Variational Delaunay Approach to the Generation of Tetrahedral Finite Element Meshes*

Petr Krysl† Michael Ortiz‡

Abstract. We describe an algorithm which generates tetrahedral decomposition of a general solid body, whose surface is given as a collection of triangular facets. The principle idea is to modify the constraints in such a way as to make them appear in an unconstrained triangulation of the vertex set à priori. The vertex set positions are randomized to guarantee existence of a unique triangulation which satisfies the Delaunay empty-sphere property. (Algorithms for robust, parallelized construction of such triangulations are available.) In order to make the boundary of the solid appear as a collection of tetrahedral faces, we iterate two operations, edge flip and edge split with the insertion of additional vertex, until all of the boundary facets are present in the tetrahedral mesh. The outcome of the vertex insertion is another triangulation of the input surfaces, but one which is represented as a subset of the tetrahedral faces. To determine if a constraining facet is present in the unconstrained Delaunay triangulation of the current vertex set, we use the results of Rajan which re-formulate Delaunay triangulation as a linear programming problem.

Keywords: Finite element method, tetrahedral mesh, boundary constraints, variational Delaunay

*Submitted on 1/21/1999 to the International Journal for Numerical Methods in Engineering.

†Staff scientist, California Institute of Technology, pkrysl@atlantis.caltech.edu

‡Professor of Aeronautics and Applied Mechanics, California Institute of Technology, ortiz@atlantis.caltech.edu

Introduction

Decomposition of arbitrarily complex three-dimensional solids into tetrahedral finite elements (tetrahedrization) is a very important aspect of finite element simulations. The ideal algorithm should be fast and robust, and the existence of a "silver bullet" technique suitable for all possible applications is improbable. Octree-based algorithms proved useful [1, 2, 3], and although they tend to be rather complicated, and not quite easily parallelizable, some new developments, such as guaranteed quality [4], make them interesting. The advancing front technique [5, 6, 7, 8, 9] is a heuristic, and although some of the features of this approach are very attractive (good quality tetrahedra tend to be produced near the external surfaces, arbitrary gradation is easy to achieve), lack of robustness in three-dimensional geometries is a serious impediment.

Algorithms based on the Delaunay empty circumsphere property are hampered in three dimensions, in contrast to planar geometries, by the nonexistence of a "constrained triangulation" [10]. (In a constrained Delaunay triangulation, the circumsphere of any simplex must be either empty, or the vertex inside the circumsphere must be occluded to any point inside the simplex by some constraining simplex.) A number of attempts to deal with this difficulty appeared in the literature, the post-processing approaches of Weatherill [11, 12] and George [13] being prominent among them.

To ellucidate our solution, we suggest to consider the following scenario: A three-dimensional solid is described by its boundary triangulation. The task is to find the decomposition of its volume into tetrahedral elements such that the triangulated boundary surfaces are "reasonably" approximated as collections of tetrahedral faces. One possibility is to represent the input constraining surfaces exactly, i.e. each triangular facet on the input surfaces corresponds to a face of at least one tetrahedron. Another approach becomes possible when the constraining surfaces are not required to appear *exactly*: the constraining surface triangulation may be modified to yield a geometrically and topologically *similar* triangulation [14], which has a potential to make the task of volume triangulation much easier. And that is precisely our point: We make it possible to use ordinary, unconstrained Delaunay triangulation, which is well understood, robust, and efficient, by modifying the constraining surface triangulations.

The principal idea of this work is actually quite well-aged. Hermeline seems to have been the first to propose incorporating boundaries of 3-D

objects into unconstrained Delaunay triangulations [15]: Instead of trying to extract the missing constraining facets a posteriori from a convex hull mesh, the input entities are complemented by additional vertices and facets which make an approximation (tesselation) of the constraining surfaces appear in the mesh ab initio. It is then easy matter either to construct only those tetrahedra which are in the region of interest, or to collect those tetrahedra from the convex hull mesh. Since then the idea re-surfaced in various guises; consult References [16, 10, 17, 18, 19].

Our approach differs from the preceding work in that we use the variational framework developed by Rajan [20] to determine if a facet of a constraining surface appears in the unconstrained Delaunay triangulation of the current set of vertices. If the facet is missing, we apply two local modification operations with the goal of improving sampling of the constraining surfaces. Decreased distances between surface vertices lead to the appearance of edges and facets. In difference to previous work we do not restrict the constraining surfaces in any way, in particular they may be non-planar, with multiple handles, and small input angles are allowed. However, this lack of restraint means that the technique currently comes without theoretical guarantees of termination.

The outline of the paper is as follows. We begin by giving a high-level view of our algorithm, and by specifying its inputs and outputs in Section 1. Subsequently, we order the sections according to the top-level algorithmic steps: Generation of internal vertices (Section 2), securing of non-denegerate vertex positions (Section 3), determination of presence of a given facet as a tetrahedron face in the unconstrained Delaunay triangulation of the vertex set (Section 4). Section 5 presents our (heuristic) algorithm of surface modification with the goal of making all the surface facets appear as simplices in the final triangulation, and finally we briefly outline the advancing front Delaunay volume triangulation in Section 6, the removal of slivers (Section 7), and parallel execution (Section 8). The paper is concluded by illustrating the present algorithm on number of applications. We also point out some points worthy of further investigations.

1 Outline of the algorithm

We first give a high-level outline of our algorithm. We resort to pseudo-code, here and in several other instances where algorithms are specified. The

notation is meant to be self-explaining.

algorithm *TRIANGULATE*
Read input and generate internal vertices.
Move vertices into non-degenerate positions.
Determine if all constraining surface facets are represented
by tetrahedron faces.
if (Are there are any non-Delaunay facets?) **then**
Apply surface Delaunayzation algorithm.
endif
Insert boundary facets into the advancing front and generate tetrahedra.
Remove slivers.

As *input*, the volume mesher is given a list of vertices and a list of surface facets. *Vertex data* comprises a unique integer identifier, coordinates, and optionally mesh size at the vertex. The *surface facets* represent boundary or interface surface triangulations, and are specified by listing the three identifiers of its vertices given counterclockwise when looking against the "outer" normal of the surface. Each facet is associated with a unique topological face. Furthermore, the 3D regions on each side of the facet are specified: The normal of the facet points into region r_1; region r_0 is on the other side. If the facet bounds just a single region (region r_1 is the semi-infinite space representing the "outside"), the facet is oriented such that its normal points out of the solid.

The *output* of our algorithm is obviously the tetrahedral decomposition of the volume (tetrahedra and their vertices), but also the modified surface triangulation (vertices and facets).

2 Generation of Internal Vertices

The mesher has the ability to generate additional vertices "in the volume" of the solid. We adopt a simple two-stage technique. First, we generate vertices close to the boundaries, which are offset appropriately inward from the barycenters of boundary facets. Next, an octree is constructed which bounds the solid so that its leaves are of size proportional to the desired edge length at the centroid of the leaf, and the vertices are generated at the center and at the mid-points of the edges of the leaves so that for a uniform edge

length specification one obtains arrangement in the form of an face-centered cubic crystal lattice [21]. The vertices are in both stages generated only if they do not fall too close to another vertex (which could have been given as input), and if they do not come too close to the surfaces of the solid. Note that it is not necessary to perform any expensive in/out tests to determine if the vertex is actually inside the solid, because the advancing front technique used to mesh the interior simply disregards vertices outside. The position of the generated vertices is randomly perturbed by a vector of magnitude of approximately 1% of the edge length.

A promising alternative approach to the generation of the internal vertices has been proposed by Fuchs [22] in the context of almost regular Delaunay triangulations.

3 Random shifting of vertex positions

In order to deal with the ambiguity associated with co-spherical points (degenerate Delaunay triangulations), we choose to move vertices into general positions by a small random shift. The default magnitude of the shift is 10^{-4} of the edge length (mesh size) at the vertex. For very closely spaced surfaces this could lead to (self-)intersections, but such situations can be avoided by shifting the vertices in tangent planes (along tangents to surface edges) only. The random perturbations yield vertex sets with a high probability of general vertex positions; see Reference [23] for a discussion.

4 Delaunayhood of Boundary Facets

First, some notation, for the sake of brevity. We will denote facets represented by a tetrahedron face as *Delaunay* facets, and those that do not coincide with any face, as *non-Delaunay* facets. The property of facet b of being Delaunay or not is called the *Delaunayhood* of b. Furthermore, $P_i(b)$ denote the vertices of the facet b, $\widehat{P}(b)$ is the fourth vertex of the tetrahedron $T(b)$ piled up on a Delaunay facet b, CircS[T] is the circumsphere of tetrahedron T. The smallest, or equatorial, circumsphere of facet b is meant by CircS[b].

How to determine if a boundary facet is represented in the unconstrained Delaunay mesh is crucial to our approach. The empty equatorial sphere criterion used by Miller et al. [17] and Shewchuk [19] is a sufficient, but not a

necessary condition. Hence, one can hope to introduce less Steiner vertices with a less restrictive criterion. Our approach is based on the optimality results for Delaunay triangulations advanced by Rajan [20]: Minimize function F defined on the convex hull of vertices $\boldsymbol{P}_i$, $\mathrm{Conv}(\boldsymbol{P}_i) \in R^n$ (in our case $n = 3$)

$$F(\boldsymbol{X}, \boldsymbol{\lambda}) = \sum_{i \in S_m} \lambda_i (\boldsymbol{P}_i - \boldsymbol{X})^2 \tag{1}$$

subject to the constraints

$$\lambda_i \geq 0, \quad \sum_{i=1}^{n} \lambda_i = 1, \quad \sum_{i \in S_m} \lambda_i \boldsymbol{P}_i = \boldsymbol{X} \ . \tag{2}$$

The coefficients λ_i are recognized as the barycentric coordinates of the point $\boldsymbol{X}$ within the enclosing tetrahedron. The set S_m comprises all input vertices, in general. However, in order to make the technique computationally efficient, we build the set S_m adaptively as described below.

In order to determine Delaunayhood of facet b we solve the linear programming problem (1) for a certain vertex set, S_m, at a point slightly offset against the direction of the facet normal (ie. into the solid). The distance is arbitrarily set at $K = 10^{-3}d$, where d is the characteristic dimension of the facet b. The vertex set S_m is initially chosen to include probable candidates for optimal solution of (1). Later, the set S_m is adjusted depending on the outcome of the solution of (1); see algorithm *TET_ON_FACET*.

The solution of (1) may be unbounded, in which case we assume there is no tetrahedron whose face represents the facet b. Another possibility is an infeasible solution (such as when the number of vertices is insufficient, or the vertices are all co-linear or co-planar). In that case the search region is inflated and the solution is re-tried. One can also arrive at a lower-dimensional solution, which happens when the sampling point $\boldsymbol{X}$ lies on an edge or on a face. In that case we remove the sampling point further from the facet and retry. Finally, the solution may give four non-zero lambdas in which case the circumsphere of the tetrahedron is checked if it is really empty as it should be (note that the vertex set used in the linear programming problem was not necessarily complete). If there is any vertex inside $\mathrm{CircS}[T(b)]$, it is an indication that the solution of (1) should be re-tried with an expanded vertex set S_m.

```
algorithm  TET_ON_FACET (Facet b): returns Tetrahedron
  Compute facet barycenter C and normal n
  Sampling point X  ← C − Kn;
  Estimate CircS[T(b)];
  B ← box enclosing CircS[T(b)];
  loop
    Solve (1) for S_m = vertices inside B
    case solution type do
    UNBOUNDED:
      return NULL
    INFEASIBLE:
      Increase B and continue
    LOWER_DIM:
      X ← C − Kn; recompute B and continue;
    OPTIMAL:
      T(b) ← tetrahedron from non-zero λ_i;
      if (Is CircS[T(b)] empty?) then
        return T
      else
        if (Solution of (1) yielded T for the second time) then
          return NULL;
        else Increase B to cover CircS[T(b)] and continue; endif
      endif
    done case
  done loop
```

The algorithm *IS_DELAUNAY?* deciding Delaunayhood of a given facet is described next. Note that the algorithm attempts to figure out status of a facet either after the facet has been created, in which case it does not know the status of the facet, or after some vertex has been added to the domain, in which case it needs to check whether the status of a previously Delaunay facet has changed or not.

```
algorithm IS_DELAUNAY?(Facet b): returns TRUE or FALSE
  if (Was b previously Delaunay?)
    if (Is CircS[T(b)] empty?) then return TRUE; endif
  endif
  if (Is CircS[b] empty?) then return TRUE;
  else
     foreach vertex P found in CircS[b] do
       P̂(b) ← P;
       if (Is CircS[T(b)] empty?) then return TRUE; endif
     done foreach
     Solve linear programing problem of Equation (1) → tetrahedron t
     if (Does b coincide with one face of tetrahedron t?) then
       return TRUE;
     endif
  endif
  return FALSE;
```

Note that in order to speed up the computations we use in the algorithm *IS_DELAUNAY?* not only the linear programming formulation (1), which in itself yields all the information one needs but at a relatively high cost, but we also perform some less expensive circumsphere tests first, which can save computational effort by enabling early decisions. In order to be able to use the circumsphere tests, we save for Delaunay facets the vertex which belongs to the tetrahedron whose face coincides with the facet.

Another speed optimization related to the solution of equation (1) is available through posing a limit on the size of the linear programming (LP) problem that is allowed to be actually solved (largest number of unknowns). When the size of the LP problem is too large, the results is assumed to be FALSE (in other words, the facet is non-Delaunay). We show the effect of this optimization in the Section 9.

5 Surface Delaunayzation

If there are any non-Delaunay facets in the constraining surfaces, the algorithm *DELAUNAYZATION* attempts to modify the surface meshes to make all of the constraining facets Delaunay. At the same time, the algorithm

endeavors to introduce as few additional boundary facets as possible, and to make the surface mesh as well-shaped as possible.

The introduction of additional vertices into the surface mesh in order to make all facets Delaunay is in general unavoidable. Unfortunately, each additional vertex has not only the potential to make some facet(s) Delaunay, it may also make others non-Delaunay. Hence, it is quite difficult to devise an algorithm which provably terminates for all possible inputs. Our algorithm is a heuristic, and we have not been able to provide theoretical guarantees of termination yet.

5.1 Delaunayzation algorithm

Two primitive surface modification operations are applied to the surface triangulation iteratively: *flip* of diagonal in a quadrilateral represented by two adjacent triangles, and *split* of a common edge by insertion of a new vertex. The first operation is inspired by the diagonal-swapping 2-D triangulation algorithm, the second is an attempt to maintain surface triangulation quality while introducing a Steiner point. The flip does not create any new vertex which could make other facets non-Delaunay, and is therefore preferable to split. On the other hand, in certain situations surface triangulations have been designed to possess some desirable properties such as convexity. Consider for example a coarsely discretized, very strongly curved leading edge of an airfoil. Concavities introduced by flips would unacceptably distort the air flow. In this case we could prohibit flips and use only splits which guarantee preservation of the "shape" of the surface triangulation.

The *DELAUNAYZATION* algorithm is described next. While there are any non-Delaunay facets, a pass over all the facets is made, flipping as many "diagonals" as possible. Next, each still non-Delaunay facet is split along its longest edge.

```
algorithm DELAUNAYZATION
  while (Is there a non-Delaunay boundary facet?) do
     perform FLIPS (NULL)
     perform SPLITS
  done while
```

The *FLIPS* algorithm works on a list of non-Delaunay facet, which can consist either of facets connected to a newly introduced vertex, or of all facets

in the surface meshes. The procedure tries to find an adjacent facet with which flip can be accomplished using the algorithm *FLIPPABLE_NEIGHBOR*.

```
algorithm FLIPS (Vertex V)
  if (Is V NULL?) then List L ← all facets in mesh
  else                 List L ← all facets connected to V
  endif
  foreach non-Delaunay b ∈ L do
    Facet b' ← FLIPPABLE_NEIGHBOR (b)
    Flip diagonal in quadrilateral b + b'
  done foreach
```

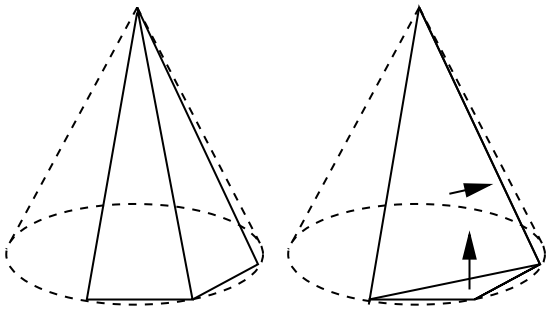


Figure 1: Prohibited flip leading to a sharp subtended angle. Left-hand side initial configuration; right-hand side situation resulting from flip.

The algorithm *FLIPPABLE_NEIGHBOR* assesses suitability of a facet adjacent to the facet b for flipping. Only facets which belong to the same topological face are considered. A facet b' is acceptable for flipping only if all the following holds:

1. Facet b' belongs to the same topological face as b; (This allows for edges to be preserved, even if they do not correspond to "true" edges along sharp folds of the surface.)

2. The quadrilateral composed of the two facets b and b' is convex as judged in projection to an average plane using an area-based measure;

3. The angle subtended by the facets in their flipped connectivity is less than some user specified limit; (This prevents creation of ugly ridges or grooves in the modified triangulation. See Figure 1, where the flipped facets on the right-hand side subtend angle close to $\pi/2$; such a flip is undesirable.)

4. The flip either increases the number of Delaunay facets, or at least improves the triangle quality measures (e.g. minimal corner angle).

```
algorithm FLIPPABLE_NEIGHBOR (Facet b): returns Facet
  L ← Order edges of b longest to shortest;
  foreach edge E ∈ L do
    Facet b' ← neighbor across E
    if (      Do b and b' belong to the same topological face?
       and    Is quadrilateral b + b' convex?
       and    Do triangles after flip subtend reasonable angle?
       and    (Does flip make b and b' Delaunay?
              or Are facets after flip of better triangle quality?)
      ) then return b'; endif
  done foreach
  return NULL
```

The algorithm for facet splitting, *SPLITS*, introduces a new vertex at the mid-point of the longest edge. For acute triangles this means increase in the minimal circumradius, but this is most probably corrected immediately by flips attempted with all triangles connected to the new vertex.

```
algorithm SPLITS
  List L ← all non-Delaunay facets
  foreach b ∈ L do
    List L' ← all facets across the longest edge of b
    perform SPLIT_FACETS (L')
    update Delaunayhood of all facets affected by the new vertex V
    perform FLIPS (V)
  done foreach
```

The procedure *SPLIT_FACETS* is rather straightforward, the only complication being that non-manifold situations need to be handled as illustrated in Figure 2.

algorithm *SPLIT_FACETS* (List L')
  Insert new vertex V at the mid-point of edge common to facets in L'
  foreach $b \in L'$ **do**
    replace b with new facets b' and b'' connected to V
  done foreach

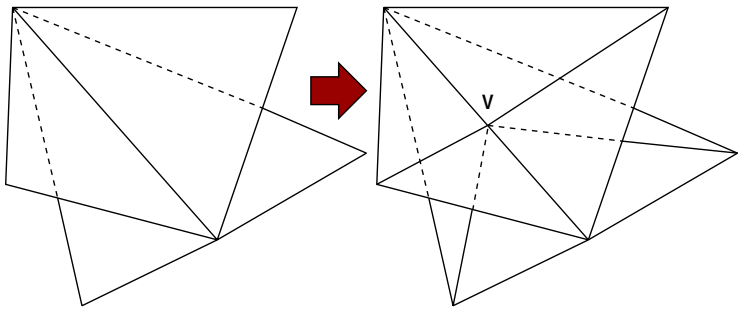


Figure 2: Splitting of facets sharing an edge.

5.2 Comparison with other published algorithms

Some ideas how to make some surfaces appear as 2-simplices of the unconstrained Delaunay triangulation have been advanced for some restricted classes of inputs. For "smooth" surfaces, Amenta and Bern [24] have shown that sufficiently dense sampling (as measured by the local feature size) makes the surface to appear in the unconstrained Delaunay tetrahedral mesh constructed with the input vertices and the Voronoi poles. The idea is to achieve "tangency" of the tetrahedron circumsphere to the constraining surface at surface vertices. Related ideas applied to the construction of shape skeletons have been advanced by Turkiyyah et al. [25].

For non-smooth surfaces, namely piecewise linear complexes (PLC; all constraining polytopes – edges and facets, have linear geometry), the sphere packing algorithm of Miller et al. [17] provably yields a boundary conforming

Delaunay triangulation. The input angles between any two boundary polytopes are restricted to 90^o. A variation of this algorithm has re-appeared in the thesis of Shewchuk [19] in the framework of a Delaunay refinement algorithm. The principal idea is to make the constraining polytopes appear by enforcing their minimal circumspheres empty of vertices starting with vertices (trivial), edges, and finally proceeding to enforce facets. The empty minimal circumsphere is a sufficient condition for an edge or a face to be present in the Delaunay triangulation, but it is not a necessary one. The restriction to PLC's is quite crucial to the feasibility of the above algorithms, since both rely on the fact that the tesselation of the constraining polygon is a Delaunay triangulation in two dimensions.

The input surface triangulation, which is the input to the present algorithm, could be understood as defining a piecewise linear complex (PLC). In such a case, the two above algorithms are applicable. However, both algorithms would modify this complex, so the input facets are not reproduced. Also, consider that all the surface triangulation edges and faces become edges (faces) of the PLC, and that the above algorithms would try to reproduce those edges (faces). There are evidently cases when these surface triangulation edges and faces are simply an artifact of the discretization process, and have no value in itself. In this respect, the ability of the proposed algorithm to perform flips is highly desirable, at least as an option.

6 Volume triangulation

Once the constraining facets are all made Delaunay, the tetrahedra can be generated by any unconstrained triangulation algorithm. We have chosen the advancing-front Delaunay [26, 27, 28, 21]. This technique is closely related to the original gift-wrapping algorithm [29] (also called incremental construction). Contrary to the classical advancing front algorithm [30], which generates nodes at the same time as tetrahedra, uses heuristics to compute the connectivity, and relies on intersection tests to ensure validity of the mesh, our implementation of the advancing front is based on the empty circumsphere property combined with the assumption of general vertex positions. The advancing-front Delaunay may not be the fastest serial algorithm, but it parallelizes easily [31].

7 Mesh Improvement

As is well known, creation of slivers (kites) cannot be avoided in Delaunay meshes constructed from pre-existing vertex sets in three dimensions. Since we use random perturbation of the vertex positions, slivers appear frequently in the interior. In addition, slivers also appear near the prescribed surfaces quite naturally. In order to reduce the number of slivers in the mesh, we modify the topology and geometry of the mesh in a post-processing step. Figure 3 shows three configurations of slivers with the adjacent tetrahedra. Slivers on the boundary can be deleted without difficulty (case A). Slivers in the interior can be deleted by swapping face $\triangle$dbe (case B), but are not removable in the case C (no two adjacent tetrahedra share a face). The non-removable slivers are opened up (expanded) by shifting slightly one of their vertices (the shift producing the largest minimal dihedral angle among all the connected tetrahedra is chosen). Further increase of the smallest dihedral angle can be achieved by insertion of additional vertices and local remeshing. This option has not been implemented yet, though.

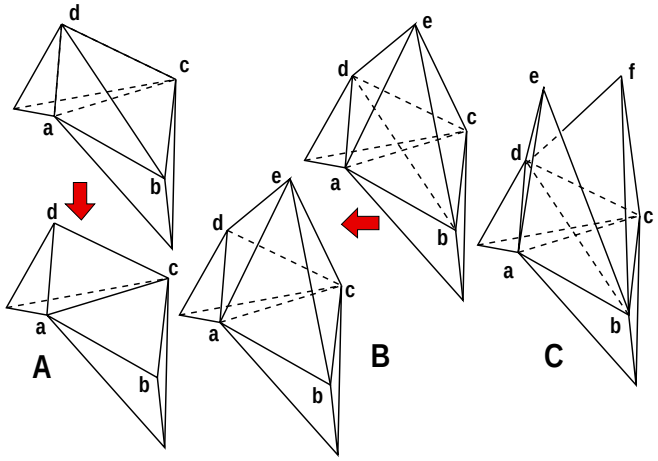


Figure 3: Removal of slivers.

8 Triangulation on parallel computers

Preliminary results suggest the present algorithm is amenable to parallelization with domain decomposition and message passing. Two major con-

stituent parts can be identified, the volume meshing itself (unconstrained Delaunay triangulation), and the surface modification.

No communication is required for the volume triangulation itself, since once can decompose the vertex set and generate the tetrahedra on each processor independently of the others. Implementation of the necessary searches can be expected to be tricky, though: It is imperative to download to each processor all the vertices that can affect triangulation on this processor. On the other hand, search efficiency dictates that only those vertices which actually participate in the triangulation on a given processor are searched.

The most expensive operation in the surface Delaunayzation is the evaluation of the Delaunayhood of affected facets once a new vertex is added to the current vertex set. The trick is, of course, to figure out which facets are affected by the vertex addition. Furthermore, communication is required in this step, since one needs to update the vertex and facet sets on each processor after each modification.

9 Examples

9.1 Mesh of Bracket

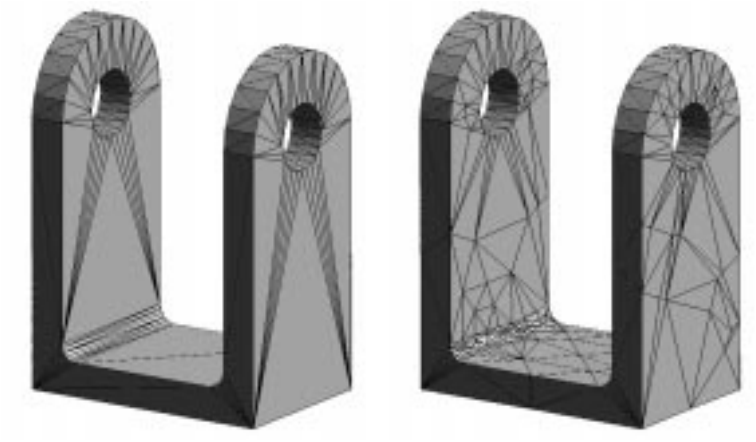
The first example illustrates the ability of the mesher to enforce the boundary constraint for a surface mesh consisting of very badly-shaped facets (Figure 4). The facets have been produced from a CSG model generated by the ACIS[1] geometry engine for the purpose of rendering with the default refinement settings. Hence the presence of needle-like and obtuse triangles. The input consisted of 180 vertices and 364 boundary facets (no vertices have been generated in the interior). The mesher inserted additional 55 vertices, increased the number of boundary facets to 474, and produced 460 tetrahedra in approximately 6.7 CPU seconds.[2]

9.2 Mesh of Wheel

In this example we illustrate the effect of turning off the check relying on the solution of (1) (Figure 5). In other words, Delaunayhood of a facet is checked

[1]Trademark of Spatial Technology Inc.

[2]SGI Octane, with 195MHz R10K CPU. All timings in this section are given for this platform.



(a) Input mesh (b) Output mesh

Figure 4: Mesh of a bracket.

simply by the empty minimal circumsphere criterion. The input consisted of 624 points and 1,248 boundary facets (no internal vertices have been generated). When the full check including the linear programming problem of (1) was used, no additional vertices have been inserted, and the mesher produced 1,378 tetrahedra (2.8 CPU seconds). When the check via (1) was turned off, the mesher added 219 surface vertices, 438 boundary facets, and produced 1810 tetrahedra (0.5 CPU seconds). The surface meshes are shown in Figure 5.

9.3 Mesh of A-PRIMED discriminator maze wheel

The maze wheel of the A-PRIMED discriminator[3] is representative of typical complex industrial geometries. Figure 6 shows fairly coarse surface meshes of the maze wheel (4,283 vertices, 8,618 boundary facets). The volume mesh of 12,022 tetrahedra was produced in 4.8 seconds (for limit on the size of the linear programming problem (1) set to 20).

Table 1 summarizes effects of setting a limit on the largest number of unknowns for which the linear programming (LP) problem of equation (1) is actually solved. Although the LP software package we have used is perhaps

[3]Designed by Gil Benavides at Sandia National Laboratories

(a) Input mesh (b) Output mesh (c) Output mesh for (1) turned off

Figure 5: Mesh of a wheel.

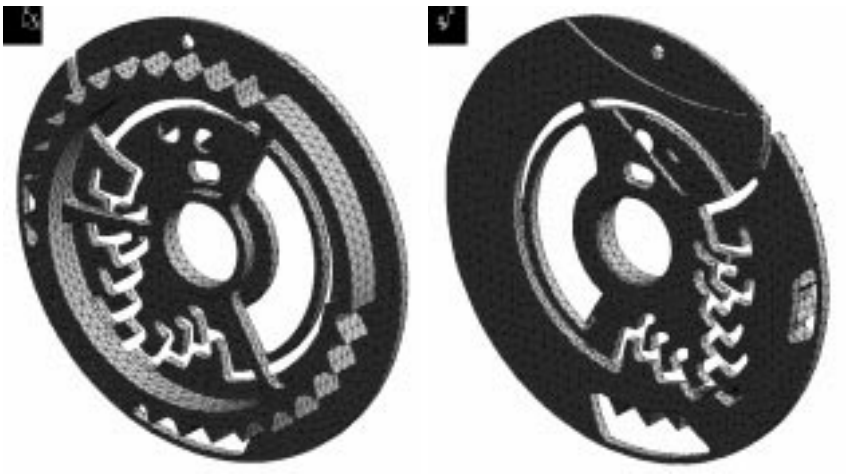
LP size limit	Delaunayzation time [s]	Total of solved LP's	Average number of unknowns	Total of added facets
0	1.03	0	0	150
20	1.10	11	16	150
200	17.56	1,164	100	150
500	152.60	2,664	197	146

Table 1: Comparison of Delaunayzation cost for different limits on size of the linear programming problem (1).

uncharacteristically inefficient and slow, the trend is clear: Solution of (1) is costly. Interestingly, this particular mesh is quite insensitive to the availability of the LP solution; only when the limit is set at 500 unknowns, a small reduction in the number of added boundary facets is achieved, and turning off the solution of (1) completely (setting the limit to zero) does not increase the number of added facets at all.

9.4 Mesh of Sphere

This example illustrates the effect of using or not using flips during the Delaunayzation process. Figure 7 shows the input surface triangulation and



(a) Front view (b) Back view

Figure 6: Mesh of a discriminator mazewheel.

the results of Delaunayzation with and without flips. Although the input triangulation is just an approximation of a smooth surface, it is reproduced as if it were a piecewise linear complex (PLC) when the flips are turned off. The increased size of the tetrahedrization is evident.

Flips allowed?	Triangulation time [s]	Total of added facets	Total of generated tetrahedra
no	1.08	192	688
yes	0.77	0	297

Table 2: Comparison of meshes of sphere with or without flips.

9.5 Mesh of Hub

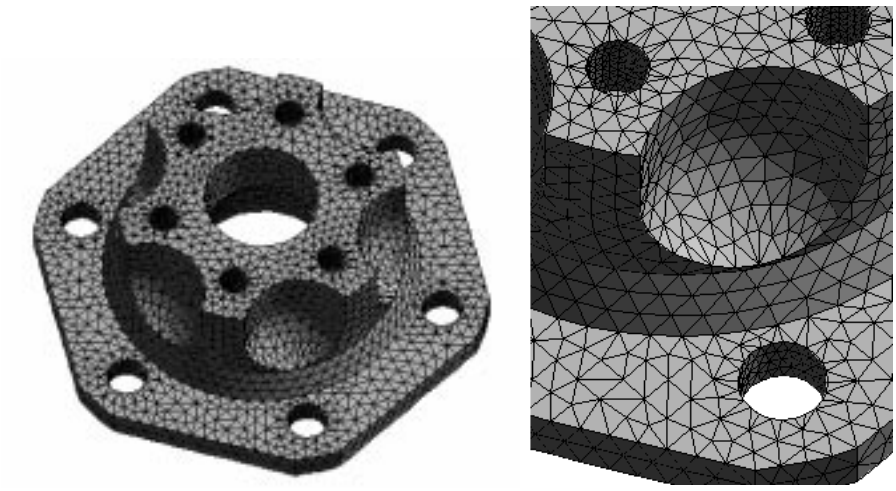
In the next example we present timings for a series of uniform meshes for a mechanical part. The surface meshes have been again produced for rendering purposes and hence are not quite as good as can be expected from

(a) Input (b) With flips (c) Without flips

Figure 7: Mesh of a sphere.

current finite element surface triangulation packages. Figure 8 shows the solid for the coarsest discretization (26,722 elements); detail of the fan-like, acute triangles near the small holes is also included. The timings are given in Figure 9 with a breakdown into major steps: reading of input, generation of internal vertices, initial evaluation of Delaunayhood of boundary facets, Delaunayzation procedure, generation of tetrahedra by the advancing front, and finally, removal of slivers. The large amount of time spent in the generation of internal points is noteworthy (about 25% of total time). This step is costly mainly because of the need to verify that vertices are not generated too close to each other and to the constraining surfaces. In order to speed up this step in adaptive analyses with many remeshings, we have devised a technique that avoids any searches and checks by generating internal vertices inside existing elements.

Figure 10 summarizes quality indexes for one particular mesh which seems to be sufficiently fine for strength analysis purposes (with 169,610 tetrahedra). The graph on the left shows the distribution of the radius ratio (inscribed sphere radius over circumsphere radius scaled by 3; ideal ratio is one); the graph on the right documents the distribution of the dihedral angle. The minimum radius ratio quality was 0.053, the minimal dihedral angle was 2.94 degrees, and the maximum dihedral angle was 173 degrees.



(a) Input mesh (b) Detail

Figure 8: Mesh of a hub.

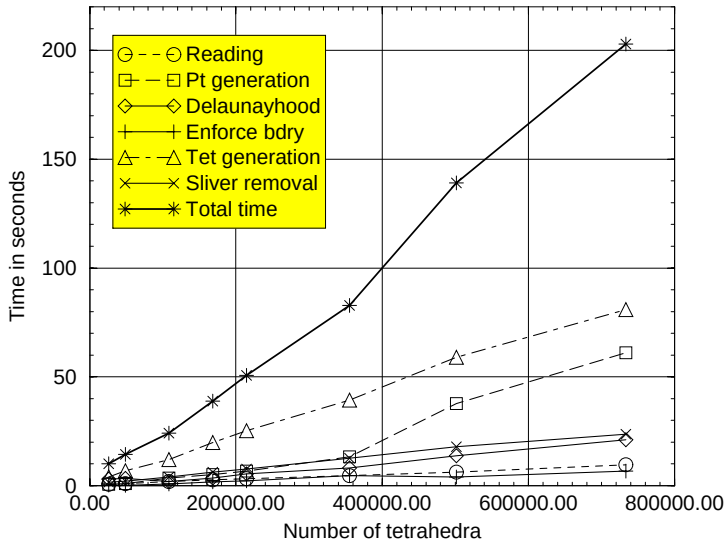
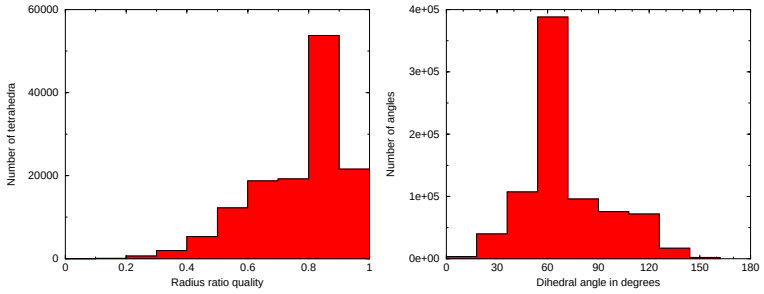


Figure 9: Timing for mesh of hub.



(a) Radius ratio (b) Dihedral angle

Figure 10: Mesh of a hub. Quality distribution for mesh with 169,610 elements.

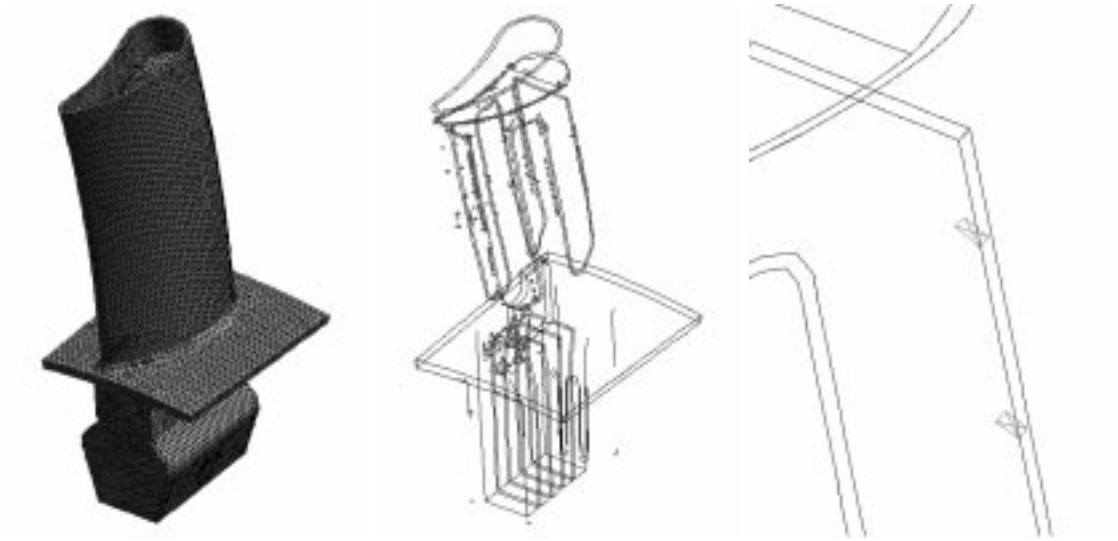
9.6 Mesh of Airfoil

In the next example we present mesh of an airfoil[4]. The topology and geometry of the object is rather complex. The input to the mesher consisted of 20,909 vertices and 41,866 boundary facets (see Figure 11(a)). The surface triangulation was of relatively good quality. Nevertheless, the mesher had to introduce a number of additional vertices (508) and facets (1,016) during the Delaunayzation procedure because of the large ratio of the mesh size and the local feature size (especially near intersections of thin walls). Figure 11(b) shows the additional vertices as dots on the background of topological edges. The tetrahedral mesh of 72,881 elements was generated in 59.9 seconds of which 62% were spent in the Delaunayzation algorithm. As documented in Figure 11(c), the presence of thin walls caused a number of slivers to appear (minimal dihedral angle 0.42 degrees, maximal dihedral angle 179.21 degrees). The reason was that our present mesh optimization technique does not handle slivers with all four vertices bound to two or more constraining surfaces. One possible cure could be introduction of an additional point at the barycenter of the sliver followed by Delaunay remeshing of the cavity remaining after the broken tetrahedra.

Mesh has also been produced with the Delaunayzation procedure based on empty minimal circumspheres (decisions based on Equation (1) were disabled). As expected, the mesher needed more additional vertices (544), and

[4]Prototype airfoil geometry courtesy of Howmet Research Corporation.

more boundary facets (1,088) to arrive at a fully Delaunay surface.



(a) Input mesh (b) Additional vertices (c) Slivers

Figure 11: Mesh of a prototype airfoil.

9.7 Mesh of Cranium and Brain

The approach presented above is applicable not only to external surfaces, but works equally well for internal surfaces, such as material interfaces or mathematically-sharp cracks. The only difference is that (in our implementation) the facets on these surfaces are not added to the initial advancing front.

One example of a mesh with material interfaces is presented in Figure 12 as an exploded view of the mesh separated into regions. The surface triangulations are polygonal models of simplified skull and brain (11,158 vertices and 22,322 boundary facets)[5]

[5]The surface triangulations have been produced from DXF geometry files purchased from Viewpoint DataLabs International, Inc. (registered trademark) by feature simplification, enforcement of topological consistency, and through removal of surface mesh interference via local geometry modifications.

There are three regions representing the skull, the cerebrospinal fluid with the meninges, and the brain. The surface mesh required relatively minor stitching – 36 additional vertices and 72 facets. The finished mesh consisted of 15,398 vertices and 82,126 tetrahedra, and took 40 seconds to generate.

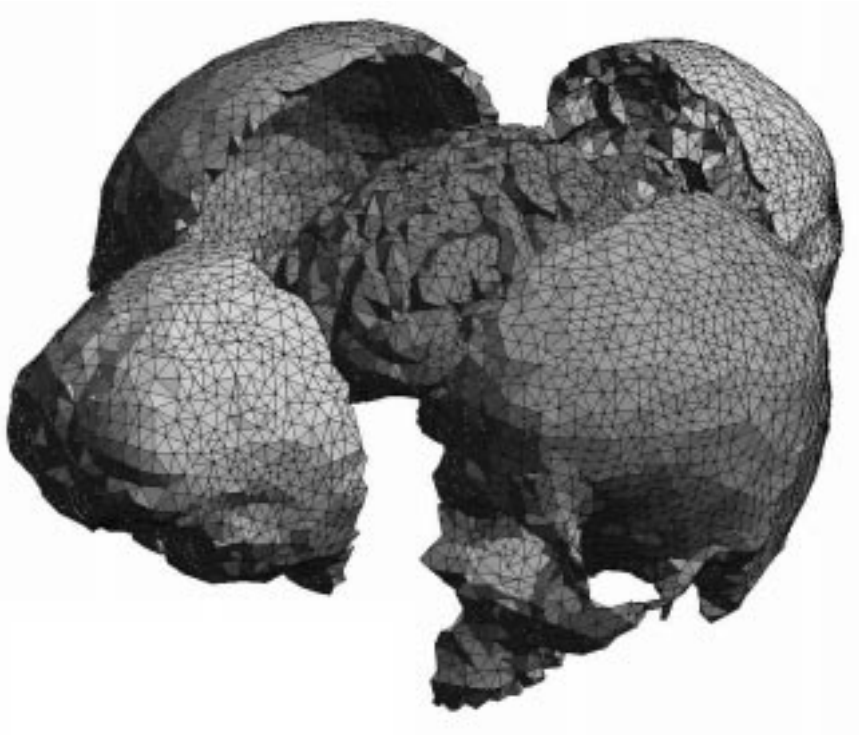
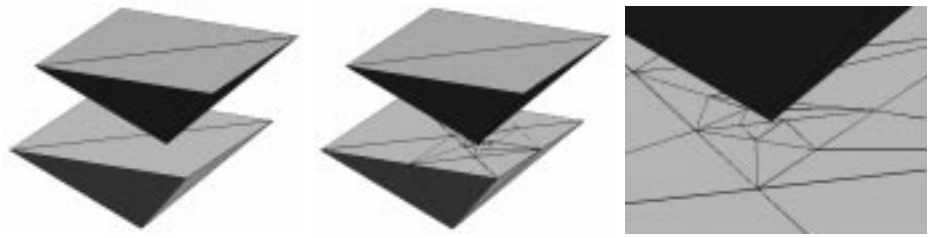


Figure 12: Mesh of cranium, cerebrospinal fluid and brain.

9.8 Mesh of Pyramids

One of the open problems is the limited ability of the algorithm to handle surfaces which are almost touching (this is actually common to all unconstrained Delaunay meshers). Consider two bodies very closely spaced. In applications to mechanics, unless the bodies touch there is no reason to take their proximity into account by refining the mesh. However, Delaunay mesh constructed for the ensemble of the two bodies will refine the neighborhood of the "almost in contact" region as shown in Figure 13, which depicts two pyramids arbitrarily offset so that the vertex of one is very close to the base of the other

(base dimensions 1×1 unit, height 1 unit, offset of the tip of the second pyramid with respect to the center of the bottom one $\{0.1454, 0.3569, 0.0019\}$). The *DELAUNAYZATION* algorithm enhanced the input mesh consisting of 10 points and 12 facets by additional 14 points and 28 facets. If the two bodies were meshed separately, this problem would not arise, but frequently bodies in self-contact or solids with cracks lead to these situations. One possible way of attack is constrained Delaunay triangulation. While constrained Delaunay triangulations are in general known not to exist in 3-D, Shewchuk demonstrates existence of conditions under which so-called conforming constrained Delaunay triangulation may be constructed [32].



(a) Input mesh (b) Output mesh (c) Detail

Figure 13: Mesh of two almost touching bodies.

10 Conclusions

We present an approach to the generation of tetrahedrizations in general three-dimensional volumes bounded by triangulated surfaces. Our method modifies the surface triangulations so that the constraints appear as collections of tetrahedron faces once the volume mesh is produced with unconstrained Delaunay triangulation of the (enhanced) vertex set. We use the results of Rajan [20] to re-formulate the construction of a Delaunay triangulation as a linear programming problem to yield a method of checking whether a given constraining surface facet is represented by a tetrahedron face in the unconstrained Delaunay triangulation of the current vertex set. If that is not the case, the surface triangulation is modified using a heuristic with two primitive operations, flip of diagonal in a pair of constraining facets, and split of edge common to two or more facets.

Need for further research is identified in several areas. Firstly, a *proof of termination* of the surface modification scheme would be reassuring. Secondly, a *parallel implementation* of the algorithm with domain decomposition and message passing is the subject of on-going research. Thirdly, *closely-spaced surfaces or vertices* placed near each other can cause the algorithm to generate an excessive number of Steiner points. The reason lies in the properties of the unconstrained Delaunay triangulation itself. Finally, in some applications it is desirable to be able to *exactly preserve the constraint surface triangulations*, for example when selectively re-triangulating some limited part of an existing triangulation. An algorithm enhancing the vertex set so that selected constraining facets are preserved in the unconstrained Delaunay triangulation would be very useful.

Acknowledgements

We are grateful for support from the Department of Energy through Caltech's ASCI Center of Excellence for Simulating Dynamic Response of Materials.

References

[1] M. A. Yerry and M. S. Shephard. Three-dimensional mesh generation by modified octree technique. *International Journal for Numerical Methods in Engineering*, 20:1965–1990, 1984.

[2] M. S. Shephard and M. K. Georges. Three-dimensional mesh generation by finite octree technique. *International Journal for Numerical Methods in Engineering*, 32:709–749, 1991.

[3] M. Saxena, P. M. Finnigan, C. M. Graichen, A. F. Hathaway, and V. N. Parthasarathy. Octree-based automatic mesh generation for non-manifold domains. *Engineering with Computers*, 11:1–14, 1995.

[4] Stephen A. Vavasis. Qmg web site. simon.cs.cornell.edu/Info/People/vavasis/qmg-home.html, 1998.

[5] R. Löhner, P. Parikh, and C. Gumbert. Interactive generation of unstructured grid for three dimensional problems. In *Numerical Grid Gen-*

eration in Computational Fluid Mechanics '88, pages 687–697. Pineridge Press, 1988.

[6] R. Löhner. Progress in grid generation via the advancing front technique. *Engineering with Computers*, 12:186–210, 1996.

[7] S. H. Lo. Volume discretization into tetrahedra-I. Verification and orientation of boundary surfaces. *Computers and Structures*, 39(5):493–500, 1991.

[8] S. H. Lo. Volume discretization into tetrahedra-II. 3D triangulation by advancing front approach. *Computers and Structures*, 39(5):501–511, 1991.

[9] E. Seveno. *Génération automatique de maillages tridimensionnels isotropes par une méthode frontale*. PhD thesis, Université Pierre et Marie Curie - Paris VI, 1998. Available as report TU-0521 from INRIA.

[10] C. Hazlewood. Approximating constrained tetrahedrizations. *Computer Aided Geometric Design*, 10:67–87, 1993.

[11] N. P. Weatherill and O. Hassan. Efficient three-dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints. *International Journal for Numerical Methods in Engineering*, 37:2005–2039, 1994.

[12] N. P. Weatherill. The reconstruction of boundary contours and surfaces in arbitrary unstructured triangular and tetrahedral grids. *Engineering Computations*, 13(8):66–81, 1996.

[13] P. L. George, F. Hecht, and E. Saltel. Automatic mesh generator with specified boundary. *Computer Methods in Applied Mechanics and Engineering*, 92:269–288, 1991.

[14] M. S. Shephard and M. K. Georges. Reliability of automatic 3D mesh generation. *Computer Methods in Applied Mechanics and Engineering*, 101(1–3):443–462, 1992.

[15] F. Hermeline. Triangulation automatique d'un polyèdre en dimension N. *RAIRO Analyse Numérique*, 16(3):211–242, 1982.

[16] J-D. Boissonnat. Geometric structures for three-dimensional shape representation. *ACM Transactions on Graphics*, 3(4):266–286, 1984.

[17] Gary L. Miller, Dafna Talmor, Shang-Hua Teng, Noel Walkington, and Han Wang. Control volume meshes using sphere packing: Generation, refinement and coarsening. In *5th International Meshing Roundtable, Sandia National Laboratories*, pages 47–62, 1996.

[18] P. Fleischmann and S. Selberherr. Three-dimensional Delaunay mesh generation using a modified advancing front approach. In *Proceedings of the 6th International Meshing Roundtable*, pages 267–278. Sandia National Laboratories, 1997.

[19] J. R. Shewchuk. *Delaunay Refinement Mesh Generation*. PhD thesis, Carnegie Mellon University, May 1997.

[20] V. T. Rajan. Optimality of the Delaunay triangulation in R^d. *Discrete Comput. Geom.*, 12:189, 202 1994.

[21] R. Radovitzky and M. Ortiz. Tetrahedral mesh generation based on node insertion incrystal lattice arrangements and advancing-front-Delaunay triangulation. *Computer Methods in Applied Mechanics and Engineering*, 1998. In press.

[22] A. Fuchs. Automatic grid generation with almost regular Delaunay tetrahedra. In *Proceedings of the 7th International Meshing Roundtable '98*, volume Sandia report SAND 98-2250, pages 133–147. Sandia, 1998.

[23] B. Barber. *Qhull manual*. The Geometry Center, Minneapolis MN, www.geom.umn.edu/software/qhull, 1998.

[24] N. Amenta and M. Bern. Surface reconstruction by Voronoi filtering. *Discrete and Computational Geometry*, 1998. submitted.

[25] G. M. Turkiyyah, D. W. Storti, M. Ganter, H. Chen, and M. Vimawala. An accelerated triangulation method for computing the skeletons of free-form solid models. *Computer-Aided Design*, 29(1):5–19, 1997.

[26] M. Tanemura, T. Ogawa, and N. Ogita. A new algorithm for three-dimensional Voronoi tesselation. *J. of Computational Physics*, 51(2):191–207, 1983.

[27] M. L. Merriam. An efficient advancing-front algorithm for Delaunay triangulation. *AIAA*, paper 91-0792, 1991.

[28] D. J. Mavriplis. An advancing-front Delaunay algorithm designed for robustness. *AIAA*, paper 93-0671, 1993.

[29] D. H. McLain. Two-dimensional interpolation from random data. *Comput. J.*, 19:178–181, 1976.

[30] R. Löhner. Progress in grid generation via the advancing front technique. *Engineering with Computers*, 12:186–210, 1996.

[31] P. Cignoni, C. Montani, R. Perego, and R. Scopigno. Parallel 3D Delaunay triangulation. In R. J. Hubbold and R. Juan, editors, *EUROGRAPHICS '93*, volume 12, pages C–129. Eurographics Association, Blackwell Publishers, 1993.

[32] J. R. Shewchuk. A condition guaranteeing the existence of higher-dimensional constrained Delaunay triangulations. In *Proc. 14th Annual Symposium on Computational Geometry*, 1998.