# THREE-DIMENSIONAL CONSTRAINED DELAUNAY TRIANGULATION: A MINIMALIST APPROACH.

Paulo Roma Cavalcanti[1], Ulisses T. Mello [2]

[1] Federal University of Rio de Janeiro, Rio de Janeiro, RJ, Brazil. (roma@watson.ibm.com)
Present Address: IBM T.J. Watson Research Center
[2]IBM T.J. Watson Research Center, Yorktown Heights, NY, U.S.A. (ulisses@watson.ibm.com)

**ABSTRACT**

In this paper we summarize our experiences with $3D$ constrained Delaunay triangulation algorithms for industrial applications. In addition, we report a robust implementation process for constructing $3D$ constrained triangulations from initial unconstrained triangulations, based on a minimalist approach, in which we minimize the use of geometrical operations such as intersections. This is achieved by inserting Steiner points on missing constraining edges and faces in the initial unconstrained triangulations. This approach allowed the generation of tetrahedral meshes for arbitrarily complex $3D$ domains.

**Keywords: Delaunay triangulation, constrained triangulations, three-dimensional mesh generation, industrial strength triangulations, tetrahedralization, computational geometry.**

## 1 INTRODUCTION

The automatic generation of unstructured meshes for three-dimensional ($3D$) objects is essential for many areas of applied sciences, such as Finite Element Analysis (FEA), Computer Graphics, Biomechanics, and etc. Automatic mesh generation is needed for productivity reasons; it takes an extensive amount of time and effort to generate meshes by any technique that is not fully automated [1].

A general problem for automatic mesh generation is to create a mesh that represents $3D$ objects bounded by polygonal faces and possibly with interior constraining faces and edges. Algorithms for mesh generation have been a research topic over the last two decades. Basically, three main families of algorithms have been described in the literature: octree methods [2, 1]; Delaunay-based methods [3, 4, 5, 6]; and advancing front methods [7, 8, 9, 10]. A good survey of these methods can be found in Owen [11].

Unlike the $2D$ case, $3D$ Delaunay triangulations do not have the property of maximizing the minimum angle. Nonetheless, they are very attractive from a robustness point of view due to the simplicity of the Delaunay criteria and because they require only two predicates. In addition, the quality of Delaunay triangulations can be improved by various local transformations [12].

The generation of unconstrained triangulations is trivial [3, 13]. However, the application of the aforementioned algorithms to generate $3D$ triangulations subject to constraining external and internal boundaries is complex and, in general implementations of these algorithms, have robustness problems when applied to industrial applications. This occurs due to the finite precision of the representation of floating point numbers and the associated round-off errors, which may cause in some cases an implementation to abort, get trapped in infinite loops or produce invalid meshes.

For industrial purposes, a mesh generator should be capable of handling thousands of polygons describing the constraining boundaries, and it is expected to produce a valid mesh in a reasonable wall-clock time. Furthermore, there should not be restrictions on the positioning of points defining polygons other than their topological consistency. In this study, we discuss a process that:

1) uses known techniques to produce Delaunay-based meshes that honor the geometry of the boundary of the object;

2) produces valid meshes; and

3) requires only the topological consistency of the input polygons.

A prerequisite for the automatic mesh generation is the robustness of the algorithms, which is directly dependent on the type of geometric operations necessary to generate the mesh. In the literature, two methods heavily dependent on intersections have been reported to generate constrained Delaunay triangulations [14, 15].

In our implementation, intersections are used only to recover missing constraining faces which were not recovered by an heuristic that we have created for inserting Steiner points. Generally, we obtain conforming meshes without calculating any intersection and using only two predicates.

Although there are various techniques for mesh improvement (e.g.,[16, 17]), an initial mesh is necessary to apply any of these techniques. Here, we will focus our discussion on the generation of the initial mesh, but we will not discuss refinement or mesh improvement techniques.

This paper summarizes our approach to generate a constrained Delaunay Triangulation in the following four steps:

1) generate unconstrained Delaunay triangulation using the object pointset (section 2.1).

2) recover missing constraining edges (section 3.1).

3) recover missing constraining faces (section 3.2).

4) perform local triangulation of tetrahedra if there still are missing faces (section 3.3).

## 2 DELAUNAY TRIANGULATIONS

In $n-$dimensions, a triangulation of a set of points $V$ is a set of non-intersecting simplices $T(V)$, whose vertices form $V$, and whose union completely fills the convex hull of $V$. Simplices of dimension 3 are called tetrahedra, whose faces are simplices of dimension 2 (triangles), whose edges are simplices of dimension 1 (segment lines), and whose vertices are simplices of dimension 0 (points).

There are many ways to triangulate the set $V$, and the Delaunay triangulation is the one in which, for any simplex of $T(V)$, there is an empty $n-$dimensional sphere that passes through its vertices. A sphere is said to be empty if there is no vertex in its interior. A Delaunay triangulation is unique if and only if there are no $(n+2)$ co-spherical vertices in the set $V$.

A $3D$ constrained Delaunay triangulation of a set of planar simple polygons $F$ is the triangulation in which $V$ is the set of vertices of $F$ and any polygon is obtained as the union of faces of $T(V)$.

Delaunay triangulations can be constructed by reducing all geometric operations to only two predicates:

$P_1$- point-in-sphere testing.

$P_2$- locating a point in relation to a plane and its two associated semi-spaces.

This reduction is an important step to minimize the errors associated with the computer's finite precision, which affects directly the robustness of the mesh generation process. A detailed discussion of a robust adaptive implementation, based on exact arithmetic of these predicates, can be found in [6].

The traditional unconstrained Delaunay triangulation algorithm uses only pointsets, and thus there is no guarantee that the boundary faces will be present on the triangulation of its convex hull. This occurs if there is a set of faces that do not appear in any triangulation of the input points [14]. However, an unconstrained mesh generator can be used as the starting point for the implementation of a constrained mesh generator. This is normally achieved by either adding extra points (Steiner points) to the triangulation on missing faces, or in some cases, by flipping faces of the tetrahedra of the starting mesh. Our process relies on Steiner points tied to a simple way of choosing their positions.

Two-dimensional constrained Delaunay triangulation

is considered a closed problem [18], but in $3D$ there are still many issues to be solved. Although it is always possible to produce a constrained triangulation of a general $2D$ domain without Steiner points, this is not true in $3D$. This is the main factor that precludes the simple generalization of $2D$ algorithms to $3D$.

### 2.1 Computing Delaunay Triangulations

Watson [3] and Bowyer [13] described the first incremental algorithms to compute $n-$dimensional Delaunay triangulations. These algorithms are based on the fact that choosing the $(n+1)$ points that lie on $n-$dimensional empty spheres, for defining $n-$dimensional simplices, guarantees the appearance of Voronoi vertices (circumcenters of the spheres) in the dual Voronoi tesselation. Therefore, these $(n+1)$ points form a minimal cluster of data that are immediately adjacent.

Watson's incremental algorithm adds one point at a time to the triangulation, which initially has a single $n-$dimensional simplex large enough to envelop all points in the set $V$. His algorithm is generically known as the *cavity* algorithm, since it deletes all $n-$dimensional simplices that are no longer empty after the insertion of the new point, forming a polyhedral cavity enveloping the newly inserted point.

The cavity is then triangulated by connecting the newly inserted point to all vertices on the cavity boundary. To prevent structural inconsistencies, the cavity must be star-shaped [19], satisfying the condition of point convexity. This condition is easily added to the cavity algorithm by enforcing:

$$n_i.(p - x_i) > 0.0, \tag{1}$$

where $n_i$ is the normal to the $i^{th}$ face on the cavity, $x_i$ is any point on the $i^{th}$ face, and $p$ is the position of the newly inserted point. When a face on the cavity is not strictly point convex, the simplex to which it is connected is deleted, forming a larger cavity.

One can easily verify by looking at the corresponding Voronoi diagram that no vertex is disconnected during this process. A word of caution: we have noticed that if tolerances are used on the right hand side of inequality (1) instead of 0.0, the produced cavity can be incorrect. The deletion of simplices may disconnect a set of vertices, or even a part of the triangulation, in the case that a newly inserted point is located too close to an existing vertex or group of vertices. On the other hand, comparisons with floating point numbers generally need tolerances. The best compromise is obtained by using the predicate $P_2$.

An alternative approach for the cavity algorithm is the $3D$ *flip* algorithm created by Barry Joe [4, 5], which generalizes to $3D$ the well known $2D$ flip algorithm of Lawson [20]. The flip algorithm is very elegant and has the advantage of keeping a triangulation at all times. However, it is more complex to implement. The $3D$ flip algorithm is based on the fact that there are only two ways for triangulating a five point convex hexahedron: one with 2 tetrahedra and the other with 3 tetrahedra. One of the two possible triangulations must be the Delaunay triangulation of the five points. Therefore, two local transformations switch between these two triangulations (from 2 to 3 tetrahedra and vice-versa), and a third local transformation is used when four out of the five points are coplanar (a 4_4 flip).

In the flip algorithm, the tetrahedron that contains the newly inserted point $v$ is retriangulated by connecting its vertices to $v$. If $v$ falls on a face or on an edge, the procedure is basically the same, but the triangulation is more complex. Then, a list $lF$ of all faces that defines a tetrahedron when connected to $v$ is constructed. These faces are the initial candidates to be flipped. The point-in-sphere testing is performed using the sphere defined by the three points of a face $f$ in $lF$ plus the fourth vertex belonging to the tetrahedron at the opposite side of $f$ from $v$. If the sphere contains $v$, the face $f$ is considered for flipping.

The preconditions to use the three different types of flips are the following:

- 3_2 flip: there must be an edge incident to $v$ with exactly three incident faces.
- 2_3 flip: the two tetrahedra sharing $f$ (one of them incident to $v$) must define a convex set.
- 4_4 flip: $v$ has to be on the same plane of $f$ and there has to be four faces incident to the visible edge of $f$ from $v$ (the edge that is going to be flipped).

After a flip, two to four new faces are added to $lF$, and the process is repeated until $lF$ is empty. Note that the 3_2 flip is the only one that does not require a geometric test, and therefore should be tried first.

The sequence of flips is completely arbitrary, but sometimes the preconditions for a flip are not satisfied, and

the face is considered non-flippable. Joe [4] describes in detail the circumstances in which every non-locally optimal interior face in the triangulation is not transformable (there is a connected NLONT-cycle). In these circumstances, the $3D$ flip algorithm produces a pseudo-locally optimal Delaunay triangulation. In our opinion, this is the major flaw of the algorithm.

Because there are many common tasks shared by $2D$ and $3D$ algorithms, we designed a $C^{++}$ base class from which we derive the classes responsible for the $2D$ and $3D$ triangulations. From the $3D$ triangulation class we further derive two classes to perform triangulations based on the flipping and the cavity algorithms.

### 2.2 Point Location

In an incremental algorithm, an accurate localization of a point in the triangulation is a critical factor to obtain a valid mesh. Although mathematically simple, a robust implementation of point location is not a simple task. To avoid a naive implementation that tests every simplex for point containment, the adjacency information available in the triangulation should be used to guide the point location through the mesh. Although the use of baricentric coordinates is the simplest way of performing a point-in-simplex testing, we use predicate $P_2$ in order to enhance the robustness of point location query.

When input points are too closely clustered very small tetrahedra are generated, and they may cause numerical problems for the point location routines. We have adopted a useful procedure that minimizes this problem by delaying the insertion of points that are closer than a predefined distance to the previously inserted points. These clustered points are only inserted in the triangulation at the end of the insertion process.

## 3 CONSTRAINED TRIANGULATIONS

Delaunay-based algorithms described on the previous section are used to triangulate the convex hull of a set of input points without any constraint. When a set of polygonal faces defining the external and internal boundaries of a model has to be part of the final triangulation, the missing edges and polygonal faces of the bounding surface must be inserted into the triangulation.

We generate these constrained triangulations in two steps. First, all constraining edges are recovered, and then the constraining faces are recovered sequentially. Since the recovery of a missing face may cause the removal of a previously recovered edge, a new edge recovery phase is then executed.

A necessary step in generating a constrained triangulation is finding the constraining edges and faces that are missing in the unconstrained triangulation. For this purpose, we maintain a list of pointers linking the triangulation vertices to the points defining the constraining polygons.

### 3.1 Edge Recovery

The process of recovering missing edges that we have used is called stitching [21, 15], and in this process a vertex is inserted in the triangulation at the midpoint of a missing edge. This recovery process exploits a property of Delaunay triangulations that each vertex in the triangulation is connected to its closest vertex by an edge. The midpoint insertion process is recursively repeated to each remaining missing half segment of an edge until the edge is recovered as a collection of triangulation edges.

Every vertex inserted in the triangulation must also be inserted on the corresponding bounding edge of the polygon. However, because a newly inserted vertex may cause the deletion of a previously present constrained edge in the triangulation, a list of deleted edges must be kept. This process is repeatedly applied until this list is empty. Unfortunately, this process does not always converge. Two constraining edges with different lengths, incident to a high degree vertex, and bearing an angle smaller than $60^{\circ}$, may lead to a mutually destructive and diminishing spiral, since there may be an interaction among all constraining edges incident to the same vertex.

Ruppert has cleverly solved this convergence problem in $2D$ by creating the concept of protecting circles around vertices [22]. We have extended this concept to $3D$ using protecting spheres that are centered at vertices to which constraining edges are incident. When these incident edges are divided at the intersection with the protecting sphere, no part of any edge remains missing or is removed, since all intersecting points are cospherical.

To recover a missing edge, Ruppert [22] describes the following edge subdivision procedure: the first subdivision is at its midpoint, while the second subdivision is at the closest power-of-two distance from the original vertex. All other subdivisions are at edge mid-

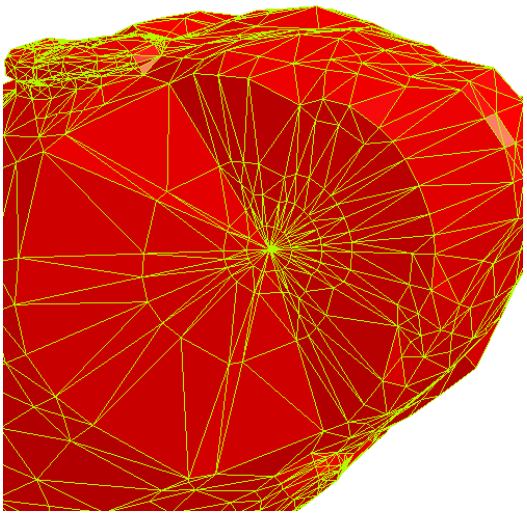

Figure 1: Triangulation of a head (base view). Note the result of the application of a protecting sphere around the central vertex.

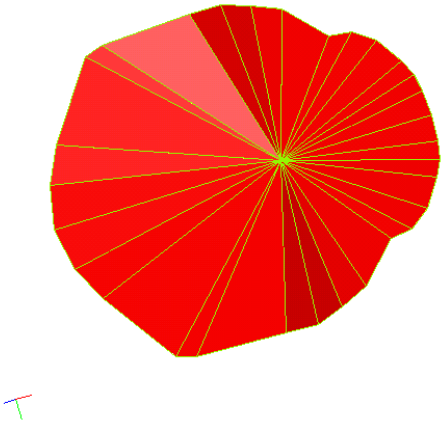

Figure 2: Constraining edges defining the base of the head shown in Figure 1. Note the high degree central vertex with 30 incident edges bearing small angles.

points and, consequently, will also be at a power-of-two distances.

This process works for any arbitrarily small, constraining, missing edge because after a few subdivisions, every vertex inserted to recover a piece of a constraining edge will be at the same power-of-two distance from the original vertex. In Figure 1, we display the result of this process applied to the $3D$ triangulation of a human head. The base of the head is not planar (Figure 2) and possesses a high degree vertex, in the center, with constraining incident edges bearing small angles. The protecting sphere around the central vertex eliminates the non-convergent destructive process during the recovery of its incident constraining edges.

### 3.2 Face Recovery

The recovery of constraining faces is much more complicated than the process of edge recovery. The first difficulty occurs in the verification of the presence of a constraining polygon in the triangulation. Even if all the edges bounding the polygon are in the triangulation, this does not imply that the corresponding polygonal face is also in the triangulation. In addition, a constraining polygon may be present in the triangulation as a collection of adjacent triangles.

In a Delaunay triangulation, if a planar polygon is present as a collection of triangular faces, these faces must form a $2D$ Delaunay triangulation over the polygon [23]. We use this property to verify the presence of a constraining face in a $3D$ triangulation, $T(V)$.

The verification process constructs a draft $2D$ triangulation of all vertices bounding a polygon $P$. This draft triangulation partitions $P$ into a set of simplices, $\Delta(P) = \{\delta_i(p_a, p_b, p_c)\}$. To simplify and speed up the generation of $\Delta(P)$, we pre-triangulate all non-convex constraining polygons.

Using $\Delta(P)$, we check if $\forall \delta_i(p_a, p_b, p_c) \in \Delta(P)$, $\exists \gamma_i(p_a, p_b, p_c) \in T(V)$. If any $\delta_i$ is missing, $P$ is recovered by inserting Steiner points, $SP$, with an heuristic described later in this section, in such a way that: $P = \cup \delta_i$, $\delta_i \equiv \gamma_i \in T(V^*)$, and $V^* = V \cup SP$.

If there are four or more co-circular points on $P$, the verification process may fail because $\Delta(P)$ is not unique. It is worthy to note that this verification process only works if $T(V)$ is strictly Delaunay. To solve this problem we make a copy of the polygon that is partitioned by the edges in $T(V)$ connecting pairs of its bounding vertices. We then verify which partition pieces on the polygon copy are triangular and correspond to a face in $T(V)$. Using this approach, it is possible to detect the presence of triangular pieces even when the points defining the polygon are not in general position.

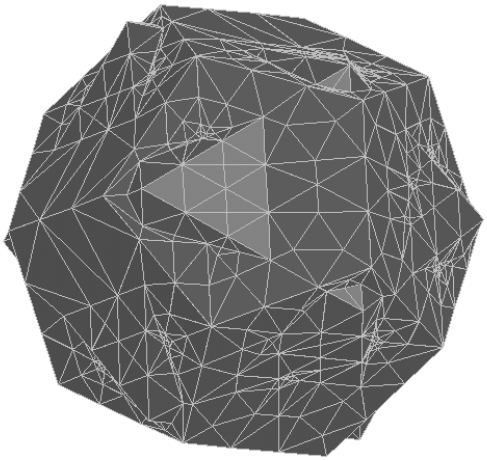

Figure 3: Resulting triangulation of the union of a dodecahedron and an icosahedron after refinement and using Steiner points to recover constraining boundary faces.

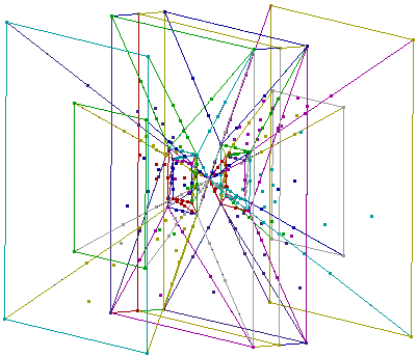

Figure 4: Distribution of Steiner points on a model with small dihedral angles.

Our heuristic for inserting Steiner points on the faces is simple: for each missing triangular piece $\delta_i$, we find if any of its bounding edges is missing. In the case that there are missing edges, one of them is chosen and a vertex is inserted at its midpoint. When all three edges are present but $\delta_i$ is not, a vertex is inserted at the midpoint of all three edges, as displayed in Figure 3. In this figure, we show a triangulated model resulting from the union of a dodecahedron and an icosahedron.

For each missing $\delta_i$, one or three Steiner points may be inserted, and $\Delta(P)$ is rebuilt. This process is repeated until $\Delta(P) \subset T(V^*)$. Because the insertion of new points may cause the removal of previously recovered faces, we maintain a list of constraining faces to be reinserted.

This heuristic facilitates significantly the process of creating Steiner points on the edges of missing subfaces $\delta_i$, by using essentially the same process described earlier for edge recovery in section 3.1.

In Figure 4, we show the Steiner points inserted to recover all the polygonal faces of a model composed by four pyramids touching upon their apices and an intersecting block.

Unfortunately, the process of face recovery does not always converge. The reason for the non-convergence is the Delaunay criteria, which is completely insensitive to anything but vertices, and a cycle of recovering faces coupled with simultaneous destruction of previously inserted faces may never converge.

The non-convergence is a function of the dihedral angle between input polygonal faces, and if all angles are greater or equal $90^\circ$ the convergence holds [6]. In practice, this is a very restrictive limitation and unacceptable for several applications.

In our implementation, we mark a face as non-recovered when either the precision of the machine is reached or a predefined number of Steiner points have been inserted on a single face.

### 3.3 Handling Non-recovered Faces

When the process of face recovering described in the previous section fails, we are forced to abandon the global Delaunay criteria and accept triangulations that are not strictly Delaunay. Hazlewood [14] has shown that it is possible to produce constrained triangulations by just retriangulating the tetrahedra whose interiors are intersected by a missing, constraining, convex polygon. First, all points on the intersection of the missing polygon and the edges and faces of tetrahedra are found. Then, every intersected tetrahedron is retriangulated locally using the Delaunay criteria. The vertex set on each side of the polygon plane is triangulated separately and there should be no visibility across this plane during the triangulation.

Note that the local retriangulation can be obtained by other criteria than Delaunay. If constrained edges are already part of the triangulation, the local retriangulation can be easily obtained by using local operators as described in [15]. These operators basically treat the cases that occur when the polygon completely intersects a tetrahedron, and partitions it into a set of new tetrahedra, and/or pyramids with quadrilateral bases, and/or prisms with two triangular and three quadrilateral faces.

Since each tetrahedron is processed separately, there is no guarantee that all intersected tetrahedra will be retriangulated consistently by these operators. This is because a prism resulting from an intersection may have all its quadrilateral faces already triangulated (by preprocessed adjacent tetrahedra) in such a way that the prism cannot be triangulated without inserting a point in its interior.

## 4 SIMPLEX CLASSIFICATION

After recovering all faces, we obtain a triangulation of the convex hull containing the constrained Delaunay triangulation of the model. Hence, the triangulation of the convex hull needs to be carved out by removing the tetrahedra outside the model, and its simplex attributes must be set according to the properties of the model. We have implemented an efficient classification scheme that takes advantage of the adjacency information to minimize containment tests.

In our applications we have to mesh multi-domain models that are composed of several regions, each one with a distinct property and composition (e.g.,[24, 25]). For completeness, we also define an external region that represents the region of the space outside the model. After a model is triangulated and refined, each simplex of the mesh is classified in order to assign properties to the elements in each region of the model and to specify the simplices corresponding to surfaces defining boundary conditions, as required by FEA.

During the edge and face recovery phases, we mark the simplices that correspond to vertices, edges and faces of the original model. These marked simplices define the internal and external boundaries of the model in the $3D$ triangulation. To classify the simplices, we make a point-in-region testing just to a single tetrahedron and determine the region where it belongs. Then, we use this tetrahedron as a seed for the traversing. Using the adjacency information, we visit and set the containment attribute of all tetrahedra that can be reached from the seed without crossing the region boundary. Simplices classified as being in the external region are simply deleted, eliminating the tetrahedra outside the model.

Finally, the attributes of each vertex, edge and face of the model are assigned to the corresponding set of simplices in the triangulation.

## 5 EXAMPLES

We have selected four examples with complex shapes (Figures 5, 6, 9, and 10) to discuss important aspects involved in the triangulation process, as well as to provide a general assessment of the number of simplices generated during the triangulation. In all these examples, the constraining faces were recovered without any intersection, and the resulting triangulations are strictly Delaunay.

The first example is the triangulation of a typical mechanical part with relatively large dihedral angles (Figure 5). In the initial unconstrained mesh, there were 24 missing edges out of a total of 3054 edges, and 3 missing faces out of 1622 faces. From these, a total of 26 edges, and 3 faces had to be reinserted. The $3D$ unconstrained triangulation of the convex hull had 8101 tetrahedra whereas the final constrained $3D$ triangulation had practically half as many (4047 tetrahedra). The reduction in the number of tetrahedra occurred because the unconstrained mesh included all the tetrahedra in the convex hull.

The second example is the $3D$ triangulation of a model with 3 regions generated by a "genetic experiment" involving the union of a deer and a dolphin (Figure 6). It is important to realize the impact of the intersection of constraining input surfaces on the number of tetrahedra in the final triangulation. These intersections induce a significant increase in the number of tetrahedra in the final mesh. A good measure for this increase can be assessed by comparing the initial number of missing edges and faces in the initial unconstrained triangulation with its respective number in the constrained triangulation. Another relevant metric is the number of edges and faces that were removed and subsequently reinserted in the triangulation when a vertex was inserted into their equatorial sphere.

In Figure 7, for instance, we show the vertices on the curve at the intersection of the two animals. Note the high density of the vertex distribution along this curve

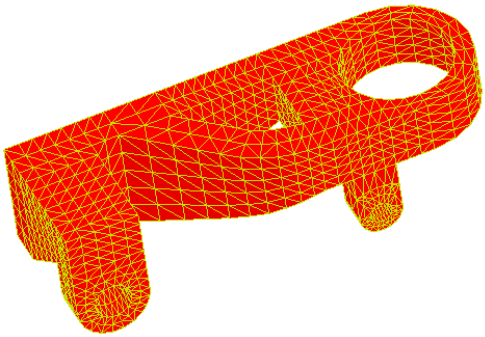

Figure 5: Triangulation of a mechanical part.

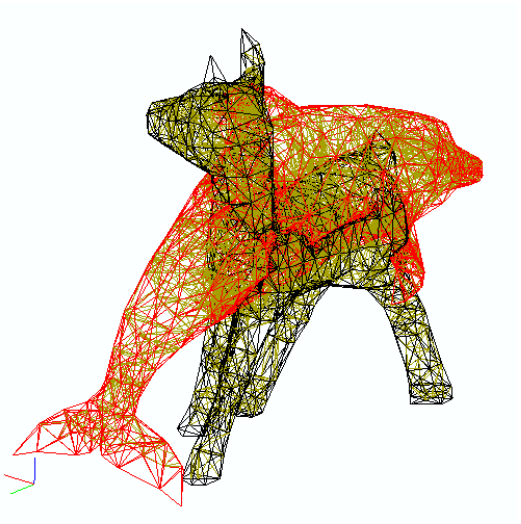

Figure 6: Triangulation of the union of two models representing a deer and a dolphin.

when compared with the average distance of vertices in the model. This poor vertex distribution provides a good robustness test.

In contrast to the first example, the final constrained $3D$ triangulation had 14944 whereas the initial unconstrainded triangulation of the convex hull was composed by only 5869 tetrahedra. The number of missing edges and faces in the initial unconstrained triangulation also increases substantially. There were 503 missing edges, and 24 missing faces out of a total of 2989 edges and 2052 faces, respectively. From these, 1473 edges, and 130 faces had to be reinserted.

The third example is the triangulation of a geological model with 6 regions generated by the intersection of surfaces representing a salt dome, three horizons, and a bounding box (Figure 8). Here, 818 edges and 20 faces were missing of a total of 6169 edges and 4259 faces, respectively. From these, a total of 5698 edges and 103 faces had to be reinserted. The initial unconstrained triangulation of the convex hull and the final constrained $3D$ triangulation had 11300 and 50935 tetrahedra, respectively.

In Figure 9, we show the $3D$ triangulation of the geological model with 31505 tetrahedra. This mesh was generated using an automatic point creation scheme to add points within the domain and onto the constraining faces. In spite of the creation of these points, no intersection was also necessary to obtain the mesh. Note the high concentration of small tetrahedra around the

Figure 7: Curve at the intersection of the models shown in Figure 6.

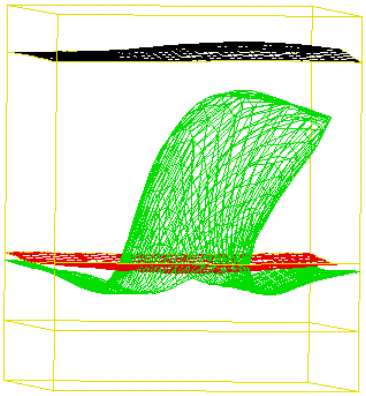

Figure 8: Input surfaces of a geological model.

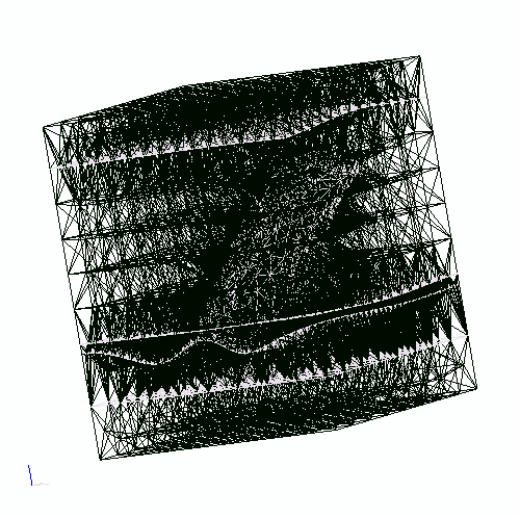

Figure 9: Triangulation of the geological model shown in Figure 8.

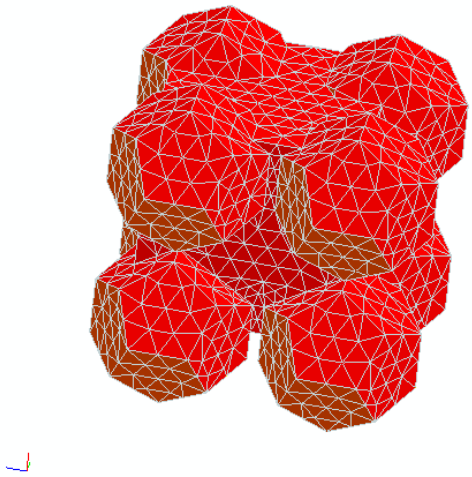

Figure 10: Triangulation of a porous sediment framework containing only 8 grains.

salt dome at the center.

The last example is a typical result of our approach after using a set of techniques to improve the mesh quality, such as automatic point creation and smart smoothing [17]. In Figure 10 we show the triangulation of a model with 9 regions representing the porous framework of a sedimentary rock in micro-scale with only 8 grains. The final triangulation had 21895 tetrahedra and 2398 points automatically inserted within the domain and onto the constrained faces.

The quality of this mesh can be assessed by the distribution of the minimum dihedral angles as shown in Figure 11. It is noteworthy that the majority of the tetrahedra has the minimum dihedral angle larger than $24.85^\circ$ $(89.194\%)$. The dihedral angles in the mesh are in the range $[2.208^\circ, 176.2^\circ]$ and the solid angles in the range $[0.1749^\circ,\ 203.6^\circ]$. Only 14 tetrahedra $(0.06394\%)$ have minimum dihedral angles smaller than $3.55^\circ$.

## 6 CONCLUSIONS

In this paper we summarize our experiences with the implementation of a robust $3D$ mesh generator using a minimalist approach to generate constrained Delaunay triangulations. This mesh generator has been a

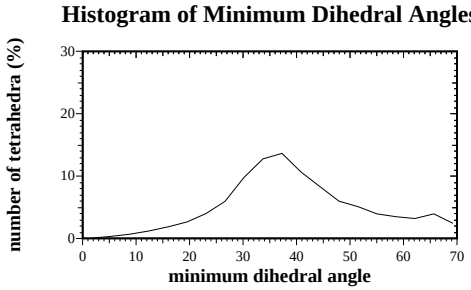


Figure 11: Histogram of minimum dihedral angles of the mesh shown in Figure 10.

valuable tool for us to generate meshes to geological and engineering applications using the Finite Element Method.

In spite of some robustness issues, the generation of unconstrained $3D$ Delaunay triangulations is straightforward. However, the generation of constrained $3D$ Delaunay triangulations requires a considerable effort.

In Geoscience it is common to create multi-domain earth models by the intersection of a set of input polygonal surfaces. These intersection operations may generate vertices closely clustered. For instance, we have built geological models in which the distance between vertices is in the range $[10^{-7}, 10^{4}]$. Despite this poor vertex distribution, we have been successful in triangulating these models.

Based on our experiences, both the cavity and flip algorithms are equivalent in terms of robustness. However, we tend to use the cavity algorithm more frequently because it is slightly faster and always produces a strict Delaunay triangulation, which is important for checking the presence of constrained faces in the triangulation. Flipping operations, nevertheless, are also useful for mesh improvement algorithms (e.g., [16]).

In our approach, we rely on a scheme for inserting Steiner points to recover missing faces. This scheme avoids geometrical operations involving intersections, and for most models it is sufficient to generate good quality meshes. The advantage of our implementation is that we constrain our operations to just two predicates most of the time. However, in some cases, when small dihedral angles between constraining faces are present, it may fail to converge. We circumvent this problem by using a local triangulation process when the recovery of a constraining face fails. These local triangulations require the computation of intersections, which it is not desirable because of its impact on the robustness of the entire process.

## Acknowledgement

We would like to thank Wendell Wiggins for his insights on tetrahedralization issues in geophysics and Mike Henderson and Tom Jackman for critically reviewing this manuscript. We would also like to thank Western Geophysical for partially funding this work.

## References

[1] Mark S. Shephard and Marcel K. Georges. Automatic three-dimensional mesh generation by the finite octree technique. *International Journal for Numerical Methods in Engineering*, 32:709–749, 1991.

[2] Will J. Schroeder and Mark S. Shephard. A combined octree/Delaunay method for fully automatic 3-d mesh generation. *International Journal for Numerical Methods in Engineering*, 29:37–55, 1990.

[3] David F. Watson. Computing the n-dimensional Delaunay tesselation with application to Voronoi polytopes. *The Computer Journal*, 24(2):167–172, 1981.

[4] Barry Joe. Three-dimensional triangulations from local transformations. *SIAM J. Sci. Stat. Comput.*, 10(4):718–741, July 1989.

[5] Barry Joe. Construction of three-dimensional Delaunay triangulations using local transformation. *Computer Aided Geometric Design*, 8:123–142, 1991.

[6] Jonathan Richard Shewchuk. *Delaunay Refinement Mesh Generation*. PhD thesis, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1997.

[7] Rainald Lohner. Generation of three-dimensional unstructured grids by the advancing front method. In *Proceedings of the 26th AIAA Aerospace Sciences Meeting*, Reno, Nevada, 1988.

[8] T.J. Barth and D.C. Jesperson. The design and application of upwind schemes on unstructured meshes. In *Proceedings of the 27th AIAA Aerospace Sciences Meeting*, Reno, Nevada, 1989.

[9] Dimitri J. Mavriplis. An advancing front Delaunay triangulation algorithm designed for robustness. Technical report 92-49, ICASE, October 1992.

[10] Rainald Lohner. Progress in grid generation via the advancing front technique. *Engineering with Computers*, 12:186–210, 1996.

[11] Steven Owen. A survey of unstructured mesh generation technology. In *Proceedings of the Seventh International Meshing Roundtable*, Dearborn, Michigan, October 1998. Sandia National Laboratories.

[12] Barry Joe. Delaunay versus max-min solid angle triangulations for three-dimensional mesh generation. *International Journal for Numerical Methods in Engineering*, 31:987–997, 1991.

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

[14] Carol Hazlewood. Approximating constrained tetrahedralizations. *Computer Aided Geometric Design*, 10:67–87, 1993.

[15] Nigel 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.

[16] Lori Freitag and Carl Olliver-Gooch. Tetrahedral mesh improvement using face swapping and smoothing. *International Journal for Numerical Methods in Engineering*, 40:3979–4002, 1997.

[17] Lori Freitag. On combining Laplacian and optimization-based mesh smoothing techniques. *Trends in Unstructured Mesh Generation, ASME Applied Mechanical Division*, 220:37–44, 1997.

[18] Joseph O'Rourke. *Computational Geometry in C*. Cambridge University Press, New York, 1994.

[19] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, 1985.

[20] C.L. Lawson. Transforming triangulations. *Discrete Math.*, 3:365–372, 1972.

[21] Nigel P. Weatherill. Delaunay triangulation in computational fluid dynamics. *Computers and Mathematics with Applications*, 24(5/6):129–150, September 1992.

[22] Jim Ruppert. *Results on Triangulation and High Quality Mesh Generation*. PhD thesis, Department of Computer Science, University of California at Berkeley, Berkeley, CA, 1992.

[23] Jean-Daniel Boissonnat. Shape reconstruction from planar cross sections. *Computer Vision, Graphics and Image Processing*, 44:1–29, 1988.

[24] Paulo Roma Cavalcanti, Paulo C.P. Carvalho, and Luiz F. Martha. Nonmanifold modeling: An approach based on spatial subdivisions. *Computer-Aided Design*, 29(3):209–220, March 1997.

[25] Ulisses T. Mello and Paulo Roma Cavalcanti. A topologically-based framework for simulating complex geological processes. Research report RC21339, IBM T.J. Watson Research Center, Yorktown Heights, New York, November 1998.