Two Discrete Optimization Algorithms for the Topological Improvement of Tetrahedral Meshes

Jonathan Richard Shewchuk

University of California at Berkeley, Berkeley, CA, U.S.A. jrs@cs.berkeley.edu

ABSTRACT

Edge removal and *multi-face removal* are two types of local topological transformation that are useful components of a tetrahedral mesh improvement method. This paper offers an algorithm to identify the optimal edge removal transformation that removes a specified edge, and an algorithm to identify the optimal multi-face removal transformation that removes a specified face.

Keywords: finite element mesh improvement, tetrahedral mesh improvement, topological transformation, bistellar flip

1 Introduction

One of the main methods for improving the quality of a finite element mesh is the use of *topological transformations*, operations that remove elements from a finite element mesh and replace them with a different set of elements occupying the same space. These transformations are usually *local*, meaning that only a small number of elements (typically fewer than twenty) are removed or introduced by a single transformation. Figure 1 illustrates several examples, called 2-3 flips, 3-2 flips and 4-4 flips. (The numbers denote the number of tetrahedra removed and created, respectively.)

Topological transformations are typically used as individual operations in a *hill-climbing* method for optimizing the quality of a mesh. A hill-climbing method defines an *objective function* that maps each possible mesh to a numerical value (or sequence of values) that describes the “quality” of the mesh. Individual elements have objective functions too, usually called *quality measures*. A typical objective function for meshes is the average quality of the elements, or the quality of the worst element. A slightly more sophisticated objective function is the sequence of the qualities of every element, ordered from worst to best. With this objective function, two meshes are compared *lexicographically* (akin to alphabetical order) so that, for instance, an improvement in the second-

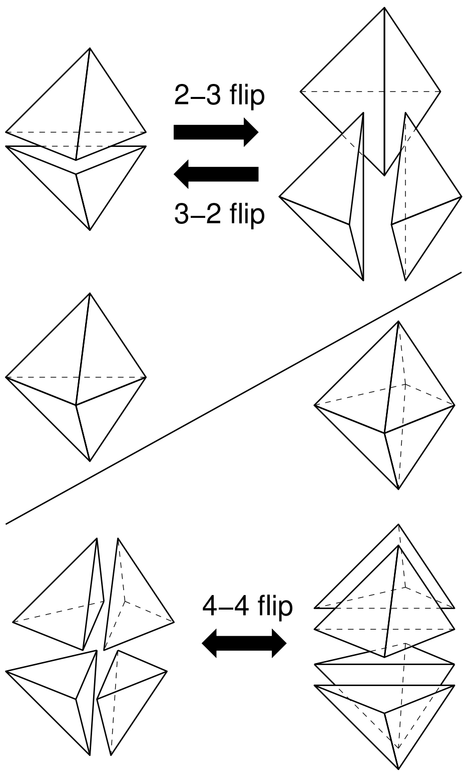


Figure 1: Examples of topological transformations.

Supported in part by the National Science Foundation under Awards ACI-9875170, CMS-9980063, and EIA-9802069, and in part by a gift from the Okawa Foundation. The views and conclusions in this document are those of the author. They are not endorsed by, and do not necessarily reflect the position or policies of, the Okawa Foundation or the U. S. Government.

worst element improves the overall objective function even if the worst element is not changed.

A hill-climbing method considers applying a topological transformation (or another operation, like node smoothing) to a specific site in the mesh. If the quality of the changed mesh will be greater than that of the original mesh, the transformation is applied; then the hill-climbing method searches for another operation that will improve the new mesh. Transformations that do not improve the value of the objective function are not applied. The method 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.

Two of the most effective local topological transformations for tetrahedral meshes are called edge removal and multi-face removal. *Edge removal*, proposed by Briere de l'Isle and George [1], is a transformation that removes a single edge from the mesh, along with all the tetrahedra that contain it. (Note that an edge removal operation may create new edges while removing the old one.) It includes the 3-2 and 4-4 flips, but also includes other transformations that remove edges shared by any number of tetrahedra. De Cougny and Shephard [2] and Frietag and Ollivier-Gooch [4] have shown dramatic evidence for its effectiveness, especially in combination with other mesh improvement operations.

Multi-face removal is the inverse of edge removal, and includes the 2-3 and 4-4 flips. (The 4-4 flip has the unique distinction of being both an edge removal operation and a multi-face removal operation.) It has been neglected in the literature; so far as I know, it has appeared only in an unpublished manuscript of de Cougny and Shephard [2]. This is unfortunate, because those authors present evidence that multi-face removal is quite effective for mesh improvement, and that transformations that remove two to five faces will improve a mesh more frequently than simple 2-3 flips.

The purpose of this paper is to offer an optimization algorithm that, given an edge, finds the optimal edge removal operation (if any) to remove that edge; and an optimization algorithm that, given a face, finds the optimal multi-face removal operation (if any) to remove that face. By the "optimal" operation, I mean the transformation that improves the quality of the worst transformed element the most. A separate problem, not addressed by this paper, is which edge or face should be targeted in the first place.

The algorithm that finds an optimal edge removal operation (for a specified edge) is a dynamic programming algorithm of Klincsek [6], which was invented long before anyone studied edge removal. (Klincsek's algorithm solves a general class of problems in optimal triangulation.) Section 2 is my effort to popularize Klincsek's algorithm, work out the details (including pseudocode) of its application to edge removal, and show that edge removal is easier to implement than Freitag and Ollivier-Gooch suggest.

An optimal multi-face removal operation (for a specified face) can be found by a new algorithm described in Section 3. Again, the implementation is relatively simple and pseudocode is provided.

The algorithms in this paper assume that each tetrahedron has a fixed topological orientation, and that the data structure that describes the mesh expresses this orientation by the ordering of the tetrahedron's vertices. For example, each tetrahedron might be expressed by listing its vertices in order so that, from the first vertex's point of view, the remaining three vertices appear to be listed in counterclockwise order. If the ordering of a tetrahedron's vertices reverses this convention, then the tetrahedron is said to be *inverted*. If the four vertices of a tetrahedron are coplanar, the tetrahedron is *degenerate*. The topology of a mesh determines the orientation of each element relative to the orientations of all the others. Hence, if two tetrahedra share a triangular face, and the fourth vertices of those tetrahedra both lie on the same side of the face, then one of the tetrahedra must be inverted.

The algorithms assume that the *quality* $q(t)$ of a tetrahedron t is positive if t has the correct topological orientation, zero if t is degenerate, and negative if t is inverted. Many such quality measures are available in the literature [3, 7].

For generality, the edge removal algorithm is designed to work even if the initial mesh has inverted tetrahedra. (Inverted tetrahedra can sometimes be removed by targeting their edges for removal.) However, the multi-face removal algorithm relies on there being no inverted elements in the mesh, and uses that assumption to limit its search. If the initial mesh does not have inverted tetrahedra, then neither algorithm will ever create any, because they only perform transformations that improve the quality of the worst element.

The algorithms take into account faces and edges of the mesh that cannot be removed. These privileged faces and edges are called *subfacets* and *subsegments*. A *bounding surface* is any surface that acts as a domain boundary—either as a boundary between meshed and unmeshed regions of space, or as an internal surface that the mesh must conform to so boundary conditions can be applied, or as an interface between regions that must be represented by distinct elements. (For example, in a heat conduction problem where two different materials meet, elements are not permitted to penetrate the interface.) A *subfacet* is any triangular face of the mesh that lies in a bounding surface, and thus cannot be removed. Sometimes, however, two subfacets that lie in the same bounding surface and share an edge can have their shared edge "flipped," so that they are replaced by two different subfacets while the bounding surface remains intact. Subfacets include every triangular face that is a face of only one tetrahedron (rather than two).

A *bounding curve* is any curve where two bounding surfaces intersect, or any curve that, for whatever reason, the mesh must conform to. A *subsegment* is any edge of the mesh that lies in a bounding curve, and thus cannot be removed. If two subfacets share an edge, and the subfacets do not lie in the same bounding surface, the shared edge must be a subseg-

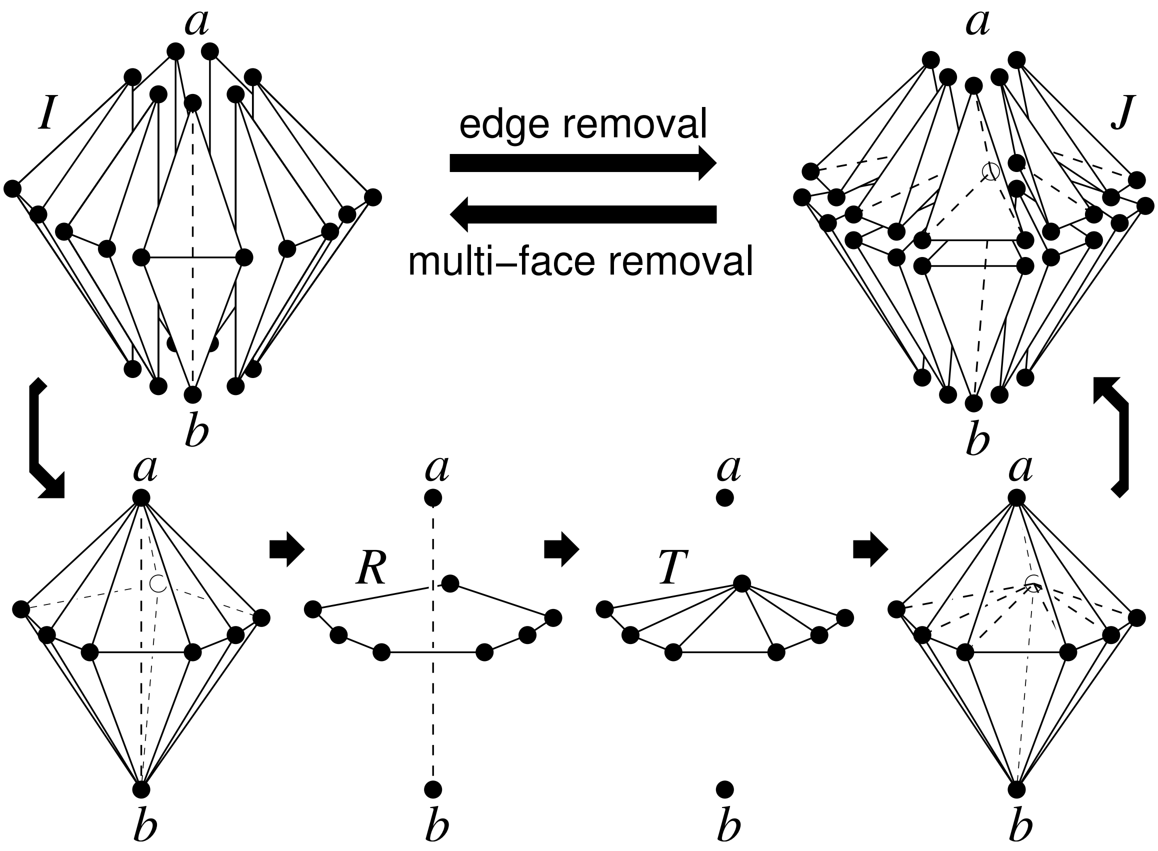


Figure 2: An edge removal transformation.

ment. If two subfacets that lie in the same bounding surface share an edge, and the shared edge is not a subsegment, it can be flipped.

2 Edge Removal

Edge removal is a transformation that removes an edge from a tetrahedral mesh. It includes the 3-2 and 4-4 flips, as well as other transformations that remove more tetrahedra. All the faces and tetrahedra that include the edge are removed as well, and replaced by other faces and tetrahedra. One (sometimes two) of the new faces cuts the deleted edge. Figure 2 illustrates an edge removal that replaces seven tetrahedra with ten. In general, edge removal replaces m tetrahedra with $2m - 4$. Its inverse, called a multi-face removal, is discussed in Section 3. Edge removal was suggested by Briere de l'Isle and George [1], and later studied by de Cougny and Shephard [2] and Frietag and Ollivier-Gooch [4].

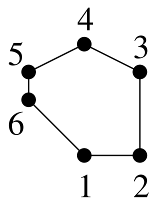
Let ab be an edge of the mesh with vertices a and b. If ab is a subsegment, it cannot be removed. Otherwise, let I be the set of tetrahedra that include ab. If ab lies in an exterior boundary of the mesh, then the tetrahedra in I subtend an angle of roughly 180° around ab, starting and ending at the boundary. (If the angle is not roughly 180°, ab must be a subsegment.) Otherwise, ab lies in the interior of the mesh, and the tetrahedra in I entirely surround ab, as in Figure 2.

Each tetrahedron in I has an edge opposite ab. Let R be the set of these edges. R forms a ring of edges around ab, as illustrated. Hence, R is a (non-planar) polygon in three-dimensional space. An edge removal transformation finds a triangulation T of R, as illustrated. T is topologically a two-dimensional triangulation, even though its triangles are embedded in three-dimensional space. Each of these triangles is connected to a and b to form two new tetrahedra. Let $J = \bigcup_{t \in T}\{\mathrm{conv}(a \cup t), \mathrm{conv}(b \cup t)\}$, as illustrated. The tetrahedra in J 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. This problem is solved by a dynamic programming algorithm of Klincsek [6], which is a general-purpose algorithm for constructing optimal triangulation of simple polygons for many different measures of optimality.

Let the vertices of R be $v_1, v_2, \ldots, v_m$ in counterclockwise order about ab (as viewed from a). The optimal triangulation of R is found by first solving a sequence of smaller subproblems. Let R_{ij} (with $i < j$) be a ring of edges whose vertices are $v_i, v_{i+1}, \ldots, v_j$. The ring R_{ij} is made up of a portion of the ring R, plus an extra edge ij. Let T_{ij} be a (two-dimensional) triangulation of R_{ij}. The triangulation

Q	1	2	3	4	5	6
1		∞	.7	.7	.7	.2
2			∞	.6	.6	.2
3				∞	.3	.2
4					∞	.2
5						∞



K	1	2	3	4	5	6
1			2	3	4	4
2				3	4	3
3					4	4
4						5
5						

Figure 4: Tables used in Klincsek's dynamic programming algorithm for finding an optimal triangulation.

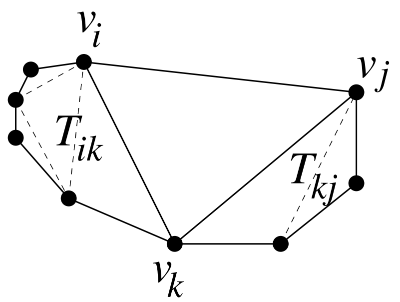


Figure 3: Decomposing a triangulation optimization problem into smaller subproblems.

T_{ij} induces a tetrahedralization $K_{ij} = \bigcup_{t \in T_{ij}} \{\mathrm{conv}(a \cup t), \mathrm{conv}(b \cup t)\}$. Say that the *quality* of T_{ij} is the quality of the worst tetrahedron in K_{ij}.

If $j = i + 1$, then T_{ij} consists of just one edge $v_i v_j$, and so the quality of K_{ij} is not defined. By convention, set $K_{i,i+1}$ equal to ∞. Otherwise, if $j \geq i + 2$, then $v_i v_j$ is the side of a triangle in T_{ij}, as Figure 3 illustrates. Let v_k be the third vertex of this triangle. The triangulation T_{ij} can be divided into three parts: T_{ik}, $\triangle ikj$, and T_{kj}.

The goal is to choose the triangulation T_{ij} so that its quality is maximized. There are two parts to this: choosing k correctly, and choosing T_{ik} and T_{kj} so that their qualities are optimal. The triangulations T_{ik} and T_{kj} are smaller subproblems that are solved recursively. The optimal choice of k is found by trying every possible value. Within the recursion, some subproblems arise repeatedly, so dynamic programming is used so that no subproblem is solved more than once.

The dynamic programming algorithm uses an $(m-1) \times m$ table Q, wherein $Q[i, j]$ records the quality of the optimal triangulation T_{ij} of vertices $v_i, v_{i+1}, \ldots, v_j$. To bootstrap the algorithm, set $Q[i, i+1]$ to ∞ for $1 \leq i < m$. The table entries $Q[i, j]$ for $j \geq i + 2$ are filled in using the recurrence

$$Q[i, j] = \max_{k \in [i+1, j-1]} \min\{Q[i, k], q(a, v_i, v_k, v_j), q(v_i, v_k, v_j, b), Q[k, j]\}, \quad (1)$$

where $q(u, v, w, x)$ is the quality of tetrahedron $uvwx$, as determined by any tetrahedron quality measure.

The table entries must be filled in by order of decreasing i and increasing j, so that $Q[i, k]$ and $Q[k, j]$ are computed before $Q[i, j]$. A second table K stores indices that are used to reconstruct the solution. $K[i, j]$ is set to the value of k that maximizes Expression (1). The tables Q and K are illustrated in Figure 4 for a sample polygon.

After the tables are filled in, if the quality $Q[1, m]$ of the optimal triangulation T_{1m} is better than the quality of the worst tetrahedron in I, it is safe to go ahead with edge removal. T_{1m} is reconstructed recursively by following the indices stored in K. The triangulation T_{ij} is composed of $\triangle v_i v_{K[i,j]} v_j$ and all the triangles in $T_{iK[i,j]}$ and $T_{K[i,j]j}$. The arrows drawn in the table K in Figure 4 illustrate how

```
RemoveEdge(a, b, R)
{ a and b are the vertices of the edge to be removed. }
{ R is a list of vertices v_1, v_2, ..., v_m around edge ab. }
    q_old ⇐ quality of the worst element that includes ab.
    (Q, K) ⇐ FillTables(a, b, R)
    q_new ⇐ Q[1, m]
    if q_new > q_old
        RemoveEdgeFlips(a, b, K)

FillTables(a, b, R)
    Let Q and K be two uninitialized (m − 2) × m tables
    for i ⇐ m − 2 downto 1
        for j ⇐ i + 2 to m
            for k ⇐ i + 1 to j−1
                q ⇐ min{q(a, v_i, v_k, v_j), q(v_i, v_k, v_j, b)}
                { q(·) is a function that returns the quality of a tetrahedron. }
                if k < j − 1
                    q ⇐ min{q, Q[k, j]}
                if k > i + 1
                    q ⇐ min{q, Q[i, k]}
                if k = i + 1 or q > Q[i, j]
                    Q[i, j] ⇐ q
                    K[i, j] ⇐ k
    return (Q, K)

RemoveEdgeFlips(a, b, K)
{ Replaces all the tetrahedra that include edge ab with tetrahedra induced by triangulation T_1m. }
{ K is an (m − 2) × m table of indices that identify the optimal triangulations T_ij. }
{ Warning: This algorithm may transiently create degenerate or inverted tetrahedra, }
{     even if they will not appear in the final tetrahedralization. }
    k ⇐ K[1, m]
    Flip23Recurse(a, b, K, 1, k)
    Flip23Recurse(a, b, K, k, m)
    Remove the edge ab with a 3-2 flip, creating the tetrahedra a v_1 v_k v_m and v_1 v_k v_m b

Flip23Recurse(a, b, K, i, j)
{ Creates the tetrahedra for triangulation T_ij. }
    if j ≥ i + 2
        k ⇐ K[i, j]
        Flip23Recurse(a, b, K, i, k)
        Flip23Recurse(a, b, K, k, j)
        Eliminate tetrahedra a b v_i v_k and a b v_k v_j with a 2-3 flip, replacing them with a v_i v_k v_j, v_i v_k v_j b, and a b v_i v_j
{ Tetrahedron a b v_i v_j will only exist transiently, and so might have }
{     much worse quality than the final tetrahedralization. }
```

Figure 5: Pseudocode for edge removal.

the triangulation is pieced together from solutions of subproblems. The new tetrahedra J are easily determined from T_{1m}.

Pseudocode for edge removal appears in Figure 5. Its running time is $\mathcal{O}(m^3)$, the amount of time the procedure FillTables needs to compute the values in tables Q and K. Because m is usually small, the cubic running time is not a great limitation.

De Cougny and Shephard [2] and Frietag and Ollivier-Gooch [4] present evidence that edge removal rarely improves the mesh for $m > 7$. Hence, a mesh improvement procedure can improve its speed without sacrificing much quality by setting an upper limit on the value of m for which edge removal is considered. The recommended limit is 7, but a programmer or user might choose any value from 5 to 10, depending on the relative importance of speed versus quality. Any hard-coded limit makes it possible to use statically allocated arrays to represent Q and K.

The top-level procedure, RemoveEdge, removes the edge

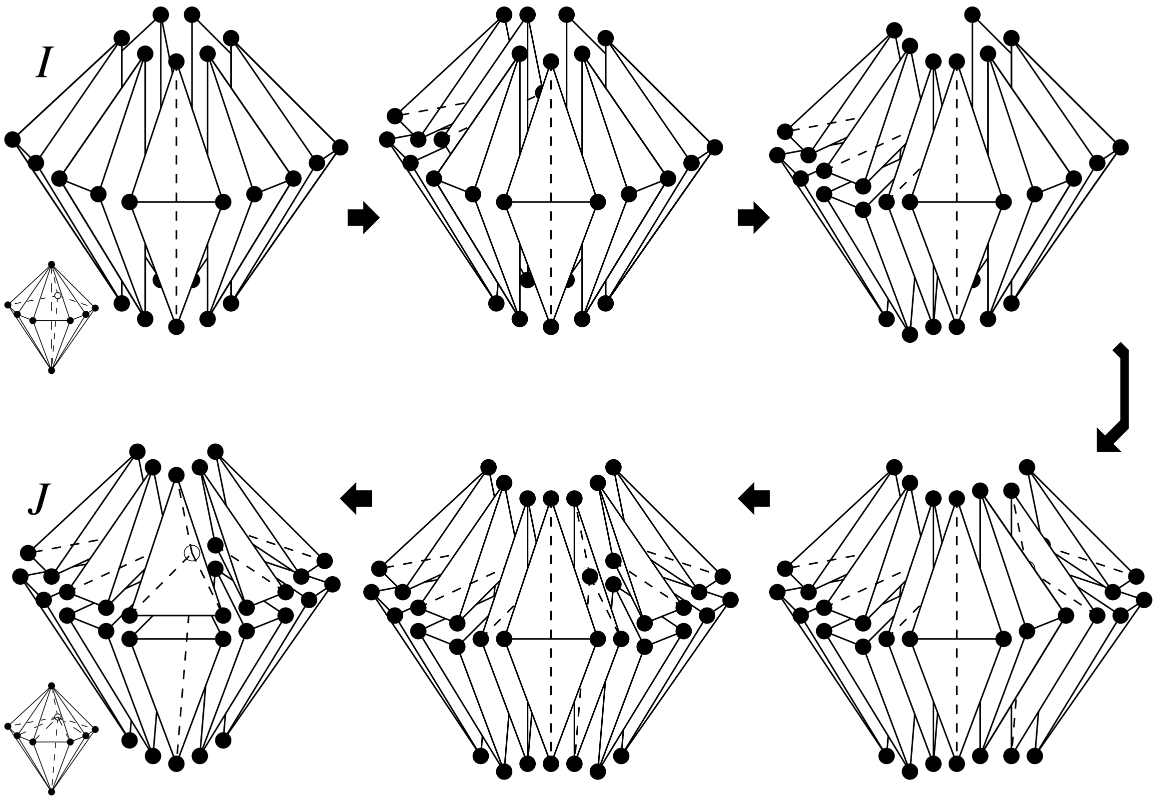


Figure 6: Implementing edge removal as a sequence of 2-3 flips followed by a single 3-2 flip.

ab only if the quality of the worst tetrahedron in J is better than the worst tetrahedron in I (Line 4). Hence, if the quality measure q is positive for tetrahedra that are correctly oriented, zero for degenerate tetrahedra, and negative for inverted tetrahedra; and I does not contain any degenerate or inverted tetrahedra, then neither will J. (This is why the pseudocode makes no explicit tests for inverted tetrahedra.) Observe that the pseudocode varies slightly from the description above, in that FILLTABLES does not fill in the entries $Q[i, i+1]$ or $K[i, i+1]$.

The pseudocode includes a routine called REMOVEEDGEFLIPS, which replaces the tetrahedra of I with the tetrahedra of J by performing a sequence of $\mathcal{O}(m)$ 2-3 flips followed by a single 3-2 flip that eliminates the edge ab. Figure 6 shows this sequence in action. These flips are topologically correct, but each 2-3 flip creates a tetrahedron that is not in J and will be eliminated by a later flip. The algorithm is oblivious to the quality of these transient tetrahedra, and they may be degenerate or inverted. However, they have no bearing on the quality of J.

There is a simple optimization to FILLTABLES that is not written into the pseudocode (to keep it simple). The idea is that FILLTABLES does not need to finish computing the value of $Q[i, j]$ if it becomes apparent that the value will be no better than q_{old}. To implement this optimization, REMOVEEDGE must pass the value of q_{old} to FILLTABLES as a parameter. If Line 10 determines that the quality of tetrahedron $av_iv_kv_j$ is no better than q_{old}, the procedure may write its quality into $Q[i, j]$ and break out of the inner loop (Lines 10 through 17) without computing the quality of $v_iv_kv_jb$. (The quality measure $q(\cdot)$ constitutes the most expensive part of the computation, so reducing the number of quality evaluations is worthwhile.) If the quality of $v_iv_kv_jb$ is no better than q_{old}, likewise the procedure can write it into $Q[i, j]$ and break out of the inner loop.

A more aggressive version of this optimization replaces the loop-based table filling (Lines 7 and 8) with a recursive table-filling procedure and memoization. Some table entries might not need to be filled in at all, so computation time is saved.

If ab lies in a bounding surface (be it an exterior or interior bounding surface), a few changes to the algorithm are necessary to ensure that J will respect the bounding surface. The modified algorithm appears in Figure 7.

If ab lies in an exterior bounding surface, then the ring R about ab is only half a ring; it is a chain beginning at some boundary vertex v_1 and ending at another boundary vertex v_m. In this case, the only change to the algorithm is that the final 3-2 flip is replaced by a 2-2 flip (which is like the 4-4 flip in Figure 1, but employs only the bottom two tetrahedra).

If ab lies in an interior boundary, the ring R is closed, but it is partitioned by the bounding surface in which ab lies into two

REMOVEBOUNDARYEDGE(a, b, C_1, C_2)
{ a and b are the vertices of the edge to be removed. }
{ C_1 is a list of vertices $v_1, v_2, \ldots, v_{m_1}$ half-way around edge ab. }
{ C_2 is either $\emptyset$ or a list of vertices $w_1, w_2, \ldots, w_{m_2}$ half-way around edge ab, }
{ with $w_1 = v_{m_1}$ and $w_{m_2} = v_1$. }
$q_{\text{old}} \Leftarrow$ quality of the worst element that includes ab.
$(Q_1, K_1) \Leftarrow$ FILLTABLES(a, b, C_1)
if $C_2 = \emptyset$
    $K_2 \Leftarrow \emptyset$
    $q_{\text{new}} \Leftarrow Q_1[1, m_1]$
else
    $(Q_2, K_2) \Leftarrow$ FILLTABLES(a, b, C_2)
    $q_{\text{new}} \Leftarrow \min\{Q_1[1, m_1], Q_2[1, m_2]\}$
if $q_{\text{new}} > q_{\text{old}}$
    REMOVEBOUNDARYEDGEFLIPS(a, b, K_1, K_2)

REMOVEBOUNDARYEDGEFLIPS(a, b, K_1, K_2)
{ Replaces all the tetrahedra that include edge ab with tetrahedra induced by triangulations T_1 and T_2. }
{ K_1 is an $(m_1 - 2) \times m_1$ table of indices that identify the optimal triangulations for C_1. }
{ K_2 is either $\emptyset$ or an $(m_2 - 2) \times m_2$ table for C_2. }
{ **Warning:** This algorithm may transiently create degenerate or inverted tetrahedra, }
{ even if they will not appear in the final tetrahedralization. }
$k \Leftarrow K_1[1, m_1]$
FLIP23RECURSE$(a, b, K_1, 1, k)$
FLIP23RECURSE(a, b, K_1, k, m_1)
if $K_2 = \emptyset$
    Remove the edge ab with a 2-2 flip, creating the tetrahedra $av_1v_kv_{m_1}$ and $v_1v_kv_{m_1}b$
else
    $l \Leftarrow K_2[1, m_2]$
    FLIP23RECURSE$(a, b, K_2, 1, l)$
    FLIP23RECURSE(a, b, K_2, l, m_2)
    Remove the edge ab with a 4-4 flip, creating the tetrahedra $av_1v_kw_1$, $v_1v_kw_1b$, $aw_1w_lv_1$, and $w_1w_lv_1b$

Figure 7: Pseudocode for edge removal in a bounding surface.

edge-disjoint chains C_1 and C_2. In this case, each chain is treated separately, and separate tables are filled out for each chain. Two independent sequences of 2-3 flips are executed, one for C_1 and one for C_2. After these are complete, ab is removed by a single 4-4 flip.

3 Multi-Face Removal

Multi-face removal is the inverse of edge removal. It removes one or more triangular faces from a tetrahedral mesh, while creating a new edge and a ring of new faces around it. It includes the 2-3 and 4-4 flips, but also includes other transformations involving more tetrahedra. An m-face removal replaces $2m$ tetrahedra with $m + 2$. (An m-face removal actually removes $3m - 2$ faces, but it is the m faces of the triangulation T, defined in Section 2, that are interesting.) Multi-face removal is suggested by de Cougny and Shephard [2], who offer intriguing evidence that a two- to five-face removal is more likely to succeed (improve the quality of the worst tetrahedron) than an individual 2-3 flip.

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 $\text{conv}(f \cup a)$ and $\text{conv}(f \cup b)$, as illustrated in Figure 8. Let F be the set of triangular faces that are sandwiched between a and b. Assuming the mesh is a geometrically valid tetrahedralization with no inverted elements, F has the topology of a two-dimensional triangulation, which makes it easy to reason about F and draw pictures of its connectivity.

Let G be a graph whose nodes represent faces in F. (For simplicity, I will call the nodes of G the same names as the corresponding faces of the mesh, and the edges of G the same names as the corresponding mesh edges.) Two nodes in G are connected if and only if the corresponding faces share an edge. Consider two faces in F, called f_1 and f_2, that share an edge e, illustrated in Figure 8. The tetrahedra $\text{conv}(f_1 \cup a)$ and $\text{conv}(f_2 \cup a)$ share a triangular face $\text{conv}(e \cup a)$, and the tetrahedra $\text{conv}(f_1 \cup b)$ and $\text{conv}(f_2 \cup b)$ share a triangular face $\text{conv}(e \cup b)$. The edge e is included in exactly four triangular faces, namely f_1, f_2, $\text{conv}(e \cup a)$, and $\text{conv}(e \cup b)$. The multi-face removal algorithm presented here uses the fact that the faces are joined by edges "of degree four" to identify a group of suitable faces.

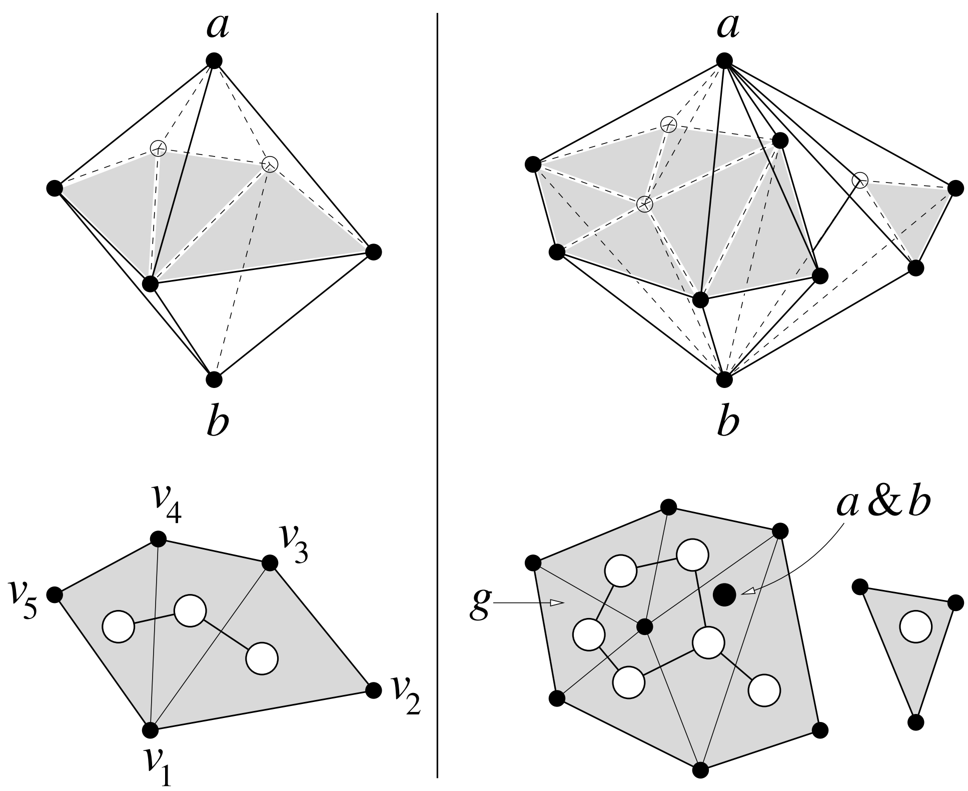


Figure 9: In the left example, the faces sandwiched between a and b form a tree, and the vertices of these faces form a ring $v_1 \ldots v_5$. All these faces can be removed by a multi-face removal transformation. In the right example, the faces do not form a tree; there are two connected components, one of which contains a cycle. A multi-face removal is still possible, but not all the faces may participate. Observe that because of the positions of a and b, the face g cannot participate without creating an inverted tetrahedron. Removing all such faces from the graph eliminates all cycles.

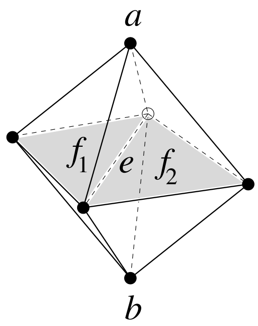


Figure 8: Two neighboring faces, f_1 and f_2, sandwiched between a and b.

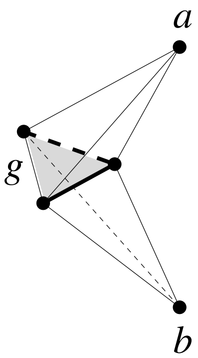


Figure 10: The union of the two tetrahedra adjoining the face g has two reflex edges (bold).

If G is a tree, then a multi-face removal of the faces in F is topologically possible, but may or may not be geometrically desirable. (In particular, it might create inverted tetrahedra.) Because G is a tree, the vertices of the faces in F form a ring $v_1, v_2, \ldots, v_m$ around the triangulation represented by F, as illustrated in Figure 9 (lower left). A multi-face removal creates the tetrahedra $abv_1v_2, abv_2v_3, \ldots, abv_mv_1$. (Refer back to Figure 2 for a picture.) It is straightforward to calculate the quality of these tetrahedra, compare them to the tetrahedra they would replace, and decide whether to perform the transformation.

However, if the goal is to maximize the minimum element quality, this goal is sometimes best served by removing just some of the faces in F. This section describes an algorithm for finding the best selection.

```
RemoveMultiFace(f)
{ f is a triangular face under consideration for removal. All faces sandwiched between }
{     the same two vertices (a and b) as f are considered for removal. }
{ Warning: This algorithm may transiently create degenerate or inverted tetrahedra, }
{     even if they will not appear in the final tetrahedralization. }
    if f lies in a bounding surface
        return        { f cannot be removed. }
    a, b ⇐ the apex vertices of the two tetrahedra that include f
    u, v, w ⇐ the vertices of f in counterclockwise order as viewed from a
    (o_uv, n_uv, g_uv) ⇐ TestNeighbor(f, a, b, u, v)
    (o_vw, n_vw, g_vw) ⇐ TestNeighbor(f, a, b, v, w)
    (o_wu, n_wu, g_wu) ⇐ TestNeighbor(f, a, b, w, u)
    q_old ⇐ min{q(a, u, v, w), q(u, v, w, b), o_uv, o_vw, o_wu}
    { q(·) is a function that returns the quality of a tetrahedron. }
    q_new ⇐ min{n_uv, n_vw, n_wu}
    if q_new > q_old
        Remove f with a 2-3 flip, creating tetrahedra abuv, abvw, and abwu
        for g ⇐ each of g_uv, g_vw, g_wu that is not ∅
            Flip32Recurse(g, f)

TestNeighbor(f, a, b, u, w)
{ f is a triangular face with vertices u and w. a and b are the vertices that sandwich f. }
    q_uw ⇐ q(a, b, u, w)
    if edge uw is included in exactly four triangular faces and edge uw does not lie in
                        a bounding curve or surface (excepting a surface that contains both a and b)
        g ⇐ the other face (besides f) adjoining edge uw and sandwiched between a and b
        v ⇐ the third vertex of g (besides u and w)
        { Note that u, v, w are in counterclockwise order as viewed from a. }
        j_uv = Orient3D(a, b, u, v)
        j_vw = Orient3D(a, b, v, w)
        j_wu = Orient3D(a, b, w, u)
        { Make sure removing g won't create a degenerate or inverted tetrahedron (see Figure 10). }
        if (j_uv > 0 and j_vw > 0) or (j_vw > 0 and j_wu > 0) or (j_wu > 0 and j_uv > 0)
            (o_uv, n_uv, h_uv) ⇐ TestNeighbor(g, a, b, u, v)
            (o_vw, n_vw, h_vw) ⇐ TestNeighbor(g, a, b, v, w)
            Let the children of g be h_uv (if not ∅) and h_vw (if not ∅)
            q_old ⇐ min{q(a, u, v, w), q(u, v, w, b), o_uv, o_vw}
            q_new ⇐ min{n_uv, n_vw}
            { q_old is the worst quality eliminated if g is removed. q_new is the worst quality created
                if g is removed. q_uw is the quality created if g is not removed, but f is. }
            if q_new > q_old or q_new > q_uw
                return (q_old, q_new, g)        { Indicates g should be removed if f is removed. }
    { If execution reaches this point, g should not be removed. }
    return (∞, q_uw, ∅)        { Replace ∞ with the maximum possible element quality. }
```

Note: $\textsc{Orient3D}(a, b, c, d) \quad = \quad \begin{vmatrix} a_x - d_x & a_y - d_y & a_z - d_z \\ b_x - d_x & b_y - d_y & b_z - d_z \\ c_x - d_x & c_y - d_y & c_z - d_z \end{vmatrix}.$

Figure 11: Pseudocode for multi-face removal, part 1. Flip32Recurse appears in Figure 13.

If G is not a tree, the vertices of the faces in F do not form a ring, and multi-face removal of F is not defined. However, multi-face removal can be enabled by pruning faces from G. Every cycle can be eliminated by pruning faces like g in Figure 9 whose participation in multi-face removal is guaranteed to create an inverted tetrahedron (no matter which other faces participate). Let $t_a = \text{conv}(g \cup a)$ and $t_b = \text{conv}(g \cup b)$ be the two tetrahedra that include g. If the polyhedron $t_a \cup t_b$ has two reflex edges, as illustrated in Figure 10, or even two edges where the faces meet at $180°$ angles, then a multi-face removal transformation cannot remove g without creating degenerate or inverted elements, because the vertex where

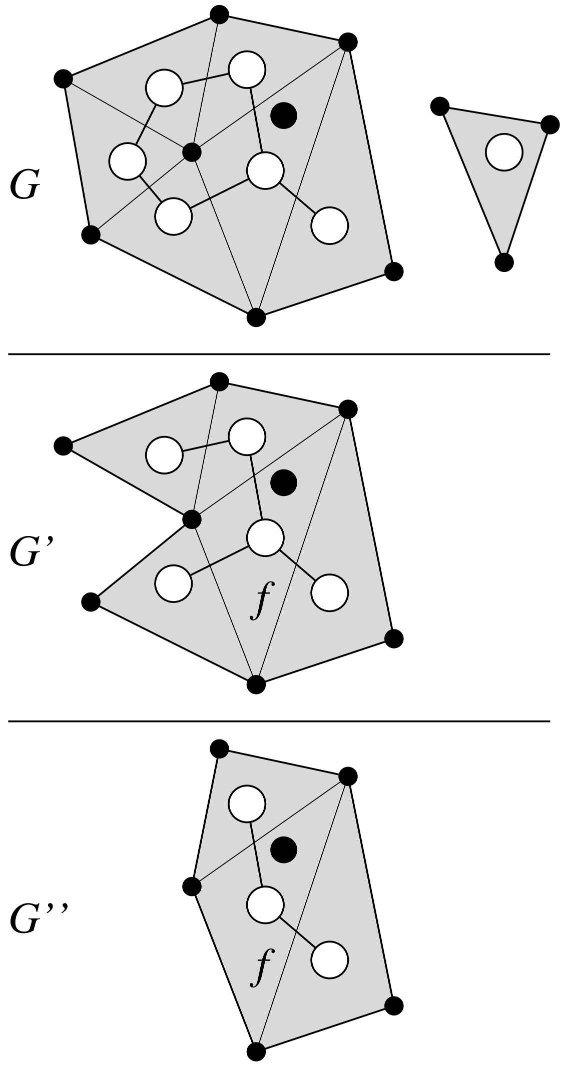


Figure 12: G is the graph of faces sandwiched between a and b. G' is the graph of faces visited by REMOVEMULTIFACE(f). G'' is the graph of faces that are actually removed.

the reflex edges meet will be inside one of the new tetrahedra.

Figure 11 lists pseudocode for an algorithm that targets a specific face f for removal, and determines the optimal set of triangular faces to remove. If f is a subfacet, it cannot be flipped. Otherwise, the algorithm uses a depth-first search of G to find other faces sandwiched between a and b. The depth-first search identifies edges of G by using the property that each corresponding mesh edge is included in exactly four triangular faces (Line 15). The search rejects faces that induce two reflex edges as depicted in Figure 10 (Lines 18–21). The search visits only the component of G reachable from f. Let G' be the graph of faces visited by the depth-first search. G' is a tree, as Figure 12 shows.

Although G' is topologically eligible for multi-face removal, better geometric results might be obtained by pruning more faces. Designate one node of G' as the root of the tree; f is a natural choice. (The choice of root is discussed further below.) Each face in G' except f has a parent, found by taking one step toward f in G'. The pruning of leaves and branches from G' yields a tree G'' (see Figure 12) which is also rooted at f.

For each face $g \in G'$ (except f), the algorithm asks the question: if g's parent is removed by multi-face removal, should g be removed? (If g's parent is not removed, then g cannot be removed either.) This question is answered by Lines 22–27, which compare the qualities of the tetrahedra that will be generated in each case. The question can only be answered for g after it has been answered for all of g's children, and the recursive TESTNEIGHBOR procedure reflects this. All decision-making starts at the leaves of the tree and propagates to the root.

The recursive procedure also constructs the graph G'', which is stored by having each face store pointers to its children (Line 24). This information is used later by the procedure FLIP32RECURSE (Figure 13), which performs the flips that effect the multi-face removal. First f is removed by a 3-2 flip (Line 11); then the other faces in G'' are removed by 2-3 flips (Line 30).

The running time of REMOVEMULTIFACE is linear in the number of faces visited. (This might be much larger than m, the number of faces removed.) REMOVEMULTIFACE does not perform the removal if the quality of the worst element will not improve (Line 10).

If the edge ab does not pass through one of the faces in G'', then a multi-face removal will create at least one inverted tetrahedron. Therefore, it makes sense to always start the search (and root the tree) at a face that intersects ab. For this purpose, the routine MULTIFACEREMOVAL is supplied in Figure 13, and is usually the best way to invoke multi-face removal. Note that MULTIFACEREMOVAL(f) may perform a multi-face removal that does not remove f. It removes the optimal set of faces sandwiched between a and b.

4 Ruminations and Open Problems

Although mesh improvement methods can sometimes turn unusable meshes into high-quality meshes, they often get stuck in local optima that are far from the global optimum. Joe [5] 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. A basic operation that worsens the mesh will be executed if it creates an opportunity for other operations to lead to new heights.

Edge removal and multi-face removal are both examples of composite operations, as they can be implemented as se-

```
FLIP32RECURSE(g, f)
{ g is a triangular face. f is its (already removed) parent. }
{ Warning: This algorithm may transiently create degenerate or inverted tetrahedra, }
{       even if they will not appear in the final tetrahedralization. }
30      Remove g with a 3-2 flip, removing the edge g used to share with f
        { Let g = △uvw, where uw was also an edge of f before f was removed. Line 30 replaces }
        {     tetrahedra auvw, uvwb, and abuw with abuv and abvw. }
31      for each child h of g       { Children are determined by Line 24. }
32          FLIP32RECURSE(h, g)

MULTIFACEREMOVAL(f)
{ Similar to REMOVEMULTIFACE, but this routine may perform a multi-face removal that }
{       does not remove f. All faces sandwiched between the same two vertices (a and b) as f }
{       may be considered for removal. }
33      a, b ⇐ the apex vertices of the two tetrahedra that include f
34      g ⇐ the first triangular face intersected by line segment ab that does not contain a or b (if one exists)
35      if g is not sandwiched between a and b or ab intersects a vertex of g or g is a subfacet
36          return      { g cannot be removed. }
37      REMOVEMULTIFACE(g)
```

Figure 13: Pseudocode for multi-face removal, part 2. MULTIFACEREMOVAL is an alternative entry point to REMOVEMULTIFACE that is more likely to find a good transformation because it does not require that f be among the faces removed.

quences of 2-3, 3-2, 2-2, and 4-4 flips. Any hill-climbing mesh improvement program that employs only the basic flips can be improved by including edge removal and multi-face removal operations as well.

This brings up the question, what other composite operations are likely to yield good results? Of course, almost any composite operation might be useful in the right peculiar circumstances, but some are no doubt useful far more often than others. There is a trade-off between the time spent searching for opportunities to apply an operation, and the amount they improve a mesh. More effort should be made to identify the composite operations that best reward the time spent finding sites to apply them to.

Joe [5] himself identifies about nine composite operations—some which appear to be of daunting complexity—and studies which are most effective. Unfortunately, there appears to have been little follow-up work by other researchers, perhaps because Joe's paper is so complicated. However, on close inspection it appears that even the most elaborate composite operations in his paper can be expressed as one or two edge removal operations. The study of composite edge removal transformations thus appears to be a fruitful direction for mesh improvement research.

The other clear direction for further research is to develop algorithms that determine where topological transformations (and node smoothing) should be applied, as quickly as possible. There are many papers describing mesh improvement operations, but few that try to develop fast algorithms for determining when and where to apply those operations. Freitag and Ollivier-Gooch [4] offer one of the best examples, but there is still a wide frontier left to be claimed.

References

[1] E. Briere de l'Isle and Paul-Louis George. *Optimization of Tetrahedral Meshes*. IMA Volumes in Mathematics and its Applications **75**:97–128, 1995.

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

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

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

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

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

[7] Jonathan Richard Shewchuk. *What is a Good Linear Element? Interpolation, Conditioning, and Quality Measures*. Submitted to the Eleventh International Meshing Roundtable, 2002.