

1A.1

Aggressive Tetrahedral Mesh Improvement

Bryan Matthew Klingner and Jonathan Richard Shewchuk

University of California at Berkeley

Summary. We present a tetrahedral mesh improvement schedule that usually 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.” Our goal is to aggressively optimize the worst tetrahedra, with speed a secondary consideration. Mesh optimization methods often get stuck in bad local optima (poor-quality meshes) because their repertoire of mesh transformations is weak. We employ a broader palette of operations than any previous mesh improvement software. Alongside the best traditional topological and smoothing operations, we introduce a topological transformation that inserts a new vertex (sometimes deleting others at the same time). We describe a schedule for applying and composing these operations that rarely gets stuck in a bad optimum. We demonstrate that all three techniques—smoothing, vertex insertion, and traditional transformations—are substantially more effective than any two alone. Our implementation usually improves meshes so that all dihedral angles are between 31° and 149° , or (with a different objective function) between 23° and 136° .

1 Introduction

Industrial applications of finite element and finite volume methods using unstructured tetrahedral meshes typically begin with a geometric model, from which a mesh is created using advancing front, Delaunay, or octree methods. Often, the next step is to use heuristic mesh improvement methods (also known as *mesh clean-up*) that take an existing mesh and try to improve the quality of its elements (tetrahedra). The “quality” of an element is usually expressed as a number that estimates its good or bad effects on interpolation error, discretization error, and stiffness matrix conditioning. The quality of a mesh is largely dictated by its worst elements. Mesh improvement software can turn a good mesh into an even better one, but existing tools are inconsistent in their ability to rescue meshes handicapped by bad elements. In this paper, we demonstrate that tetrahedral mesh improvement methods can push the worst elements in a mesh to levels of quality not attained by any previous technique, and can do so consistently.

There are two popular mesh improvement methods. *Smoothing* is the act of moving one or more mesh vertices to improve the quality of the elements

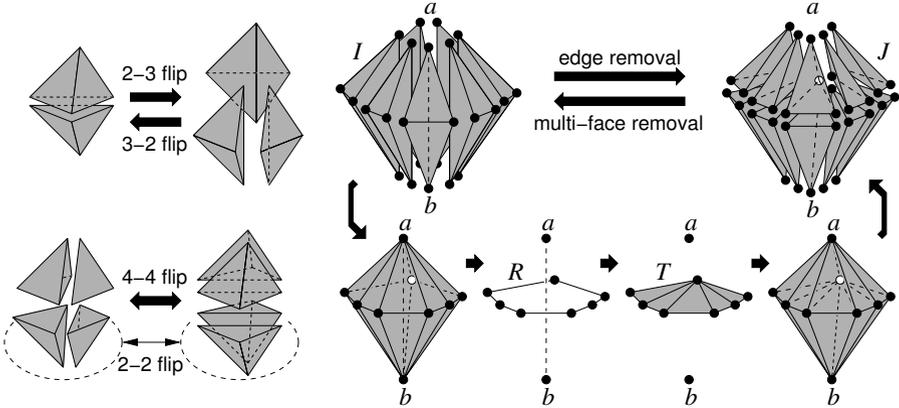


Fig. 1. Examples of topological transformations.

adjoining them. Smoothing does not change the topology (connectivity) 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.

Topological transformations are usually *local*, meaning that only a small number of elements are changed, removed, or introduced by a single operation. Figure 1 illustrates several examples, including 2-3 flips, 3-2 flips, 4-4 flips, and 2-2 flips. The numbers denote the number of tetrahedra removed and created, respectively. 2-2 flips occur only on mesh boundaries, and cause an edge flip on the surface.

Smoothing and topological transformations are usually used as operations in a *hill-climbing* method for optimizing the quality of 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 applying an operation to 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 applied; 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 applied. 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.

In our opinion, the best research to date on tetrahedral mesh improvement is the work of Freitag and Ollivier-Gooch [11], who combine optimization-based smoothing with several topological transformations, including 2-3 flips, 3-2 flips, and an operation sometimes called *edge removal*. They report the

performance on a variety of meshes of several schedules for applying these operations, show that their best schedule eliminates most poorly shaped tetrahedra, and offer empirical recommendations about what makes some schedules better than others.

Although we have long felt that the paper by Freitag and Ollivier-Gooch is a model of excellent meshing research, we also suspected that yet better results were possible through a more aggressive set of operations. Delaunay mesh generation algorithms achieve good results by inserting new vertices [12], often boosting the smallest dihedral angle to 19° or more [22]. But no “mesh clean-up” paper we know of uses transformations that add new vertices to the mesh—a strange omission. No doubt this oversight stems partly from the desire not to increase the size (number of elements) of a mesh.

We show here that vertex-creating transformations make it possible to achieve levels of mesh quality that, to the best of our knowledge, are unprecedented. Given meshes whose vertices are somewhat regularly spaced (as every competent tetrahedral mesh generator produces), our implementation usually improves them so that no dihedral angle is smaller than 31° or larger than 149° . It sometimes achieves extreme angles better than 40° or (with a different objective function) 120° . No previous software we 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 scores cannot be improved by any single transformation at hand), and it is unlikely that any mesh improvement algorithm will ever find the *global optimum*—the best possible mesh of the domain—or even come close.

However, our goal is merely to find a local optimum whose tetrahedra are all excellent. Intuitively, a powerful enough repertoire of operations ought to “smooth out” the objective function, thereby ensuring that few poor local optima exist. The question is, what is a powerful enough repertoire?

In this paper, we take the operations explored by Freitag and Ollivier-Gooch and enrich them considerably, adding the following.

- A topological transformation that inserts a new vertex (usually into a bad tetrahedron). This operation is not unlike Delaunay vertex insertion, but it is designed to optimize the worst new tetrahedron instead of enforcing the Delaunay property. Sometimes it deletes vertices as well.
- Smoothing of vertices constrained to lie on the boundary of the mesh.
- Edge removal for edges on the boundary of the mesh.
- The *multi-face removal* operation of de Cougny and Shephard [7].

- Compound operations that combine several other 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.

These additions are motivated by two observations: to repair a tetrahedralization it is often necessary to repair the boundary triangulation; and inserting a new vertex often breaks resistance that cannot be broken by topological operations that do not change the set of vertices in a mesh. We have implemented and tested schedules that use these operations, and we investigate the consequences of turning different operations on or off.

The main goal of this paper is to answer the question, “How high can we drive the quality of a tetrahedral mesh, assuming that speed is not the highest priority?” Questions like “How quickly can we consistently fix a mesh so all its dihedral angles are between, say, 20° and 160° ?” are important too. But we think it is hard to do justice to both questions in one short paper, and studying the former question first will make it easier to answer the latter question in future work.

2 Mesh Quality

The success of the finite element method depends on the shapes of the tetrahedra. Large dihedral angles (near 180°) cause large interpolation errors and rob the numerical simulation of its accuracy [14, 17, 24], and small dihedral angles render the stiffness matrices associated with the finite element method fatally ill-conditioned [2, 24]. Although anisotropic tetrahedra with extreme angles are desirable and necessary in some contexts, such as aerodynamics, we restrict our attention here to isotropic simulations, which are the norm in mechanics, heat transfer, and electromagnetics. Often, a single bad tetrahedron can spoil a simulation. For example, a large dihedral angle can engender a huge spurious strain in the discretized solution of a mechanical system. Therefore, our top priority is to produce a mesh in which the worst tetrahedra are as good as possible.

Most mesh improvement programs encapsulate the quality of a tetrahedron t as a single numerical *quality measure* $q(t)$. Many such quality measures are available [9, 24]. All the mesh operations we use are flexible enough to accommodate almost every measure in the literature. We assume each measure is “normalized” so that a larger value of $q(t)$ indicates a better tetrahedron, and $q(t)$ is positive if t has the correct topological orientation, zero if t is degenerate, and negative if t is “inverted” (meaning that there is a wrinkle in the fabric of the mesh). We assume no input mesh has inverted tetrahedra; all our operations will keep it that way.

We tried four quality measures in our implementation.

- The minimum sine of a tetrahedron’s six dihedral angles, or the *minimum sine measure* for short. This measure penalizes both small and large dihedral angles, and Freitag and Ollivier-Gooch [11] find it to be the most

effective measure they considered. It also has the advantage that dihedral angles are intuitive.

- The *biased minimum sine measure*, which is like the minimum sine measure, but if a dihedral angle is obtuse, we multiply its sine by 0.7 (before choosing the minimum). This allows us to attack large angles much more aggressively without much sacrifice in improving the small angles.
- The *volume-length measure*, suggested by Parthasarathy, Graichen, and Hathaway [19] and denoted V/ℓ_{rms}^3 , is the signed volume of a tetrahedron divided by the cube of its root-mean-squared edge length. We multiply it by $6\sqrt{2}$ so that the highest quality is one, the measure of an equilateral tetrahedron.
- The *radius ratio*, suggested by Cavendish, Field, and Frey [5], is the radius of a tetrahedron’s inscribed sphere divided by the radius of its circumscribing sphere. We multiply it by 3 so that the highest quality is one, the measure of an equilateral tetrahedron. We experimented with this measure because of its popularity, but we found that it is inferior to the volume-length measure in mesh optimization, even when the goal is to optimize the radius ratio. So we will revisit it only once—in Section 5 where we demonstrate this fact.

The first two measures do not penalize some tetrahedra that are considered bad by the last two measures. For example, an extremely skinny, needle-shaped tetrahedron can have excellent dihedral angles, whereas its skinniness is recognized by the volume-length measure and the radius ratio. There is evidence that a skinny tetrahedron with good dihedral angles is harmless, hurting neither discretization error nor conditioning [24]; its worst crime is to waste vertices, because its accuracy is inversely proportional to the length of its longest edge, not its shortest. Moreover, such a tetrahedron is indispensable at the tip of a needle-shaped domain. Readers not convinced by this argument will find the volume-length measure invaluable.

We need to extend quality measures from individual tetrahedra to whole meshes. The worst tetrahedra in a mesh have far more influence than the average tetrahedra, so the objective function we optimize is the *quality vector*: a vector listing the quality of each tetrahedron, ordered from worst to best. Two meshes’ quality vectors are compared *lexicographically* (akin to alphabetical order) so that, for instance, 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 small subset of tetrahedra in a mesh with new ones, we 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. Our software never needs to compute the quality vector of an entire mesh.

3 The Fundamental Tools: Mesh Operations

Here we describe the mesh transformation operations that form the core of our mesh improvement program. Simultaneously, we survey the previous work in mesh improvement.

3.1 Smoothing

The most famous smoothing technique is *Laplacian smoothing*, in which a vertex is moved to the centroid of the vertices to which it is connected [13]. 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 [20, 4]. 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 [10] propose a more sophisticated *non-smooth* optimization algorithm, which makes it possible to optimize the worst tetrahedron in a group—for instance, to maximize the minimum 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 [11] had great success with this algorithm, and we use it essentially unchanged (though we have our own implementation).

Whereas Freitag and Ollivier-Gooch only smooth vertices in the interior of a mesh, we also implemented constrained smoothing of boundary vertices. If the boundary triangles adjoining a vertex appear (within some tolerance) to lie on a common plane, our smoother assumes that the vertex can be smoothed within that plane. Similarly, we identify vertices that can be moved along an edge of the domain without changing its shape. However, we did not implement constrained smoothing for curved domain boundaries, so some of our meshes do not benefit from boundary smoothing.

We always use what Freitag and Ollivier-Gooch call *smart smoothing*: if a smoothing operation does not improve the minimum quality among the tetrahedra changed by the operation, then the operation is not done. Thus, the quality vector of the mesh never gets worse.

3.2 Edge Removal

Edge removal, proposed by Brière de l’Isle and George [3], is a topological transformation that removes a single edge from the mesh, along with all the tetrahedra that include it. (The name 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 we prefer the earlier name.) It includes the 3-2 and 4-4 flips, but also includes other transformations that remove edges shared by any number of tetrahedra. In general, edge removal replaces m tetrahedra with $2m - 4$; Figure 1 (right) illustrates replacing seven tetrahedra with ten. De Cougny and Shephard [7] and Freitag and Ollivier-Gooch [11] 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 . 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 $J = \bigcup_{t \in T} \{\text{conv}(\{a\} \cup t), \text{conv}(\{b\} \cup t)\}$, as illustrated, 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 . We solve this problem with a dynamic programming algorithm of Klincsek [16], which was invented long before anyone studied edge removal. (Klincsek’s algorithm solves a general class of problems in optimal triangulation. Neither Brière de l’Isle and George nor Freitag and Ollivier-Gooch appear to have been aware of it.) The algorithm runs in $O(m^3)$ time, but m is never large enough for its speed to be an impairment.

3.3 Multi-Face Removal

Multi-face removal is the inverse of edge removal, and includes the 2-3 and 4-4 flips. An m -face removal replaces $2m$ tetrahedra with $m + 2$. It has been neglected in the literature; so far as we know, it has appeared only in an unpublished manuscript of de Cougny and Shephard [7], who present evidence that multi-face removal is effective for mesh improvement.

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 1, 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 .)

Our software 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

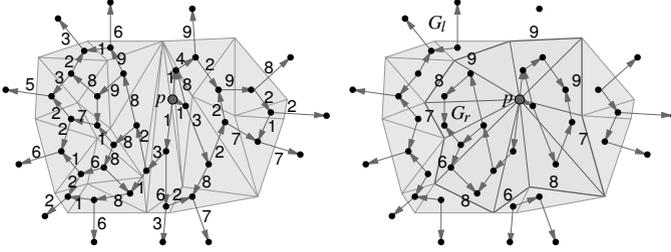


Fig. 2. Vertex insertion as graph cut optimization. In this example, the smallest cut has weight 6. The weights of the cut edges are the qualities of the new elements.

two tetrahedra adjoining f . The optimal multi-face removal operation does not necessarily remove *all* the faces sandwiched between a and b . We use the algorithm of Shewchuk [23] 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.

3.4 Vertex Insertion

Our main innovation in this paper is to show that mesh improvement is far more effective with the inclusion of transformations that introduce new vertices. We use an algorithm similar to Delaunay vertex insertion: we choose a location p to insert a new vertex and a set I of tetrahedra to delete, such that p lies in, and can “see” all of, the star-shaped polyhedral cavity $C = \bigcup_{t \in I} t$. We fill the cavity with a set of new tetrahedra $J = \{\text{conv}(\{p\} \cup f) : f \text{ is a face of } C\}$. Choosing the position of p is a black art; see Section 4.2 for how we choose it. To choose I , we solve this combinatorial problem: given a point p , which tetrahedra should we delete to maximize the quality of the worst new tetrahedron?

Our algorithm views the mesh as a graph M with one node for each tetrahedron, as depicted in Figure 2. For simplicity, we identify nodes of the graph with the tetrahedra they represent. M 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 set I (i.e., 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, we direct the edge arbitrarily.) Although M can have cycles, they are rare, so we adopt some nomenclature from trees: if $(v, w) \in M$ then w is a *child* of v and v is a *parent* of w . Any tetrahedron that contains p is a *root* of M . Usually there is just one root tetrahedron, but sometimes we insert a new vertex on a boundary edge of the domain, in which case all the tetrahedra sharing that edge are roots. If a vertex is inserted at p , all the roots must be deleted.

Our algorithm for finding an optimal cavity computes a cut in M that induces a cavity in the mesh. It begins by constructing the subgraph G of M whose nodes are the roots of M and all the tetrahedra that are reachable in M from the roots by a directed path of length six or less. We select G this way because we do not want to search the entire graph M for a cut, and we find that in practice, tetrahedra further away from the root rarely participate in the optimal cavity. We find that G typically has 5–100 tetrahedra. For each triangular face that belongs to only one tetrahedron in G , we add a “ghost node” to G to act as a neighboring tetrahedron. Then, every leaf of G is a ghost node, as Figure 2 shows.

The tetrahedra in G , except the leaves, are candidates for deletion. For each edge $(v, w) \in G$, let f be the triangular face shared by the tetrahedra v and w . Our 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, as illustrated in Figure 2. 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 . Our goal is to find the partition that satisfies this constraint and maximizes the smallest edge cut (because that edge determines the worst new tetrahedron).

The algorithm in Figure 3 computes this optimal cut. (We omit the proof.) 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 the ninth line of pseudocode. In practice, I typically comprises 5–15 tetrahedra. After an initial $O(|G| \log |G|)$ -time sorting step, the rest of the algorithm runs in $O(|G|)$ time.

Sometimes, a vertex insertion operation deletes one or more of the other vertices, as in Figure 2. When a tetrahedron with three parents is deleted, the vertex it shares with all three parents is deleted too. Thus, our vertex insertion operation sometimes reduces the number of vertices in the mesh.

3.5 Composite Operations

Mesh improvement methods often get stuck in local optima that are far from the global optimum. Joe [15] suggests that this problem can be ameliorated by composing multiple basic operations to form new operations. These composite operations sometimes get a hill-climbing optimizer across what was formerly a valley in the objective function, thereby leading the way to a better local optimum.

We have found that vertex insertion, as described in Section 3.4, rarely improves the quality vector of the mesh immediately, but it is frequently effective

<pre> Sort edges of G from smallest to largest quality. $H \leftarrow$ a graph with the same vertices as G but no edges (yet). (H need not be stored as a separate graph; let each edge of G have a bit that indicates whether it is in H too.) All vertices of G are initially unlabeled. Label every root of G "cavity." Label every leaf of G "anti-cavity." for each directed edge (v, w) of G (in sorted order) if v is labeled "cavity" if w is labeled "anti-cavity" Record (v, w), which determines a new tetrahedron in J. else if w is unlabeled CAVITY(w) else if v is unlabeled if w is labeled "anti-cavity" ANTI-CAVITY(v) else { w is unlabeled } Add (v, w) to H. </pre>	<pre> CAVITY(w) Label w "cavity." for each unlabeled parent p of w in G CAVITY(p) for each unlabeled child c of w in H CAVITY(c) ANTI-CAVITY(v) Label v "anti-cavity." for each unlabeled child c of v in G ANTI-CAVITY(c) for each unlabeled parent p of v in H ANTI-CAVITY(p) </pre>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Fig. 3. Algorithm for computing the cavity that optimizes the new tetrahedra when a new vertex is inserted. Upon completion, the tetrahedra to be deleted are labeled "cavity."

if traditional smoothing and transformations follow. To create an operation that composes vertex insertion with subsequent operations, we implemented a rollback mechanism that allows us to attempt a sequence of transformations, then reverse all the changes if the final mesh is not better than the initial one.

The ATTEMPTINSERT pseudocode in Figure 4 shows how we follow vertex insertion with smoothing and topological transformations, then decide whether to roll back the insertion. Immediately after inserting the new vertex (as described in Section 3.4), we smooth it (by optimization), then we run passes of topological transformations and smoothing on the tetrahedra adjoining it in an attempt to improve them. (See the pseudocode for details.) Finally, we compare the quality vector of all the new and changed (smoothed) tetrahedra with the quality vector of the deleted tetrahedra (the set I). If the mesh has not improved, we roll it back to the instant before we inserted the new vertex.

Even though the algorithm in Figure 3 is technically optimal, we have learned by experiment that *composite* vertex insertion is more effective if we bias the algorithm to prefer larger cavities than it would normally choose. To encode this bias, our implementation multiplies edge weights by 1.0, 1.4, 1.8, or 2.1 if they are a distance of zero, one, two, or greater than two from the nearest root, respectively. These weights sometimes cause worse-than-optimal tetrahedra to be created, but these are often improved by subsequent operations. In the end, the vertex insertion operation is only accepted (not rolled back) if the *unbiased* quality vector improves.

4 Scheduling the Operations

4.1 Examples from Previous Work

Joe’s algorithm [15] checks each face of the mesh to see if any of his transformations (including composite 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 our experiments, we eliminated all tetrahedra with radius ratios below 0.51 by optimizing the objective V/ℓ_{rms}^3 .)

Freitag and Ollivier-Gooch’s schedule [11] 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.

Edelsbrunner and Guoy [8] demonstrate that a theoretically motivated technique called *sliver exudation* [6], 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 tetrahedra under 10° or over 160° , but it does not eliminate all mediocre tetrahedra, especially on the boundary. (See the STGALLEN and STAYPUFT input meshes in Section 5.) Note that variational meshing is a standalone mesh generation algorithm, and cannot be used as a mesh improvement algorithm, because the mesh it generates does not conform to a specified triangulated boundary.

4.2 Our Mesh Improvement Schedule

Pseudocode for our mesh improvement implementation appears in Figure 4. Like all such schedules, ours is heuristic and evolved through trial and error.

We find that prescribing a fixed number of improvement passes, as Freitag and Ollivier-Gooch do, is too inflexible, and we get better results by adapting our schedule to the mesh at hand. We begin with a pass of optimization-based

```

MESHAGGRESSION( $M$ )  {  $M$  is a mesh }
  Smooth each vertex of  $M$ .
  TOPOLOGICALPASS( $M$ )
   $failed \leftarrow 0$ 
  while  $failed < 3$ 
     $Q \leftarrow$  list of quality indicators for  $M$ .
    Smooth each vertex of  $M$ .
    if  $M$  is sufficiently better than  $Q$ 
       $failed \leftarrow 0$ 
    else
      TOPOLOGICALPASS( $M$ )
      if  $M$  is sufficiently better than  $Q$ 
         $failed \leftarrow 0$ 
      else
        if  $failed = 1$  { desperation pass }
           $L \leftarrow$  list of tets with a dihedral
             $< 40^\circ$  or  $> 140^\circ$ .
        else  $L \leftarrow$  list of the worst 3.5% of
          tets in  $M$ .
        INSERTIONPASS( $M, L$ )
        if  $M$  is sufficiently better than  $Q$ 
           $failed \leftarrow 0$ 
        else  $failed \leftarrow failed + 1$ 


---


INSERTIONPASS( $M, L$ )  {  $L$  is a list of tets }
  for each tetrahedron  $t \in L$  that still exists
    for each face  $f$  of  $t$  on the mesh
      boundary (if any)
       $p \leftarrow$  point at barycenter of  $f$ 
      if ATTEMPTINSERT( $M, p$ )
        Restart outer loop on next tet.
   $p \leftarrow$  point at barycenter of  $t$ 
  if ATTEMPTINSERT( $M, p$ )
    Restart outer loop on next tet.
  for each edge  $e$  of  $t$  on the mesh
    boundary (if any)
     $p \leftarrow$  point at midpoint of  $e$ 
    if ATTEMPTINSERT( $M, p$ )
      Restart outer loop on next tet.


---


TOPOLOGICALPASS( $L$ )  {  $L$  is a list of tets }
  for each tetrahedron  $t \in L$  that still exists
    for each edge  $e$  of  $t$  (if  $t$  still exists)
      Attempt to remove edge  $e$ .
    for each face  $f$  of  $t$  (if  $t$  still exists)
      Attempt to remove face  $f$  (multi-face
        or 2-3 or 2-2 flip).
  return the surviving tetrahedra of  $L$  and
  all the new tetrahedra
  created by this call.


---


ATTEMPTINSERT( $M, p$ )  { New vertex at  $p$  }
   $I \leftarrow$  deleted tetrahedra, computed as
    discussed in Section 3.4
   $q \leftarrow$  quality of the worst tetrahedron in  $I$ 
  Replace  $I$  with the new tetrahedra  $J$  (see
    Section 3.4 and Figure 3).
   $attempts \leftarrow 8$ 
  repeat
    Smooth  $p$ .
    { In next line, the cavity may expand }
     $J \leftarrow$  TOPOLOGICALPASS( $J$ )
     $attempts \leftarrow attempts - 1$ 
  while  $attempts > 0$  and some topological
    change occurred
   $K \leftarrow J \cup$  the tetrahedra in  $M$  that share
    a vertex with  $J$ 
  repeat
     $q' \leftarrow$  quality of the worst tet in  $J$ 
    Smooth each vertex of  $J$ .
     $q'' \leftarrow$  quality of the worst tet in  $K$ 
     $attempts \leftarrow attempts - 1$ 
  while  $attempts > 0$  and  $q'' > q'$ 
  if  $q'' > q$ 
    return true.
  Roll back all changes since the beginning
    of this procedure call.
  return false.

```

Fig. 4. Our mesh improvement schedule.

smoothing (smoothing each vertex once) and a pass of topological transformations (leaving out vertex insertion), because these are always fruitful. Our topological pass visits each tetrahedron once, and searches for a transformation that will eliminate it and improve the quality vector of the changed tetrahedra (and therefore of the entire mesh). If no edge removal operation succeeds in removing a particular tetrahedron, we try multi-face removal on each face of that tetrahedron that lies in the mesh interior. However, in Section 5 we test the effect of disabling multi-face removal but still allowing faces to be removed by 2-3 flips (which are easier to implement). If an interior face has an edge on the mesh boundary, we also test the possibility that a 2-2 flip on that edge might improve the mesh.

Our implementation then performs passes over the mesh until three subsequent passes fail to make sufficient progress. We gauge progress using a small list of *quality indicators*: the quality of the worst tetrahedron in the entire mesh, and seven *thresholded means*. A mean with threshold d is computed by calculating the quality of each tetrahedron in the mesh, reducing to d any quality greater than d , then taking the average. The purpose of

a thresholded mean is to measure progress in the lowest-quality tetrahedra while ignoring changes in the high-quality tetrahedra. For the minimum sine measure, we compute thresholded means with thresholds $\sin 1^\circ$, $\sin 5^\circ$, $\sin 10^\circ$, $\sin 15^\circ$, $\sin 25^\circ$, $\sin 35^\circ$, and $\sin 45^\circ$. (Each tetrahedron is scored according to its worst dihedral angle; we do *not* compute thresholded means of *all* dihedral angles.) A mesh is deemed to be sufficiently improved (to justify more passes) if at least one of the thresholded means increases by at least 0.0001, or if the quality of the worst tetrahedron increases at all.

Each pass begins with smoothing. If smoothing does not make adequate progress, a topological pass follows. If progress is still insufficient, we resort to vertex insertion, which is less desirable than the other operations both because it often increases the size of the mesh and because our compound vertex insertion operation is slow. Vertex insertion is usually aimed at the worst 3.5% of tetrahedra in the mesh, but our implementation never gives up without trying at least one “desperation pass” that attempts to insert a vertex in every tetrahedron that has an angle less than 40° or greater than 140° .

Most mesh generation algorithms create their worst tetrahedra on the boundary of the mesh, and boundary tetrahedra are the hardest to repair. Thus, when our vertex insertion pass targets a tetrahedron on the boundary of the mesh, it always tries to insert a vertex at the barycenter of the boundary face(s) first. For tetrahedra where that fails, and tetrahedra not on the boundary, we try the tetrahedron barycenter next. We also try the midpoints of tetrahedron edges that lie on the boundary, but we try them last, because (for reasons we don’t understand) we obtain better meshes that way.

5 Results and Discussion

We tested our schedule on a dozen meshes.

- CUBE1K and CUBE10K are high-quality meshes of a cube generated by NETGEN [21].
- TFIRE is a high-quality mesh of a tangentially-fired boiler, created by Carl Ollivier-Gooch’s GRUMMP software.
- TIRE, RAND1 and RAND2 come courtesy of Freitag and Ollivier-Gooch, 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 with curved boundaries, generated by isosurface stuffing [18]. The curvature prevents us from smoothing the original boundary vertices.

- STGALLEN and STAYPUFT are medium- to low-quality meshes with curved boundaries, generated by two different implementations of variational tetrahedral meshing [1], courtesy of Pierre Alliez and Adam Bargteil, respectively.
- HOUSE and P are Delaunay meshes generated by Pyramid [22] configured so the vertices are nicely spaced, but no effort is made to eliminate sliver tetrahedra.

Tables 1 and 2 show these meshes before and after improvement with the MESHAGGRESSION schedule in Figure 4. We 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_{TMS}^3 (lower left) as objectives. (We omit meshes optimized for the radius ratio objective, which was not competitive with the volume-length measure, even as measured by the radius ratios of the tetrahedra.)

Our main observation is that the dihedral angles improved to between 31° and 149° for the minimum sine objective, between 25° and 142° for the biased minimum sine objective, and between 23° and 136° for the volume-length measure. Even the pathological meshes RAND1 and RAND2 end with excellent quality. These numbers put our implementation far ahead of any other tetrahedral mesh algorithm we have seen reported. Freitag and Ollivier-Gooch reported angle bounds as good as 13.67° and 156.14° for TIRE, versus our 28.13° and 125.45° ; as good as 15.01° and 159.96° for RAND1, versus our 36.95° and 119.89° ; and as good as 10.58° and 164.09° for RAND2, versus our 34.05° and 126.61° .

Of course, to obtain such high quality takes time. Meshes that begin with high quality take a modest amount of time to improve. RAND1 and RAND2 take disproportionately longer—both because our implementation tries to hold back vertex insertions until they prove to be necessary, and because the composite vertex insertion operation is slow, often accounting for about 90% of the running time. Of that 90%, about one third is spent in the basic vertex insertion operation, one third in smoothing the cavity, and one third in topological transformations in the cavity. It seems impossible to predict which quality measure will run faster on a given mesh, and the differences in running times are erratic.

No mesh increased in size by more than 41%, and some meshes shrank (because the vertex insertion operation can also delete vertices).

Table 3 shows the effects of turning features on or off. The top half of the page explores the question of which features are most important to have if the programmer’s time is limited. We 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. We were surprised to see that it alone can substantially improve some meshes, even though most vertex insertion operations fail when neither smoothing nor

Table 1. Twelve meshes before and after improvement (continued in Table 2). In each box, the upper left mesh is the input, the upper right mesh is optimized for the minimum sine measure, the lower right mesh is optimized for the biased minimum sine measure, and the lower left mesh is optimized for V/ℓ_{rms}^3 . Running times are given for a Mac Pro with a 2.66 GHz Intel Xeon processor. Red tetrahedra have dihedral angles under 5° or over 175° , orange have angles under 15° or over 165° , yellow have angles under 30° or over 150° , green have angles under 40° or over 140° , and better tetrahedra do not appear. Histograms show the distributions of dihedral angles, and the minimum and maximum 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.

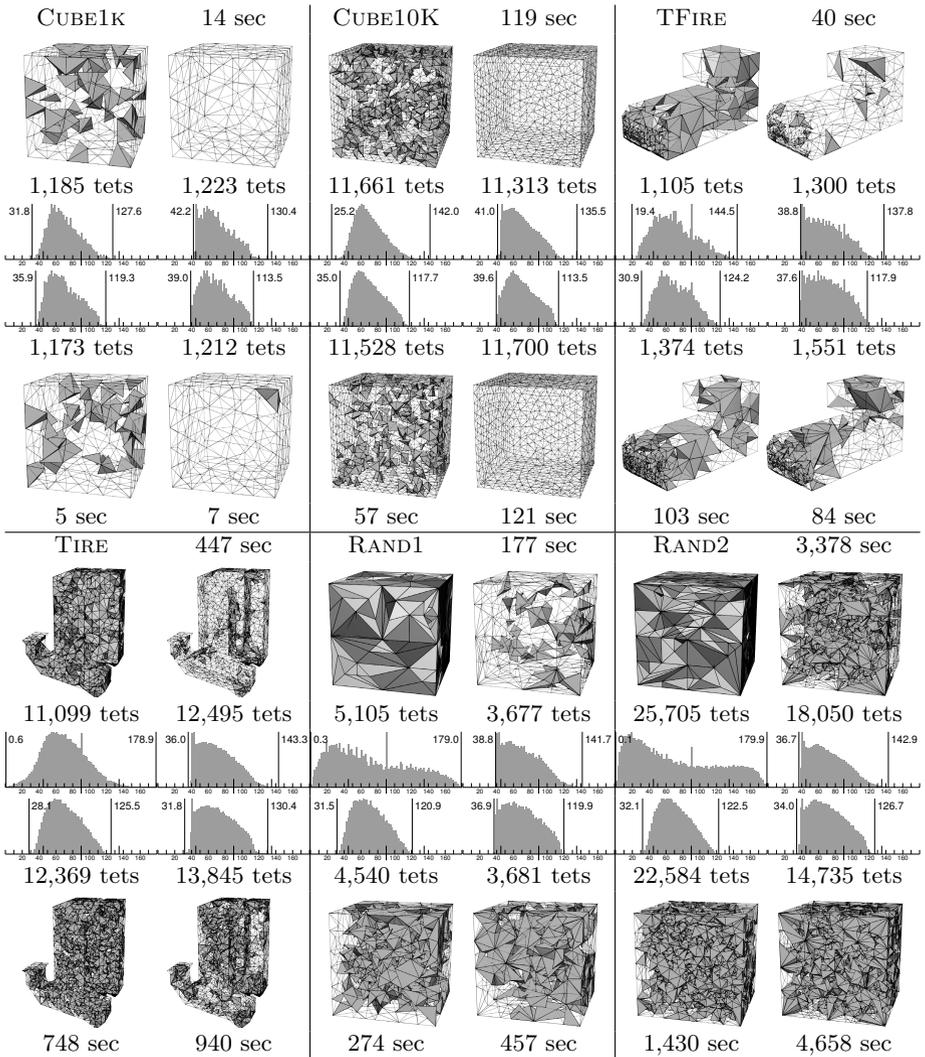
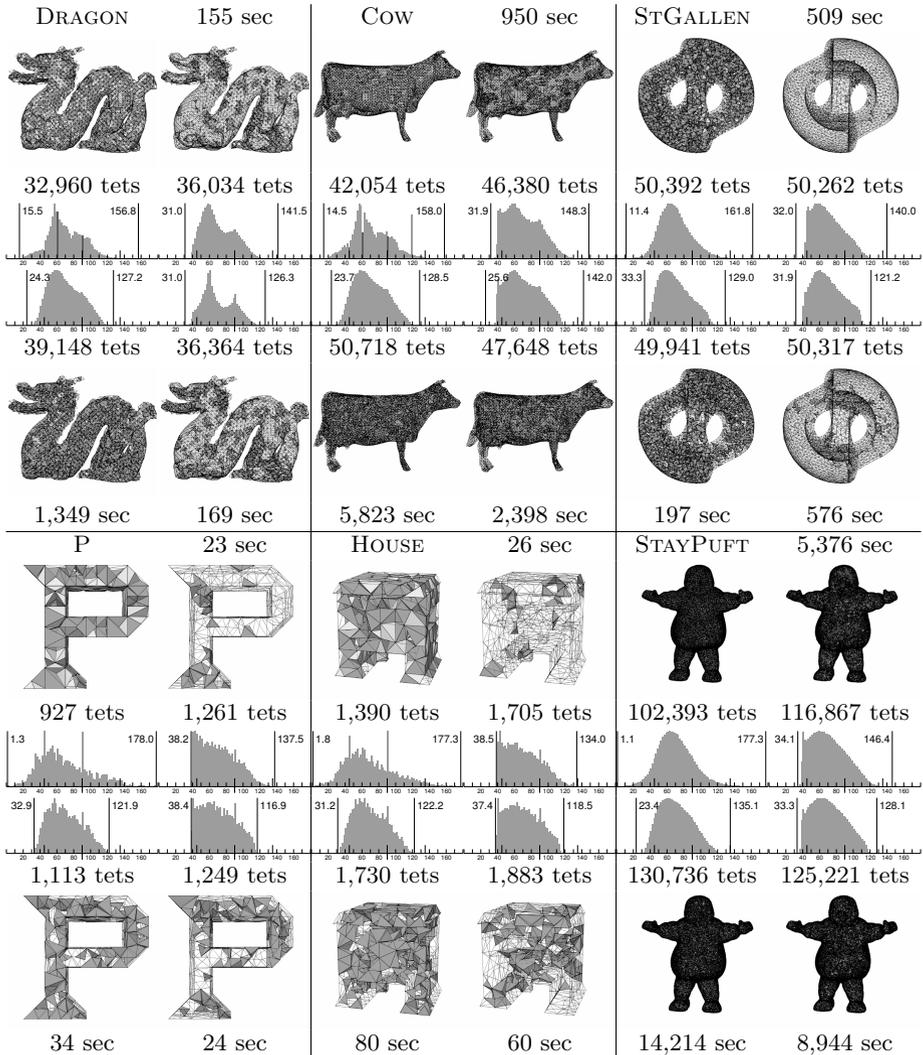


Table 2. Continuation of Table 1. Red histogram bars should have their heights multiplied by 20 to account for the fact that in the semi-regular meshes DRAGON and COW, angles of 45° , 60° , and 90° occur with high frequency.



other topological transformations are available to create a compound operation (as described in Section 3.5). Edge removal ranks last. Any combination of two of these operations gives a substantial advantage over one, and having all three gives another substantial advantage.

Implementing *all* the features discussed in this paper (“maximum aggression”) gives another substantial advantage, but these additional features

Table 3. Histograms showing the dihedral angle distribution, and minimum and maximum dihedral angles, for several meshes optimized with selected features turned on (upper half of page) or off (lower half). The objective was to maximize the biased minimum sine measure. Multiply the heights of the red histogram bars by 20. “Maximum aggression” has all features turned on.

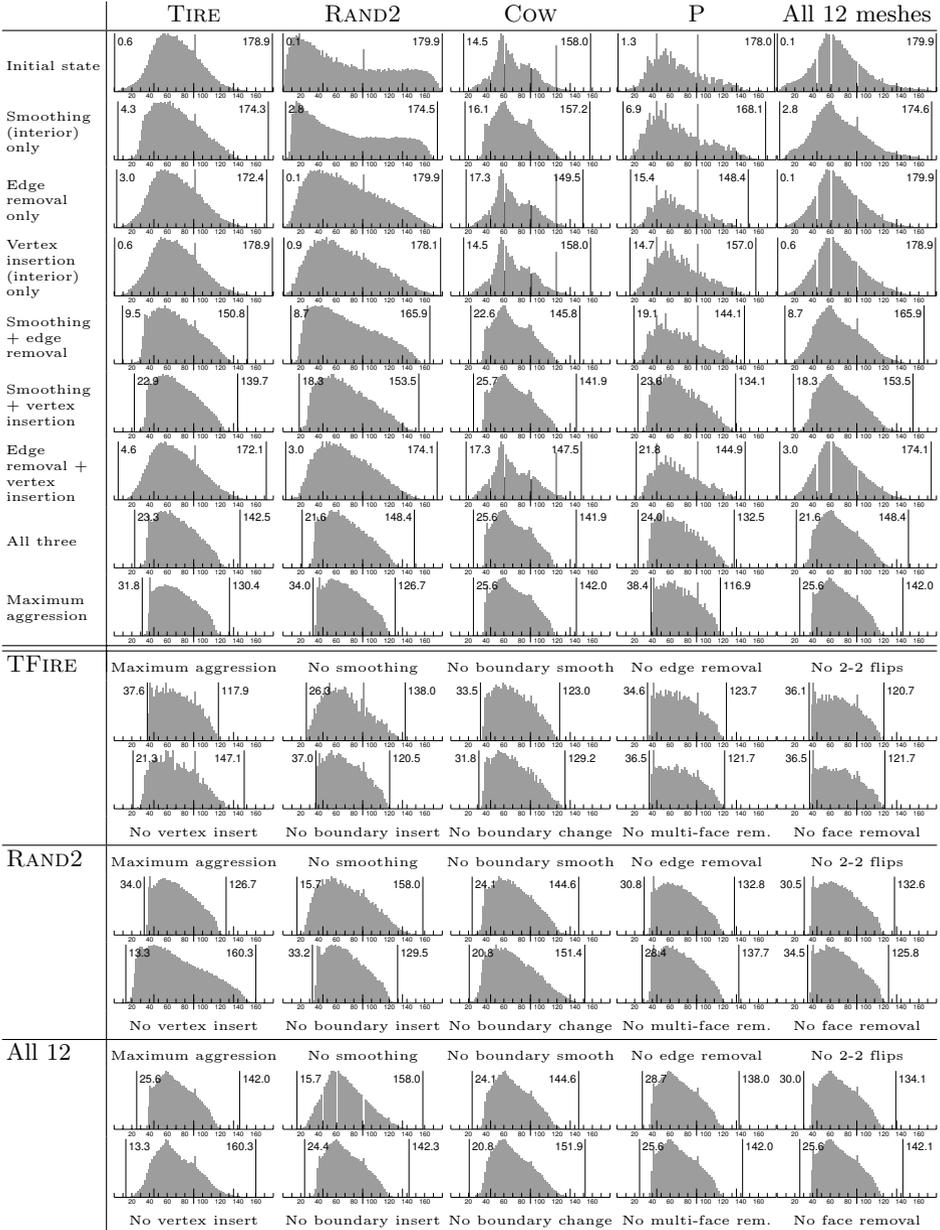
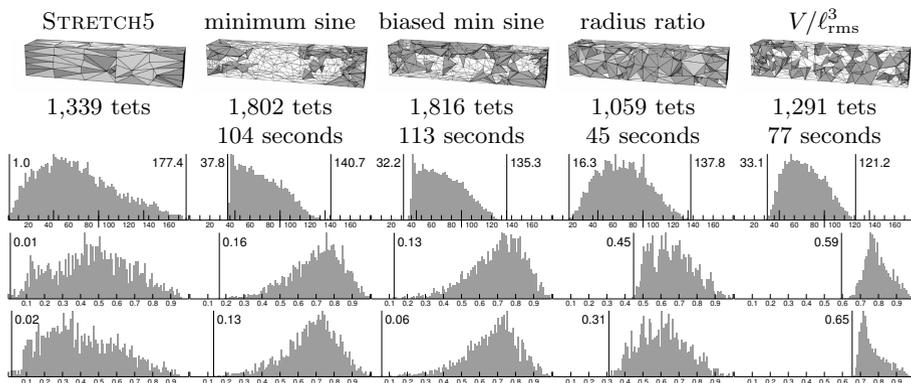


Table 4. A stretched input mesh and four output meshes optimized with different quality measures as objective functions. The histograms tabulate, from top to bottom, dihedral angles, radius ratios (times 3), and $6\sqrt{2}V/\ell_{\text{rms}}^3$.



(multi-face removal, boundary smoothing, boundary insertion) are individually responsible for only small increments. The bottom half of Table 3 shows the effects of turning just a single feature off. Some of the switches listed there are inclusive of other switches. “No smoothing” turns off all smoothing—in the interior and on the boundary. Likewise, “no vertex insert” turns off all insertion. “No face removal” turns off multi-face removal and 2-3 flips, whereas “no multi-face removal” turns off only the former.

Smoothing and vertex insertion are clearly the most disastrous operations to lose. The effort to extend smoothing and vertex insertion so that they can operate on the boundary of the mesh was also well rewarded. Besides vertex insertion, no single topological operation is crucial if the others are present.

The papers by Freitag and Ollivier-Gooch and by de Cougny and Shephard both concluded that edge removal is rarely successful for an edge that adjoins more than seven tetrahedra; but our experience contradicts this. We see many successful removals of edges adjoining eight tetrahedra, and even the occasional removal of eleven tetrahedra. (Klincsek’s algorithm makes this easy to implement.) However, we observe that edge removal is most likely to be successful for edges that adjoin four tetrahedra, and multi-face removals that remove two faces predominate, so the most common beneficial topological change is a 4-4 flip.

Table 4 illustrates the effect of optimizing our four quality measures on a mesh called STRETCH5, which is CUBE1K scaled along one axis by a factor of five. This mesh highlights the weakness of the minimum sine measure and its biased counterpart as objective functions—namely, they sometimes permit skinny tetrahedra to survive. The other two quality measures are better for improving the distribution of vertices and producing “rounder” tetrahedra. The minimum sine objective is best for optimizing the smallest dihedral angle, but the volume-length measure is the best all-around objective of the four. It

even optimizes the radius ratio better than the radius ratio does! (We suspect that the radius ratio behaves poorly as an objective function because of the instability of the circumscribing radius as a function of vertex position.) In our twelve-mesh test suite, the volume-length objective always improved the worst radius ratio to at least 0.51, whereas the radius ratio objective left behind many worse tetrahedra, the worst having a radius ratio of 0.30.

6 Conclusions

We see two important avenues for future work. First, our mesh improvement implementation assumes that the spacing of the vertices in the input mesh is already correct. A better code would take as input a *spacing function* that dictates how large the tetrahedra should be in different regions of the mesh, and insert or delete vertices accordingly. Second, algorithms and schedules that achieve results similar to ours in much less time would be welcome. Our composite vertex insertion operation accounts for most of the running time, so a more sophisticated vertex insertion algorithm might improve the speed dramatically.

Because we 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 pathological inputs, we suggest 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.

Acknowledgments. We thank Pierre Alliez, Adam Bargteil, Moshe Mahler, and Carl Ollivier-Gooch for meshes and geometric models. Pixie rendered our meshes. This work was supported by the National Science Foundation under Awards CCF-0430065 and CCF-0635381, and by an Alfred P. Sloan Research Fellowship.

References

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. 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.
3. 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.

4. Scott A. Canann, Michael Stephenson, and Ted Blacker. *Optsmoothing: An Optimization-Driven Approach to Mesh Smoothing*. Finite Elements in Analysis and Design **13**:185–190, 1993.
5. 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.
6. 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.
7. Hugues L. de Cougny and Mark S. Shephard. *Refinement, Derefinement, and Optimization of Tetrahedral Geometric Triangulations in Three Dimensions*. Unpublished manuscript, 1995.
8. Herbert Edelsbrunner and Damrong Guoy. *An Experimental Study of Sliver Exudation*. Tenth International Meshing Roundtable (Newport Beach, California), pages 307–316, October 2001.
9. David A. Field. *Qualitative Measures for Initial Meshes*. International Journal for Numerical Methods in Engineering **47**:887–906, 2000.
10. 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.
11. 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.
12. 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.
13. 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.
14. P. Jamet. *Estimations d’Erreur pour des Éléments Finis Droits Presque Dégénérés*. RAIRO Analyse Numérique **10**:43–61, 1976.
15. Barry Joe. *Construction of Three-Dimensional Improved-Quality Triangulations Using Local Transformations*. SIAM Journal on Scientific Computing **16**(6):1292–1307, November 1995.
16. G. T. Klincsek. *Minimal Triangulations of Polygonal Domains*. Annals of Discrete Mathematics **9**:121–123, 1980.
17. Michal Křížek. *On the Maximum Angle Condition for Linear Tetrahedral Elements*. SIAM Journal on Numerical Analysis **29**(2):513–520, April 1992.
18. François Labelle and Jonathan Richard Shewchuk. *Isosurface Stuffing: Fast Tetrahedral Meshes with Good Dihedral Angles*. ACM Transactions on Graphics **26**(3), August 2007. Special issue on Proceedings of SIGGRAPH 2007.
19. 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.
20. 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.
21. 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.

22. 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.
23. _____. *Two Discrete Optimization Algorithms for the Topological Improvement of Tetrahedral Meshes*. Unpublished manuscript at <http://www.cs.cmu.edu/~jrs/jrspapers.html>, 2002.
24. _____. *What Is a Good Linear Element? Interpolation, Conditioning, and Quality Measures*. Eleventh International Meshing Roundtable (Ithaca, New York), pages 115–126, September 2002.

