Anisotropic Surface Mesh Generation

Manolo Castro-Diaz, Frédéric Hecht

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Anisotropic Surface Mesh Generation

M.J. Castro Díaz and F. Hecht

N° 2672

Octobre 1995

PROGRAMME 6

ISSN 0249-6399

INRIA
ROCQUENCOURT

Anisotropic Surface Mesh Generation

M.J. Castro Díaz* and F. Hecht

Programme 6 — Calcul scientifique, modélisation et logiciel numérique
Projet MENUSIN

Rapport de recherche n° 2672 — Octobre 1995 — 31 pages

Abstract: Anisotropic adapted unstructured triangular surface grids are generated using a new and efficient procedure based upon the Delaunay principle. The mesh anisotropy is controlled by a change of metrics over the surface. In order to improve the surface grid definition, a generalization of the Farin's algorithm has been developed to obtain 'G^1' surfaces by joining triangular Bézier surface patches. Different local mesh operators are proposed to control the surface grid generation process and a remeshing anisotropic grid generation algorithm is described. This algorithm has been developed keeping in mind its 3D generalization.

Key-words: Surface Interpolation, Anisotropic Mesh Generation, Anisotropic Surface Mesh Generation, Finite Elements.

(Résumé : tsvp)

*Dpto. de Análisis Matemático, Universidad de Málaga, Campus Universitario de Teatinos s/n, 29080 Málaga.

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
Téléphone : (33 1) 39 63 55 11 – Télécopie : (33 1) 39 63 53 30

Génération de Maillages Anisotropes Triangulaires Surfaciques

Résumé : Nous présentons ici une méthode nouvelle et efficace, basée sur le principe de Delaunay, pour générer des maillages de surfaces adaptées et composées de triangles anisotropes non structurés. L'anisotropie des maillages est contrôlée par un changement de métrique de la surface. Dans le but d'optimiser la définition du maillage, nous développons une généralisation de l'algorithme de Farin, ce qui nous a conduit à des surfaces G^1 en joignant des patchs triangulaires de surfaces de Bézier. Nous proposons différents opérateurs locaux pour contrôler cette génération de maillages, et un algorithme général de remaillage anisotropique. Sa conception tient compte d'une future généralisation tridimensionnelle.

Mots-clé : Interpolation de Surfaces, Génération des Maillages Anisotropes, Génération Anisotropes de Maillages Surfaciques, Éléments Finis.

Contents

List of Figures

1 Introduction

Two major advantages of unstructured grids over structured grids are their applicability to complex geometries and easy incorporation for grid adaptation (see [10], [11], [22],[23]). The introduction of mesh anisotropy minimizes the number of grid elements if the simulated physical phenomena are strongly directional (see [18] and [22]) as shocks and boundary layers in fluid mechanics. A new algorithm to generate *anisotropic adapted unstructured triangular surface grids* has been developed. The generation procedure is fast, taking a few minutes to generate large grids when run in workstations.

A 3D anisotropic mesh adapted generation algorithm consists in:

1. the computation of the anisotropic adapted mesh criterion;
2. the generation of the anisotropic adapted surface grid;
3. the generation of the anisotropic adapted field grid.

In this paper only the description of the anisotropic adapted surface grid generation algorithm is undertaken. In future works the anisotropic adapted mesh criterion associated to an *a posteriori error estimator* and the anisotropic adapted field grid generation will be studied

Let S be a surface and $\mathcal{T}_0$ a mesh of S, both input data and let $\mathcal{M}$ be a mesh criterion over $\mathcal{T}_0$. The problem addressed in this paper is the generation of a mesh, $\mathcal{T}_1$, from $\mathcal{T}_0$ so that it is 'well adapted' with respect to the criterion $\mathcal{M}$ and 'close to' the initial surface S. In general, the exact definition of S is only known at the C.A.D step of the process (when generating the surface) and not easily accessible. For this reason, an approximation S_1 of the surface S is constructed from an initial mesh, $\mathcal{T}_G$ (possibly $\mathcal{T}_G = \mathcal{T}_0$), using a generalized Farin's algorithm ([4], [9], [12]). Thus the problem becomes:

Generate a new mesh $\mathcal{T}_1$ so that it is 'well adapted' with respect to the criterion $\mathcal{M}$ and 'close to' the approximated surface S_1, constructed from $\mathcal{T}_G$ using a generalized Farin's algorithm.

Therefore three meshes must be considered at the same time: the first one ($\mathcal{T}_G$) is used to define the approximated surface S_1; the second one ($\mathcal{T}_0$) is the mesh to be adapted and the last ($\mathcal{T}_1$) is the resulting adapted mesh.

In section 2, two different kinds of mesh generation *control spaces* are considered: those in which only the size of elements is taken into account (the shape of the elements is imposed to be equilateral), and those in which indications about size and shape of elements can be given as data. Size and shape of elements is controlled by a change of metric over the whole domain as in [22]. In section 3, Farin's algorithm is outlined as well as some applications. A generalized Farin's algorithm is used to construct a 'G^1' approximation of S_1 (see definition 4) from the initial mesh $\mathcal{T}_G$. In order to introduce a new point, x, on the approximate surface S^1 constructed from $\mathcal{T}_G$, the element $K_G \in \mathcal{T}_G$ containing x or its projection must be found (the same issue arises when computing the metric at x). In section 4, two different projection algorithms are proposed. In section 5, two different kinds of local mesh operators are considered: those used to suppress or add mesh items (vertices, edges and triangles) and those which improve the quality of the grid. Finally, a grid generation algorithm for remeshing anisotropic adapted surfaces is proposed in section 6, as well as different applications.

2 Mesh generation control space

During automatic mesh generation it is necessary to give some indications, depending on the future use of the mesh. These indications define *the control space* which governs the grid generation. For example, the user can specify the length of the edges over the different subdomains, the shape of the elements, etc. Two different kinds of control spaces can be considered depending on the nature of the user's indications. Nevertheless, we are mainly interested in those including element's shapes and sizes.

2.1 Isotropic control space. Refinement function

Let Ω be a $\mathbb{R}^n$ connected domain. Any function $h: \Omega \mapsto \mathbb{R}^+$ giving the size of the elements with respect to their position is called a *refinement function*. This function may represent an average length of the edges having x as ending point, or the diameter of an element with x as vertex.

The construction of the mesh from a refinement function can be regarded as the following optimization problem:

Find a triangulation $\mathcal{T}_h$ of Ω, so that the function $H_{\mathcal{T}_h}: \mathcal{T}_h \longrightarrow \mathbb{R}^+$ where $H_{\mathcal{T}_h}(K)$ is given by the diameter of K, is 'close to' the given function h.

Some restrictions may also be considered, as the absence of obtuse angles, the existence of n fixed points into the mesh, ... This problem is not a classical optimization problem as in [5] because the mesh set is not a vector space. Instead, mesh generation may be considered as a combinatorial optimization problem: a triangulation is transcribed under its graph format. Thus, the problem becomes:

Find an heuristic that modifies the graph of a triangulation to minimize its associated cost-function.

When controlling the mesh generation just by using a refinement function, there exists an implicit condition about the shape of the elements: it must be close to the equilateral shape. In this case, no directions are preferred. This type of control space is called **isotropic control space**. On the other hand, control spaces where the element stretching is specified are called **anisotropic control spaces**.

2.2 Anisotropic control space. Metric tensor

This study is not restricted to equilateral triangles, i.e. the control space may contain size and stretching element indications. Therefore, at every point $x \in \Omega \subset \mathbb{R}^2$, three parameters (six in $\mathbb{R}^3$) must be given. This kind of control is equivalent to an isotropic control imposing a metric tensor all over the domain. For the sake of simplicity, we consider in this section that $\Omega \subset \mathbb{R}^2$. The same results are obtained if Ω defines a surface or if $\Omega \subset \mathbb{R}^3$.

It is easy to prove that if K_0 is a non degenerated element then, there exists an unique metric tensor $\mathcal{M}$ where K_0 is equilateral with edges of length unity. Thus, in order to obtain triangles with the same stretching and size over a subdomain, it suffices to construct an isotropic mesh with respect to using the metric tensor $\mathcal{M}$ with associated refinement function, h, equal to 1.

Only metrics induced by a scalar product are considered. In that case $\mathcal{M}$ is identified with its scalar product matrix. For example, in $2D$ domains, $\mathcal{M}$ is given, by

$$\mathcal{M} = \mathcal{R} \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \mathcal{R}^{-1}, \tag{1}$$

where $\lambda_1 > 0$ and $\lambda_2 > 0$ are the $\mathcal{M}$ eigenvalues and $\mathcal{R}$ is the rotation matrix of angle α that maps the canonical basis in $\mathbb{R}^2$ over the unit eigenvectors of $\mathcal{M}$. Hence, defining $\mathcal{M}$ is equivalent to give the parameters $(\alpha, \lambda_1, \lambda_2)$. In this case, the control space is constituted by the function $\mathcal{M}: \Omega \mapsto \mathbb{R}^3$, where $\mathcal{M}(x)$ is the metric at point x. The given data set $\{\mathcal{M}(x),\ x \in \Omega\}$ defines a Riemanian metric over the variety $\Omega \subset \mathbb{R}^2$. $\mathcal{M}$ is a covariant tensor of degree 2 and $\mathcal{M}(x)$ is positive definite, continuous for all $x \in \Omega$. Therefore, $\mathcal{M}$ is a metric tensor over Ω and the variety Ω becomes a $\mathbb{R}^2$ Riemanian variety.

Elementary differential geometry says that the length of a parametric curve $\Gamma(t)$, $t \in [0,1]$ with respect to the metric tensor $\mathcal{M}(x)$ is defined by:

$$L(\Gamma) = \int_0^1 \sqrt{\Gamma'(t)\mathcal{M}(\Gamma(t))\Gamma'(t)}dt. \tag{2}$$

Up to now, two different control spaces have been presented: those composed by a refinement function and those constituted by a Riemanian metric tensor. In both cases constraints as 'maximal number of vertex connectivities', minimal angles, etc..., can be considered.

Remark 1 *Isotropic control is a particular case of anisotropic control. It suffices to consider the metric tensor:*

$$\mathcal{M}(x) = \frac{1}{h^2(x)} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \forall x \in \Omega, \tag{3}$$

where $h: \Omega \mapsto \mathbb{R}^+$ is the refinement function. This Riemanian metric over Ω is constructed so that an equilateral element containing x, with edge length equals to 1 with respect to this metric, is an equilateral element with edge length $h(x)$ in the canonical Euclidean metric.

The following discretization of the metric tensor $\mathcal{M}$ is then proposed. Suppose that $\mathcal{M}$ is given at the vertex mesh

$$\{\mathcal{M}(x_i),\ i = 1, \ldots, n_v\},$$

where $\{x_i,\ i = 1, \ldots, n_v\}$ is the $\mathcal{T}_h$ vertex set. Over each mesh triangle, K_0, the $\mathcal{M}$ coefficients are linearly interpolated, therefore a continuous discretization of the metric tensor $\mathcal{M}$ is obtained.

Let $\Gamma = [x_0, x_1]$ be a segment so that $\Gamma \subset K_0$. The variation of $\mathcal{M}$ over Γ is linear and a possible parametrization of Γ is $\Gamma(t) = x_0 + t(x_1 - x_0)$, $t \in [0,1]$. Computing its length using equation (2), we obtain

$$\begin{aligned} L(\Gamma) &= \int_0^1 \sqrt{(\Gamma')^T \mathcal{M}(\Gamma(t))\Gamma'}dt & (4) \\ &= \int_0^1 \sqrt{l_0^2 + t(l_1^2 - l_0^2)}dt & (5) \\ &= \frac{2}{3}\frac{l_0^2 + l_0 l_1 + l_1^2}{l_0 + l_1}, & (6) \end{aligned}$$

where $l_i = \sqrt{(\Gamma')^T \mathcal{M}(x_i)\Gamma'}$, $i = 0, 1$.

This integration formula is exact when $\mathcal{M}$ is linear over Γ. Otherwise, Γ can be decomposed into a finite number of segments, each contained in a mesh triangle. Hence, its length can be computed as a finite sum of terms which are expressed as in equation (6). This last decomposition is numerically expensive, thus formula (6) will be always used as an approximation of the length of an arbitrary edge.

Remark 2 *Another formula to compute the length of an edge can be considered. In fact, the previous formula is not optimal when there exits large variations in the metric definition, i.e., when the metric is not continuous as in the great majority of real mesh adaptation problems. In that case the following formula can be used*

$$L(\Gamma) = \begin{cases} \frac{l_0\, l_1}{l_0 - l_1} \log(\frac{l_0}{l_1}) & \text{if } l_0 \neq l_1 \\ l_0 & \text{if } l_0 = l_1, \end{cases} \tag{7}$$

where $l_i = \sqrt{(\Gamma')^T \mathcal{M}(x_i)\Gamma'}$, $i = 0, 1$. *This formula is exact supposing that the metric is isotropic and considering linear interpolation of the refinement function* $h(x)$ *and not of metric coefficients. In that case the length of the segment* Γ *is given by*

$$\begin{aligned} L(\Gamma) &= \int_0^1 (l_0^{-1}(1-t) + l_1^{-1})^{-1} t \; dt \\ &= \int_0^1 \frac{l_0\, l_1}{l_0 t + l_1(1-t)} \; dt \\ &= \frac{l_0\, l_1}{l_0 - l_1} \log(\frac{l_0}{l_1}). \end{aligned}$$

This formula will be also used in anisotropic mesh adaptation. In that case, to be coherent with the previous formula, the metric at a point $p \in K_0$ *is computed by*

$$\mathcal{M}(p) = \left(\sum_{i=1}^{3} \lambda_i(p) M(x_i^0)^{-1} \right)^{-1},$$

with $\lambda_i(p)$, $i = 1, 2, 3$ *the barycentric coordinates of the point* p *with respect to triangle* K_0 *and* x_i^0, $i = 1, 2, 3$ *the vertices of* K_0.

3 'C.A.D. reconstruction'

Let $\mathcal{T}_G$, a triangulation of the surface S be given. This triangulation defines a polygonal surface S_0. We are interested in remeshing it by adding, suppressing and moving points. In these three processes, points (vertices) must be added to the given surface. The continuous polygonal surface defined by the triangulation $\mathcal{T}_G$ may be used, but is it possible to do better? Could we obtain 'a better definition' of the initial surface S using 'G^1' interpolation over $\mathcal{T}_G$? Farin's algorithm (see [4], [8] and [12]), which was primarily developed to obtain 'G^1' surfaces (see definition 4 below) by connecting triangular Bézier surface patches can be applied to this purpose. This algorithm solves the following problem:

Given a triangulation $\mathcal{T}_G$ *of a surface* S, *find an approximated '*G^1*' surface,* S_1, *easy to construct and as close as possible to* S.

Farin gave a 'G^1' interpolation scheme using Bernstein polynomials, which are considered a modified HCT interpolation [15]. This algorithm gives an interpolating function over each triangle $K_0 \in \mathcal{T}_G$ by subdividing K_0 into 3 subtriangles. This interpolating function is expressed by 31 Bézier control points over K_0, and the problem becomes the construction of suitable control points over the triangles of $\mathcal{T}_G$.

Farin's algorithm, which solves the previous problem supposing that the initial surface is 'sufficiently smooth', will be presented. More general algorithms can be derived from this initial one, in order to consider surfaces with sharp edges (see [4] and [12]). But first, let us introduce some concepts.

Definition 1 (Bézier surface patch) *The I-th Bernstein polynomial of degree n in two variables (note that $u_1 + u_2 + u_3 = 1$) is defined by:*

$$B_I^n(\mathbf{u}) = \frac{n!}{i_1! i_2! i_3!} u_1^{i_1} u_2^{i_2} u_3^{i_3}, \quad \mathbf{u} = (u_1, u_2, u_3), \tag{8}$$

with

$$I = (i_1, i_2, i_3),\ n = |I| = i_1 + i_2 + i_3,\ u_1 + u_2 + u_3 = 1. \tag{9}$$

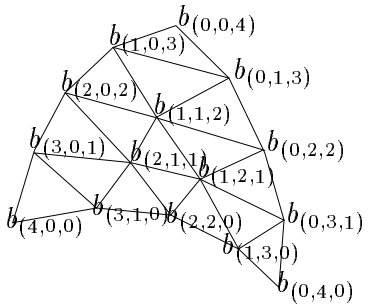

Figure 1: Control points for Bézier surface patch of order $n = 4$.

Then **a triangular Bézier surface patch of order** n with *control points* b_I, $|I| = n$ in $\mathbb{R}^3$ is defined by (see fig.1)

$$S_B = \{S_b(\mathbf{u}) = \sum_{|I|=n} b_I B_I^n(\mathbf{u}) :\ u_i \geq 0,\ \sum_{i=1,\ldots,3} u_i = 1\}. \tag{10}$$

This surface patch can be regarded as a mapping

$$S_b \colon K_0 \longrightarrow \mathbb{R}^3, \quad p \in K_0 \mapsto S_b(\mathbf{u}(p)) \in \mathbb{R}^3,$$

from a triangle K_0 into $\mathbb{R}^3$, with $\mathbf{u}(p)$ the barycentric coordinates of the point $p \in K_0$ into $\mathbb{R}^3$.

The three control points $b_{(n,0,0)}$, $b_{(0,n,0)}$ and $b_{(0,0,n)}$ are called the vertices of the patch S_B. Note that the surface patch, S_B, passes through the vertices and not necessarily through the other control points.

Definition 2 (Bézier surface) *A surface defined as the union of triangular Bézier surface patches over a given mesh is called a Bézier surface over the mesh. If all the surface patches are of order n, then the surface is said to be of order n.*

Definition 3 (Boundary curves) *The boundary of a triangular Bézier patch is defined by*

$$\bigcup_{i=1}^{3}\{S_b(\mathbf{u}):\ u_i=0,\quad u_j\geq 0,\quad \sum_{j=1,\dots,3} u_j=1\}.$$

So each k-th boundary edge ($k=1,\dots,3$) is a Bézier curve of order n and depends on the control points b_I with $i_k=0$

Definition 4 (Visually continuous or 'G^1'-regularity [6]) *Let ϕ and φ be two surface patches that have a common boundary curve Γ, and let $\Gamma'(v)$ be its tangent vector at point $\Gamma(v)$. Let $D^1_{d_1}\phi(v)$ be a cross-boundary derivative of ϕ at $\Gamma(v)$, i.e. $D^1_{d_1}\phi(v)$ lies in the tangent plane of ϕ at $\Gamma(v)$ and is not colinear with $\Gamma'(v)$. Analogously, we define a cross-boundary derivative $D^1_{d_2}\varphi(v)$. Now, the condition of 'G^1'-continuity is*

$$det\left[D^1_{d_1}\phi(v), D^1_{d_2}\varphi(v), \Gamma'(v)\right]=0.$$

Let us state now two propositions that will be used for the construction of 'G^1' Bézier surfaces (more details can be found in [4], [6] and [12]).

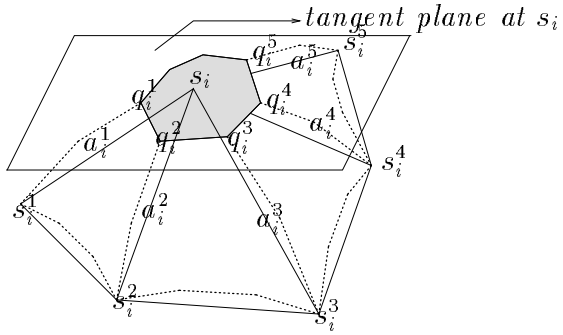

Figure 2: Control points around s_i

Proposition 1 (Punctual 'G^1'-continuity) *If, for each triangle K containing s_i as vertex, all the control points around this vertex s_i are coplanar then the Bézier surface is 'G^1' at s_i and the plane of coplanarity will be the common tangent plane at s_i, for all the surface patches containing s_i (see fig.2).*

Proposition 2 (Inter-element 'G^1'-continuity) *Let S_{b_1} and S_{b_2} be two adjacent patches of a continuous Bézier surface of order $n+1$. Let Γ be their common boundary curve (of order $n+1$) and suppose Γ can be considered as a Bézier curve of order n. Let b_0, $b_1,\dots,b_n$ be the control points of Γ as a curve of order n and let b_0^*, $b_1^*,\dots,b_n^*$ (respectively $\bar{b}_0$, $\bar{b}_1,\dots,\bar{b}_n$) be the adjacent control points to Γ of S_{b_1} (respectively S_{b_2}) (see fig 3). Then, a sufficient condition for the 'G^1'-continuity of the Bézier surface across Γ is:*

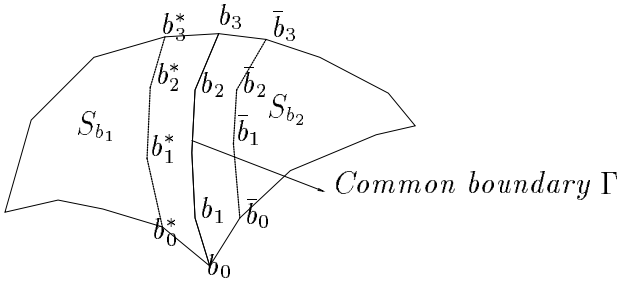

Figure 3: Control points for 'G^1'-continuity ($n = 3$).

The existence of $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ *and* $\alpha \in \mathbb{R}$ *verifying,* $\alpha_1+\alpha_2+\alpha = 1$ *and* $\alpha_3+\alpha_4+\alpha = 1$, *such that,*

$$\bar{b}_0 = \alpha_1 b_0 + \alpha_2 b_1 + \alpha b_0^*, \tag{11}$$

$$\bar{b}_i = \frac{n-i}{n}(\alpha_1 b_i + \alpha_2 b_{i+1} + \alpha b_i^*) + \frac{i}{n}(\alpha_3 b_{i-1} + \alpha_4 b_i + \alpha b_i^*), \quad 0 < i < n, \tag{12}$$

$$\bar{b}_n = \alpha_3 b_{n-1} + \alpha_4 b_n + \alpha b_n^*. \tag{13}$$

Remark 3 *On other words, equations (11) and (13) indicate that* b_0, b_1, b_0^* *and* $\bar{b}_0$ *(respectively* b_{n-1}, b_n, b_n^* *and* $\bar{b}_n$*) are coplanar and*

$$\frac{|\bar{b}_0 b_1 b_0|}{|b_0^* b_0 b_1|} = \frac{|\bar{b}_n b_n b_{n-1}|}{|b_n^* b_{n-1} b_n|}. \tag{14}$$

Using propositions 1 and 2, it is possible to develop an algorithm for the construction of a 'G^1' Bézier surface over a given triangulation, where surface tangent planes at the vertices are supposed to be known. Three different steps may describe this algorithm:

Construction of cubic surface patches over each triangle. The purpose is to define, over each mesh triangle, cubic surface patches, so that the resulting global surface will be 'G^1' at every vertex. A cubic triangular surface patch has 10 control points that can be obtained in the following way:

1. Around each vertex, s_i, the control points $q_i^1, \ldots q_i^{nv_{s_i}}$ (nv_{s_i} is the number of vertices around s_i) over the sides $a_i^1, \ldots, a_i^{na_{s_i}}$ (na_{s_i} is the number of mesh edges connected with s_i) (see fig. 2) satisfying
   - q_i^j, $j = 1, \ldots, nv_{s_i}$ lie on the tangent plane at s_i and
   - $\frac{|s_i q_i^j q_i^{j+1}|}{|s_i s_i^j s_i^{j+1}|}$ =constant (independent of $j = 1, \ldots, nv_{s_i} - 1$ and s_i) which is taken as $\frac{1}{9}$,

   are computed

2. From these control points q_i^j, the control points b_I with $|I| = 3$ for each triangle K can be obtained excepting $b_{(1,1,1)}$ that can be computed from

$$S_K^3(\mathbf{u_0}) = \frac{1}{4}[b_{(2,1,0)} + b_{(1,2,0)} + b_{(0,2,1)} + b_{(0,1,2)} + b_{(1,0,2)} + b_{(2,0,1)}] - \frac{1}{6}[b_{(3,0,0)} + b_{(0,3,0)} + b_{(0,0,3)}],$$

where $S_K^3(\mathbf{u})$ is the triangular Bézier surface patch of order 3 over the triangle K and $\mathbf{u_0} = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. Note that this formula gives quadratic precision if the control points around the vertices have been computed exactly.

Subdivision and degree elevation. From every triangular Bézier surface patch over K, for each $K \in \mathcal{T}_h$ is subdivided at its barycenter into 3 subpatches of order 3 using a *subdivision algorithm* (see [9], [4]). Thus 19 control points are computed. Every subpatch of degree 3 is expressed as a patch of order 4 by a *degree elevation algorithm* (see [9], [4]) so that, 31 control points are obtained.

'G^1' corrections. For every inter-element boundary Γ of the original triangles, the interior control points b_1^*, b_2^* and $\bar{b}_1$, $\bar{b}_2$ adjacent to Γ (see fig. 3) are modified so that condition (12) is satisfied. These modifications may disturb the interior 'G^1'-continuity of the original triangles. Then a suitable correction is made to obtain global 'G^1'-continuity. For more details see [6], [9], [12] and [4].

The Bézier surface of order 4 over the triangulation, obtained from the above control points, is 'G^1' .

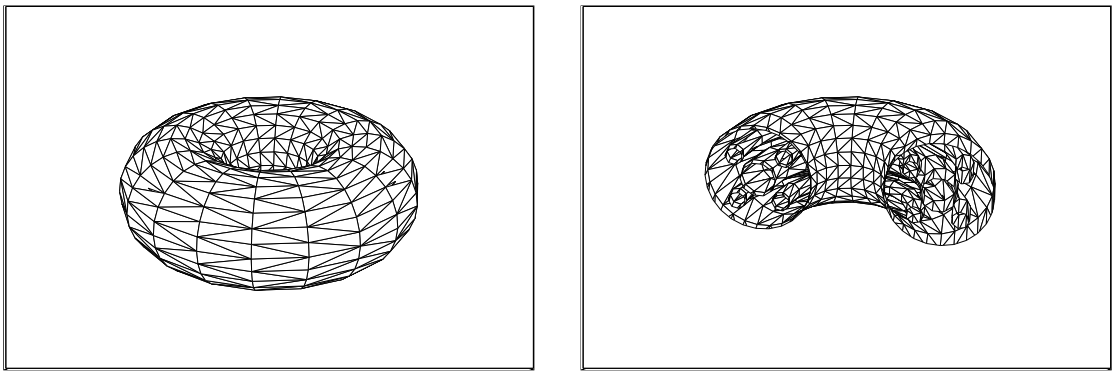

Figure 4: Initial mesh of a torus recovering 5 others.

3.1 Examples

In this section, some examples computed using a generalized Farin's algorithm are shown. The generalized algorithm can deal with surfaces with sharp edges (see [4]). The resulting meshes are finer than the original ones and they are obtained subdividing each triangle into 4, 9, 16,... and so on.

Six Torus: In this case, the initial mesh is not connected. Figure 4 shows the initial triangulation (courtesy of E. Saltel, INRIA) and a detail of it. Figure 5 shows final mesh obtained subdividing each element of the initial mesh into 4 elements.

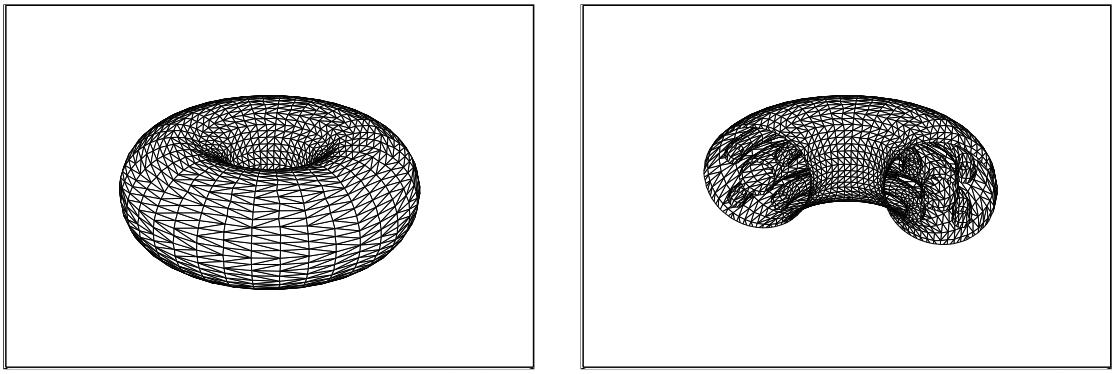

Figure 5: Final torus mesh.

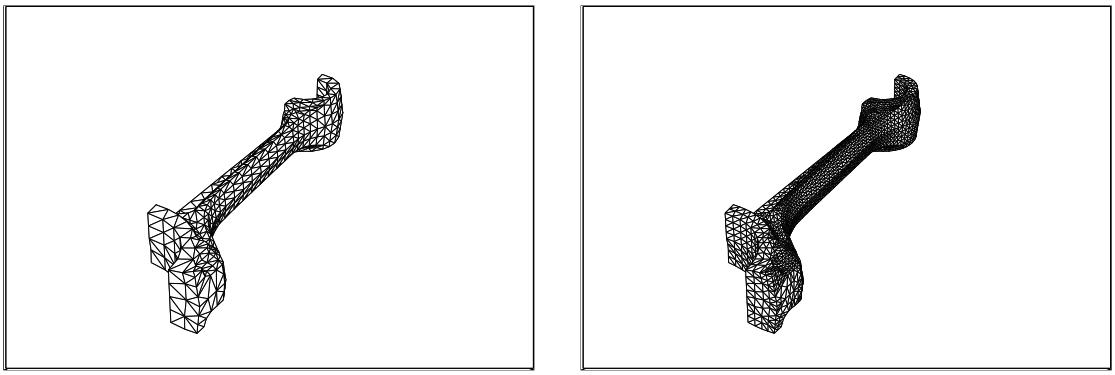

Figure 6: Initial mesh of a connecting rod and the refined mesh obtained.

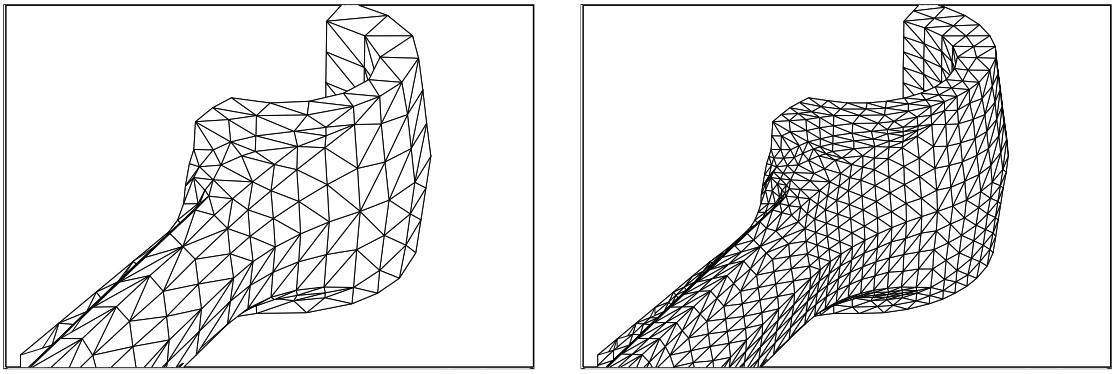

Figure 7: Zoom of the initial and final connecting rod meshes.

Connecting rod: Figure 6 shows the initial triangulation (courtesy of DataVision) of a connecting rod and the final obtained mesh. Figure 7 shows the intersection lines between two 'G^1' patches, and the final result. As can be observed, intersection lines are also 'visually continuous'. The final mesh has been obtained by splitting each triangle into 4 new ones.

4 Projection algorithms over meshes

The informations controlling the mesh generation process, as the metric tensor $\mathcal{M}$ and C.A.D definition are given over different grids, $\mathcal{T}_0$ and $\mathcal{T}_G$, respectively. The two following problems then appear:

In order to position a new point x over the surface, the element $K_G \in \mathcal{T}_G$ containing the point must be found and its barycentric coordinates with respect to K_G computed. Finally its position on the surface defined using Farin's algorithm (see section 3) must be determined. The same problem appears when the metric at a given point x must be computed. In all generality, if Ω_0 is the support of the discrete domain $\mathcal{T}_h$ and x is an arbitrary point, the problem to be solved during the mesh adaptation process can be formulated as :

Find:

1. *a point $x' \in \Omega_0$ so that $d(x', x)$ is minimal, that is, x' is a projection of x over $\overline{\Omega_0}$ (this projection may be not unique if Ω_0 is not convex),*
2. *an element $K \in \mathcal{T}_h$ containing x',*
3. *the barycentric coordinates of x' with respect to K.*

In the following sections, two different algorithms to solve this problem, are proposed together with an initialization algorithm. The first one is a global projection algorithm, used in $2D$ cases. The second is a local algorithm used to locate points on surfaces and the initialization algorithm is based on topological mesh connectivities.

4.1 Projection over $2D$ domains

As a first approach, let us suppose that Ω_0 is convex and let $\mathcal{T}_h$ be an arbitrary triangulation of it. Let x be a point and $\hat{K} \in \mathcal{T}_h$ an initial element. Let us construct a series of elements $\{K_i\}_{i=0}^n$ verifying:

- $K_0 = \hat{K}$
- K_{i+1} is a neighbor of K_i, i.e., they have a common edge and
- K_n is a triangle of $\mathcal{T}_h$ containing x or its projection x'. Once K_n has been found, the barycentric coordinates of x' with respect to K_n are computed.

The algorithm that constructs the previous series is based on 'oriented area' of $2D$ domains. The algorithm works as follows: assuming that K_i is known, K_{i+1} is found in the following way:

First the triangles, K_i^j, $j = 1, \ldots, 3$, obtained by joining x with the 3 'oriented' edges of K_i (see fig. 8) are constructed. Only three cases[1] are possible:

[1]The case where the three constructed triangles have 'negative' areas is geometrically impossible

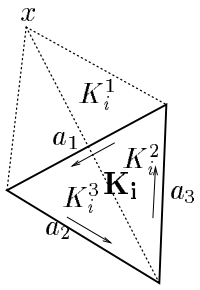

Figure 8: Triangle K_i its K_i^j, $j = 1, \ldots, 3$ triangles.

1. the three triangles have 'positive' areas. Then $x \in K_i$, and the algorithm ends;

2. only one of the triangles K_i^j, $j = 1, 2, 3$, noted as $K_i^{j_0}$, has a 'negative' area. In that case, let a_j be the edge corresponding to $K_i^{j_0}$. If a_j is an interior edge, then K_{i+1} is the neighbor triangle to K_i having a_j as common edge. If a_j is a boundary edge, then we project x over K_i and the series is constructed and the algorithm ends;

3. two of the triangles have 'negative' area. Then the triangle with smaller area and its edge a_j are retained. Therefore, we are in the same conditions than in the previous case, thus the same process can be applied.

As it can be observed, this algorithm has no arbitrary choice. It must choose among one of the four different actions: stop or take as K_{i+1} one of the three neighbor triangles of K_i. The series $\{K_i\}_{i=0}^n$ constructed with the previous algorithm verifies that every two successive elements have a common edge an the distances, $d_i = d(x, K_i)$, constitute a decreasing series.

As it has been noted, this algorithm only works if Ω_0 is convex. For non convex sets, this algorithm may project the point x onto x', but there may exist points in Ω_0 closer to x than x'. Supposing that the algorithm stops at point x' and $d(x, x') > 0$ implies that $x' \in \Gamma_0 = \partial\Omega_0$. If the domain is convex then $x \in \Omega_0^C$ and x' is the x projection over $\overline{\Omega_0}$. Nevertheless, if Ω_0 is non-convex, then this situation does not give any information about neither x nor its projection.

Let us denote each closed curve defining Γ_0 by Γ_k, $k = 1, \ldots, m$ and let x' be the point given by the previous algorithm, $x' \in \Gamma_0$ with $d_0 = d(x, x') > 0$. If

$$d_0 \leq d(x, y), \quad \forall y \in \Gamma_k, \ k = 1, \ldots, m,$$

then x' is one of the possible projections of x over $\overline{\Omega_0}$ (note that Ω_0 is not necessarily convex), i.e.

$$d_0 \leq d(x, y), \quad \forall y \in \overline{\Omega_0}.$$

Thus, the previous projection algorithm is modified in order to obtain an appropriate projection of x, even in non convex domains as follows: *if the algorithm stops at point* x' *with* $d_0 = d(x, x') > 0$, *then* $x' \in \Gamma_0$. *Let* $\{a_i, \ i = 1, \ldots, na_f\}$ *be the set of boundary edges. Computing the distances*

$$\hat{d}_j = d(x, a_j), \quad j = 1, \ldots, na_f,$$

if $d_0 \leq \hat{d}_j$ then x' is one of the projections of x over $\overline{\Omega_0}$. In other case, there exists j_0, such that $\hat{d}_{j_0} < d_0$. Let K' be the unique element of $\mathcal{T}_h$ having a_{j_0} as edge, then K' will be the following series element and the construction of the series continues till $d(x, K_n) = 0$ or $d(x, x')$, $x' \in \Gamma_0$, minimizes the functional,

$$D: \overline{\Omega_0} \mapsto \mathbb{R}^+, \quad D(y) = d(x, y), \ \forall y \in \overline{\Omega_0}.$$

This modification is numerically expensive, but it allows to state whether a point is exterior to an arbitrary domain or not. Observe that the distances $\hat{d}_j = d(x, a_j), j = 1, \ldots, na_f$ are only computed once, when arriving at the very first time to a boundary edge.

4.2 Projection over surfaces

In this section, a local projection algorithm over arbitrary surfaces is described. This algorithm is quite efficient if the initial element, K_0, is close to the final projection. Therefore, it will be necessary to develop an initialization algorithm that provides suitable initial elements to this local projection algorithm.

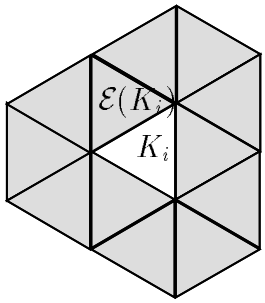

Figure 9: K_i neighborhood.

Definition 5 *Let K be an element of $\mathcal{T}_h$. We define the neighborhood of K as (see fig. 9):*

$$\mathcal{E}(K) = \{K' \in \mathcal{T}_h, \ K' \neq K \text{ and } K' \cap K \neq \emptyset\}. \tag{15}$$

As in the previous section, a series of triangles $\{K_i\}_{i=0}^n$ are constructed verifying the conditions:

- $K_{i+1} \cap K_i \neq \emptyset$,
- $d_{i+1} < d_i$, where $d_i = d(x, K_i)$, and
- K_n is the triangle of $\mathcal{T}_h$ containing x or its projection x'.

In order to avoid calculating the distance $d(x, K')$ twice, a marker over the elements is used. Supposing K_{i-1} and d_{i-1} are known, then K_i and d_i can be computed as follows:

1. Let $\mathcal{E}(K_{i-1})$ be the neighborhood of K_{i-1}.

2. For each non marked element $K' \in \mathcal{E}(K_{i-1})$, $d(x, K')$, is computed and then K' is marked. Let d' be the minimum of these distances. Two situations are possible:

   (a) If $d' \leq d_{i-1}$ take $K_i = K'_0$, where $K'_0 \in \mathcal{E}(K_{i-1})$ satisfies $d' = d(x, K'_0)$, and $d_i = d'$.

   (b) If $d' > d_{i-1}$ then d_{i-1} is a local minimum of the functional $d(x, y)$, $\forall y \in \Omega_0$, x' is the projection of x over the triangle K_{i-1} and the construction of the series ends.

This algorithm is slightly more expensive than the previous one but, if our initial element is close to the minimum, it converges in very few iterations. Let us remark that to guarantee that boundary points and points over the curves of intersection of surfaces are well preserved, we check that their projections are also boundary points or points over the curves of intersection of surfaces, respectively.

4.3 Initialization algorithm

For the set up of the two projection algorithms presented in this paper, an initial element K must be given. In the first algorithm, this initial element is only important to ensure a fast convergence, because it was a global projection algorithm. In the second algorithm, the selection fo the initial element is crucial. If this element is far away from the projection x' of x, then this local algorithm does not lead an adequate projection. Therefore, a robust initialization algorithm is needed.

In order to initialize the projection algorithm, a data structure is associated to the mesh $\mathcal{T}_1$: each vertex, s_i^1, $i = 1, \ldots, n_v^1$, has two triangle-pointers, one pointing to a triangle of mesh $\mathcal{T}_0$ and the other pointing to a triangle K_0^G in $\mathcal{T}_G$. Let $x \in K^1 \in \mathcal{T}_1$, then a suitable initial element for x will be one of the elements pointed by its vertices. Therefore, $\mathcal{T}_1$ must be projected on the other two meshes in order to initialize this data structure (note that in this stage, $\mathcal{T}_1 = \mathcal{T}_0$, thus, only the projection $\mathcal{T}_1 \rightarrow \mathcal{T}_G$ is needed).

The initialization algorithm presented in this paper for whole meshes is based on their topological connectivities. The idea is simple. Let's take s_1^1 the first vertex in $\mathcal{T}_1$ and let x it coordinates. An arbitrary triangle is chosen as initial element and one of the previous projection algorithm applied. When using the local projection algorithm, this is controlled by testing the distance $d(x, x')$, where x' is the approximated x-projection. If $d(x, x')$ is smaller than a mesh step dependent parameter ϵ, then the algorithm stops. When $d(x, x') > \epsilon$, then the algorithm continues by taking another non marked element. Therefore, if possible, a new projection x'' verifying $d(x, x') < d(x, x'')$ will be obtained. This process is repeated till $d(x, x^*) < \epsilon$ or all elements are marked. Thus, the local projection becomes a global one. Now, we associate to each vertex of $\mathcal{T}_1$ connected with s_1^1 the corresponding final triangle containing the projection of s_1^1. Those vertices are close to s_1^1, so this final element associated to s_1^1 will be a good initial triangle for the vertices connected to s_1^1. The same process is repeated for each vertex. A suitable data structure for the implementation of this algorithm is the heap.

Numerical experiences show that this algorithm is quite efficient and that the global projection will be only needed when projecting the first vertex.

5 Remeshing tools

In this section, five different tools used for adapting meshes are described. They can be grouped into two types: those resulting in a change of the number of mesh items (i.e. the number of triangles, edges and vertices are modified) and those used to improve the quality of the mesh.

The two first operators allow to suppress topological elements (vertices, edges and elements), the third one allows to add them and the other two are optimization tools. The algorithms developed are geometrically simple and numerical experiences show that they are quite efficient and robust, i.e., the final mesh is well adapted with respect to the tensor metric and is topologically correct. In order to improve the quality of the final mesh some mesh optimization algorithms must be used, in particular, after adding or suppressing a mesh item edge swapping is always used.

5.1 Edge suppression

Let s_i be a vertex of $\mathcal{T}_h$. The **neighborhood of the vertex** s_i over the mesh $\mathcal{T}_h$ is defined as

$$\mathcal{E}(s_i) = \{K' \in \mathcal{T}_h,\ s_i \in K'\}. \tag{16}$$

Let a_i be an ege of $\mathcal{T}_h$ and let s_0^i and s_1^i be its vertices and let $\mathcal{E}_i$ (see figure 10) be

$$\mathcal{E}_i = \mathcal{E}(s_0^i) \cup \mathcal{E}(s_1^i).$$

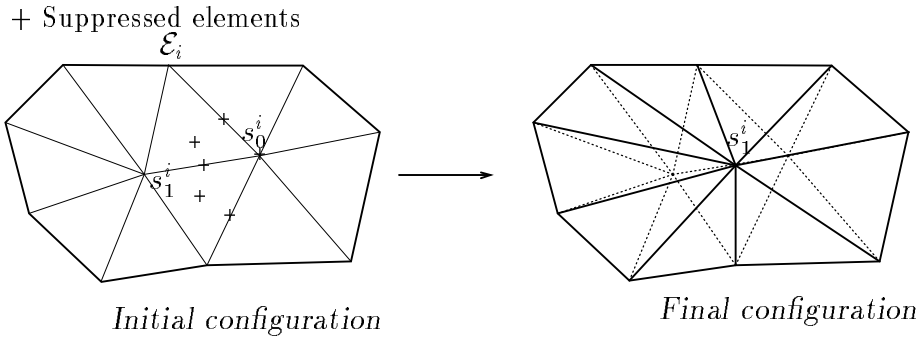

Figure 10: Edge suppression algorithm.

The edge suppression operator, as the others operators presented in this section, is local, that is, the suppression of an edge a_i only affects a small area ($\mathcal{E}_i$) around a_i. The algorithm modifying $\mathcal{E}_i$ in order to suppress the edge a_i consists in:

1st step: In order to verify if a_i can be suppressed or not, the following restrictions, topological or geometrical, are checked before edge suppression:

- If its two vertices, s_0^i and s_1^i, are fixed points then a_i can not be suppressed.
- If a_i is an interior and/or non inter-subdomain edge and s_0^i is a boundary and/or inter-subdomain vertex and s_1^i is a boundary and/or inter-subdomain vertex, then a_i can not be suppressed.

- If a_i is a boundary edge (respectively an inter-subdomain edge) and its suppression entails a great variation from the initial configuration of the boundary line (resp. inter-subdomain curve), i.e., their tangent vectors are quite different, then a_i can not be suppressed.
- When obtaining the final configuration, if the oriented normal vectors associated to the resulting elements have significantly changed with respect to the initial normal vectors, then a_i can not be suppressed. This verification is numerically expensive but allows to avoid the two following situations:
  - large tangent plane variations and
  - triangle overlapping.

2nd step: Once the filtering step is undertaken, if a_i must be suppressed, this second step is applied. The algorithm proposed here gives the result shown in figure 10.

If a_i is a boundary edge, its suppression implies the suppression of one vertex, another edge and the element containing it. However, if a_i is an interior edge, then one vertex, two other edges and the two elements containing it are suppressed (see fig. 10). The elimination of and edge always implies the suppression of one vertex (one of its vertices). Its selection is crucial to preserve the geometry shape as can observed in figure 11 and it is made as follows:

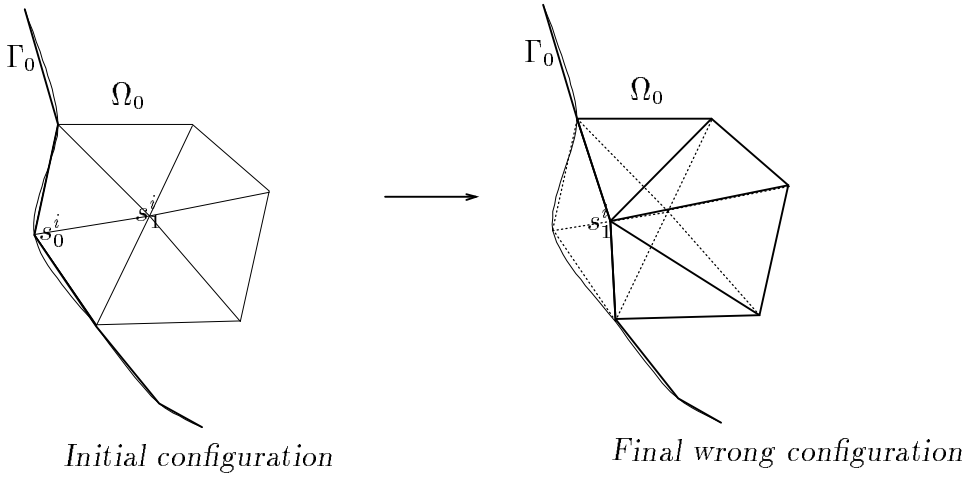

Figure 11: Example of wrong edge suppression due to a wrong choice of the suppressed vertex.

- If the two points are similar, i.e., both are boundary, inter-subdomain or interior ones, and non fixed points, then we can arbitrary take one of them.
- If there exist one fixed point[2], the other vertex is chosen.
- If there is at least one interior non fixed a_i vertex, that can be assumed to be s_0^i, with s_0^i not an inter-subdomain vertex, s_0^1 is chosen.

[2]Two fixed points configuration are impossible because we suppose that a_i can be suppressed.

After this choice and assuming that s_0^i is the chosen vertex and s_1^i is the remaining, two possibilities may happen:

- The two vertex are non fixed points. Then

$$s_1^i = \mathcal{P}\left(\frac{s_0^i + s_1^i}{2}\right),$$

  where $\mathcal{P}: \mathcal{T}_h \longrightarrow S_1$ is the C.A.D. projection operator, i.e., it projects a point p into the defined Bézier surface using Farin's algorithm together with a projection algorithm (see section 3).

- If s_1^i is a fixed vertex, it remains unchanged.

The final configuration is obtained identifying s_0^i with s_1^i and the corresponding edges and elements. Finally, the marked triangles and edges in figure 10 as well as the vertex s_0^i are suppressed.

Two remarks:

- The final mesh configuration is correct, that is, it preserves the mesh conformity that has been checked when a_i goes through the initial filter. This process frees the memory filled by the suppressed elements.

- The edge suppression process carries out a decrease in the quality of the elements with respect to the metric criterion, therefore a local optimization step is necessary (edge swapping, see subsection 5.4) to improve the quality of the elements.

5.2 Vertex suppression

Let s_i be a non fixed vertex of $\mathcal{T}_h$ to be suppressed. Basically, the algorithm proposed for vertex suppression is analogous to the edge suppression algorithm described above. If a vertex has to be eliminated, it will be associated to an edge so that its elimination implies the suppression of s_i. Therefore, it is only necessary to determine an edge associated to s_i and then apply the previous algorithm to this edge. The edge association procedure is made as follows:

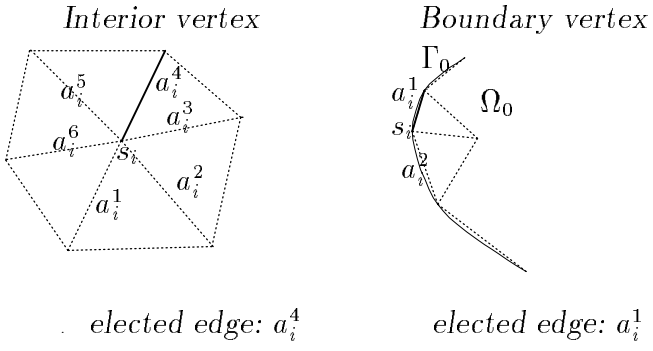

Figure 12: Edge association in vertex suppression algorithm

1. If s_i is a boundary or inter-subdomain vertex, there are only two possible edge elections in order to preserve the boundary or inter-subdomain curve (see figure 12). Let a_i^1 and a_i^2 be the two possible edges, being s_i their common vertex. Let l_0 (respectively l_1) be the length of a_i^1 (respectively a_i^2) computed using the defined metric. Then, the smallest edge is taken.

2. In the other case, no restrictions are imposed, therefore the lengths of all the mesh edges connected to s_i are computed and the smallest is retained.

At this point, the algorithm describe in 5.1 is used, to suppress the selected edge, imposing the suppressed vertex to be s_i but it may happen that it can not be removed because one of the restrictions in edge suppression is violated. As in the preceding algorithm, an optimization step is needed to improve the local quality of the modified triangles.

5.3 Vertex addition

The vertex addition process is used when the length of a mesh edge is greater than a certain parameter, l_{max}, controlling the maximal length allowed for mesh edges. This process also implies the creation of one or two triangles and two or three new edges depending on the nature of the edge (boundary edge or not) where the new vertex has to be positioned (see figure 13).

Let p_0 be the new vertex to be located on the edge a_0. The algorithm developed is as follows:

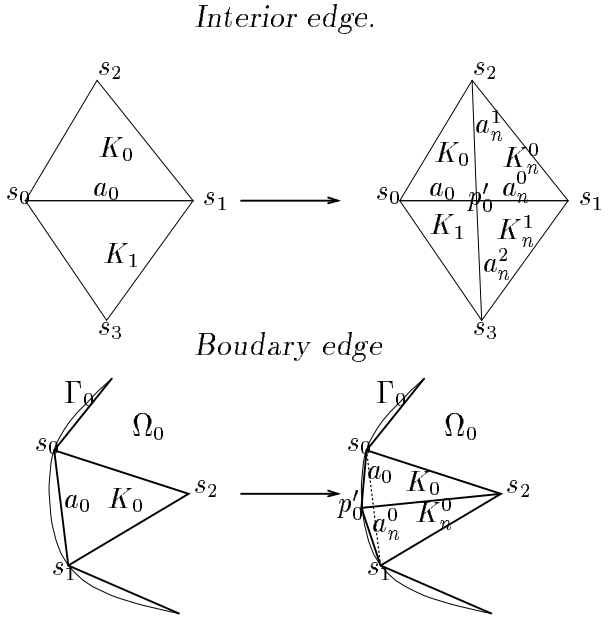

Figure 13: Vertex addition algorithm.

1. If a_0 is a boundary edge then, adding a new vertex over it, implies the generation of two new edges and one triangle (see figure 13). In order to respect the geometrical

domain definition obtained by Farin's algorithm, the projection p_0' of p_0 over the interpolated surface S_1 is considered. It may happen, as $p_0' \notin a_0$ and depending on the efficiency of the projection algorithm, that the final configuration is not admissible (triangle overlapping). In that case $p_0' = p_0$ is imposed.

2. In other case, the vertex addition process implies the creation of three new edges and two new triangles (see figure 13). As in the previous case, the projection[3] p_0' of p_0 is considered and it is checked if the final configuration is admissible or not. If not, $p_0' = p_0$ is imposed. At that stage, the length of the edges $p_0's_3$ and $p_0's_2$ are computed (see figure 13). If the length of these two new edges are both smaller than a parameter l_{min} controlling the smallest edge length allowed and a_0 can be swapped, then p_0' is not added and a_0 is swapped (see figure 14). This last modification allows to adapt the mesh without adding many vertices.

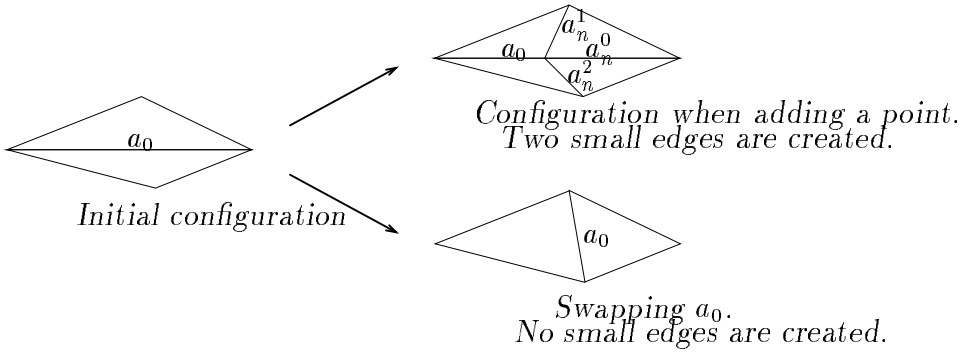

Figure 14: Edge swapping instead of vertex addition

As in previous algorithm, a local optimization step is necessary to maximize the quality of the new elements.

5.4 Edge swapping

Diagonal swapping is the best tool that can be used in mesh optimization. This method allows to improve considerably the quality of the final mesh by means of a very simple process, keeping the same number of vertices, edges and elements. The diagonal swapping algorithm proposed here is local and its principle is quite simple:

Let K_0 and K_1 be two adjacent triangles having a_0 as common edge. First of all, some restrictions imposed to the diagonal swapping algorithm in order to decide if a_0 can be changed or not must be considered:

1. First, remark that a_0 can not be a boundary edge, because it is common to two triangles. If a_0 is an inter-boundary edge, then it can not be changed in order to respect the geometrical restrictions concerning the inter-subdomain curve definition.

2. Let us consider the two unit normal oriented vectors associated to K_0 and K_1 before the diagonal swapping. If the two modified triangles resulting from the diagonal

[3]This projection is only computed to position points over surfaces. In $2D$ domains $p_0' = p_0$.

swapping process are quite different from the initial ones, then a_0 can not be changed. This condition avoids element overlapping , large changes of the initial and final tangent planes[4] and guarantees mesh conformity.

3. Let c_0 and c_1 be the initial quality criteria of the two triangles K_0 and K_1 computed using the associated tensor metric and let $c^i = \min(c_0, c_1)$. The criterion for measure the quality of the triangles used here is the following:

$$c_K = \frac{27(p-l_1)(p-l_2)(p-l_3)}{p^3}, \tag{17}$$

where l_i, $i = 1, \ldots, 3$ are the lengths of each edge of K computed using the metric given at their vertices and p is the half-perimeter of K. This criterion has been chosen because is easy to compute when changing the metric tensor. The minimal criterion, c^f, associated to the final configuration supposing that a_0 can be swapped is also computed. If $c^f \leq c^i$ then a_0 can not be changed.

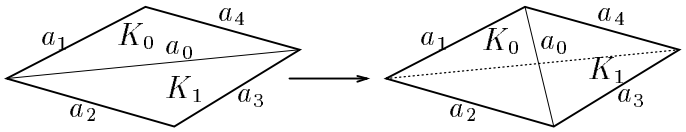

Figure 15: Edge swapping.

Now, if a_0 has been changed, this process is repeated for the four edges a_1, a_2, a_3 and a_4 with their corresponding elements (see fig. 15).The data structure appropriate for the implementation of this algorithm is the heap, where the elements and edges needed in the following algorithm step are stored. This algorithm ends when the heap is empty or a maximal number of iterations is made. In general, numerical experiences show that this maximal number of iterations is not reached and only the control 'heap empty?' is needed.

5.5 Moving vertices

Let s_i be a mesh vertex and let us consider, $\{s_i^j\}_{j=1,\ldots,nv_{s_i}}$, the set of vertices connected with s_i (see figure 16). The algorithm tries to place the vertex s_i (in a first approach, s_i can be supposed to be an interior and non inter-subdomain vertex) so that the quality of the triangles connected with it increase. As the control space is given by the metric tensor $\mathcal{M}$, it must play an important role when deciding the optimal position, p_0, for s_i. In order to find it, the following minimization problem is solved:

Find p_0 so that the cost-functional $f: \Omega \longrightarrow \mathbb{R}^+$ defined by

$$f(x) = \sum_{j=1,\ldots,nv_{s_i}} (x - s_i^j)^T \mathcal{M}(x)(x - s_i^j)$$

[4]Only considered with surface meshes

is minimized. Now, the following simplification is made: the metrics at p_0 and s_i^j are supposed to be equal. They are denoted by $\mathcal{M}_j$ and defined as

$$\mathcal{M}_j = \left(\frac{\mathcal{M}_{s_i}^{-1} + \mathcal{M}_{s_i^j}^{-1}}{2} \right)^{-1}, \quad j = 1, \ldots, nv_{s_i},$$

where $\mathcal{M}_{s_i}$ is the metric associated to s_i and $\mathcal{M}_{s_i^j}$ those associated to s_i^j $j = 1, \ldots, nv_{s_i}$.

The cost-functional $f: \Omega \longrightarrow \mathbb{R}^+$ becomes

$$f(x) = \sum_{j=1,\ldots,nv_{s_i}} (x - s_i^j)^T \mathcal{M}_j (-s_i^j). \tag{18}$$

At this stage, p_0 is computed as the solution of

$$\nabla f(x) = 0$$

that can be computed exactly obtaining the expression:

$$p_0 = \left(\sum_{j=1,\ldots,nv_{s_i}} \mathcal{M}_j \right)^{-1} \sum_{j=1,\ldots,nv_{s_i}} \mathcal{M}_j s_i^j. \tag{19}$$

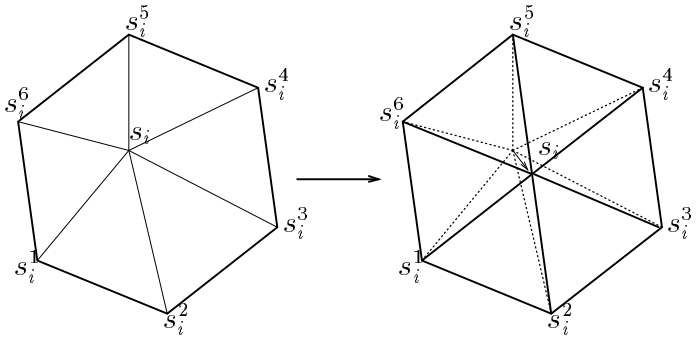

Figure 16: Moving vertex process.

Let us consider the displacement vector $dx = \omega(p_0 - s_i)$, where $\omega \in [0, 1]$ and let $dt = \|dx\|$. If $dt > d_{\max}$, where $d_{\max}$ is the maximal displacement allowed, then dx is truncated so that $\|dx\| \leq d_{\max}$. Now, s_i is placed at point $s_i + dx$. If s_i is a surface vertex, then it must be projected over the Bézier surface computed by Farin's + projection algorithm. Finally the latest element configuration is checked in order to guarantee that:

1. it is correct, i.e., no element overlapping;
2. the criteria of quality of the triangles are greater than in the initial configuration.

If one of the previous conditions are not satisfied, then the norm of the displacement vector $\|dx\|$ is decreased and s_i is placed using the new value for dx. This process is repeated until the two previous conditions are satisfied or a maximal number of iterations ($\approx 8 - 10$) is reached.

If s_i is a boundary or inter-subdomain vertex, the previous algorithm is modified, so that, only the two contiguous connected vertices are taken into account when computing the displacement vector. Now, s_i (after it is placed at $s_i + dx$) is projected imposing its image to be a boundary or inter-subdomain point.

Remark 4 *Numerical experiments show that this algorithm is efficient if repeated several times ($\approx 10 - 12$) for all mesh vertices. In this case, during the first iterations, condition 2 can be changed by a less restrictive constraint allowing a smoother decrease for the criterion of quality of triangles from the previous configuration.*

6 Remeshing algorithm

In this section, a remeshing algorithm is presented. This algorithm uses the local tools described in the previous section. As noted, the above algorithm needs of three different grids: one defining the geometry of the domain, called *'C.A.D. mesh'*, $\mathcal{T}_G$, the second denoted by $\mathcal{T}_0$ contains the control space information, i.e., the metric tensor definition and, finally, the last is the adapted mesh $\mathcal{T}_1$. The algorithm solves the following problem:

Find 'an optimal mesh', $\mathcal{T}_1$, adapted from $\mathcal{T}_0$ with respect to the criterion imposed by the metric tensor $\mathcal{M}$ and verifying that it is close to $\mathcal{T}_G$.

In each grid adaptation loop, $\mathcal{T}_G$ is fixed (geometry definition), $\mathcal{T}_0$ is the mesh to be adapted and $\mathcal{T}_1$ is the final adapted mesh. In order to provide an initial solution over $\mathcal{T}_1$ that can be used to initialize the finite element computations, the approximate solution computed with the mesh $\mathcal{T}_0$ is interpolated over $\mathcal{T}_1$. Therefore, the meshes $\mathcal{T}_G$ and $\mathcal{T}_0$, together with the metric tensor $\mathcal{M}$ are supposed given and, possibly, a finite element solution over $\mathcal{T}_0$ is provided. A preliminary topological verification together with a triangle orientation check are carried out in order to guarantee that the given meshes are correct and oriented.

Before starting the grid adaptation process, it will be necessary to create some auxiliary data as: mesh connectivity arrays, boundary and inter-subdomain (geometrical and physical)[5] edges with their associated tangent vectors, fixed point localization, etc. Generalized Farin's algorithm (see section 3) is used to define a 'G^1' Bézier surface over $\mathcal{T}_G$. and $\mathcal{T}_1$ is set initially to be equal to $\mathcal{T}_0$. Pointer initialization for each vertex of $\mathcal{T}_1$ is performed. Each vertex of $\mathcal{T}_1$ has three associated pointers to triangles each of one containing the considered vertex in each of the three working grids. The two pointers to triangles corresponding to the meshes $\mathcal{T}_0$ and $\mathcal{T}_1$ are quite easy to construct since they are identical but $\mathcal{T}_G$ and $\mathcal{T}_1$ may not be equal. Therefore, the pointer corresponding to the grid $\mathcal{T}_G$ is created using an initialization algorithm (see paragraph 4.3). Depending on user's choice, an initial optimization step (edge swapping) is made in order to improve the quality of the initial mesh. Numerical experiences show that the final result is better if this initial optimization is carried out.

The grid adaptation process starts ordering the topological mesh elements related with $\mathcal{T}_1$ in a particular data structure: double dynamic list (DDL). This structure allows to suppress and add elements in a very simple way. DDL for vertices, edges and triangles are considered.

[5]The first ones correspond to intersections of regular surfaces and the second ones correspond to intersections of user-defined subdomains.

Assuming that the first $i-1$ edges have been treated, the algorithm for treating the edge a_i is the following:

1. Let d_i be the length of a_i computed with the tensor metric $\mathcal{M}$. Three possible cases may occurred:
   - If $d_i > l_{max}$ ($l_{max} \approx 1.4$) then a_i must be cut in two edges using the algorithm for vertex addition(see paragraph 5.3). The new mesh elements (vertices, edges and triangles) created are added at the end of their corresponding lists. The new edge length, d_i, is computed, the process being repeated till $d_i \leq lmax$.
   - If $d_i < l_{min}$ ($l_{min} \approx 0.6$) then a_i is suppressed using the algorithm described in paragraph 5.1. If a_i has been eliminated, then new edges have to be checked. If they are larger than l_{max}. the previous process is applied to the edges, if they are smaller than l_{min} this process is repeated.
   - if $l_{min} \leq d_i \leq l_{max}$, a_i is kept.
2. If the mesh has been modified, a local optimization step must be carried out. In this case, the edge swapping algorithm is applied to the new or modified edges. This optimization step increases the mesh quality.

This process is repeated for all the edges in the list.

Remark 5 *A memory catch data structure is implemented to avoid repeating processes like the iterative creation and destruction of the same edge. Memory catch avoids these cyclical processes, but it can be numerically expensive, depending on its dimension. In this study, memory catch only retains the last* 10 *mesh modifications. Numerical experiments show that this capacity is sufficient since these periodical loops appear at the end of the edge double list, and their periods are smaller than* 10.

Remark 6 *A metric tensor drastically changing the initial mesh (i.e. creating* 30 *elements where the initial mesh had only one), provides a low quality final mesh. In order to improve the quality of the final mesh, a relaxation of the metric tensor is proposed. In this case, the final mesh is constructed in several iterations, each of them computed with a relaxation of the initial metric tensor and imposing it only at the final step. This method generates very well adapted meshes with respect to the given metric tensor, but is more expensive than imposing the original metric tensor at the first iteration.*

Finally, a global optimization process is considered. It consists on:

1. Non fixed, non inter-subdomain interior vertices belonging to less that 4 elements are suppressed by using the remeshing procedure proposed in paragraph 5.2.
2. All non fixed vertices are barycentred using the algorithm described in paragraph 5.5. This process is carried out several times ($\approx 8-10$).

Thus, the final mesh $\mathcal{T}_1$ adapted from $\mathcal{T}_0$ with respect to the metric tensor $\mathcal{M}$ respects the geometrical and topological constraints. If an initial solution over the initial mesh has been done, then this solution is interpolated over the final mesh using the projection algorithm. As it can be observed, the remeshing algorithm always works locally. This strategy allows to remesh surfaces without knowing its global parametrization, only a local parametrization is needed.

7 Examples

In this section, several meshes adapted by using the algorithm described in this paper are presented. In all cases, an initial mesh is given together with an associated metric. In this paper metric generation algorithms are not considered and they will be discussed in future works.

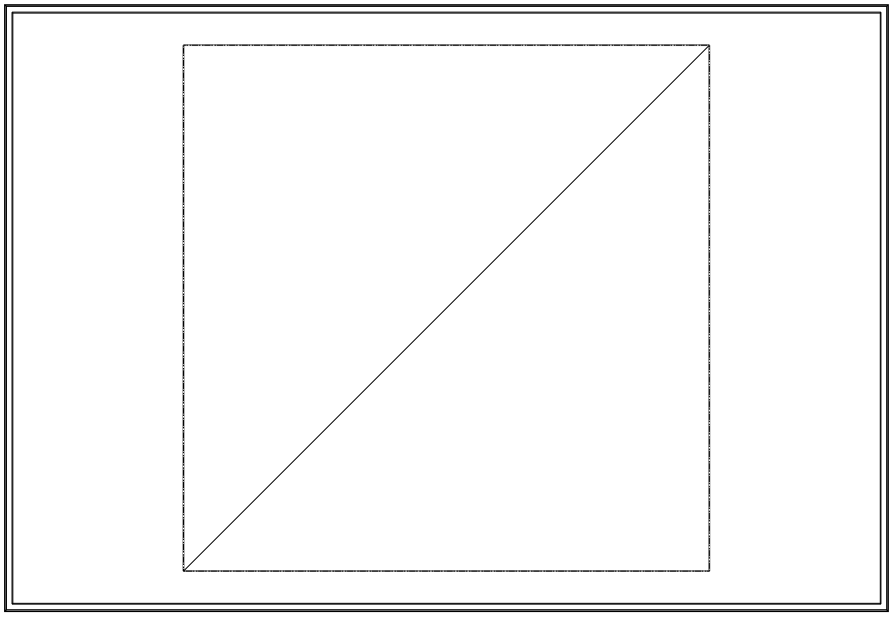

Figure 17: Initial mesh for the unit square.

7.1 Unit square

Case 1:

Let $\mathcal{T}_h = \Omega$ be the triangular mesh depicted in figure 17 and suppose that the function $f: \Omega \longrightarrow \mathbb{R}$ defined by $f(x,y) = 100x^2$ is given. The problem is to find an 'optimal' mesh, $\mathcal{T}_{op}$, minimizing the interpolation error $||f - \Pi_{op} f||_{L^2(\Omega)}$, where Π_{op} is the linear interpolation operator.

An anisotropic metric is computed so that it associated adapted mesh minimizes its error. The first and second adapted meshes obtained are shown in figure 18. Table 1 gathers their characteristics.

Meshes	*Vertices*	*Triangles*	*L^2-interpolation error*
Initial mesh	4	2	$18.2574 \cdot 10^{0}$
1^{st} adapted mesh	27	32	$2.8527 \cdot 10^{-1}$
2^{nd} adapted mesh	195	256	$4.4573 \cdot 10^{-3}$

Table 1. Case 1: Anisotropic mesh results.

Anisotropic meshes minimize the interpolation error without adding many new vertices, while isotropic meshes need much more triangles (32 times the number of anisotropic triangles!!!) to obtain an equivalent grid error as shown in figure 19 and table 2.

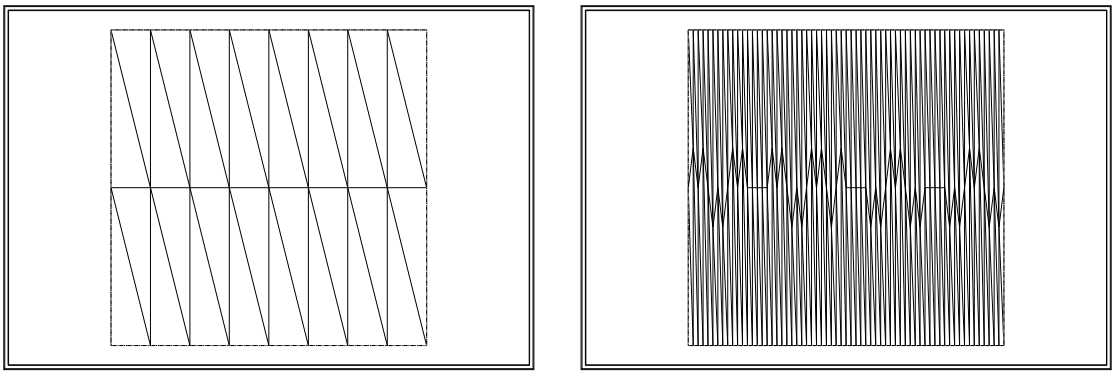

Figure 18: First and second anisotropic adapted meshes.

Meshes	*Vertices*	*Triangles*	L^2-*interpolation error*
Initial mesh	4	2	$18.2574 \cdot 10^{0}$
1^{st} adapted mesh	25	32	$1.1410 \cdot 10^{0}$
2^{nd} adapted mesh	81	128	$2.8527 \cdot 10^{-1}$
3^{rd} adapted mesh	289	512	$7.1318 \cdot 10^{-2}$
4^{th} adapted mesh	1089	2048	$1.7829 \cdot 10^{-2}$
5^{th} adapted mesh	4225	8192	$4.4691 \cdot 10^{-3}$

Table 2. Case 1: Isotropic mesh results.

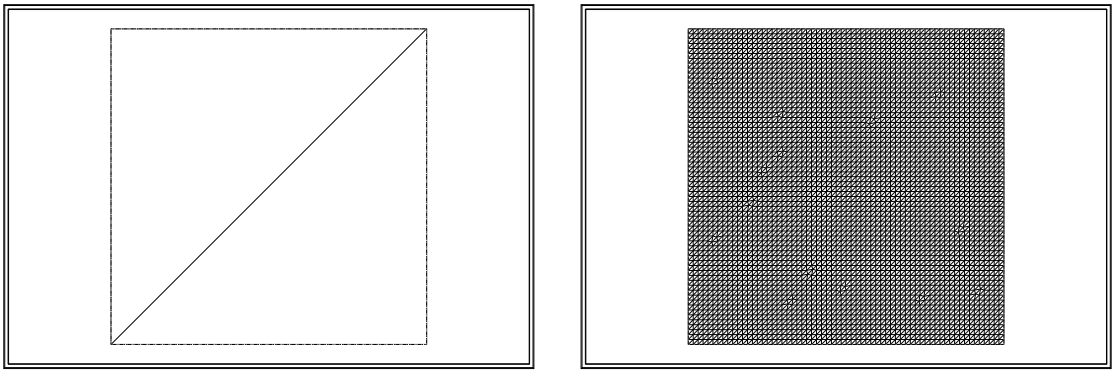

Figure 19: Initial mesh and final isotropic adapted mesh.

Case 2:

The discontinuous function $f: \Omega \longrightarrow \mathbb{R}$ defined by

$$f(x,y) = \begin{cases} 100x^2 & \text{if } x < 0.3 \\ 5 & \text{otherwise} \end{cases}$$

is considered

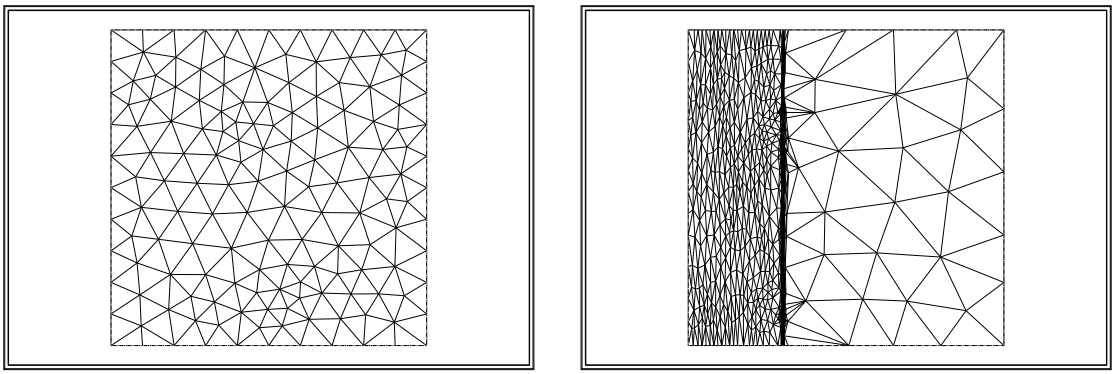

Figure 20: Initial mesh and final adapted mesh.

Meshes	*Vertices*	*Triangles*	L^2*-interpolation error*
Initial mesh	151	250	$4.5218 \cdot 10^{-1}$
1^{st} adapted mesh	543	1004	$2.0084 \cdot 10^{-1}$
2^{nd} adapted mesh	799	1527	$9.4020 \cdot 10^{-2}$
3^{rd} adapted mesh	987	1894	$5.9563 \cdot 10^{-2}$

Table 3. *Case 2: Anisotropic mesh results.*

Solving the same problem of minimizing L^2-interpolation error over meshes, the results shown in figure 20 are obtained. Even if the initial mesh has no information about function line discontinuity, the final grid succeeds in positioning it at its correct location as shown in figure 20. Vertex suppression can also be appreciated where the function f is constant. It can be said that the final mesh is 'optimal' in the sense that it minimizes the L^2-interpolation error with a minimal number of triangles.

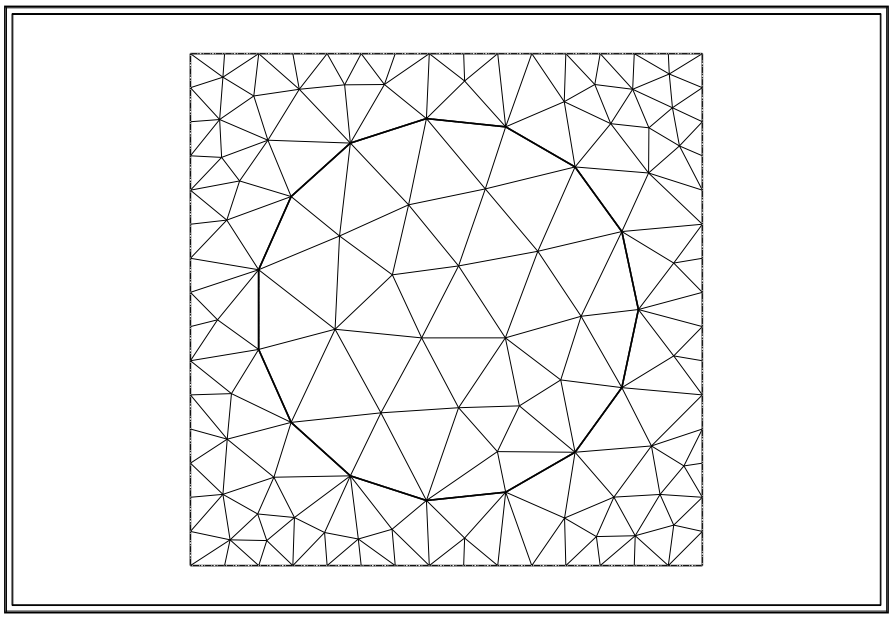

Figure 21: Initial mesh of a square plate with two subdomains.

Case 3:

Finally, we consider the same domain Ω but divided into two subdomains (see fig. 21): the circle of center $(0,0)$ and radius 0.75, noted as Ω_1, and $\Omega \setminus \Omega_1$. Let us consider the same minimization problem with a new continuous function f defined by

$$f(x,y) = \begin{cases} \sqrt{0.75^2 - x^2 - y^2} & \text{if } (x,y) \in \Omega_1 \\ 0 & \text{otherwise} \end{cases}$$

The obtained results are shown in figure 22 and summarized in table 4:

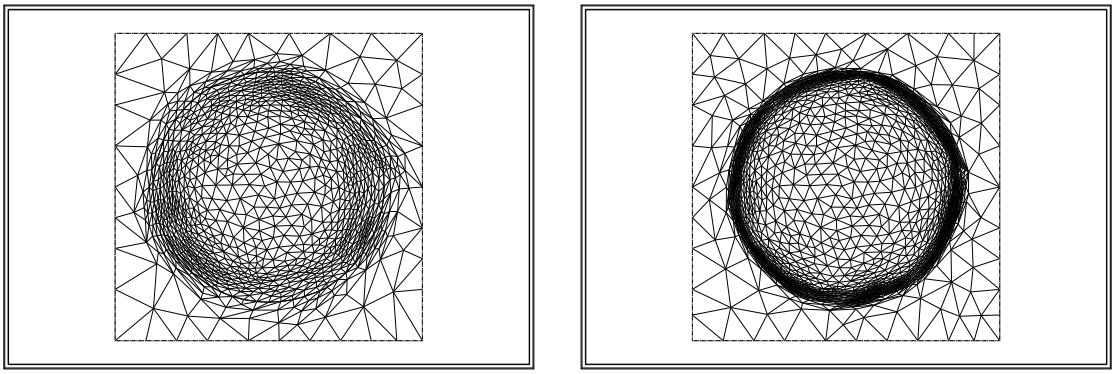

Figure 22: Meshes obtained in the second and third anisotropic grid adaptation steps.

Meshes	*Vertices*	*Triangles*	L^2-*interpolation error*
Initial mesh	133	204	$1.5107 \cdot 10^{-1}$
1^{st} adapted mesh	286	537	$5.0120 \cdot 10^{-2}$
2^{nd} adapted mesh	958	1880	$1.4492 \cdot 10^{-2}$
3^{rd} adapted mesh	1633	3227	$4.7901 \cdot 10^{-3}$

Table 4. Case 3: Anisotropic mesh results.

7.2 Hemi-sphere

Give an initial hemi-sphere mesh $\mathcal{T}_h$ (see fig. 23), let S_1 be the approximated 'G^1' surface defined over $\mathcal{T}_h$ using Farin's algorithm. A new mesh, $\mathcal{T}_f$, is obtained by isotropic refinement of the initial one, $\mathcal{T}_h$, positioning the new points over the approximated surface by using the algorithm described in section 6. Now, a measure of L^∞ and L^2 approximation errors between the exact surface S and the two grids are computed. The results obtained are presented in table 5.

Meshes	*Vertices*	*Triangles*	L^∞ *error*	L^2 *error*
Initial mesh ($\mathcal{T}_h$)	66	106	$1.7102 \cdot 10^{-1}$	$1.7139 \cdot 10^{0}$
Final mesh ($\mathcal{T}_f$)	995	1892	$4.9282 \cdot 10^{-2}$	$2.9342 \cdot 10^{-1}$

Table 5. Initial and final mesh 'approximation' errors.

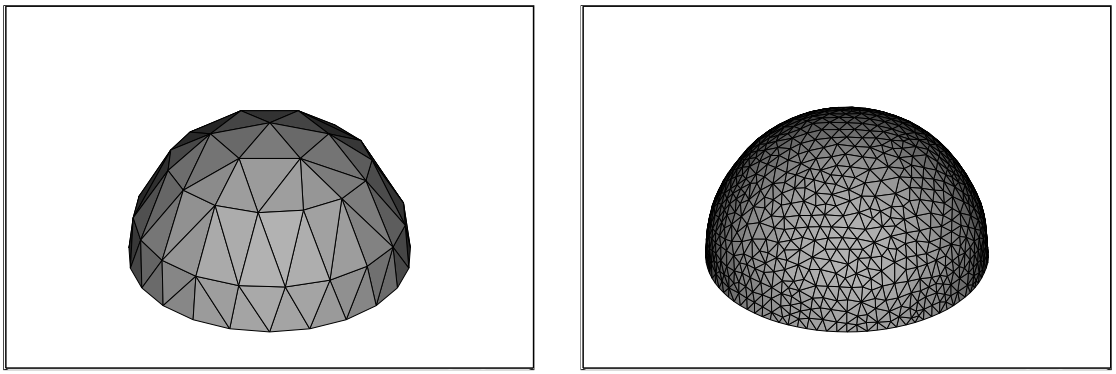

Figure 23: Hemi-sphere initial mesh and final geometrically improved mesh.

As shown in figure 23 and table 5, $\mathcal{T}_f$ is a better approximation of the exact surface S that $\mathcal{T}_h$. It seems that a combined remeshing-'C.A.D. reconstruction' —Farin's algorithm— strategy may be quite powerful in order to obtain better final grid surface definitions.

7.3 Hemi-sphere and half-circle intersection

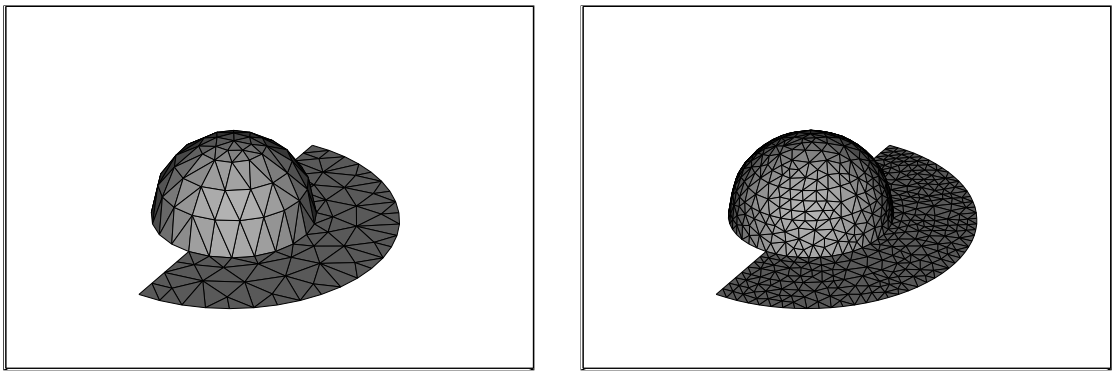

Figure 24: Hemi-sphere and half-circle intersection initial mesh and final refined mesh with a constant isotropic metric.

Let $\mathcal{T}_i$ be an initial mesh of a hemi-sphere and half-circle intersection (see fig. 24). The remeshing algorithm will be tested for this example in different cases : isotropic refinement, discontinuous anisotropic metric and grid element suppression.

Mesh refinement: The following isotropic metric is imposed over $\mathcal{T}_i$ in order to obtain triangles with edge lengths ≈ 0.8:

$$M(\vec{x}) = \begin{pmatrix} 1.5625 & 0 & 0 \\ 0 & 1.5625 & 0 \\ 0 & 0 & 1.5625 \end{pmatrix}, \quad \forall \vec{x} \in \mathcal{T}_i.$$

The final grid, $\mathcal{T}_r$ is shown in figure 24.

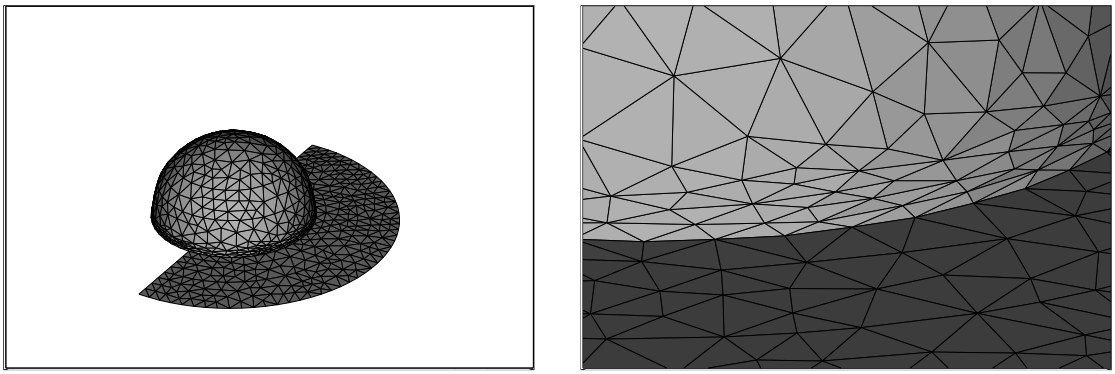

Figure 25: Anisotropic mesh constructed from the previous refined mesh and a zoom of the anisotropic zone.

Discontinuous anisotropic metric: Now, let us consider $\mathcal{T}_r$ as initial mesh. The following discontinuous anisotropic metric is defined over it.

$$M(x,y,z) = \begin{cases} \begin{pmatrix} 1.5625 & 0 & 0 \\ 0 & 1.5625 & 0 \\ 0 & 0 & 64 \end{pmatrix} & \text{if } z \leq .2 \\ \\ \begin{pmatrix} 1.5625 & 0 & 0 \\ 0 & 1.5625 & 0 \\ 0 & 0 & 1.5625 \end{pmatrix} & \text{otherwise.} \end{cases}$$

This metric introduces anisotropic triangles near the intersection of surfaces. In practice, anisotropic triangles must verify that they are ≈ 10 times shorter in the z-direction than in the x-y directions as can be appreciated in figure 25.

Grid element suppression: As in previous example, $\mathcal{T}_r$ will be the initial mesh. Many metrics to suppress grid elements can be considered. Let us take, for example, $\mathcal{M}$ defined by

$$M(x,y,z) = \begin{cases} \begin{pmatrix} 6.25 \cdot 10^{-2} & 0 & 0 \\ 0 & 6.25 \cdot 10^{-2} & 0 \\ 0 & 0 & 6.25 \cdot 10^{-2} \end{pmatrix} & \text{if } z \leq 0 \\ \\ \begin{pmatrix} 8 \cdot 10^{-1} & 0 & 0 \\ 0 & 8 \cdot 10^{-1} & 0 \\ 0 & 0 & 8 \cdot 10^{-1} \end{pmatrix} & \text{otherwise} \end{cases}$$

The final mesh is shown in figure 26. The number of triangles has been reduced from 1116 to 480.

7.4 Falcon (courtesy of Dassault Aviation)

As final example, a more complicated geometry is considered: an initial mesh of a Falcon plane with 1879 vertices and 3762 triangles (see fig. 27) is treated in order to give a geometrically better surface grid definition. The final mesh has 15104 elements and 7550 vertices and its geometrical quality has been improved as can be appreciated in figures 27 and 28.

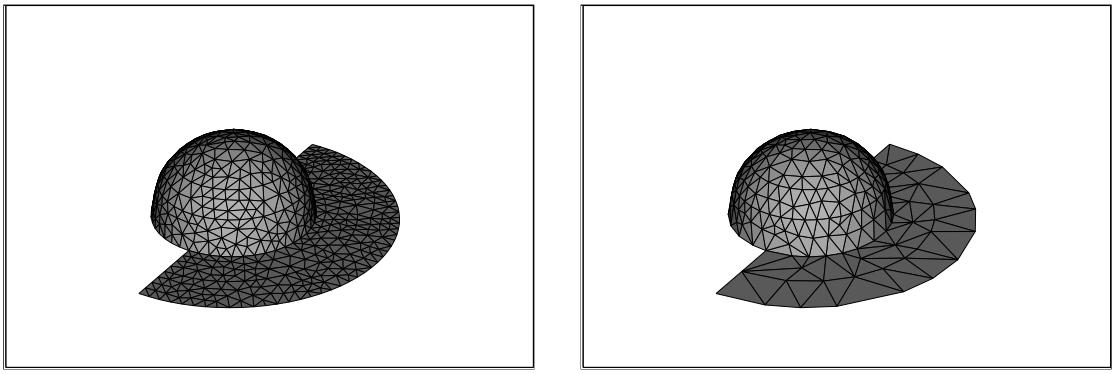

Figure 26: Refined mesh of the intersection of an hemi-sphere and half-circle and the final mesh obtained by suppressing triangles with an isotropic metric.

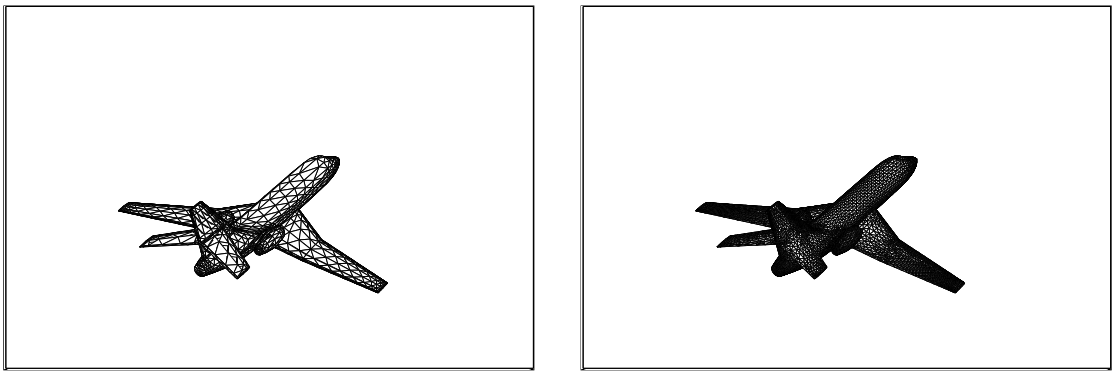

Figure 27: Initial falcon mesh (Courtesy of Dassault Aviation) and final geometrically adapted mesh.

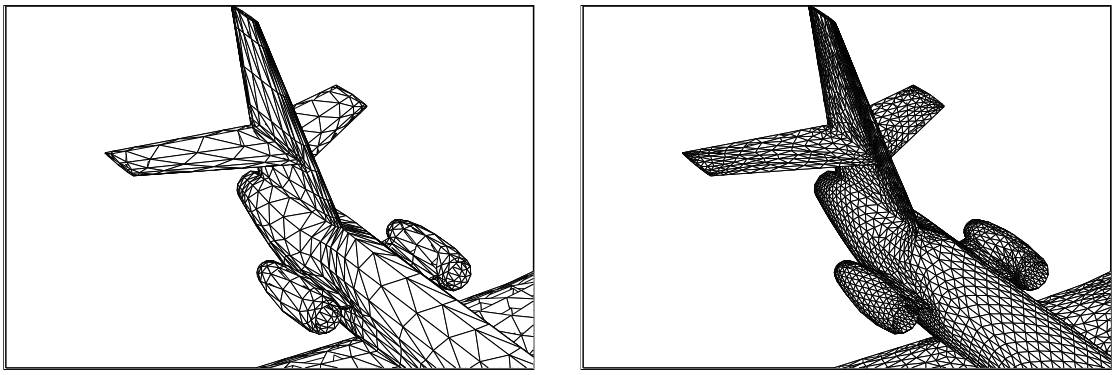

Figure 28: Initial and final falcon mesh zooms.

8 Conclusions

The paper has outlined a new method for anisotropic surface grid adaptation using a metric tensor to control the anisotropy of the mesh. Different examples of generalized Farin's algorithm shows its efficiency. Examples of surface remeshing corresponding to different metrics are given. They show the improvement of the quality of the surface definition or simply demonstrate how anisotropic elements can be generated, added or suppressed. Good results have been obtained in all cases, even with quite difficult geometries. This method also seems to be more flexible and cost-effective than a conventional isotropic mesh generator. Future work will be focused on $3D$ anisotropic mesh generation and on how metrics can be derived using *a posteriori* error estimates.

Acknowledgments

We would like to thank to our friend J. Macias Sánchez for his helpful remarks.

References

[1] W. Boehm, 'Generating the Bézier points of B-splines'. *Computer Aided Design*, **13**, no. 6: (1981).

[2] W. Boehm, G. Farin and J. Kahmann, 'A survey of curve and surface methods in CAGD'. *Computer Aided Geometric Design* **1**, no. 1: pp. 1–60, (1984).

[3] A. Bowyer, 'Computing Dirichlet tessellations'. *Computer Journal* **24**, no. 2: pp. 162–166, (1981).

[4] M.J. Castro Díaz, 'Mesh refinement over surfaces'. *Rapport de recherche INRIA, Rocquencourt* no. 2462, (1994).

[5] P. G. Ciarlet, *The Finite Element Method for Elliptic Problems*. North-Holland, 1977.

[6] G. Farin, 'A construction for the visual C^1 continuity of polynomial surface patches'. *Computer Graphics and Image Processing*, **20**: pp. 272–282, (1982).

[7] G. Farin, *Curves and surfaces for computer aided geometric design. A practical guide*. Academic Press, Inc., S. Diego, 1988.

[8] G. Farin, 'Smooth interpolation top scattered 3-D data'. *In R. Barnhill and W. Boehm, editors. Surfaces in Computer Aided Geometric Design*, North-Holland, (1982).

[9] G. Farin, 'Triangular Bernstein-Bézier patches'. *Computer Aided Geometric Design*. **3**, no. 2: pp. 83–128 (1986).

[10] P.L. George, *Automatic mesh generation, application to Finite Element Methods*. Wiley & Sons, Chichester, 1991.

[11] P.L. GEORGE, F. HECHT, E. SALTEL, 'Automatic mesh generatior with specified boundary'. *Computer methods in applied mechanics and engineering*. **92**: pp. 269–288 (1991).

[12] S. GOPALSAMY AND O. PIRONNEAU, 'Interpolation C^1 de resultats C^0'. *Rapport de recherche INRIA, Rocquencourt*, no. 1000 (Mars 1989).

[13] F. HETCH AND E. SALTEL, 'EMC^2 un logiciel d'édition de maillages et de contours bidimensionnels'. *Rapport Technique INRIA, Rocquencourt*, no. RT-0118 (Avril 1990).

[14] F. HERMELINE, *Une méthode automatique de maillage en dimension n*. Paris VI, Paris, 1980.

[15] C. L. LAWSON, 'Software for C^1 surface interpolation'. *Mathematical Software III*, J. R. Rice (ed.), Academic Press, New York, pp. 161–194, (1977).

[16] C. L. LAWSON, 'Properties of n-dimentional triangulations'. *Computer aided geometric design*, North-Holland, **3**: pp. 161–194, (1986).

[17] L. MOULARD, *Génération et optimisation de maillages en 3 dimensions*. Université Joseph Fourier, Grenoble, 1992.

[18] B. PALMERIO AND A. DERVIEUX, 'Application of a FEM Moving Node Adaptive Method to Accurate Shock Capturing'. *Numerical Grid Generation in CFD*. (1986).

[19] L. PIEGL 'A CAGD theme: Geometric continuity of polynomial surface patches'. *Computer Aided Design*, **10**, no. 19: pp. 556–567, (1987).

[20] L. PIEGL 'A geometric investigation of the rational Bézier scheme in computer aided geometric design'. *Computer in Industry*, **7**: pp. 401–410, (1987).

[21] R. SIBSON 'Locally equangular triangulations'. *The computer journal*, British computer Society **21** no. 3: pp. 243–245, (1977).

[22] M.G. VALLET, *Génération de maillages Éléments Finis anisotropes et adaptatifs*. Université Paris VI, Paris, 1992.

[23] N. P. WEATHERILL ET ALL. 'Grid adaptation using a distribution of sources applied to inviscid compressible flow simulations'. *International Journal for Numerial Methods in Fluids*, **19**: pp. 739–764, (1994).

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399