Fast Adaptive Quadtree Mesh Generation

Pascal J. FREY and Loïc MARECHAL
INRIA, Gamma Project,
Domaine de Voluceau, Rocquencourt,
BP 105, 78153 Le Chesnay Cedex, France.

Abstract. *A size-governed quadtree mesh generation method is presented in this paper to deal with planar domains of arbitrary shape. The tree decomposition provides a convenient control space, which can be used to determine the element sizes, as well as a neighboring space, which allows for the quick searching of mesh items. The sizes of the tree cells are adjusted to match the size specifications (defined as a continuous element-size distribution function in $\mathbb{R}^2$). Hence, the proposed method can be used in the context of mesh adaption in numerical simulations based on the finite element method. Several application examples are provided to emphasize the main features of this approach.*

Keywords. Quadtree, Spatial decomposition, Mesh generation, Mesh adaption.

Introduction

The application of spatial decomposition methods (so-called because they combine quadtree decomposition techniques with quadrant-level meshing procedures) to finite element mesh generation have been pionneered about two decades ago. A survey of the literature devoted to spatial decomposition methods for mesh generation purposes can be found in [Thacker-1980] and [Shephard-1988]. Some of these approaches have yet proved to be robust and reliable and are stilll commonly used in a wide range of engineering applications and even in some commercial packages. One of the main feature of such an approach is its ability to deal with a domain represented by a boundary discretization or to interact directly with a CAD modelling system (to generate both the boundary discretization and the mesh of the domain). In some sense, this type of method have contributed to the avent of generic mesh generation techniques (capable of handling arbitrarily shaped domains), in the same way as advancing-front or Delaunay-based methods [George-1991].

The classical approach consists of two main stages, the domain $\Omega \in \mathbb{R}^2$ is first recursively decomposed into a set of variably sized disjoint cells (*i.e.*, the *quadrants*) representing a partition of a bounding box $\mathcal{B}(\Omega)$ of Ω (cf. Figure 1.(ii)). A second stage consists in subdividing each terminal cell into finite elements (triangles and/or quadrilaterals) so as to complete the mesh of the domain. The elements external to the domain are then discarded from the resulting mesh. Usually an optimization stage based on node smoothing and mesh modifications (by means of geometric or topological operations) is used to improve the shape quality of the mesh elements.

Related work. The use of a quadtree decomposition for meshing purposes was pioneered fifteen years ago by [Yerry,Shephard-1983] and several variants have been proposed (see for instance [Kela *et al.* 1986] or [Perucchio *et al.* 1989]). The element creation stage commonly involves pre-defined patterns (the so-called *templates*) to meet the requirements of conformity (between adjacent quadrants) as well as of efficiency. On the other hand, during the spatial decomposition stage, a filtering operation can be introduced to control the element shape quality and to avoid creating badly-shaped elements. Recent papers have investigated this *a priori* evaluation of the mesh element quality, an upper bound for the angles being eventually exhibited [Mitchell,Vavasis-1992].

To our knowledge, in most of the publications related to quadtree mesh generation, the hierarchical structure is mainly used as a *neighboring space* (used to locate the cells adjacent to a given cell). However, the possibility of using the tree as a *control space* (used to prescribe the desired elements sizes) has not been clearly devised so far (except perhaps for specific Euler computations in CFD [Shephard *et al.* 1988]). If size specifications are supplied, the tree decomposition can be adjusted so that the cell sizes locally match the desired element sizes [1]. In the context of mesh adaption (in which the size specification is usually provided by an error estimate), this feature allows to increase the convergence of the computational scheme and the accuracy of the numerical results [Ciarlet-1978].

[1] as the length of the mesh edges are closely related to the cell sizes.

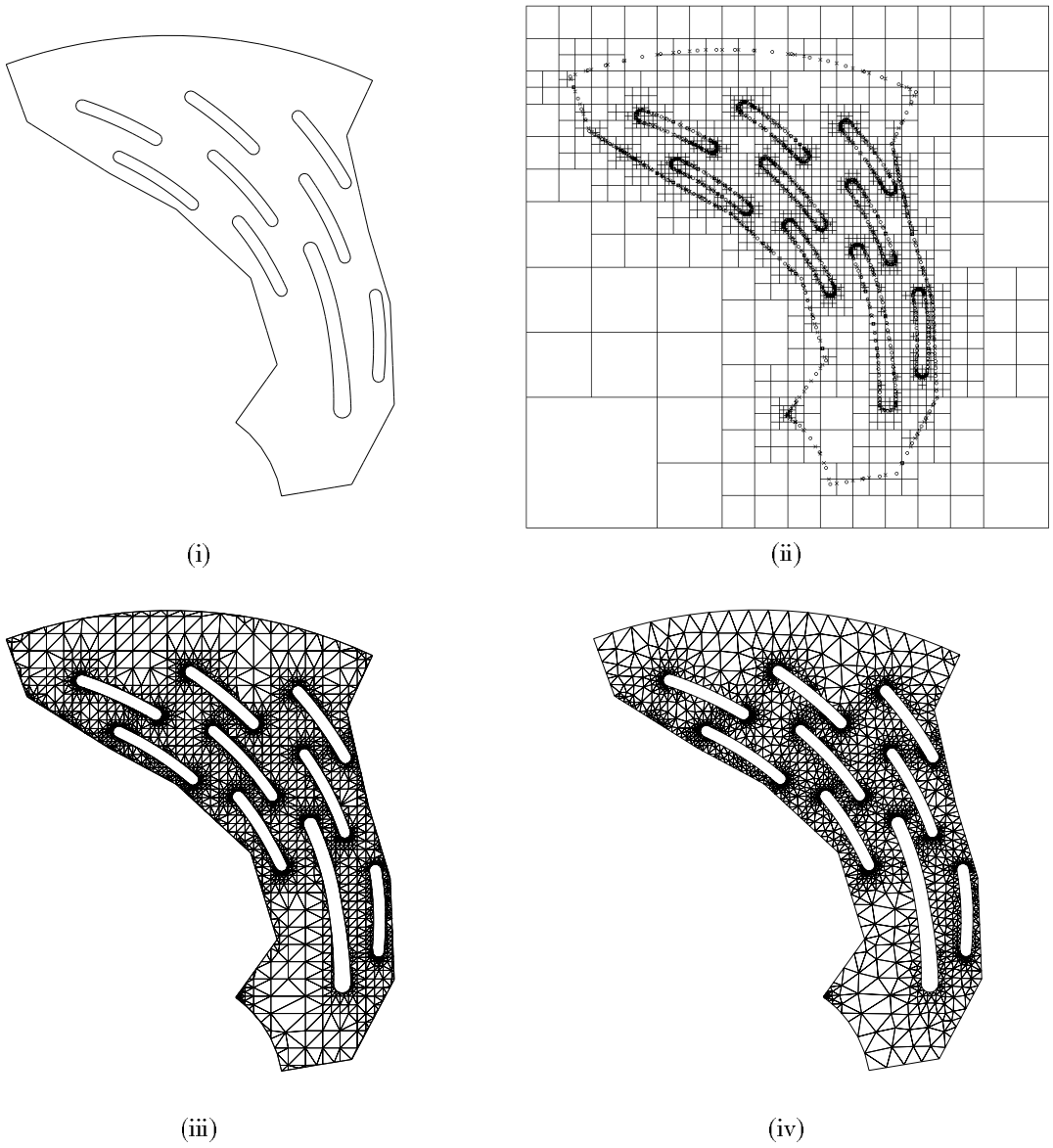

Figure 1: Two dimensional domain Ω, a rotor : domain boundaries (polygonal contour $\Gamma(\Omega)$) (i), spatial decomposition of a bounding box $\mathcal{B}(\Omega)$ (ii), resulting *raw* mesh before optimization (no filtering) (iii) and final mesh of Ω after optimization (iv).

Scope. In this paper, we would like to discuss the tree decomposition stage and propose some improvements of the classical scheme to cope with a given size map. The motivation of this work has been the will to design a robust and efficient algorithm capable of handling two-dimensional domains of arbitrary shape (e.g. non-manifold models as well as domains with features of widely different scales).

A common idea relates the efficiency of the tree decomposition meshing algorithm to the data structures used to represent the tree and to store the mesh. Although these structures contribute obviously to the overall performance of the meshing algorithm (for instance by allowing to reduce the number of tree searching and traversal operations), the most time consuming operation is the filtering step. This treatment attempts to remove closely spaced entities in a terminal quadrant and therefore involves extensive topological checks to maintain the topological consistency of the decomposition. Thus, for efficiency purposes, the proposed approach does not include a filtering stage, which is then replaced by an adequate mesh optimization procedure.

Outline. This paper is divided into five sections. Section 1 recalls the terminology used with quadtree decompositions and summarizes the general scheme of the classical quadtree mesh generation. Section 2 proposes a governed quadtree mesh generation algorithm and eventually discusses the boundary discretization problem. Section 3 presents the extension of the general scheme of the size-governed quadtree algorithm in the context of mesh adaption. Several application examples are proposed Section 4 to emphasize the main features of the proposed approach. A brief section concludes the paper by mentioning the possible extensions of this work.

1 Quadtree decomposition

In this section, we recall some terminology and the basic definitions of the spatial decomposition structures in two dimensions [Samet-1984]. The input of the problem is an arbitrarily shaped domain $\Omega \in I\!R^2$, represented by a boundary representation (*i.e.*, a polygonal contour $\Gamma(\Omega)$ described by a set of vertices $\mathcal{V}$ and a list of edges $\mathcal{E}$).

Some terminology. The basic concept of any spatial representation consists of enclosing the domain Ω into a bounding box (usually a square), denoted $\mathcal{B}(\Omega)$, corresponding to the *root* of the spatial decomposition tree. This box is subdivided into four equally-sized *cells* (the *size* of a cell c is the length of a side of c), each of which being recursively subdivided several times. The *stopping criterion* used to subdivide a cell can be based on the local geometry of the domain (e.g., the local curvature of the boundary) or user-defined (e.g., the maximum level of refinement).

The four vertices at the corners of a cell are called *corners*. The edges connecting each consecutive corners are the *sides* of the cell. The edges of the decomposition that belong to the boundary of the cell are called the *edges* of the cell. Hence, each side of a cell contains at least one edge. Two cells are *adjacent* if they share an edge. Neighbors are identified by the four cardinal directions, $\{N, S, E, W\}$ and the quadrants obtained by subdividing a cell are identified by their relative position within the parent cell : $\{NW, NE, SW, SE\}$. The *level* of a cell corresponds to its depth in the related tree, *i.e.*, the number of subdivisions required to obtain the cell. The bounding box is at level 0. The *depth* of the tree corresponds the maximum level of subdivision. Any cell that is not subdivided is a *terminal* cell or a *leaf*.

At each stage, any cell can be subdivided into four sub-cells or not. Hence, the resulting decomposition tree may be quite unbalanced (the number of subdivision levels being not constant in each cell). This remark leads to introduce the following rule[2] : any side of a terminal cell must contain at most one corner[3] This condition is known as the [2:1] *rule* that was first suggested by [Yerry,Shephard-1983].

Operations on quadtrees. Two types of operations are frequently applied with trees : topological operations (e.g. quadrant creation, finding the cells adjacent to a given cell in a given direction) and geometric operations (e.g. finding the cell that contains a given point, finding the intersections between an edge and the current cell). These operations have not all the same frequency and not the same computational cost. The *neighbor finding* operation concerns the search of the cells adjacent to a given cell and is frequently called during the quadtree decomposition, for instance by a post-processing algorithm used to enforce the [2:1] rule.

[2] which is also used to control the mesh gradation (*i.e.*, the size variation between neighboring elements.)
[3] This condition is equivalent of writing that the sizes of any two adjacent cells differ by at most a factor two.

Data structures. Three approaches are possible to represent trees [Knuth-1975] :

- A tree structure using pointers. In this obvious approach, each internal cell requires four pointers, one for each of the subtrees and a bit of information to indicate the cell classification (internal or terminal). In addition, extra pointers can be added to improve the efficiency, such as links to parent cell, links to neighboring cells, ...
- A list of the nodes encountered by a traversal of the structure. With this implementation, intersection algorithms can be efficiently performed, although other algorithms may be less efficient. For instance, visiting the second subtree of a cell requires to visit each node of the initial tree to locate the root of the subtree.
- A system of locational codes (e.g. binary encoding), for instance with a *linear quadtree*.

The information requirements almost dictate the choice of a representation : for instance the need to quickly identify neighbors leads to favor a linear structure or a pointer-based structure containing pointers to the parent cell and to the adjacent cells. We have retained this last representation as most of the treatments are localized and involve mainly the adjacent cells of a given cell (for instance the balance condition).

General scheme. The aim of the quadtree-based mesh generation approach is to obtain a valid and accurate representation of a planar domain by a set of triangles suitable for finite element analysis. More specifically, the domain is decomposed into a set of cells that have a size distribution compatible with the desired mesh gradation and store all the information required by the meshing step to generate a finite element mesh of the domain. Several features of the method are common to all quadtree-based meshing techniques :

- the data structure is used for localization and searching purposes,
- the mesh generation is twofold, first the tree is generated, then the mesh is created,
- the cell entities (corners, edges) are mesh entities (vertices, edges),
- the mesh gradation is controlled by the level of refinement of the cells (for instance, using the [2:1] rule).

Schematically, the classical quadtree-based meshing approach consists of three successive steps and can be summarized as follows :

1. the *quadtree construction* : decomposition (insertion of the entities of $\mathcal{V}$ and $\mathcal{E}$, balance enforcement ([2:1] rule), filtering and warping of the quadrant entities,
2. the *mesh generation* : point and element creation (based on templates),
3. the *mesh optimization* : node smoothing, topological and geometric mesh modifications.

The next section explains how this approach differs from this general scheme to account for a specified size map.

2 Governed quadtree mesh generation

Let consider a planar domain $\Omega \in I\!R^2$ represented by a polygonal contour $\Gamma(\Omega)$ described by a list $\mathcal{V}$ of points and a list $\mathcal{E}$ of edges (*curved sections* in the boundary description are allowed). In addition to this input data, a size map can be supplied to prescribe the sizes of the mesh elements. In this section, we will explain how the size specifications can be taken into account during the tree construction stage.

2.1 Control space

To govern the internal point and element creation, it is convenient to use a control space. Formally speaking, this control space is defined as follows [George-1991]:

Definition 2.1 *(Δ, H) is a control space associated with a mesh $\mathcal{T}$ of a domain Ω if*

- *$\Omega \subseteq \Delta$ where Δ is a covering up of Ω,*
- *$\forall P \in \Delta, \quad \exists H(P, \overrightarrow{d})$, where $\overrightarrow{d}$ is a direction of the disk S^1 : $H(P, \overrightarrow{d}) : \Delta \times S^1 \longrightarrow I\!R$.*

From the geometric point of view, it is obviously convenient to consider the tree decomposition as the desired covering up Δ. To construct the function H, we will use a Q^1 interpolation, from the sizes h_{P_i} associated with the vertices of the tree cells. If no size map is provided, an *intrinsic* size map is computed. A discrete sizing function is obtained from the tree decomposition, at each quadrant vertex P the size h_P can be defined by averaging the Euclidean lengths of the cell sides incident to this vertex. Using a generalized interpolation scheme, a continuous size function H can be defined at any point of the computational domain Ω. Note that this sizing function accounts for the mesh gradation as controlled by the [2:1] rule.

On the other hand, if a size function is supplied, it can be used to govern the tree decomposition. The element sizes are commonly specified by :

- parameters associated with domain entities,
- values associated with the vertices of the previous mesh (in an adaption scheme),
- parameters related to the domain geometry (e.g. the local curvature).

As the elements created in a quadrant almost span the cell, the length of a mesh edge is then approximately equal to the cell size containing it. Let consider a point P in a tree cell c. If the desired size h'_P differs from the intrinsic size h_P at P (based on the current decomposition) and is such that $\frac{h_P}{h'_P} > \sqrt{2}$, the cell is refined into four sub cells and the new cell containing P is recursively analyzed. At the same time the cell is divided into four sub-cells, the intrinsic sizes are updated at the tree vertices and the tree is balanced. At completion of this procedure, the intrinsic size function and the specified size function are close (the ratio between h_P and h'_P is bounded by the value $\sqrt{2}$).

2.2 Boundary discretization

The quadtree method can either generate the boundary representation of the domain along with the domain covering-up or start fom a given polygonal representation and create internal elements only. The first approach has been commonly adopted by several authors (the mesh generation algorithm being interfaced with a geometric modelling system [Grice *et al.* 1988], [Shephard-1988]). In the proposed approach, the boundary discretization is clearly disconnected from the tree decomposition stage. This choice is performed to disconnect the quadtree approach from any modelling system and to allow a better control on the geometric approximation of the domain boundaries.

Curvature-based refinement (*i.e.*, finer meshes in highly curved regions and coarser meshes in regions of low curvature) provides a convenient means of controlling the geometric approximation of the mesh elements. Basically, the goal is to obtain a polygonal approximation of the curve $\Gamma(\Omega)$ that deviates from the underlying true geometry by no more than a (user) given tolerance value ε.

Geometric support construction. The set of polygonal segments of $\Gamma(\Omega)$ is a best approximation of the underlying continuous curve Γ and thus represents a geometric support of the curve. The construction of the polygonal support is such that the distance between any segment and the portion of the curve it represents must be bounded. Hence, refinement will be more important in highly curved regions.

Let h be the distance between a point and the supporting edge, let d be the length of the edge, we like to have $h/d < \varepsilon$, ε being the desired tolerance. If the tolerance is not exceeded, the segment is correct. Otherwise, the segment is subdivided into two sub-segments, each of them being recursively analyzed. The resulting set of segments represents the *geometric support* of the curve (cf. Figure 2, right).

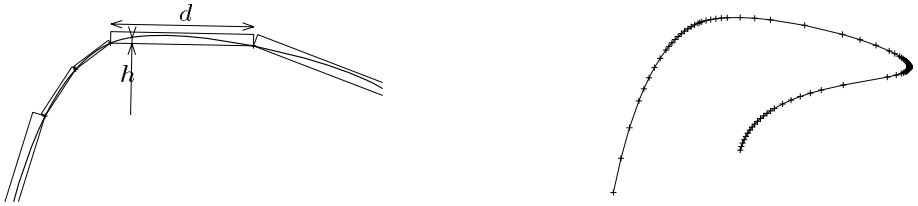

Figure 2: Approximation using a polygonal segment with respect to a user-specified tolerance value $\varepsilon = 0.08$ (left), $\varepsilon = 0.01$ (right).

Geometric approximation. Curvature-based mesh generation requires the ability to evaluate the local curvature of a curve or a surface[4]. In our approach, the geometric support is constructed using a third order polynomial approximation (Hermite function) from the discrete tangents. Given a polygonal segment P_iP_{i+1}, the direction of the tangent τ_{P_i} at P_i is parallel to the line $P_{i-1}P_{i+1}$, its module is given by $\|\overrightarrow{P_{i-1}P_i}\|$. The desired element size h_P at P must then be proportional to the radius of curvature r_P at P, $h_P = \alpha r_P$. The control of the geometric approximation consists in defining α so that, for a given deviation value ε,we have : $\delta/r_P \leq \varepsilon$ (the osculating circle of radius r_P being a second order approximate of the curve at P). Having two parameters ε and α, for the desired size, their relation is : $\alpha \leq 2\sqrt{\varepsilon(2-\varepsilon)}$ [Frey,Borouchaki-1998].

Boundary discretization. The boundary discretization aims at defining a polygonal approximation of the boundaries curves that conforms to a given tolerance value. To this end, once the geometric support is constructed, the initial polygonal discretization is analyzed, based on the edge lengths computation. The size map proportional to the radii of curvature provides a size function that will enable us to compute the length ℓ_{AB} of any segment AB as :

$$\ell_{AB} = (s_B - s_A) \int_0^1 \frac{1}{H(s)} dS, \tag{1}$$

where s_A, s_B are the curvilinear abcissa and $H(S)$ represents an interpolation function of the size function $h(s)$ along the segment $[s_A, s_B]$, $S = (s - s_A)/(s_B - s_A) \in [0, 1]$. The problem is then to split the segment AB into n equally sized unit sub-segments (*i.e.* having a length close to 1) with respect to the given size map [Laug *et al.* 1996]. The resulting polygonal segment will serve as input for the quadtree decomposition.

2.3 Quadtree decomposition

The objective of the tree decomposition stage is to relate the desired elements sizes to the depth of the tree. This can be achieved by mesh parameters specifications. The quadtree representation of a domain Ω is defined from a square (the bounding box $\mathcal{B}(\Omega)$) that fully encloses the points of $\mathcal{V}$. This box is then subdivided into four quadrants, each of them being then tested recursively to decide whether it needs further refinement. The process stops when each quadrant contains less than two points and when the domain is resolved to a satisfactory resolution.

Quadtree construction. The depth p of the tree can be related to the length h of a mesh edge and the size b of the bounding box $\mathcal{B}(\Omega)$ as : $p = \log_2(b/h)$. As the size h is proportional to the minimal of the principal radii of curvature (in the intrinsic size map), we obtain a lower bound for the depth of the tree : $p \geq \log_2(b/(r\alpha))$, where α is the coefficient introduced before. This result can be used to define a stopping criterion in the tree decomposition stage.

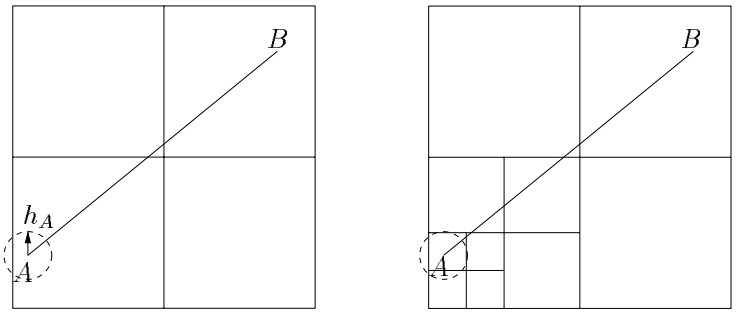

Figure 3: *Tree decomposition using a prescribed size. The size at point A forces the initial tree box containing A (left hand-side) constructed without explicit sizing prescription to be decomposed two levels deeper (right hand-side) to adjust to the size h_A represented by the dashed circle.*

Stopping criteria. Since the domain is described as a list of points and a list of edges, the stopping criterion must account for both types of entities : each leaf contains at most one connected component of $\Gamma(\Omega)$. If a leaf contains no point of $\mathcal{V}$, then it contains at most one edge or portion of an edge of $\mathcal{E}$. An additional criterion is introduced, each quadrant side must contain at most one boundary intersection point, except if the quadrant contains a point of

[4]This measure can be easily computed from the derivative information or returned by a modelling system [doCarmo-1976]

$\mathcal{V}$. Notice that special attention must then be paid to corners (vertices where two edges form an accute angle) and non-manifold points (points having more than two incident edges). The cell containing such points is not subdivided, thus violating deliberately the [2:1] rule. This will prevent the tree from degenerating when trying to separate closely spaced edges sharing a common vertex.

The tree construction consists of introducing successively all entities of $\mathcal{V}$ and all entities of $\mathcal{E}$ (cf. Figure 4). Notice that the tree decomposition may change dramatically the initial discretization as it always attempt to separate closely spaced entities that belong to different connected components (cf. Figure 9, enlargement).

Remark 2.1 *To avoid numerical roundoff problems, the vertices of $\mathcal{V}$ that coincide with a quadrant corner or lie on a quadrant side have their coordinates slightly changed. The original coordinates will be restored after the* raw *is created.*

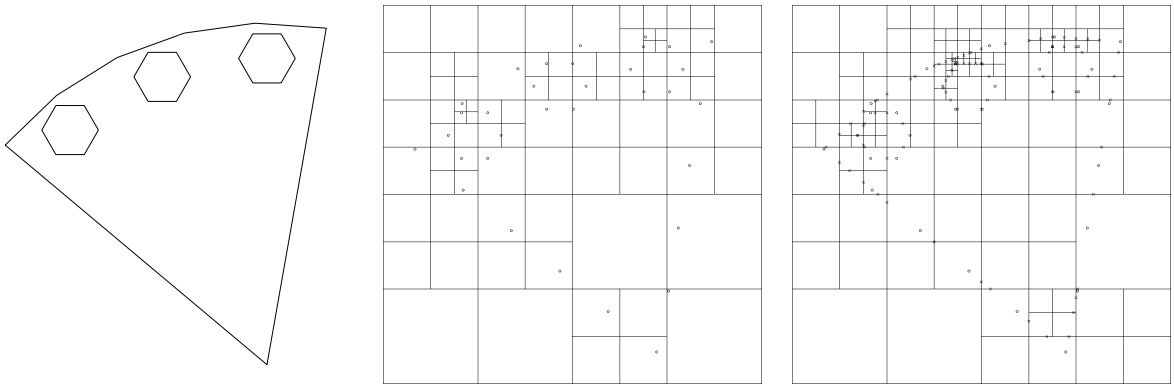

Figure 4: *Quadtree decomposition : initial domain (left), after the insertion of the points of $\mathcal{V}$ (middle) and after the insertion of the segments of $\mathcal{E}$ (right).*

Balance condition. Partly to control the mesh gradation and also to simplify the element creation step (*i.e.*, to reduce the number of templates), the levels of adjacent quadrants are checked to see if two neighboring cells differ by more than a factor two in size (according to the [2:1] rule). This procedure results in more quadrant splits to propagate accross the structure, thus increasing the total number of terminal cells. The procedure is carried out during the construction stage (without first constructing the unbalanced quadtree). At the same time a cell is subdivided, the sizes associated with the mesh vertices are updated to preserve the underlying continuous size map.

Remark 2.2 *As no check on the edge-quadrant intersections is performed, newly created vertices may be close to quadrant entities (corners or sides), thus leading to poorly-shaped elements. However, the extensive amount of topological checks required to preserve the integrity and the conformity of the domain does not seem justified, especially as a clever mesh optimzation is able to remove this undesirable elements.*

2.4 Mesh generation

The sequence of operations has concerned so far the quadtree structure. The topology of the final mesh is constructed based on the quadrant classification : all terminal cells are visited by a tree traversal procedure and the cells are triangulated accordingly. The final mesh is the union of all the quadrants triangulations. The balance procedure leads to a substantial reduction of the number of possible transition cases and allows the introduction of predefined basic patterns (the so-called *templates*). Hence, the generation of triangles in quadrants is really straightforward.

Actually, an internal quadrant can have any combination of the four neighboring quadrants, thus corresponding to $2^4 = 16$ possible triangulations (that can be reduced to only six templates using the various symmetry properties). Practically, a four bit index is created to serve as a pointer into a table that gives all quadrant triangles corresponding to a given quadrant configuration.

However, if a box contains a vertex of the boundary discretization, the treatment is slightly different. This point is star-shaped with respect to the other quadrant entities (corners or additional intersection points along the sides). Thus, this point is simply connected to all quadrants vertices so as to create a triangulation of the quadrant. On

the other hand, if the quadrant contains intersection points, an element edge must connect two intersection points (or two corners) so as to result in a valid triangulation of the portion of the domain enclosed between the boundary $\Gamma(\Omega)$ and the boundary of the internal mesh (cf. Figure 5).

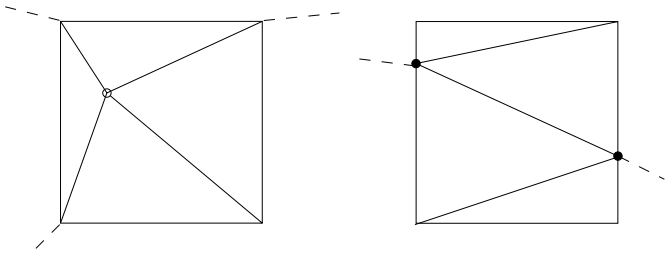

Figure 5: *Triangulation of the boundary quadrant : for a vertex of the boundary discretization (left) and for intersection points (right).*

Remark 2.3 *This meshing technique is intrinsically very simple to implement. However, one has to notice that the resulting mesh is a mesh of the bounding box $\mathcal{B}(\Omega)$ rather than a mesh of Ω only. In other words, many elements external to the domain have been created that need to be removed at this stage. Therefore, a simple coloring scheme is applied to identify the possible sub-domains and to remove the external triangles. This procedure retrieve the different connected components of the domain using adjacency relationships between the triangles.*

2.5 Mesh optimization

As no filtering stage is applied on the terminal quadrants, a (small) number of poorly-shaped elements can be created in the *raw* mesh. The first step in mesh optimization consists in removing these elements using a specific procedure. A basic observation shows that two types of porrly-shaped elements can occur, depending on whether a vertex is too close from a quadrant corner (Figure 6, left) or from a quadrant side (Figure 6, right). Any combination of these elements can potentially be created in the boundary quadrants (cf. Figure 7).

Figure 6: *Poorly-shaped element due to quadrant-model interactions, a* needle *(left) and a flat element (right).*

During the tree decomposition stage, the points that may cause the creation of bad elements are tagged as well as the elements having these points as vertices. Then an edge collapsing technique is iteratively applied to remove the badly-shaped elements, in combination with edge swapping operations to locally improve the mesh quality. The identification of the edges too small is made possible by use of the normalized edge length with respect to the current size map. At completion of the process, no unacceptable element remains left in the mesh.

Remark 2.4 *The use of the normalized edge lengths preserve the small features in the mesh, for instance in highly curved regions.*

The next step consists of pulling the newly created nodes on the boundary using the geometric support. As the geometric approximation has been controlled, this operation does not result in invalid or overlapping elements.

The triangulation of internal quadrants leads to well-shaped elements, for which the aspect-ratio is bounded *a priori*. However, the aspect ratio of triangles created in boundary quadrants can be significantly degraded as the triangulation of such quadrant is arbitrary and is, for instance, related to the relative position of the point of $\mathcal{V}$ contained in this quadrant with respect to the quadrant sides. Therefore, the resulting mesh requires some amount of mesh quality improvement.

Aspect ratio. The aspect ratio of a triangle K is defined as :

$$\mathcal{Q}_K = \alpha \frac{h_{max}}{\rho_K} = \alpha \frac{h_{max} p_K}{S_K}, \tag{2}$$

where h_{max} is the *diameter* of K, (*i.e.*, its longest edge), ρ_K is the in-radius of K, p_K is the half-perimeter of K and S_K is the surface of K. The coefficient $\alpha = \sqrt{3}/6$ is a normalization coefficient such that the aspect ratio value of a regular triangle is equal to 1. The aspect ratio measures the degradation of the element shape quality and is a value ranging from 1 to ∞.

The objective of the mesh optimization procedure is to improve the overall shape quality of the mesh (*i.e.*, to skew the histogramm of shape quality towards the left). In addition of improving the shape quality, this procedure also accounts for the size quality of the mesh. The size quality is measured by means of the normalized lengths of the element edges. The optimal size quality is 1.

The tools used to perform mesh optimization are twofold, geometric mesh modifications (node relocation) and topological modifications (edge collapsing, edge swapping).

Remark 2.5 *As the quadtree decomposition provides a good distribution of internal mesh vertices, there is non need to introduce new nodes by means of edge splitting.*

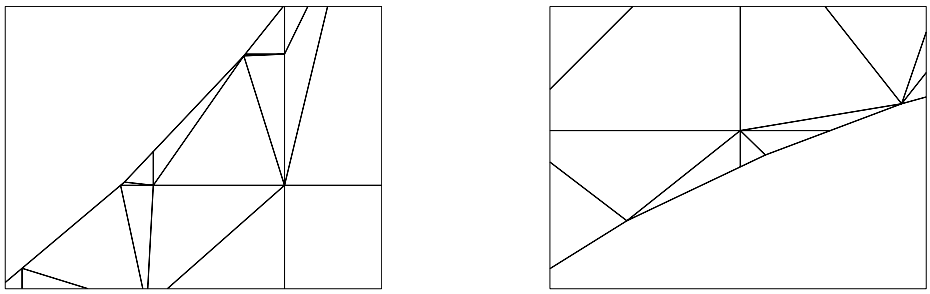

Figure 7: *Curved domain boundary, different combinations of poorly-shaped elements in the vicinity of the boundary.*

These operations are performed sequentially provided the element shape quality improves. The point relocation procedure is based on a weighted barycentrage technique that accounts for optimal edge lengths, the computation of the optimal points is based on the unit length. In the isotropic case, a simple Laplacian smoothing [Freitag-1997] or a Lagrangian relaxation procedure [George-Borouchaki-1997] can also be applied. Figure 8 shows the result of mesh optimization on a computational CFD domain.

3 Mesh adaption

The main purpose of a computational adaption scheme is to ensure the reliability of the finite element solutions. In this respect, a complete adaptive scheme usually includes a solver, an error estimate and a mesh adaption procedure. The result of the error estimation is translated in terms of sizes specifications. This sizing requirement is used to obtain a new tree decomposition and consequently a new mesh that conforms better to the desired sizes than the previous decomposition. The general scheme can be summarized as follows :

1. construction of an initial mesh $\mathcal{T}_{i=0}$,
2. computation of the solution on the current mesh,
3. solution analysis using an *a posteriori* error estimate,
   - if the solution is converged, end.
   - else
     - construction of a discrete size map,
     - construction of a mesh $\mathcal{T}_{i+1}$ governed by the above size map,
     - back in step 2.

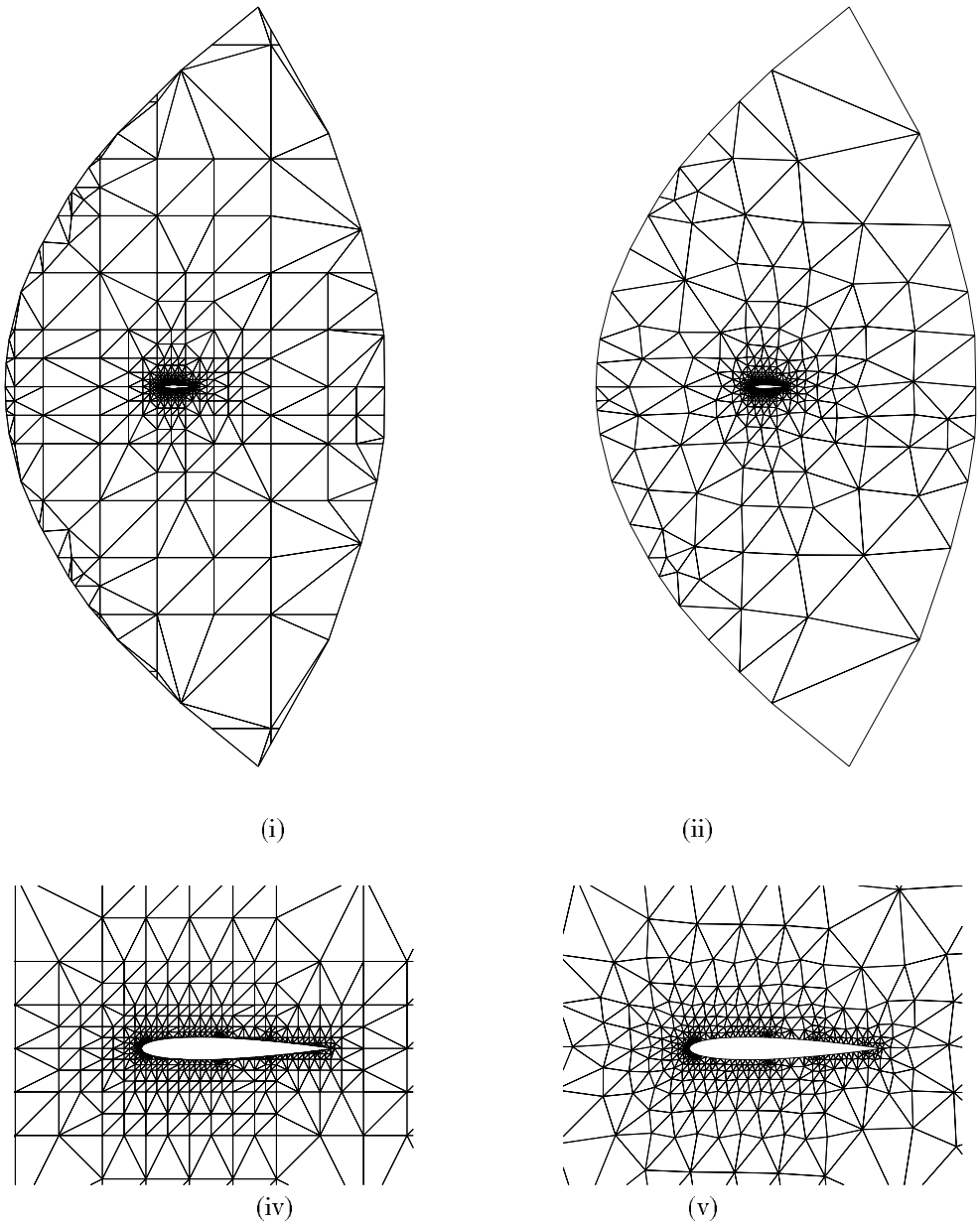

Figure 8: *Computational CFD mesh around a Naca012 airfoil : initial quadtree mesh (i) vs. final optimized mesh (ii). Local enlargments around the airfoil, original mesh (iii) and optimized mesh (iv).*

The size specifications are associated with the vertices of the current mesh and define a so-called *discrete* size map. This discrete map can be used to define a control space (cf. Section 2.1) and to govern the tree decomposition procedure.

In the adaptive context, the tree is constructed based on the boundary discretization of the domain and the specified sizing function. Usually, the current mesh is used as the covering-up Δ in the control space and the discrete size map is the function H. In our approach, there is no need of using the current mesh as control space as the tree can be used to serve this purpose. As explained previously, the tree is refined according to the desired size specified at the boundary entities. Then, the insertion of each vertex of the current mesh is simulated to see whether or not the cell containing it has the desired size. More precisely, the desired size h' at the vertex is compared with the size h obtained by a bilinear interpolation in the cell. If the ratio $h/h' > \sqrt{2}$ then the cell is refined. It is then straightforward to create a size conforming quadtree decomposition.

The element creation is exactly the same as that of the classical scheme. The mesh optimization procedure remains also unchanged as care has been taken to preserve the size of the element during the mesh modifications. The use of normalized edge lengths with respect to the metric map prevent the small elements to be removed. The aim of the optimization stages is to provide a unit mesh (*i.e.*, a mesh having all its edges of unit length w/r size map). In this context, the node smoothing procedure (moving a vertex P) is equivalent of finding an *optimal* point O_j such that each edge P_jP incident to P is of unit length :

$$O_j = P_j + \frac{\overrightarrow{P_jP}}{\ell_{\mathcal{M}}(P_j, P)}\,, \tag{3}$$

where $\ell_{P_j,P}$ is the length of edge P_jP with respect to the size variation function associated with the edge.

A length efficiency index is used to measure the mesh quality with respect to the metric map. Let ℓ_{AB} be the length of the edge AB with respect to the given size map. The *efficiency index*, denoted τ, of a mesh $\mathcal{T}$ can be 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 :

$$\tau = 1 - \frac{1}{na}\sum_{i=1}^{na} e_{AB}^2\,, \tag{4}$$

with $e_{AB} = 1 - \ell_{AB}$ if $\ell_{AB} < 1$ or $e_{AB} = 1 - \ell_{AB}^{-1}$ if $\ell_{AB} > 1$. This coefficient seems adequate to quickly estimate the mesh quality with respect to a given metric map. In particular, it indicates that the edge lengths are l times too long or too short (a value $\ell = 5$ or $\ell = 0.2$ means that all edges are 5 times too long or 5 times too short) as compared with the desired sizes. The optimal value being $\ell = 1$, any value $\ell > 0.91$ ensures a reasonable mesh quality with respect to the given metric map.

4 Application examples

Several application examples are provided to illustrate the main features of the proposed algorithm. These examples concern realistic objects as frequently encountered in numerical simulations using the finite element method. Table 1 reports statistics for the quadtree meshes. In this table, N_p, N_e represent the number of vertices and elements, $\mathcal{Q}_{min}$ and $\mathcal{Q}_{avg}$ are the worst and the average shape quality values of $\mathcal{T}$, ℓ_{min}, ℓ_{max} and ℓ_{avg}, e_i correspond to the minimum, maximum, average edge lengths and efficiency index values. The values *cpu* and *speed* indicates the total cpu time and the speed up (number of triangles per second) of the method, as measured on a HP 9000 PA8200, 200Mhz workstation. This cpu time correspond to the tree decomposition and the generation of the final mesh, neglecting the external elements[5], but including the optimization procedure. The mesh elements generated using this method are globally well-shaped and conform to the desired sizes. However, the average edge length is closer to $\sqrt{2}$ than to the unit length (this is most likely due to the discrete [2:1] transition introduced during the tree decomposition). The efficiency index confirms that the elements sizes are compatible with the desired sizes.

In the next example, the size map has been analytically defined. Two metric fields are specified inside the computational domain. The size functions have been defined at any point $P(x, y)$ as follows :

$$h1(x,y) = 4|r_1 - \|\overrightarrow{C_1P}\|| + 0.05 \quad \text{and} \quad h2(x,y) = 2|r_2 - \|\overrightarrow{C_2P}\|| + .02\,,$$

[5]Note : in most cases, more than half the number of elements are outside the computational domain or discarded during the optimization stage.

mesh	N_p	N_e	$\mathcal{Q}_{min}$	$\mathcal{Q}_{avg}$	ℓ_{min}	ℓ_{max}	ℓ_{avg}	τ	cpu	speed	Fig
naca012	683	1,218	3.2	1.28	0.36	1.6	0.74	0.92	0.06	20,300	8
cooler	2,204	3,794	2.9	1.29	0.25	1.92	0.75	0.91	0.17	22,317	9
rotor8	3,533	6,024	3.22	1.28	0.31	2.38	0.76	0.92	0.25	24,096	1
px12	6,003	10,248	2.7	1.26	0.26	1.84	0.71	0.92	0.48	21,350	
tbm94	10,405	17,908	2.5	1.26	0.34	1.72	0.73	0.92	0.93	19,256	
fourche	46,823	80,221	3.57	1.25	0.24	1.52	0.73	0.92	4.08	19,662	

Table 1: *Statistics relative to the numerical evaluation of the quadtree meshes.*

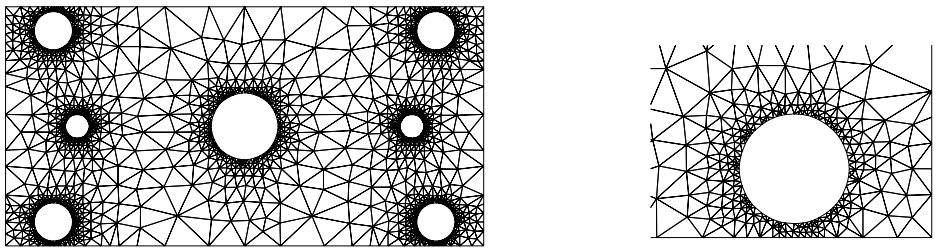

Figure 9: *Final optimized mesh of 'cooler' (left) and local enlargement (right).*

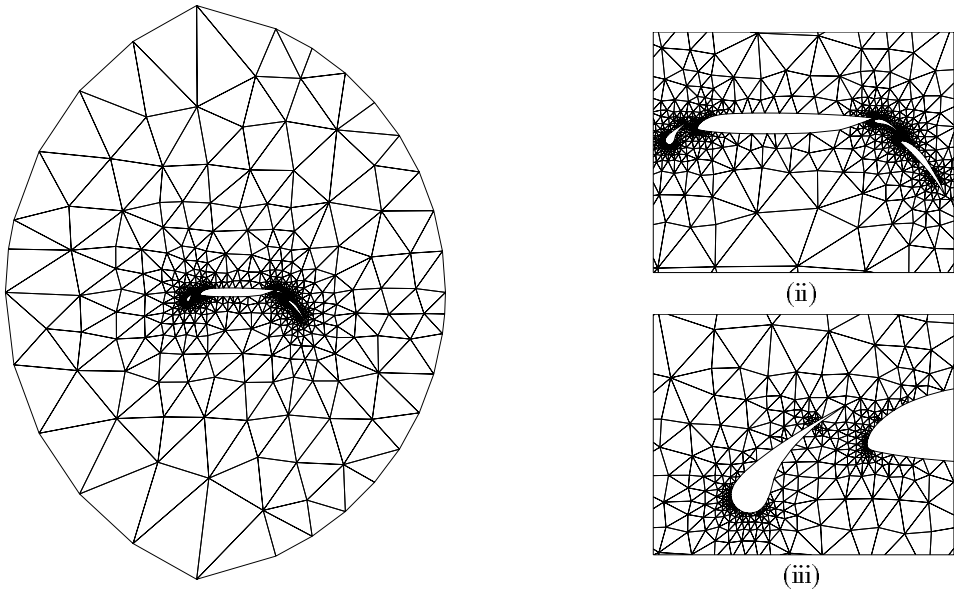

Figure 10: *Computational CFD mesh around a multi airfoil (i) and local enlargments around the airfoil (ii) and (iii).*

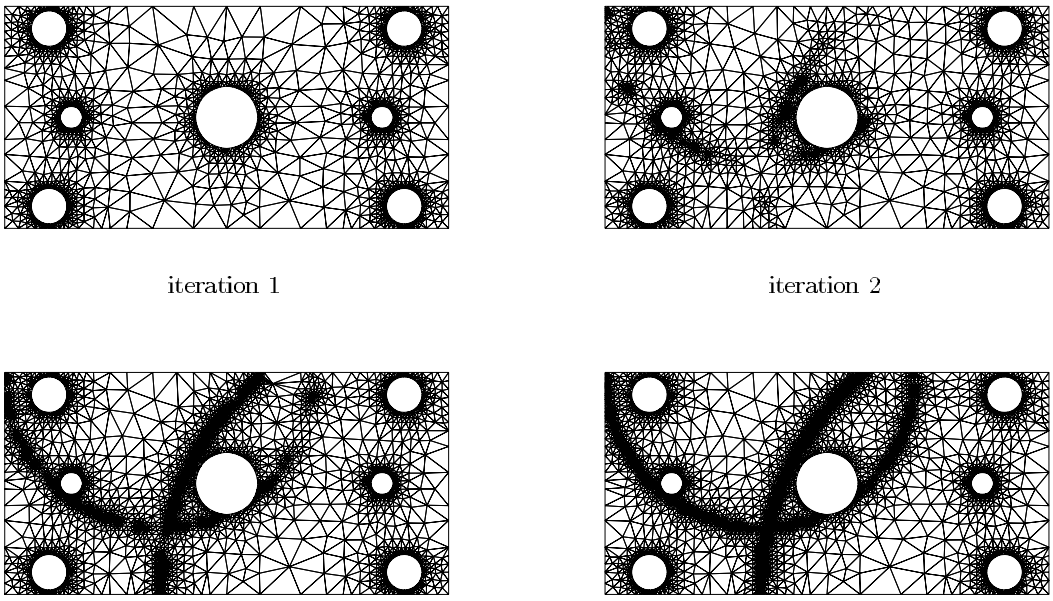

Figure 11: *Example of mesh adaption where two metric fields have been specified. The meshes illustrate iterations 1 (initial mesh), 2, 4 and 8 respectively.*

where $C_1 = (-3, 5)$ and $C_2 = (10, -5)$ are the centers of two disks of radius $r_1 = 7$ and $r_2 = 13$ respectively. Eight iterations of tree and mesh adaption haven been necessary to capture the size fields. Table 2 and Figure 11 illustrate the result of quadtree adaption for this analytical example. Notice that the tree decomposition favors the creation of small edges, and more especially with this analytical function as the mesh gradation given by the sizing function is not compatible with the (almost) discrete gradation introduced by the [2:1] rule. Nevertheless, the tree decomposition is able to capture the behavior of the function.

mesh - iter	N_p	N_e	$\mathcal{Q}_{min}$	$\mathcal{Q}_{avg}$	ℓ_{min}	ℓ_{max}	ℓ_{avg}	τ
cooler - 1	2,204	3,794	2.9	1.29	0.24	8.33	0.75	0.88
cooler - 2	2,808	4,975	2.9	1.28	0.21	21.13	0.77	0.89
cooler - 4	5,090	9,502	3.64	1.27	0.24	29.9	0.75	0.90
cooler - 8	7,772	14,847	2.9	1.27	0.24	1.78	0.74	0.92

Table 2: *Statistics relative to the numerical evaluation of the adapted quadtree meshes.*

5 Conclusions

After a brief review of some elementary definitions related to quadtree-based mesh generation, we have introduced a method suitable to create meshes conforming a prescribed size map (or an intrinsic size map if no specific map was specified). The proposed method attempts to adapt the tree decomposition so that the cell sizes match the required local sizes. No filtering stage is required to remove closely spaced quadrant entities as a specific mesh optimization procedure has been designed to take care of these small features. The mesh optimization procedure is based on the normalized lengths of the mesh edges and aims at providing an optimal, unit size, mesh with respect to the given metric map. This approach can be used in an adaption scheme were the size map is provided by an a posteriori error estimate after the analysis of the numerical solution. Several application examples of planar meshes have been provided to illustrate the main features and the efficiency of the approach. The approach needs yet to be validated in the context of realistic numerical simulation examples (for instance in CFD Euler and Navier-Stokes computations).

This work can be considered as a preliminary stage toward the comprehension of tree-based decomposition methods, prior to investigate an algorithm in three dimensions.

References

[doCarmo-1976] M.P. do Carmo (1976), Differential geometry of curves and surfaces, *Prentice Hall*, New Jersey.

[Ciarlet-1978] P.G. Ciarlet (1978), *The Finite Element Method*, North Holland.

[Frey,Borouchaki-1998] P.J. Frey and H. Borouchaki (1998), Geometric evaluation of finite element surface meshes, to appear in *Finite Element in Analysis and Design*.

[Frey,Maréchal-1998] P.J. Frey et L. Maréchal (1998), Génération de maillages respectant une carte de tailles par une méthode de type 'quadtree', *RR INRIA*, to appear.

[Frey,George-1999] P.J. Frey and P.L. George (1999), *Mesh Generation and Related Topics. Application to Finite Elements*, book to appear.

[Freitag-1997] L.A. Freitag (1997), On combining Laplacian and optimization-based mesh smoothing techniques, *Joint ASME/ASCE/SES Summer Meeting McNu'97*, Northwestern Univ., USA, June 29-July 2, 37-43.

[George-1991] P.L. George (1991), *Automatic mesh generation. Applications to finite element methods*, Wiley.

[George-Borouchaki-1997] P.L. George et H. Borouchaki (1997), *Triangulation de Delaunay et maillage. Applications aux éléments finis*, Hermès, Paris. Also as *Delaunay triangulation and meshing. Applications to Finite Elements*, Hermès, Paris.

[Grice *et al.* 1988] K.R. Grice, M.S. Shephard, C.M. Graichen (1988), Automatic, topologically correct, three-dimensional mesh generation by the finite octree technique, *Technical Report*, RPI Center for Interactive Computer Graphics.

[Kela *et al.* 1986] A. Kela, R. Perucchio and H. Voelcker (1986), Toward automatic finite element analysis, *Comp. Mech. Eng.*, 57–71.

[Knuth-1975] D.E. Knuth (1975), The Art of Computer Programming, 2nd ed., *Addison-Wesley*, Reading, Mass.

[Laug *et al.* 1996] P. Laug, H. Borouchaki et P.L. George (1996), Maillage de courbes gouverné par une carte de métriques, *RR INRIA*, **2818**.

[Mitchell,Vavasis-1992] S. Mitchell and S. Vavasis (1992), Quality Mesh Generation in Higher Dimensions, *Proc. 8th ACM Symp. Comp. Geometry*, 212-221.

[Perucchio *et al.* 1989] R. Perucchio, M. Saxena and A. Kela (1989), Automatic mesh generation from solid models based on recursive spatial decompositions, *Int. j. numer. meth. eng.*, **28**, 2469–2501.

[Samet-1984] H. Samet (1984), The Quadtree and Related Hierarchical Data Structures, *ACM Computing Surveys*, **16**(2), 187-260.

[Shephard-1988] M.S. Shephard (1988), Approaches to the automatic generation and control of finite element meshes, *Applied Mechanics Reviews*, **41**, 169-185.

[Shephard *et al.* 1988] M.S. Shephard, F. Guerinoni, J.E. Flaherty, R.A. Ludwig, P.L. Baehmann (1988), Adaptive solutions of the Euler equations using finite quadtree and octree grids, *Computers & Structures*, **30**, 327-336.

[Shephard,Georges-1991] M.S. Shephard and M.K. Georges (1991), Automatic three-dimensional Mesh Generation by the Finite Octree Technique, *Int. j. numer. meth. eng.*, **32**, 709-749.

[Thacker-1980] W.C. Thacker (1980), A brief review of techniques for generating irregular computational grids, *Int. j. numer. methods eng.*, **15**, 1335-1341.

[Yerry,Shephard-1983] M.A. Yerry and M.S. Shephard (1983), A modified-quadtree approach to finite element mesh generation, *IEEE Computer Graphics Appl.*, **3**(1), 39–46.