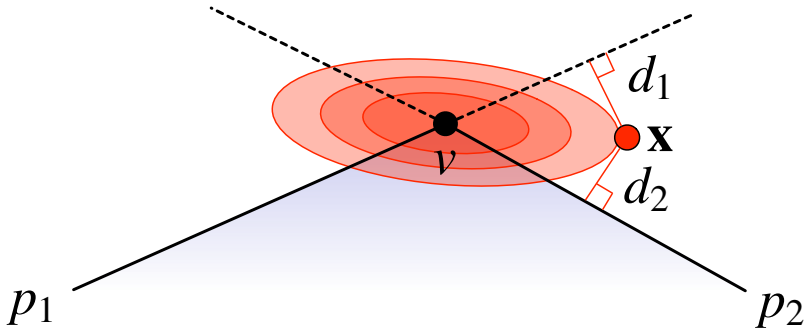


Figure 3.9. A two-dimensional view of quadric error. Red ellipses are the isosurfaces of the quadric error for vertex v, which is computed as the sum of the squares of the distances d_i of a query point $\mathbf{x}$ from the planes p_i defined by all faces incident to v.

v. The quadric error for a point $\mathbf{x}$ relative to v is

$$Q_v(\mathbf{x}) = \sum_{i \in P} d_i(\mathbf{x})^2, \tag{3.2}$$

where $d_i(\mathbf{x})$ is the distance of $\mathbf{x}$ from the ith plane. Each face incident to v induces a plane $\{\mathbf{x} : \mathbf{n}_i^\mathsf{T}\mathbf{x} + \delta_i = 0\}$, where $\mathbf{n}_i$ is a vector normal to the plane, and δ_i is a scalar offset. We have

$$d_i(\mathbf{x})^2 = (\mathbf{n}_i^\mathsf{T}\mathbf{x} + \delta_i)^2 = \mathbf{x}^\mathsf{T}(\mathbf{n}_i\mathbf{n}_i^\mathsf{T})\mathbf{x} + 2(\delta_i\mathbf{n}_i^\mathsf{T})\mathbf{x} + \delta_i^2. \tag{3.3}$$

Substituting this identity into the quadric error gives a formula wherein the summations are independent of $\mathbf{x}$, namely

$$Q_v(\mathbf{x}) = \sum_{i \in P} d_i(\mathbf{x})^2 = \mathbf{x}^\mathsf{T}\left(\sum_{i \in P} \mathbf{n}_i\mathbf{n}_i^\mathsf{T}\right)\mathbf{x} + 2\left(\sum_{i \in P} \delta_i\mathbf{n}_i^\mathsf{T}\right)\mathbf{x} + \sum_{i \in P} \delta_i^2. \tag{3.4}$$

We can precompute the summations and then quickly evaluate Q_v for many different values of $\mathbf{x}$. Garland and Heckbert associate one such function with each mesh vertex. They call the functions *quadrics* because their isosurfaces, $Q_v(\mathbf{x}) = c$ for some constant c, are quadric surfaces—ellipsoids and planes. A two-dimensional depiction of several such isosurfaces for one vertex is shown in Figure 3.9. If the vertex moves along the surface, not much quadric error is incurred. However, if the vertex moves perpendicular to the surface, quadric error accrues rapidly. So, if we limit quadric error, we limit deviation from the original surface, while still permitting some freedom of movement along the surface.

The original application for quadric error was fast, high-quality surface simplification. In Garland and Heckbert's quadric-based simplification algorithm, a triangle mesh is simplified by successively contracting edges. The two endpoints of an edge are replaced by a new vertex that inherits

the quadrics of both endpoints by adding them together. This new vertex is placed at the location that minimizes its total quadric error.

I repurpose quadric error for smoothing boundary vertices in tetrahedral meshes. I assign a quadric error function to each boundary vertex of a tetrahedral mesh. By trading off this vertex quality against the qualities of the adjoining tetrahedra, I can smooth boundary vertices while controlling how much error is introduced into the domain shape. The quality of a tetrahedron varies from zero to one for the volume-length measure, and from zero to $\frac{2\sqrt{2}}{3} \doteq 0.943$ for the minimum sine measures (I use the biased minimum sine measure for all tests of quadric smoothing). To compare the quality of a tetrahedron with a vertex (whose quadric error ranges from zero to infinity, zero being ideal), I assign to each surface vertex a quality

$$q_{v_0}(v) = \alpha - \beta Q_{v_0}(v), \tag{3.5}$$

where v is the vertex's current location, v_0 is its original location (in the input mesh), α is an offset parameter, and β is a scale parameter.

I modify the optimization-based smoothing algorithm to include the qualities of the boundary vertices as additional quality functions alongside the qualities of the tetrahedra. These vertex qualities are taken into account when computing active sets and search directions (Section 3.2.1). The objective function for smoothing a boundary vertex v is

$$f(v) = \min\{q_{v_0}(v), f_1(v), f_2(v), \ldots, f_m(v)\}. \tag{3.6}$$

This makes it possible to smooth boundary vertices while limiting the errors in domain shape incurred while smoothing them. Moreover, it provides a memory of the original domain shape, so that if a surface vertex is smoothed in pursuit of better tetrahedron quality and subsequently all the incident tetrahedra improve through topological changes, the smoothing algorithm will try to move the vertex closer to its original position.

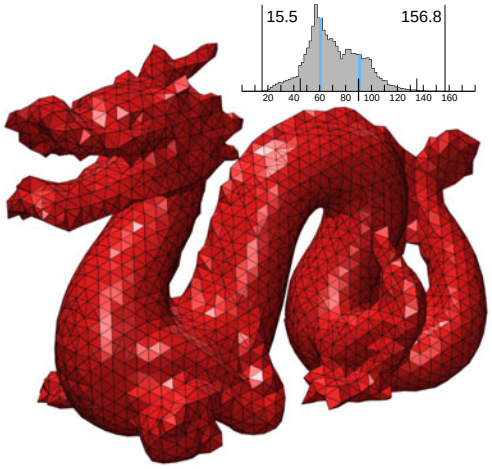
The scale parameter β controls how quickly a vertex is penalized as it moves away from its original position. The offset parameter α has a more subtle effect—it determines how bad a tetrahedron must be to justify the movement of a surface vertex. For example, if we set the offset parameter to 0.6, then a tetrahedron must have a quality below 0.6 to justify moving one of its vertices from its

original position. Unless some tetrahedron incident to a surface vertex has a quality lower than 0.6, the surface vertex will remain the worst active quality function, and will only move in the direction of its "home." Alternatively, if we choose α to be larger than the quality of a perfect (equilateral) tetrahedron, there is "slack" in the surface vertex positions: a vertex in its original position has better quality than a perfect tetrahedron, and can move some distance before there is any chance it will enter the active set. Small offset values restrict the freedom of the smoother to improve poor quality boundary tetrahedra, and it is counterproductive to set α lower than the minimum tetrahedron quality you expect to achieve through mesh improvement, because setting α equal to the worst final tetrahedron quality suffices to ensure that every vertex ends in its original position.

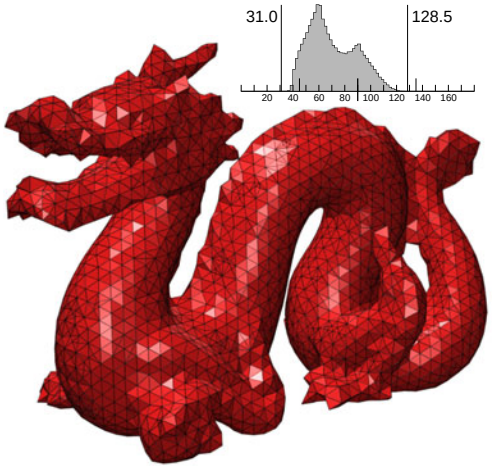
Figure 3.10 illustrates the surface appearance of the Dragon mesh for five different values of surface error. I use the publicly available MESH software [3] to compute the error as the mean symmetric Hausdorff distance between samples on the original mesh boundary and the boundary after improvement. Figure 3.10a shows the unimproved mesh, which has extreme dihedral angles of 15° and 157°. After improvement (using the biased minimum sine measure, described in Section 2.1, as the objective function) with the boundary vertices fixed in place (Figure 3.10b), the extreme angles improve to 31° and 129°. In Figures 3.10c–f, increasingly lenient values of α and β yield better tetrahedron quality and worse surface fidelity.

The default values in my implementation are $\alpha = 0.8$ and $\beta = 500$, used in Figure 3.10d. At this level of error, the differences in the surface shape are barely perceptible, while mesh quality gets a big boost. Even better extreme angles can be obtained by allowing a larger error, as shown in Figures 3.10e–f, but the deviation from the original surface becomes quite pronounced.

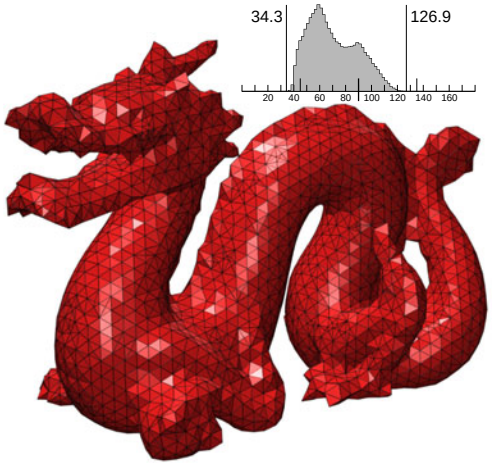
Table 3.1 compares the surface errors and minimum dihedral angles for some combinations of values of the offset and scale parameters α and β. The MESH software is imperfect and reports a small, nonzero error even when there is no difference between the surfaces ($\alpha = 0$). The values shown are averaged over the twelve test meshes described in Section 5. Overall, error increases as the offset parameter α increases and decreases as the scale parameter β increases. Better mesh quality can be obtained by sacrificing surface fidelity, as illustrated by the concordant rise of surface error and minimum dihedral angle. Although the minimum dihedral angle is a somewhat stochastic



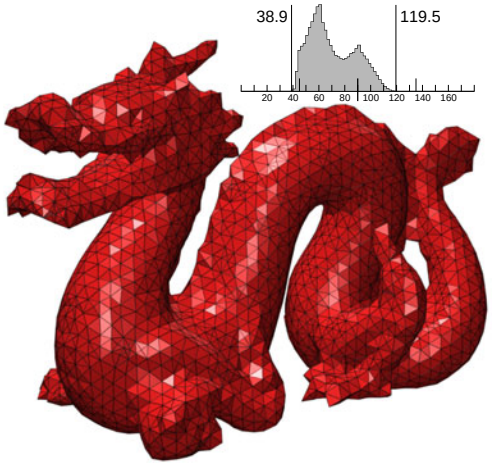
(a) original mesh



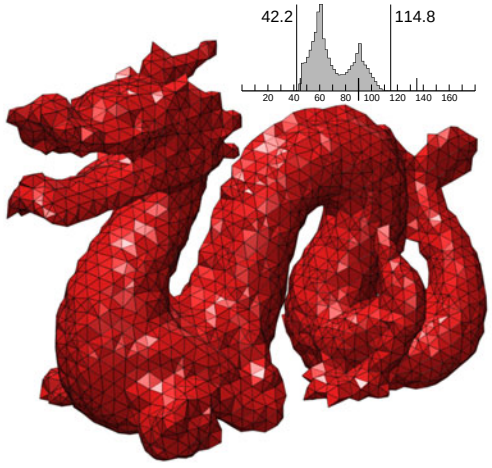
(b) no change in surface shape



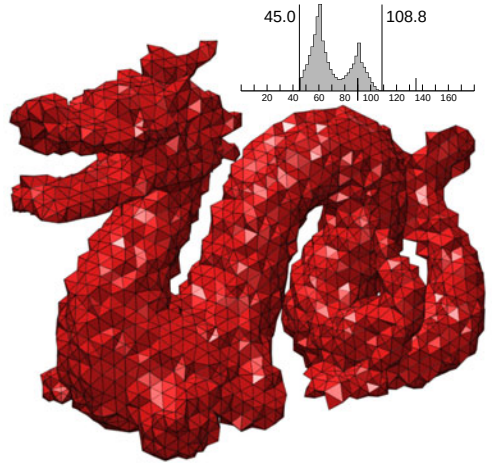
(c) $\alpha = 0.6, \beta = 10{,}000$, mean error 0.00005% BB



(d) $\alpha = 0.8, \beta = 500$, mean error 0.07% BB



(e) $\alpha = 0.9, \beta = 10$, mean error 0.11% BB



(f) $\alpha = 1.1, \beta = 1$, mean error 0.23% BB

Figure 3.10. Surface appearance for different values of α and β in quadric smoothing. In each case, the objective function is the biased minimum sine. Above each mesh is a histogram of dihedral angles, which tabulate mesh quality. In (a), the height of the blue histogram bars should be multiplied by 20 to account for the frequent occurrence of 45°, 60°, and 90° angles. Specular shading emphasizes the differences in surface shape.

measure of mesh quality, it clearly tends to increase as the offset parameter α is increased and decrease as the scale parameter β is increased.

Table 3.1. Top: Approximation error as a percentage of the bounding box size for some combinations of the offset parameter α and the scale parameter β. Bottom: Minimum dihedral angle, in degrees, for the same combinations of the offset parameter α and the scale parameter β. The values shown are averaged over the twelve test meshes described in Section 5.

	$\alpha = 0.0$	0.6	0.7	0.8	0.9	1.0	1.1
$\beta = 1$	0.00006	0.13053	0.37932	0.84340	1.09502	1.22895	1.33258
10	0.00006	0.04135	0.15642	0.33743	0.46754	0.54043	0.62262
100	0.00006	0.01473	0.05133	0.11827	0.15830	0.19429	0.21516
500	0.00006	0.00658	0.02631	0.04649	0.07524	0.08892	0.10088
1000	0.00006	0.00597	0.01881	0.03839	0.05263	0.06258	0.07258
10000	0.00006	0.00173	0.00585	0.01101	0.01599	0.01939	0.02147

	$\alpha = 0.0$	0.6	0.7	0.8	0.9	1.0	1.1
$\beta = 1$	34.31	34.85	39.01	40.71	40.76	40.99	40.89
10	34.31	34.63	37.45	39.57	38.82	39.63	40.00
100	34.31	34.26	36.23	37.09	37.21	38.42	37.99
500	34.31	34.02	35.23	36.56	37.19	36.55	37.36
1000	34.31	33.58	35.69	36.04	36.57	36.27	36.43
10000	34.31	34.15	34.86	35.35	34.79	35.85	36.52

Exactly how much surface error you are willing to tolerate depends on the application. For example, when working with a mesh that was generated from a smooth boundary, one could choose the same error tolerance that was allowed in the original piecewise linear discretization of the surface. As a benchmark, I aimed for a mean error of about 0.05% of the bounding box length, which is the best surface fidelity obtained by the variational mesh generation technique of Alliez, Cohen-Steiner, Yvinec, and Desbrun [1]. For most meshes, I achieve the highest tetrahedral mesh quality with such a surface error by using my default parameter values of $\alpha = 0.8$ and $\beta = 500$. Sometimes, better results are achieved by tuning the parameters for a particular mesh. Finer surface triangulations can tolerate more liberal values of α and β without incurring much surface error.

3.3 Edge contraction

Edge contraction (sometimes called an *edge collapse*) removes an edge from the mesh, replacing its two endpoints with a single vertex. The tetrahedra that share the contracted edge are deleted from the mesh. Edge contraction is commonly used to coarsen tetrahedral meshes [19; 34], sometimes in methods that seek to preserve or improve tetrahedron quality during simplification [38; 16]. Figure 3.11a shows a two-dimensional example of edge contraction.

It is possible for edge contraction to invert a triangle or tetrahedron, as Figure 3.11b illustrates. In my implementation, the vertex that replaces the contracted edge is placed at one of the edge's original endpoints. Contraction fails if both of these locations yield inverted or degenerate tetrahedra.

When one or more of an edge's endpoints lie on the boundary of the mesh, care must be taken not to alter the domain shape. Edge contraction is allowed for an edge with one vertex on the boundary, as long as the vertex that replaces the edge is in the same place as the boundary endpoint. Edge contraction is sometimes possible between two vertices on the boundary, with constraints. For example, a boundary edge between two facet vertices can be contracted without changing the domain shape. When quadric smoothing is used, edge contraction between two boundary vertices is not allowed.

Edge contraction succeeds in improving the quality vector of the mesh if the worst tetrahedron incident to the new vertex is better than worst tetrahedron incident to either of the original endpoints before the contraction. In practice, contraction is unlikely to succeed if the new vertex remains at its original position. Instead, I apply nonsmooth optimization-based smoothing (described in Section 3.2) to the vertex before evaluating the success of the contraction. This greatly improves the chances that an edge contraction operation will improve the quality of the mesh.

3.4 Vertex insertion

Vertex insertion has a long history in mesh generation [24; 13; 50; 43; 14; 45], but surprisingly has been ignored in papers on triangular or tetrahedral mesh improvement. My vertex insertion

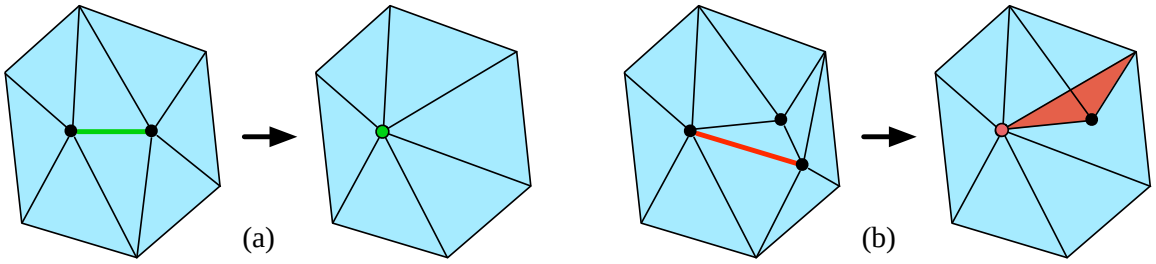


Figure 3.11. Two examples of edge contraction in two dimensions. In (a), the green edge contracts to its left endpoint. In (b), the red edge contracts to its left endpoint, yielding an inverted triangle (colored red).

operator aims to improve a particularly bad tetrahedron of the mesh by inserting a new vertex inside it or on its boundary. It proceeds as follows.

1. Choose a point in a bad tetrahedron at which to create a new vertex.
2. Create a star-shaped cavity around the point by deleting tetrahedra, including the bad one.
3. Retriangulate the cavity with new tetrahedra that all adjoin the new vertex.
4. Improve the tetrahedra in and near the cavity with smoothing and topological operations.
5. Evaluate the success of the insertion operation.
6. If the insertion operation has not improved the mesh, reverse all the changes made during the insertion attempt.

The process is illustrated in Figure 3.12. I first choose a point in a bad tetrahedron as a candidate place to insert a vertex, as described in Section 4.2.4. After choosing a trial point, I hollow out a cavity around the point by deleting nearby tetrahedra. Whichever tetrahedra I choose to delete, their deletion will create a polyhedral cavity in the mesh that must be retriangulated. Ideally I would like the optimal triangulation of the cavity, but that would put me in the position of solving the tetrahedral mesh generation problem all over again. Instead, I choose the simplest retriangulation of the cavity: one tetrahedron for each triangular face of the cavity, with the inserted vertex being the fourth vertex of every new tetrahedron. This retriangulation will induce inverted tetrahedra unless the cavity is star-shaped from the point of view of the inserted vertex. (A polyhedron is *star-shaped* from the perspective of a particular point if you can see all of the polyhedron's boundary from that point.)

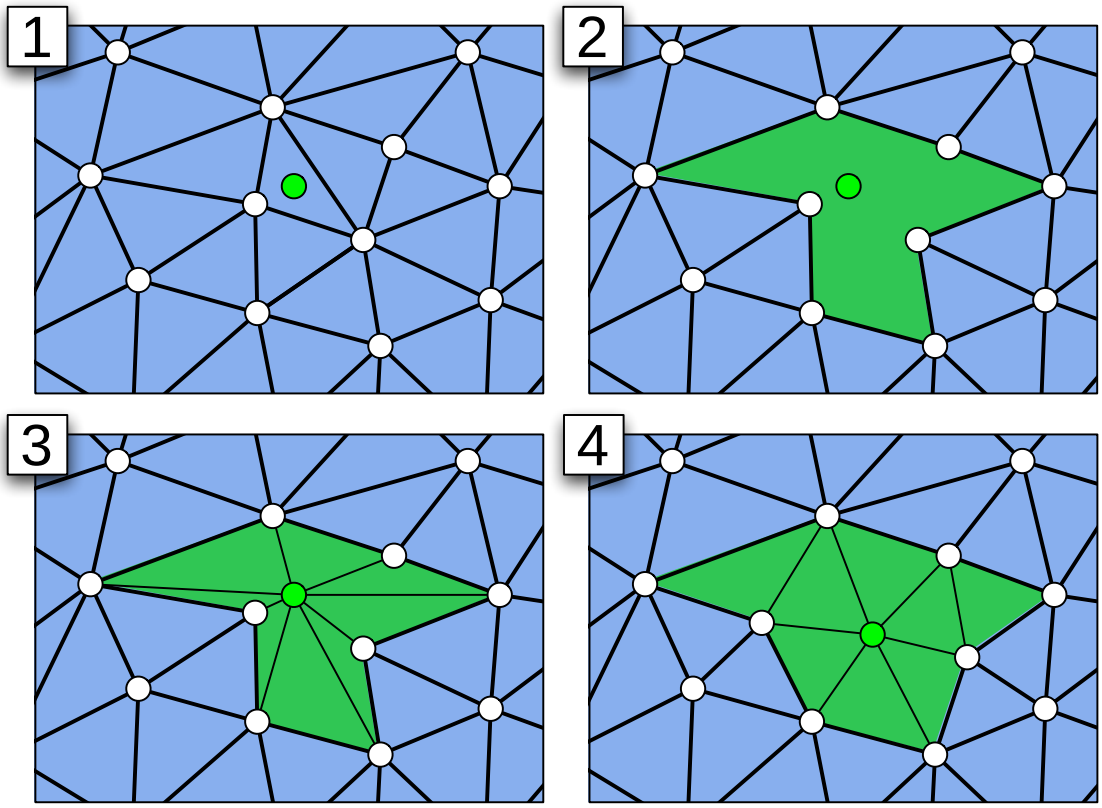


Figure 3.12. An overview of the vertex insertion process. (1) Choose a point in a bad tetrahedron. (2) Create a star-shaped cavity around the point by deleting tetrahedra. (3) Triangulate the cavity. (4) Improve the mesh in and near the cavity in an effort to improve the worst modified tetrahedron's quality.

How I choose the tetrahedra to delete is discussed in Sections 3.4.1 and 3.4.4. My goal is to find the star-shaped cavity that gives subsequent local improvement operations the best chance of yielding an improved mesh.

3.4.1 Finding the optimal cavity

Jonathan Shewchuk devised an algorithm for finding the optimal star-shaped cavity, which first appeared in our paper *Aggressive Tetrahedral Mesh Improvement* [33]. The algorithm views the mesh as a directed graph G with one node for each tetrahedron, as depicted in Figure 3.13. For simplicity, we identify nodes of the graph with the tetrahedra they represent. Let p be the location of the inserted vertex. G contains a directed edge (v, w) if the tetrahedron v shares a triangular

face with the tetrahedron w, and v occludes w from p's point of view. The edge (v, w) reflects the geometric constraint that w can only be included in the cavity C if v is included—that is, the cavity must be star-shaped from p's perspective. (If p is coplanar with the triangular face that v and w share, direct the edge arbitrarily.) Although G can have cycles, they are rare, so we adopt some nomenclature from trees: if $(v, w) \in G$ then w is a *child* of v and v is a *parent* of w. Any tetrahedron that contains p is a *root* of G. Usually there is just one root tetrahedron, but if p lies on a face or edge, all the tetrahedra sharing that face or edge are roots. Because the new vertex is inserted at p, all the roots must be part of the cavity.

The algorithm for finding an optimal cavity computes a cut in G that induces a cavity in the mesh. For each triangular face that belongs to only one tetrahedron in G, add a "ghost node" to G to act as a neighboring tetrahedron. Then, every leaf of G is a ghost node, as Figure 3.13 shows. (If the domain is not convex, not every ghost node need be a leaf).

The tetrahedra in G, except the ghost nodes, are candidates for deletion. For each edge $(v, w) \in G$, let f be the triangular face shared by the tetrahedra v and w. The algorithm labels (v, w) with the quality of the tetrahedron $\text{conv}(\{p\} \cup f)$—the tetrahedron that will be created if v is deleted but w survives.

The problem is to partition G into two subgraphs, G_r and G_l, such that G_r contains the root tetrahedra and G_l contains the leaves (and other ghost nodes), as illustrated in Figure 3.13. The deleted tetrahedra I will be the nodes of G_r, and the surviving tetrahedra will be the nodes of G_l. Because the cavity $C = \bigcup_{t \in I} t$ must be star-shaped from p's perspective (to prevent the creation of inverted tetrahedra), no tetrahedron in G_l may be a parent of any tetrahedron in G_r. The goal is to find the partition that satisfies this constraint and maximizes the minimum weight among the cut edges (because the cut edge with minimum weight determines the worst new tetrahedron).

The algorithm in Listing 3.1 computes this optimal cut. The algorithm iterates through the edges of G, from worst quality to best, and greedily ensures that each edge will not be cut, if that assurance does not contradict previous assurances. Upon termination, the tetrahedra labeled "cavity" become the set I of tetrahedra to be deleted, and the set J of tetrahedra to be created are determined by the triangular faces of the cavity C, which are recorded by line 10 of the pseudocode. After an initial

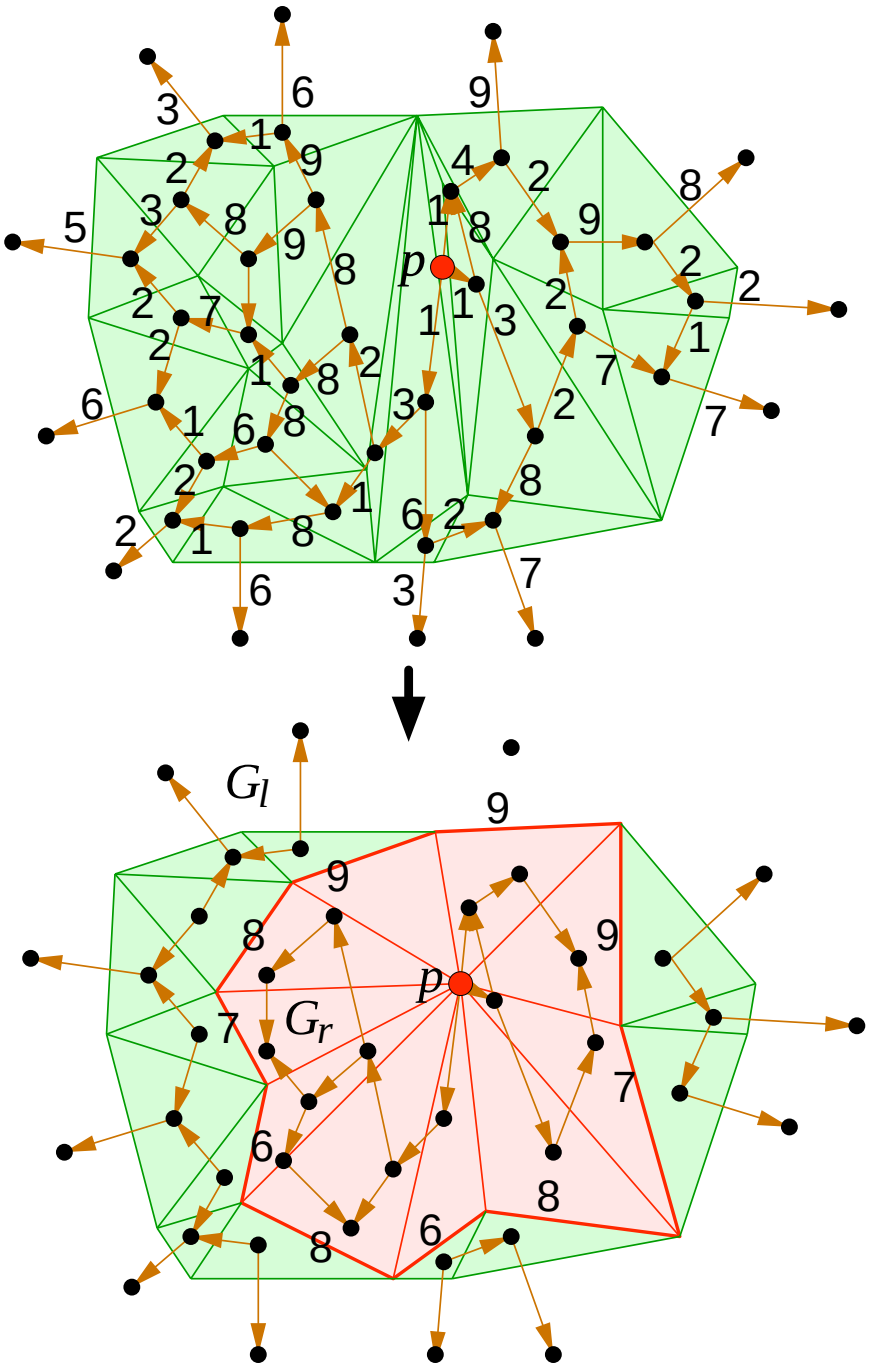


Figure 3.13. Vertex insertion as graph cut optimization. In this example, the smallest cut has weight 6. The weights of the cut edges are the qualities of the new triangles.

```
FindOptimalCavity(G)
1    Sort the edges of G from smallest to largest quality.
2    H ⇐ a graph with the same nodes as G but no edges (yet).
3    All vertices of G are initially unlabeled.
4    Label every root of G "cavity."
5    for each ghost node g of G
6        AntiCavity(g)

7    for each directed edge (v, w) of G (in sorted order)
8        if v is labeled "cavity"
9            if w is labeled "anti-cavity"
10               Record (v, w), which determines a new tetrahedron in J.
11           else if w is unlabeled
12               Cavity(w)
13       else if v is unlabeled
14           if w is labeled "anti-cavity"
15               AntiCavity(v)
16           else { w is unlabeled }
17               Add (v, w) to H.

Cavity(w)
18   Label w "cavity."
19   for each unlabeled parent p of w in G
20       Cavity(p)
21   for each unlabeled child c of w in H
22       Cavity(c)

AntiCavity(v)
23   Label v "anti-cavity."
24   for each unlabeled child c of v in G
25       AntiCavity(c)
26   for each unlabeled parent p of v in H
27       AntiCavity(p)
```

Listing 3.1: Algorithm for computing the cavity that optimizes the quality of the new tetrahedra when a new vertex is inserted. Upon completion, the tetrahedra to be deleted are labeled "cavity."

sorting step, the rest of the algorithm runs in $O(n)$ time, where n is the number of tetrahedra in the graph. In principle we could use radix sort, yielding a linear-time algorithm, but our implementation uses an $O(n \log n)$-time quicksort.

In an implementation of the algorithm, there is no need to store H as a separate graph. Instead, each edge of G can store a bit indicating whether it is in H too.

Insertion	Attempts	Successes	Success rate	Dihedral angle bounds
without cavity improvement	17,555	52	0.3%	20.7° / 158.7°
with cavity improvement	5,631	379	6.7%	38.2° / 140.7°

Table 3.2. A comparison of the insertion success rate with and without cavity improvement. The improvement algorithm optimized the minimum sine objective function on the RAND1 mesh introduced in Chapter 5. The rightmost column lists the minimum and maximum dihedral angles of the mesh after improvement terminates.

3.4.2 Improving the mesh near the cavity

To succeed, the vertex insertion operator must leave behind a set of tetrahedra whose worst member is better than the worst member of the set it replaces. If the insertion operation stopped after triangulating the optimal star-shaped cavity, it would seldom succeed. The initial retriangulation, which is formed by creating tetrahedra connecting the inserted vertex to each triangular face of the cavity, usually contains some bad tetrahedra, the worst of which is often even poorer than the tetrahedron the insertion operation was meant to replace.

I have found that vertex insertion is far more powerful if it is followed by improvement of the retriangulated cavity prior to evaluating the success of the insertion operation. Table 3.2 compares the success rate of insertion with and without improvement of the cavity tetrahedra. For this mesh, cavity improvement increases the likelihood that insertion succeeds by a factor of twenty. Without cavity improvement, there are more total insertion attempts because failed insertions are repeatedly retried. These retried insertions skew the success rate, but observe that cavity improvement yields more than seven times as many successes despite the smaller number of attempts. The final dihedral angle bounds reveal that following vertex insertion with cavity improvement leads to a far better final mesh quality.

Listing 3.2 gives pseudocode for my schedule for improving the cavity retriangulated by a vertex insertion operation. The schedule employs the topological improvement operations and optimization-based smoothing discussed in Sections 3.1 and 3.2. IMPROVECAVITY maintains a variable K that stores the set of modified tetrahedra. At any time, K includes all the tetrahedra currently in the mesh that have been created or modified by the insertion operation or subsequent cavity im-

provement operations, including those that have had a vertex smoothed. (K does *not* include deleted tetrahedra.)

```
ImproveCavity(p, J, M)
{ p is the inserted vertex }
{ J is the set of tetrahedra in the retriangulated cavity }
{ M is the entire mesh under improvement, with J ⊆ M }
1   K ⇐ J      { Initialize the set of modified tetrahedra }
2   attempts ⇐ 8

    { Smooth the inserted vertex and improve the cavity topologically. }
3   repeat
4       q_K ⇐ quality of the worst tetrahedron in K
5       Smooth p.
6       K ⇐ TopologicalPass(K, M)      { K may expand; see Section 4.2.2 }
7       q'_K ⇐ quality of the worst tetrahedron in K
8       attempts ⇐ attempts − 1
9   while attempts > 0 and q'_K > q_K

    { Smooth the vertices of all the affected tetrahedra. }
10  repeat
11      q_K ⇐ quality of the worst tetrahedron in K
12      K ⇐ SmoothingPass(K, M)      { K may expand; see Section 4.2.1 }
13      q'_K ⇐ quality of the worst tetrahedron in K
14      attempts ⇐ attempts − 1
15  while attempts > 0 and q'_K > q_K and |K| < 250

16  return q'_K.
```

Listing 3.2: The schedule for improving the tetrahedra in the retriangulated cavity after vertex insertion. Pseudocode for TopologicalPass appears in Listing 4.2. Pseudocode for SmoothingPass appears in Listing 4.1.

When the vertex insertion operation is complete, but before subsequent cavity improvement, the pseudocode initializes K to equal J, the newly created tetrahedra that fill the optimal cavity. The cavity improvement procedure begins by smoothing the inserted vertex p. Then it applies topological mesh improvement operations to all the tetrahedra in K. The cavity $\bigcup_{t \in K} t$ may grow as these improvement operations occur, because nearby tetrahedra not in K can be deleted by topological operations. Next, the procedure recomputes the quality of the worst tetrahedron in K. If it has improved, the procedure again smooths p and performs a round of topological operations. When these operations no longer yield further improvement, the procedure moves on and smooths *all* the vertices of the tetrahedra in K. Smoothing causes the cavity $\bigcup_{t \in K} t$ to grow again, because

```
AttemptInsert(p, M)
{ p is the position at which to insert a new vertex }
{ M is the mesh under improvement }
1    I ⇐ deleted tetrahedra, computed as discussed in Section 3.4.1
2    q_I ⇐ quality of the worst tetrahedron in I
3    Replace I with the new tetrahedra J.      { see Section 3.4.1 and Listing 3.1 }
4    q_K ⇐ ImproveCavity(p, J, M)      { see pseudocode in Listing 3.2 }
5    if q_K > q_I
6        return true.
7    Roll back all changes since the beginning of this procedure call.
8    return false.
```

Listing 3.3: The complete vertex insertion algorithm, which starts by creating and triangulating the optimal cavity, and then improves the tetrahedra in the cavity.

the tetrahedra in K share vertices with other tetrahedra in the mesh. Smoothing is repeated as long as the worst tetrahedron's quality continues to improve and K does not grow too large. Finally, the procedure notes the quality of the worst tetrahedron in K, which will be used to judge the overall success of the insertion operation.

The reader may wonder why I do not include smoothing and topological operations in a single loop. The purpose of separating them is to limit the growth of K while its tetrahedra improve. Each pass of smoothing expands K substantially. If the smoothing and topological operations were together in one loop, the cost of applying topological operations to so many tetrahedra would slow down the insertion operation considerably.

3.4.3 Evaluating the success of an insertion operation

After cavity improvement is complete, I evaluate the success of the insertion operation. Let q_p be the quality of the tetrahedron originally targeted by insertion, let q_I be the quality of the worst tetrahedron deleted to form the optimal star-shaped cavity, and let q_K be the quality of the worst tetrahedron in K, the set of modified tetrahedra at the end of cavity improvement. Note that $q_I \leq q_p$ because the tetrahedron targeted by insertion is always deleted as a part of cavity retriangulation, but other, poorer tetrahedra may also have been deleted. The insertion operation is declared a success if $q_K > q_I$, even if $q_K < q_p$—insertion succeeds whenever the worst modified tetrahedron in K is better than the worst deleted tetrahedron in I.

Another reason to restrict the size of K during cavity improvement is now apparent. The larger I allow K to grow, the less likely it becomes that q_K will exceed q_I. I am trying to fix a particularly bad tetrahedron with the insertion operation; if K grows too large, I risk including in it other hard-to-fix tetrahedra and hurting my chances of success. Through trial and error, I have designed the insertion operation to be as local as possible while maximizing its chances of success in practice.

If insertion fails, I must restore the mesh to its state before the insertion operation began. In my implementation, I keep a journal of every change that is made to the mesh. Before beginning an insertion attempt, I make a note in the journal. If insertion fails, I reverse each operation until the mesh is back to its previous state. The complete algorithm for vertex insertion is given in Listing 3.3.

3.4.4 Heuristics for successful vertex insertion

I have observed that the optimal subgraph rarely includes tetrahedra that are far (as measured by path length) from a root. Thus, to save time, I do not actually consider the whole graph G when computing the optimal cavity. Instead, I limit myself to a subgraph of G whose nodes are the roots of G and all the tetrahedra that are reachable in G from the roots by a directed path of length six or less. This subgraph typically has 5–100 tetrahedra. On rare occasions, this heuristic may prevent the algorithm from finding the true optimal cavity, but that is a small price to pay to avoid the cost of inspecting the entire mesh. Observe that if the domain is not convex, a tetrahedron that is reachable from a root might have an ancestor that is not (because the latter tetrahedron and the new vertex are separated by the exterior of the domain), in which case the former tetrahedron and its descendants must be labeled "anti-cavity."

I find that insertion attempts with subsequent improvement (as described in Section 3.4.2 and Listing 3.3) are more often successful with larger cavities than with the optimal cavities. To promote the selection of larger cavities, I exaggerate the quality assigned to outgoing edges in G that correspond to faces of tetrahedra further away from the roots. I label each tetrahedron t in G with a depth that is one greater than the minimum depth of its parents. Then, I multiply the quality assigned to the outgoing edges of t by an exaggeration factor according to its depth. Table 3.3 lists the factors

Depth	Factor
0	1.0
1	1.6
2	2.3
3	2.9
4+	3.3

Table 3.3. The factors by which the edge weights are exaggerated according to the depth of the parent tetrahedron.

I use, which were obtained by trial-and-error exploration in an effort to maximize the rate at which insertion attempts succeed. The set I of deleted elements typically comprises 5–15 tetrahedra.

Chapter 4

A Static Mesh Improvement Schedule

A mesh improvement schedule specifies how mesh improvement operations (Chapter 3) are applied to a tetrahedral mesh. Mesh improvement schedules are inherently heuristic and are usually developed experimentally, so the literature contains diverse approaches. After reviewing some previous work, I describe my static mesh improvement schedule. Recall that in static mesh improvement, we try to improve a single mesh to as high a quality as possible, changing as much of the mesh and expending as much computation time as is necessary to rescue a mesh and make it usable. A *dynamic* mesh improvement schedule is discussed in Chapter 8.

4.1 Prior work on mesh improvement schedules

Barry Joe [30] uses 2-3 and 3-2 flips and a variety of composite transformations that are made of sequences of flips and designed to get a hill-climbing optimizer across what was formerly a valley in the objective function, thereby leading the way to a better local optimum. Some of these composite operations are equivalent to edge removal. His mesh improvement algorithm checks each face of the mesh to see if any of his topological transformations will improve the local tetrahedra. It performs passes over the entire mesh (checking each face), and terminates when a pass makes no changes. His experiments show that he can eliminate most, but not all, tetrahedra with radius ratios below

0.3. (In my experiments, I eliminate all tetrahedra with radius ratios below 0.55 by optimizing the volume-length objective V/ℓ_{rms}^3. See Section 5.1.)

Freitag and Ollivier-Gooch's schedule [23] begins with a pass of 2-3 flips that enforce the Delaunay in-sphere criterion (testing each interior face of the mesh once), then a pass of 2-3 flips that optimize the minimum sine measure, then a pass of edge removal operations that optimize the minimum sine, then two passes of optimization-based smoothing. Next, a procedure that targets only the worst tetrahedra in the mesh attempts to remove them with 2-3 flips and edge removal operations. Two more passes of smoothing complete the schedule. For many of their meshes, they obtain dihedral angles bounded between about 12° and 160°, but these results are not consistent across all their test meshes. Dihedral angles less than 1° occasionally survive, and in more examples dihedral angles under 10° survive. (In my experiments, I eliminate all tetrahedra with dihedral angles below 34° or above 131° by optimizing the biased minimum sine objective. See Section 5.1.)

Edelsbrunner and Guoy [20] demonstrate that that a theoretically motivated technique called *sliver exudation* [12], which uses sequences of 2-3 and 3-2 flips to remove poor tetrahedra from meshes, usually removes most of the bad tetrahedra from a mesh, but rarely all. Again, dihedral angles less than 1° sometimes survive, and in most of their examples a few dihedral angles less than 5° remain.

Alliez, Cohen-Steiner, Yvinec, and Desbrun [1] propose a "variational meshing" algorithm that alternates between optimization-based smoothing (using a smooth objective function) and computing a new Delaunay triangulation from scratch. This algorithm generates meshes that have only a small number of dihedral angles under 10° or over 160°, but it does not always eliminate all mediocre tetrahedra, especially on the boundary. Note that variational meshing is a standalone mesh generation algorithm, and cannot be used as a mesh improvement algorithm in many contexts because the mesh it generates does not conform to a specified triangulated boundary.

4.2 Four mesh improvement passes

I package mesh improvement operations into *passes* that operate on sets of tetrahedra. In all the passes, I change the mesh only if the quality of the worst tetrahedron improves.

```
SmoothingPass(K, M)    { K is a subset of the tetrahedra in the mesh M }
1     V ⇐ set of all vertices of the tetrahedra in K
2     for each vertex v in V
3           Perform nonsmooth optimization-based smoothing on v.
4     K′ ⇐ K ∪ all the tetrahedra in M that had a vertex moved by smoothing
5     return K′.
```

Listing 4.1: Pseudocode for SmoothingPass, which applies nonsmooth optimization-based smoothing to each vertex of each tetrahedron in the set K.

4.2.1 The smoothing pass

I encapsulate the vertex smoothing operation (Section 3.2) in a smoothing pass, specified in the SmoothingPass pseudocode in Listing 4.1. The pass operates on a set K of tetrahedra, smoothing each vertex of each tetrahedron in turn. Smoothing a vertex modifies all the tetrahedra incident to that vertex, some of which might not be in K; therefore, the pass returns an updated set of tetrahedra K' that includes all the tetrahedra altered by the pass. This bookkeeping is necessary to implement the vertex insertion operator (Section 3.4), which uses SmoothingPass as part of its cavity improvement algorithm.

4.2.2 The topological pass

I combine the the edge and face removal operations described in Section 3.1 into a single improvement pass called TopologicalPass, shown in Listing 4.2. The algorithm first attempts to remove each edge of the tetrahedra in K, and then to remove each face of the tetrahedra in K. Normally, each attempt to remove a face employs the multi-face removal operation of Section 3.1.2. However, our software includes an option to turn off multi-face removal while still leaving intact the ability to remove a face by a 2-3 flip or, if an edge of the face lies on a flat domain boundary, a 2-2 flip. A 2-2 flip performs an edge flip on a flat boundary, replacing two tetrahedra with two others (see Figure 3.2). (In Chapter 5, I investigate the consequences of turning off multi-face removal while maintaining the more easily implemented flips, as well as the consequences of turning off face removal, or edge removal, altogether.)

The pseudocode returns a set K' that contains all the surviving tetrahedra from K and all the

```
TopologicalPass(K, M)      { K is a subset of tetrahedra in the mesh M }
1    E ⇐ set of all edges of tetrahedra in K
2    F ⇐ set of all faces of tetrahedra in K
3    for each edge e ∈ E
4        if e still exists
5            Attempt to remove edge e.      { See Section 3.1.1 }
6    for each face f ∈ F
7        if f still exists
8            Attempt to remove face f (multi-face or 2-3 or 2-2 flip).      { See Section 3.1.2 }
9    K′ ⇐ the surviving tetrahedra of K and all the new tetrahedra created by this call.
10   return K′.
```

Listing 4.2: Pseudocode for TopologicalPass. The algorithm attempts edge removal on each edge in K, then face removal on each face in K.

```
EdgeContractPass(K, M)      { K is a subset of tetrahedra in the mesh M }
1    E ⇐ set of all edges of tetrahedra in K
2    for each edge e ∈ E
3        if e still exists
4            Attempt to contract edge e.      { See Section 3.3 }
5    K′ ⇐ the surviving tetrahedra of K and the tetrahedra in M altered by edge contractions
6    return K′.
```

Listing 4.3: Pseudocode for EdgeContractPass, which attempts edge contraction on each unique edge of each tetrahedron in K.

newly created tetrahedra. Again, this bookkeeping is needed by the cavity improvement algorithm that follows a vertex insertion operation.

4.2.3 The edge contraction pass

The mesh improvement pass EdgeContractPass (Listing 4.3) takes a set of tetrahedra, computes the set of their edges, and attempts to contract each edge once, as described in Section 3.3. As usual, an edge contraction is performed only if it improves the quality vector of the mesh. Note that when the endpoint of an edge is moved by smoothing following the contraction of another edge, the moved edge is still considered to be the same edge by line 3 of the pseudocode.

4.2.4 The vertex insertion pass

The vertex insertion pass takes as input a set K of tetrahedra which are usually all of bad quality. It attempts to insert a vertex into each tetrahedron in K, as described in Section 3.4. Pseudocode for InsertionPass appears in Listing 4.4. A tetrahedron may have one or more faces or edges on the boundary of the domain. By inserting a vertex on the boundary, the algorithm has a chance to repair a bad boundary triangulation as well as remove a bad tetrahedron. Therefore, the algorithm attacks a tetrahedron by attempting to insert a vertex a various locations in turn. First, InsertionPass tries to insert a vertex at the barycenter of each boundary face. If these insertions fail, or if the tetrahedron has no boundary faces, InsertionPass attempts to insert at the tetrahedron's barycenter. If this attempt also fails, InsertionPass attempts to insert a vertex at the midpoint of each boundary edge of the tetrahedron. This ordering of insertion locations is based on trial and error. Of all the possible permutations, this one performs the best in practice: it produces more successful insertions and leads to the best final mesh quality.

In Section 3.2.5, I describe an optimization-based smoothing algorithm that uses quadric errors to smooth vertices on curved domain boundaries. When I insert a new vertex on the boundary, I must assign it a quadric. The neighbors of the new vertex might be displaced from their original positions, and the displacement might be temporary. Improvement of the mesh may allow them to return to their original positions, so it is important that the new vertex has a "home" position that makes sense relative to its neighbors' original positions.

Therefore, when I insert a new vertex, I compute two positions for it. I place the vertex at a face barycenter or edge midpoint, determined by the current positions of the neighbors. I also compute a "home" position for the new vertex at a face barycenter of edge midpoint determined by the *original* positions of the neighbors. After the new vertex is inserted, it adjoins three boundary triangles if it is a face barycenter, or four triangles if it is an edge midpoint. I compute a quadric function for the new vertex from these triangles, as if their vertices were at their original positions. This quadric will have zero error if the new vertex ever moves to its home position. Because the current position of the new vertex is calculated from its neighbors' current positions, the new vertex may be born with nonzero quadric error.

```
InsertionPass(K, M)      { K is a subset of tetrahedra in the mesh M }
1    for each tetrahedron t ∈ K that still exists
2        for each face f of t on the mesh boundary (if any)
3            p ⇐ point at barycenter of f
4            if AttemptInsert(M, p)
5                Restart outer loop on next tetrahedron.
6        p ⇐ point at barycenter of t
7        if AttemptInsert(M, p)
8            Restart outer loop on next tetrahedron.
9        for each edge e of t on the mesh boundary (if any)
10           p ⇐ point at midpoint of e
11           if AttemptInsert(M, p)
12               Restart outer loop on next tetrahedron.
13   K′ ⇐ the surviving tetrahedra of K and all tetrahedra changed or created by this call.
14   return K′.
```

Listing 4.4: Pseudocode for InsertionPass, which applies my new vertex insertion operation on a set of tetrahedra K. The pseudocode for AttemptInsert is given in Listing 3.3.

4.3 Success criteria for a pass

By design, the improvement passes never worsen the quality vector of a mesh. Mesh improvement software could execute the passes over and over until the mesh stops changing. This strategy is practical for purely combinatorial mesh transformations that do not create new vertices. Joe's improvement schedule performs topological transformations repeatedly until it reaches a locally optimal mesh [30]. But smoothing can always make a tiny improvement to a mesh. It is impractical to wait for smoothing to find a local optimum.

Instead, I control the mesh improvement process by evaluating the effectiveness of each pass, and halting improvement when too little progress is made. How do I measure progress? I summarize the mesh quality vector with several *thresholded means* and judge the success of a pass by comparing the values of these means before and after.

A *thresholded mean* $m(x, M)$ is the mean of the qualities of all tetrahedra in M, with the tetrahedra whose qualities exceed x artificially assigned the quality x. For example, if our objective function is the minimum sine measure, $m(\sin 5°, M)$ is the mean of the "thresholded" qualities of all the tetrahedra in M: the tetrahedra with a dihedral angle less than 5° or greater than 175° are assigned their actual qualities, and the better tetrahedra are assigned a quality of $\sin 5°$. The purpose

minimum sine and biased minimum sine	volume-length ratio	radius ratio
$\sin 1^\circ$	0.1	0.1
$\sin 5^\circ$	0.2	0.2
$\sin 10^\circ$	0.3	0.3
$\sin 15^\circ$	0.4	0.4
$\sin 25^\circ$	0.5	0.5
$\sin 35^\circ$	0.6	0.6
$\sin 45^\circ$	0.7	0.7

Table 4.1. The thresholds for each of the seven means used to approximate quality vector improvement and judge the success of improvement passes.

of a thresholded mean is to measure progress in the low-quality tetrahedra while ignoring changes in the good ones. If the thresholded mean $m(x, M)$ improves, then the tetrahedra whose original quality was less than x have improved. I gauge the overall improvement of the quality vector by taking several means with differing thresholds.

After each pass, I compute seven thresholded means as well as the quality of the worst tetrahedron in the mesh. I declare a pass successful if at least one thresholded mean improves by at least 0.0001 or if the quality of the worst tetrahedron in the mesh improves at all. The thresholds for each quality measure tested appear in Table 4.1.

4.4 The static mesh improvement schedule

My static improvement schedule is an algorithm for applying the smoothing, topological, edge contraction, and vertex insertion passes to an input mesh to produce the highest quality output mesh I can. In a departure from much existing work, I design the schedule to adapt to the input. If an input mesh responds well to fast improvement operations like vertex smoothing, it makes sense that slow operations like vertex insertion should be delayed until smoothing is no longer effective. Moreover, users often wish to preserve the density of vertices in their meshes, in which case transformations that change the number of vertices are less desirable. Accordingly, I have ordered the passes in nested loops of increasing "aggression," starting with passes of smoothing, then moving on to topological passes, and finally applying passes of edge contraction and vertex insertion.

The mesh improvement schedule ImproveMesh appears in Listing 4.5. I begin with one global smoothing pass, one global topological pass, and one global edge contraction pass, which often carry out the easiest repairs. Then I apply smoothing passes to the entire mesh so long as they are successful according to the criterion described in Section 4.3. If smoothing fails, I apply a pass of topological improvement to the entire mesh. If progress is still insufficient, I resort to one pass each of edge contraction and vertex insertion. These two passes usually target only the worst 3.5% of tetrahedra in the mesh. However, at least once before the schedule ends, these two passes target every tetrahedron with a dihedral angle less than 40° or greater than 140°; I call this a "desperation pass."

When all four passes in sequence fail to make sufficient progress, the schedule increments a failure count. The failure count is reset to zero whenever any pass succeeds. The schedule terminates when the failure count reaches three.

This schedule is designed to maximize the quality of the mesh, not to minimize the running time. I spend time extravagantly in pursuit of the highest quality mesh I can obtain. In practice, about 80% of the total mesh improvement is usually achieved in the first 10–20% of the running time of the schedule. I explore the tradeoffs between running time and improvement in Section 5.2. If the user is satisfied with a specified minimum quality, the schedule can stop when that quality is reached.

```
ImproveMesh(M)      { M is a tetrahedral mesh }
1     SmoothingPass(M, M)      { See Section 4.2.1 }
2     TopologicalPass(M, M)      { See Section 4.2.2 }
3     EdgeContractPass(M, M)      { See Section 4.2.3 }
4     failed ⇐ 0
5     while failed < 3
6         Q ⇐ list of quality indicators for M      { See Section 4.3 }
7         SmoothingPass(M, M)
8         if M is sufficiently improved compared to Q
9             failed ⇐ 0
10        else
11            TopologicalPass(M, M)
12            if M is sufficiently improved compared to Q
13                failed ⇐ 0
14            else
15                if failed = 1
                      { desperation pass }
16                    L ⇐ list of tetrahedra in M with a dihedral angle < 40° or > 140°
17                    EdgeContractPass(L, M)
18                    L ⇐ list of tetrahedra in M with a dihedral angle < 40° or > 140°
19                    InsertionPass(L, M)      { See Section 4.2.4 }
20                else
21                    L ⇐ list of the worst 3.5% of tetrahedra in M
22                    EdgeContractPass(L, M)
23                    L ⇐ list of the worst 3.5% of tetrahedra in M
24                    InsertionPass(L, M)      { See Section 4.2.4 }
25                if M is sufficiently improved compared to Q
26                    failed ⇐ 0
27                else
28                    failed ⇐ failed +1
```

Listing 4.5: Static mesh improvement schedule.

Chapter 5

Static Mesh Improvement Results and Discussion

Trial and error guided the creation of the static mesh improvement schedule described in Chapter 4. An exploration of the empirical effectiveness of the schedule not only reveals its strengths and weaknesses, but also explains many of my design decisions. A few key observations supported by experiments in this chapter are:

- The mesh improvement schedule succeeds in improving meshes created with different mesh generation techniques, meshes that possess different boundary shapes, and meshes with varying numbers of tetrahedra.
- Some improvement operations are more productive than others. Smoothing and vertex insertion are the most critical to mesh quality. Edge removal and face removal are important, but less important than anticipated.
- Operations that change the mesh boundary improve mesh quality substantially and are worth the extra effort to implement.
- The best overall objective function is the volume-length measure, which is most effective in removing large dihedral angles. However, the minimum sine measures are useful for attacking small dihedral angles.

- The worst objective function is the radius ratio. The volume-length objective function optimizes the radius ratio better than the radius ratio objective does.
- Vertex insertion and edge contraction account for most of the running time, and can alter the number of tetrahedra in the mesh considerably.

I tested the mesh improvement schedule on a dozen meshes, shown in Figure 5.1. The meshes come from a variety of sources.

- CUBE1K and CUBE10K are high-quality meshes of a cube generated by Joachim Schöberl's NETGEN software [44].
- HOUSE and P are Delaunay meshes generated by Jonathan Shewchuk's Pyramid software [45] configured so that the vertices are nicely spaced, but no effort is made to eliminate sliver tetrahedra.
- TFIRE is a high-quality mesh of a tangentially-fired boiler, created by Carl Ollivier-Gooch's GRUMMP software [39].
- TIRE, RAND1 and RAND2 come courtesy of Freitag and Ollivier-Gooch [23], who used them to evaluate their mesh improvement algorithms. TIRE is a tire incinerator. RAND1 and RAND2 are *lazy triangulations*, generated by inserting randomly located vertices into a cube, one by one. Each vertex was inserted by splitting one or more tetrahedra into multiple tetrahedra. (Unlike in Delaunay insertion, no flips took place.) The random meshes have horrible quality.
- DRAGON and Cow are medium-quality meshes generated by isosurface stuffing [36]. Their curved boundaries offer an opportunity to evaluate the quadric smoothing method described in Section 3.2.5.
- STGALLEN is a medium-quality mesh generated by variational tetrahedral meshing [1], courtesy of Pierre Alliez. Its curved boundary is also suitable for quadric smoothing.
- STATOR is a mediocre-quality Delaunay mesh of a mechanical part generated by Hang Si's TetGen software [49].

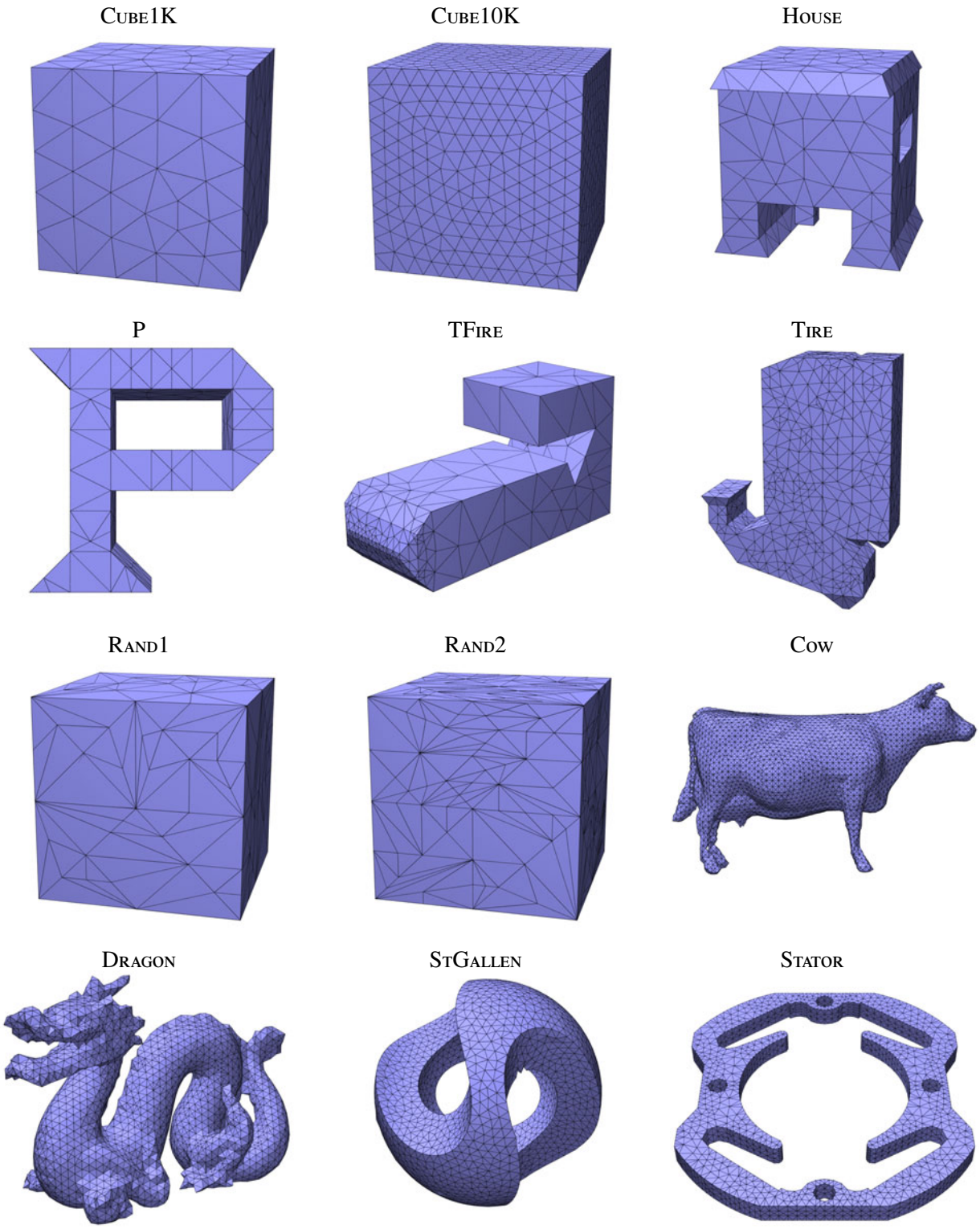


Figure 5.1. The twelve test meshes.

mesh name	min / max input dihedral angle	Freitag et. al min / max	my min / max
TIRE	0.66° / 178.88°	13.67° / 156.14°	34.40° / 130.65°
RAND1	0.32° / 178.97°	15.01° / 159.96°	39.61° / 120.78°
RAND2	0.10° / 179.83°	10.58° / 164.09°	38.32° / 124.19°

Table 5.1. A comparison of the minimum and maximum dihedral angles for three meshes before improvement (second column), after improvement by Freitag and Ollivier-Gooch [23] (third column), and after improvement using my mesh improvement schedule with the biased-minimum sine objective function (fourth column).

5.1 Twelve meshes improved

Tables 5.2–5.4 show statistics for the twelve test meshes before and after improvement by the IMPROVEMESH schedule in Listing 4.5. I tested the minimum sine measure (upper right corner of each box), the biased minimum sine measure (lower right), and the volume-length measure V/ℓ_{rms}^3 (lower left) as objective functions. (I omit meshes optimized for the radius ratio objective, which was not competitive with the volume-length measure.)

Dihedral angles are improved to between 36° and 143° for the minimum sine objective, between 34° and 131° for the biased minimum sine objective, and between 29° and 124° for the volume-length measure. These numbers put my implementation far ahead of any tetrahedral mesh generation or mesh improvement algorithm I have seen reported. Table 5.1 compares the extreme angles reported by Freitag and Ollivier-Gooch with those achieved by my mesh improvement schedule.

Across all objective functions, the final quality is greatest for the meshes that start with high-quality tetrahedra and well-spaced vertices, like CUBE1K and CUBE10K. Nearly as good are the meshes that began with poor-quality tetrahedra but simple boundary shapes, such as RAND1, RAND2, HOUSE, and P. Meshes with curved boundaries (COW, DRAGON, and STGALLEN) also finish with excellent quality, but their boundary vertices are smoothed using quadric smoothing (see Section 3.2.5) and the domain shape is not preserved exactly. Improvement is least successful on meshes with complex boundaries that are preserved exactly—TIRE and STATOR. These results indicate that the boundary shape is influential on the final mesh quality.

Because the vertex insertion and edge contraction operations may add or remove vertices from

Table 5.2. Twelve meshes before and after improvement (continued in Tables 5.3 and 5.4). In each box, the upper left mesh is the input, the upper right mesh is optimized for the minimum sine objective, the lower right mesh is optimized for the biased minimum sine objective, and the lower left mesh is optimized for the volume-length objective. Running times are given for a Mac Pro with a 2.66 GHz Intel Xeon processor. Red tetrahedra have dihedral angles under 10° or over 165°, orange under 20° or over 150°, yellow under 30° or over 135°, green under 40° or over 120°, and better tetrahedra do not appear. Histograms show the distributions of dihedral angles and the minimum and maximum dihedral angles in each mesh. Histograms are normalized so the tallest bar always has the same height; absolute numbers of tetrahedra cannot be compared between histograms.

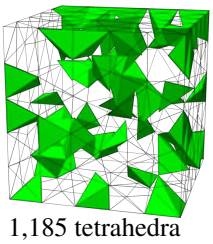
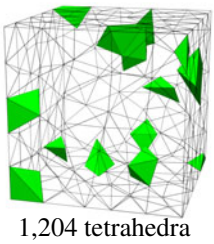
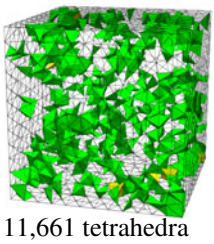
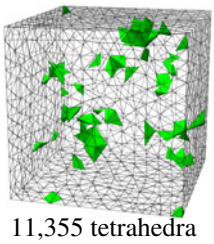
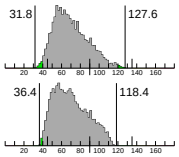
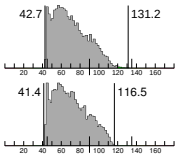
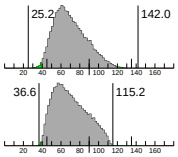
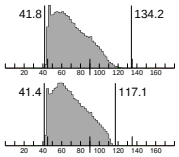
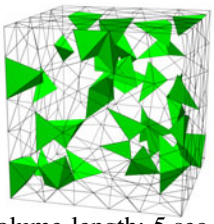
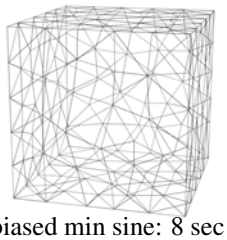
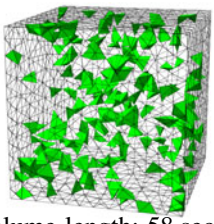
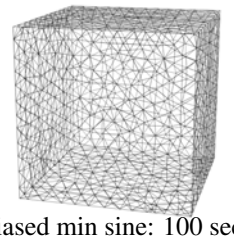
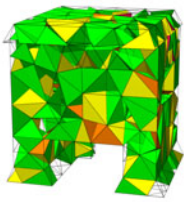
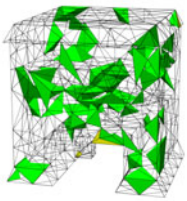
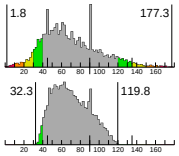
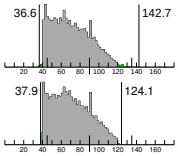
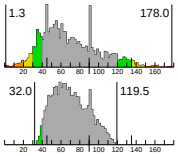
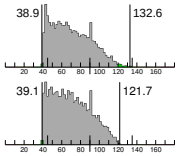
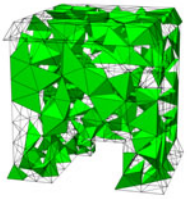
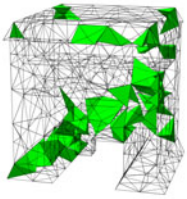
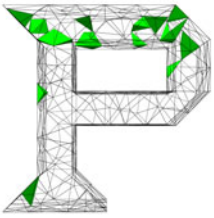
CUBE1K	min sine: 11 sec	CUBE10K	min sine: 165 sec
			
1,185 tetrahedra	1,204 tetrahedra	11,661 tetrahedra	11,355 tetrahedra
			
1,212 tetrahedra	1,200 tetrahedra	11,432 tetrahedra	11,504 tetrahedra
			
volume-length: 5 sec	biased min sine: 8 sec	volume-length: 58 sec	biased min sine: 100 sec
HOUSE	min sine: 36 sec	P	min sine: 19 sec
			
1,390 tetrahedra	1,735 tetrahedra	927 tetrahedra	1,159 tetrahedra
			
1,623 tetrahedra	1,751 tetrahedra	1,026 tetrahedra	1,106 tetrahedra
			
volume-length: 50 sec	biased min sine: 42 sec	volume-length: 25 sec	biased min sine: 15 sec

Table 5.3. Continuation of Table 5.2.

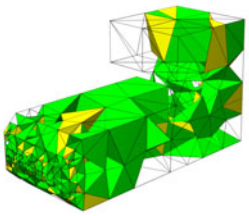
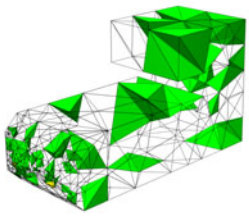
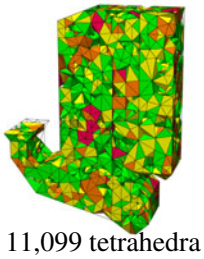
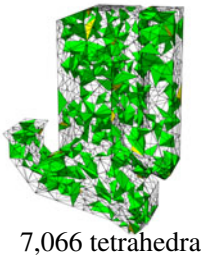
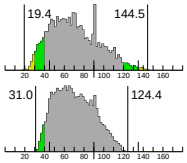
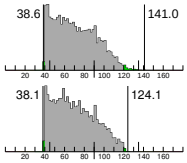
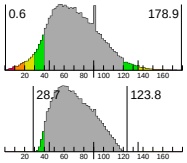
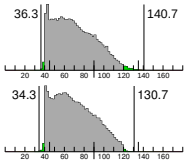
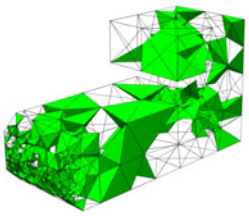
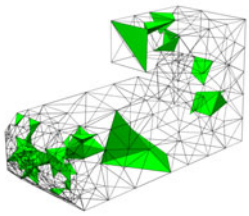
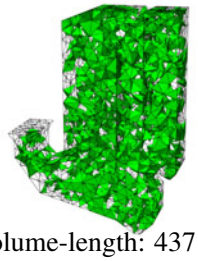
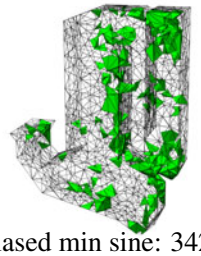
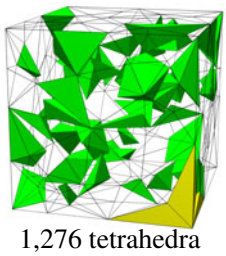
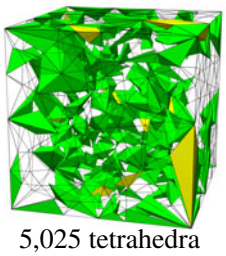
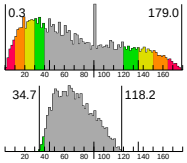
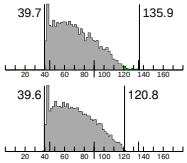
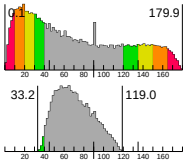
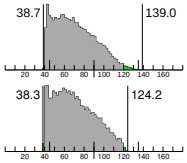
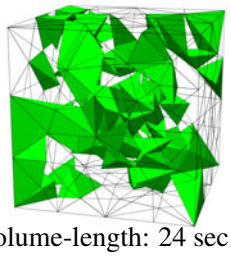
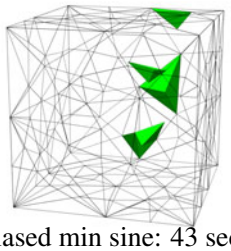
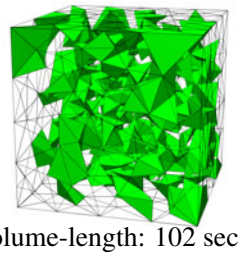
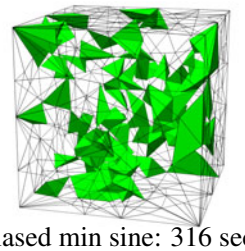
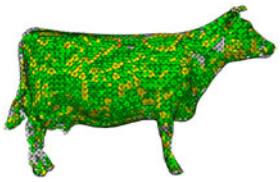
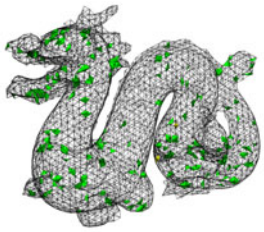
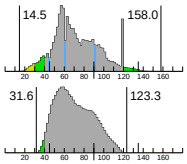
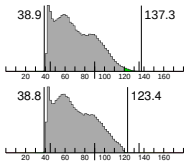
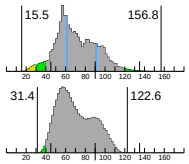
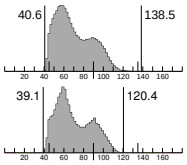
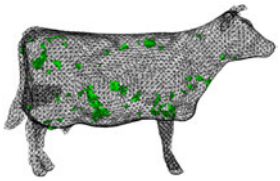
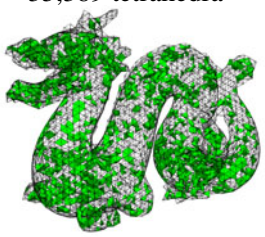
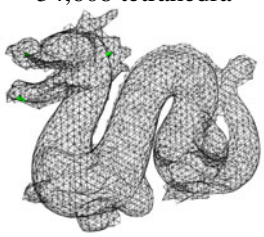
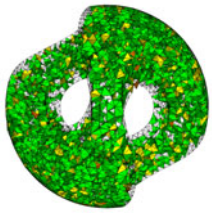
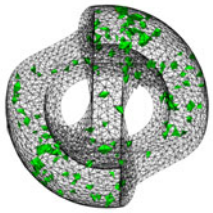
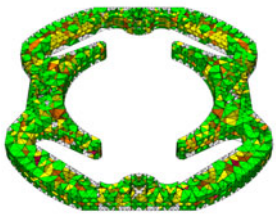
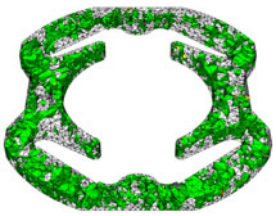
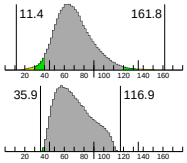
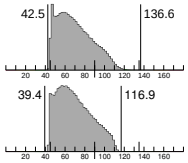
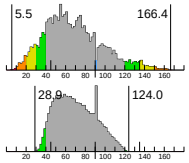
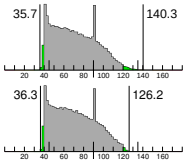
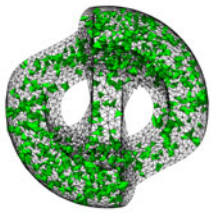
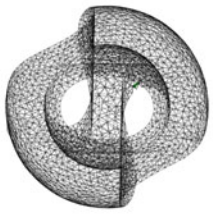
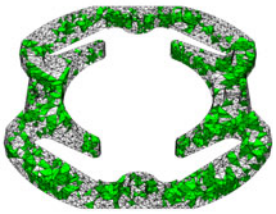
TFire	min sine: 29 sec	Tire	min sine: 163 sec
			
1,105 tetrahedra	1,332 tetrahedra	11,099 tetrahedra	7,066 tetrahedra
			
1,277 tetrahedra	1,380 tetrahedra	10,886 tetrahedra	11,675 tetrahedra
			
volume-length: 81 sec	biased min sine: 37 sec	volume-length: 437 sec	biased min sine: 342 sec
Rand1	min sine: 31 sec	Rand2	min sine: 234 sec
			
5,105 tetrahedra	1,276 tetrahedra	25,705 tetrahedra	5,025 tetrahedra
			
1,099 tetrahedra	1,233 tetrahedra	3,730 tetrahedra	4,263 tetrahedra
			
volume-length: 24 sec	biased min sine: 43 sec	volume-length: 102 sec	biased min sine: 316 sec

Table 5.4. Continuation of Tables 5.2 and 5.3. Blue histogram bars should have their heights multiplied by 20 to account for the fact that in the semi-structured meshes DRAGON and Cow, angles of 45°, 60°, and 90° occur with high frequency.

Cow	min sine: 812 sec	DRAGON	min sine: 747 sec
			
42,054 tetrahedra	42,318 tetrahedra	32,960 tetrahedra	34,672 tetrahedra
			
44,002 tetrahedra	42,563 tetrahedra	35,389 tetrahedra	34,608 tetrahedra
			
volume-length: 2,370 sec	biased min sine: 967 sec	volume-length: 752 sec	biased min sine: 645 sec
STGALLEN	min sine: 575 sec	STATOR	min sine: 588 sec
			
50,392 tetrahedra	48,562 tetrahedra	7,517 tetrahedra	12,148 tetrahedra
			
49,526 tetrahedra	49,365 tetrahedra	12,011 tetrahedra	12,715 tetrahedra
			
volume-length: 702 sec	biased min sine: 988 sec	volume-length: 777 sec	biased min sine: 659 sec

the mesh, the number of tetrahedra can change substantially. The largest increase I observed was 69% when improving the STATOR mesh using the biased minimum sine measure as the objective function. The largest decrease came when optimizing RAND2 for the volume-length objective, which reduced it to 14% its original size. This reduction is probably due to the fact that RAND2 is a lazy triangulation and contains many unnecessarily short edges that were collapsed by edge contraction. (A method for explicitly controlling the number of tetrahedra, by controlling their sizes, appears in Chapter 6.)

Tables 5.2–5.4 reveal a lot about the relative performance of the different objective functions. If the sole objective of mesh improvement is to eliminate small dihedral angles, the minimum sine measure (upper right in each box) is a good choice. The biased minimum sine measure (lower right in each box), which exaggerates the badness of obtuse dihedral angles, does a better job of eliminating the large angles while only sacrificing performance on the small angles a little. Most effective at removing large dihedral angles is the volume-length objective function (lower left in each box), although it does not perform as well against small angles as either the unbiased or biased minimum sine objective functions. It is the simplest measure to compute, although this is only sometimes reflected in the running times.

Tables 5.5–5.7 report the performance of each objective function (including the radius ratio, which does not appear in Tables 5.2–5.4) in optimizing each quality measure. The radius ratios and volume-length measures are normalized so that equilateral tetrahedra have a quality of 1 and degenerate tetrahedra have a quality of 0.

Tables 5.5 and 5.6 illustrate the effects of using each of the four quality measures as objective functions on the RAND1 and STGALLEN meshes. Tables for the other ten test meshes appear in Appendix A. In the meshes optimized for the minimum sine or biased minimum sine objective, where the tetrahedra are colored according to their volume-length measures, the colored tetrahedra are spires (see Section 2.2), having good dihedral angles but poor volume-length measures. Looking to the right, you can see that the volume-length objective eliminates them entirely, but the radius ratio objective does not. These trends persist across the twelve test meshes, as Table 5.7 shows. The volume-length objective's strong performance in both controlling dihedral angles and eliminating spires makes it the best all-around objective function.

As an objective function, the radius ratio measure underperforms every other quality measure. It does a worse job of eliminating small dihedral angles than any other measure, though it sometimes beats the unbiased minimum sine measure in reducing the maximum dihedral angle. As expected, it does a better job of optimizing the radius ratio than the minimum sine measures, but the volume-length measure does a better job still. As Table 5.7 illustrates, the volume-length objective improves the worst radius ratio to at least 0.56, whereas the radius ratio objective left behind many worse tetrahedra, the worst with a radius ratio of 0.31. The radius ratio behaves poorly as an objective function because the circumscribing radius is an unstable function of vertex position. I see little reason to recommend the radius ratio as an objective, considering the superior performance and relative simplicity of the others.

5.2 Mesh improvement as a function of running time

Tables 5.8–5.10 illustrate some properties of the mesh improvement process as a function of running time. In all these examples, the objective function is the biased minimum sine measure. The first column of these tables shows how the minimum sine of each mesh improves over time. Although the minimum sine is a noisy indicator of overall mesh quality, it reveals some interesting properties of the mesh improvement process. Most meshes achieve 80% or more of their total improvement within the first 20% of running time. This rapid initial improvement reflects the early effectiveness of vertex smoothing and topological improvement operations. As improvement continues, the focus shifts to the few bad tetrahedra that remain, which often can be eliminated only through more costly composite operations (edge contraction and vertex insertion). By halting the schedule at a minimum quality of, for example, $\sin 30°$, users could reduce the running time by 80% or more.

The second column of Tables 5.8–5.10 shows the proportion of cumulative running time accounted for by vertex smoothing, topological operators (edge and face removal), edge contraction, and vertex insertion. Smoothing and topological improvement are usually responsible for most of the initial mesh improvement, and they run much faster than edge contraction and vertex insertion. After the initial rush of progress, the slower composite operations—edge contraction and vertex

Table 5.5. The RAND1 mesh optimized with four different quality measures as objective functions. The histograms tabulate, from top to bottom, dihedral angles, volume-length measures $6\sqrt{2}V/\ell_{\text{rms}}^3$, and radius ratios (times 3). The two ratios are normalized so an equilateral tetrahedron has quality 1 and a degenerate tetrahedron has quality 0. Above and below the dihedral angle histogram, the tetrahedra are colored by minimum acute and maximum obtuse dihedral angle, respectively. Above the $6\sqrt{2}V/\ell_{\text{rms}}^3$ and radius ratio histograms, tetrahedra are colored by their ratios.

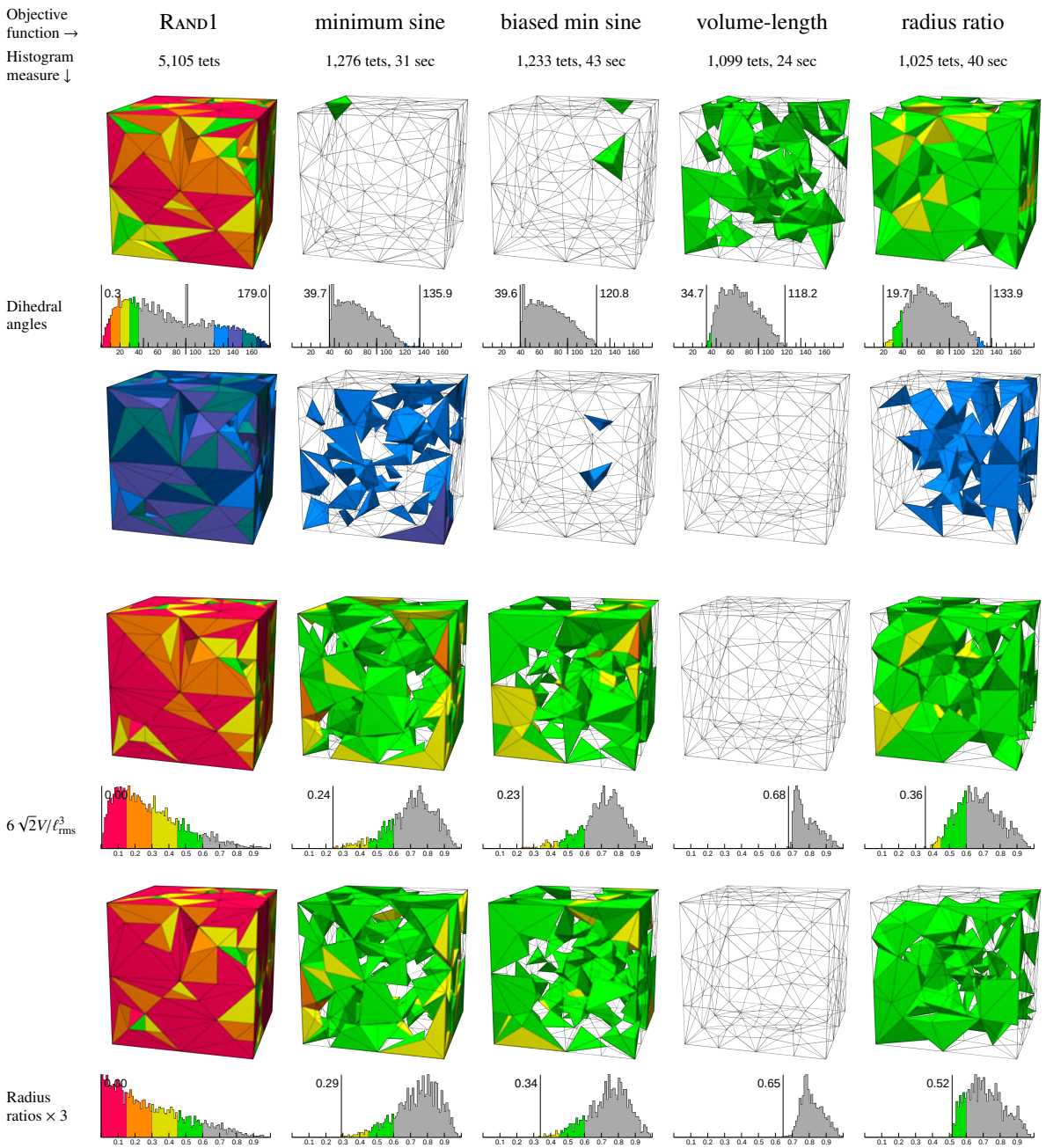


Table 5.6. The StGallen mesh optimized with four different quality measures as objective functions. The histograms tabulate, from top to bottom, dihedral angles, volume-length measures $6\sqrt{2}V/\ell_{\text{rms}}^3$, and radius ratios (times 3). The two ratios are normalized so an equilateral tetrahedron has quality 1 and a degenerate tetrahedron has quality 0.

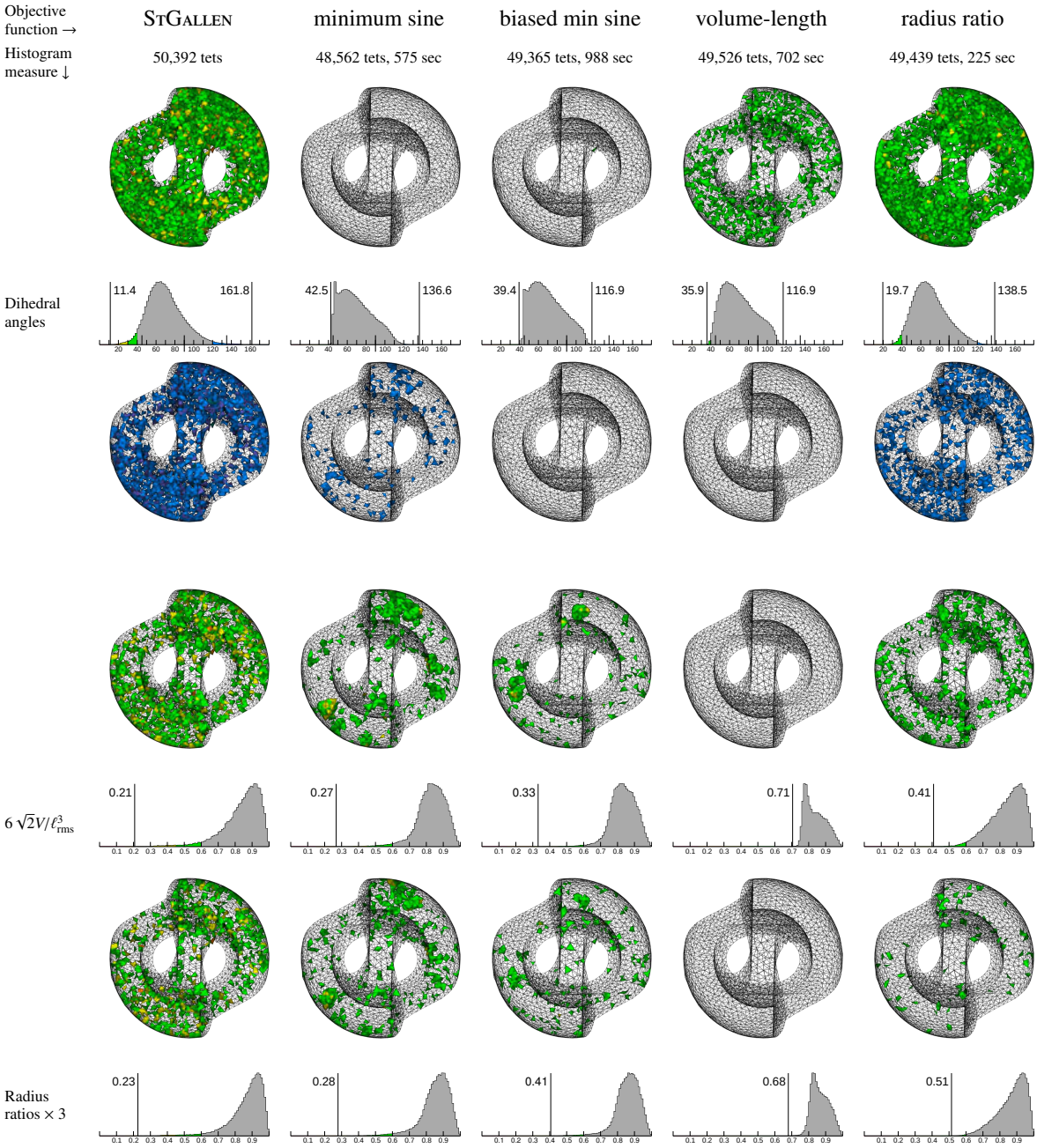
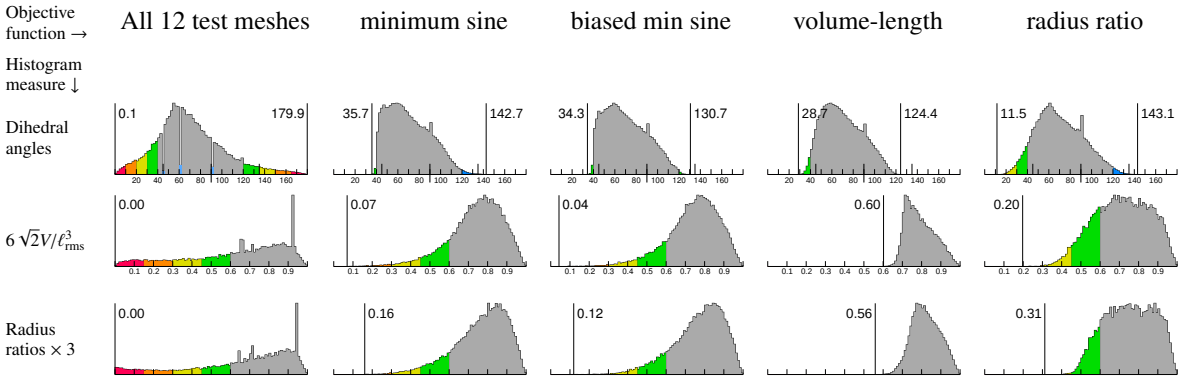


Table 5.7. A comparison of optimization of all twelve test meshes with four different quality measures as objective functions. The histograms tabulate, from top to bottom, dihedral angles, volume-length measures $6\sqrt{2}V/\ell_{\text{rms}}^3$, and radius ratios (times 3). These quantities are normalized so an equilateral tetrahedron has quality 1 and degenerate tetrahedron has quality 0.



insertion—quickly come to dominate the running time. For most meshes, the composite operations ultimately account for more than 75% of the total running time. Vertex insertion is usually more costly than edge contraction, except for RAND1.

The third column of Tables 5.8–5.10 plots the success rate of improvement operations as a function of running time. Smoothing is most often successful, justifying its presence in the outermost loop of the improvement schedule (see Listing 4.5). Insertion succeeds in about 10% of attempts, making it the next most successful operator. Despite its relatively high frequency of success, the schedule does not try insertion often because it accounts for such a large proportion of running time. Topological operations are less often successful than smoothing or insertion, but they run so quickly that there is no harm in attempting them more frequently than the composite operations. Edge contraction is least often successful, succeeding only once every 1,000 to 10,000 attempts or less for most of the meshes. (The rate for STGALLEN is so low that it does not appear in the plot.) Two notable exceptions are RAND1 and RAND2, which both have edge contraction success rates over 1%, because their random vertex positions induce many short edges.

5.3 Which mesh improvement operations are most important?

Table 5.11 explores the effects of switching on combinations of improvement operations in order to explore the question: which operations are the most important to have if the programmer's

Table 5.8. Mesh improvement statistics as a function of running time for the twelve test meshes (continued in Tables 5.9 and 5.10). The objective function is the biased minimum sine measure. The first column graphs the minimum sine as a function of running time. The second column graphs the proportion of cumulative running time spent on the four types of improvement operations as a function of running time. The red area is the proportion of time spent on vertex insertion, cyan is edge contraction, green is edge and face removal, and blue is vertex smoothing. The third column graphs the cumulative success rate (on a logarithmic scale) for each type of operation as a function of running time.

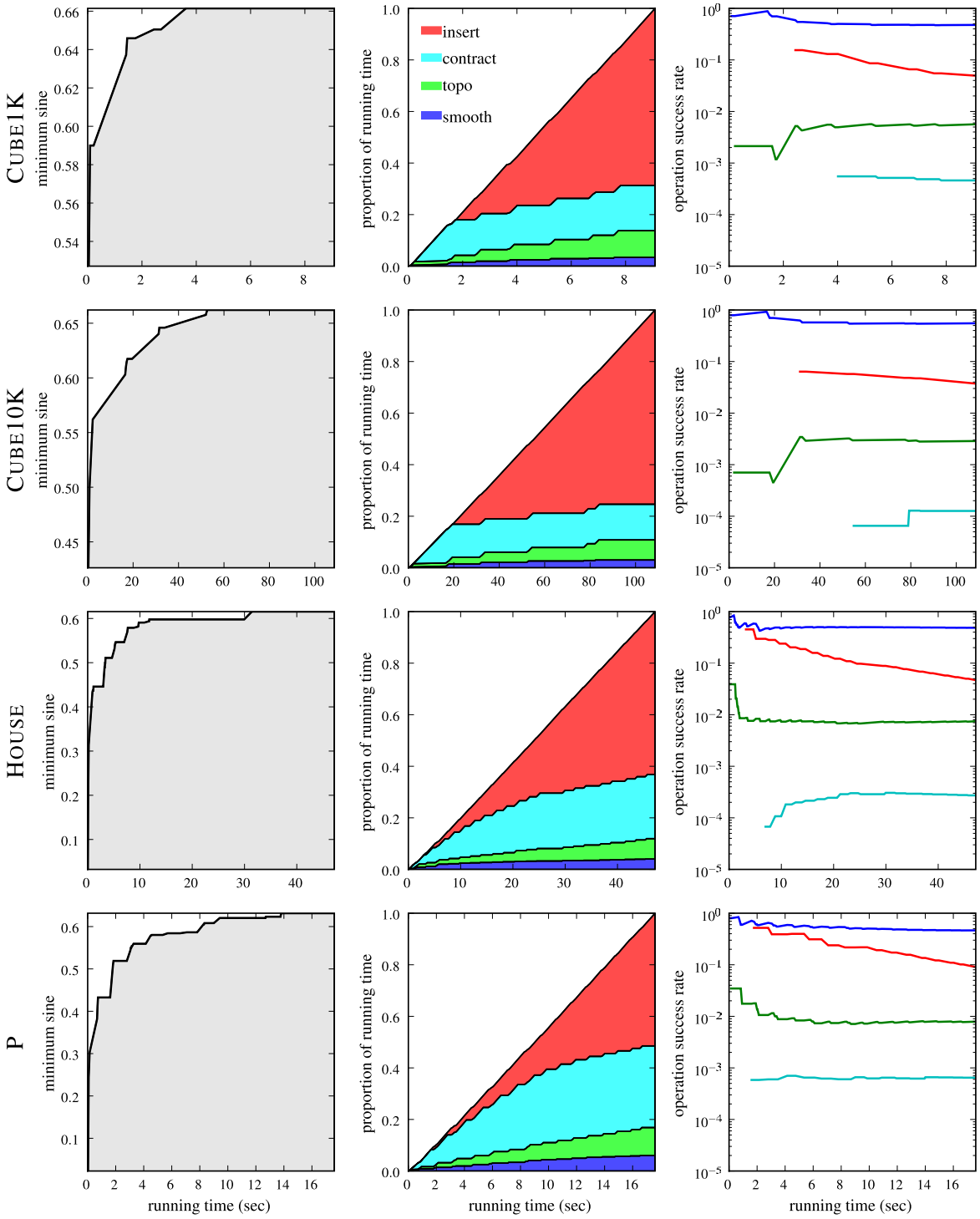


Table 5.9. Continuation of Table 5.8.

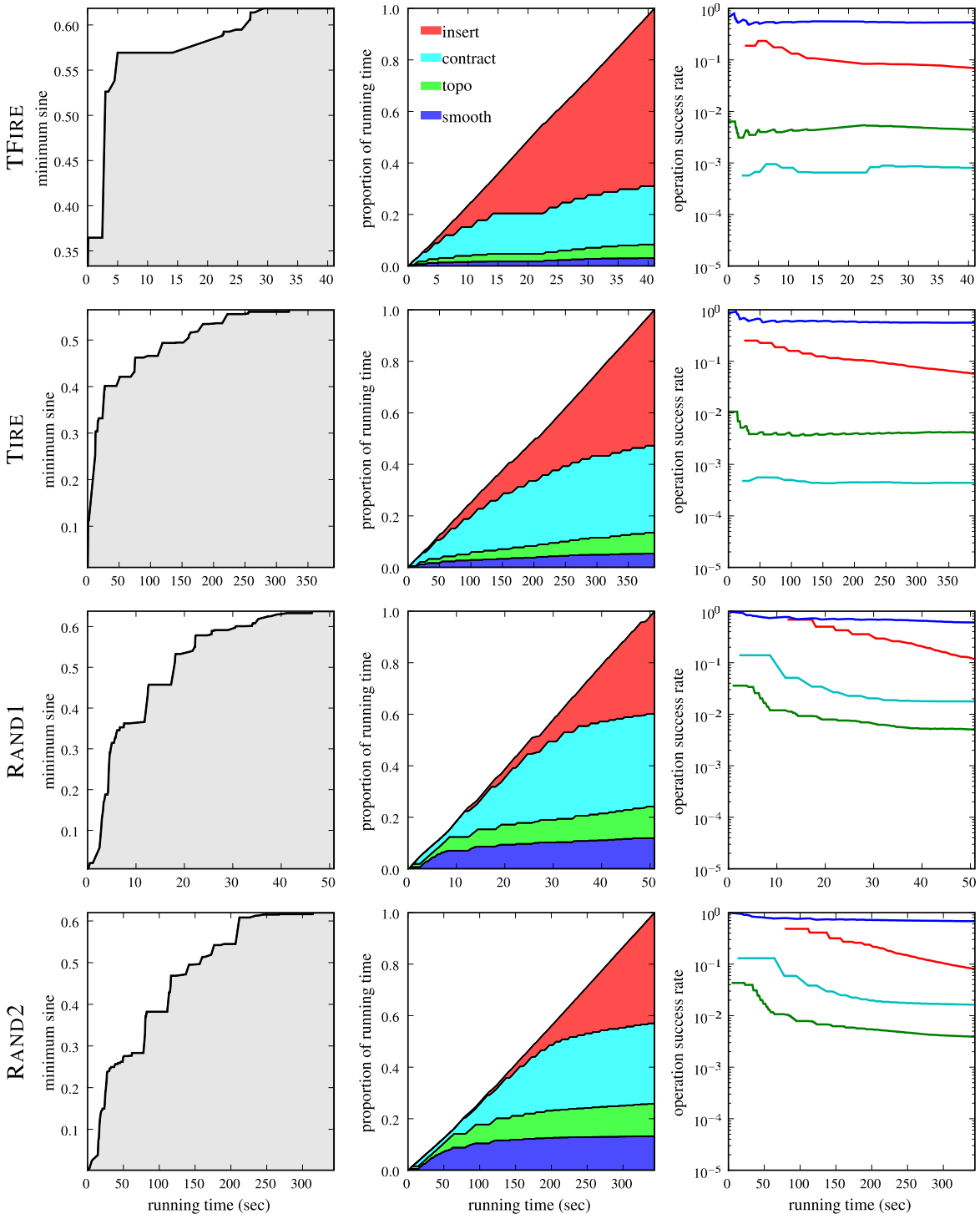
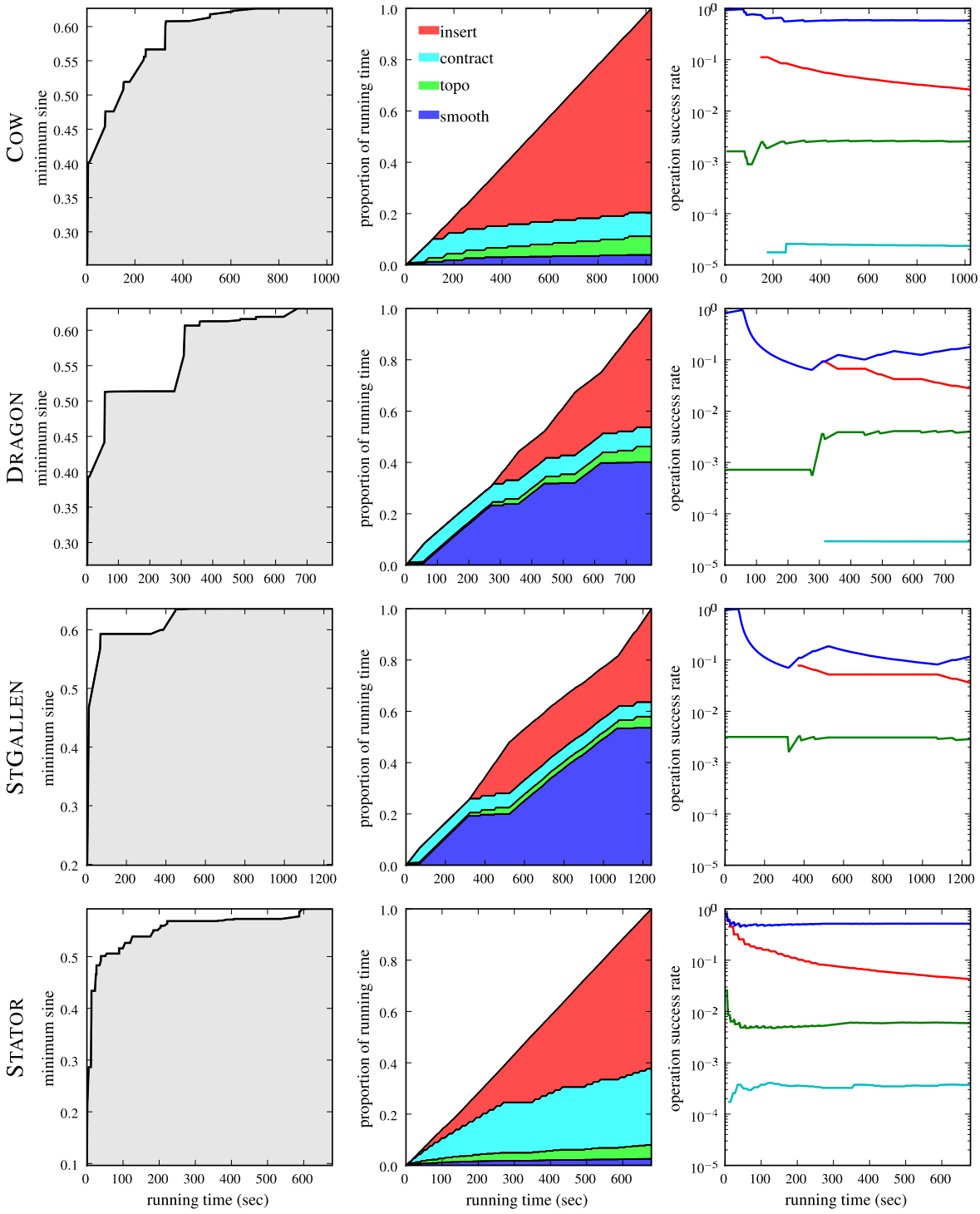


Table 5.10. Continuation of Tables 5.8 and 5.9.



time is limited? I try all combinations of three operations: optimization-based vertex smoothing in the interior of the mesh (but not on mesh boundaries); vertex insertion in the interior of the mesh (but not on boundaries); and edge removal (but no other topological transformations).

Smoothing proves to be the most indispensable; substantial progress is almost impossible without it. Vertex insertion is the second-most powerful operation. I was surprised to see that it alone can substantially improve some meshes, even though vertex insertion operations fail far more frequently when neither smoothing nor other topological transformations are available to improve the cavity after insertion (as described in Section 3.4.2). Vertex insertion comes at the cost of a large running time. Edge removal is the least beneficial of the three operations in isolation.

Edge removal and vertex insertion together are less effective than smoothing alone. Combining smoothing with edge removal, however, provides a significant boost over any single operation. Smoothing and vertex insertion together are even more powerful. Implementing all three features provides another modest bump in mesh quality. The addition of all the other operations, such as multi-face removal, edge contraction, and especially operations that modify the mesh boundary (boundary smoothing and vertex insertion) provides another significant boost.

Table 5.3 examines the effects of selectively disabling one or a few operations while keeping all the others. The loss of smoothing is the most disastrous, followed by vertex insertion. Switching off all the vertex-preserving topological transformations ("No edge or face removal") typically worsens the extreme angles by about 5°, which is significant but less than I expected. The effect of switching off edge removal and the effect of switching off face removal seem to be additive—the two types of topological transformations are not redundant.

Disabling changes to the mesh boundary hurts mesh quality more than disabling any operation besides smoothing or vertex insertion. Even just disabling vertex insertion on the boundary is more deleterious than disabling edge and face removal. Because the worst, hardest-to-remove tetrahedra in a mesh lie at or near the boundary, success often hinges on the ability to improve the boundary triangulation.

Sometimes, disabling a feature increases running time, which is counterintuitive. For example, disabling edge contraction increases the total running time for the twelve test meshes from 4,162

Table 5.11. Histograms showing the dihedral angle distributions, and minimum and maximum dihedral angles, for several meshes optimized with only selected improvement features switched on. The objective function is the biased minimum sine measure. Multiply the heights of the blue histogram bars by 20. Running times appear under each histogram.

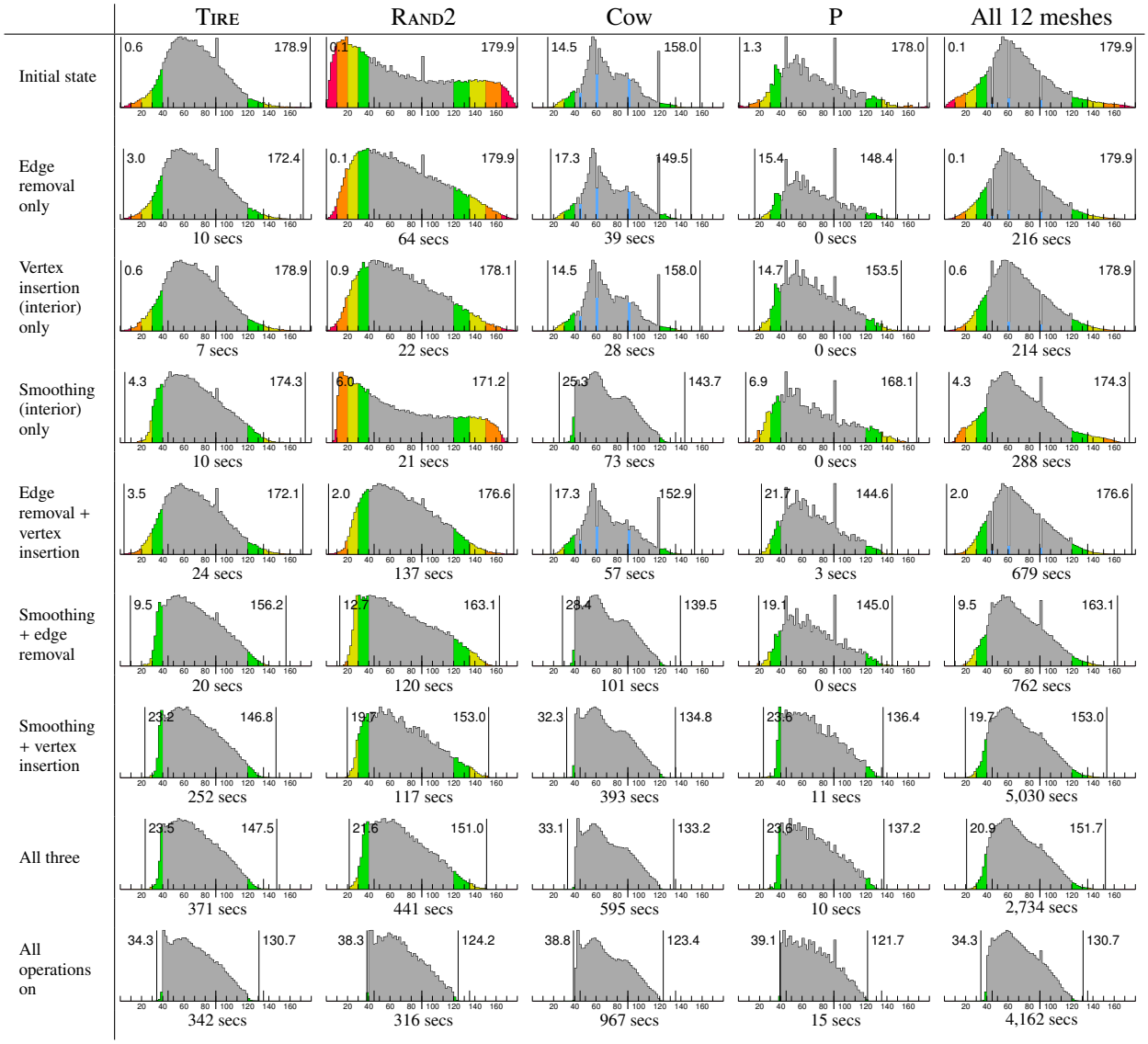
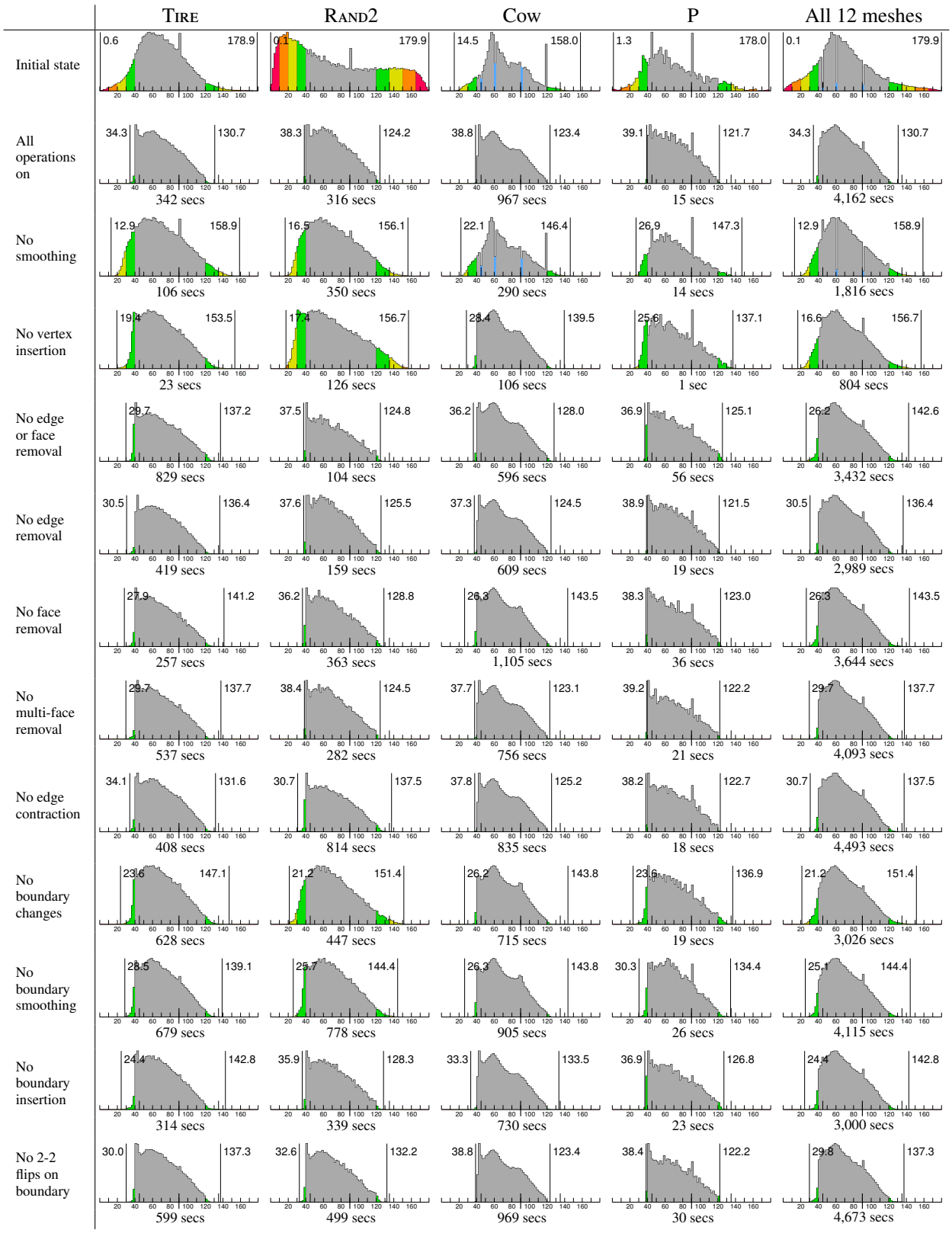


Table 5.12. (Next page.) Histograms showing the dihedral angle distributions, and minimum and maximum dihedral angles, for several meshes optimized with selected improvement operations disabled. The objective function is the biased minimum sine measure. Multiply the heights of the blue histogram bars by 20. Running times appear under each histogram. "No face removal" means that 2-2 flips, 2-3 flips, and multi-face removal are all disabled. "No boundary smoothing" means no constrained vertex smoothing nor quadric smoothing.


TIRE
RAND2
Cow
P
All 12 meshes
Initial state
0.6 178.9
0.1 179.9
14.5 158.0
1.3 178.0
0.1 179.9
All operations on
34.3 130.7
342 secs
38.3 124.2
316 secs
38.8 123.4
967 secs
39.1 121.7
15 secs
34.3 130.7
4,162 secs
No smoothing
12.9 158.9
106 secs
16.5 156.1
350 secs
22.1 146.4
290 secs
26.9 147.3
14 secs
12.9 158.9
1,816 secs
No vertex insertion
19.4 153.5
23 secs
17.4 156.7
126 secs
28.4 139.5
106 secs
25.6 137.1
1 sec
16.6 156.7
804 secs
No edge or face removal
29.7 137.2
829 secs
37.5 124.8
104 secs
36.2 128.0
596 secs
36.9 125.1
56 secs
26.2 142.6
3,432 secs
No edge removal
30.5 136.4
419 secs
37.6 125.5
159 secs
37.3 124.5
609 secs
38.9 121.5
19 secs
30.5 136.4
2,989 secs
No face removal
27.9 141.2
257 secs
36.2 128.8
363 secs
26.3 143.5
1,105 secs
38.3 123.0
36 secs
26.3 143.5
3,644 secs
No multi-face removal
29.7 137.7
537 secs
38.4 124.5
282 secs
37.7 123.1
756 secs
39.2 122.2
21 secs
29.7 137.7
4,093 secs
No edge contraction
34.1 131.6
408 secs
30.7 137.5
814 secs
37.8 125.2
835 secs
38.2 122.7
18 secs
30.7 137.5
4,493 secs
No boundary changes
23.6 147.1
628 secs
21.2 151.4
447 secs
26.2 143.8
715 secs
23.6 136.9
19 secs
21.2 151.4
3,026 secs
No boundary smoothing
28.5 139.1
679 secs
25.7 144.4
778 secs
26.3 143.8
905 secs
30.3 134.4
26 secs
25.1 144.4
4,115 secs
No boundary insertion
24.4 142.8
314 secs
35.9 128.3
339 secs
33.3 133.5
730 secs
36.9 126.8
23 secs
24.4 142.8
3,000 secs
No 2-2 flips on boundary
30.0 137.3
599 secs
32.6 132.2
499 secs
38.8 123.4
969 secs
38.4 122.2
30 secs
29.8 137.3
4,673 secs

seconds to 4,493 seconds. There are a couple of possible explanations for this increase. First, edge contraction deletes vertices from the mesh, making it smaller so that further improvement runs faster. Second, some operations are better suited to removing certain kinds of bad tetrahedra than others, and so disabling an operation may mean that an element that could be easily removed by that operation must instead be removed through time-consuming combinations of other operations.

Chapter 6

Tetrahedron Size Control

All of the quality measures discussed in Section 2.1 are *size invariant*, meaning that uniformly scaling a tetrahedron has no effect on its quality. Consequently, the improvement process is agnostic about the size of a mesh's tetrahedra. Over the course of the improvement process, there may be significant changes to tetrahedron size and the number of tetrahedra in a mesh.

For example, consider the RAND2 input mesh shown in Figure 6.1a. This mesh was created by randomly inserting vertices into a cubical domain, yielding terrible tetrahedron quality. The mesh has both a bad spatial distribution of vertices and a bad choice of connectivity between these vertices. The schedule described in Section 4.4 removes many of the vertices by edge contraction and vertex insertion. The mesh produced this way (Figure 6.1b) has good quality, but also a 50% increase in median edge length.

Most applications of meshes are particular about how large the tetrahedra should be, because interpolation accuracy and computation time both increase as tetrahedron size shrinks. Users might want to improve meshes without such drastic changes in the tetrahedron sizes. Alternatively, users might desire tetrahedra substantially smaller or larger than their meshes possess. They may want to *refine* a mesh, reducing the average edge length, or *coarsen* it, increasing the average edge length. They may want a *graded* mesh that has smaller tetrahedra in some areas and larger tetrahedra in others. *Size control* is the process of matching the sizes of tetrahedra in a mesh with the demands of the user.

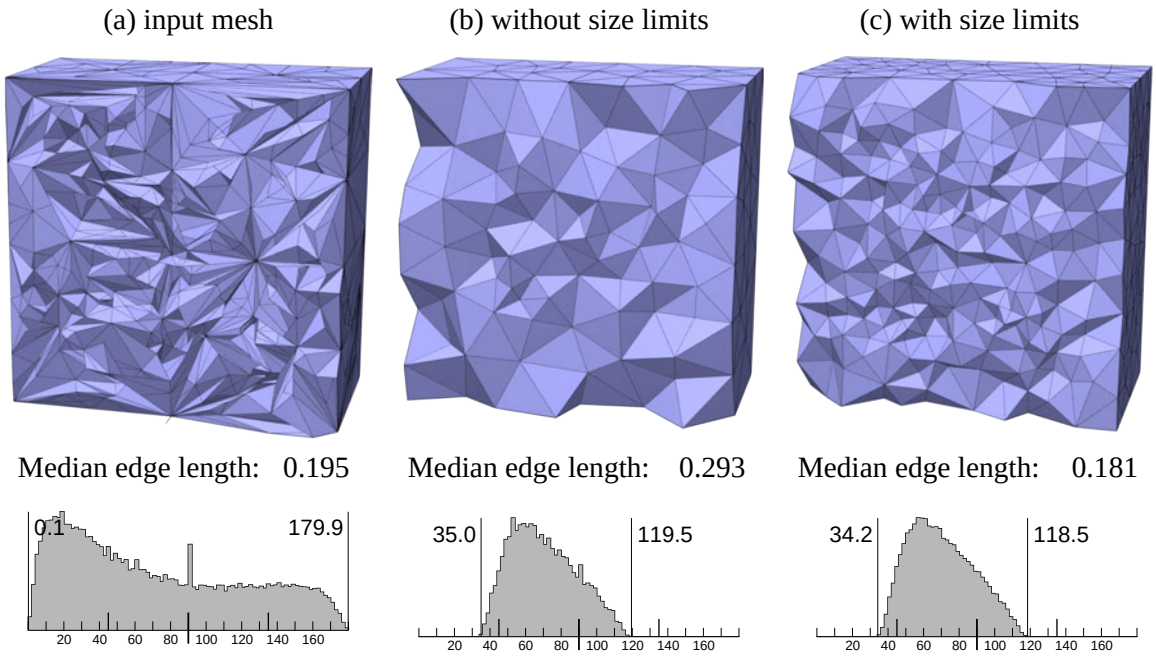


Figure 6.1. (a) A cutaway view of RAND2, a randomly triangulated input mesh with poor quality, with a histogram of its dihedral angles. (b) The mesh improvement schedule (Section 4.4), by optimizing the volume-length objective, significantly coarsens the mesh—its median edge length increases by 50%. (c) By constraining the lengths of the edges that operations are permitted to produce, coarsening is prevented. The schedule achieves high quality with just a 7.2% decrease in the median edge length.

6.1 How to make size control work

One way to control tetrahedron size is to fold tetrahedron size and quality into a single objective function, and optimize both simultaneously. This approach has many problems. First, factoring size into the objective function would entail the design of complicated quality measures and their gradient computations. The fact that the objective function must punish both too-large and too-small tetrahedra introduces a lot of complexity (which turns out to be unnecessary). Second, it would make it difficult for users to install their own favorite quality measures into my mesh improvement software. Third, smoothing, edge removal, and face removal do not change the number of vertices in the mesh, and so are ineffective for size control. Factoring size into the objective function for these operations would compromise quality. (One could use different objective functions for different operations, at the cost of losing the guarantee that hill-climbing optimization will not revisit the same mesh over and over.) Fourth, I find that it is most effective to coarsen through edge contractions and

to refine through vertex insertions. If substantial refinement or coarsening is requested, it proceeds much faster if the algorithm ignores tetrahedron quality until after the tetrahedron sizes have been correctly adjusted.

Because of these considerations, I propose that tetrahedron size should be measured and adjusted independently of tetrahedron shape.

How should the sizes of tetrahedra be measured? One possibility is to measure their volumes. But the volumes of tetrahedra fluctuate a lot, even if their vertices are regularly spaced. A better choice is to measure edge lengths, which are more stable, especially once the vertices achieve the nice spacing they invariably have in high-quality meshes. The goal of my size control schedule is to produce a mesh in which every edge is as close as possible to some ideal length. The ideal length may be a space-varying field, specifying a graded mesh. Edges that are too long must be shortened, and edges that are too short must be lengthened.

It is impossible to produce a mesh in which every edge is exactly ideal in length. We must define a range of acceptable edge lengths. The size control schedule terminates when all the edges are within the range of acceptable lengths. The wider this range is, the easier it is for the mesh improvement schedule to achieve high quality.

I control tetrahedron size in two ways. I add to the mesh improvement schedule a *size control phase* that runs before quality improvement starts. This phase enforces user-specified limits on edge length and performs all necessary refinement, coarsening, and grading of the mesh. If a mesh starts out with edges of the right length, the size control phase can be skipped. After mesh improvement begins, I reject mesh improvement operations that produce edges whose lengths fall outside the acceptable range. I discuss this second idea first, in the next section, then I describe the size control phase in Section 6.3.

6.2 Placing size limits on mesh improvement operations

Suppose a mesh starts with acceptable edge lengths. I have found that by disallowing improvement operations that create an edge of unacceptable length, I can effectively control the tetrahedron sizes with little sacrifice in quality, as long as the range of acceptable lengths is not too narrow.

Suppose the user specifies an ideal edge length of ℓ_{ideal}. Let s and l be two coefficients with $0 < s \leq 1$ and $l \geq 1$ that specify how much longer or shorter than ℓ_{ideal} edges are permitted to be. I constrain the length ℓ of any edge created or altered by mesh improvement operations to fall within the range

$$s\ell_{\text{ideal}} \leq \ell \leq l\ell_{\text{ideal}}. \tag{6.1}$$

I enforce this constraint by adding a check to the end of every improvement operation. If some edge created or changed by improvement falls outside the specified range, I reject the operation and revert the mesh to its state before the change.

Figure 6.1c shows the RAND2 mesh after improvement with size limits imposed. The ideal edge length ℓ_{ideal} is 0.195 (the input mesh's median edge length) and the coefficients that specify the acceptable range are $s = 1.0$ and $l = 2.5$. The final mesh is no coarser than the original, but its quality is roughly as high as the mesh improved without size control.

6.3 The size control phase

Simply constraining the edge lengths produced by improvement operations works well when an input mesh starts out with edges near the ideal length. Sometimes, though, a mesh has edges shorter or longer than those an application demands. In these circumstances, the improvement schedule begins with a size control phase that adjusts the edge lengths in the mesh before proceeding with mesh improvement. The size control phase is permitted to worsen the quality of the mesh, and thus it is faster than it would be if it were only permitted to use hill-climbing operations. This creates the risk that if the input mesh has very high quality, the output mesh might not be as good. But if a mesh has truly high quality, it is probably infeasible to substantially coarsen or refine it without a temporary drop in quality.

6.3.1 Controlling tetrahedron size with edge contraction and vertex insertion

The size control phase uses edge contraction operations to remove too-short edges and vertex insertion operations to remove too-long edges. These operators are described in Sections 3.3 and 3.4, respectively. When I use them for tetrahedron size control, I change them in just one way: I allow them to proceed even if they worsen the quality vector of the mesh. The criteria for performing an operation are that the too-long or too-short edge is removed from the mesh, and that no degenerate or inverted tetrahedra are created. To prevent an infinite loop of alternating coarsening and refinement, I also forbid any edge contraction operation that creates a too-long edge, and any vertex insertion operation that creates a too-short edge. (These should be infrequent if the range of acceptable edge lengths is wide enough.) Observe that the operations do not entirely ignore mesh quality; they still try to optimize the new configuration.

Suppose an edge e_{short} is too short. The size control phase attempts to remove it by invoking the algorithm EdgeContractSize, shown in Listing 6.1. The algorithm first tries to contract e_{short} without regard to the quality of the resulting tetrahedra, so long as it produces no degenerate or inverted tetrahedra. It is not always possible to contract e_{short}; it could create inverted tetrahedra as in Figure 3.11, or the endpoints of e_{short} may be fixed on the mesh boundary. If contraction fails, the algorithm attempts to contract each edge of the tetrahedra that share e_{short} until some contraction succeeds. Contracting any of these edges will have the side effect of eliminating e_{short} from the mesh. On rare occasions, e_{short} may survive all these efforts, in which case we hope that subsequent mesh transformations will make it possible to return and fix the edge later.

Suppose an edge e_{long} is too long. The size control phase attempts to remove it by invoking VertexInsertSize, shown in Listing 6.2. Vertex insertion proceeds much as described in Section 3.4.1 with an important modification to the cavity selection step: the vertex insertion operation is not permitted to delete any vertices from the mesh, thus guaranteeing that the operation will increase the number of vertices and not inadvertently coarsen the mesh. This constraint entails an easy change to the algorithm that selects the optimal cavity. Cavity improvement follows as usual, evening out the spacing of the vertices near the newly inserted vertex.

```
EdgeContractSize(M, e_short, ℓ_ideal, l)
{ e_short is a too-short edge to eliminate from a mesh M }
{ ℓ_ideal is the ideal edge length }
{ l is the coefficient specifying an upper bound on edge lengths }
1     Attempt to contract e_short (Section 3.3), disregarding the resulting tetrahedron quality.
2     if the contraction succeeded and the longest changed edge length ≤ lℓ_ideal
3         return success.
4     Reverse the edge contraction.
5     for each edge e of the tetrahedra sharing e_short
6         Attempt to contract e, disregarding the resulting tetrahedron quality.
7         if the edge contraction succeeded and the longest changed edge length ≤ lℓ_ideal
8             return success.
9         Reverse the edge contraction.
10    return failure.
```

Listing 6.1: Pseudocode for EdgeContractSize, which attempts to eliminate an edge e_{short} from a mesh with quality-insensitive edge contraction. An edge contraction succeeds if it can be performed without changing the domain shape or creating degenerate or inverted tetrahedra.

```
VertexInsertSize(M, e_long, ℓ_ideal, s)
{ e_long is a too-long edge to eliminate from a mesh M }
{ ℓ_ideal is the ideal edge length }
{ s is the coefficient specifying a lower bound on edge lengths }
1     p ⇐ the midpoint of e_long
2     I ⇐ deleted tetrahedra, computed as in Section 3.4.1 with the constraint that
          no mesh vertex lies in the cavity's interior.
3     J ⇐ replacement tetrahedra, which adjoin a new vertex at p
4     Replace I with the new tetrahedra J.      { see Listing 3.1 }.
5     ImproveCavity(p, J, M)      { see Listing 3.2 }
6     if the shortest new/changed edge length ≥ sℓ_ideal
7         return success.
8     Reverse the vertex insertion.
9     return failure.
```

Listing 6.2: Pseudocode for VertexInsertSize, which attempts to eliminate the edge e_{long} from the mesh using quality-insensitive vertex insertion.

6.3.2 The size control schedule

The schedule for the size control phase is listed in Listing 6.3. It begins with a global pass of smoothing (described in Section 4.2.1) and a global pass of topological improvement (described in

```
SizeControl(M, ℓ_ideal, s, l)
{ M is the entire mesh }
{ ℓ_ideal is the ideal edge length }
{ s is the coefficient specifying a lower bound on edge lengths }
{ l is the coefficient specifying an upper bound on edge lengths }
1    SmoothingPass(M, M)      { See Section 4.2.1 }
2    TopologicalPass(M, M)    { See Section 4.2.2 }
3    ℓ_max ⇐ length of the longest edge in the mesh
4    ℓ_min ⇐ length of the shortest edge in the mesh
5    while ℓ_max > lℓ_ideal or ℓ_min < sℓ_ideal
6        S ⇐ a list of edges in M shorter than sℓ_ideal
7        for each edge e ∈ S that still exists and is still shorter than sℓ_ideal
8            EdgeContractSize(M, e, ℓ_ideal, l)
9        L ⇐ a list of edges in M longer than lℓ_ideal
10       for each edge e ∈ L that still exists and is still longer than lℓ_ideal
11           VertexInsertSize(M, e, ℓ_ideal, s)
12       ℓ'_max ⇐ length of the longest edge in the mesh
13       ℓ'_min ⇐ length of the shortest edge in the mesh
14       if ℓ'_max ≥ ℓ_max and ℓ'_min ≤ ℓ_min
15           SmoothingPass(M, M)      { See Section 4.2.1 }
16           TopologicalPass(M, M)    { See Section 4.2.2 }
17       ℓ_max ⇐ ℓ'_max
18       ℓ_min ⇐ ℓ'_min
```

Listing 6.3: Pseudocode for SizeControl, the schedule for controlling tetrahedron sizes in a mesh.

Section 4.2.2). I start with these passes to help improve the distribution of vertices in the mesh and their connectivity before I identify which edges are excessively long or short.

Next, the schedule repeatedly tries to contract all the short edges and split all the long edges until all the remaining edges are in the acceptable range $[s\ell_{\text{ideal}}, l\ell_{\text{ideal}}]$. If the edge contraction and vertex insertion passes fail to improve the shortest and longest edges in the mesh (line 14), I perform another global pass of smoothing and topological improvement. These additional passes help to improve the global vertex distribution and its connectivity, which in practice improves the success of the size control operations.

Figure 6.2 shows examples of mesh coarsening and refinement performed by SizeControl. In both cases, s is 0.5 and l is 1.5. These coefficients allow for edge lengths from half as short to one and a half times as long as ℓ_{ideal}. More leeway is given for short edges than long edges because the topological constraints on edge contraction make it fail more frequently. Figure 6.2a shows an

input mesh with 11,661 tetrahedra and a median edge length of 0.912. Figure 6.2b shows the mesh after coarsening with $\ell_{\text{ideal}} = 1.824$, twice the original median. The final mesh has 1,611 tetrahedra (13.8% of the original number, which is close to the 12.5% that is expected with a doubling of edge length) and a median edge length of 1.77. Figure 6.2c shows the mesh after refinement with $\ell_{\text{ideal}} = 0.456$, half the original median. This mesh has 85,335 tetrahedra (7.32 times the original number; an eight-fold increase is expected with a halving of edge length) and a median edge length of 0.483.

SizeControl can produce graded meshes by letting ℓ_{ideal} be a function of spatial position. Figure 6.3 shows two examples of grading. In Figure 6.3b, ℓ_{ideal} is larger on the left side of the mesh and smaller on the right. In Figure 6.3c, ℓ_{ideal} is smaller near the center of the mesh and larger near the edges.

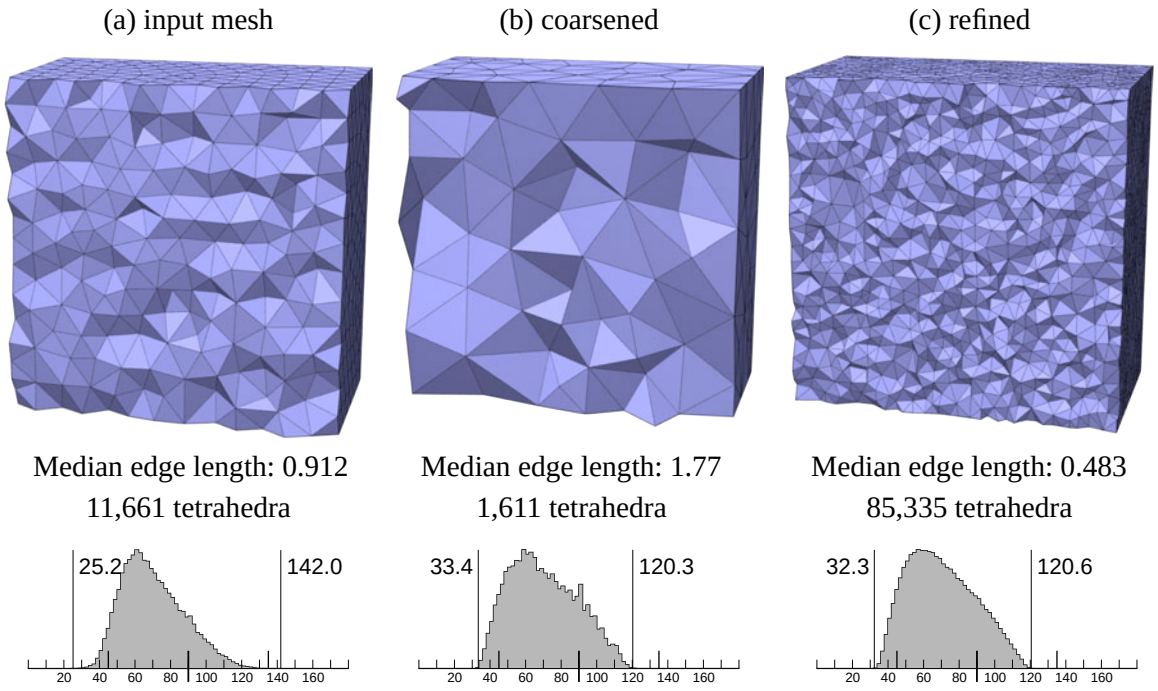


Figure 6.2. (a) A cutaway view of a high-quality cube mesh and a histogram of its dihedral angles. (b) Size control with an ideal edge length of twice the original median coarsens the mesh. (c) Size control with an ideal edge length of half the original median refines the mesh.

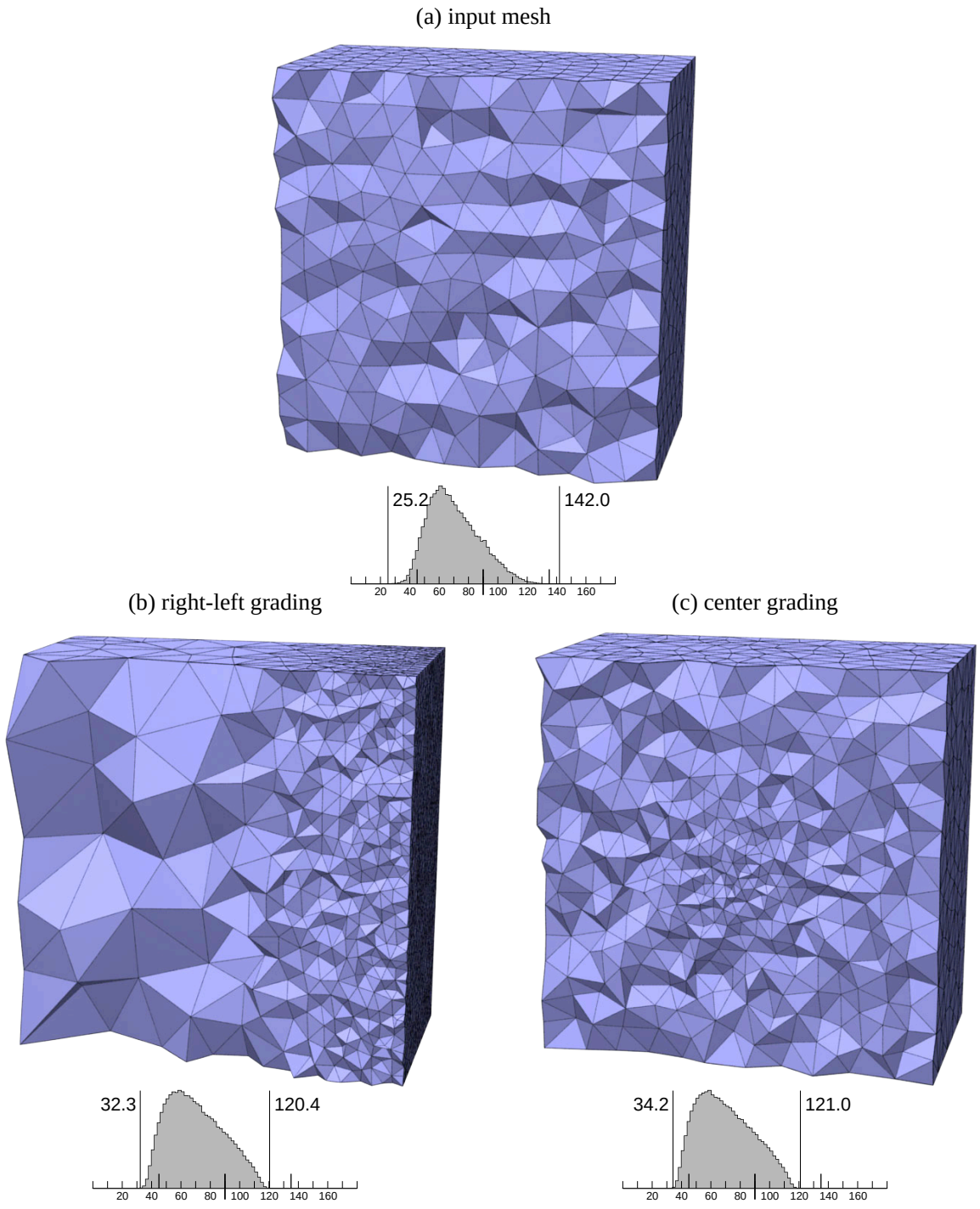


Figure 6.3. (a) A cutaway view of a cube mesh. (b) With a large ideal edge length on the left and a small one on the right, the mesh grades from large to small tetrahedra. (c) Specifying a smaller ideal edge length at the center of the mesh yields grading from small tetrahedra in the middle to large ones near the boundary.

Chapter 7

Anisotropy

Most mesh generation algorithms try to create tetrahedra that are close to equilateral. But for some applications of interpolation and numerical modeling, the mesh must be *anisotropic*: having long, skinny tetrahedra with orientations and eccentricities dictated by the curvature of the function being approximated [2; 29; 42]. In particular, anisotropic tetrahedra are essential for modern aerodynamics simulations. For example, laminar air flow over an airplane wing is best modeled with extremely thin slab-shaped elements aligned with the surface of the wing. By providing high resolution in a direction transverse to the wing, along which the velocity field has large second derivatives, but low resolution along the wing, thereby reducing the number of tetrahedra, an anisotropic mesh makes it computationally tractable to simulate a very high-resolution physical phenomenon.

The ideal orientations and eccentricities of the tetrahedra vary (usually smoothly) from one point in space to another, making anisotropic meshing more difficult than the traditional mesh generation problem. I find that, with only minor changes, my mesh improvement schedule can produce high-quality anisotropic tetrahedra.

7.1 Isotropic space and tetrahedron quality

All of the quality measures described in Section 2.1 reward isotropic tetrahedra. D'Azevedo extends the classical quality measures to anisotropic tetrahedra by affinely mapping tetrahedra to

an *isotropic space* before measuring their quality [17]. A tetrahedron is ideal if it is equilateral in isotropic space.

A user of my mesh improvement software specifies the desired anisotropy by providing a 3×3 symmetric positive definite *scaling tensor* $\mathbf{E}$ that affinely maps a tetrahedron t in physical space to a tetrahedron t' in isotropic space. If the vertices of t are $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$, and $\mathbf{v}_4$ (in physical space), then the vertices of t' in isotropic space are $\mathbf{E}\mathbf{v}_1$, $\mathbf{E}\mathbf{v}_2$, $\mathbf{E}\mathbf{v}_3$, and $\mathbf{E}\mathbf{v}_4$. The user's preferred isotropic quality measure then evaluates the quality of t', and this quality is assigned to t in physical space.

The simplest interpretation of $\mathbf{E}$ is as a linear transformation that prescribes that space is stretched or squashed along three mutually orthogonal unit vectors $\mathbf{u}_1$, $\mathbf{u}_2$, and $\mathbf{u}_3$ by scaling factors of S_1, S_2, and S_3. The scaling tensor $\mathbf{E}$ is

$$\mathbf{E} = \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \mathbf{u}_3 \end{bmatrix} \begin{bmatrix} S_1 & 0 & 0 \\ 0 & S_2 & 0 \\ 0 & 0 & S_3 \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \mathbf{u}_3 \end{bmatrix}^{\mathsf{T}}. \tag{7.1}$$

Readers familiar with eigenanalysis will recognize Equation 7.1 as the eigendecomposition of $\mathbf{E}$.

Anisotropy and size control (Chapter 6) work together elegantly. The size control algorithm simply applies the linear map $\mathbf{E}$ to each edge before measuring its length and comparing it against the ideal edge length ℓ_{ideal}, which is specified in isotropic space.

The scaling tensor $\mathbf{E}$ can be constant for an entire mesh, but usually it is a space-varying field $\mathbf{E}(\mathbf{x})$, because the ideal tetrahedron eccentricity and orientation varies according to the curvature of a function being interpolated. When transforming a tetrahedron or edge, it is important to use the same value of $\mathbf{E}$ to map each vertex to isotropic space; otherwise, for instance, the transformation might invert a tetrahedron. When my implementation transforms a tetrahedron (to judge its quality), it uses the value of $\mathbf{E}$ at its barycenter; and when it transforms an edge (to judge its length), it uses the value at the midpoint.

7.2 Examples of anisotropic meshes

Several examples illustrate the effect of the scaling tensor $\mathbf{E}$ on the shapes of tetrahedra. Figure 7.1 shows the effect of several scaling tensor fields on a unit cube mesh centered at the origin

with about 10,000 tetrahedra. In the figure, the histograms show the dihedral angles measured in isotropic space, not physical space.

In Figure 7.1a, we see the original mesh, which is the input. Figure 7.1b shows the effect of passing the mesh through size control and quality improvement with a constant scaling tensor

$$\mathbf{E}_b = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

This tensor makes distances in the x-direction three times longer in isotropic space than in physical space. For the tetrahedra to be equilateral in isotropic space, they must have edges in physical space one third as long in the x-direction than those of the input. The "squished" tetrahedra of the output are proportioned like frisbees: short along one dimension, long along two.

Figures 7.1c and 7.1d show the effect of position-dependent scaling tensor *fields*. Let the position-dependent scale factor be

$$S_c(\mathbf{x}) = \frac{1}{|\mathbf{x}| + 0.5}.$$

The scaling tensor field for Figure 7.1c is

$$\mathbf{E}_c(\mathbf{x}) = \begin{bmatrix} S_c(\mathbf{x}) & 0 & 0 \\ 0 & S_c(\mathbf{x}) & 0 \\ 0 & 0 & S_c(\mathbf{x}) \end{bmatrix}.$$

This field produces edges that are smaller in physical space as the magnitude of $\mathbf{x}$ decreases. A similarly constructed field produces a mesh where the scale is smaller on the left and larger on the right for Figure 7.1d.

Even more flexibility is possible if both the direction and the scale vary with position. Figure 7.2 illustrates the effect of scaling tensor fields on a unit sphere mesh centered at the origin with about 10,000 tetrahedra. At the top is the original input mesh. Define the position dependent scale factor

$$S_s(\mathbf{x}) = 1 + \sqrt{|\mathbf{x}|}.$$

This factor increases with the square root of the magnitude of $\mathbf{x}$. Define a unit vector that points radially outward from the origin

$$\mathbf{u}_r(\mathbf{x}) = \frac{\mathbf{x}}{|\mathbf{x}|},$$

with additional mutually orthogonal unit vectors $\mathbf{u}_s$ and $\mathbf{u}_t$ to complete the basis. The scaling tensor field

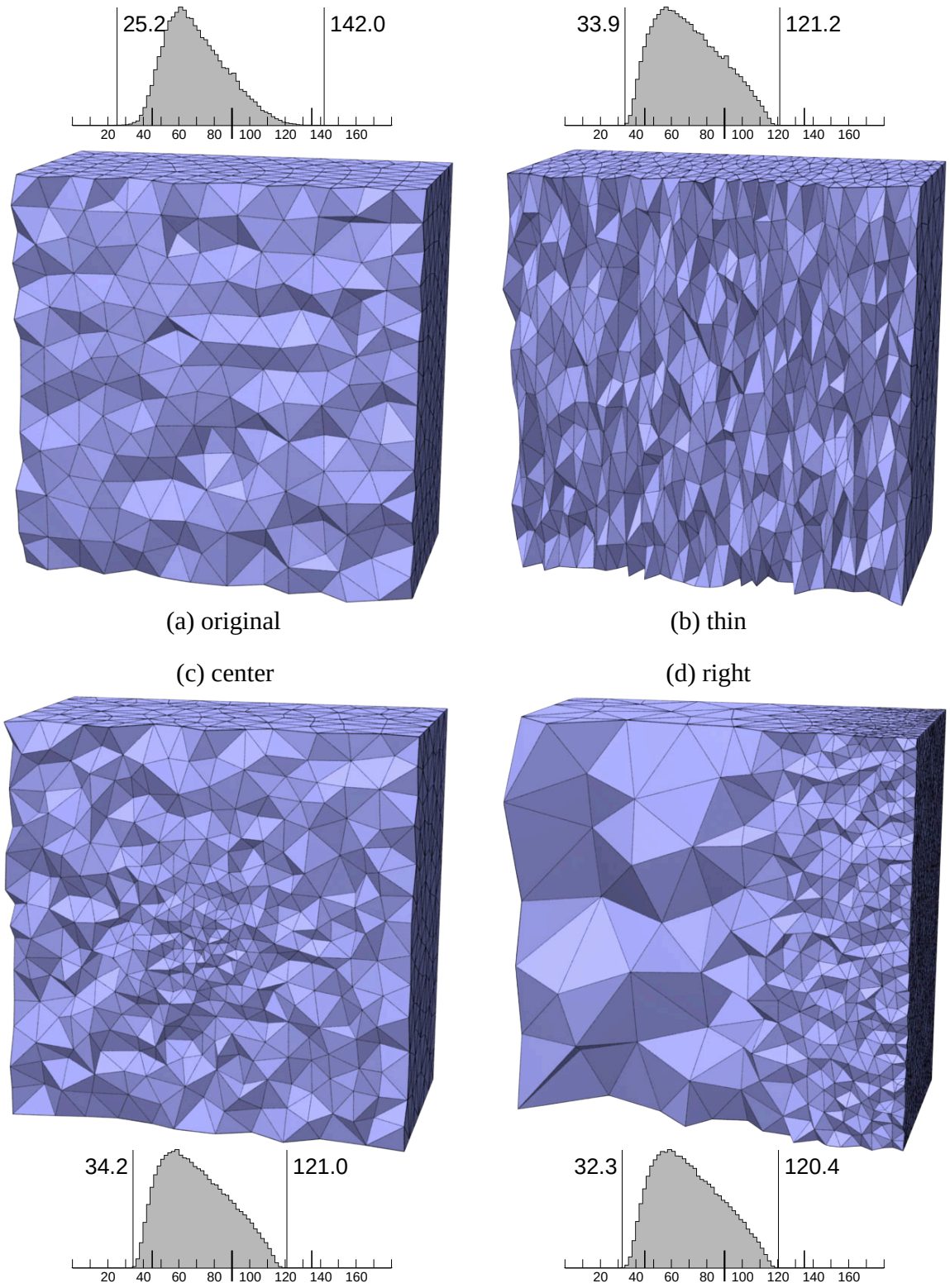


Figure 7.1. A demonstration of anisotropic scaling tensor fields applied to a cube mesh. (a) The original mesh. (b) A mesh with thin tetrahedra oriented vertically. (c) A mesh with smaller tetrahedra in the center. (d) A mesh with larger tetrahedra on the left and smaller tetrahedra on the right. All the histograms tabulate the dihedral angles in isotropic space.

$$\mathbf{E}_{\text{sink}}(\mathbf{x}) = \begin{bmatrix} \mathbf{u}_r & \mathbf{u}_s & \mathbf{u}_t \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & S_s(\mathbf{x}) & 0 \\ 0 & 0 & S_s(\mathbf{x}) \end{bmatrix} \begin{bmatrix} \mathbf{u}_r & \mathbf{u}_s & \mathbf{u}_t \end{bmatrix}^{\mathsf{T}}$$

creates the "sink" effect of Figure 7.2b. Replacing the diagonal values of the scale matrix with $S_s(\mathbf{x})$, 1, and 1 produces the "swirl" effect of Figure 7.2. Figure 7.3 shows two position-varying tensor fields based on a sine wave applied to a cube.

Small domain angles (angles between boundary faces of the input) are a source of difficulty in isotropic mesh generation, and they can be particularly pernicious in anisotropic meshing, because even if a domain has no small angles in physical space, it could have small angles in in isotropic space, which hold down the minimum tetrahedron quality. Figure 7.4 shows the Cow mesh before and after adaptation to a tensor based on a sine wave. In physical space, the smallest dihedral angle of the Cow domain is 45°. The tensor field warps the boundary, reducing the minimum domain angle to 14°. Although most of the tetrahedra in the adapted mesh have good quality, the small domain angles in isotropic space hold the minimum and maximum dihedral angles to 10.3° and 160.6°. The third row of meshes shows only those tetrahedra with a dihedral angle smaller than 20° or larger than 140°. Both before and after improvement, the few bad tetrahedra that remain are all on the boundary of the mesh.

These are artificial examples used to illustrate the effect of the scaling tensor and its flexibility. In practice, the anisotropy may be determined by the properties of a function being approximated, the inherent anisotropy of a partial differential equation, or some other practical consideration.

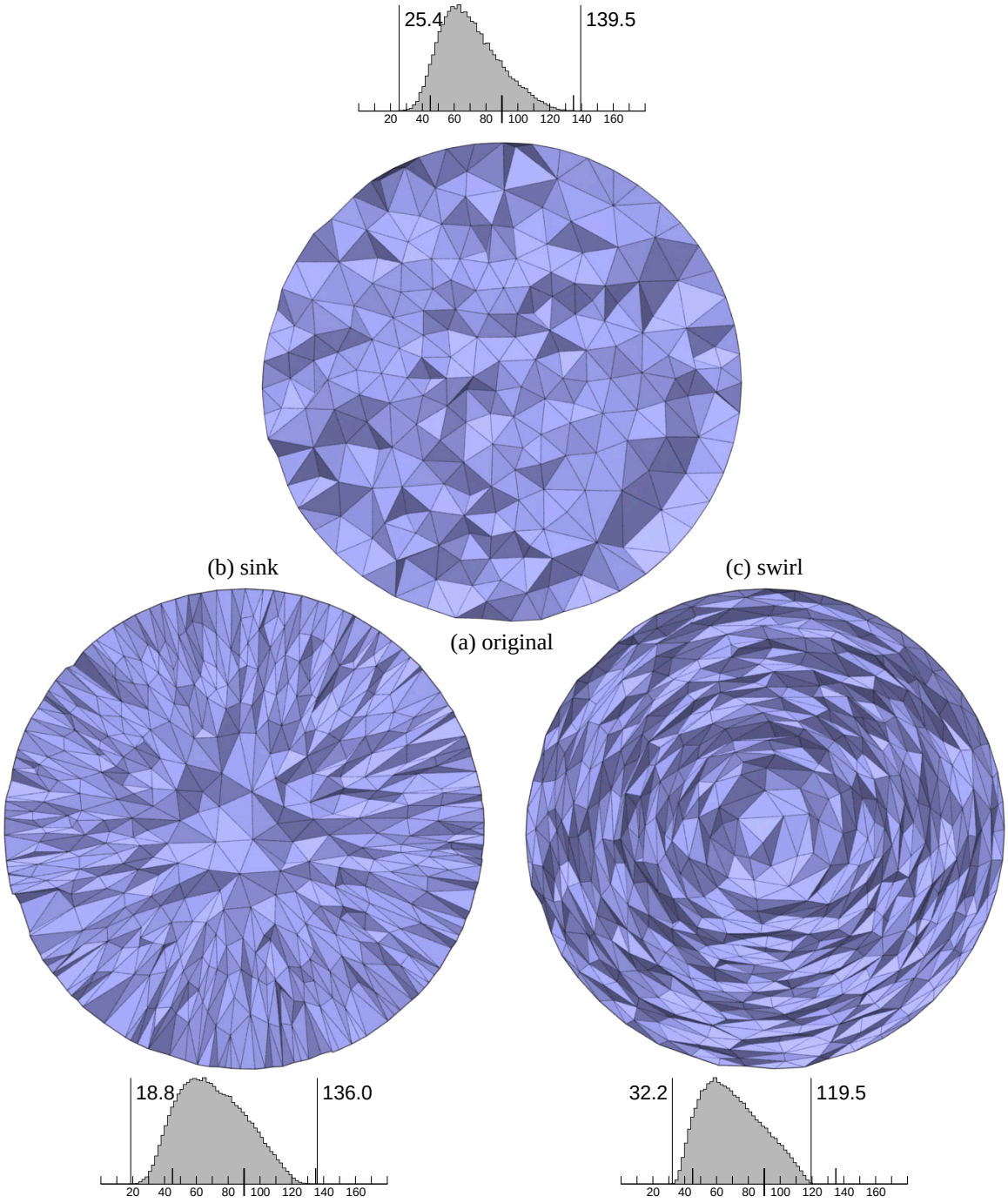


Figure 7.2. A demonstration of anisotropic adaptation of a unit sphere mesh. (a) The original mesh. (b) A mesh with thinner tetrahedra oriented toward the center of the sphere. (c) A mesh with thinner tetrahedra oriented around the center of the sphere. The histograms show the dihedral angles in isotropic space.

(a) scaled along a sine wave

(b) scaled normal to a sine wave

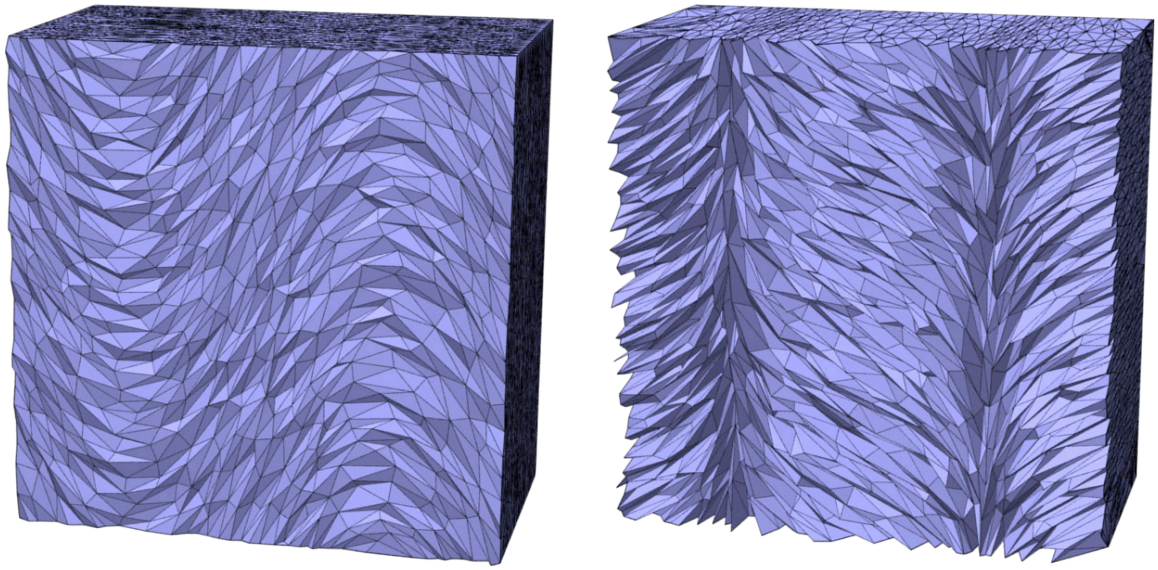


Figure 7.3. Two cube meshes after adaptation to anisotropic tensor fields based on a sine wave. In (a), the field is oriented along the sine wave. In (b), the field is oriented normal to the sine wave.

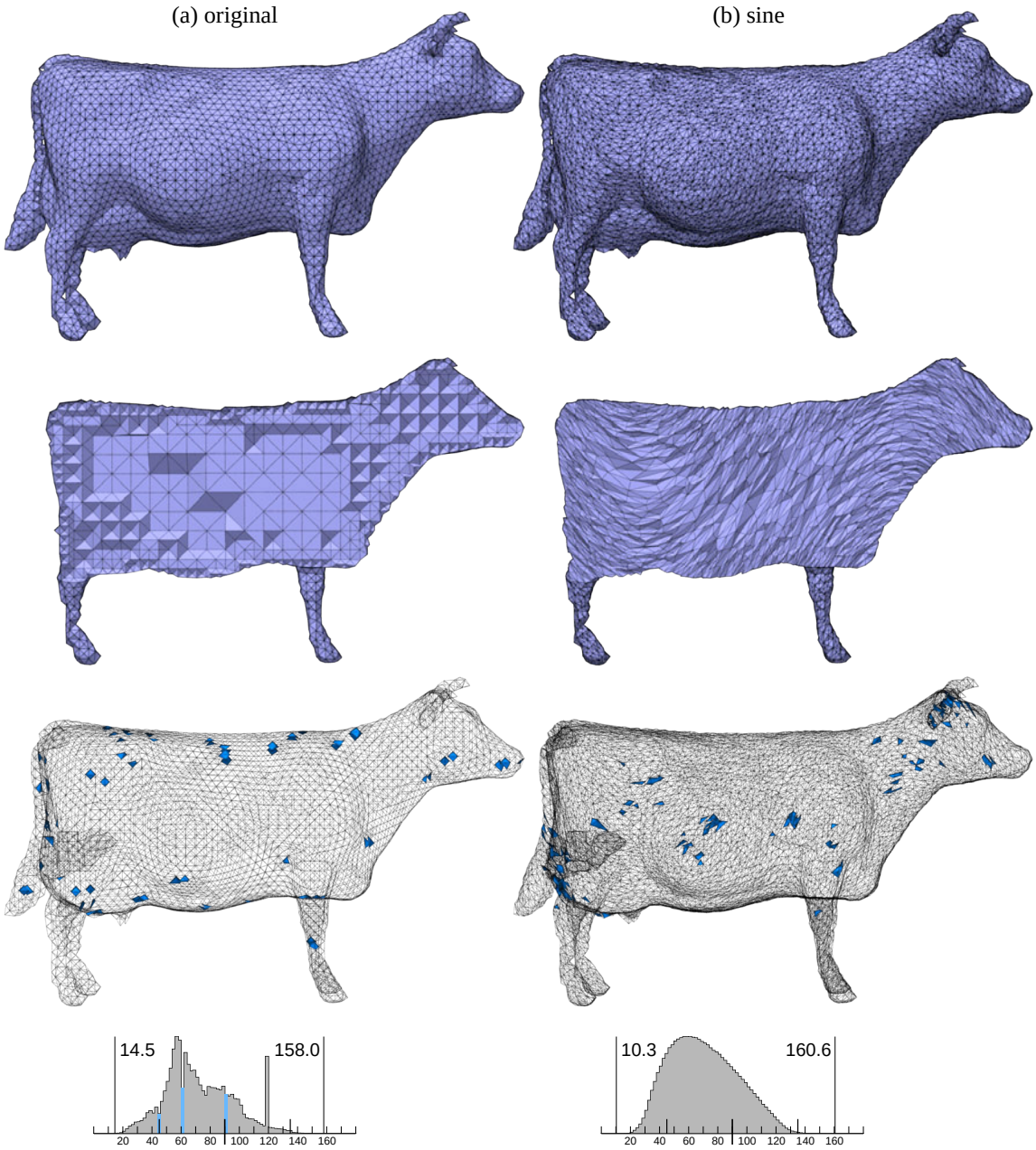


Figure 7.4. The Cow mesh before (a) and after (b) anisotropic adaptation to a tensor field based on a sine wave. The top row shows the meshes. The second row shows cutaways of their interiors. In the third row, tetrahedra with dihedral angles smaller than 20° or larger than 140° are colored blue. The histograms show the dihedral angles in isotropic space.

Chapter 8

A Dynamic Mesh Improvement Schedule

In many physical processes, the geometric domain changes shape with time. In car crashes, muscle movements, explosions, and melting wax, solids undergo large elastic deformations, and often experience plastic flow. Fluids (liquid or gaseous) reshape themselves with ease. Figure 8.1 illustrates a simulation by Klingner, Feldman, Chentanez, and O'Brien [32] of smoke being stirred by a moving paddle. They remesh the air around the paddle for each simulated timestep. The problem of maintaining a mesh of a domain whose shape is grossly changing is called *dynamic mesh generation*.

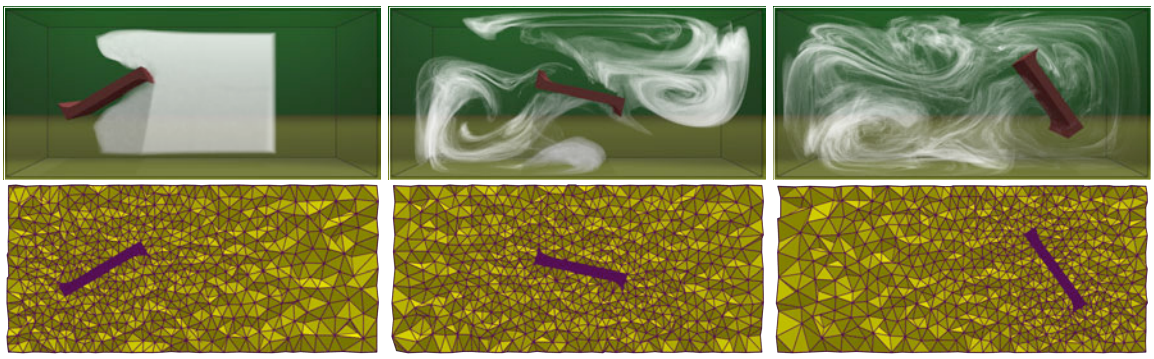


Figure 8.1. Top: A paddle mixes smoke in a tank. Bottom: Cross-sections of the tetrahedral simulation meshes used for each frame.

There are two obvious (but flawed) ways to mesh dynamically through time. One is to stretch a mesh to follow the deformation of a domain. During each timestep of a simulation, the material deforms in response to internal and external forces. The vertices of the mesh move in a *Lagrangian* fashion to track this deformation, as if they are particles of material. This approach can accommodate only limited deformations, because large displacements create poorly shaped tetrahedra—or even inverted tetrahedra—which engender wrong simulation results.

The second way is to simply generate a new mesh from scratch at every timestep of a simulation, as is done in Figure 8.1, or whenever the quality of the mesh falls below a threshold, as in the work of Bargteil, Wojtan, Hodgins, and Turk [6]. Unfortunately, this method quickly accumulates large numerical errors because of the need to reinterpolate physical properties such as velocity and strain from the old mesh to the new mesh. This rapid accumulation of error is called *artificial diffusion*, because the sampled properties diffuse unnaturally through the material.

Here I investigate a hybrid approach in which the mesh vertices track particles of material and small portions of the mesh are remeshed, but only when necessary. The problem of artificial diffusion obliges us to try to maintain a mesh with as much *temporal coherence* as possible, meaning that the topology of the mesh changes as little as possible during each timestep. The tetrahedra can move, their vertices following the motion of the material, without creating artificial diffusion; but smoothing of the vertices to improve tetrahedron quality (as opposed to movement that tracks the material) does introduce artificial diffusion. Topological transformations also introduce artificial diffusion. Therefore, as the vertices move, I want to preserve as many of the tetrahedra as possible. But I must repair tetrahedra that have become too skinny. Unfortunately, it is rarely possible to replace *just* the bad tetrahedra. The goal is to balance the errors introduced by artificial diffusion with the errors introduced by poorly shaped tetrahedra.

I judge a dynamic mesher by several criteria. First, it should enforce a minimum tetrahedron quality as the mesh changes. Second, it should also control the sizes of the tetrahedra. Third, it must maintain the conformity of the tetrahedra to the physical domain. Fourth, the cumulative volume remeshed over a sequence of timesteps should be as small as possible. (If a physical region is remeshed multiple times, I count it multiple times in the *cumulative volume remeshed*, because every remeshing worsens the artificial diffusion. Thus, the cumulative volume remeshed can exceed

100% of the total volume of the mesh.) Fifth, as the mesh changes, physical quantities should be interpolated in a manner that conserves (or at least does not increase) mass, energy, and momentum. The fifth goal is really a property of the interface between the dynamic mesher and the physical simulation, and I do not address it here.

The literature on this approach is small, because it is difficult to implement. A particularly good two-dimensional example is the dynamic meshing procedure created by Cardoze, Cunha, Miller, Phillips, and Walkington [10] for the simulation of circulating blood and the deforming blood cells transported by it. Cardoze et al. use triangular meshes with curved boundaries. A two-dimensional system for modeling large plastic deformations in metal cutting was proposed by Borouchaki, Laug, Cherouat, and Saanouni [7].

In three dimensions, it is difficult to devise a local improvement algorithm that improves tetrahedra reliably enough for dynamic meshing. After all, if even one tetrahedron is allowed to invert, the simulation cannot continue. A preliminary exploration of adaptive improvement of tetrahedral meshes to maintain the accuracy of penetration simulations is given by Mauch, Noels, Zhao, and Radovitzky [37], but they offer little hard data on tetrahedron quality and its effects on simulation accuracy. My mesh improvement techniques work consistently enough to maintain high quality over many timesteps, thereby making accurate dynamic simulations possible.

I perform dynamic meshing with the same ideas and transformations I use for static mesh improvement. However, the two applications have different goals, and require different mesh improvement schedules. In static meshing, I try to improve the mesh to as high a quality as possible, changing as much of the mesh as necessary. For speed, my static mesh improvement schedule tries the fastest transformations (like smoothing) first, then the more expensive ones. In contrast, there is a cost to changing tetrahedra during dynamic improvement. In dynamic meshing, I act only when some tetrahedron falls below a minimum threshold for quality; and then I try to fix it while changing as few tetrahedra as possible. My dynamic mesh improvement schedule tries the most local transformations first, then the more disruptive ones (like smoothing). Figure 8.2 shows an example of my dynamic improvement schedule maintaining the quality of the Dragon mesh as an artificial deformation field twists its vertices.

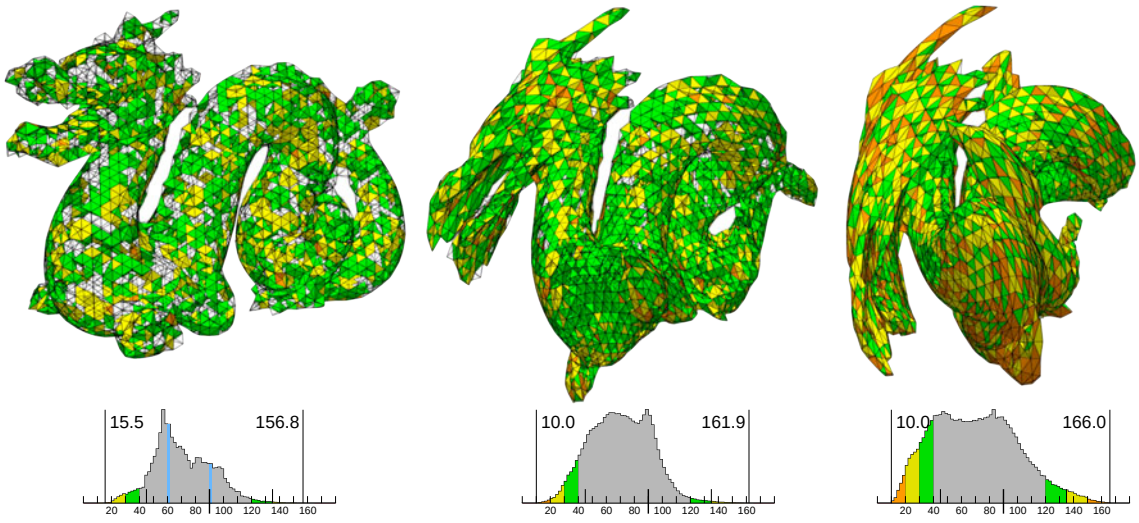


Figure 8.2. An example of dynamic mesh improvement. An artificial deformation field twists the DRAGON mesh. At each timestep, the dynamic improvement schedule ensures that all the dihedral angles are no less than 10° and no greater than 170°. These images show the mesh initially, after 45 timesteps, and after 90 timesteps. Red tetrahedra have dihedral angles under 10° or over 165°, orange under 20° or over 150°, yellow under 30° or over 135°, green under 40° or over 120°, and better tetrahedra do not appear. Histograms show the distributions of dihedral angles, and the minimum and maximum dihedral angles, in each mesh. Multiply the heights of the blue histogram bars by 20. Histograms are normalized so the tallest bar always has the same height; absolute numbers of tetrahedra cannot be compared between histograms.

A common misconception about dynamic meshing is that its purpose is to be appreciably faster that remeshing from scratch. I am not optimistic that dynamic meshing will ever have a large speed advantage. Simply computing the quality of every tetrahedron in a mesh to check if any remeshing is needed takes an amount of time only a few times smaller than the running time for some of the faster tetrahedral mesh generation algorithms (based on grids, Delaunay triangulations, or advancing front methods). In a simulation with small timesteps (where little remeshing is needed during any single timestep), dynamic meshing might have the potential to be twice as fast as remeshing from scratch, but it will not fundamentally change what is possible. Accuracy provides a much stronger motivation for dynamic meshing than speed.

8.1 The improvement passes

In Section 4.2, I describe "passes" that package improvement operations for use in the static mesh improvement schedule. I use these same passes for dynamic mesh improvement. Recall that each improvement pass (Listings 4.1–4.4) accepts a set of tetrahedra K and returns a set of tetrahedra K'. The modified tetrahedra in K' include all the surviving tetrahedra from K, any new tetrahedra created by the pass, and (for passes that smooth vertices) any tetrahedron with a vertex smoothed by the pass.

For dynamic improvement, I want to modify as small a proportion of the mesh as possible. All the improvement passes have the potential to return K' with more tetrahedra than K, but some passes change more tetrahedra than others. The topological pass (described in Section 4.2.2) is the most conservative; only new tetrahedra created by topological operations are added. The edge contraction pass (Section 4.2.3) is worse; it adds to K' all the tetrahedra incident to the endpoints of each contracted edge. The insertion pass (Section 4.2.4) is worse still; it may employ smoothing, edge removal, and face removal near the inserted vertex, and includes the tetrahedra created or modified by any of these operations in K'. The smoothing pass (Section 4.2.1) is the worst of all, because it modifies many tetrahedra. Every tetrahedron incident to a smoothed vertex is included in K'. The relative disruptiveness of the different passes guides my design of the dynamic mesh improvement schedule.

8.2 The dynamic mesh improvement schedule

Listing 8.1 lists pseudocode for a dynamic improvement schedule that enforces a minimum tetrahedron quality while changing as little of the mesh as it can. For each tetrahedron in a mesh whose quality is worse than some specified minimum quality $q_{\min}$, the schedule invokes the procedure ImproveTet to try to improve it. ImproveTet maintains a set A of tetrahedra that currently exist in the mesh. Initially A contains just a single bad tetrahedron, but as ImproveTet works, it adds to A all the tetrahedra it creates or modifies.

```
ImproveTet(M, t, q_min)
{ t is a tetrahedron in mesh M }
{ q_min is the minimum acceptable tetrahedron quality }
1     i ⇐ 0
2     A ⇐ { t }
3     q ⇐ the quality of t
4     while i < 10
5         do
6             A ⇐ TopologicalPass(A, M)
7             q ⇐ the quality of the worst tetrahedron in A
8             if q ≥ q_min
9                 return success.
10        while A is sufficiently improved (see Section 8.2.1)
11        A ⇐ ContractPass(A, M)
12        q ⇐ the quality of the worst tetrahedron in A
13        if q ≥ q_min
14            return success.
15        A ⇐ InsertPass(A, M)
16        q ⇐ the quality of the worst tetrahedron in A
17        if q ≥ q_min
18            return success.
19        A ⇐ SmoothPass(A, M)
20        q ⇐ the quality of the worst tetrahedron in A
21        if q ≥ q_min
22            return success.
23        i ⇐ i + 1
24    return failure.

DynamicImproveMesh(M, q_min)
1     B ⇐ set of tetrahedra in M with quality less than q_min
2     for each tetrahedron t ∈ B
3         if t still exists and has quality less than q_min
4             if ImproveTet(M, t, q_min) = failure
5                 return failure.
6     return success.
```

Listing 8.1: The dynamic mesh improvement schedule.

ImproveTet runs improvement passes on A: first, topological operations that do not change the vertices of the mesh (edge removal and multi-face removal), then the more disruptive operations of edge contraction, vertex insertion, and finally smoothing. Each pass may cause A to grow. If a pass improves the worst tetrahedron in A to a quality of at least $q_{\min}$, then ImproveTet succeeds and terminates. If all the tetrahedra in the mesh with quality less than $q_{\min}$ are removed by calls to ImproveTet, DynamicImproveMesh succeeds.

The fact that the insertion pass is less disruptive than the smoothing pass is counterintuitive. After all, the insertion pass itself smooths vertices as part of the cavity improvement process (see Listing 3.2). In practice, though, a single smoothing pass rarely brings a bad tetrahedron above the minimum quality; if smoothing can do it at all, it usually takes multiple passes, each of which enlarges A. In contrast, insertion (and subsequent cavity improvement) employs a suite of local improvement operations and can often surgically remove a single bad tetrahedron in one attempt. Experiments show that putting the smoothing pass before the vertex insertion pass leads to more remeshing.

8.2.1 Success criteria for the topological pass

TopologicalPass is the only improvement pass in the dynamic schedule that is permitted to run repeatedly, as long as it makes progress in improving A. The schedule evaluates the effectiveness of TopologicalPass to determine when it should terminate the inner **while** loop of ImproveTet (lines 5–10). I use the same success criteria as for static mesh improvement (Section 4.3), except that instead of computing the thresholded means and the minimum tetrahedron quality across the entire mesh, I only consider the tetrahedra K' returned by the pass. If the worst tetrahedron in K' is better than the worst tetrahedron in K (the tetrahedra the pass is asked to improve), or if one of the thresholded means improves by 0.005 (as opposed to 0.0001 for static improvement), I consider the pass successful.

Because topological transformations can bring in additional tetrahedra, it is possible that the minimum quality of K' is actually lower than that of K. To keep A from growing too much due to repeated runs of TopologicalPass, I add a check to TopologicalPass that compares the worst quality

of K' to that of K and reverses all the operations of the pass if the minimum quality has worsened. No such check is performed on the other improvement passes.

8.2.2 A reason to keep good tetrahedra under attack

The reader might wonder whether ImproveTet should, after each improvement pass, trim from A all the tetrahedra with quality better than $q_{\min}$, thereby checking the expansion of A and focusing improvement efforts on the bad tetrahedra. I have tried this, and found that it often causes ImproveTet to fail when it would otherwise succeed. The good tetrahedra in A serve an important role: they make more of the mesh near the bad tetrahedra available for modification, and thus help the improvement operations to replace bad tetrahedra that could not be removed in isolation. In practice, I occasionally see a single tetrahedron that resists improvement for six or more iterations of the loop, when A has grown to include hundreds or thousands of nearby tetrahedra.

8.2.3 Why not repeat passes that are working?

Unlike ImproveTet, the static improvement schedule (Section 4.4) arranges improvement passes hierarchically: it runs smoothing passes over and over as long as they are effective, and only then proceeds to more expensive passes such as edge and face removal, edge contraction, and vertex insertion. Some tetrahedra improve in early passes; others succumb only to more aggressive operations. This arrangement works fine for global mesh improvement, but it is unsuitable for dynamic meshing where we need to improve only a few bad tetrahedra and wish to limit the amount of remeshing.

Suppose the set A contains a single bad tetrahedron t that cannot be removed by TopologicalPass and ContractPass, but is easily eliminated by InsertionPass. A hierarchical arrangement might run TopologicalPass and ContractPass over and over, as long as the other tetrahedra in A improve incrementally. After every pass, A grows. Eventually, the schedule moves on to InsertionPass and eliminates t. The cost of delaying InsertionPass is that many more tetrahedra are altered.

Instead, ImproveTet runs each kind of pass only once before moving on (except TopologicalPass, which modifies the fewest tetrahedra and does not move vertices). This way, every type of

improvement operation gets a crack at improving problem tetrahedra sooner. In my experience, this serial arrangement of passes results in much less remeshing.

8.2.4 Why not use two quality thresholds?

The dynamic improvement schedule in Listing 8.1 uses a single quality threshold $q_{\min}$ to determine which tetrahedra need improvement (lines 1 and 3 of DynamicImproveMesh) and to enforce a minimum quality of the tetrahedra after improvement (lines 8, 13, 17, and 21 of ImproveTet). Another design I considered uses two quality thresholds: a *trigger threshold* that identifies tetrahedra in need of improvement, and a higher *goal threshold* that dictates the minimum quality of the modified tetrahedra after improvement. The reasoning was that by boosting the quality of the tetrahedra significantly above the trigger threshold, it would be less likely that these tetrahedra would need improvement again after another timestep, and the number of changes to the mesh over many timesteps would be fewer.

Instead, I discovered that, for a fixed trigger threshold, dynamic mesh improvement changes the mesh more when the goal threshold is higher than the trigger threshold than it does when they are the same—even cumulatively, over many timesteps. For example, consider using the minimum sine objective function (Section 2.1.1) to enforce a minimum quality (trigger threshold) of $\sin 10°$ while the Dragon mesh twists as depicted in Figure 8.2. Cumulatively over 90 timesteps, the dynamic improvement schedule alters 19.10% of the total mesh volume if the goal threshold is also $\sin 10°$, compared with 41.67% of the total mesh volume if the goal threshold is $\sin 13°$.

Why is more of the mesh altered when I use two thresholds? My experience observing dynamic improvement with two thresholds reveals an interesting pattern. Commonly, many neighboring tetrahedra might fall below the goal threshold before a single one of them falls below the trigger threshold. The first iteration of the improvement loop in ImproveTet might eliminate the bad tetrahedron, leaving no tetrahedra below the trigger threshold; but the set A has grown to include several tetrahedra below the goal threshold. Each successive improvement pass enlarges A further. Improvement terminates only when every tetrahedron in A reaches the goal threshold. To improve just one tetrahedron that fell below the trigger threshold, many other tetrahedra were remeshed that

might never have fallen below the trigger threshold. It is hard to maintain locality of improvement when you have to lift the quality of a tetrahedron to a point substantially higher than that of its neighbors.

Chapter 9

Dynamic Mesh Improvement Results and Discussion

This chapter demonstrates the dynamic improvement schedule from Chapter 8 in two circumstances. In the first, I artificially deform meshes by scripting the motion of their vertices. The artificial displacements make no attempt to simulate physical principles. In the second, a simulation of elastic solid mechanics determines the movement of the vertices. The simulation is *Lagrangian*, meaning that the mesh vertices attempt to follow the motions of particles of material. In both the artificial and simulation-driven examples, the dynamic mesh improvement schedule aims to enforce a minimum tetrahedron quality at each step of deformation.

The effort of dynamic mesh improvement is justified only if it has a clear advantage over two easier alternatives: remeshing from scratch, and not changing the topology of the mesh at all. It is easy to find circumstances in which an object is deformed so radically that a simulation cannot work without remeshing; once a tetrahedron becomes inverted, the simulation cannot be trusted at all. In Section 9.4, for example, I present a simple example where a mesh is poked with a narrow rod, and remeshing is necessary to avoid inverting tetrahedra.

It is not obvious, though, that dynamic meshing can offer a real improvement in accuracy over remeshing from scratch at each timestep. In Section 9.4, I show two examples in which it is plainly apparent that substantially more artificial diffusion occurs with frequent remeshing than with dy-

namic remeshing, and noticeable visual artifacts appear. In these examples, the algorithm I use for "remeshing from scratch" is my static mesh improvement schedule from Section 4.4. Therefore, these examples also show that the differences between the static and dynamic schedules have a real benefit. The static mesh improvement schedule does not always replace every tetrahedron, so it likely underestimates the amount of error that would be introduced by total remeshing.

In these simulations, the exact solution is unknown, so we cannot measure simulation accuracy directly. However, the visual differences are more than strong enough to validate the dynamic meshing schedule for use in computer graphics. Therefore, it seems useful to investigate the behavior of the dynamic mesher under more extreme but non-physical deformations. For these examples, I compute the percentage of the total domain volume that is replaced or modified by mesh improvement operations, cumulative over all the timesteps of the simulation; and I use this figure as a proxy for the amount of artificial diffusion that would be introduced into a simulation. This changed volume includes the volume of every new tetrahedron created by topological operations as well as every tetrahedron that has a vertex smoothed; and because it is cumulative over timesteps, it may exceed 100%. For example, if a mesh is truly recreated from scratch on every timestep, the cumulative volume remeshed will increase by 100% on every timestep, reflecting a high error introduced by artificial diffusion.

9.1 Dynamic remeshing with artificial displacements

Figure 9.1 shows the DRAGON mesh twisting along its length like a wrung-out towel, then returning to its original configuration. My dynamic mesh improvement schedule keeps all the dihedral angles between 10° and 170°. At the extreme point of this deformation (step 90), the features of the dragon are heavily warped. The distribution of dihedral angles, shown in the histograms of the right column, has widened significantly from the regular structure of the starting mesh. Still, only 19.1% of the domain volume has to be remeshed to maintain good angles. As the mesh twists back to its initial configuration, most tetrahedra return to their original shape, and only a handful need improvement after the twisting motion reverses, bringing the cumulative remeshed volume to 19.4%.

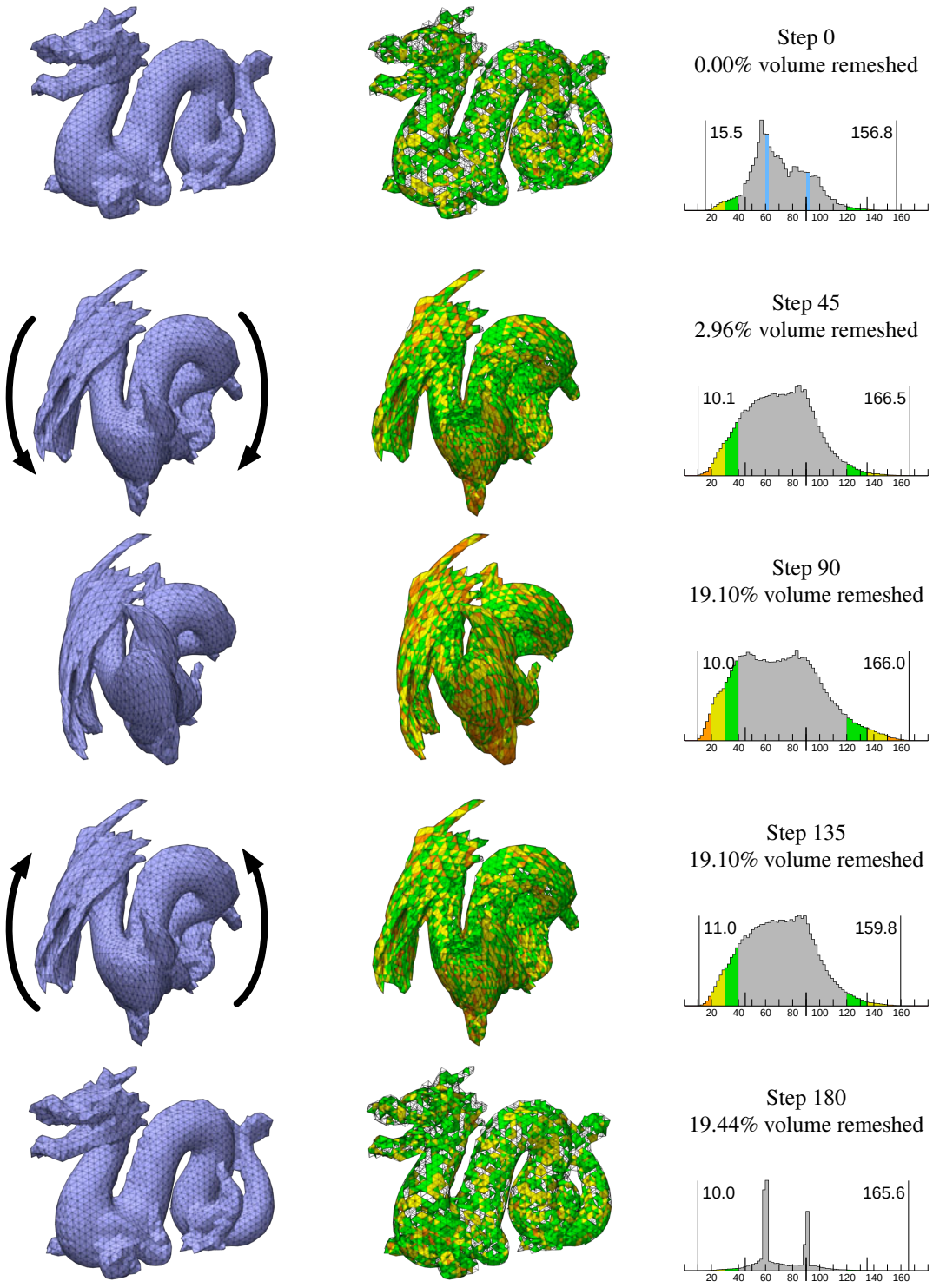


Figure 9.1. The DRAGON mesh twists along its length, causing tetrahedra to deform and worsen; then it returns to its original shape. Histograms on the right show the dihedral angles at selected timesteps, and the cumulative domain volume that has been remeshed. Tetrahedra in the center column and histograms are colored as described in the caption for Table 5.2. At each step, dynamic remeshing keeps all the dihedral angles between 10° and 170°.

Only the deforming parts of the mesh should be changed by remeshing. Figure 9.2 depicts a cylinder of about 10,000 tetrahedra twisting on its right end. The leftmost quarter of the cylinder (shaded dark) is fixed and does not rotate. Because the tetrahedra in the fixed region keep their original quality, the dynamic improvement schedule does not alter them.

Coarser meshes make it harder for dynamic improvement operations to remain local. Figure 9.3 shows the HOUSE mesh bowing inward and then returning to its original shape. Because there are only about 10–20 tetrahedra along any dimension of the mesh, topological operations meant to replace a single bad tetrahedron are more likely to influence a large proportion of the mesh, as reflected by the 64.3% cumulative remeshed volume.

9.2 The effects of quality thresholds

The amount of remeshing can be quite sensitive to the minimum quality threshold. In Figure 9.4, the Cow mesh is pinched in the center, then it returns to its original shape. Dynamic meshing keeps all the dihedral angles between 20° and 160°. Cumulatively over 180 timesteps, 74% of the volume is remeshed. In Figure 9.5, the quality threshold increases to keep dihedral angles between 25° and 165°, and the cumulative remeshed volume increases to 411%. These two examples begin to exhibit sharply different amounts of remeshing after step 45, when dihedral angles drop below 25°. The more aggressive angle bounds force many more tetrahedra to be replaced as the pinch compresses the body of the cow, and then again as the body returns to its original shape.

The difference in remeshed volume for different quality thresholds depends on the kind of deformation—here, the deformation distorts the bulk of the tetrahedra—as well as the average quality of the tetrahedra in the mesh. If, as in Figure 9.5, many tetrahedra have quality only a bit higher than the minimum threshold, more remeshing takes place.

9.3 Good and bad examples of extreme deformations

Figure 9.6 depicts an example where the dynamic improvement schedule maintains good dihedral angles while the volume and proportions of the mesh vary wildly. First, the CUBE1K mesh

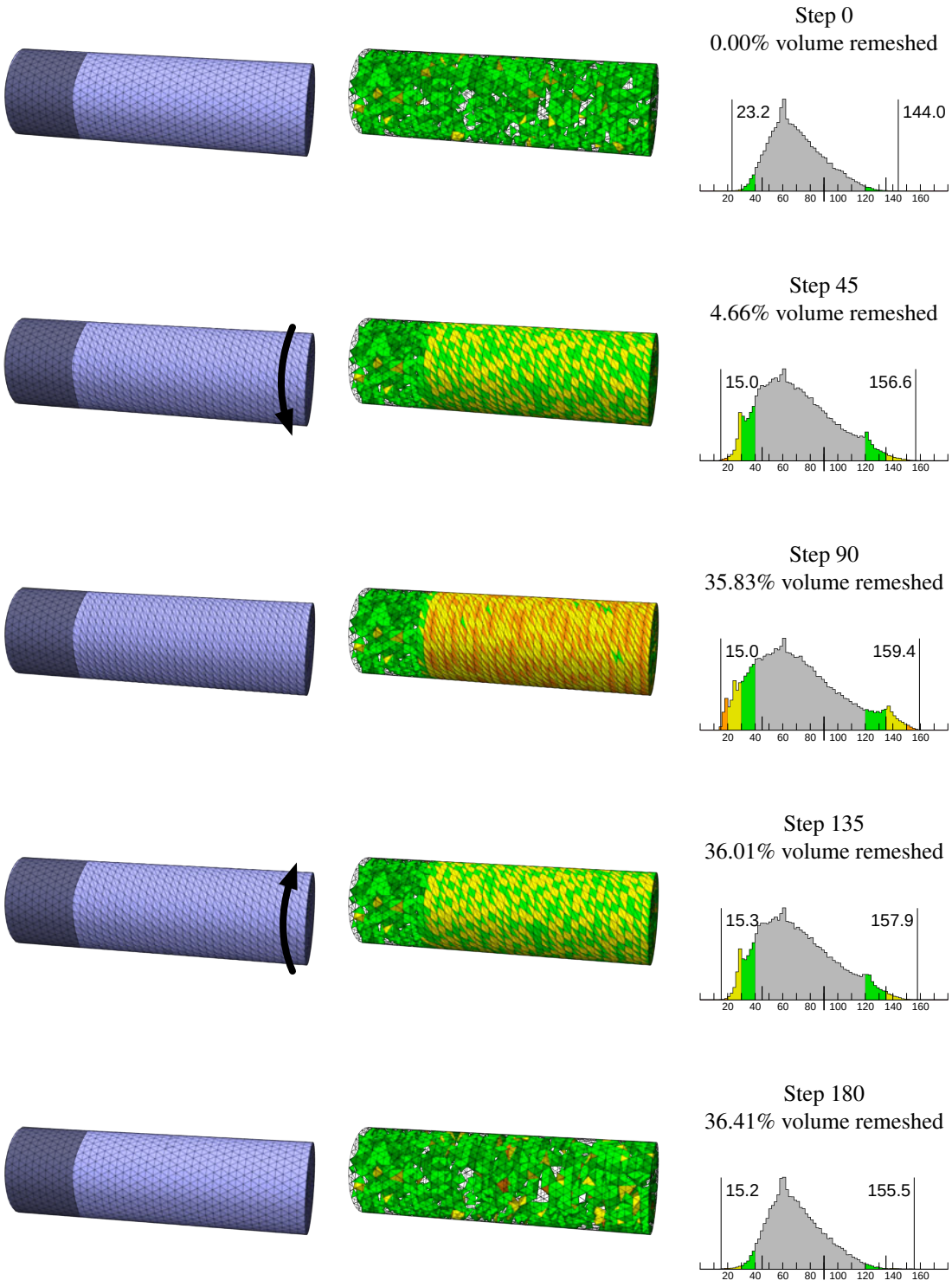


Figure 9.2. A cylinder mesh, held fixed in the shaded region, twists along its length, then returns to its original shape. Tetrahedra in the fixed region maintain good quality and are not altered by improvement. At each step, dynamic remeshing keeps all the dihedral angles between 15° and 165°.

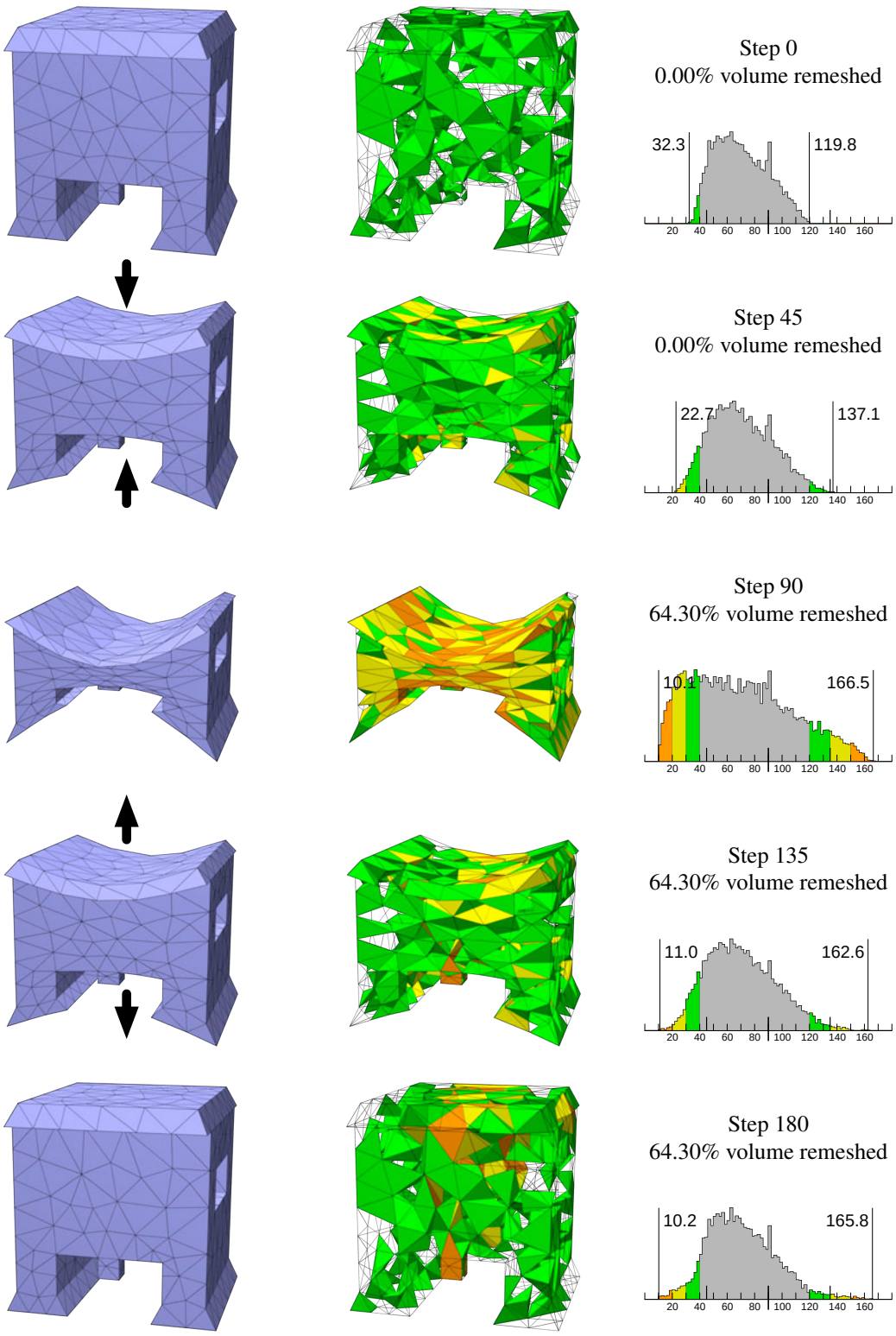


Figure 9.3. The HOUSE mesh pinches in the middle, then returns to its original shape. At each step, dynamic remeshing keeps all the dihedral angles between 10° and 170°.

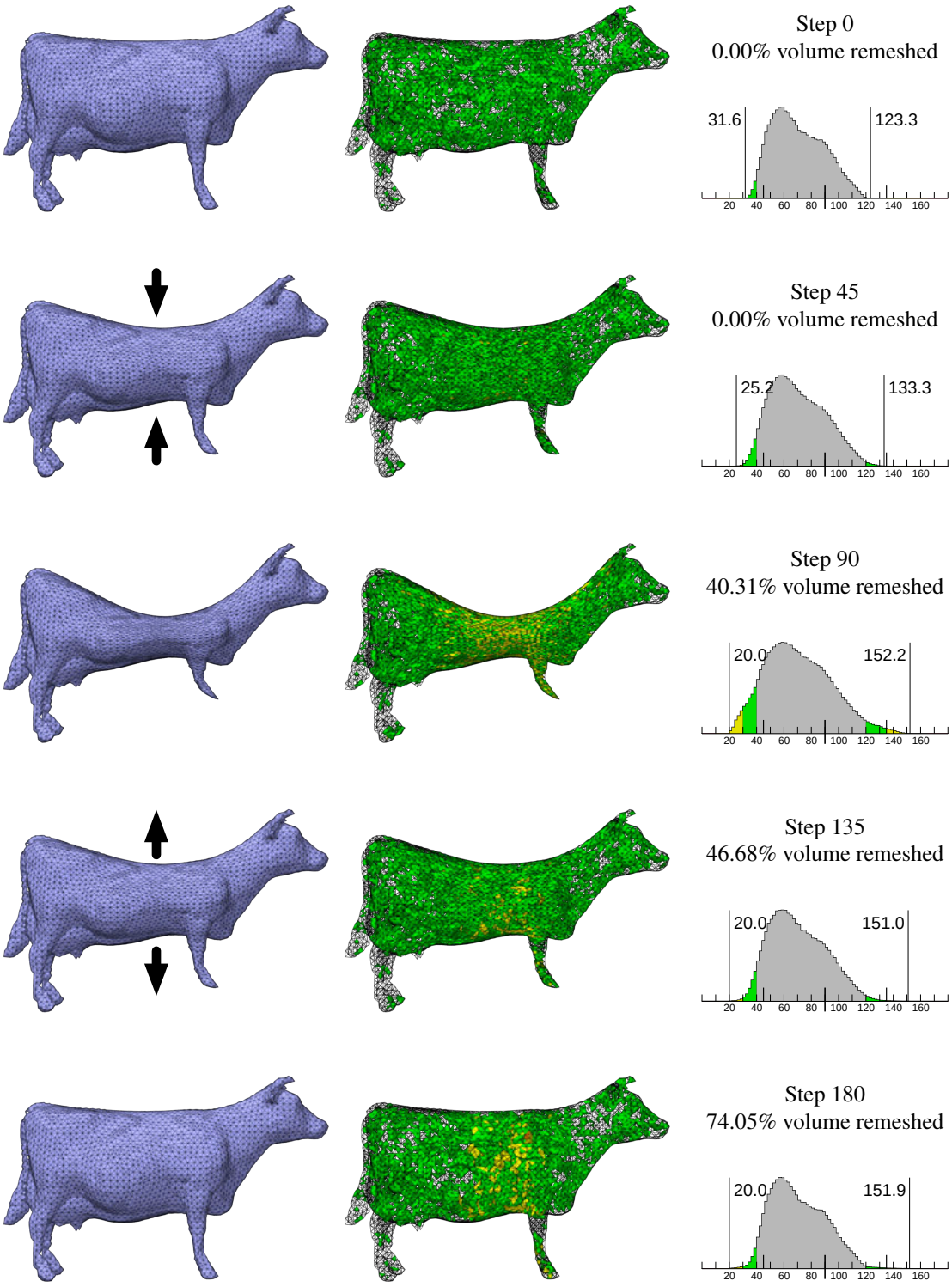


Figure 9.4. The Cow mesh pinches in the middle, then returns to its original shape. At each step, dynamic remeshing keeps all the dihedral angles between 20° and 160°.

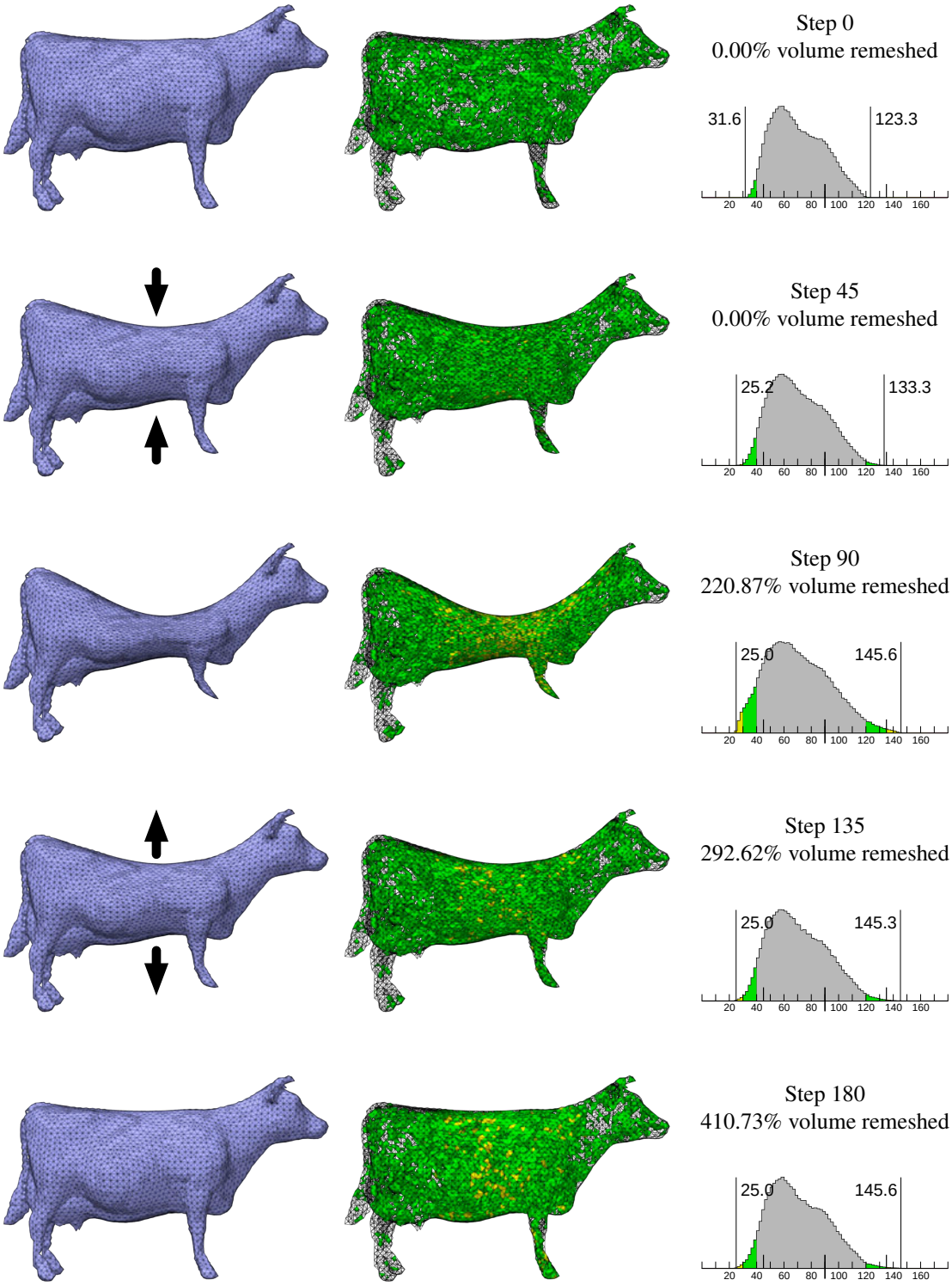


Figure 9.5. As in Figure 9.4, the Cow mesh pinches in the middle and returns to its original shape; but here dynamic remeshing keeps all the dihedral angles between 25° and 155°. Substantially more remeshing takes place.

stretches along one axis to over five times its original width. Then, it compresses along the same axis to less than one fifth of its original width. Finally, it returns to its original proportions. Throughout these deformations, the dynamic improvement schedule keeps all the dihedral angles between 15° and 165°. The cumulative proportion of the domain remeshed is huge—over 2,700%, all told—and few of the original tetrahedra survive at the end of the deformation. Dynamic meshing is worthwhile nonetheless, because an average tetrahedron survives about six timesteps. If this deformation were coupled with a simulation, the amount of artificial diffusion error introduced by reinterpolation would probably be about one sixth of the error introduced if the mesh were generated from scratch on every timestep.

In Figures 9.1–9.5, dynamic improvement uses quadric smoothing of surface vertices (described in Section 3.2.5) to aid improvement. Quadric smoothing works well when the surface is smooth and refined, but it has limitations. Figure 9.7 shows the CUBE10K mesh twisting severely before returning to its original state. Throughout the deformation, dynamic improvement keeps the dihedral angles between 10° and 170° degrees. During the twisting, the faces of the cube are curved, not flat, and the discretization of this curvature creates a washboard texture on the surface. (The smoother knows nothing about the true geometry of the edges and faces of the cube.) The smoothing of boundary vertices on this textured curved surface creates deformations that remain once the cube untwists. If quadric smoothing is disabled so that the domain shape is better maintained, dynamic improvement can only keep the dihedral angles between 4° and 176°.

The twisting cube also presents the problem of small dihedral angles along the boundary of the mesh. As the edges of the cube twist into helices, the dihedral angles where faces of the cube meet become small. If these angles fall below the minimum quality threshold, the dynamic improvement schedule fails.

9.4 Dynamic improvement coupled with an elastic simulation

I tested my dynamic improvement schedule coupled with a simple elastic simulator written by Nuttapong Chentanez. The simulator determines the deformation of the mesh during each timestep with a numerical model of elastic material. After each timestep of simulation, the dynamic im-

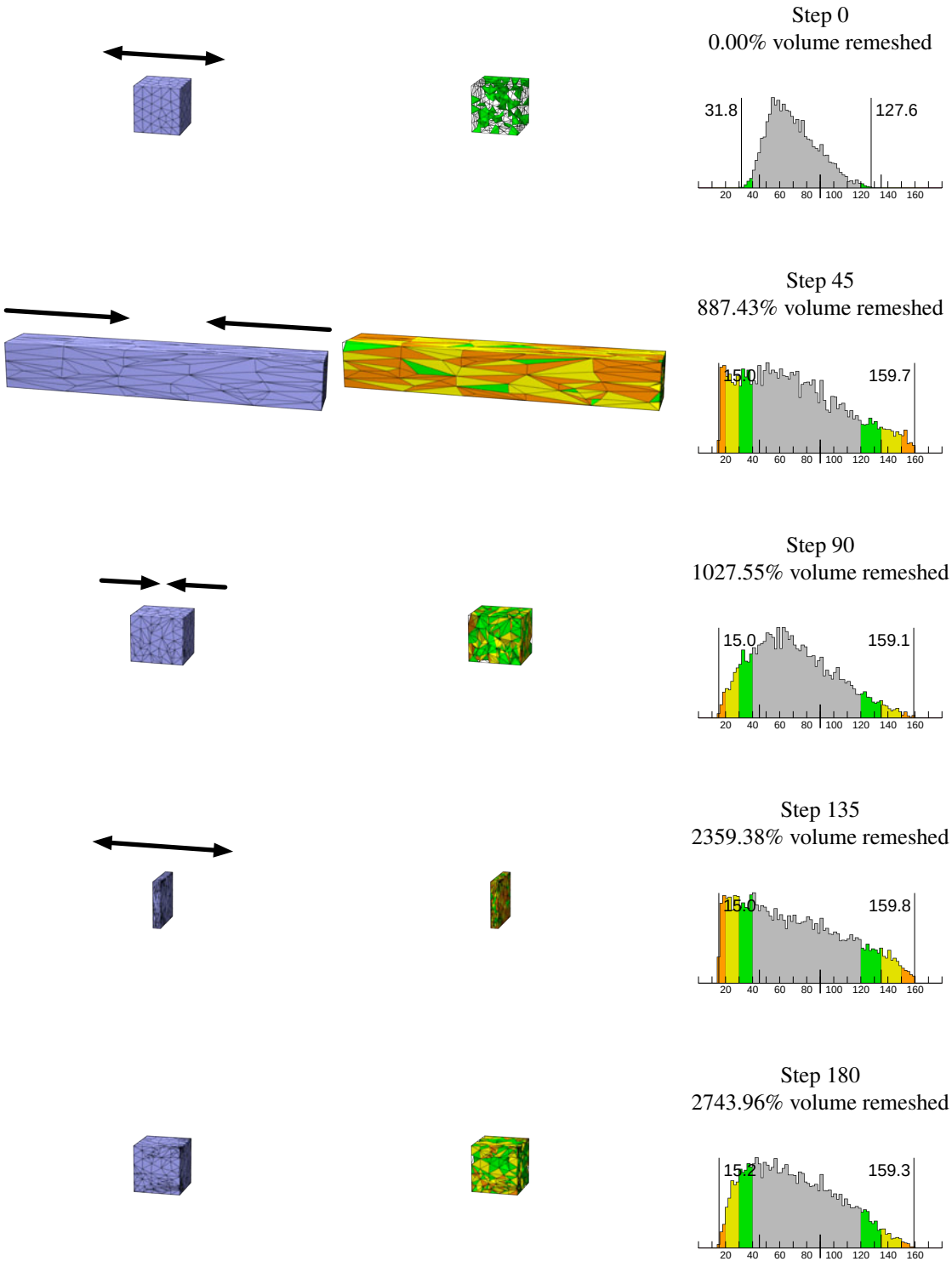


Figure 9.6. The CUBE1K mesh stretches to more than five times its original width, compresses to less than one fifth its original width, and then returns to its starting shape. At each step, dynamic remeshing keeps all the dihedral angles between 10° and 170°.

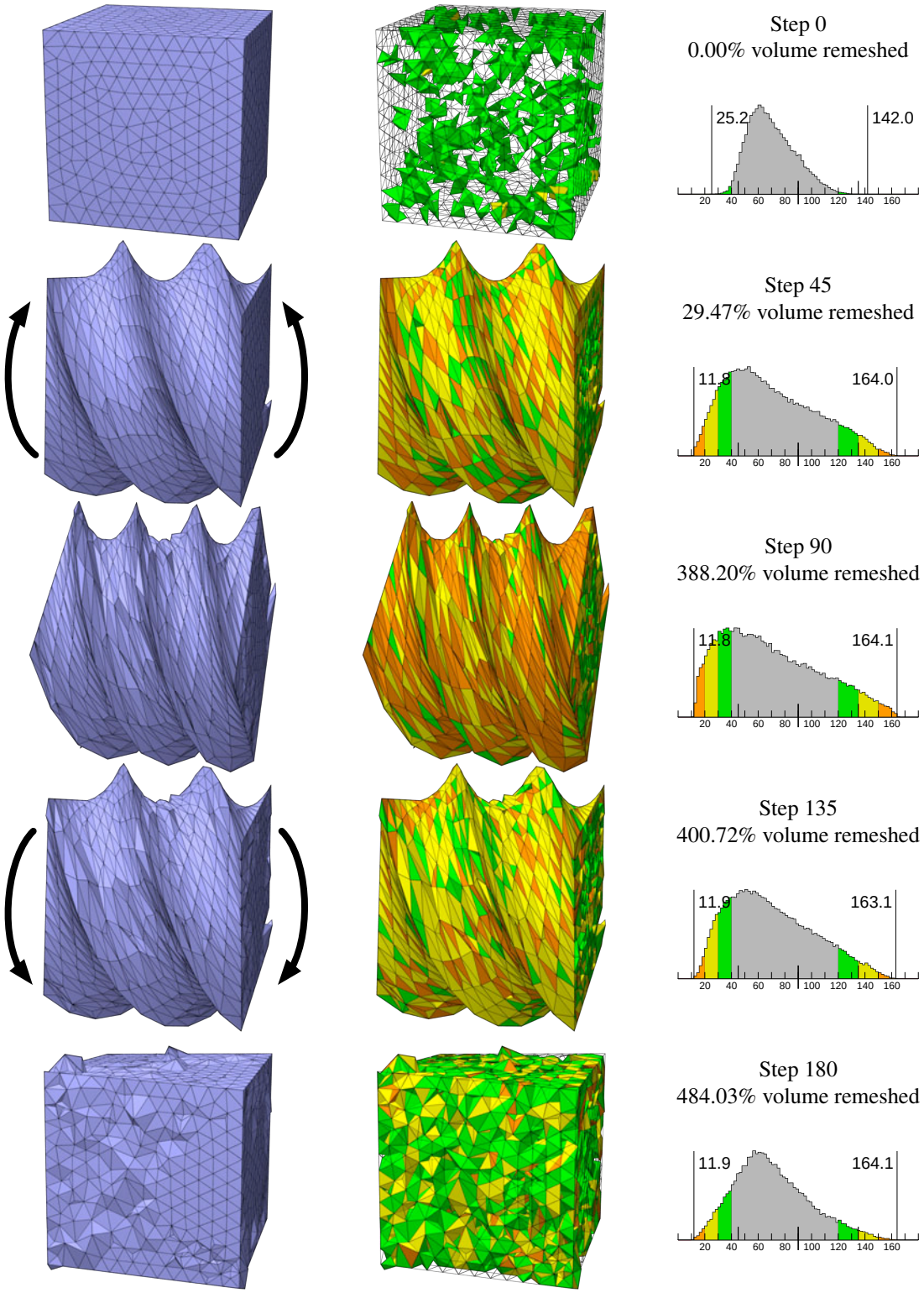


Figure 9.7. The CUBE10K mesh twists along its width several times, then returns to its original shape. At each step, dynamic remeshing keeps all the dihedral angles between 10° and 170°. Quadric smoothing, which makes these angles possible, disturbs the flat surfaces of the cube.

provement schedule improves any tetrahedra that have fallen below the minimum quality threshold. Physical properties are transferred to the remeshed portions of the mesh using a method described by Bargteil, Wojtan, Hodgins, and Turk [6].

In these simulations, the discretized strain field is piecewise constant over the tetrahedra. The main source of artificial diffusion is the reinterpolation of these elastic strains wherever remeshing occurs. If remeshing causes too much artificial diffusion, the error in the strains will grow, and the mesh will not return to its original shape when external forces are removed.

To control artificial diffusion, it is not only necessary to limit the amount of remeshing; it is also important to keep the tetrahedra small in regions where the strain field has a large gradient. Rather than try to estimate local strain gradients from a piecewise constant strain approximation, I use the strain as a proxy for its gradient, on the assumption that the gradient is large where the strain is large. For the examples in this section, this approximation is reasonable. I use the strains to determine a space-varying ideal edge length for size control during mesh improvement (see Chapter 6). Larger strains locally engender smaller tetrahedra.

The coupling between the simulator and the dynamic mesher requires communication in both directions. The simulator performs a timestep and sends updated vertex positions and tetrahedron strains to the mesher. The mesher sends back a new mesh. The simulator interpolates the strain field from the old mesh to the new. Artificial diffusion introduces error wherever tetrahedra have changed, and the error is greatest where the strain gradient and the tetrahedra are both large.

To demonstrate that my dynamic mesh improvement schedule can introduce substantially less error than my static mesh improvement schedule (or remeshing from scratch), Figure 9.8 shows an elastic ball placed on a plane. The ball first settles into the plane under its own weight; then another plane moves down from above to crush it. Finally, the upper plane disappears and the top of the sphere bounces back. As the sphere is crushed, tetrahedra at the top and the bottom of the ball deform and dihedral angles are pushed outside the acceptable range of 10° to 170°. Dynamic mesh improvement fixes the bad tetrahedra while employing size control that prevents tetrahedra from coarsening where the sphere is compressed. After the top plane is removed, the sphere bounces back.

Figure 9.9 shows the same simulation as Figure 9.8, but instead of using the dynamic improvement schedule described in Section 8.2 to enforce the dihedral angle bounds, it uses the static schedule from Section 4.4 (still with strain-directed size control). The static schedule is modified to terminate as soon as the angle bounds are met. With the static schedule, much more remeshing occurs to fix just a few bad tetrahedra—so much so that interpolation errors in the top and bottom of the ball leave it permanently dented. It is visually clear that excessive remeshing causes significant error to accumulate through artificial diffusion, whereas my dynamic meshing schedule mitigates this error enough that it is not perceptible.

The elastic simulator I use is stable even in the presence of inverted tetrahedra. This stability is unusual—most simulators crash with degenerate or inverted tetrahedra. Nevertheless, the numerical results of a simulation that has inverted tetrahedra cannot be trusted. Figure 9.10 depicts a simulation in which a thin rod pokes deeply into an elastic bunny mesh, with no dynamic mesh improvement. The rod is modeled as a single vertex on the chest being pushed deeply into the mesh interior. After 49 timesteps, tetrahedra near the poked vertex invert, and remain inverted throughout the simulation. Figure 9.11 shows the same simulation, but here dynamic improvement keeps all the dihedral angles between 10° and 170°. These bounds ensure that no tetrahedra can invert—a critical guarantee for most simulations in practice. Figure 9.12 shows the same simulation again but with the minimum tetrahedron quality maintained by the static improvement schedule, which performs more remeshing.

In Figure 9.13, a sphere falls from a height onto a plane and then bounces back up. It deforms as it strikes the plane, and the dynamic improvement schedule keeps all the dihedral angles between 10° and 170°. Very few tetrahedra stray outside these bounds, and so the remeshed volume is small. Figure 9.14 shows the same simulation using the static improvement schedule to maintain quality. As in Figure 9.9, interpolation error leaves the sphere permanently dented.

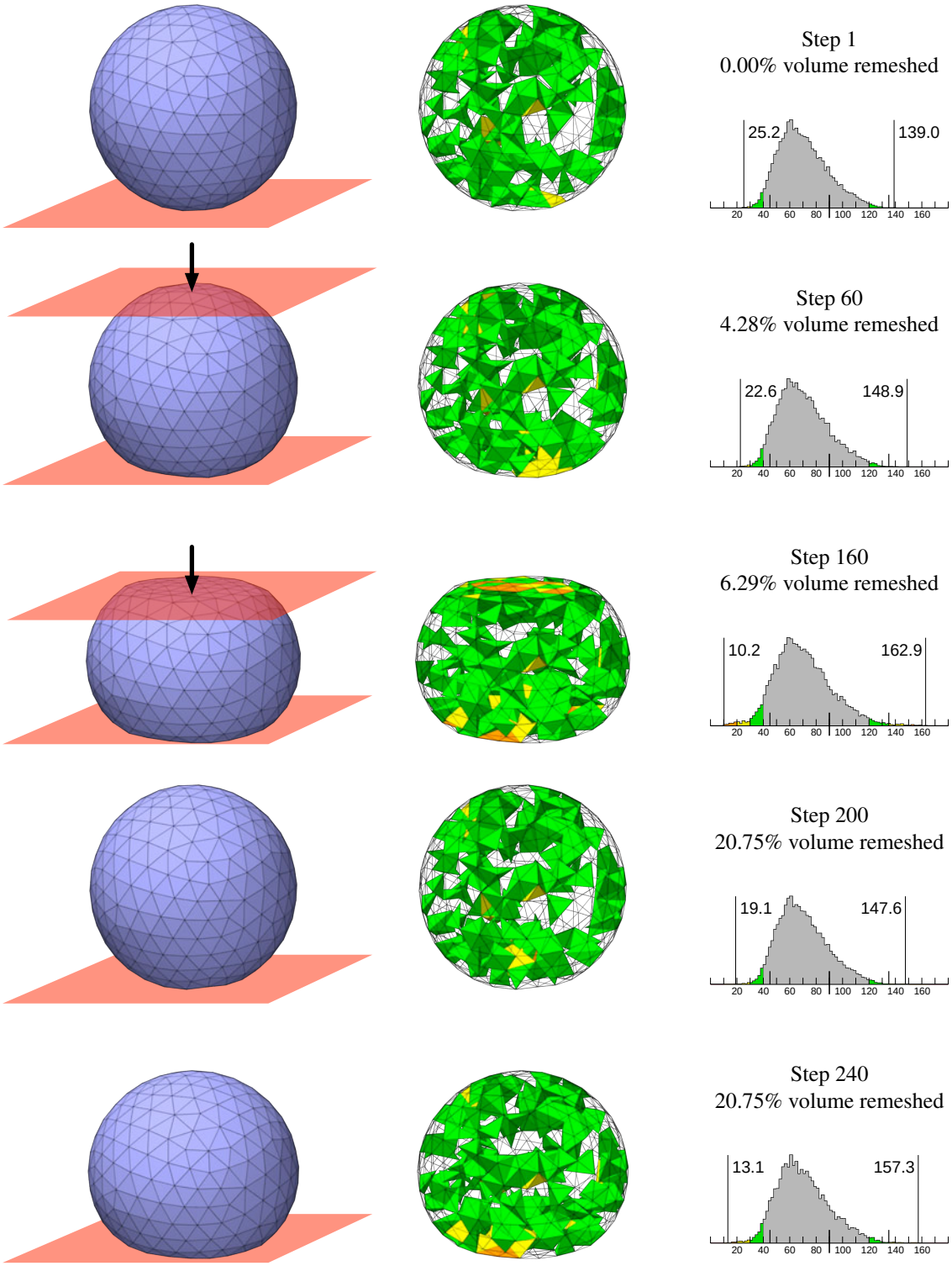


Figure 9.8. A sphere is crushed between two planes, and then the top plane is removed. At each step, dynamic remeshing keeps all the dihedral angles between 10° and 170°.

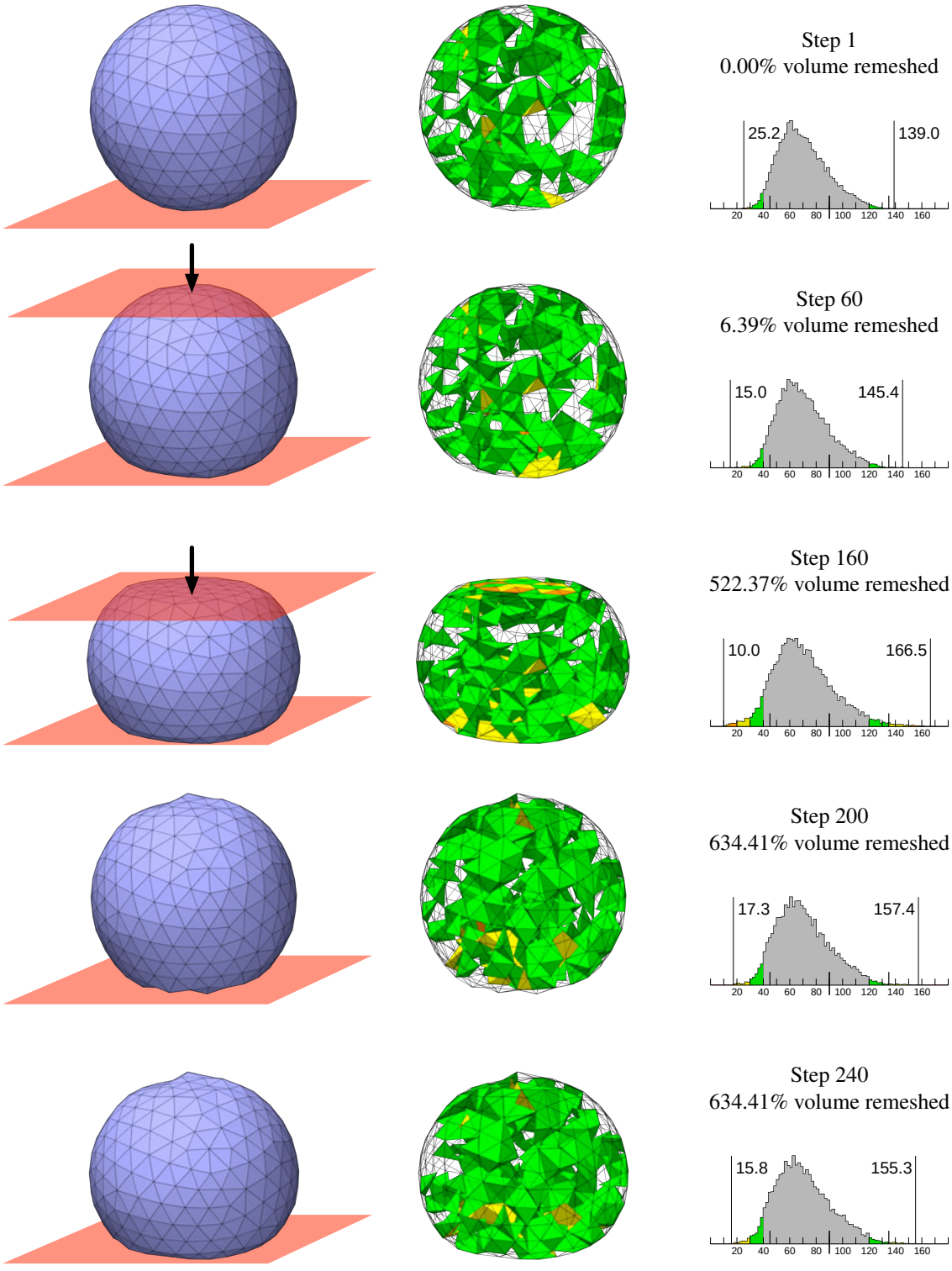


Figure 9.9. A sphere is crushed between two planes as in Figure 9.8, but the mesh is improved by the static improvement schedule of Section 4.4 instead of the dynamic improvement schedule. At each step, static remeshing keeps all the dihedral angles between 10° and 170°.

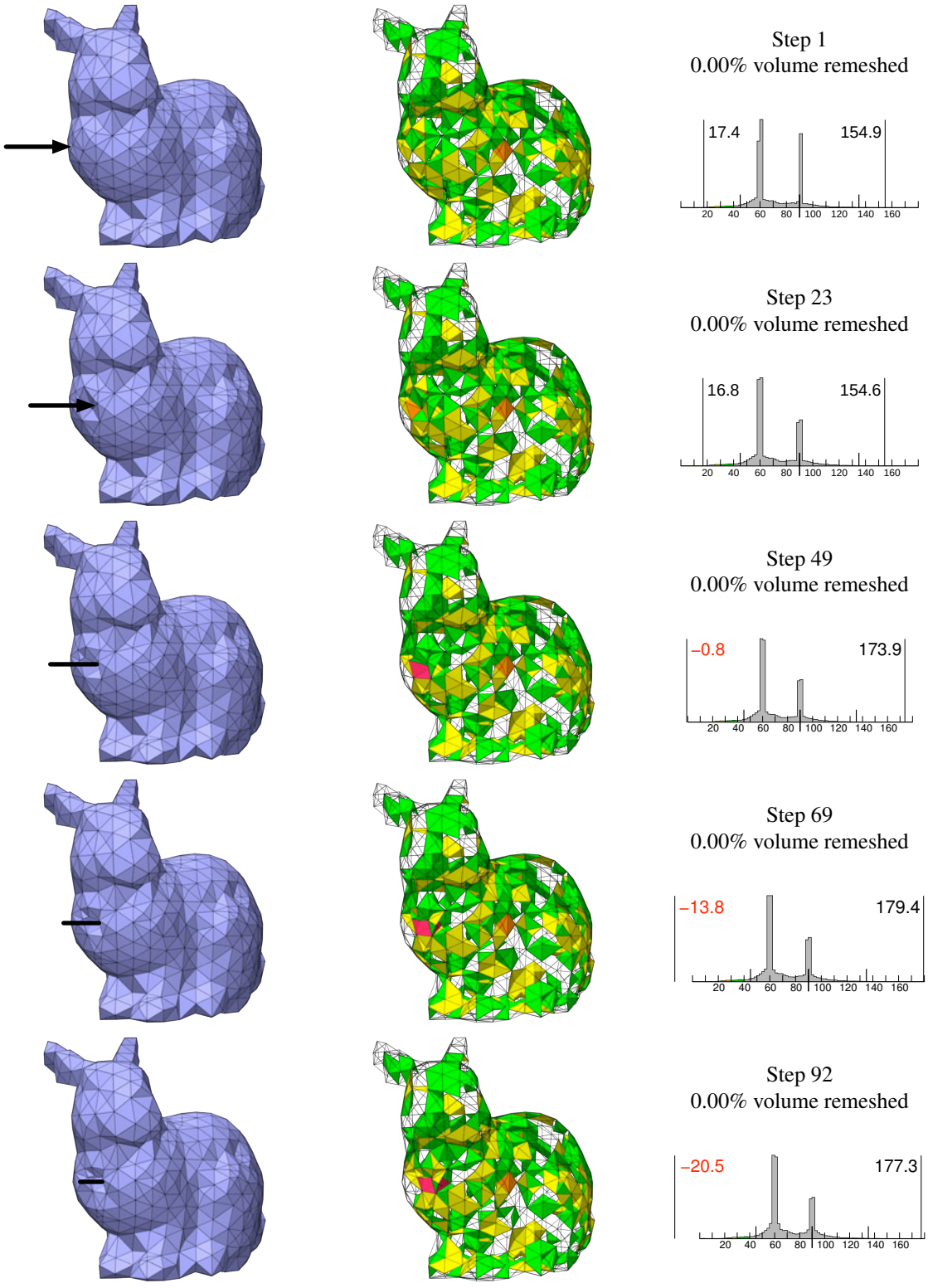


Figure 9.10. A bunny mesh has a vertex on its chest forced into the interior of the mesh. Tetrahedra invert near the forced vertex. No mesh improvement takes place.

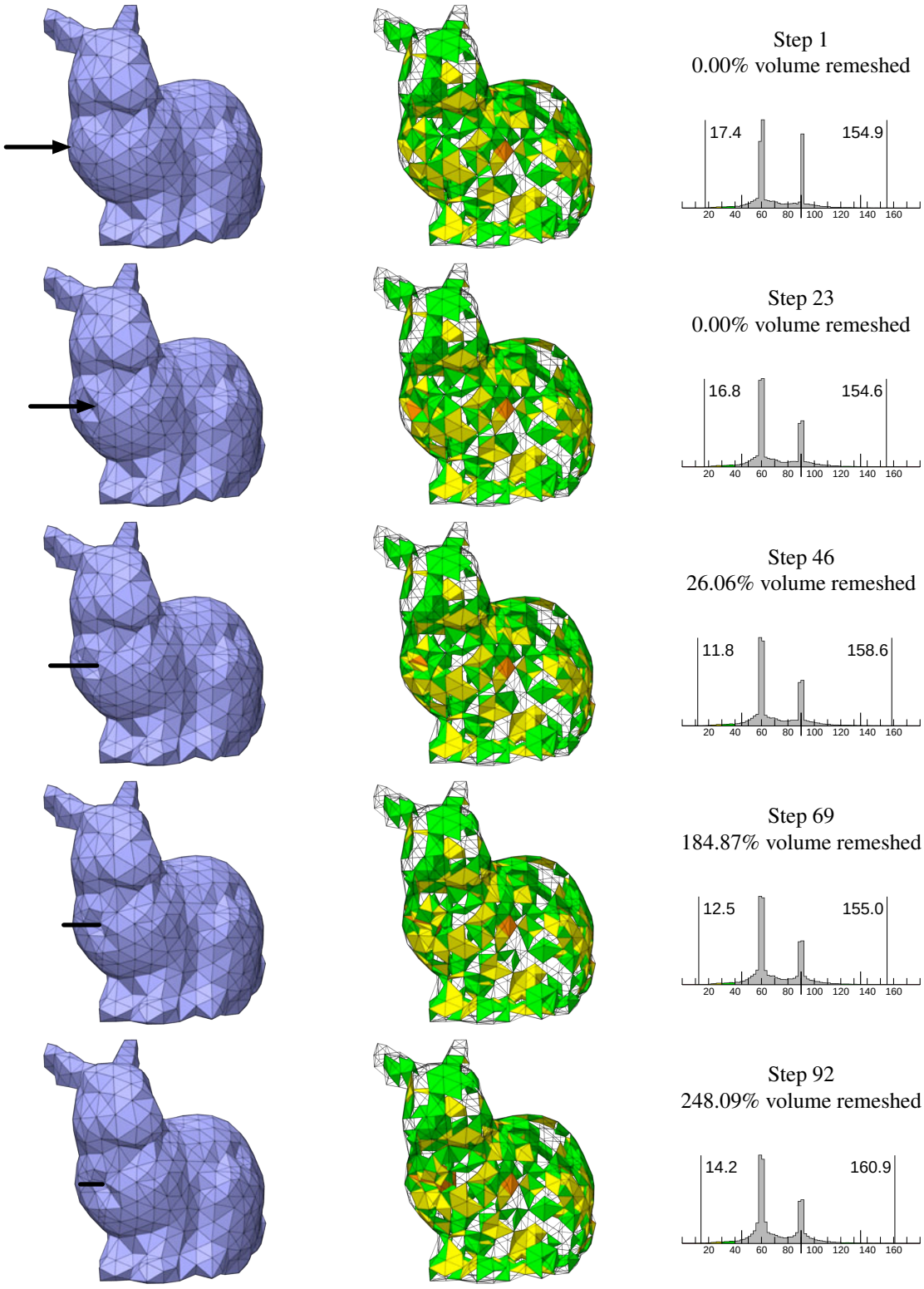


Figure 9.11. As in Figure 9.10, a bunny has a vertex forced into its chest. Dynamic improvement keeps all the dihedral angles between 10° and 170°, preventing any tetrahedra from inverting.

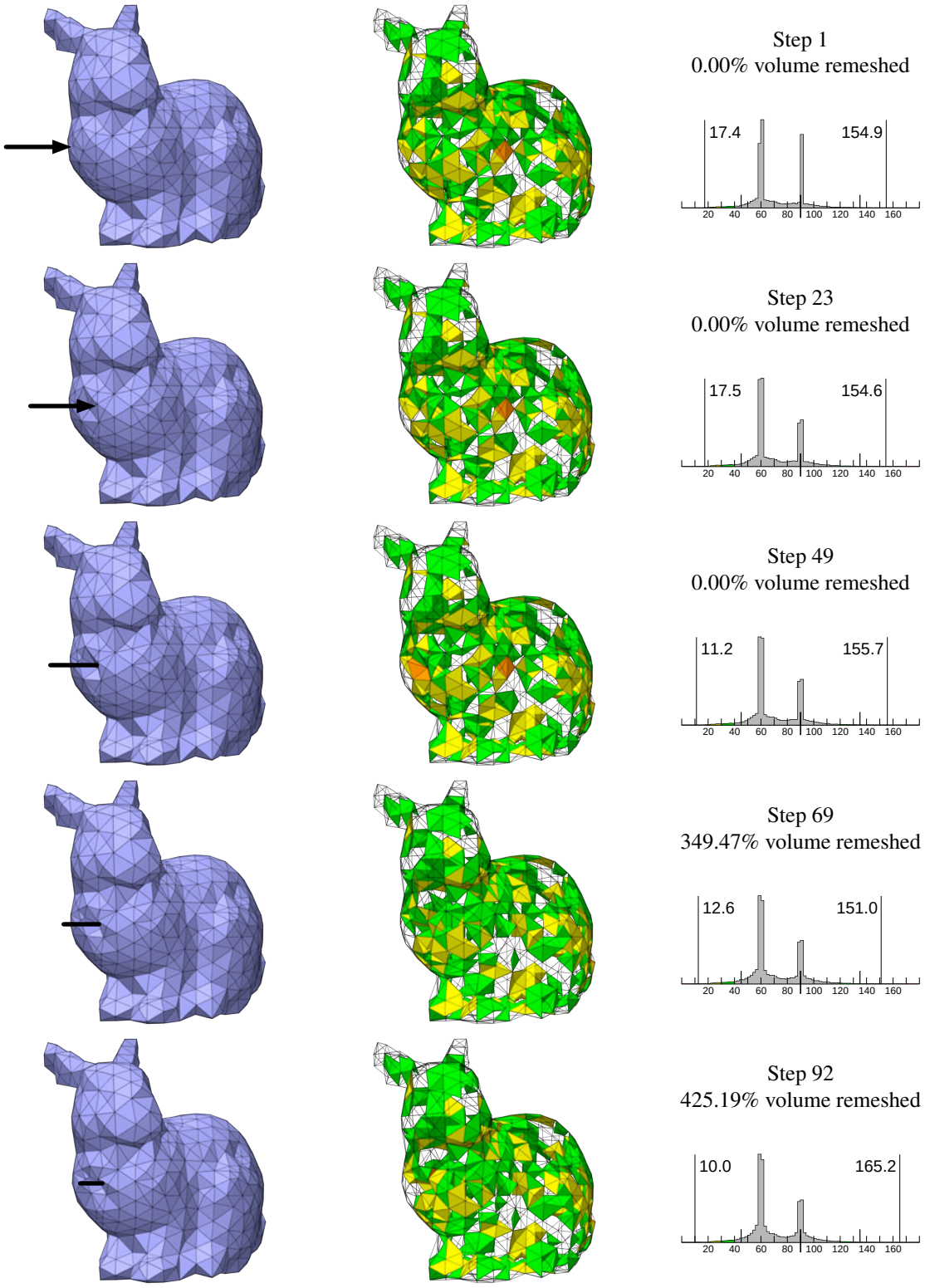


Figure 9.12. The bunny again has a vertex forced into its chest, but the static (not dynamic) improvement schedule of Section 4.4 keeps all the dihedral angles between 10° and 170°.

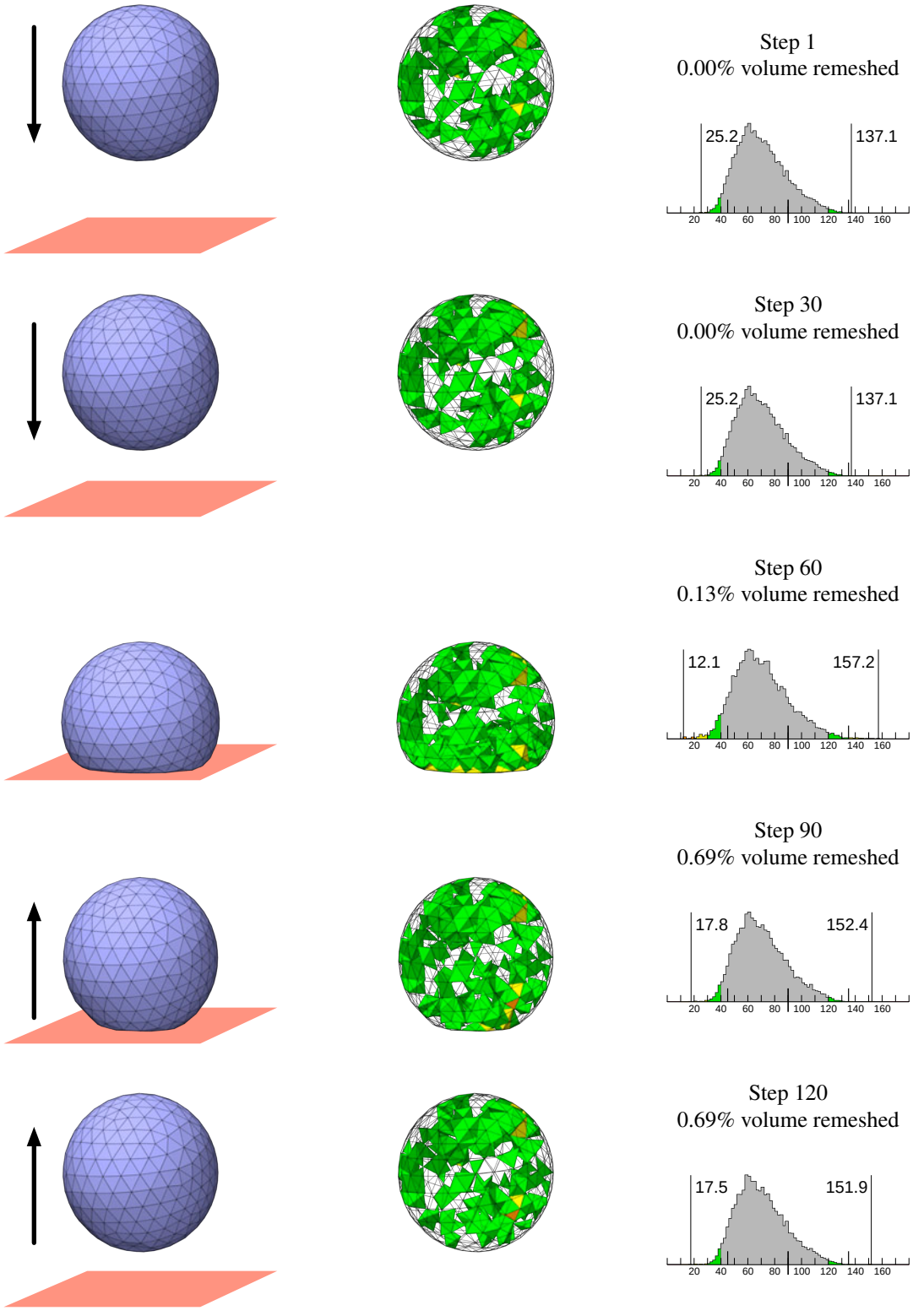


Figure 9.13. A ball falls on to a plane and bounces back. Dynamic improvement keeps all the dihedral angles between 10° and 170°.

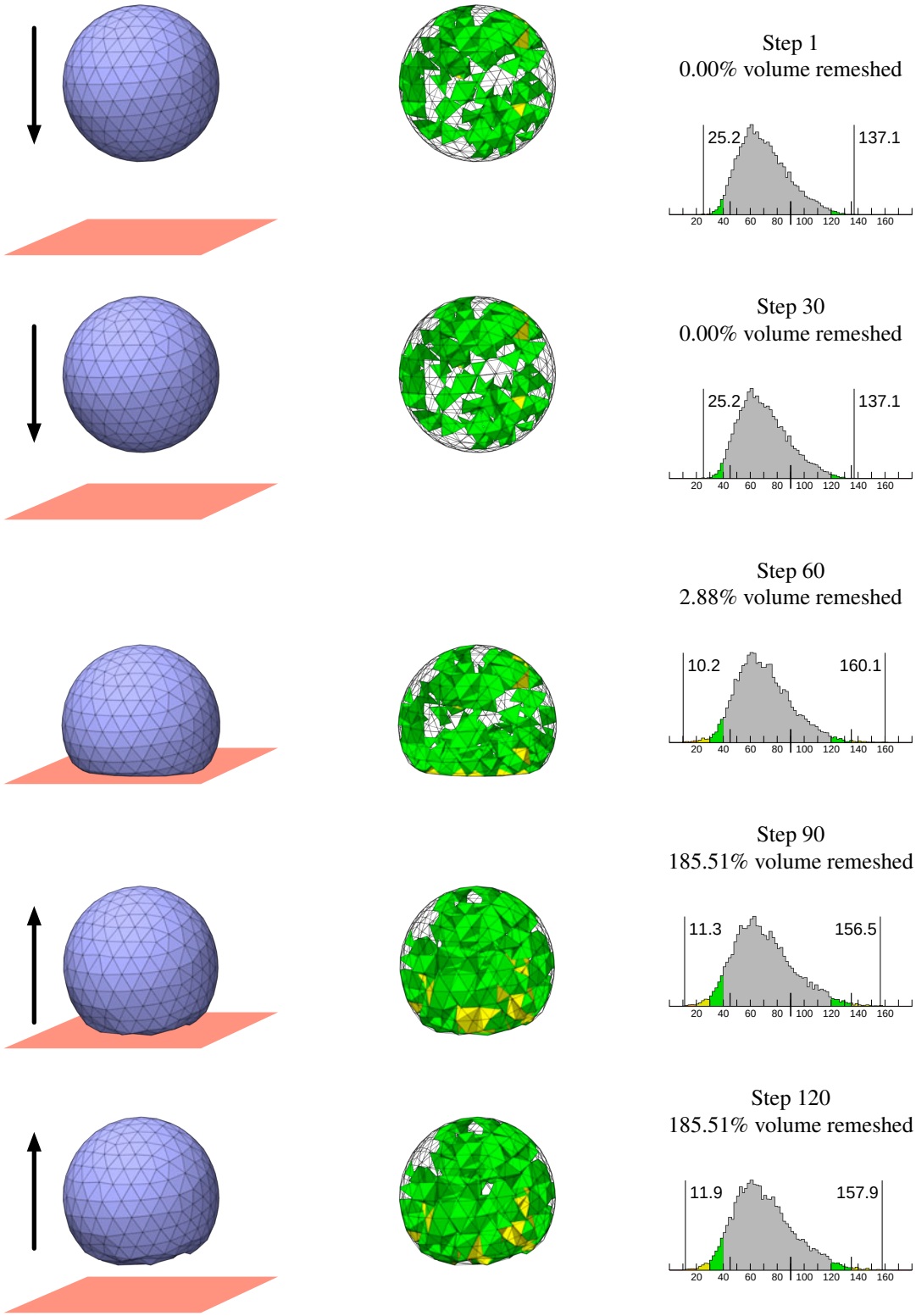


Figure 9.14. A ball falls on to a plane and bounces back as in Figure 9.13, but the static improvement schedule of Section 4.4 maintains the quality instead of the dynamic improvement schedule. Static improvement terminates when all the dihedral angles are between 10° and 170°.

Chapter 10

Conclusions and Future Work

My static improvement schedule offers the chance to rescue tetrahedral meshes that might previously have been unusable, and pushes the quality of already good meshes to far higher quality than I have seen reported elsewhere. The improvement operations I discuss can create high-quality anisotropic tetrahedral meshes, a problem for which there is no guaranteed-quality algorithm. Dynamic mesh improvement opens the door for more practical simulations of deforming domains. There remain many directions this work could be extended to the benefit of practitioners of mesh generation and numerical simulation.

10.1 Handling small domain angles

Some domains are impossible to fill with only good quality tetrahedra. If two faces of a domain's boundary meet at a small dihedral angle, there must be a tetrahedron in the mesh with a dihedral angle at least as small. In a simulation, a mesh may start with no small domain angles but develop them as it deforms. A complete mesh improvement algorithm would identify regions where no more progress can be made, halt improvement efforts there, and alert the user to the difficulty, perhaps offering to repair the poor elements at the cost of changing the domain shape. The algorithm should still improve the elements that can be improved, which means it has to know how to

tell the difference. Telling the difference is sometimes difficult, because some tetrahedra cannot be improved despite having no edge along a small domain angle.

10.2 Better termination criteria for mesh improvement

The static improvement schedule described in Chapter 4 does a good job of improving tetrahedral meshes, but most of the gains in quality occur early during improvement. The majority of the computational effort is spent on the last 10–20% of quality improvement, as Tables 5.8–5.10 show. A better schedule would predict when further improvement passes will have little effect on mesh quality, and quit then.

10.3 Smoothing boundary vertices on smooth surfaces

Constrained vertex smoothing (Section 3.2.4) provides a way to smooth vertices on flat boundaries without changing the domain shape. Quadric smoothing (Section 3.2.5) can smooth vertices on the boundary of curved domains that have been approximated using triangles, but it changes the domain shape in the process.

Sometimes users have access to the curved domain surfaces from which a mesh was generated. CAD software often integrates domain design (using parametric primitives) with mesh generation and improvement, so it seems natural for the smoothing algorithm to refer to the original domain definition rather than just a triangulated approximation. When this information is available, vertices should be smoothed exactly along the curved surfaces that represent them. It is straightforward to modify the constrained vertex smoothing algorithm to do this: only the line search needs to change.

It is more difficult to correctly maintain the shape of a deforming domain, because there is no CAD model for the domain after it begins moving; its shape is determined by the physical process. However, some physical simulation methods can maintain a smooth representation of the domain boundary that has higher resolution than the tetrahedral mesh; an example is the semi-Lagrangian contouring method of Bargteil, Goktekin, O'Brien, and Strain [5], which expresses the boundary at

a timestep as an implicit surface. If the boundary vertices are constrained to be smoothed along this surface, the domain shape can be maintained with better fidelity.

10.4 Mesh improvement as mesh generation

Because I can produce meshes that usually have far better quality than those produced by any previous algorithm for mesh improvement or mesh generation, even when given input meshes with pathologically bad tetrahedra, I think it is possible that algorithms traditionally considered "mesh improvement" might become standalone mesh generators. If the barrier of speed can be overcome, the need to write separate programs for mesh generation and mesh improvement might someday disappear.

Appendix A

Additional Static Mesh Improvement Results

The next ten pages depict ten of my twelve test meshes optimized with four different quality measures as objective functions. In order, these meshes are Cube1K, Cube10K, House, P, TFire, Tire, Rand2, Cow, Dragon, and Stator. The other two meshes appear in Chapter 5: Rand1 in Table 5.5 and StGallen in Table 5.6.

The histograms tabulate, from top to bottom, dihedral angles, radius ratios (times 3), and the volume-length measure $6\sqrt{2}V/\ell_{\text{rms}}^3$. The latter two measures are normalized so an equilateral tetrahedron has quality 1 and a degenerate tetrahedron has quality 0. Each histogram shows the measure of the worst tetrahedron in the mesh. The dihedral angles histograms show both the smallest and largest dihedral angle in each mesh. Histograms are normalized so the tallest bar always has the same height; absolute numbers of tetrahedra cannot be compared between histograms.

Above each dihedral angle histogram is a depiction of the mesh in which tetrahedra with dihedral angles under 40° are colored by their minimum angles. Red tetrahedra have dihedral angles under 10°, orange under 20°, yellow under 30°, green under 40°, and better tetrahedra do not appear.

Below each angle histogram is a depiction coloring the tetrahedra with dihedral angles over

120°. Navy blue tetrahedra have dihedral angles over 165°, turquoise over 150°, violet over 135°, medium blue over 120°, and better tetrahedra do not appear.

Above the radius ratio and volume-length histograms, tetrahedra are colored by their qualities. Red tetrahedra have quality under 0.15, orange under 0.3, yellow under 0.45, green under 0.6, and better tetrahedra do not appear.

Running times are given for a Mac Pro with a 2.66 GHz Intel Xeon processor.

Table A.1.

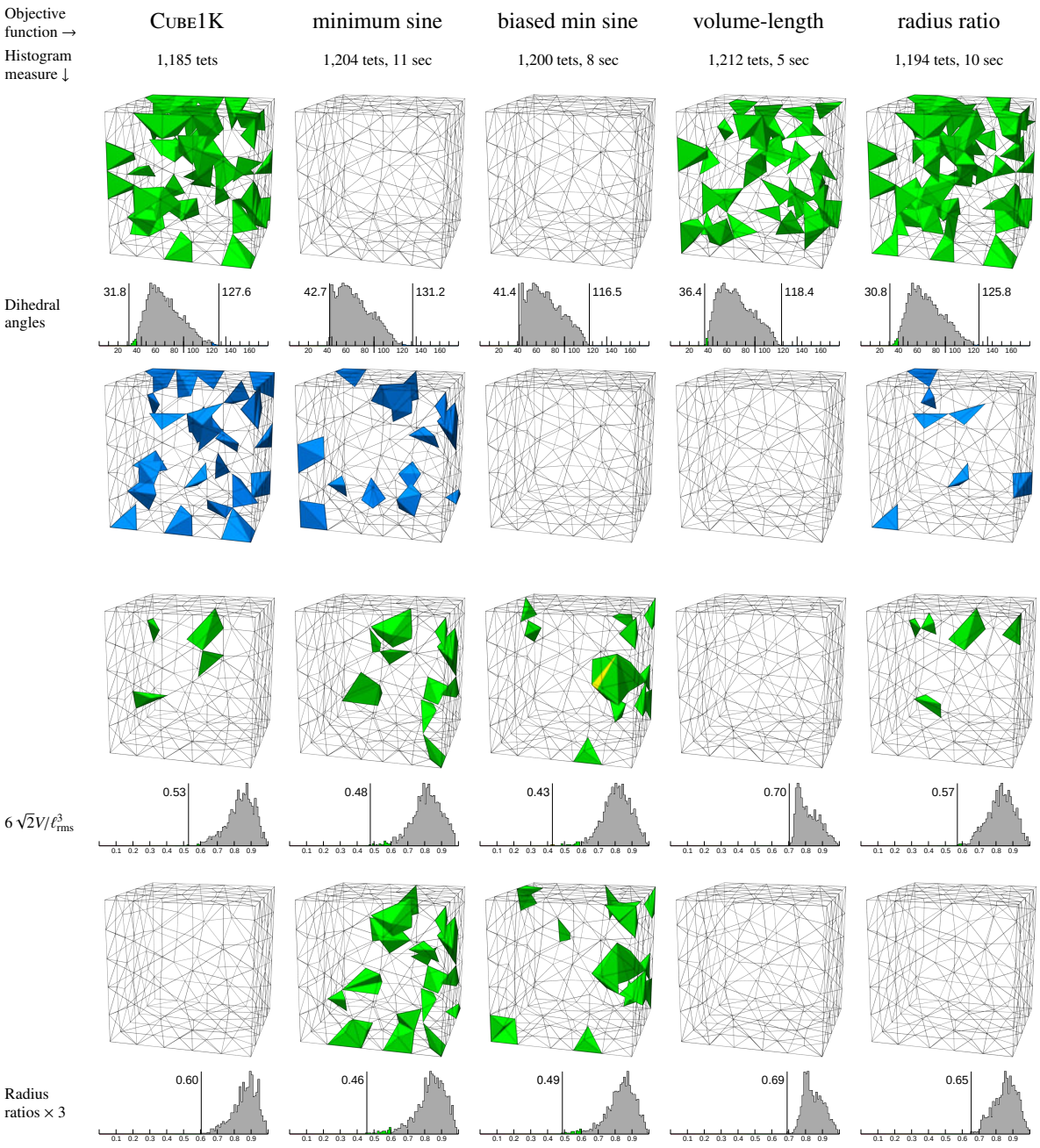

Objective function →
Histogram measure ↓
CUBE1K
minimum sine
biased min sine
volume-length
radius ratio
1,185 tets
1,204 tets, 11 sec
1,200 tets, 8 sec
1,212 tets, 5 sec
1,194 tets, 10 sec
Dihedral angles
31.8
127.6
42.7
131.2
41.4
116.5
36.4
118.4
30.8
125.8
$6\sqrt{2}V/\ell_{\mathrm{rms}}^3$
0.53
0.48
0.43
0.70
0.57
Radius ratios × 3
0.60
0.46
0.49
0.69
0.65

Table A.2.

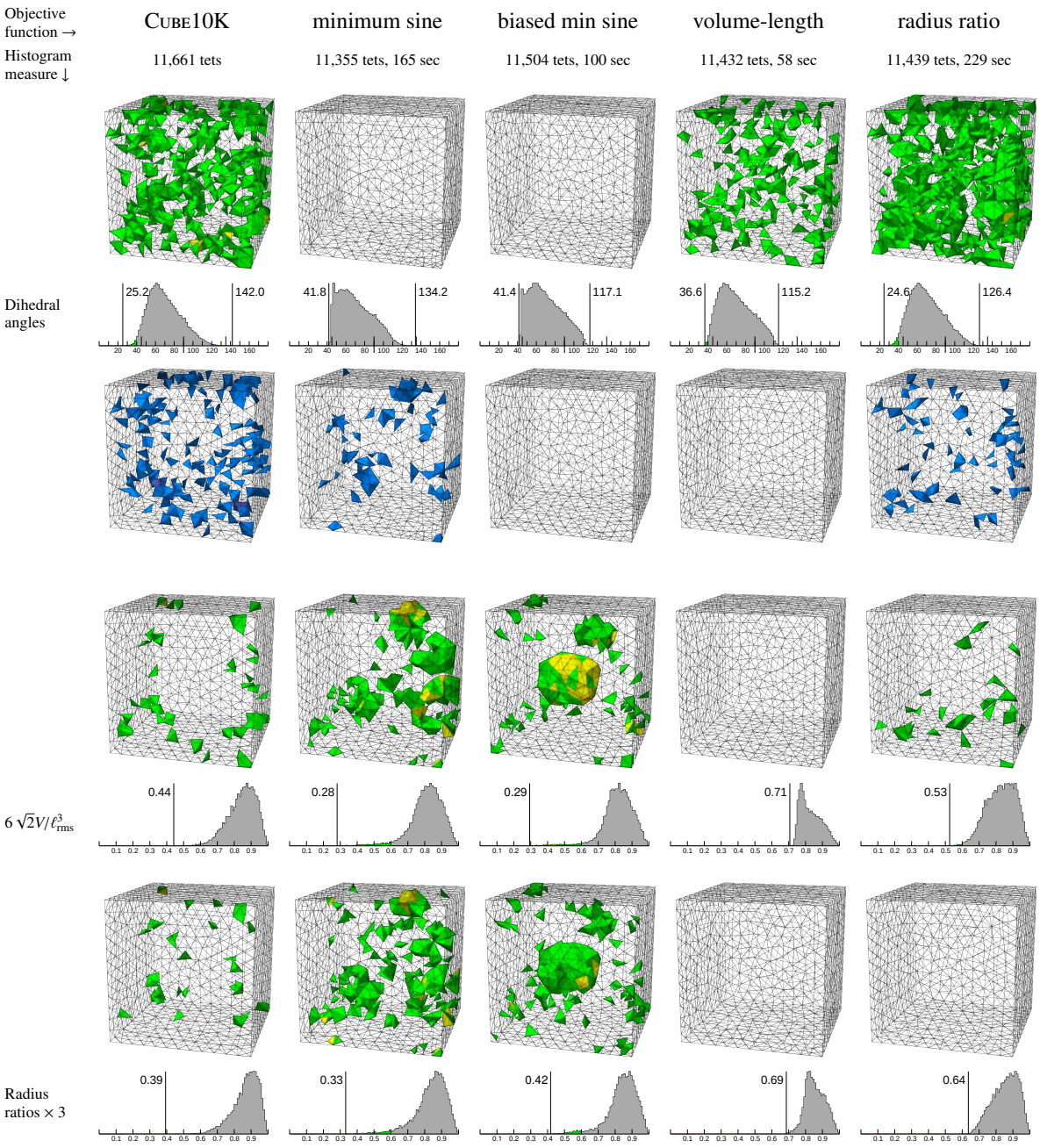

Objective function →
CUBE10K
minimum sine
biased min sine
volume-length
radius ratio
Histogram measure ↓
11,661 tets
11,355 tets, 165 sec
11,504 tets, 100 sec
11,432 tets, 58 sec
11,439 tets, 229 sec
Dihedral angles
25.2
142.0
41.8
134.2
41.4
117.1
36.6
115.2
24.6
126.4
$6\sqrt{2}V/\ell_{\mathrm{rms}}^3$
0.44
0.28
0.29
0.71
0.53
Radius ratios × 3
0.39
0.33
0.42
0.69
0.64

Table A.3.

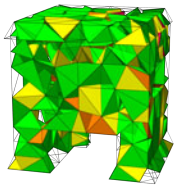
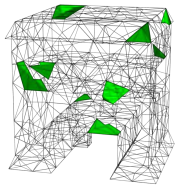
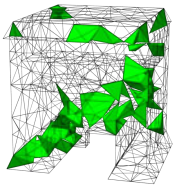
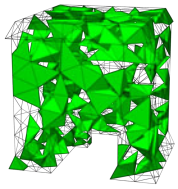
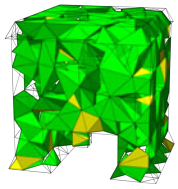
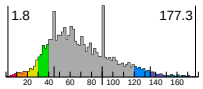
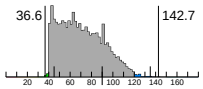
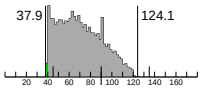
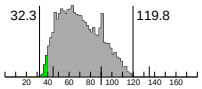
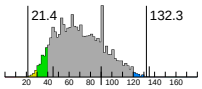
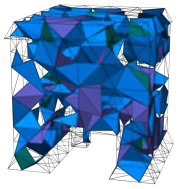
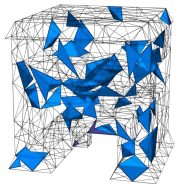
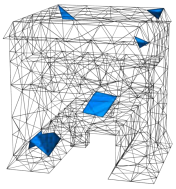
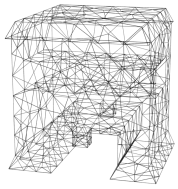
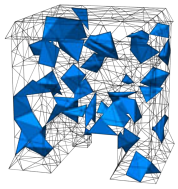
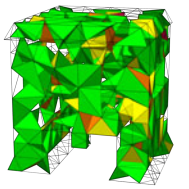
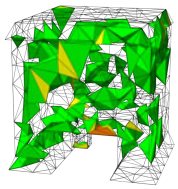
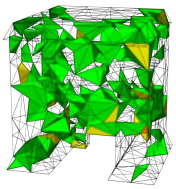
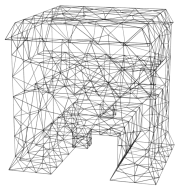
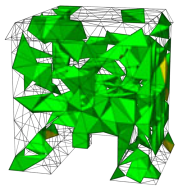
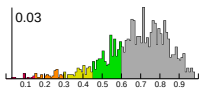
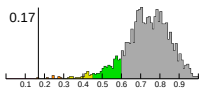
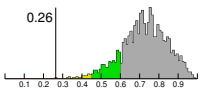
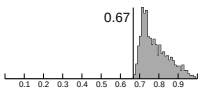
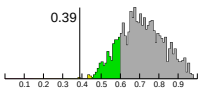
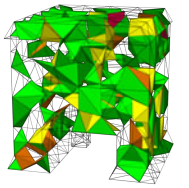
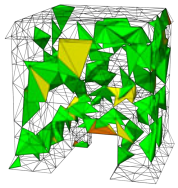
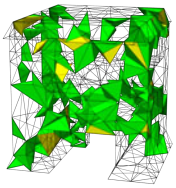
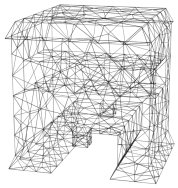
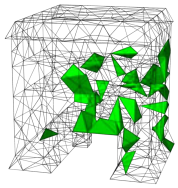
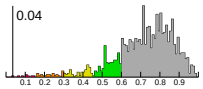
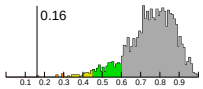
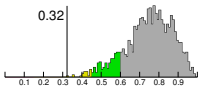
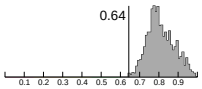
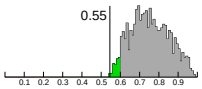
Objective function → / Histogram measure ↓	House	minimum sine	biased min sine	volume-length	radius ratio
	1,390 tets	1,735 tets, 36 sec	1,751 tets, 42 sec	1,623 tets, 50 sec	1,387 tets, 14 sec
					
Dihedral angles					
					
					
$6\sqrt{2}V/\ell_{\mathrm{rms}}^3$					
					
Radius ratios × 3					

Table A.4.

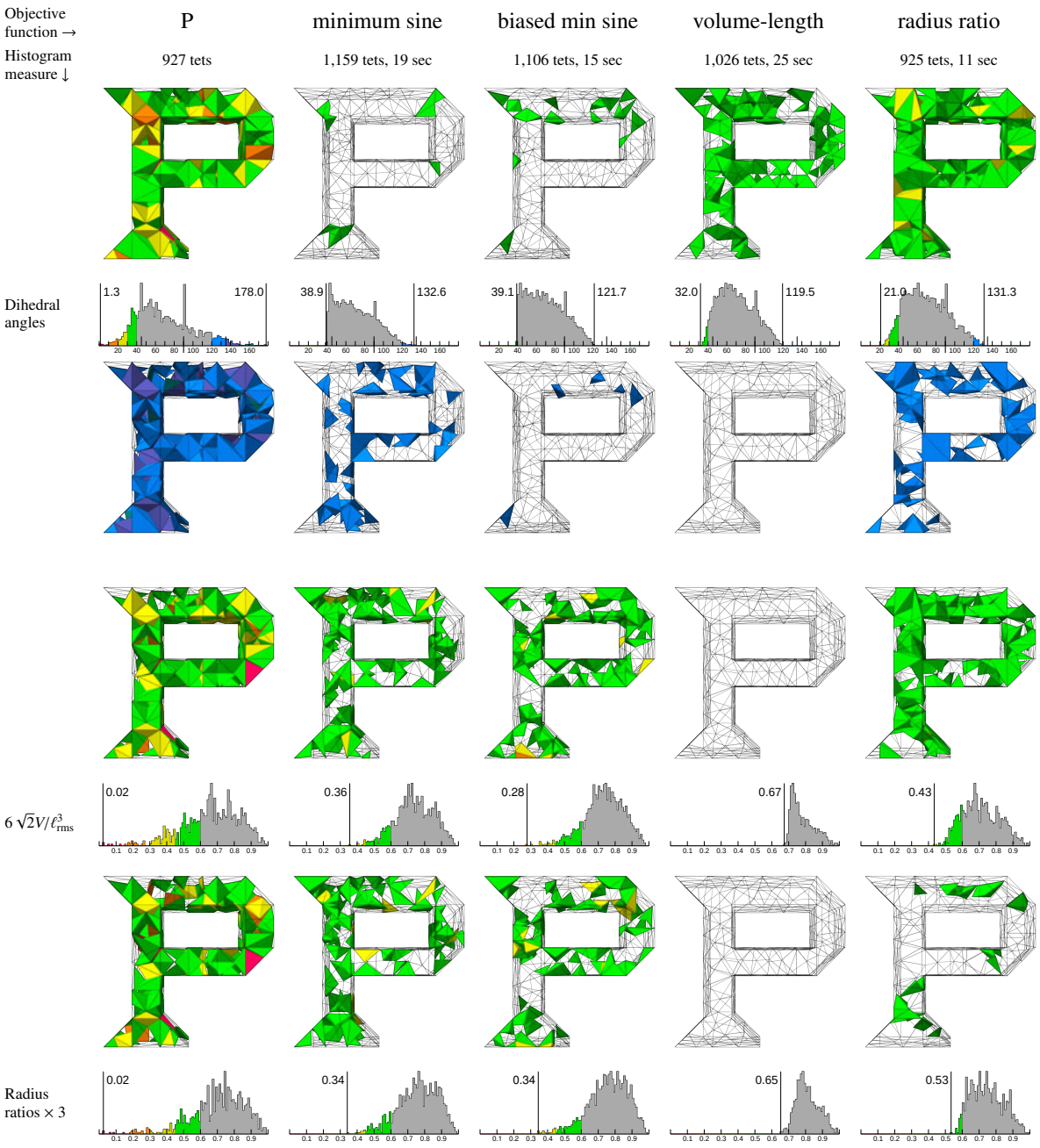

Objective function →
P
minimum sine
biased min sine
volume-length
radius ratio
Histogram measure ↓
927 tets
1,159 tets, 19 sec
1,106 tets, 15 sec
1,026 tets, 25 sec
925 tets, 11 sec
Dihedral angles
1.3
178.0
38.9
132.6
39.1
121.7
32.0
119.5
21.0
131.3
$6\sqrt{2}V/\ell_{\mathrm{rms}}^3$
0.02
0.36
0.28
0.67
0.43
Radius ratios × 3
0.02
0.34
0.34
0.65
0.53

Table A.5.

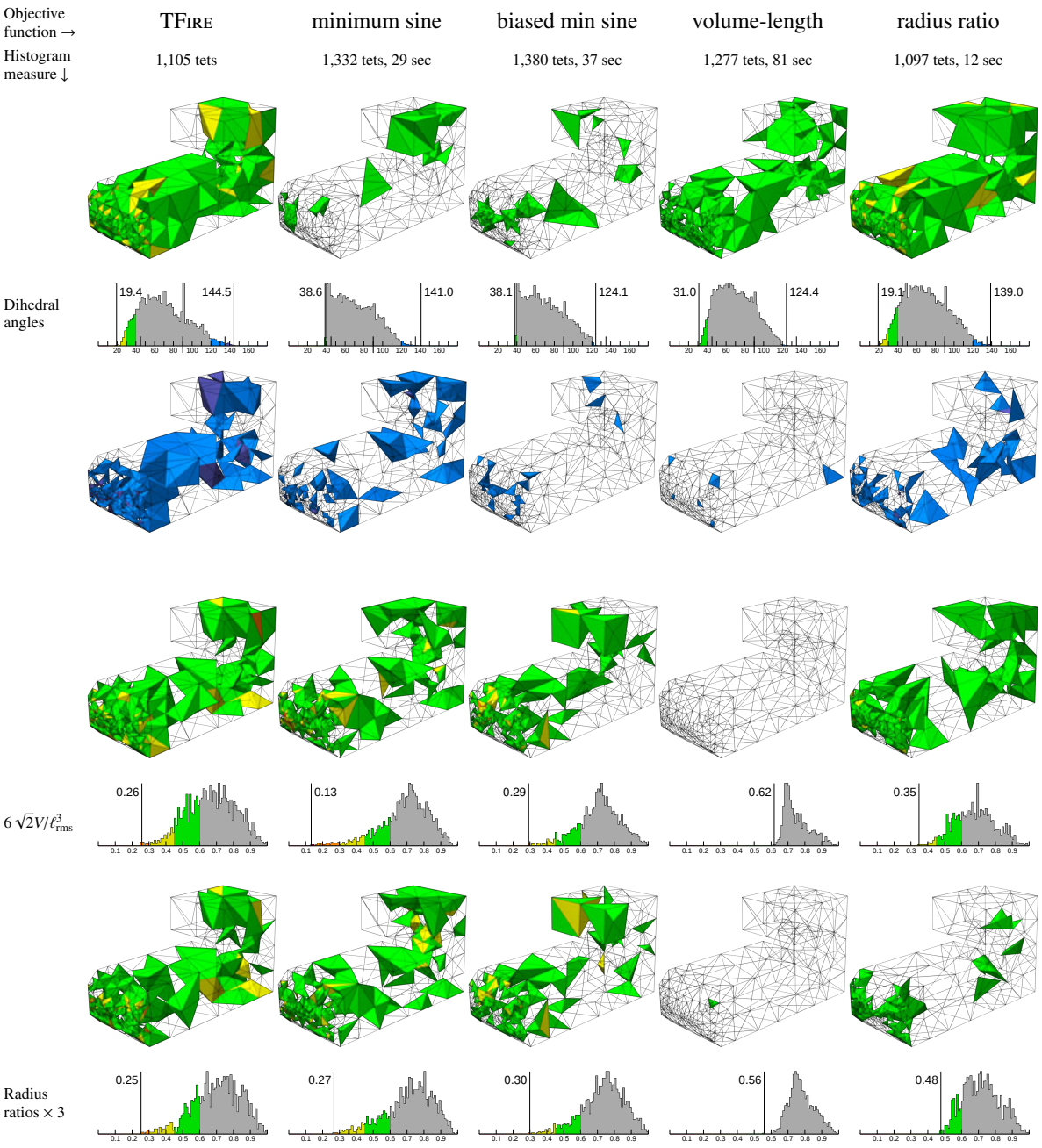

Objective function →
TFire
minimum sine
biased min sine
volume-length
radius ratio
Histogram measure ↓
1,105 tets
1,332 tets, 29 sec
1,380 tets, 37 sec
1,277 tets, 81 sec
1,097 tets, 12 sec
Dihedral angles
19.4
144.5
38.6
141.0
38.1
124.1
31.0
124.4
19.1
139.0
$6\sqrt{2}V/\ell_{\mathrm{rms}}^3$
0.26
0.13
0.29
0.62
0.35
Radius ratios × 3
0.25
0.27
0.30
0.56
0.48

Table A.6.

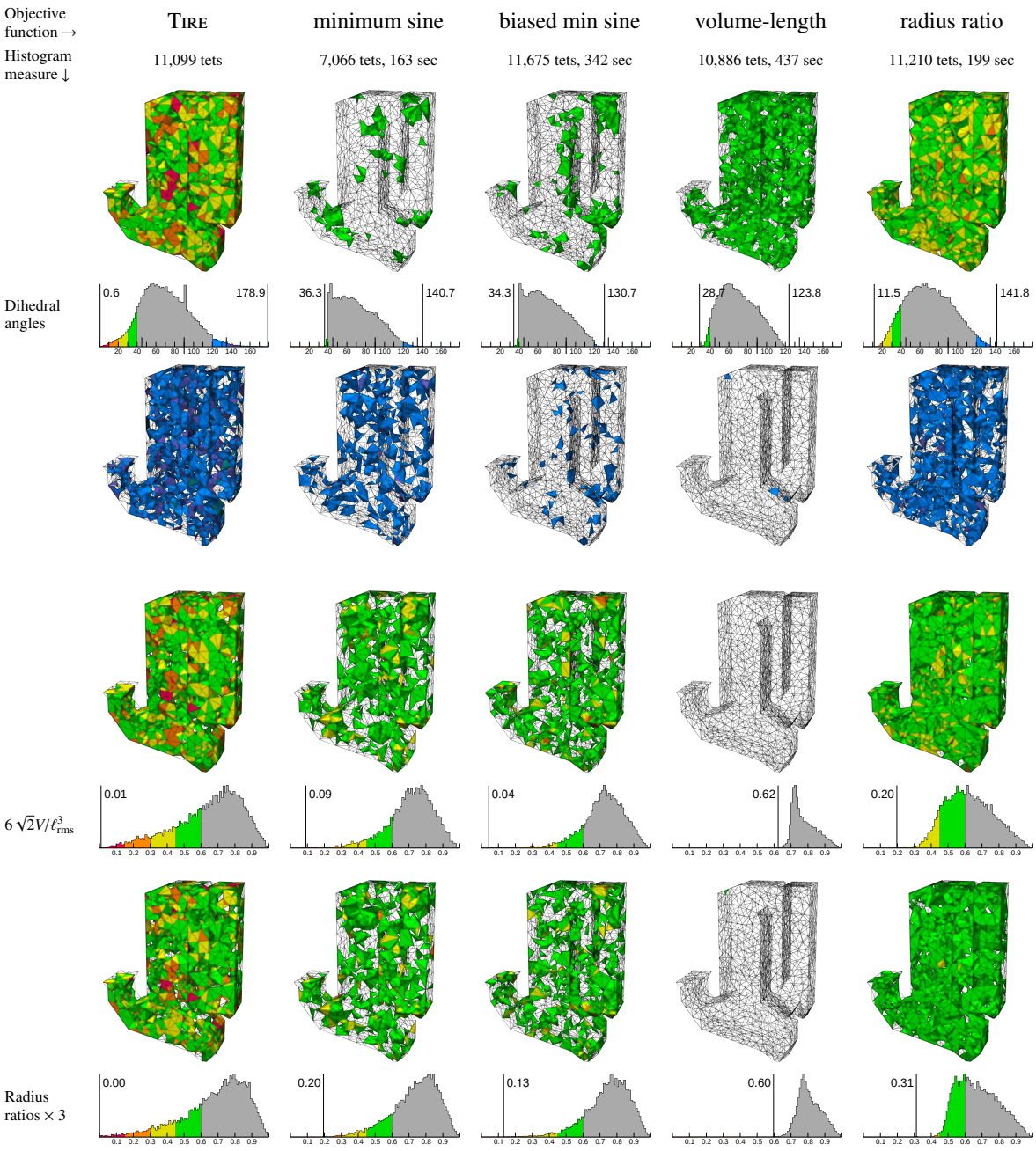

Objective function →
TIRE
minimum sine
biased min sine
volume-length
radius ratio
Histogram measure ↓
11,099 tets
7,066 tets, 163 sec
11,675 tets, 342 sec
10,886 tets, 437 sec
11,210 tets, 199 sec
Dihedral angles
0.6
178.9
36.3
140.7
34.3
130.7
28.7
123.8
11.5
141.8
$6\sqrt{2}V/\ell_{\text{rms}}^3$
0.01
0.09
0.04
0.62
0.20
Radius ratios × 3
0.00
0.20
0.13
0.60
0.31

Table A.7.

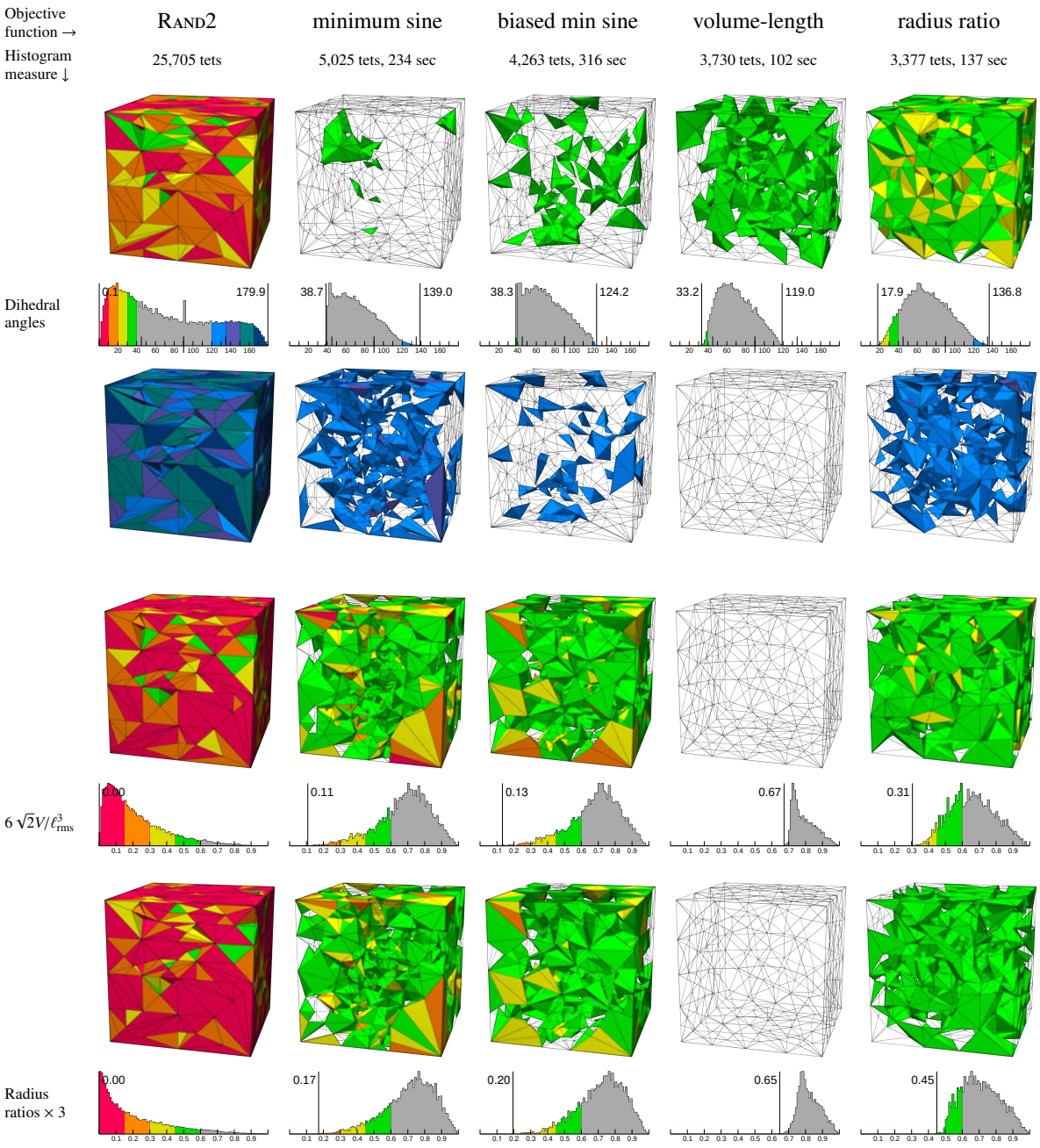

Objective function →
RAND2
minimum sine
biased min sine
volume-length
radius ratio
Histogram measure ↓
25,705 tets
5,025 tets, 234 sec
4,263 tets, 316 sec
3,730 tets, 102 sec
3,377 tets, 137 sec
Dihedral angles
0.1
179.9
38.7
139.0
38.3
124.2
33.2
119.0
17.9
136.8
$6\sqrt{2}V/\ell_{\mathrm{rms}}^3$
0.00
0.11
0.13
0.67
0.31
Radius ratios × 3
0.00
0.17
0.20
0.65
0.45

Table A.8.

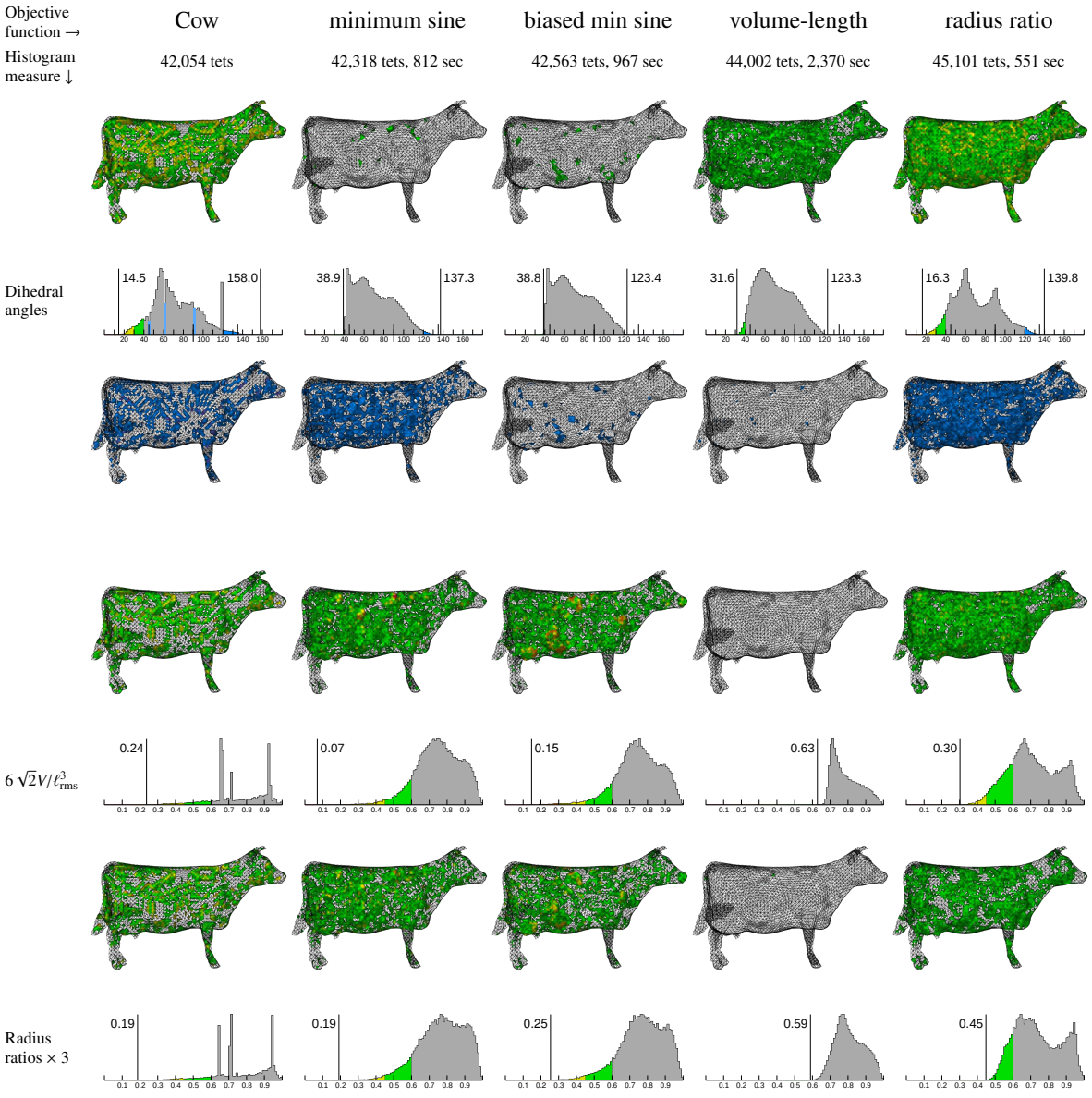

Objective function →
Histogram measure ↓
Cow
minimum sine
biased min sine
volume-length
radius ratio
42,054 tets
42,318 tets, 812 sec
42,563 tets, 967 sec
44,002 tets, 2,370 sec
45,101 tets, 551 sec
Dihedral angles
14.5
158.0
38.9
137.3
38.8
123.4
31.6
123.3
16.3
139.8
$6\sqrt{2}V/\ell^3_{\mathrm{rms}}$
0.24
0.07
0.15
0.63
0.30
Radius ratios × 3
0.19
0.19
0.25
0.59
0.45

Table A.9.

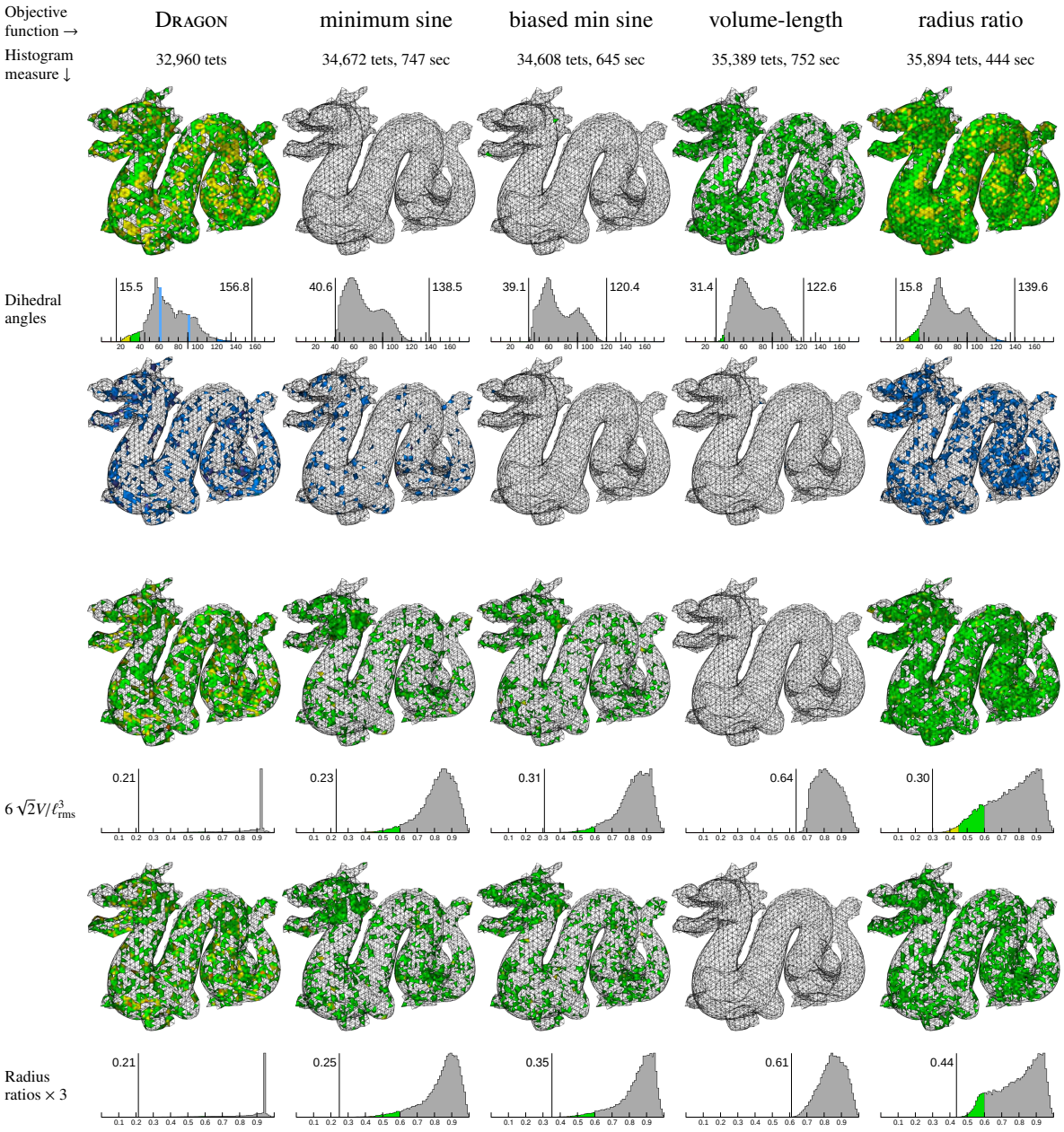
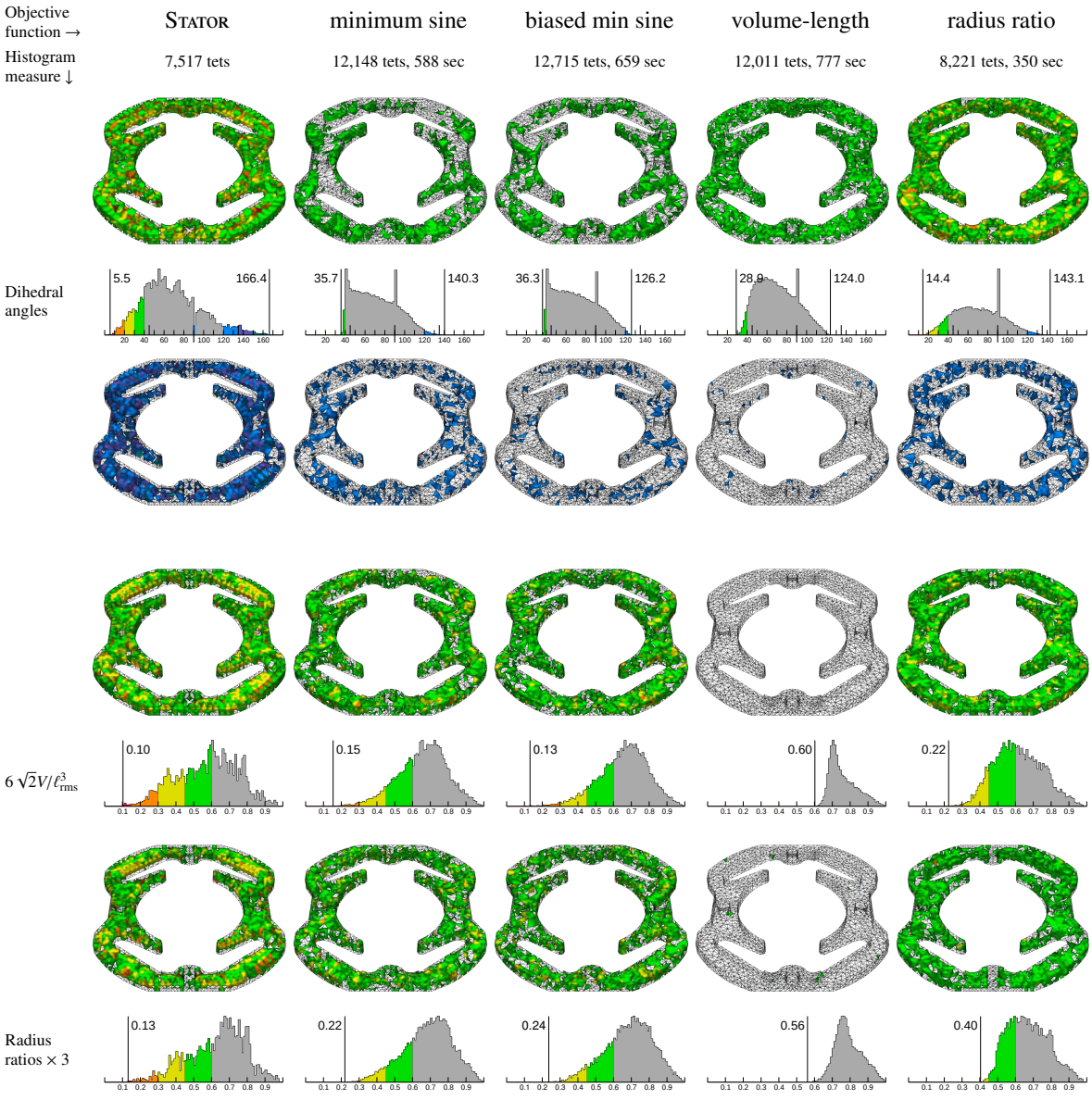

Objective function →
Histogram measure ↓
DRAGON
minimum sine
biased min sine
volume-length
radius ratio
32,960 tets
34,672 tets, 747 sec
34,608 tets, 645 sec
35,389 tets, 752 sec
35,894 tets, 444 sec
Dihedral angles
15.5
156.8
40.6
138.5
39.1
120.4
31.4
122.6
15.8
139.6
$6\sqrt{2}V/\ell_{\mathrm{rms}}^3$
0.21
0.23
0.31
0.64
0.30
Radius ratios × 3
0.21
0.25
0.35
0.61
0.44

Table A.10.


Objective function →
Histogram measure ↓
STATOR
7,517 tets
minimum sine
12,148 tets, 588 sec
biased min sine
12,715 tets, 659 sec
volume-length
12,011 tets, 777 sec
radius ratio
8,221 tets, 350 sec
Dihedral angles
5.5
166.4
35.7
140.3
36.3
126.2
28.9
124.0
14.4
143.1
$6\sqrt{2}V/\ell_{\mathrm{rms}}^3$
0.10
0.15
0.13
0.60
0.22
Radius ratios × 3
0.13
0.22
0.24
0.56
0.40

Bibliography

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

[2] Thomas Apel. *Anisotropic Finite Elements: Local Estimates and Applications*. B. G. Teubner Verlag, Stuttgart, 1999.

[3] N. Aspert, D. Santa-Cruz, and T. Ebrahimi. *Mesh: Measuring errors between surfaces using the Hausdorff distance*. Proceedings of the IEEE International Conference on Multimedia and Expo, volume I, pages 705–708, 2002. `http://mesh.epfl.ch`.

[4] Randolph E. Bank and L. Ridgway Scott. *On the Conditioning of Finite Element Equations with Highly Refined Meshes*. SIAM Journal on Numerical Analysis **26**(6):1383–1394, December 1989.

[5] 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.

[6] Adam W. Bargteil, Chris Wojtan, Jessica K. Hodgins, and Greg Turk. *A finite element method for animating large viscoplastic flow*. ACM Transactions on Graphics **26**(3):16.1–16.8, July 2007.

[7] H. Borouchaki, P. Laug, A. Cherouat, and K. Saanouni. *Adaptive remeshing in large plastic strain with damage*. International Journal for Numerical Methods in Engineering **63**:1–36, 2005.

[8] E. Brière de l'Isle and Paul-Louis George. *Optimization of Tetrahedral Meshes*. Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations, IMA Volumes in Mathematics and its Applications, volume 75, pages 97–128. 1995.

[9] Scott A. Canann, Michael Stephenson, and Ted Blacker. *Optismoothing: An Optimization-Driven Approach to Mesh Smoothing*. Finite Elements in Analysis and Design **13**:185–190, 1993.

[10] David Cardoze, Alexandre Cunha, Gary L. Miller, Todd Phillips, and Noel Walkington. *A Bézier-Based Approach to Unstructured Moving Meshes*. Proceedings of the Twentieth Annual Symposium on Computational Geometry (Brooklyn, New York), pages 310–319. Association for Computing Machinery, June 2004.

[11] James C. Cavendish, David A. Field, and William H. Frey. *An Approach to Automatic Three-Dimensional Finite Element Mesh Generation*. International Journal for Numerical Methods in Engineering **21**(2):329–347, February 1985.

[12] 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.

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

[14] ———. *Guaranteed-Quality Delaunay Meshing in 3D*. Proceedings of the Thirteenth Annual Symposium on Computational Geometry (Nice, France), pages 391–393, June 1997.

[15] R. Courant, K. Friedrichs, and H. Lewy. *Über die partiellen Differenzengleichungen der Mathematischen Physik*. Mathematische Annalen **100**:32–74, August 1928.

[16] Barbara Cutler, Julie Dorsey, and Leonard McMillan. *Simplification and improvement of tetrahedral models for simulation*. Symposium on Geometry Processing, pages 95–104, June 2004.

[17] Ed F. D'Azevedo. *Are Bilinear Quadrilaterals Better than Linear Triangles?* Technical Report ORNL/TM-12388, Computer Science and Mathematics Division, Oak Ridge National Laboratories, Oak Ridge, Tennessee, August 1993.

[18] Hugues L. de Cougny and Mark S. Shephard. *Refinement, Derefinement, and Optimization of Tetrahedral Geometric Triangulations in Three Dimensions*. Unpublished manuscript, 1995.

[19] Tamal K. Dey, Herbert Edelsbrunner, Sumanta Guha, and Dmitry V. Nekhayev. *Topology preserving edge contraction*. Publications de l'Institut Mathematique (Beograd) **60**(80):23–45, 1999.

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

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

[22] Lori A. Freitag, Mark Jones, and Paul Plassman. *An Efficient Parallel Algorithm for Mesh Smoothing*. Fourth International Meshing Roundtable (Albuquerque, New Mexico), pages 47–58, October 1995.

[23] 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.

[24] 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.

[25] Michael Garland. *Quadric-Based Polygonal Surface Simplification*. Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, August 1999. Available as Technical Report CMU-CS-99-105.

[26] Michael Garland and Paul Heckbert. *Surface Simplification Using Quadric Error Metrics*. Computer Graphics (SIGGRAPH '97 Proceedings), pages 209–216, August 1997.

[27] 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.

[28] P. Jamet. *Estimations d'Erreur pour des Élements Finis Droits Presque Dégénérés*. RAIRO Analyse Numérique **10**:43–61, 1976.

[29] Kenneth E. Jansen, Mark S. Shephard, and Mark W. Beall. *On Anisotropic Mesh Generation and Quality Control in Complex Flow Problems*. Tenth International Meshing Roundtable (Newport Beach, California), pages 341–349. Sandia National Laboratories, October 2001.

[30] Barry Joe. *Construction of Three-Dimensional Improved-Quality Triangulations Using Local Transformations*. SIAM Journal on Scientific Computing **16**(6):1292–1307, November 1995.

[31] G. T. Klincsek. *Minimal Triangulations of Polygonal Domains*. Annals of Discrete Mathematics **9**:121–123, 1980.

[32] Bryan M. Klingner, Bryan E. Feldman, Nuttapong Chentanez, and James F. O'Brien. *Fluid Animation with Dynamic Meshes*. ACM Transactions on Graphics **25**(3):820–825, July 2006. Special issue on Proceedings of SIGGRAPH 2006.

[33] Bryan Matthew Klingner and Jonathan Richard Shewchuk. *Aggressive Tetrahedral Mesh Improvement*. Proceedings of the 16th International Meshing Roundtable (Seattle, Washington), pages 3–23, October 2007.

[34] Martin Kraus and Thomas Ertl. *Simplification of nonconvex tetrahedral meshes*. Hierarchical and Geometrical Methods in Scientific Visualization, pages 185–196. Springer-Verlag, 2002.

[35] Michal Křížek. *On the Maximum Angle Condition for Linear Tetrahedral Elements*. SIAM Journal on Numerical Analysis **29**(2):513–520, April 1992.

[36] François Labelle and Jonathan Richard Shewchuk. *Isosurface Stuffing: Fast Tetrahedral Meshes with Good Dihedral Angles*. ACM Transactions on Graphics **26**(3):57.1–57.10, July 2007. Special issue on Proceedings of SIGGRAPH 2007.

[37] S. Mauch, L. Noels, Z. Zhao, and R. Radovitzky. *Lagrangian Simulation of Penetration Environments Via Mesh Healing and Adaptive Optimization*. Proceedings of the 25th Army Science Conference (Orlando, Florida), November 2006.

[38] Carl Ollivier-Gooch. *Coarsening unstructured meshes by edge contraction*. International Journal for Numerical Methods in Engineering **57**:391–414, 2003.

[39] Carl Ollivier-Gooch and Charles Boivin. *Guaranteed-quality simplicial mesh generation with cell size and grading control*. Engineering with Computers, volume 17, pages 269–286, 2001.

[40] V. N. Parthasarathy, C. M. Graichen, and A. F. Hathaway. *A Comparison of Tetrahedron Quality Measures*. Finite Elements in Analysis and Design **15**(3):255–261, January 1994.

[41] V. N. Parthasarathy and Srinivas Kodiyalam. *A Constrained Optimization Approach to Finite Element Mesh Smoothing*. Finite Elements in Analysis and Design **9**:309–320, 1991.

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

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

[44] Joachim Schöberl. *NETGEN: An Advancing Front 2D/3D-Mesh Generator Based on Abstract Rules*. Computing and Visualization in Science **1**(1):41–52, July 2007.

[45] Jonathan Richard Shewchuk. *Tetrahedral Mesh Generation by Delaunay Refinement*. Proceedings of the Fourteenth Annual Symposium on Computational Geometry (Minneapolis, Minnesota), pages 86–95, June 1998.

[46] ———. *Two Discrete Optimization Algorithms for the Topological Improvement of Tetrahedral Meshes*. Unpublished manuscript at http://www.cs.cmu.edu/~jrs/jrspapers.html, 2002.

[47] ———. *What Is a Good Linear Element? Interpolation, Conditioning, and Quality Measures*. Eleventh International Meshing Roundtable (Ithaca, New York), pages 115–126, September 2002.

[48] ———. *Delaunay Mesh Generation*. Unpublished manuscript, 2008.

[49] Hang Si and Klaus Gärtner. *Meshing Piecewise Linear Complexes by Constrained Delaunay Tetrahedralizations*. Proceedings of the Fourteenth International Meshing Roundtable (San Diego, California) (Byron W. Hanks, editor), pages 147–163, September 2005.

[50] Nigel P. Weatherill. *Delaunay Triangulation in Computational Fluid Dynamics*. Computers and Mathematics with Applications **24**(5/6):129–150, September 1992.