Automatic 3D mesh generation conforming a prescribed size map.

Paul Louis George [†] and Houman Borouchaki [‡]

September 2, 1997

[†] INRIA, Gamma Project,
Domaine de Voluceau, Rocquencourt,
BP 105, 78153 Le Chesnay Cedex, France.
email: paul-louis.george@inria.fr
and
[‡] UTT, GSM-LASMIS
12 rue Marie Curie
10010 Troyes cedex, France.
email: houman.borouchaki@univ-troyes.fr

Abstract. *The generation of an adequate mesh is an essential prerequisite in any finite element simulation of a physical phenomenon described in terms of PDE's. This paper introduces a method enabling to generate a three-dimensional mesh conforming a user-specified size map. This Delaunay-type method creates isotropic tetrahedral meshes conforming the specified size map. This method proved to be especially suitable in mesh adaption schemes (mesh, numerical computation, error estimate)*

Keywords. Mesh generation - Finite Element - Adaptation - Delaunay - Adapted mesh - Isotropic mesh.

1 Introduction

An essential pre-requisite in the numerical finite element simulation of physical problems expressed in terms of PDE's is related to the construction of an initial adequate mesh of the domain [4]. This first stage, usually involving any fully automatic mesh generation method (cf. [5]), is then followed by a computational step. The numerical solution obtained with the initial mesh is generally analyzed via an error estimate, which will indicate whether or not the process has converged, based on the quality of the solution. The latter is closely related to the mesh adequation with the underlying physical phenomenon. Hence, the role of the error estimate is to indicate if the mesh density is too coarse or to fine in some regions. In the first case, the computational scheme is too time consuming, as in the other case, the solution behavior is probably not even captured. To match these requirements, an adapted mesh need to be generated, its refinement been related to the above specifications.

We do not focuss explicitely on the error estimate part, however we like to emphasize the meshing technology and, in the isotropic case, we propose a method enabling to create a mesh, such that

- the elements are as equilateral as possible,
- the element size corresponds to a pre-specified size (supplied by the error estimate).

The proposed method is Delaunay-based. At first, Section 2, we recall the main features of this type of method (in the classical case, *e.g.* without a size map). Section 3, we mention several approaches used to define a size map in the context of mesh adaption. The selected approach consists in using a background mesh, similar to the current mesh, and Section 4, we describe a procedure to create the mesh vertices using the background mesh. To this end, we recall the notion of normalized unit length. Section 5, we review several mesh optimization tools. Section 6 deals briefly with the (analytical) surface remeshing to conform a size map. Section 7, several application examples will emphasize the efficiency of the proposed method and, finally, Section 8, future developments and extensions of this approach will be mentioned.

2 Classical scheme of a Delaunay-type method

The aim of this section is to recall the main features of a Delaunay- type mesh generation method. We describe the principal steps of this supposedly well-known method, [1], [16], ..., in order to show what modifications are needed in the mesh generation scheme to respect a pre-specified size map.

We consider the classical case, in other words we assume a conforming mesh of the domain boundary. The problem we face is to mesh the domain based on this sole information.

2.1 Synthetic scheme

A possible scheme for a Delaunay-type mesh generation method involves the following successive steps (cf [7])

- Preliminary step: data input (point coordinates, boundary entities (faces and edges) and internal entities (if needed), construction of a bounding box (to consider a convex context), mesh of this box with five tetrahedra.
- Generation of the mesh of the box: point insertion using the *Delaunay kernel.*
- Constrained boundary mesh construction: find and recover all missing specified edges and faces, identification of the connected components.
- Internal point creation and insertion: (A) Analysis of the current (internal) mesh edges, creation of points along these edges and point insertion, then back to (A).
- Mesh definition: remove all elements not strictly included inside the domain, classification of the remaining elements with respect to the connected components.
- Mesh optimization: edge swapping, node relocation (vertex smoothing), etc.

This scheme is based on four basic algorithms, namely a point insertion algorithm (the kernel), a boundary recovery algorithm, an internal point creation algorithm and finally an optimization algorithm.

2.2 Incremental point insertion process

The Delaunay kernel is an incremental procedure allowing to insert a point in an existing triangulation (we consider here a convex domain, any point to be inserted being an internal point).

Let consider the Delaunay triangulation $\mathcal{T}_i$ of the aforementioned bounding box, containing the first ith points of the set of points to insert as element vertices. Let consider the $i+1$th point of this set, denoted as P. The Delaunay kernel allows to construct, from $\mathcal{T}_i$, the triangulation $\mathcal{T}_{i+1}$ containing P as mesh vertex. This can be written as (cf. [11])

$$\mathcal{T}_{i+1} = \mathcal{T}_i - \mathcal{C}_P + \mathcal{B}_P \tag{1}$$

where $\mathcal{C}_P$ is the *cavity* associated with the point P, namely the set of tetrahedra of $\mathcal{T}_i$ whose circumscribed sphere includes P, and $\mathcal{B}_P$ is the *ball* of P, *e.g.* the set of tetrahedra obtained by joining P to the external faces of $\mathcal{C}_P$.

2.3 Boundary recovery

Using the procedure (1) applied to the set of points of the domain boundary, we obtain a mesh of the bounding box, such that the mesh vertices are, apart from the corners of this box, the face vertices (the edge endpoints) of the boundary mesh. However, this procedure does not usually ensure the existence of all constrained edges and faces, thus preventing the extraction of an initial mesh of the domain from the current mesh of the box. The current mesh must then be modified to recover the missing boundary entities.

A set of topological mesh modifications (edge and face swapping), possibly coupled with a point creation step (the so-called *Steiner* points), is iteratively applied and leads, usually, to the sought result (cf. [9] and [17], for instance). The domain is then correctly defined and internal points can be introduced.

2.4 Internal point creation

The proposed method consists in analyzing the internal mesh edges and, eventually, creating points along these edges. Once all the points have been created, they are then inserted using again the algorithm (1).

Let AB be a current mesh edge and let h_A and h_B be the desired sizes at points A and B[1]. The aim of the method is to decide whether or not it is necessary to construct one or more point along AB, and, if yes, how many point have to be created. Let n be this number of points, the locations of these points have to be determined in order to saturate the edge according to a smooth distribution.

Let consider an arithmetic progression for this distribution of points. Hence, cf. [10], if $h(0) = h_A$ and $h(n+1) = h_B$ are the sizes respectively associated with $P_0 = A$ and with $P_{n+1} = B$, it is then possible to define a series as

$$\left\{ \begin{array}{lcl} \alpha_0 & = & h(0) + r \\ \alpha_n & = & h(n+1) - r \\ \alpha_i & = & d(P_i, P_{i+1}) \end{array} \right. \tag{2}$$

where $d(P_i, P_{i+1})$ is the (Euclidean) distance between P_i and P_{i+1}, and r is the reason of the progression. The problem is equivalent to solve the system

$$\left\{ \begin{array}{lcl} \sum\limits_{i=0}^{n} \alpha_i & = & d \\ \alpha_{i+1} & = & \alpha_i + r \end{array} \right. \tag{3}$$

to find both r and n, d being the length of AB. We deduce

$$n = \frac{2d}{h(0) + h(n+1)} - 1 \tag{4}$$

[1]If A is a boundary vertex, its size is set as the average value of the edge lengths of all boundary edges sharing A. If A is an internal vertex, the size h_A associated with A is known, it results from an interpolation at the time A has been created.

and

$$r = \frac{h(n+1) - h(0)}{n+2}. \quad (5)$$

As n is an integer, the solution is eventually modified so that n and r lead to an exact discretization of the current edge. Once n and r are known, the α_i's are determined and the series of points is fully defined. A value h is then associated with each point, this value being computed from the corresponding values associated with the supporting edge.

This process is iterated for all mesh edges and the resulting set of points is then filtered. The filtering operation is required as the vertices are properly ordered with respect to each edge independantly, although this property may not hold globally (for instance in case of vertices contructed from neighboring edges emanating from a unique point). The points retained are then inserted via the relation (1) and the whole process is iterated as long as mesh edges need to be sudivided.

2.5 Mesh optimization

The mesh resulting after point insertion is then optimized to improve the element shape quality. The optimization procedure consists in

- removing a free face (swapping),
- removing a free edge (local remeshing),
- removing a free edge by a colapsing operation,
- relocating a free node,
- removing a free vertex,

as long as the mesh quality is improved. The mesh quality criterion is globally defined as

$$Q_{\mathcal{T}} = \max_{K \in \mathcal{T}} Q_K \quad (6)$$

and, for each element, a shape quality value is defined as

$$Q_K = \alpha \frac{h_{max}}{\rho_K} \quad (7)$$

where α is a normalization coefficient, h_{max} is the diameter of the element K considered and ρ_K is the in-sphere radius. This quantity measures the shape of the element and is the so-called *element shape quality.*

3 Size map specification

The classical method, as briefly described above, is governed by the sole nature of the mesh of the boundary of the domain. To make the control more accurate, additional information regarding the desired sizes in some region of the domain need to be specified. Several solutions have been proposed to supply this information. We briefly recall the three most commonly used approaches.

3.1 Source points

In this approach, several source points are supplied inside the domain. A source point is defined by its coordinates and a value h, corresponding to the desired element size in its neighborhood. The relevant information can be taken into account in two ways

- either each source point becomes a mesh vertex and will contribute (via its h) to the mesh edge analysis (the classical method described above is used here),
- or each source, later possibly inserted as a mesh vertex, has an influence in an area defined by proximity.

The first approach is quite easy to implement and has been described in [7] (cf. the example given hereafter). However, the second approach is more tedious to implement and requires the determination of all regions of influence where the different sources interact one with each other.

3.2 Specific structure

In this case, the size specification is supplied by an adequate structure (usually very simple, a regular grid, an octree, etc,) and the way to use it. For example, if we consider an octree structure (cf. [3] or [18]), each octant codes the desired size in its neighborhood either via a value or directly via its size or even by supplying the octant vertices as mesh vertices. It is easy to realize that the resulting size is strongly dependant on the nature of the octree (depth level, ...). The reader is also refered to [13] where this technique is used within an advancing-front method.

3.3 Background mesh

In this approach, a so-called *background mesh* is supplied with sizes specifications at the vertices. This approach seems obvious, especially within a mesh adaption scheme. The background mesh at stage i is the mesh at stage $i - 1$ provided the mesh vertices contain the desired information (obtained via the error estimate, for instance). In the following sections, this approach is the one we retain (apart from the first test example).

4 Internal point construction

The same idea as the classical method is followed here. The current mesh edges are analyzed to construct the internal points. For a given edge, the distance between two consecutive points is set to one, this unit length being measured with respect to the size map. We precise now the notion of unit length.

4.1 Unit edge length

The sole lengths considered here are the edge lengths, evaluated with respect to the size map. Let AB be an edge and $h(t)$ be the size specification function (t varying between 0 and 1, such that $t = 0$ at point A and $t = 1$ at point B), the length of the straight segment AB is defined as (cf. [2])

$$l_{AB} = d_{AB} \int_0^1 \frac{1}{h(t)}\, dt \tag{8}$$

where d_{AB} represents the (usual) Euclidean distance of AB.

In our case, the function $h(t)$ is known in a discrete manner (*e.g.* at the background mesh vertices), thus the above formulation is approximated. Hence, the edge length AB is

- if $h(A)$ and $h(B)$ are the sole values known

$$l_{AB} = \frac{d_{AB}}{2}\left(\frac{1}{h(A)} + \frac{1}{h(B)}\right) \tag{9}$$

- or, if besides $h(A)$ and $h(B)$, the sizes $h(M_i)$ at n points M_i, $(i = 1, n)$, along AB are known, ($M_0 = A$ and $M_{n+1} = B$), then

$$l_{AB} = \sum_{i=0}^{i=n} l_{M_iM_{i+1}} \tag{10}$$

  in addition (cf. Formula (9))

$$l_{M_iM_{i+1}} = \frac{d_{M_iM_{i+1}}}{2}\left(\frac{1}{h(M_i)} + \frac{1}{h(M_{i+1})}\right). \tag{11}$$

The proposed method is based on the knowledge of the points M_i and the sizes h_i associated, the background mesh is used to extract this information.

4.2 Background mesh interpolation

For each edge (of the current mesh), the background mesh (denoted as $\mathcal{T}_F$) is used to extract the information related to the desired sizes. Let AB be a given edge, the intersection between AB and $\mathcal{T}_F$ is performed to find,

- which element of $\mathcal{T}_F$ contains A,
- which element of $\mathcal{T}_F$ contains B,
- every elements of $\mathcal{T}_F$ intersected by AB.

This operation is based on the algorithmic localisation of a point in the mesh. This type of approach works well only in a convex environment. Therefore, the implementation of this algorithm must be carefully and judiciously peformed, as $\mathcal{T}_F$ is not necessarily a convex domain.

At completion, the set of elements of $\mathcal{T}_F$ intersecting the edge AB is clearly identified as well as the set of points M_i (introduced above) and their relative sizes h_i, obtained using an interpolation based on the sizes h of the element of $\mathcal{T}_F$ including these points.

4.3 Edge analysis and point creation

The edge analysis consists in computing the edge length and in comparing this length to one. Let AB be the current edge and let M_i $(i = 1, n)$ be the set of points considered above, then l_{AB} is obtained using the Relationship (10). The construction of these points along AB is performed using the following algorithm

```
• $l = 0$.
• Loop for $i = 0, n$
    – $l = l + l_{M_i M_{i+1}}$
    – (A) - if $l > 1$
        ∗ introduce a point between $M_i$ and $M_{i+1}$ at a unit distance with the
          previous points ($M_0 = A$ to start),
        ∗ $l = l - 1$ back to(A).
• End loop.
```

The sizes h of the newly created points are computed using an interpolation between the sizes along the segment $M_i M_{i+1}$. After all edges have been processed, the set of points is filtered (to avoid conflicting points created along neighbouring edges).

The remaining points are then inserted using the Relationship (1) in the current mesh and the whole process of edge analysis is iterated, as long as edge lengths longer than 1 are encountered.

Notice that, in practice, the value 1 is not the exact value used in the computational scheme. Indeed, the edge length is usually not an integer value and thus cannot be exactly subdivided into unit length segments). Moreover, if an edge, which length is longer than 1 and smaller than a given threshold value, is splitted may lead to two edges having a length violating the requirement (instead of one edge in the original configuration).

5 Mesh optimization

The aim of this stage is to improve the edge lengths while preserving an acceptable element shape quality.

5.1 Shape optimization

This operation is similar to the classical case (cf. above) and involves the conventional set of mesh modifications.

5.2 Size optimization

This operation is a specific characteristic of the governed mesh generation scheme. The basic tools are similar to those of the classical case, at least formally speaking. The aim is, while preserving an acceptable element shape quality, to analyze the current mesh edges having an incompatible length (practically, the number of such edges is reduced, at least when the size map does not contain any dramatic variation). Three mesh modification tools are specifically used

- an edge colapsing operator to remove an edge too short by merging its two endpoints into one,
- an edge splitting operator to subdivide an edge too long into two sub-edges,
- a unit length node relocation operator.

The two first operators are classical (the only difference being the way to compute the edge lengths). The last operator can be written as, if P is the point to be relocated

$$\overline{P}_j = P_j + \frac{P_jP}{||P_jP||}\overline{h}_j \qquad (12)$$

where the P_j's are the points of the existing edges PP_j (the P_j's represent the vertices, different of P, of all tetrahedra sharing P) and $\overline{h}_j$ is defined such that $l_{PP_j} = 1$. Thus, we obtain the points $\overline{P}_j$, optimal poisitions of P respectively to the edges PP_j and a new average point is deduced which is used to define the new location of the current point P.

5.3 Efficiency coefficient

Let l_i be the length of the edge i with respect to the size map. Th efficiency coefficient of the mesh is defined as the average value of the squares of the differences to 1 of all mesh edge lengths (let na be the number of mesh edges), hence

$$\tau = 1 - \frac{\sum_{i=1}^{na} e_i^2}{na} \qquad (13)$$

with $e_i = 1 - l_i$ if $l_i < 1$ or $e_i = 1 - \frac{1}{l_i}$ if $l_i > 1$.

This coefficient seems adequate to quickly estimate the mesh quality with respect to a given size map. Table 1 reports the sensitivity of this measure, l being constant for all mesh edges (which is highly unlikely although it shows the effect of the size variation on τ) and indicates that the edge lengths are l times too long or too short (a value $l = 5$ or $l = 0.2$ means that all edges are 5 times too long or 5 times too short). The optimal value is $l = 1$ and practically, any value greater than 0.91 ensures a reasonable mesh quality with respect to the size map.

l	100	20	10	5	3	2	$\sqrt{2}$	1.3	1.2	1.1	1
τ	0.019	0.097	0.19	0.36	0.51	0.75	0.91	.9467	.9722	.9917	1.

Table 1: Sensitivity of the efficiency coefficient.

The reader is refered to this table in order to interpret the numerical results described hereafter.

5.4 Unit surface remeshing

Let assume the surface to be meshed is geometrically defined, either analytically or via an adequate mesh.

If the surface is directly known, the useful information can be extracted from the re-meshing process (using, for instance, queries to a geometric CAD modelling system). On the other hand, if the surface is defined from a mesh, the latter is used to construct a geometric support (cf. [8]). This support emulates, in some sense, the geometric modelling system and the information (normal, tangent plane, curvature, ...) can be extracted, too.

Let assume a given arbitrary mesh (which can be the mesh acting as support), we would like to introduce a method allowing to construct a new mesh with respect to a given property. This problem can be seen as a *remeshing* problem. The information required to construct such a mesh are related to

- the snapping of a point onto the surface,
- the knowledge of the surface geometry at a given point (radii of curvature, minimal of the principal radii of curvatures,...).

Therefore, we define several local operations allowing to

- create a point,
- remove a point,
- swap an edge.

These operation use almost the same local operators as the mesh optimization procedure. More precisely, from the topological point of view, the following tools are required

- edge swapping,
- "polygon" retriangulation (to remove a vertex).

Any surface can be remeshed using this basic set of tools. The main idea consists in remeshing so as to create unit edge lengths (using a metric related to the geometry) while preserving the element shape quality and the accuracy of the geometric approximation. Notice that we consider the remeshing of the ridges prior to any other remeshing operation.

6 Results

Several mesh examples are provided in this section to illustrate the governed mesh generation method. Notice that, the proposed method follows the general scheme introduced above and only the internal point creation and optimisation stages have been modified.

The first set of examples concerns the case where one and two source points have been introduced. The other examples are governed by a background mesh for which a size function is analytically specified at the vertices (in this way, we simulate a situation which is more realistic than if an analytical size function is used everywhere) . The first example assumes a constant boundary (the surface mesh respects the size map) as for the other examples, the surface needs to be remeshed at every stage of the iterative scheme (the last example concerns a non-convex domain).

6.1 Example 1 (constant boundary) with sources

Let consider two examples. The domain corresponds to a sphere of radius 1, centered at the origin. The surface, of constant mesh size, dictates an almost constant size of the order of 0.08.

A source point is supplied (Figure 1), defined by its coordinates, $(x = .25, y = .25, z = .50)$, and the size is set to $h = .025$. This point acts like an attractor. The mesh refinement is more important in the vicinity of this point. The second example (Figure 2) includes two source points, $(x = .25, y = .25, z = .50)$ and $(x = -.25, y = -.25, z = .50)$ and the sizes are respectively $h = .015$ and $h = .30$. The first point has an attractive influence as the second as a repulsive effect. A slice corresponding to the plane $z = .5$, is given for each example.

The mesh of the Figure 1 contains 171 151 tetrahedra, the mesh of the Figure 2 contains 107 404 tetrahedra. In these examples, the mesh generation speed is about 800 000 tets per minute (HP9000/C180).

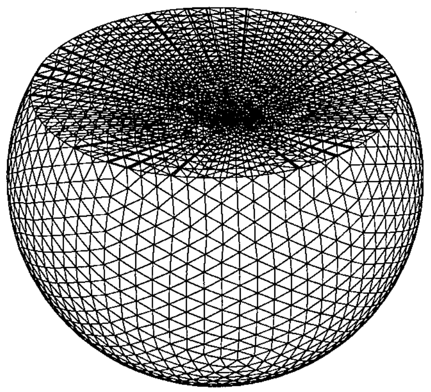
Figure 1: *Slice through the mesh at* $z = .5$ *(one source).*

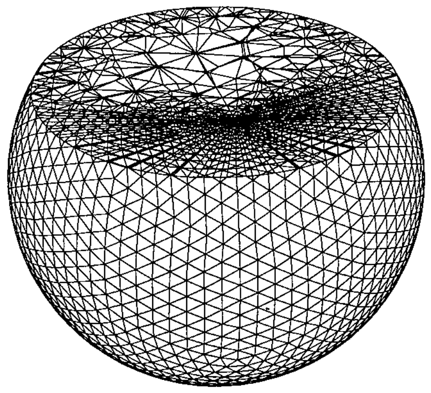
Figure 2: *Slice through the mesh at* $z = .5$ *(two sources).*

This approach, using source points as vertices, is pretty fast (the speed-up is the same as the classical mesh generation method). Although it presents some drawbacks, mainly related to

- the difficulty to supply the source points,
- the effect of these points (not exactly predictible),
- etc.

Hence, depending on the application, this approach can be either adequate or unpredictible.

6.2 Example 1 (constant boundary) with background mesh

The domain considered is a sphere of radius 1, centered at the origin. The size map is analytically defined using the following relationship

$$h(x, y, z) = 0.45 \times ||d - .15|| \times ||d - .65|| + 0.261$$

where d is the distance between the point (x, y, z) and the origin. Any slice through a plan $z = ct$ constant, shows two circles, centered at the intersection between the slice plane and the z axis.

The mesh generation speed (on these examples) ranges between $300,000$ and $450,000$ elements per minute (HP9000/C180).

In this relatively simple example (there is no large size variations), the convergence is quickly obtained (cf. Table 2, where *np*, *ne*, τ represent respectively the number of points, elements and the efficiency coefficient, *per* is the percentage of

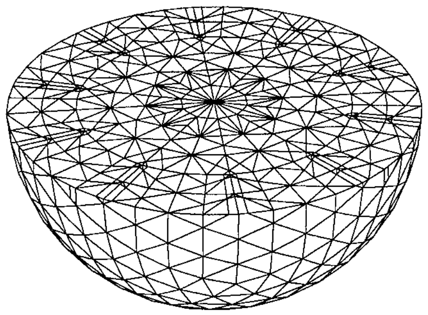

Figure 3: *Slice through the current mesh, $z = 0$.*

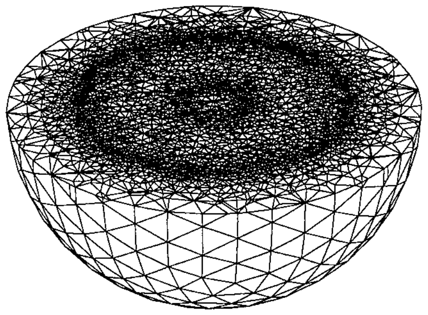

Figure 4: *Slice through the mesh at iteration 4, $z = 0$.*

-	np	ne	τ	per	Q	$1-2$	$2-3$
Initial mesh	1,663	8,340	.5857	3	2.11	98	1
Iteration 1	48,514	289,746	0.9719	94	4.97	96	2
Iteration 2	52,977	316,927	0.9749	97	4.47	97	2
Iteration 3	53,426	319,731	0.9755	97	4.99	97	2
Iteration 4	53,642	321,081	0.9759	98	4.89	97	2
Iteration 5	53,718	321,545	0.9761	98	3.90	97	2
Iteration 6	53,798	322,070	0.9763	98	4.26	97	2
Iteration 7	53,887	322,553	0.9765	98	4.11	97	2

Table 2: Statistics relative to the different iterations.

edges having a length between $\frac{1}{\sqrt{2}}$ and $\sqrt{2}$). Moreover, Q being the mesh quality, the column $1-2$ (resp. $2-3$) reports the percentage of elements having a shape quality between 1 and 2 (resp. 2 and 3). The result is almost stable after iteration 3.

6.3 Example 2 (free boundary) with background mesh

The domain considered is a simple sphere of radius 1, centered at the origin. The size map (sensibly more constrained than the previous example) is analytically defined as

$$h(x,y,z) = \min_{i=1,6}(h_i(x,y,z))$$

where

$$h_1(x,y,z) = 1.5 \times \|d_1 - 1.\| + 0.006$$

$$h_2(x,y,z) = 1.5 \times \|d_2 - 1.\| + 0.006$$
$$h_3(x,y,z) = 1.5 \times \|d_3 - 1.\| + 0.012$$
$$h_4(x,y,z) = 1.5 \times \|d_4 - 1.\| + 0.012$$
$$h_5(x,y,z) = 1.5 \times \|d_5 - 1.\| + 0.018$$
$$h_6(x,y,z) = 1.5 \times \|d_6 - 1.\| + 0.018$$

where d_1 is the distance between the points (x,y,z) and $(1.,0.,0.)$, d_2 is the distance to the point $(-1.,0.,0.)$, d_3 is the distance to the point $(0.,1.,0.)$, d_4 is the distance to the point $(0.,-1.,0.)$, d_5 is the distance to the point $(0.,0.,1.)$ and d_6 is the distance to the point $(0.,0.,-1.)$. Practically, this size map is only known at the vertices of the background mesh at the current iteration.

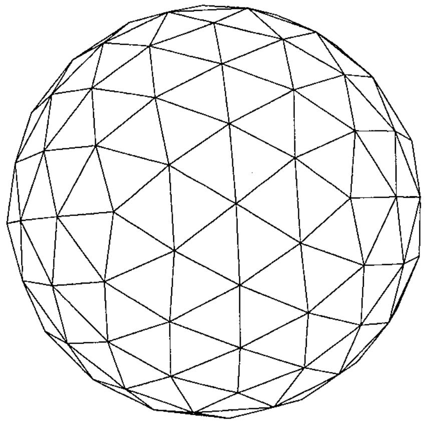
Figure 5: *Initial surface mesh (*122 *vertices and* 240 *triangles).*

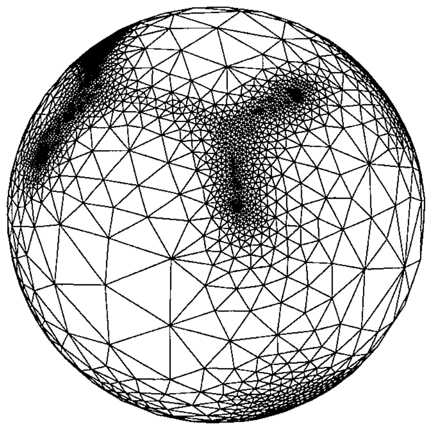
Figure 6: *Surface mesh at iteration 1 (*3,489 *vertices and* 6,974 *triangles).*

We consider, at first, the evolution of the surface mesh relatively to the iterations (Figures 5 to 12). The Figures show the effect of the background mesh on the capture of the controlling size map. The mesh of the Figure 5 missed the size field, indeed, this mesh has been generated without any knowledge of the size map. The mesh of the Figure 6 does not capture the size map and also is locally over refined. This phenomenon can be explained by the existence of edges having size specifications too small at the endpoints and no additional information is supplied by the background mesh. Hence, these edges are subdivided in too many segments. The mesh of the Figure 7 is a fairly good mesh with respect to the size map, although it presents locally a lack of mesh density. The final mesh (Figure 8) is a satisfactory mesh.

The Figures 9 to 24 show several slices through the volumetric mesh at different iterations.

The following table reports the statistics relative to the diffrent iterations with the parameters introduced in Table 2.

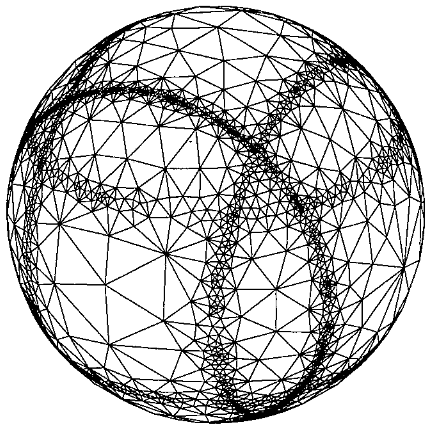

Figure 7: *Surface mesh at iteration 3 (*3, 964 *vertices and* 7, 924 *triangles).*

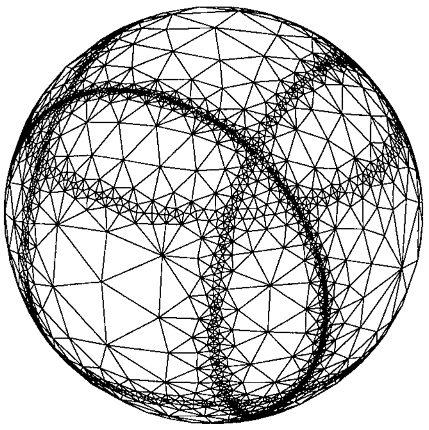

Figure 8: *Surface mesh at iteration 7 (*4, 766 *vertices and* 9, 528 *triangles).*

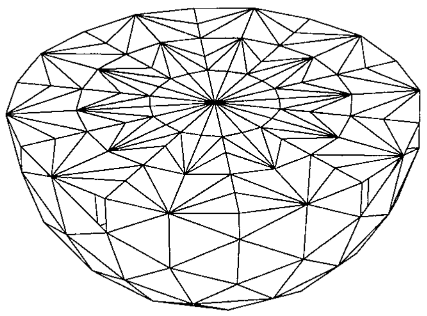

Figure 9: *Slice through the initial mesh,* $z = 0$.

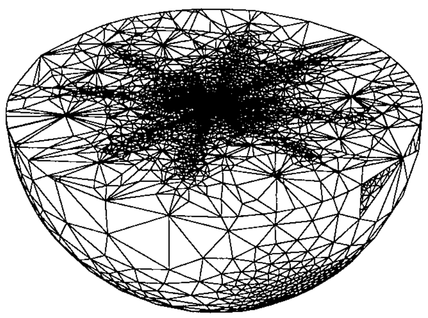

Figure 10: *Slice through the mesh at iteration 1,* $z = 0$.

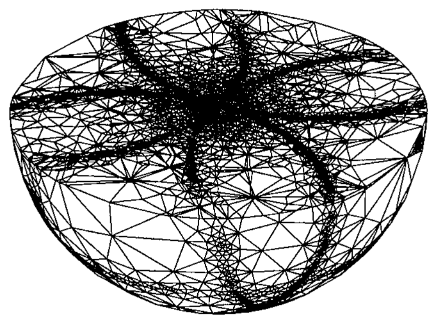

Figure 11: *Slice through the mesh at iteration 3,* $z = 0$.

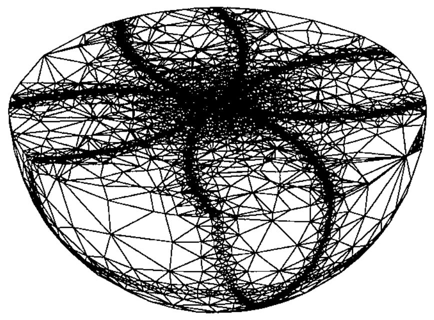

Figure 12: *Slice through the mesh at iteration 7,* $z = 0$.

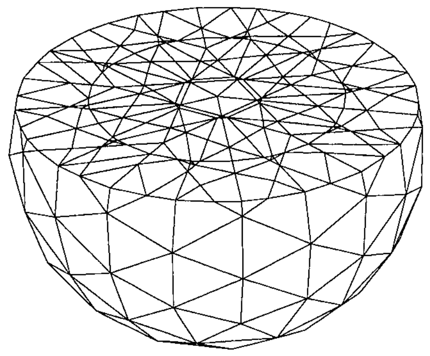

Figure 13: *Slice through the initial mesh,* $z = 0.25$.

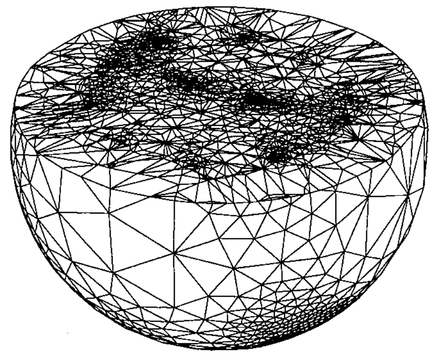

Figure 14: *Slice through the mesh at iteration 1,* $z = 0.25$.

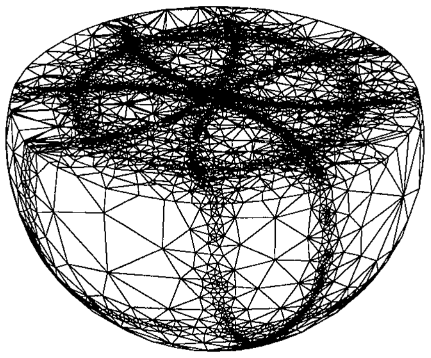
Figure 15: *Slice through the mesh at iteration 3,* $z = 0.25$.

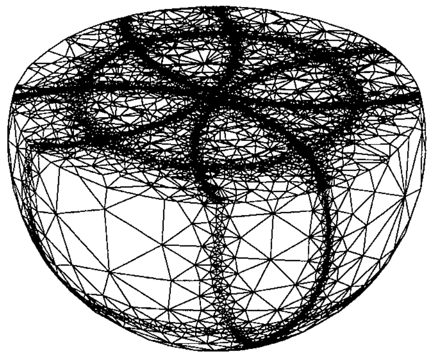
Figure 16: *Slice through the mesh at iteration 7,* $z = 0.25$.

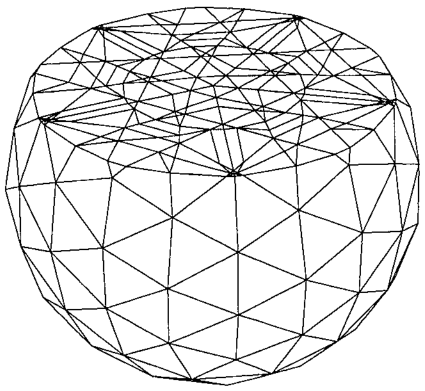
Figure 17: *Slice through the initial mesh,* $z = 0.50$.

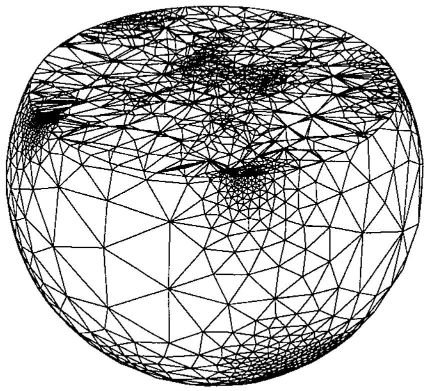
Figure 18: *Slice through the mesh at iteration 1,* $z = 0.50$.

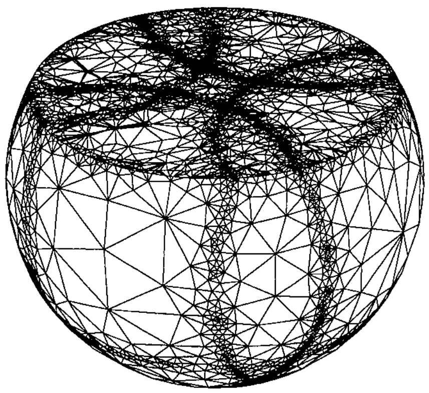
Figure 19: *Slice through the mesh at iteration 3,* $z = 0.50$.

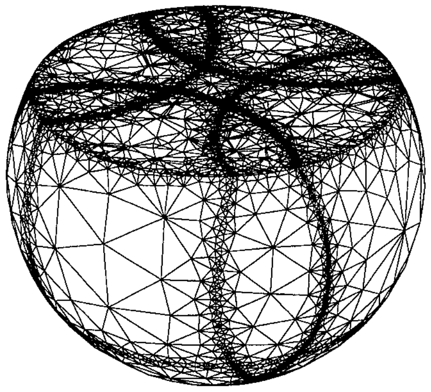
Figure 20: *Slice through the mesh at iteration 7,* $z = 0.50$.

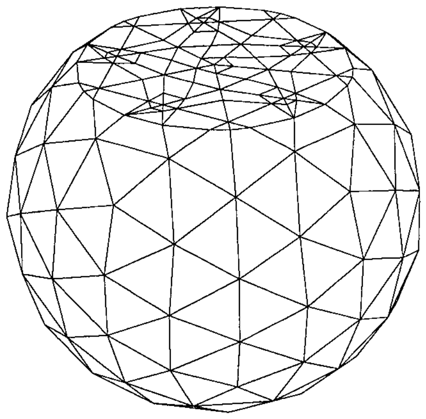
Figure 21: *Slice through the initial mesh,* $z = 0.75$.

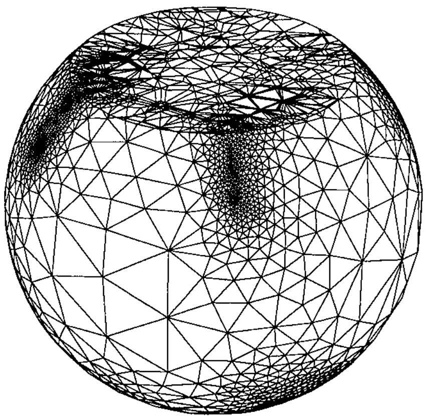
Figure 22: *Slice through the mesh at iteration 1,* $z = 0.75$.

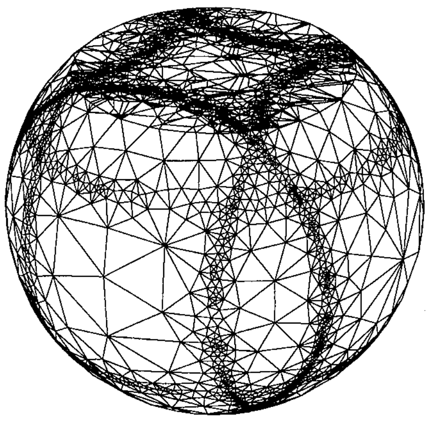
Figure 23: *Slice through the mesh at iteration 3,* $z = 0.75$.

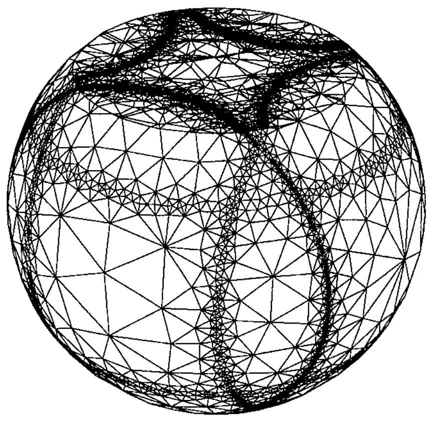
Figure 24: *Slice through the mesh at iteration 7,* $z = 0.75$.

-	np	ne	τ	per	Q	$1-2$	$2-3$
Initial mesh	277	1,200	0.515	7	1.84	100	-
Iteration 1	23,023	124,362	0.814	37	47.	81	11
Iteration 3	115,215	647,119	0.9448	78	12.	78	20
Iteration 7	253,068	1,416,617	0.9616	86	8.	74	24

Table 3: Statistics relative to the different iterations.

The mesh generation speed ranges from $200,000$ to $350,000$ elements per minute (HP9000/C180).

6.4 Non-convex example (free boundary) with background mesh

This example corresponding to a mechanical device (data courtesy of the Mac Neal-Schwendler Corp., USA) is representative of a class of problems (non-convex, arbitrary complex shape, thin sections,...). The controlling size map is analytically defined, and constructed from the bounding box (a parallelepiped), a size variation is specified along six spheres centered in the middle of the faces of the box. The desired size increases as the distance from the spheres increases.

The Figure 25 shows the initial triangulation of the surface of this object and the Figures 26 to 28 show the meshes corresponding to the iterations 1 to 3 of the adaption loop.

The four last figures show slices for a plane $x + y + z + 0.19 = 0$ through the volumetric meshes at iterations 0 (original mesh, not adapted) and 3. Table 4

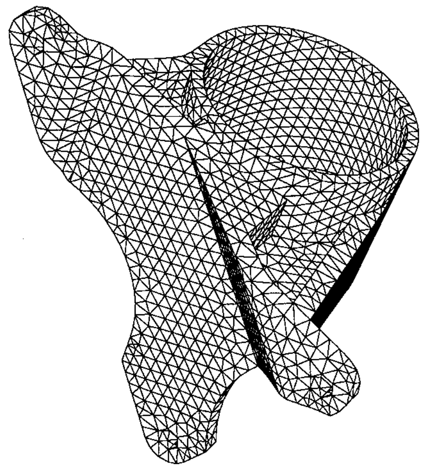

Figure 25: *Initial surface mesh (data courtesy of MSC)*

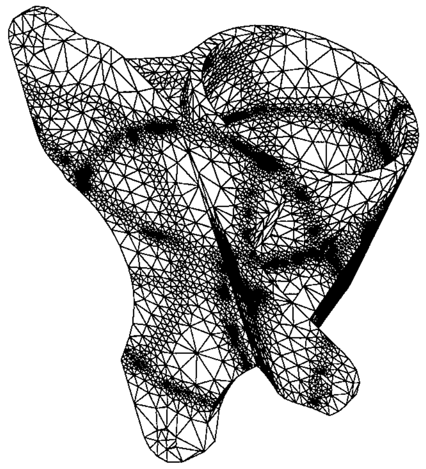

Figure 26: *Surface mesh at iteration 1.*

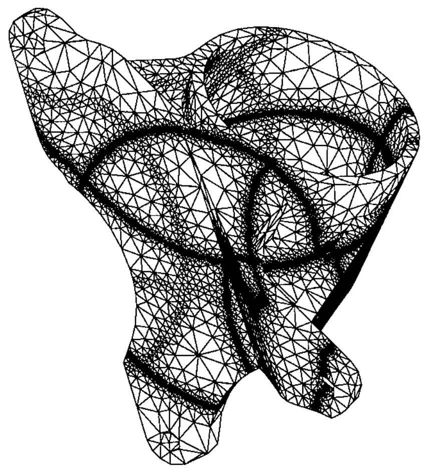

Figure 27: *Surface mesh at iteration 2.*

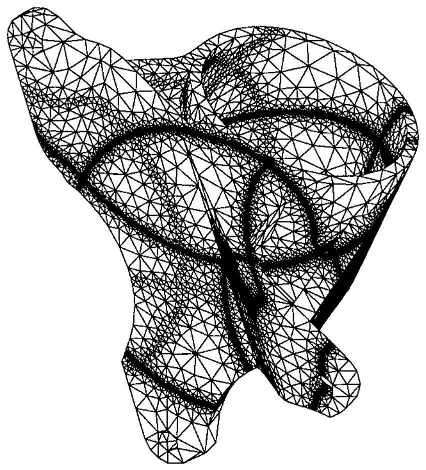

Figure 28: *Surface mesh at iteration 3.*

reports the statistics about the meshes.

-	np	ne	τ	per	Q	$1-2$	$2-3$
Initial mesh	3,703	12,993	0.784	33	5.63	95	3
Iteration 1	26,701	126,296	0.905	61	7.60	90	8
Iteration 2	59,398	306,830	0.959	86	7.69	89	10
Iteration 3	70,292	366,798	0.967	91	11.00	89	10
Iteration 4	76,888	405,436	0.971	94	11.51	90	9

Table 4: Statistics relative to the mechanical device 'lhmount'

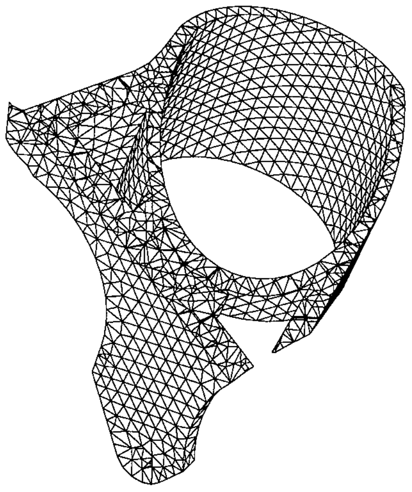
Figure 29: *Initial volumetric mesh.*

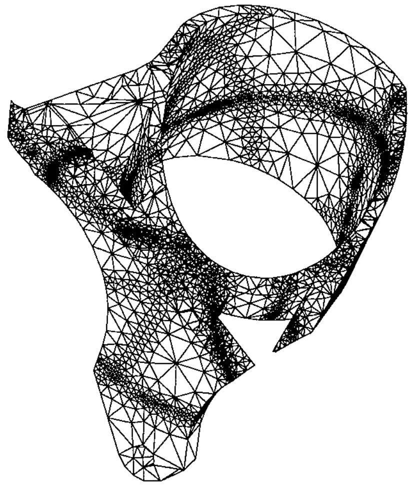
Figure 30: *Volumetric mesh at iteration 1.*

The mesh generation speed is about $300,000$ elements per minute (HP9000/C180). The results seem reasonable, as compared to the statistics and the related figures. This indicates that no extra difficulty is expected to process a realistic object with respect to the academic example.

7 Extensions and conclusions

The validation of the proposed mesh generation method is ongoing, especially for industrial application examples. In particular, solid mechanics problems have to be investigated, within a complete mesh adaption loop (mesh generation, numerical computation, error estimate) where the size function is obtained uaing the error estimate (and not through an analytical expression). In addition, the proposed

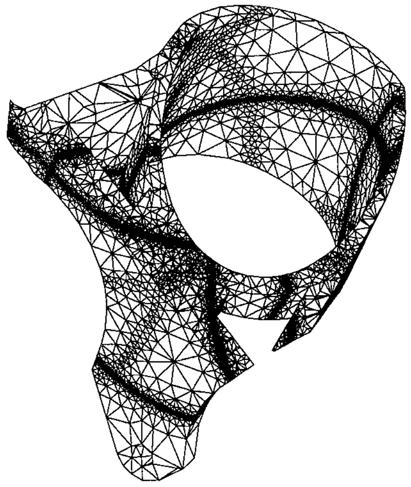
Figure 31: *Volumetric mesh at iteration 2.*

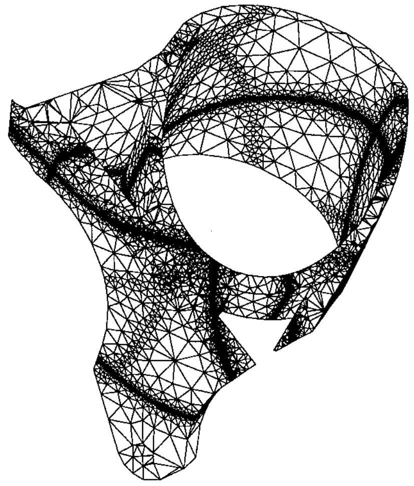
Figure 32: *Volumetric mesh at iteration 3.*

approach, including the whole reconstruction of the mesh at each iteration, has to be compared with local mesh modification methods.

From the technical point of view, it seems important to improve the mesh generation speed by at least a factor 2, to match the classical mesh generation speed.

Finally, the extension to the generation of anisotropic meshes is obvious, especially by changing the notion of unit length and by generalizing the Delaunay kernel (cf. [2]).

References

[1] T.J. BAKER, Generation of tetrahedral meshes around complete aircraft, *Numerical grid generation in computational fluid mechanics'88*, Miami, 1988.

[2] H. BOROUCHAKI ET P.L. GEORGE, Triangulation de Delaunay et métrique riemannienne. Applications aux maillages éléments finis, *Revue européenne des éléments finis* **5**(3), 323-340, 1996.

[3] J.H. CHENG, P.M. FINNIGAN, A.F. HATHAWAY, A. KELA AND W.J. SCHOEDER, Quadtree/octree meshing with adaptive analysis, *Numerical grid generation in computational fluid mechanics'88*, Miami, 1988.

[4] P.G. CIARLET, *The finite element method for elliptic problem*, North Holland, 1978.

[5] P.L. GEORGE, *Automatic mesh generation. Applications to finite element methods*, Wiley, 1991.

[6] P.L. GEORGE, Automatic Mesh Generation and Finite Element Computation, in Handbook of Numerical Analysis, vol IV, Finite Element methods (Part 2), Numerical Methods for Solids (Part 2), P.G. Ciarlet and J.L. Lions Eds, North Holland, 69-190, 1996.

[7] P.L. GEORGE, Improvement on Delaunay based 3D automatic mesh generator, *Finite Elements in Analysis and Design*, **25**(3-4), 297-317, 1997.

[8] P.L. GEORGE ET H. BOROUCHAKI,*Triangulation de Delaunay et maillage. Applications aux éléments finis*, Hermes, Paris, 1997.

[9] P.L. GEORGE, F. HECHT AND E. SALTEL, Automatic mesh generator with specified boundary, *Comp. Meth. in Appl. Mech. and Eng.*, **92**, 269-288, 1991.

[10] P.L. GEORGE, F. HECHT AND M. G. VALLET, Creation of internal points in Voronoi's type method, Control and adaptation, *Adv. in Eng. Soft.*, **13**(5/6), 303-313, 1991.

[11] P.L. GEORGE, F. HERMELINE, Delaunay's mesh of a convex polyhedron in dimension d. Application to arbitrary polyhedra, *Int. Jour. Num. Meth. Eng.*, **33**, 975-995, 1992.

[12] R. LÖHNER, P. PARIKH, Generation of 3D unstructured grids by advancing front method, *AIAA 26 Aerospace Sciences meeting*, Reno Nevada, 1988.

[13] A. RASSINEUX, Maillage automatique tridimensionnel par une méthode frontale pour la méthode des éléments finis, PhD Thesis, Nancy I, 1995.

[14] M.S. SHEPHARD, F. GUERINONI, J.E. FLAHERTY, R.A. LUDWIG AND P.L. BAEHMANN, Finite octree mesh generation for automated adaptive 3D flow analysis, *Numerical grid generation in computational fluid mechanics'88*, Miami, 1988.

[15] M.G. VALLET, Génération de maillages éléments finis anisotropes et adaptatifs, PhD Thesis, Paris 6, 1992.

[16] D.F. WATSON, Computing the n-dimensionnal Delaunay tesselation with applications to Voronoi polytopes, *Computer Journal* **24**(2), 167-172, 1981.

[17] N.P. WEATHERILL,The integrity of geometrical boundaries in the 2-dimensional Delaunay triangulation, *Comm. in Appl. Num. Meth.*, **6**, 101-109, 1990.

[18] M.A. YERRI, M.S. SHEPHARD, Automatic 3D mesh generation by the modified-octree technique, *Int. Jour. Num. Meth. Eng.*, **20**, 1965-1990, 1984.