



ELSEVIER

Available online at www.sciencedirect.com

SCIENCE @ DIRECT®

Comput. Methods Appl. Mech. Engrg. 193 (2004) 2547–2563

**Computer methods
in applied
mechanics and
engineering**

www.elsevier.com/locate/cma

Boundary recovery for three dimensional conforming Delaunay triangulation [☆]

Qiang Du ^{a,b,*}, Desheng Wang ^b

^a *Department of Mathematics, Penn State University, 218 McAllister Building, University Park, PA 16802, USA*

^b *Lab for Scientific and Engineering Computing, Academy of Sciences, Beijing, China*

Received 1 May 2002; received in revised form 4 September 2003; accepted 18 December 2003

Abstract

We present an efficient boundary recovery method for conforming Delaunay triangulation of three dimensional domains. Our method combines local transformations like edges/faces swappings and modified constrained Delaunay refinement like edges/faces splittings to recover the missing boundary edges and faces. Its convergence is theoretically proved. Its effectiveness is also demonstrated through some meshing examples.

© 2004 Elsevier B.V. All rights reserved.

Keywords: Delaunay triangulation; Boundary recovery; Mesh generation

1. Introduction

Efficient and robust mesh generation is an essential part of many scientific and engineering computational challenges. In the field of unstructured meshing, methods such as Advancing-front, Octree and Delaunay [9–12,14–17] have been extensively studied in the last two decades. In particular, tremendous progress has been made on the theoretical analysis and robust implementation of Delaunay-based methods. Currently, several important issues on Delaunay based triangulations remain under investigation, one of which is on preserving the integrity of the boundary, or in another word, the boundary recovery.

The input data for a Delaunay-based algorithm are often given as a discrete description of the domain boundary which, in the three dimensional space, may represent a surface triangulation of the boundary. Delaunay-based methods usually produce a triangulation that forms the convex hull of the points on the boundary. For an arbitrary domain, such a triangulation may not match with the prescribed boundary surface, i.e., it may not satisfy the constraints (edges and faces in 3D) of the surface triangulation. A

[☆] Supported in part by China state major basic research project G199903280 and US NSF-CCR 9988303.

* Corresponding author. Address: Department of Mathematics, Penn State University, 218 McAllister Building, University Park, PA 16802, USA. Tel.: +1-814-865-3674; fax: +1-814-865-3735.

E-mail addresses: qdu@leibniz.math.psu.edu, qdu@math.psu.edu (Q. Du).

problem then arises, namely, how to recover the geometric constraints from the constructed triangulation of the boundary. This problem has been successfully resolved in two dimensional spaces [2,3]. In three dimensions, due to the Schrodert configuration [14,15], the existence of a constrained Delaunay triangulation that matches with a prescribed surface triangulation is not guaranteed. George et al. [14] and Weatherill [15] introduced ingenious techniques to reconstruct a prescribed surface triangulation. They recovered the surface edges and faces by a series of edge/face swaps (or flips), and occasional heuristic insertions of interior points (Steiner points). The heuristic procedures have not been proved to work in all cases, as several examples in Baida [13] provided evidences that the above methods may fail in certain situations. Schewchuk [16] and Shephard [17] proposed Delaunay refinement methods to construct a conforming triangulation to match with the surface geometry. Their methods use local mesh modifications such as edge/face splitting to recover a constraint as the concatenation of edges/faces. On one hand, the effectiveness of such kind of conforming boundary recovery has been demonstrated in many cases, even though no theoretical proof is provided for the convergence of these methods. On the other hand, the refinement steps may require the insertion of an excessive number of points to the missing constraints, hence violate the local sizing specification (prescribed by the surface triangulation or by other methods). Moreover, when inserting a point to a constraint in the Delaunay method, the recovered constraints may be deleted in the refinement process.

In this paper, an algorithm is presented which combines local transformations (edge/face swaps) with a modified Delaunay refinement (edge/face splits) to recover the boundary constraints in a conforming manner. As a main objective of this paper, a theoretical proof of the convergence of the method is provided.

The algorithm presented here involves two stages, with the first stage consisting of three basic local edge/face swaps that are applied to recover a portion of the missing constraints. These basic local transformations are not as complicated as those in [14]. For each missing constraint, comparisons are only made between the set of tetrahedra connecting the missing item and the required configurations of the three basic swaps. Whenever a match is found, a corresponding swapping is performed to recover the missing item. Numerical experiments show that these local transformations usually recover almost 80% of the missing constraints. In the second stage, the refinement method is used to recover the remaining missing items. Different from the previous works [16,17] where one first recovers edges and then faces, the missing faces are recovered sequentially and at the same time, their missing edges are also recovered. For each missing edge of a missing face, the intersection points of the edge with the current triangulation of the boundary points are found, and the nearest-mid intersection points are added one by one in a modified Delaunay insertion procedure until the recovery of the edge is achieved. When the missing edges of a missing face are all recovered, if the face is still missing, the intersection points of the face with the current triangulation are computed and they are inserted one by one into the modified Delaunay insertion procedure until the face is recovered. The modified Delaunay insertion shares the following property: when inserting a point to a constraint, no existing or recovered constraint is deleted.

Regarding our two-stage recovery algorithm, the use of local transformations in the first stage greatly reduces the need for points insertion in the second stage, thus the algorithm not only keeps the boundary recovery simple but also keeps the mesh in good conformation with the local sizing specification. The idea of keeping recovered constraints *non-violated*, that is, protected, was also addressed by Wright and Jack [19], but the protection method used there only becomes viable by recognizing a consistent node ordering of the faces of the *inserted polyhedra*, or Cavity [19]. The proposed method given in this paper via the modification of the Delaunay kernel is systematic and applicable for all cases. It can also be easily implemented. Moreover, since the number of missing constraints are finite, the convergence of the boundary recovery procedure becomes an easy consequence. The combination of local transformations and Delaunay refinement method can adequately address the various issues of conforming boundary recovery and also makes the recovery procedure very efficient and effective.

The remaining part of the paper is organized as follows: an overview of our method is described in Section 2 which covers the basic steps of the boundary recovery algorithm. Section 3 discusses the modified Delaunay insertion procedure. Our new recovery algorithm is explained in detail in Section 4. Section 5 presents the convergence proof of the recovery. Some application examples and preliminary comparisons with other recovery schemes are presented in Section 6. Finally, conclusion remarks are given in Section 7.

2. An overview of the method

Let S be the input surface triangulation of the boundary, which can be generated by various methods [4–6]. Let $P = \{\text{vertices of } S, \text{ i.e. the boundary points}\} \cup \{\text{eight vertices of the bounding box which covers the entire domain}\}$. Denote the Delaunay triangulation of P by T_d . Our goal is to modify T_d locally and to obtain a triangulation T such that each element of S either exists in T or is the union of a set of triangles of T , and the geometry of the boundary of T conforms with that of S . That is to say, constructing a conforming triangulation T of S is our objective. Our method combines local edges/faces swappings with edges/faces splittings. The basic steps of the method are briefly summarized as follows:

1. First, we collect information from S on the connecting edges and connecting faces of each boundary point in S . If an edge of S exists in T_d , this edge is marked *Recovered* and the corresponding edge in T_d is marked *Protected*, i.e. this edge has been recovered and must not be deleted in later mesh modifications. For the other edges in S , we mark them *Unrecovered* (or *Missing*). For the other remaining edges in T_d , we mark them *Unprotected*. For the faces of S , we perform a similar marking operation. The area of each face of S is computed and is to be used later.
2. For the missing items (edges/faces) of S , we use three basic types of single step swappings, namely T23, T32, T44 (as illustrated in Fig. 1). (1) T23 corresponds to trading $abcd$ and $abce$ with $deab$, $debc$, $deca$, where de intersects the interior of abc ; (2) T32 corresponds to the inverse of T23; (3) T44 corresponds to trading $abde$, $bcde$, $abfd$, $bcfd$ with $acde$, $abce$, $acdf$, $abcf$, where ac intersects the interior of bde or bdf . We do not apply the more complicated edge/face swapping procedure proposed in [14]. The above three single-step swappings are sufficient for our algorithm to recover a large part of missing items. T32 is for the recovery of a face, T23 for an edge, and T44 is for an edge and the simultaneous recovery of two

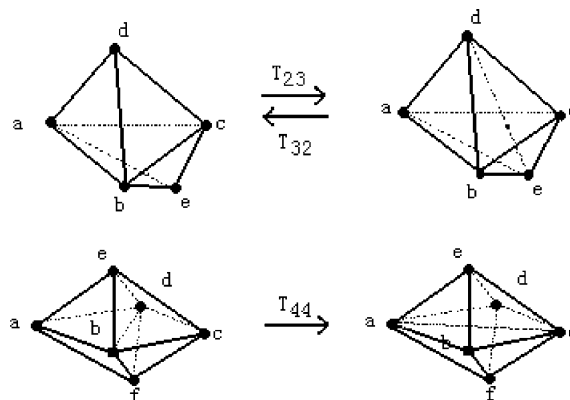


Fig. 1. Three basic types of swappings T23, T32 and T44.

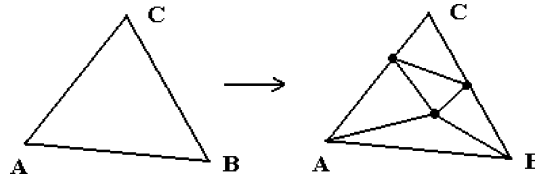


Fig. 2. Edge/face splits for recovering a face: a missing face and its two missing edges AC, BC (left), and the recovered face and its recovered edges (right).

faces. The local transformations we use here is aimed at recovering some missing edges/faces in a constrained manner while not violating the boundary sizing specifications. We denote the transformed T_d by T .

3. After the local transformations, some missing constraints may still remain to be recovered. We then apply the Delaunay refinement method that consists of the edges/faces splittings. The missing faces are recovered one by one, and the edges are simultaneously recovered (see Fig. 2). This leads to a different approach from that of [16,17]. For the recovery of each face, the procedure consists of two parts:

- (a) If a missing face has edges which are not recovered, we first recover these edges. Let A, B be the two end points of a missing edge. The set of intersection points $\{P_k\}$ of the line segment AB with T is computed. In $\{P_k\}$, the point P_m are chosen which is the nearest point to the midpoint of AB , and P_m is inserted into T via the modified Delaunay insertion procedure (discussed in Section 3). If the updated T has edges AP_m and P_mB , AB is recovered as a concatenation of the two edges. Else, if AP_m or P_mB is not in T , the nearest point in $\{P_k\}$ to its midpoint is again chosen and likewise is inserted into T . Recursively, we perform the insertions of points nearest to the midpoints until AB is reconstructed as a union of some edges in T . And in the process of the above insertion, when one edge in T is a part of AB , we mark it *Protected*. Also, after recovering AB , we mark it *Recovered*. At the same time, in the process of the above edge recovery, some faces (including the missing face) of S which connect AB may be recovered. They are also marked *Recovered*. Likewise, if some faces of T is a part of the missing face, we mark them *Protected*.
- (b) If the three edges of the missing face are recovered, but the face is still not recovered, a face splitting is performed. Similarly, the intersection points $\{Q_k\}$ of the missing face with T are computed. A point Q in $\{Q_k\}$ is found which does not intersect any recovered parts of the missing face and it is inserted into T through the modified Delaunay insertion. If the face is recovered, we move on to deal with other missing faces; else, recursively we find the another point Q (with the same meaning as above) and insert it into T until the missing face is re-established as a concatenation of faces in T . In the process of the above recursive insertion, if any face of T is a part of the missing face, it is marked *Protected* and when a missing face is recovered, it is marked *Recovered*.

4. Finally, the elements lying outside the domain are deleted from T and a conforming triangulation T of S is obtained, i.e. the boundary recovery process is completed.

3. Modified Delaunay insertion

Let $\{P_k\}$ be a finite set of points in R^d and for each k , the Voronoi cell of P_k is the set V_k defined by $V_k = \{p : \|p - P_k\| \leq \|p - P_j\|, j \neq k\}$. $\{V_k\}$ forms the well-known Voronoi (or Dirichlet) Tessellation of

the entire space with respect to the points set $\{P_k\}$. The Delaunay triangulation of $\{P_k\}$ is defined as the dual of the Voronoi Tessellation; see [1,16] for more details on the properties of Delaunay triangulation.

To construct a Delaunay triangulation of a set of points, the most effective approach is the incremental Delaunay insertion method introduced by Hermeline [18] and Watson [1], which has become very mature based on the works of Hecht and George [7,14] and others [2,6–8,12]. For inserting a new point into the current triangulation, the Delaunay kernel consists of three parts: *Base*, *Cavity*, *Ball*. Here, the *Base* is defined as the element(s) containing the new point; the *Cavity* is defined as the set of elements whose circumballs contain the new point; the *Ball* is defined as the elements which are formed by the new point and the boundary facets of the *Cavity*. With the *Cavity* substituted by the *Ball*, the current triangulation is updated. Usually, the *Cavity* is constructed using the method of recursive neighboring search beginning from the *Base*.

The robustness of the above incremental Delaunay insertion method mainly depends on the validity of the *Cavity*, i.e., whether it is star-shaped.

To assure the validity of the *Cavity*, various techniques have been developed; see [7,12] for details. In [7], a correction procedure is proposed to the constructed *Cavity*, which checks the positiveness of the determinants formed by the inserted point and the outside facets of the *Cavity* and perform some deletions accordingly.

In our proposed boundary recovery procedure, after the local transformations (edges/faces swapping), the remaining missing edges/faces are recovered by the method of edges/faces splitting, that is, adding points to the constraints (edges/faces) using a modified version of the Delaunay insertion procedure. *The objective of the modification is: when adding a point to a constraint, any Protected edge/face, and hence any recovered constraint, is not deleted in the Delaunay insertion process.* Thus, with the insertion of the intersection points, the number of missing constraints is decreasing monotonically. Since the number of the missing constraints is finite, the convergence of the boundary recovery becomes obvious. A more detailed proof is provided in Section 5.

The modifications to the Delaunay insertion procedure consist mainly of two parts: the modified construction of the *Cavity*, and the modified correction of the *Cavity*.

Modification to the construction of the Cavity. To prevent from deleting the existing boundary constraints in the triangulation, the construction of the *Cavity* is modified as follows: When the recursive neighboring searching from the *Base* meets a face which has been marked as *Protected*, the search is stopped in the direction of the face and it is advanced towards other directions. Thus, the *Protected* face may not be included as an inner face of the *Cavity*. In the subsequent searches, it is possible, but much less likely in practice, that this face sometimes may be again included as an inner face, This leads to considerable saving of computation time later in the modified correction of the *Cavity*.

Modification to the correction of the Cavity. The *Cavity* constructed from the *Base* may contain some *Protected* edges or faces each of which corresponds to some boundary constraint of the surface triangulation S or a part of some boundary constraint. These constraints may be included as the inner edges/faces of the *Cavity* if no additional treatment is made to the constructed *Cavity*. In the construction of the *Balls*, these *Protected* edges/faces are deleted, which is counter-productive to our boundary recovery. To achieve the goal of protecting these edges/faces, the following corrections are applied:

- (1) For each *Protected* inner edge of the *Cavity*, the elements connecting it are found and one of the elements not contained in the *Base* is deleted.
- (2) For each *Protected* inner face of the *Cavity*, the elements connecting it are found and one of the elements not contained in the *Base* is deleted.
- (3) After each element deletion, the cavity is corrected through the tetrahedral determinants positiveness checking [7].

With our modified Delaunay insertion procedure, a rigorous proof is given in Section 5 to show that when adding an intersection point to the constraint by the above modified Delaunay insertion procedure, any recovered or previously existing constraint of T is never deleted.

4. Boundary recovery algorithm

Let S , T_d , T be defined as in Section 2. Here, we give a detailed algorithmic description on how to derive a conforming triangulation T from the initial Delaunay triangulation T_d .

Three procedures are presented: the initial comparison; the local transformation; and the edge/face splitting for the remaining constraints.

4.1. Initial comparison

Since only the vertex coordinates and the element vertices are contained as the input data of the surface triangulation S , the areas of input surface triangles and the connectivity data for the edges/faces of each point need to be first computed. For each point, the edges connecting it are found; for each edge, the triangles connecting it are found. The former can be done by looping over the edges, while the latter by looping over the faces. Using these connectivity data, comparisons between S and T_d are made. If an edge l of S exists in T_d , this edge is marked *Recovered*, and the corresponding edge l^* of T_d is marked *Protected*. Similar comparisons are performed for the faces of S and T_d .

4.2. Local transformation

After the initial comparison, we know which edges/faces of S are recovered in T_d and which are missing. For each missing face, we compare the configuration of the tetrahedral set which connect the face with the required configurations of the three basic single step local transformations (see Fig. 1): T23, T32, T44. If the configuration is in agreement with one of the three required, the corresponding transformation is performed and some constraints are recovered. For a missing face F of S , if its edge AB is missing in T_d , the corresponding points A^* , B^* in T_d are found first; then the faces of T_d which A^*B^* intersects are found. If there is only one such face F^* which A^*B^* intersects and the configuration of the tetrahedral set connecting F^* and A^*B^* agrees with the required one of T23, the flipping T23 is performed and the edge AB is recovered. For T32 and T44, the corresponding procedures are applied to check the consistency of the configuration and then a specific flipping is performed to recover some constraints. When performing a flipping, the newly generated edges/faces are compared with the connection data of the missing constraint, and the information on recovery are updated accordingly. After each flipping, the updated triangulation is still denoted by T_d .

4.3. Refinement: edges/faces splitting

After the above local transformation, a percentage of the total missing constraints can be recovered. Usually, if the quality of the surface triangulation S is of reasonable quality, this percentage can be substantial, say, about 80%. For the missing items not yet recovered, we apply the refinement method: edges/faces splitting. We do not adopt the often followed method of first recovering all missing edges then all missing faces. Instead, we apply the procedure of sequentially recovering the missing faces, and recovering the edges simultaneously. In the following, we first discuss the recovery of the missing edges of a missing face by edge splitting: nearest-mid intersection points insertion; then we address the complete recovery of faces by face splitting. When inserting a point into a constraint, the updated triangulation is still denoted by T (the triangulation after local transformation is denoted by T as well).

4.3.1. Edge recovery by recursive nearest-mid intersection points insertion

Let F be a missing face and E be one of its missing edge with end points A, B . The corresponding points of A, B in T are A^* and B^* . To recover AB (or equivalently recover A^*B^*), the intersection points $\{P_k\}$ of A^*B^* with the faces of T are found first. From $\{P_k\}$, the point P^* is found, which is closest to the midpoint of A^*B^* . Such a point P^* is called the nearest-mid intersection point (of $\{P_k\}$) with respect to A^*B^* . Then P^* is inserted into T in the modified Delaunay insertion procedure (discussed in Section 3) and P^* is deleted from the set $\{P_k\}$. After the insertion, the set NE of newly generated edges of T is compared with $Sl = \{P^*A^*, P^*B^*\}$. If an edge el in NE agrees with one of Sl , the edge is marked *Protected*, and Sl is updated by $Sl = Sl - \{el\}$. That is to say, when a newly generated edge is a subpart of the missing edge, it is marked *Protected*, and it will not be deleted in the successive points insertions. Also the recovered information data are updated. If Sl becomes empty, it means A^*B^* is recovered and E is marked *Recovered*. Else, for each element of Sl , from $\{P_k\}$, the nearest-mid intersection point with respect to that line segment is found and they are added recursively into T by the modified Delaunay insertion procedure until Sl becomes empty, i.e., until the missing edge is recovered completely. Also, after inserting each nearest-mid intersection point, the newly generated faces NF of T are compared with the connection data of AB 's faces. If a face F_s of NF is a part of a missing face F_m connecting AB , F_s is marked *Protected*, and the recovering data of F_m are updated accordingly. This means, if a newly generated triangle is a subpart of the missing triangle, it is marked *Protected*, i.e., not deleted in the subsequent points addition. And when a subpart of a missing triangle is recovered in the *Protected* manner, it is added to the already recovered subparts, and the sum of area of these recovered subparts is called *RecoveredArea*. If the *RecoveredArea* is equal to that of the missing triangle, it (F_m) is marked as *Recovered*. It means that in the recovery procedure the faces F_m of AB are recovered simultaneously, such an event happens very often in practice.

For clarity, the algorithmic details given above is presented in the following pseudo-code:

BEGIN

Let A, B be the end points of E , find the corresponding A^*, B^* in T ;

Find the set of intersection points $\{P_k\}$ of A^*B^* and the faces of T ;

Initialize Sl to be $\{A^*B^*\}$;

Continue;

For each element le (with endpoints P_a, P_b) of Sl , do

Find the point P^* from $\{P_k\}$ nearest to the midpoint of le ;

Insert P^* into T in the modified Delaunay insertion procedure;

Delete P^* from $\{P_k\}$: $\{P_k\} = \{P_k\} \setminus P^*$

For each newly generated edge E_n , do

If E_n is a part of le , then

Mark E_n as *Protected*;

Endif

Enddo

For each newly generated face F_n , do

If F_n is a part of AB connecting missing face F_m , then

Mark F_n as *Protected* and Update the *RecoveredArea* of F_m ;

If the *RecoveredArea* of F_m equals the area of F_m ,

Mark F_m as *Recovered*,

Endif

Endif

Enddo

If P^*P_a and P^*P_b are both newly generated edges then

Delete le from Sl : $Sl = Sl - \{le\}$;

```

Else
  Add  $P^*P_a$  or  $P^*P_b$  to  $Sl$ :  $Sl = Sl + \{P^*P_a\}(\{P^*P_b\})$  accordingly, or both;
Endif
Enddo
If  $Sl$  is not empty, go to Continue; else, Endif
END

```

4.3.2. Face recovery by splitting

Let F be a missing face of S with its three edges recovered. In this case, there must be some edges in T which intersect F or the *Unrecovered* parts of F . For the complete recovery of F , the intersection points of these edges associated with F are found first. These points do not intersect any recovered part of the missing face, since the recovered sub-triangles are protected during the points-addition procedure with the modified Delaunay insertion procedure. Here, the accurate intersection checking is the key to the robustness of the whole conforming recovery. Then, these intersection points are inserted recursively into T , i.e., the current triangulation by the modified Delaunay insertion procedure until the *RecoveredArea* of F is equal to that of F , that is to say until the missing face is recovered completely. Before an intersection point for insertion is chosen, we check whether it is in any recovered subpart of the missing face. If it is, this choice is given up and we go for another. With the insertions of the intersection points, the *RecoveredArea* of the missing face is becoming larger and larger, or say, more and more subparts of the missing face are recovered, and finally, the complete recovery is obtained. And the recovery is performed independently from any other recovered or existing face.

Again for clarity, the pseudo-code of the final recovery by face splitting described in the above is given as follows:

```

BEGIN
Find the intersection points set  $\{P_k\}$  of  $F$  with the edges of  $T$ ;
  Continue
Find a point  $P^*$  from  $\{P_k\}$  such that:  $P^*$  is not in any recovered part of  $F$ 
Insert  $P^*$  into  $T$  in the modified Delaunay Insertion procedure;
Delete  $P^*$  from  $\{P_k\}$ :  $\{P_k\} = \{P_k\} \setminus P^*$ 
For each newly generated face  $F_n$ , do
  If  $F_n$  is a part of  $F$  then
    Mark  $F_n$  as Protected and Update the RecoveredArea of  $F_n$ ;
  Endif
Enddo
If the RecoveredArea of  $F_n$  equals that of  $F_n$  then
  Mark the missing face  $F$  as Recovered; go to End;
Else
  Go to Continue for the insertion of another intersection point;
Endif
END

```

4.4. Deletion of the outside elements

When the missing faces are all recovered, the recursive neighboring search method proposed in [2,14] is used to classify each element of T and mark the class of the elements not in the domain to be zero. Deleting the zero class elements from T , the conforming triangulation of S is then obtained, i.e. the boundary recovery is completed.

5. Proof of convergence

Based on the algorithm in the Section 4, we know that our boundary recovery contains two parts: first the local transformation, then the edges/faces splitting.

For the local transformations, it is obvious that the three single step flippings (swappings) do not delete any existing or recovered constraint. Hence, the proof of the convergence of our boundary recovery scheme is reduced to the proof of the convergence of the second part.

In the following, we first prove that adding an intersection point to a missing constraint in the modified Delaunay insertion procedure never deletes any *Protected* edge/face, thus the existing or recovered constraint, of the triangulation T ; then we prove that each missing edge/face can be recovered as a union of edges/faces in T by adding finite intersection points to it in the modified Delaunay insertion procedure; finally, we combine these two proofs to obtain a complete proof of the convergence of our boundary recovery algorithm. The results are presented sequentially in three theorems and the final conclusion is made in Theorem 4.

Theorem 1. *Let S, T_d, T be defined as in Section 2, and assume that E (of S) is a missing constraint (edge or face) in T . Denote the intersection points set of E with T by $\{P_k\}$. When adding a point p of $\{P_k\}$ into the modified Delaunay insertion procedure described in Section 3, no *Protected* edge/face of T is deleted. Hence, the *Recovered* edge/face of S , i.e., the *constrained* edge/face of S which exists in T_d or has been recovered in T is not deleted.*

Proof. We first show that the *Base* of p contains no *Protected* edge/face as an inner edge/face. Depending on the different positions of p in its *Base*: inside an element, on an edge, or on a face, we present the proof respectively for each of the cases.

- (1) Inside an element (see Fig. 3(a)): p is in the interior of the element t , which implies that the *Base* of p has only one element T . Obviously there is no inner edge/face in T and hence no *Protected* inner edge/face is contained in the *Base*.
- (2) On an edge (see Fig. 3(b)): p is on an edge l , then the *Base* of p equals the tetrahedra set TS connecting l . There is only one such edge l being an inner edge. If l is *Protected*, then the constraint of S to be recovered must intersect l , which is inconsistent with the configuration of S . And there are n inner faces with n being the number of the elements of TS . Similarly, if there is one inner face which is *Protected*, the geometric inconsistency is again obvious. So, there is no inner *Protected* face in the *Base* of p .
- (3) On a face (see Fig. 3(c)): p is on a face F , then the *Base* of p consists of two elements which connect F . In the *Base*, there is no inner edge and only one inner face F . If F is *Protected*, similar to (2), let the constraint to be recovered be either an edge or face, we have that this constraint intersects F , which contradicts to the geometric configuration of S .

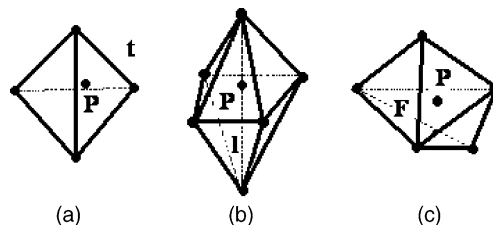


Fig. 3. (a) In an element, (b) on an edge, and (c) on a face.

Secondly, we verify the claim that there is no inner *Protected* edge/face in the elements added to the *Cavity* of p after correction.

From the added corrections to the *Cavity* (described in Section 3) we know that if there is an inner edge/face in the *Cavity* which is marked *Protected*, this edge/face is to be treated so that one element (not in the *Base*) connecting the edge/face to the *Cavity* is deleted. Then this edge/face becomes a boundary element of the updated *Cavity*. That is to say, this edge/face is no longer an inner *Protected* element of the *Cavity*. Since the *Base* of p contains no inner *Protected* edge/face, the *Cavity* after correction has no inner *Protected* edge/face.

Finally, since the corrected *Cavity* of p has no inner edge/face which is marked *Protected*, and in the construction of the Ball of p , only the inner edge/face is deleted, there must be no *Protected* edge/face which is deleted in the modified Delaunay insertion procedure of p . From the relationship between the *Protected* edges/faces of T and the *Recovered* elements of S , we conclude that no *Recovered* edge/face of S , i.e. no constrained edge/face of S which exists in T_d or has been recovered in T , is deleted.

Before moving to the second part of the proof, we present a property of the modified Delaunay insertion procedure. \square

Proposition 1. *Let $S, T, E, \{P_k\}$ be defined as in Theorem 1. When inserting a point p (in $\{P_k\}$) into T in the modified Delaunay insertion procedure, there is no new intersection point of E and the updated T , which is different from any one in $\{P_k\}$.*

Proof. We prove the proposition according to the geometry of the constraint E , an edge or a face, respectively.

- (1) E is an edge: Let A, B be the two end points of the missing edge E in T (see Fig. 4). Now p is inserted into T in the modified Delaunay insertion procedure. From the construction of the Ball of p , we know that all newly generated edges and faces must pass through p and have p as a vertex. If one new face intersects AB in Q and Q is not in $\{P_k\}$, then one edge of the new face must intersect AB in Q . Denote the end points of the edge by C, D . According to the construction of the Ball of p , there must exist a face F which connects CD and F is an existing face of the *Cavity* of p . Therefore, AB intersects F and the intersection point is Q . That is to say, Q is in $\{P_k\}$, which contradicts to the assumption on Q . Hence, there is no new intersection point of E and T .
- (2) E is a face: Let A, B and C be the three vertices of the missing face E in T (see Fig. 5). p is inserted onto E in the modified Delaunay insertion procedure. Similarly as in (1), all newly generated edges/faces must pass through p and have p as a vertex. If one new edge intersects the triangle ABC in Q , then the line segment Qp lies on the triangle ABC . Therefore, pQ is just the new edge and Q must be a boundary of the *Cavity* of p . Hence, there must be a boundary edge QO (of p 's *Cavity*) which has O as a vertex. Obviously, the edge QO is an existing edge of p 's *Cavity*, and QO intersects the triangle ABC in Q , so

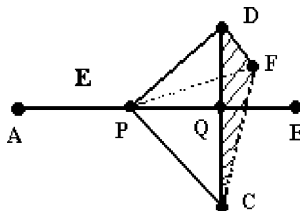


Fig. 4. E is an edge.

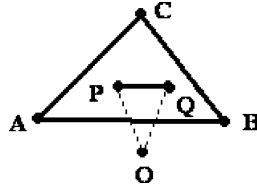


Fig. 5. E is a face.

Q is in $\{P_k\}$, which contradicts the assumption on Q . Hence, there is no new intersection point of E and T . \square

With the above preparation, we come to the second part of the proof, which states that each missing edge/face can be recovered through a finite number of insertions of intersection points in the modified Delaunay insertion procedure. We first discuss the recovery of edges then faces.

Lemma 1. *Let T be defined as in the Theorem 1, assume A, B are the two inner (not boundary) vertices of T with the line segment AB not an edge in T , i.e. AB is missing in T . Then, AB intersects at least a face F of T .*

Proof. Let S_a be the set of tetrahedra having A as a vertex (see Fig. 6). Because A is an inner vertex, S_a must be connected and closed and A is in the interior of the polyhedron S_a forms. Denote the boundary faces of S_a by S_o . Since B is a vertex of T , it cannot be in the interior of S_a . If B is on S_o , it must be a vertex of one of faces in S_o , then AB is an edge in T , contradicting to the assumption of the lemma. Hence, B is in the outside of S_a . Thus, AB must intersects a face in S_o , and thus a face of T . \square

Lemma 2. *Let S, T be defined as in the Theorem 1. Assume that E (of S) is a missing edge in T . A and B are the two end points of E . A^*, B^* are the corresponding points in T of A, B (the coordinates are the same, but possibly with different numbering). $\{P_k\}$ is the set of intersection points of T with the line segment A^*B^* . Assume that in performing the nearest-mid intersection points insertion (described in Section 4) for the recovery of AB , a nearest-mid intersection point P (of $\{P_k\}$) has been inserted into T . Then, in the nearest-mid intersection insertion procedure, there must be two nearest-mid intersection points P_l and P_r in $\{P_k\}$ or two points P_l and P_r in $\{A, B\}$ such that P_lP, PP_r exist as two edges in T .*

Proof. We only prove the case of P_l (similar argument works for P_r).

Assume in the insertion procedure of the nearest-mid intersection points that, when inserting P into T , there is no nearest-mid intersection point in $\{P_k\}$ or no point in $\{A, B\}$ which lies on the left of P (see Fig. 7) as a vertex of T and connects P by an edge in T , then there must exist a point P^* (on the left of P) which is an inner vertex of T or is A and the line segment P^*P is not an edge of T . From Lemma 1, there must exist at

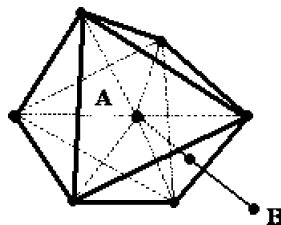


Fig. 6. AB intersects at least a face.

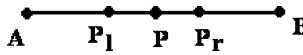


Fig. 7. P must be connected by its left and right points.

least a face of T which P^*P intersects. Denote the set of intersection points of T and P^*P by $\{Q_k\}$. From Proposition 1, we know $\{Q_k\}$ is a subset of $\{P_k\}$. Then, from $\{Q_k\}$, or equivalently from $\{P_k\}$, the nearest-mid intersection point P^{**} of P^*P are found and P^{**} is added in the same way as P . If $P^{**}P$ is still not an edge of T , the above procedure is performed iteratively. Since $\{P_k\}$ is finite, the iteration stops in finite steps. And the last point P_l must exist in $\{P_k\}$ or in $\{A, B\}$, and it must connect with P as an edge of T ; else, by Lemma 1, there is at least a point in $\{P_k\}$ which is between P_l and P , and certainly we can find the nearest-mid intersection point from $\{P_k\}$ and insert it, which contradicts to the assumed convergence of the nearest-mid intersection points insertion procedure. \square

Combining Lemma 2 and the Proposition 1, we have the following:

Theorem 2. *Let S, T be defined as in Theorem 1. And assume E (of S) is a missing edge and $\{P_k, k = 1, n(E)\}$ is the intersection points set of E and T . Then E can be recovered as a union of edges of T through a finite nearest-mid intersection point insertions in the modified Delaunay insertion procedure, and the number of insertion is no more than $n(E)$.*

To prove the recovery of the face splitting, we first present some definitions.

S-Polygon: A simply connected polylateral domain on a plane F (of 3D) is called a S-Polygon with respect to F .

M-Polygon: A multi-connected polylateral domain on a plane F (of 3D) is called a M-Polygon with respect to F .

P-Polygon: Let $SP(F) = \{\text{all S-Polygons with respect to } F\}$; and $MP(F) = \{\text{all M-Polygons with respect to } F\}$. If E is the union of some elements in the set $\{t: t \text{ is of } SP(F) \text{ or } MP(F)\}$, then E is called a P-Polygon.

A triangle of T is a P-Polygon. If a part of the triangle is removed from the triangle, the remaining part is a P-Polygon, independent of its position.

In Fig. 8(a), P_1, \dots, P_n are all on the plane F and they form a polylateral domain L . L is then an S-Polygon with respect to F . The simplest S-Polygon is a triangle of T (defined as before). In Fig. 8(b), P_1, \dots, P_n form L , Q_1, \dots, Q_m form L_1 , and $L \setminus L_1$ is an M-Polygon with respect to F .

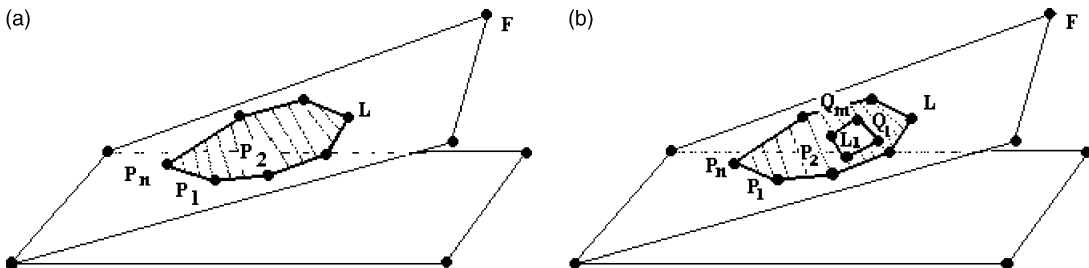


Fig. 8. (a) An S-Polygon and (b) an M-Polygon.

Lemma 3. *Let T be defined as in Theorem 1. M is a P-Polygon with respect to a certain plane. The vertices and edges of M all exist as inner vertices and edges in T , but there is no triangle of T which is a part of M . Then, for each edge E of M , there must exist an edge L of T which intersects M and the intersection point is in the interior of M . In addition, E and L form a tetrahedron of T , and there exist at least an intersection point of M and T .*

Proof. Without the loss of generality, we just prove the case of M being an S-Polygon. In Fig. 9, M is an S-Polygon, E is an edge of M . From the assumption, we know E is an inner edge of T , then the tetrahedra set TS connecting E form a closed and connected polyhedron and E is an interior edge of this polyhedron. Because M passes through E , there must be an element t of TS which M intersects. If M intersects t on a face of t , then the face is a part of M , which contradicts the assumption. Hence, M must intersect an edge L of t and the intersection point is between the two endpoints of the edge. Because M passes E , L must be the opposite edge to E in t , i.e., L and E form a tetrahedron in T . As E is an arbitrary edge M , there must exist at least an intersection point of M and T . \square

Theorem 3. *Let S, T be defined as in Theorem 1, assume that F (of S) is a missing face and $\{P_k, k = 1, n(F)\}$ is the set of intersection points of F and T (actually with the edges of T), and the edges of F have been recovered using the nearest-mid intersection points insertion described in Section 4. Then F can be recovered as a concatenation of faces of T using the intersection points insertion procedure described in Section 4, and the number of insertions is no more than $n(F)$.*

Proof. No matter whether F is partially recovered or no part is recovered after the recovering procedure of its edges, the remaining part of F forms a P-Polygon M_0 . Let E_0 be an edge of M_0 , then according to Lemma 3, there is an edge L with which E_0 forms a tetrahedron t of T , and the intersection point P_o of L with M_0 is in the interior of M_0 . From Theorem 1, we know that P_o is in $\{P_k\}$. In the process of inserting the intersection points of $\{P_k\}$ for the recovery of F , if a point P in $\{P_k\}$ is inserted and t is contained in the corrected Cavity of P , then by the definition of the modified Delaunay insertion procedure, since E_0 is Protected, P and E_0 forms a new face F_n in the construction of P 's Ball. Else, P_o is inserted into t which makes t being part of the Base of P_o . Obviously, P_o with E_0 forms a new face F_n and F_n is clearly a part of M_0 . Let $M_1 = M_0 \setminus F_n$. Then when advancing the intersection points insertion procedure, the initial P-Polygon is reduced to another P-Polygon M_1 , whose area is less than that of M_0 . Iteratively, we get a sequence of P-Polygons: M_0, M_1, M_2, \dots . Since $\{P_k\}$ is finite, the iteration is convergent, or say finite. Denote the converged or the final one by M_c . We claim that M_c is empty. Else, according to Lemma 3, there must exist an intersection point Q in M_c and

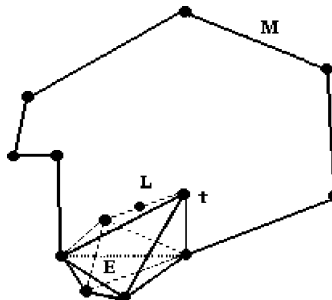


Fig. 9. M intersects T with at least one intersection point.

Q is in $\{P_k\}$, then we must insert Q into T , which is in contradiction to our assumption on the convergence of the iteration and the face splitting procedure. Hence, F can be re-established as a union of faces of T and obviously the number of insertions is no more than $n(F)$. \square

Combining Theorems 1–3, we have the following conclusion:

Theorem 4. *Let S , T_d , T be defined as in Theorem 1. The boundary recovery algorithm presented in Section 4 for recovering the missing edges/faces of S is convergent, i.e. the missing constraints can be recovered as a concatenation of edges/faces of T through local transformations (edges/faces swapping) or through a finite number of intersection points insertion in the modified Delaunay insertion procedure.*

6. Application examples

Our boundary recovery which combines local transformations (edge/face swapping) with Delaunay refinement method (edges/faces splitting) has been introduced into a complete mesh generation procedure to deal with various geometric models. As some preliminary illustrations, the surface triangulations and the cross-sectional views of their conforming 3D triangulations of four examples, none having interior points, i.e., having only boundary vertices, are presented in Figs. 10–13.

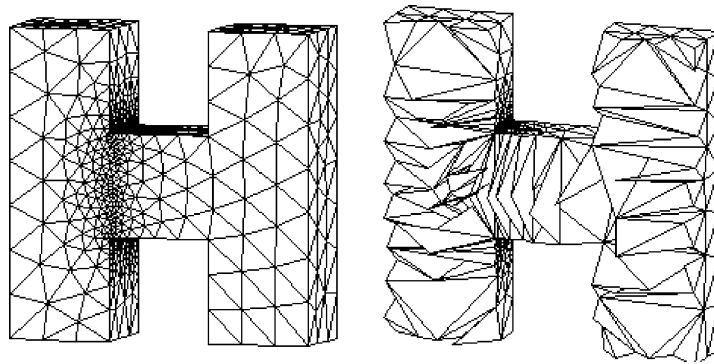


Fig. 10. The surface triangulation and the cross-sectional view for an H-shape model.

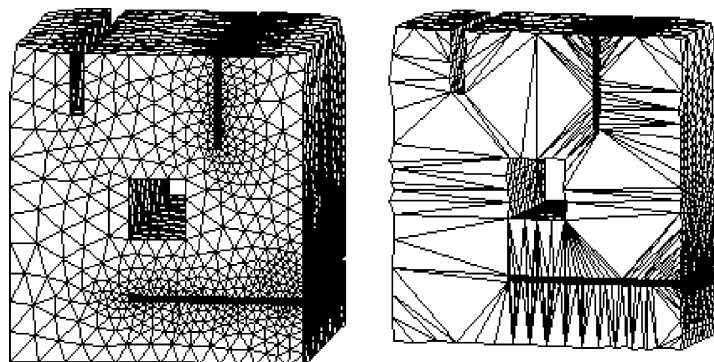


Fig. 11. The surface triangulation and the cross-sectional view for a model with cavities.

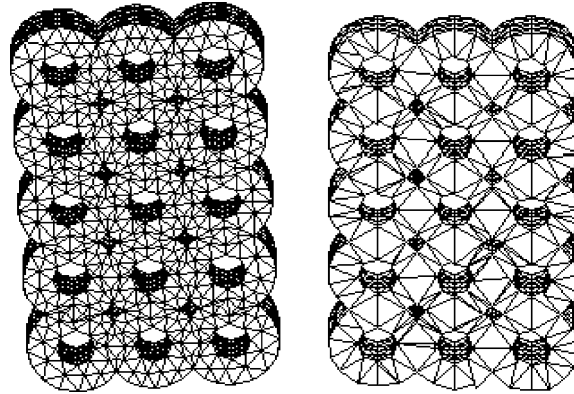


Fig. 12. The surface triangulation and the cross-sectional view for a model with 23 holes.

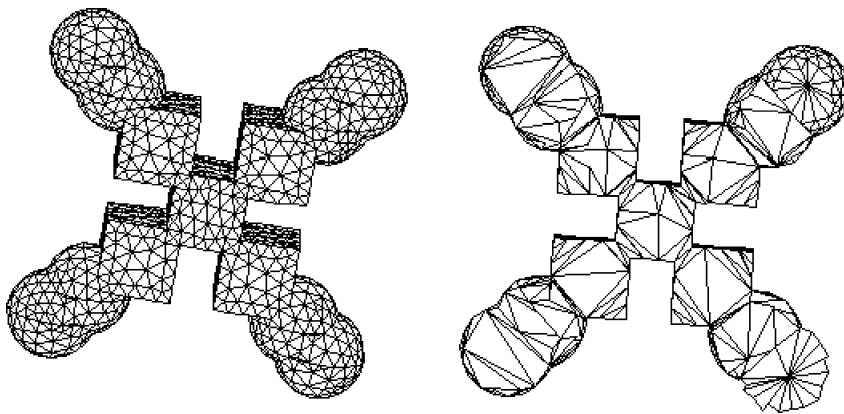


Fig. 13. The surface triangulation and the cross-sectional view for intersecting objects.

Table 1
Mesh statistics of the examples

Examples	Fig. 10	Fig. 11	Fig. 12	Fig. 13
Total boundary vertices	866	8015	3011	1456
Original missing faces	12	32	108	64
Added vertices	2	6	18	12
Faces recovered by swapping	8	24	79	48
Added vertices by ODR	6	21	64	36

In Table 1, the geometric data for each example are provided which include the total number of boundary vertices in the triangulation, the number of missing boundary faces in the original triangulation, the number of recovered missing boundary faces through simple swapping and the number of vertices added on the boundary. *Note that no interior vertices are added to the triangulation.* For comparison, in Table 1, we also report the number of added vertices if we use purely the original Delaunay refinement [16,17] (abbr. ODR, meaning no modification of the Delaunay insertion) to recover the missing boundary faces. From the ratio of the constraint recovered by edges/faces swapping (almost up to 80%) we see that it

Table 2
Ratios of CPU times of boundary recovery to that of complete meshing

Examples	Fig. 10	Fig. 11	Fig. 12	Fig. 13
Simple swap	0.24	0.74	2.34	1.34
Modified Delaunay refinement	0.86	2.58	5.86	4.36
Our method (the sum of the above two)	1.1	3.32	8.2	5.7
Complex flipping	3.0	5.1	27.2	19.2
Original Delaunay refinement (ODR)	3.1	4.9	19.1	13.3

is very useful to add the edges/faces swapping as the first step of the boundary recovery. Furthermore, the comparison of the numbers of added vertices of our method with that of the ODR shows that our modified Delaunay insertion procedure is very efficient. If only using ODR, too many vertices are added on the boundary which violates the given sizing specification of the area neighboring the added vertices.

In Table 2, the ratios (in percentage) of the CPU times of the boundary recovery step to the whole meshing process (including other steps such as field points generation, Delaunay insertion and mesh optimization) of various methods are compared. In addition to our method, two other methods used as references are the method of complex flipping and ODR, with the former following the method proposed by George [14]. For comparison, we apply different methods of boundary recovery, but use the same procedures for other steps. For the first pair of examples (Figs. 10 and 11), ODR and complex flipping methods have somewhat similar efficiency for the recovery. But for the second pair of examples (Figs. 12 and 13), the ratios indicate that the recoveries are substantially slower than before, and ODR is faster than the complex flipping method due to the more complex geometry involved in the later two examples. For a complex geometry, more time is obviously needed for complex flipping due to the heuristic nature of the method. For ODR, the efficiency is mainly related to the number of the missing constraints and the CPU time for the intersection checking is a large part of the whole process. Of course, the complexity of the underlying geometry also influences its efficiency, as illustrated in the splitting recovery: the more complex the model is, the more easily for some existing or previously recovered edges or faces to be deleted. For the four examples given here, the CPU time ratios of our methods are much less than that of the ODR and the complex flipping. This is because, for most of the missing faces, we first use the simple and efficient swapping procedures for the recovery, then for the remaining items, the modified Delaunay refinement we proposed recovers them one by one, and avoids the deletion of the recovered constraint. Especially for complex geometry, the introduction of the combined techniques has obviously improved the effectiveness and the robustness of the boundary recovery and hence, the whole process of mesh generation.

7. Conclusions and future works

An efficient boundary recovery algorithm is presented, which combines the local transformations: edges/faces swapping with the Delaunay refinement method: edges/faces splitting. In the splitting of constraints, we modify the Delaunay insertion procedure and guarantee that when inserting an intersection point to a constraint, existing or recovered constraints in the volumetric triangulation will never be deleted. This enhances the efficiency of the algorithm and guarantees the convergence of the algorithm. While this paper contains mainly a complete and rigorous proof of our algorithm, several preliminary meshing examples are also presented which demonstrate the effectiveness and robustness of the presented boundary recovery algorithm. One future direction of research is to improve the positions of the inserted intersection points in order to improve the quality of the final mesh neighboring the added points.

Acknowledgements

The authors would like to thank the referee for valuable suggestions and comments.

References

- [1] D.F. Watson, Computing the n -dimensional Delaunay tessellation with applications to Voronoi polytopes, *Comput. J.* 24 (1981) 167–172.
- [2] H. Borouchaki, P.L. George, Aspects of 2-D Delaunay mesh generation, *Int. J. Numer. Methods Engrg.* 40 (1997) 1957–1975.
- [3] N.P. Weatherill, The integrity of geometrical boundaries in the two-dimensional Delaunay triangulation, *Comput. Appl. Numer. Methods* 6 (1990) 101–109.
- [4] L.P. Chew, Guaranteed quality mesh generation for curved surfaces, in: *Proc. 9th Annual Comput. Geom.*, 1993, pp. 274–280.
- [5] S.H. Lo, Automatic mesh generation over intersecting surfaces, *Int. J. Numer. Methods Engrg.* 38 (1995) 943–954.
- [6] H. Borouchaki, P. Laug, P.L. George, Parametric surface meshing using a combined advancing-front generalized Delaunay approach, *Int. J. Numer. Methods Engrg.* 49 (2000) 233–259.
- [7] H. Borouchaki, P. George, S. Lo, Optimal Delaunay point insertion, *Int. J. Numer. Methods Engrg.* 39 (1996) 3407–3437.
- [8] H. Borouchaki, S. Lo, Fast Delaunay triangulation in three dimensions, *Comput. J. Numer. Methods Engrg.* 128 (1995) 153–167.
- [9] A. Rassineux, Generation and optimization of tetrahedral meshes by advancing front technique, *Int. J. Numer. Methods Engrg.* 41 (1998) 651–674.
- [10] M. Shephard, M.K. Georges, Automatic three-dimensional mesh generation by the finite octree technique, *Int. J. Numer. Methods Engrg.* 32 (1991) 709–749.
- [11] R. Lohner, P. Parikh, Generation of three-dimensional unstructured grids by the advancing-front method, *Int. J. Numer. Methods Engrg.* 8 (1988) 1135–1149.
- [12] P.J. Frey, H. Borouchaki, P.L. George, 3D Delaunay mesh generation coupled with an advancing-front approach, *Comput. Methods Appl. Mech. Engrg.* 157 (1998) 115–131.
- [13] A. Liu, M. Baida, How far flipping can go towards 3D conforming/constrained triangulation, 9th International Meshing Roundtable, 2000.
- [14] P.L. George, F. Hecht, E. Saltel, Automatic mesh generation with specified boundary, *Comput. Methods Appl. Mech. Engrg.* 92 (1991) 169–188.
- [15] N. Weatherill, O. Hassan, Efficient three dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints, *Int. J. Numer. Methods Engrg.* 37 (1994) 2005–2039.
- [16] J. Shewchuk, Delaunay Refinement Mesh Generation, Ph.D. Thesis, Computer Science Dept. Carnegie Mellon, Univ., 1997.
- [17] B.K. Karamete, M.W. Beall, M.S. Shephard, Triangulation of arbitrary polyhedra to support automatic mesh generators, *Int. J. Numer. Methods Engrg.* 49 (2000) 167–191.
- [18] F. Hermeline, Une methode automatique de maillage en dimension, Thesis lect., Universite Paris 6, Paris, 1980.
- [19] J. Wright, A. Jack, Aspects of three-dimensional constrained Delaunay meshing, *Int. J. Numer. Methods Engrg.* 37 (1994) 1841–1846.