Chapter 5

# Quadtree-octree based methods

## Introduction

Spatial decomposition methods were originally proposed as a way to represent approximations of geometric objects [Knuth-1975], [Samet-1984]. Quadtree- and octree-based mesh generation methods have been a topic of research for about two decades (see especially the surveys of the literature on spatial decomposition algorithms for mesh generation by [Thacker-1980] and [Shephard-1988]). In this context, decomposition approaches[1] have been designed to meet the needs of fully automatic mesh generation of arbitrary complex non-manifold objects and to reduce the extensive amount of time and effort required to generate meshes with semi-automatic methods [Yerry,Shephard-1983]. These approaches have proved to be robust and reliable and are commonly used in a wide range of engineering applications, see for instance, [Kela *et al.* 1986], [Perucchio *et al.* 1989], and [Shephard,Georges-1991]. In this sense, they are often compared to advancing-front (Chapter 6) or Delaunay-type methods (Chapter 7).

In this type of approach applied to mesh generation, the object to be meshed is first approximated with a union of disjoint and variably sized *cells* representing a partition of the domain. These cells are obtained from a recursive refinement of a root cell enclosing the domain (*i.e.*, a bounding box). Therefore, we obtain a covering up of a spatial region enclosing the object rather than of the object itself. In a second stage, each terminal cell is further decomposed into a set of elements whose union constitutes the final mesh of the domain (cf. Figure 5.1). The basic principle behind the method is that as the subdivision becomes finer, the geometry of the portion of the region in each cell becomes simpler, which simplifies the task of the second stage. One of the main features of the approach is its ability to proceed either directly from a given discretization of the domain boundary or, more generally, to interact with a geometric modeling system (a C.A.D. system) and generate the boundary representation of the domain as a part of the whole meshing process.

[1] so-called because they combine quadtree/octree decomposition techniques with quadrant-octant level meshing procedures.

This chapter is divided into five sections. The first section is devoted to the main concepts underlying spatial decomposition approaches and reviews the principal algorithms for constructing and searching with hierarchical structures. The second section discusses the construction of the tree representation and describes the algorithm used to subdivide the boxes into finite elements, in two and three dimensions. The third section extends the tree decomposition to the generation of meshes governed by a size distribution function. In the fourth section, techniques that combine tree decomposition and other meshing techniques are introduced. In the fifth section, extensions of the method to the generation of surface meshes and adapted meshes are briefly mentioned. Finally, a conclusion summarizes the main features of this kind of mesh generation method.

## 5.1 Overview of spatial decomposition methods

In this section, we recall some terminology and the basic definitions of the spatial decomposition structures in two and three dimensions, within the framework of mesh generation. PR-quadtrees will be used here as meshing structures (see Chapter 2 for an introduction to this structure).

In order to fix the terminology, let us consider a meshing problem for which the domain to be meshed is an arbitrary non-convex domain $\Omega$ in $\mathbb{R}^2$ ($\mathbb{R}^3$) represented by a boundary representation. The latter is in fact a polygonal (polyhedral) contour $\Gamma(\Omega)$ composed of a set of vertices $\mathcal{V}$, a list of edges $\mathcal{E}$ and (in three dimensions) a list of faces $\mathcal{F}$.

### 5.1.1 Terminology and definitions

The basic concept of any spatial decomposition consists first of enclosing an arbitrarily shaped domain $\Omega$ in a bounding box, denoted $\mathcal{B}(\Omega)$, (a square or a rectangle in two dimensions and a cube or a parallelepiped in three dimensions). This box is then subdivided into four (resp. eight) equally-sized *cells* (the *size* of a cell $c$ is the length of the longest side of $c$), each of which may possibly be recursively refined several times. The *stopping criterion* used to subdivide a cell can be based on the local geometric properties of the domain (e.g., the local curvature of the boundary) or user-defined (the maximum level of refinement, for instance).

**Remark 5.1** *The* stopping criterion *used for finite element mesh generation is usually different from the standard geometric criterion used in classical space partitioning procedures.*

The four (resp. eight) vertices at the corners of a cell are called the *corners*. The edges connecting two 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. By definition,



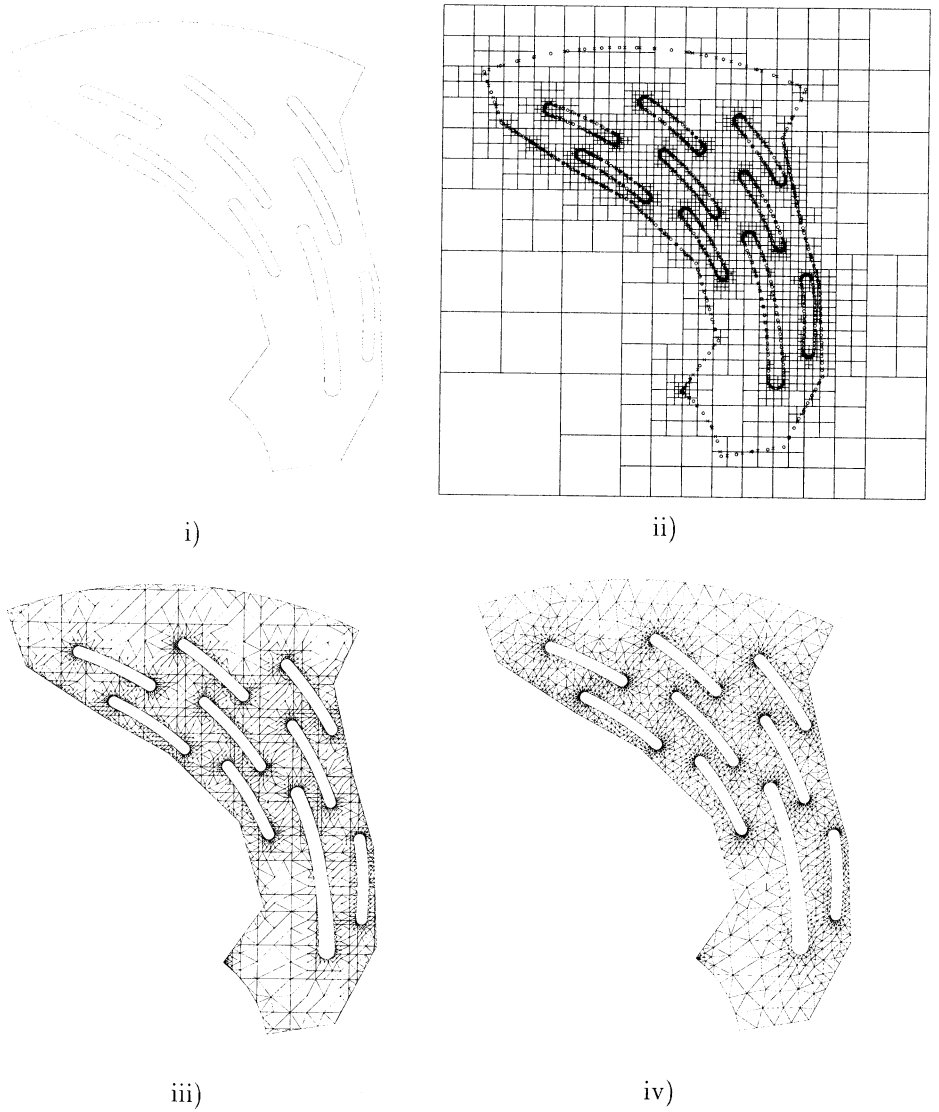



Figure 5.1: *A two-dimensional domain $\Omega$ i), spatial decomposition of $\mathcal{B}(\Omega)$, the bounding box of $\Omega$ ii), the resulting mesh iii) and the final mesh of $\Omega$ iv).*

two cells are said to be *adjacent* if they share an edge (cf. Figure 5.2, right-hand side). The set of cells composes the *tree* (the tree structure) associated with the spatial decomposition. The subdivision of a cell consists of adding four cells to the node of the tree for that cell. The bounding box $\mathcal{B}(\Omega)$ is the *root* of the spatial decomposition tree.

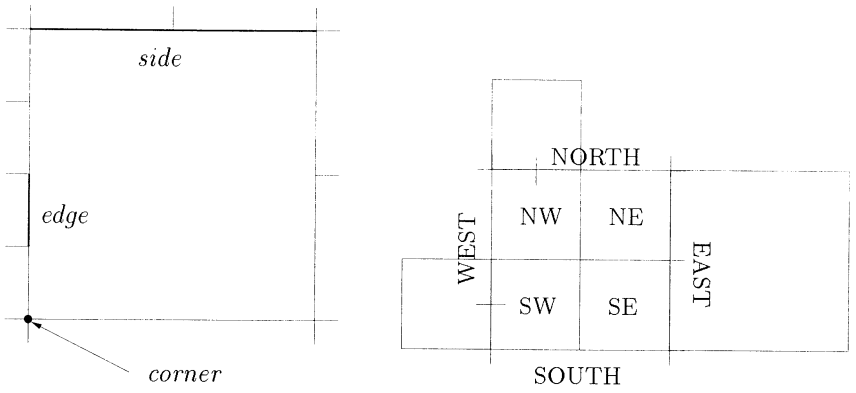



Figure 5.2: *Terminology : side, edge and corner of a cell (left-hand side). Adjacent cells sharing a common edge, canonical notations (right-hand side).*

The *level* of a cell corresponds to its depth in the related tree (*i.e.*, the number of subdivisions required to reach a cell of this size). The bounding box is at level 0. The *depth* of the tree corresponds the maximum level of subdivision. In Figure 5.3, the depth of the tree is 4. A cell that is not subdivided further is called a *terminal* cell or a *leaf*. Each non-terminal cell is called an *internal* cell.

In the classical spatial decomposition scheme, the current cell is subdivided into four (resp. eight) cells, the quadrants (resp. octants) and the set of points $\mathcal{V}$ is partitioned accordingly into several subsets. The insertion of a point into the tree consists of identifying the cell containing it. If this cell is empty, the point is inserted. Otherwise, the cell is refined and the process is iterated on each of the sub-cells. The recursive subdivision stops when the set of points associated with a cell is reduced to a single point or is the empty set. Initially, the choice of the bounding box is arbitrary and is usually a square (resp. cube) containing all of $\mathcal{V}$. This box is determined by computing the extrema of the coordinates of the points in $x$, $y$ (and $z$) directions (*Note* : the corresponding algorithm is linear in time : $O(np)$, $np = Card(\mathcal{V})$.

**Remark 5.2** *At each stage, a cell can be subdivided into sub-cells. This does not automatically imply that the set of points $\mathcal{V}$ is split accordingly : all points can belong to the same sub-cell. In this case, the resulting decomposition tree will be quite unbalanced (the number of subdivision levels not being constant in each cell).*

From the previous remark one can see that the size (*i.e.*, the number of cells) and the depth of the tree are not determined only by the number of points in $\mathcal{V}$. The depth of the tree is related to the smallest distance $d$ between two points of



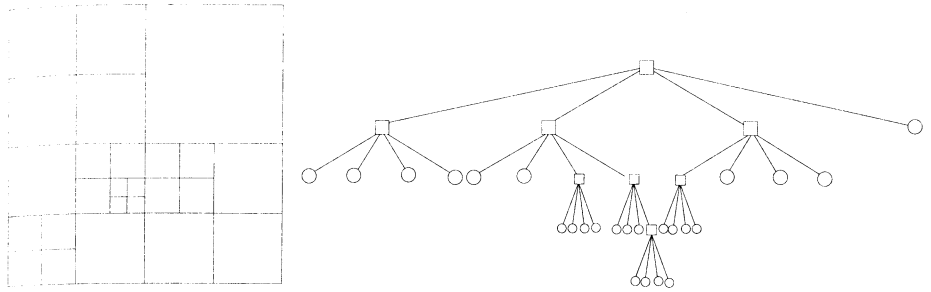


Figure 5.3: *Quadtree decomposition and corresponding data structure (here the tree depth is $p = 4$).*

$\mathcal{V}$ as well as to the length of a side of the bounding box, as stated in the following lemma, in two dimensions :

**Lemma 5.1** *The depth $p$ of the quadtree obtained by inserting all points of $\mathcal{V}$ is bounded by $p < \log(b/d) + 1/2$, with $d$ the smallest distance between two points of $\mathcal{V}$ and $b$ the length of a side of $\mathcal{B}(\Omega)$.*

**Proof.** In a quadrant (a square for the sake of simplicity), the largest distance between two corners (*i.e.*, the diagonal) is $l\sqrt{2}$), with $l$ the length of a cell side. At each level of refinement, the size $l$ of a cell is divided by two. At level $i$, the size $l$ is then equal to $b/2^i$, with $b$ the size of the bounding box and thus the length of the diagonal is equal to $\sqrt{2}b/2^i$. The smallest distance between two points of $\mathcal{V}$ being $d$, we must have $\sqrt{2}/2^i \geq d$, which relates the level of a non-terminal cell (*i.e.*, containing at least two points) to the distance $d$. The maximum level sought (the depth of the tree) is then : $p \leq 1 + log(\sqrt{2}b/d)$. □

**Exercise 5.1** *Find the corresponding upper bound in three dimensions.*

The previous remark about tree balancing leads to the following rule, which can be justified by the need to control somehow the mesh gradation in finite elements methods (*i.e.*, the size variation between neighboring elements).

**Definition 5.1 [2:1] rule.** *A tree subdivision is* balanced *if every side of a terminal cell contains at most one corner (cf. Figure 5.4).*

This definition is equivalent to writing that the sizes of any two adjacent cells differ by at most a factor two.

Moreover, a key aspect of spatial decomposition methods is their ability to classify the cells with respect to the domain $\Omega$. The minimum classification requires a decision about whether a cell is fully inside the domain, fully outside the domain or contains portions of the domain boundary $\Gamma(\Omega)$ (e.g., the cell contains a point of $\mathcal{V}$ or is intersected by an edge of $\mathcal{E}$ or a face of $\mathcal{F}$, the list of faces). These cells are respectively denoted as $\mathcal{I}$ (inside), $\mathcal{O}$ (outside) and $\mathcal{B}$ (boundary).

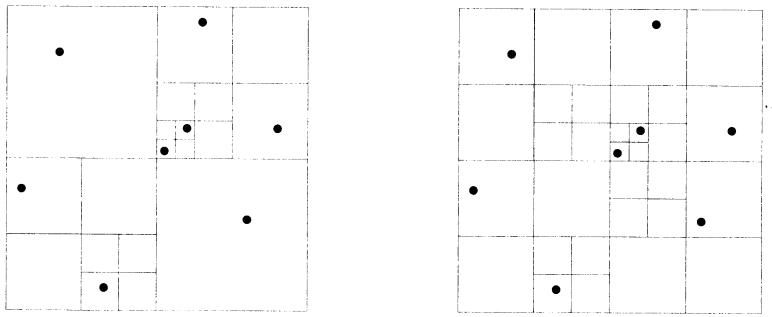


Figure 5.4: *Balanced tree according to [2:1] rule. Initial quadtree (left) and balanced quadtree (right).*

### 5.1.2 Operations on trees

Two kinds of operations are commonly performed on trees :

- topological operations : creating four (resp. eight) cells from a given cell, finding the cells adjacent to the current cell in a given direction,
- geometric operations : finding the cell containing a given point, calculating the intersections between an edge of $\mathcal{E}$ and the sides of the current cell.

These operations do not all have the same frequency nor the same computational cost. The *neighbor finding* operation concerns seeking the cells adjacent to a given cell : let $c$ be a cell and consider a direction, the problem is to find a cell $c'$ adjacent to $c$ in the given direction. Usually, $c$ is a leaf and the problem is one of finding the adjacent cell that is of level less than or equal to that of $c$ (*i.e.*, the size of $c'$ must be greater or equal to the size of $c$). Basically, the algorithm checks the current cell and compares the direction with the relative position of the cell in its *parent cell* $c'$ (the non-terminal cell containing $c$). Notice that in two cases (*i.e.*, directions), the adjacent cell belongs to the same parent of the current cell. In the other cases, the algorithm analyzes the parent of $c$ to find a neighbor $c'$ in the given direction. If $c'$ is a leaf, $c'$ is the neighbor of $c$, otherwise the neighbor of $c$ is any of the *children* $q_i$ corresponding to the subdivision of $c'$.

In two dimensions, the neighbors are identified by the four cardinal directions, $\{NORTH, SOUTH, EAST, WEST\}$ and the quadrants obtained during the refinement of a cell are identified by their relative position within the parent cell : $\{NW, NE, SW, SE\}$ (cf. Figure 5.2). These quadrants are so-called *siblings* to indicate that they result from the subdivision of the same cell. The notation $SE - child(parent(c))$ denotes the $SOUTH - EAST$ quadrant (non necessarily a leaf) of a cell $c$. Algorithm 5.1 searches for the $EAST$-neighbor (not necessarily a leaf) of a cell $c$ in a quadtree $\mathcal{Q}$. The following lemma specifies the complexity of this algorithm.

**Lemma 5.2** *Let $\mathcal{Q}$ be a quadtree of depth $p$. The searching algorithm used to identify the neighbor of a cell $c \in \mathcal{Q}$ in a given direction is of complexity $\mathcal{O}(p+1)$.*



**Proof.** First, notice that the tree is not assumed to be well balanced (the size difference between adjacent cells is not bounded). For each recursive call, the local complexity is in $\mathcal{O}(1)$ and the depth of the sub-tree traversed is decreased by 1. Thus, the complexity of the algorithm is linearly related to the depth of the tree. □

**Algorithm 5.1** *Search for the $EAST$ neighbor of depth not greater than that of a given cell $c$ in the tree $\mathcal{Q}$.*

```
Procedure EastNeighbor(Q,c)
IF c = B(Ω) (look for the root of Q)
   RETURN B(Ω)
IF c=NW-child(parent(c)) THEN (c has a sibling on this side)
   RETURN NE-child(parent(c))
IF c=SW-child(parent(c)) THEN
   RETURN SE-child(parent(c))
c' = EastNeighbor(Q,parent(c)) (recursive call)
IF IsLeaf(c') THEN
   RETURN c'
ELSE
   IF c=NE-child(parent(c)) THEN
      RETURN NW-child(parent(c'))
   ELSE
      RETURN SW-child(parent(c'))
   END IF
END IF.
```

**Exercise 5.2** *Update the Algorithm (5.1) to always return a terminal cell. Write a more general algorithm to return a neighbor in any direction.*

This algorithm is only one of the components of the *a posteriori* tree balancing algorithm. The quadtree is balanced in a post-processing step that immediately follows the construction stage. Each and any leaf must conform to the [2:1] rule introduced previously. Hence, the computational cost of the balancing stage is related to the number $m$ of leaves in the tree. According to [deBerg *et al.* 1997], the balancing operation is of complexity $O(m(p+1))$.

**Exercise 5.3** *It is possible to balance a quadtree during the construction stage, without first constructing the unbalanced version. Describe such an algorithm and analyze its complexity.*

From the geometric point of view, the searching operation to find a cell enclosing a given point of $\mathcal{V}$ is an operation based on a comparison of the coordinates of the corners of a cell at a given level. Depending on the result of the comparison, a cell containing this point is identified and becomes the new current cell. The operation is then repeated recursively until a leaf is found that contains the point. Similarly, the intersections of an edge of $\mathcal{E}$ with a tree $\mathcal{Q}$ can be found.

### 5.1.3 Data structures

From the algorithmic point of view, D. Knuth identified in his book three possible approaches to representing trees [Knuth-1975] :

- A tree structure using pointers. In this *natural* approach, each non terminal cell requires four (resp. eight) pointers, one for each of the subtrees and some information to indicate the cell classification. In addition, extra pointers can be added to improve the efficiency, such as links to the parent cell, links to neighboring cells, etc.

- A list of the nodes encountered by a traversal of the structure. Within 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 visiting each node of the initial tree to locate the root of the subtree.

- A system of locational codes, for instance with a *linear quadtree* (see below).

For example, Figure 5.3 (right-hand side) shows a tree representation associated with the quadtree decomposition depicted in the same figure (left-hand side). In practice, the context in which the method is applied dictates which of the three possible representations should be used. For instance, the need to quickly identify neighbors leads to favoring a linear structure or a pointer-based structure containing pointers to the parent cell and siblings.

**An example : the linear quadtree.**

We show here the linear quadtree implementation in two dimensions. The quadtree decomposition approach can be implemented as a hierarchical grid (Chapter 2), the number of boxes in any direction then being a power of two. A linear quadtree avoids the allocation of explicit pointers to maintain the spatial order of quadrants, a single array index being used to access the data structure. The idea behind the representation is rather simple. Each quadrant is represented by a number using a systematic scheme. For instance, label 1 corresponds to the bounding box, labels 2-5 represents the four quadrants of the bounding box. The labels are used as array indices to access the cell information. Each possible subdivision of a quadrant leads to four consecutive indices referring the four sub-quadrants (cf. Figure 5.5).

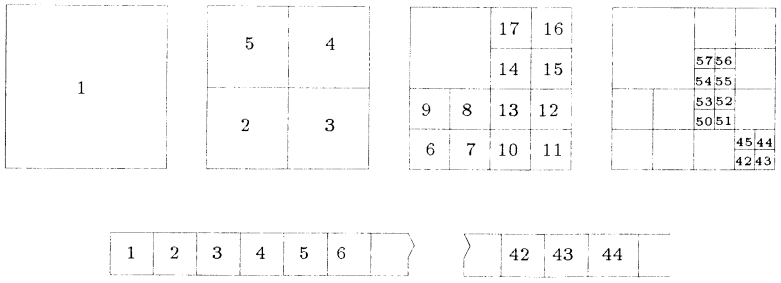



Figure 5.5: *Linear quadtree encoding : spatial addressing structure and numbering.*



Hence, it is pretty easy to calculate the index of any cell in the tree from its size and location. For instance, the four children of a cell $c$ are given by $4c-2+j$, $j=0,3$ and its parent cell is trivially obtained as : $E((c+2)/4)$ where $E(i)$ denotes the integer part of $i$. Conversely, from this index, the size and position of any quadrant are easy to retrieve.

**Exercise 5.4** *Indicate how to determine the neighbor indices of a given cell $c$. Write the algorithm to find the index of a cell enclosing a given point $P \in \mathcal{V}$.*

If the lowest level grid requires $n^2$ cells, the maximum depth $p$ of the linear quadtree is given by : $p = \log_2 n$. Although the address of a box can be easily computed, this representation carries a storage penalty. Indeed, the total amount of memory $m$ required to store the whole tree (in the worst case scenario) corresponds to the sum of a geometric series of ratio $2^2$, *i.e.*,

$$m = \frac{2^{2p}-1}{2^2-1}\,.$$

As compared with $n^2$ boxes required for its lowest level (corresponding to a regular grid of size $n$), 1/3 more memory in two dimensions is required for a hierarchical linear tree.

**Exercise 5.5** *Establish the previous formula and justify the extra storage penalty carried out by the structure as compared with a classical tree structure using pointers. How more storage memory is required in three dimensions to store a hierarchical linear tree as compared with a regular grid of same resolution ? (Hint : see [Gargantini-1982] or [Ibaroudene,Acharya-1991] for a possible solution).*

**Remark 5.3** *An alternative implementation approach consists of using a binary encoding : the locational index of each quadrant is composed of a sequence of digits $c_i$ in base 4, each $c_i$ being obtained by concatenation of the $x_i$'s and $y_i$'s of the binary representation of the box coordinates $x, y$.*

## 5.2 Classical tree-based mesh generation

The use of a quadtree decomposition for meshing purposes was pioneered about fifteen years ago, in particular by [Yerry,Shephard-1983]. Several variants have been proposed (see for instance, [Kela *et al.* 1986] or [Perucchio *et al.* 1989]). Currently, such a meshing technique is usually based on three successive steps dealing with :

Stage 1 : the parameterization of the mesh (specification of the element size distribution function, discretization of the boundary, etc.),

Stage 2 : construction of a spatial covering up from a bounding box of the domain,

Stage 3 : internal point and element creation.

This general scheme is somewhat different from that used in advancing-front or Delaunay type methods (Chapters 6 and 7), in which the boundary discretization is a necessary input of the problem. In fact, in this approach, the construction

of the covering up can be either based on a given boundary discretization[2] or performed using geometric queries to a geometric modeling system. The covering up is such that its cells

- have a size distribution compatible with the desired mesh gradation,
- maintain an efficient hierarchical data structure and
- store all the information required by the meshing step to generate a valid mesh of the geometric model (Stage 3 of the previous scheme).

In a spatial decomposition method, the mesh vertices are created by the element creation procedure. But unlike other mesh generation techniques (advancing-front or Delaunay type, for example), the internal vertices are usually the vertices of the decomposition (*i.e.*, the corners of the cells). The boundary vertices are the initial points of $\mathcal{V}$ and some additional points, created during the tree construction and corresponding to the intersection of the boundary with the tree. Notice that therefore the initial boundary discretization is not usually preserved in this approach.

### 5.2.1 General scheme

Let us consider here a boundary discretization represented by a polygonal (polyhedral) contour described as a list of points, a list of edges and possibly a list of faces (in three dimensions). For the sake of simplicity, the boundaries of the domain are considered as straight segments (curved boundaries are discussed in Section 5.4).

Formally speaking, the spatial decomposition technique is an iterative procedure that builds the covering tree of the domain from its description before meshing each terminal cell. At each stage of the tree construction, each leaf is analyzed and refined into $2^d$ (with $d$ the space dimension) equally sized cells, based on a specific criterion. Schematically, a spatial decomposition method can be described as follows :

Stage 1 : preliminary definitions :

- construction of a size distribution function defined by interpolation (governed mesh generation) or implicitly[3],
- boundary discretization (general case),
- creation of a bounding box of the domain,

Stage 2 : initialization of the tree by the bounding box,

Stage 3 : tree construction (insertion of each and any entities of the boundary discretization) :

[2] in which case, the points and the elements created at Stage 3 are internal to the domain.
[3] see Section 5.3.1



- (a) selection of a boundary entity, in ascending order (points, edges and faces),
- (b) identification of the cell in the current tree that contains this entity,
- (c) analysis of the cell, if it already contains an entity of the same dimension then refine the cell, otherwise back to (b),
- (d) insertion of the entity in the cell and back to (a)

Stage 4 : tree balancing, the level difference between any pair of adjacent cells is limited to one.

Stage 5 : point filtering, in each terminal cell intersected by the boundary,

Stage 6 : creation of the internal points and the mesh elements :

- predefined patterns (the *templates*) are used to mesh internal cells,
- boundary leaves are meshed using an automatic technique (decomposition into simpler domains, ...),
- removal of the external element, using a coloring algorithm,
- intersection points are then adjusted on the true geometry,

Stage 7 : mesh optimization.

Despite conceptual differences among the various approaches published, several aspects of any tree-based approach used for meshing purposes are consistent :

- the data structure is used for localization and searching purposes,
- the mesh generation has two stages, first the tree is generated, then the mesh is created based on cell classification within the tree,
- the cell classification drives the mesh generation stage (although boundary cells may need special care),
- the cell corners are generally used as mesh vertices,
- the mesh gradation is controlled by the level of refinement of the cells (for instance, according to the [2:1] rule described above).

In the following sections, we will describe the classical tree-based meshing approach as described in the previous scheme, and indicate the main difficulties that may arise.

**Main difficulties.** The general scheme above hides several potential difficulties. Most of these problems only occur in three dimensions, especially regarding the robustness of the method. In fact, the construction of the tree requires enforcing a specific criterion, the latter possibly being incompatible with the geometric requirements. For instance, we could specify a maximum level of refinement, thus leading to problems when trying to discriminate closely spaced entities (when the distance is much smaller than the specified cell size).

Expected difficulties can easily be identified and are usually related to :

- the knowledge of the local neighborhood of a (boundary) entity during its insertion in the current tree,
- the intersection computations during the insertion of the entity,
- the filtering of closely spaced points, which requires a robust algorithm in order to preserve the topology and the geometry of the domain.

Notice that in this approach, the points are not optimal (in the sense that they do not necessarily lead to the creation of optimal elements, cf. Chapter 1), but result from intersection calculations (boundary/cells) or correspond to the corners of the cells. Therefore, the resulting meshes must be optimized (cf. Chapter 18).

**Remark 5.4** *These difficulties are essentially twofold, numerical (intersection calculations) and algorithmical (neighborhood, ...).*

### 5.2.2 Preliminary requirements

Unlike advancing-front method (cf. Chapter 6), no specific assumption is made on the nature of the input data for boundary discretization. Obviously, the data structures are important in this approach, the decomposition tree acting as a neighborhood space as well as a control space (cf. Chapter 1).

**Boundary mesh.** Similar to Delaunay-type methods, the boundary mesh is not necessarily oriented. However, for efficiency purposes during the insertion of the boundary entities, it may be useful to enforce an orientation of the discretization, so as to insert sequentially neighboring entities during the tree construction stage, thus reducing tree traversals as much as possible (in searching operations).

**Data structures.** As searching and insertion operations are usually local features, it is important to use suitable data structures in the process. So, the tree structure (*quadtree* in two dimensions and *octree* in three dimensions) should contain in each terminal cell all the information required for a further analysis of the cell. Therefore, the boundary items are stored in the terminal cells of the tree. Two adjacent cells can share some common information (an intersection point, for instance), so it is desirable to refer to this information in the cell rather than duplicating it (to avoid numerical errors). Further updates of the tree (for example, a cell refinement operation) will then propagate the information to the terminal cells. The tree construction requires only refinement operations, no cell deletion is involved.

### 5.2.3 Tree construction

The tree representation of an arbitrary domain $\Omega$ is defined from the bounding box $\mathcal{B}(\Omega)$ that fully encloses the set of points of $\mathcal{V}$ of the boundary discretization $\Gamma(\Omega)$. The tree is initialized by $\mathcal{B}(\Omega)$. The items of $\Gamma(\Omega)$ are iteratively inserted into the cells of the current tree, so that each terminal cell contains at most one item of $\Gamma(\Omega)$. Any cell that contains more than one such item is refined into $2^d$ identical sub-cells and the insertion procedure is resumed for each cell of the sub-tree.



The insertion of the boundary items is a tedious process that can be implemented in many ways. The common method consists of inserting the items according to the ascending order of their dimensions (points, edges and faces). Moreover, if the boundary has been oriented, it may be interesting to insert adjacent items, so as to benefit from the local aspect of the operations and thus to limit somehow the tree traversals. Generally speaking, it is desirable to avoid :

- the loss of previously computed or known information (for example, numerical values related to the intersection points),
- the degradation of the computational cost of tree operations (cache default, cf. Chapter 2).

**Point insertion.** The localization of the cell containing a boundary point or intersected by a boundary item requires special care (especially in three dimensions). Actually, a point can be located inside a cell, along a cell side (and then it belongs to two adjacent cells) or even on a corner (it is then associated with all the cells sharing this common corner). One can easily imagine that numerical problems can arise when trying to correctly identify the different configurations.

**Remark 5.5** *A commonly used technique consists of slightly modifying the position of the point so that it belongs to only one cell. Its initial position will be restored after the mesh generation.*

When the cell containing a point has been identified, two situations can arise. First, if this cell is empty (*i.e.*, does not include any boundary item), the process consists of associating the related information with the cell. Otherwise, the cell is refined into $2^d$ identical cells and the analysis is propagated to each of the sub-cells.

In practice, the processing is even more complex if the item is an edge or a face as several sub-cells can be intersected.

**Peculiar points.** From a computational point of view, two kinds of points require specific attention, the *corners* and the *non-manifold* points. Corners are boundary points at which incident items form a small (dihedral) angle (cf. Figure 5.6, left-hand side). In two dimensions, non-manifold points are boundary points with more than two incident edges (cf. Figure 5.6, right-hand side). In three dimensions, this concept is more tedious to handle as non-manifold edges (shared by more than two faces) have to be defined first, their endpoints then being non-manifold points.

The test allowing to decide whether a terminal cell must be further refined is called the *stopping criterion*. Several such criteria have been suggested in the context of mesh generation.

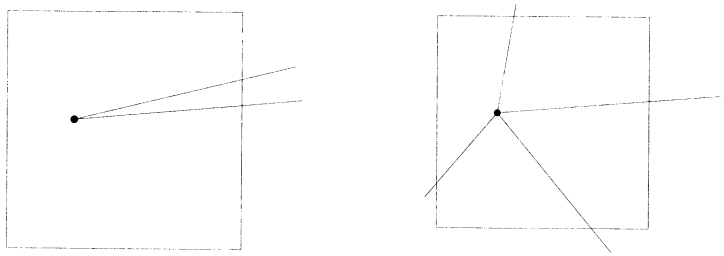


Figure 5.6: *Identification of corners (left-hand side) and non-manifold points (right-hand side) in two dimensions.*

**Stopping criteria.** Usually, the stopping criterion must take into account the different types of items to be inserted. The most commonly used criterion is such that any leaf of the tree contains at most one connected component of $\Gamma(\Omega)$. If a leaf contains no point of $\mathcal{V}$, then it should contain at most one edge or portion of an edge (resp. face) of $\Gamma(\Omega)$ (cf. Figure 5.7, right-hand side).

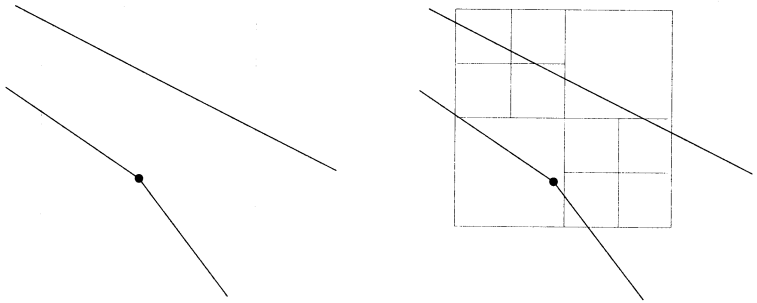


Figure 5.7: *Tree refinements based on two different stopping criteria in two dimensions : any leaf contains at most one vertex of $\Gamma(\Omega)$ (left-hand side), each leaf contains at most one point (shared by two edges) or a small part of an edge of the discretization (right-hand side).*

For arbitrary domains, the stopping criterion is of importance and affects the depth of the resulting tree and thus the complexity of the whole algorithm. Figure 5.7 illustrates the impact of the stopping criterion on the decomposition. On the left, the quadtree is intersected by three segments and contains one point of $\Gamma(\Omega)$. The corresponding criterion consists of stopping the decomposition when each cell contains no more than one point. On the right side, the stopping criterion is such that each cell must be intersected by no more than one edge (or part of an edge) if it does not include a vertex of $\mathcal{V}$.

**Remark 5.6** *In two dimensions, the most strict stopping criterion consists of forcing each cell side to include no more than one intersection point, unless the cell contains a corner or a non-manifold point [Frey,Maréchal-1998].*



**Remark 5.7** *In three dimensions, the classical criteria assign control parameters to the boundary items (for instance, related to the local curvature of the geometric features, see Section 5.5.1).*

The construction stage attempts to separate two opposite sides of the domain based on a distance criterion. In other words, if the domain is such that locally two sides are very close to each other, with really different discretization sizes, the tree decomposition will automatically reduce this shock between the sizes, by inserting additional points and refining the boundary items. This is in fact equivalent to refining the tree until the cell sizes become compatible with the local distance between the opposite sides of the domain. The resulting mesh gradation is thus controlled in an implicit fashion. Moreover, this control is also increased by the balancing stage (see above).

**Remark 5.8** *As the tree construction stage may introduce additional points, the initial boundary discretization is no longer preserved (unlike the advancing-front or Delaunay-based mesh generation methods, for which the boundary discretization is a constraint that has to be strictly preserved).*

**Remark 5.9** *The specification of additional (non boundary) points in the input dataset does not lead to substantial modification of the classical algorithm. These points will be inserted in the tree just after the insertion of the vertices of the boundary discretization.*

Figure 5.8 depicts the different steps of the tree construction of the decomposition of a planar domain in two dimensions. The domain is illustrated on the left-hand side of the figure. The tree (Figure 5.8, middle) corresponds to the decomposition obtained after the insertion of all boundary points. At this stage of the process, no additional point has been created. Finally, the insertion of boundary edges leads to the domain decomposition shown in Figure 5.8 (right-hand side). Several intersection points have been created and the tree has been locally refined to accommodate the stopping criterion.

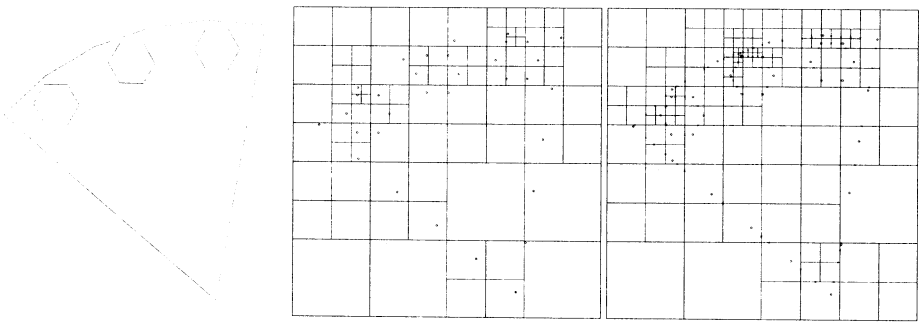


Figure 5.8: *Quadtree decomposition of a two-dimensional domain. Initial discretization (left-hand side), resulting quadtree after the insertion of the set of points $\mathcal{V}$ (middle) and after the insertion of the edges of $\mathcal{E}$ (right-hand side).*

**Exercise 5.6** *[deBerg et al. 1997] Let us consider a two-dimensional domain $\Omega$ such that the points of $\mathcal{V}$ have integer coordinates and that each boundary edge makes an angle of $0°, 45°, 90°$ or $135°$ with the x-axis. The stopping criterion could be such that a quadrant is not subdivided if the intersection between an edge of $\mathcal{E}$ and its sides is empty or if the minimal size is reached. Show that the interior of the quadrants can be intersected by the polygonal contour only in a single way. Write the resulting construction algorithm for this specific stopping criterion.*

Notice that a variant of the construction procedure described above has been proposed by [Yerry,Shephard-1983]. In this approach, a limited number of intersections (*i.e.*, of segments coming from boundary edges intersections) is tolerated in boundary quadrants. To preserve the basic structure, intersection points are chosen as quarter, half or endpoints of the quadrants sides. This technique does not improve the geometric accuracy of the representation although it improves the numerical approximation and the consistency of the algorithm. It allows the use of integer coordinates to represent the domain boundary discretization.

The result of the tree decomposition is a covering up of the bounding box of the domain in which adjacent cells may have dramatically different sizes. In other words, the quadtree may be quite unbalanced. This feature can become a major drawback during the element mesh generation stage (as the mesh gradation is not explicitly controlled). To overcome this problem, a balancing stage is applied.

### 5.2.4 Tree balancing

As mentioned previously, the tree construction rules concerned the boundary items of the given discretization. Hence, the tree resulting from the general scheme can be rather unbalanced (according to Definition 5.1). This feature, related to the difference between the levels of each pair of adjacent cells, can be inconvenient in many ways. First, the size variation between neighboring cells tends to increase the cost of tree traversals (in searching operations), the number of levels explored varying from one cell to the next one. Then, as spatial covering up is used to generate a mesh, the mesh edges will have a length close to the cell size that is used to generate them. In fact, when the size distribution function is implicitly defined (no user-specified function is explicitly supplied), the cell size reflects the average size of the elements generated in this cell. Finally, as a cell can be surrounded by cells of arbitrary dimensions, the number of possible combinations is not limited, thus preventing the use of an automatic procedure (based on predefined patterns) to mesh the cells.

Therefore, in order to control mesh gradation and also to simplify the mesh generation stage, another rule is introduced so as to limit the difference between the levels of two adjacent cells to a factor 1 at most. This rule is the so-called *[2:1] rule*.

This procedure affects the spatial decomposition of the domain obtained using the former construction rules. In particular, it leads to refining several cells and thus propagates the refinement recursively throughout the tree, to the neighboring cells. The final decomposition usually contains (experimental result) up to 10% more leaves than the initial one.



**Implementation of the [2:1] rule.** From a practical point of view, the [2:1] rule can be added to the construction rules. This is equivalent to saying that it is not necessary to initially construct an unbalanced tree, on which the balancing rule is applied *a posteriori*. In fact, if the current tree is already balanced, it is sufficient to check whether the balancing rule is enforced in each leaf that is subdivided during the construction stage. Otherwise, the levels of the adjacent cells are adjusted (refined) until the balance has been achieved. In practice, one can observe that the propagation of the refinement is usually contained in two cells adjacent to the current cell. This feature results in a local balancing procedure, which is easy to implement and computationally inexpensive as only limited tree traversals and updates of the tree structure are involved.

**Remark 5.10** *Notice that similar to the construction stage, the corners and non-manifold points are not concerned with the balancing stage (*i.e.*, the cell containing such a point is not refined). Figure 5.9 illustrates such a situation where the tree is locally unbalanced because of a non-manifold point.*

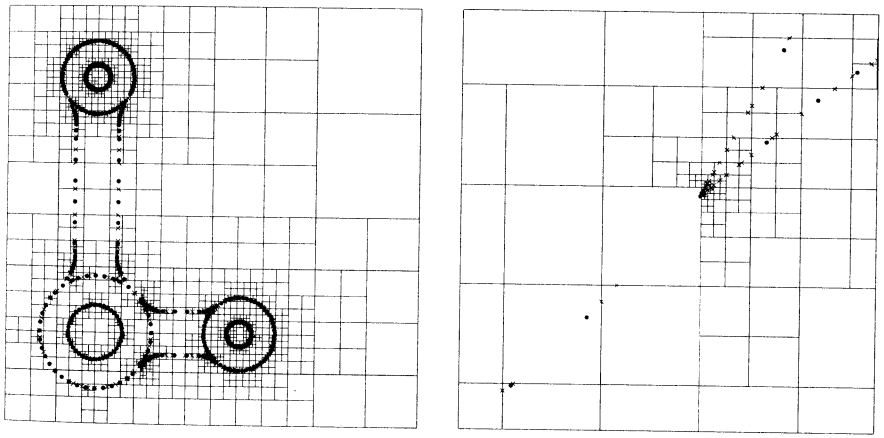


Figure 5.9: *Example of an unbalanced tree in the vicinity of a non-manifold point, in two dimensions. Spatial decomposition of the whole domain (left-hand side) and close up in the vicinity of the cell containing the non-manifold point corresponding to the intersection of three boundary edges (right-hand side).*

**Remark 5.11** *It is however possible to balance a given tree* a posteriori. *To do so, a procedure based on a* breadth-first *tree traversal is applied (starting from the root of the tree).*

The tree construction procedure can potentially introduce some points that are not part of the initial dataset. For instance, the points resulting from the intersection between the cell and the boundary discretization. These intersection points can be very close to the cell sides, thus leading to the creation of poorly-shaped elements. To overcome this problem, a filtering procedure is carried out on the boundary points.

### 5.2.5 Filtering of the intersection points

Figure 5.10 (left-hand side) illustrates one of most common cases of intersection points creation during the insertion of a boundary edge into the current tree, in two dimensions. The two intersection points that belong to the upper-left quadrant are closely spaced to each other as well as close to the quadrant corner. This usually leads to the generation of a triangle which has these three points as vertices and whose size does not match the local size specified by the distribution function.

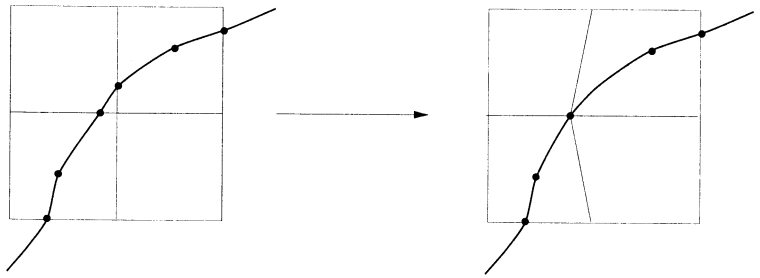


Figure 5.10: *Insertion of the boundary edges in the tree. Left, two intersection points are close to the quadrant corner. Right, the points have been merged together into a single one, the quadrant corner then being moved toward the resulting point.*

As the construction rules do not explicitly take into account inter-point distances (the discretization of the domain boundary is the input data), a post-processing stage applied on the resulting tree appears to be the only solution to this problem.

The intersection points filtering procedure consists in removing some vertices in the tree based on a distance criterion. This tedious operation is not easy to implement as it affects the geometry of the spatial decomposition. The local geometric modifications carried out mainly consist of :

- associating a cell corner with the closest (boundary or intersection) point belonging to the cell side,
- merging two intersection points into one,
- associating an intersection point and the closest boundary point within the same cell.

All these checks can be carried out provided the topology of the geometric model and that of its decomposition are not altered. Thus, two intersection points can be merged together if and only if they are classified onto the same geometric boundary item. Moreover, the geometry of the initial boundary discretization should not be modified. For instance, the deletion of a boundary point (a vertex of $\mathcal{V}$) is not allowed.

At completion of the procedure, there should be no point closer to another one than a given tolerance value[4]. Figure 5.10 (right-hand side) depicts the merging

[4] this tolerance can vary according to the type of entity to be snapped.



between two intersection points followed by the relocation of the quadrant corner onto the remaining intersection point.

**Remark 5.12** *As the filtering procedure may change the coordinates of a quadrant corner, this cell may lose its axis-alignment property. This can result in additional difficulties for the searching procedure (e.g. to identify which quadrant contains a given vertex).*

Figure 5.11 shows the result of the filtering stage on a two-dimensional computational domain. One can clearly observe (right-hand side) the quadrant deformation due to the relocation of the tree vertices. Although many extra intersection points have been discarded, the initial boundary discretization is not preserved (*i.e.*, some of the intersection points still remain).

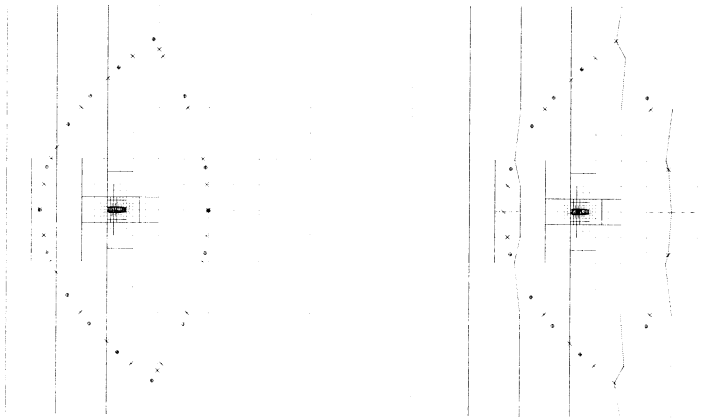


Figure 5.11: *Initial decomposition of a two-dimensional computational domain (left-hand side) and resulting decomposition after the filtering of the intersection points (right-hand side). Notice the deformation of the spatial decomposition structure.*

**Remark 5.13** *As the topological consistency of the tree representation is rather difficult to preserve (especially in three dimensions), in some of the proposed approaches points are not filtered, [Frey,Maréchal-1998]. In this case, the resulting mesh elements that may have an unacceptable size or quality are further processed, during the optimization stage (cf. Section 5.2.7).*

The filtering stage marks the end of the processing on the tree structure. At completion of this stage, the tree structure remains unchanged and the following steps are exclusively based on tree traversals, especially the element creation stage.

### 5.2.6 Vertex and element creation

At this stage of the procedure, we have to go from a spatial decomposition structure to a mesh of the domain. To this end, the basic idea of this approach is to mesh the

leaves of the tree independently of one another. The final mesh will be the union of all the elements created at the cell level, the global conformity being ensured by the conformity of the cell side discretization.

**Vertex creation and insertion.** The creation of the mesh vertices consists in inserting sequentially each cell corners and each intersection point in the mesh structure. From a practical point of view, it may be noticed that the number of mesh vertices is known beforehand and corresponds to the sum of the number of boundary vertices, the number of the cell corners and the number of intersections points (if any). This allows the data structures to be allocated correctly at the beginning of the program.

This step is based on a tree traversal to find all terminal cells. Here, we meet again the need to have an efficient and flexible data structure.

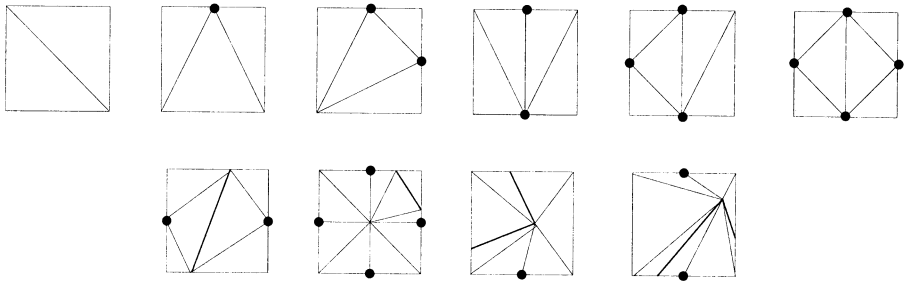


Figure 5.12: *A set of six plausible patterns to triangulate the internal quadrants (the other patterns can be retrieved using the rotational symmetry properties) and four patterns used to mesh boundary quadrants (in the last two patterns, the intersection point sees the other quadrant points).*

**Mesh element generation.** The creation of the mesh topology (*i.e.*, the mesh elements) is a more tedious operation. First notice that the balancing rule led to a substantial reduction in the number of possible configurations for a given cell, as compared with its neighbors. Therefore, knowing of the sizes of the adjacent cells is sufficient to be able to mesh a terminal cell. One has however to distinguish the case of boundary cells from that of internal ones.

The case of an internal cell is indeed very simple as it involves a set of predefined patterns (the so-called *templates*) used to get the list (the connectivity) directly of the elements that form a conforming covering up of the cell. Global mesh conformity is automatically ensured, as seen above.

In two dimensions, one has to consider $2^4 = 16$ possible configurations for each internal cell. From the computational point of view, this number can be reduced to only six templates using the various symmetry properties. Figure 5.12 depicts a set of templates used to mesh internal cells.

In three dimensions, one has to consider $2^{12} = 4,096$ possible configurations for each internal cell, this number could be reduced to only 78 (according to [Yerry,Shephard-1984]) using the various symmetry and rotational properties, as in two dimensions. As the full enumeration of these cases is error prone, an simpler approach is preferred to mesh any internal octant. In practice, any octant having all its neighbors equally sized is decomposed into six tetrahedra. Otherwise, the octant is decomposed into six pyramidal elements, the vertices of which are the octant corners and the centroid of the octant. These pyramids are then decomposed into tetrahedral elements according to the neighboring octant configurations (cf. Figure 5.13).



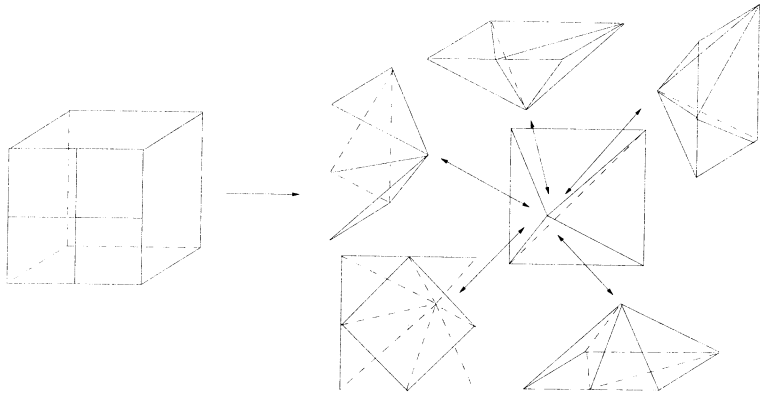


Figure 5.13: *Automatic triangulation of an internal octant. The octant is subdivided into six pyramids that are then meshed according to the neighboring octant sizes.*

When dealing with boundary cells, the procedure requires more attention. Mesh elements are created in two steps [Grice *et al.* 1988] :

- the sides of the cell are first discretized so as to ensure conformity with the neighboring cells,
- the barycenter of all points within the cell (boundary, intersection points and corners) is inserted and then connected to these points to form tetrahedral elements, in three dimensions.

Obviously, one has to check that the edges and faces corresponding to the boundary discretization are correctly formed in this procedure. In particular, two intersection points issued from the same edge must be connected with a mesh edge (to preserve the topology of the domain).

From the algorithmic point of view, the procedure is based on a tree traversal (a depth-first or a breadth-first traversal, the order not being significant at this stage). The data structure should permit the quick identification of the neighbors of a given leaf. Moreover, the topology of the resulting mesh is explicitly given as a list of elements, each element being described by the references of its vertices. Therefore, a mesh data structure can be very simple, as it is not used for neighborhood searching purposes, for instance. This is indeed a difference with

the advancing-front type methods, for which the current mesh is always searched to determine the candidate points in the vicinity of a given point (cf. Chapter 6).

**Remark 5.14** *The proposed method leads to a simplicial mesh of the domain (with triangular or tetrahedral elements). In two dimensions, it is also possible to obtain a mixed mesh (composed of triangles and quadrilaterals), other conforming patterns need then to be defined.*

**Removal of external elements.** The spatial decomposition obtained at completion of the previous stage is a covering up of the bounding box of the domain rather than a covering of the domain. The resulting mesh is then a mesh of the bounding box of the domain (similar to that obtained by constrained Delaunay methods, Chapter 7). The boundary discretization is present in the current mesh, thus making it easy to identify internal elements.

A very clever and simple algorithm, based on a coloring scheme (see Chapter 2), is used to mark the different connected components of the domain. It is then possible to keep the mesh of one specific component only. This algorithm can be summarized as follows :

1. Assign the value $v = -1$ to all bounding box mesh elements and set $c = 0$,
2. Find and stack an element having a value equal to $-1$.
3. Pop an element, while the stack is not empty,
   - if the value associated with the element is different from $-1$, go to Step 4,
   - assign the value $v = c$ to the element,
   - stack the three (resp. four) adjacent elements if the common edge (resp. face) is not a boundary item,
   - back to Step 3,
4. Set $c = c + 1$ and while there exists an element having a value $v = -1$, go back to Step 2.

Other techniques can be used to achieve the same result, all based on the identification of the various connected components of the domain.

**Boundary points relocation.** So far, we have considered the items of the boundary discretization to be straight segments. Thus, the introduction of intersection points during the tree construction does not pose any particular problems. However, if the boundary discretization is a piecewise polygonal (resp. polyhedral) approximation of the true boundary, it is fundamental that the resulting mesh vertices all belong to the true underlying curve (resp. surface). Indeed, for numerical simulation purposes, it is desirable to have a polygonal (resp. polyhedral) representation of the curve (resp. surface) for which the gap between the underlying geometry is bounded by a given tolerance value $\varepsilon$.



If a geometric model is available, queries to the modeling system provide the exact position of vertices corresponding to the intersections of the tree cells and the curve (resp. surface). This operation usually concerns a small number of vertices and does not lead to a substantial increase in the computational cost.

When the boundary discretization is the only data available, it is possible to construct a $G^1$ continuous geometric support that reasonably emulates the features of a geometric modeling system (especially, given a point, to find the closest surface point in a given direction, Chapter 19).

Notice that, if the initial boundary discretization is a geometric one (for which the edge lengths are proportional to the local curvature of the surface, Chapters 14 and 15), the final position of a point onto the geometric support is usually very close to its initial position. However, if the discretization is arbitrary, the node relocation can lead to the creation of invalid elements. That is why, is some cases, the relocation may be refused. Figure 5.14 illustrates the notion of geometric approximation for a given tolerance value (see also Section 5.4.1).

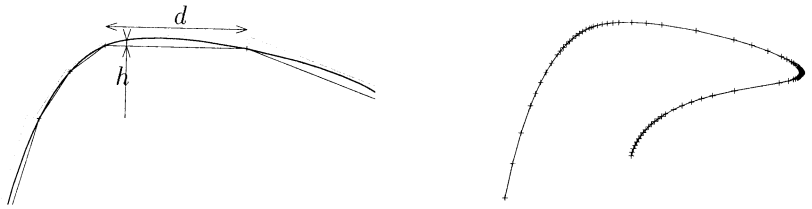



Figure 5.14: *Geometric approximation using a polygonal segment with respect to a given tolerance value $\varepsilon = 0.08$ (left-hand side) and $\varepsilon = 0.01$ (right-hand side). The relative tolerance is such that $h/d < \varepsilon$, with $h$ the distance between a point and the supporting edge and $d$ the edge length.*

### 5.2.7 Optimization

When all decomposition points have been inserted and all elements have been created within the terminal cells, we obtain a mesh of the domain that can be optimized. The goal is to obtain a finite element mesh and the optimization criterion must reflect this objective.

A quality measure for a mesh element $K$ is given by the following formula, Chapter 18 :

$$\mathcal{Q}_K = \alpha \frac{h_{max}}{\rho_K}, \tag{5.1}$$

with $h_{max}$ the diameter of $K$ and $\rho_K$ the radius of the inscribed circle (sphere). This leads to a raw value at the mesh level :

$$\mathcal{Q}_\mathcal{T} = \max_{K \in \mathcal{T}} \mathcal{Q}_K . \tag{5.2}$$

The optimization stage aims at minimizing this value. Notice however, that this quality measure has not been explicitly taken into account during the tree construc-

tion nor during the element creation step. Moreover, the mesh vertices corresponding to the cell corners confer a certain rigidity to the resulting mesh. Therefore, the optimization stage concerns, *a priori*, all mesh elements, noticing however that the elements created in internal cells are usually well-shaped.

**Optimization procedures.** The basic idea is to locally modify the mesh so as to progressively and iteratively improve its global quality. We commonly identify two categories of local modifications (see Chapter 18) :

- topological modifications that preserve the point positions and modify the connections between mesh vertices and
- metric modifications that change the point positions while keeping their connections.

Several operators allow these local mesh modifications to be carried out, and, more precisely to :

- move the points (for example using a weighted barycentrage technique);
- remove points;
- remove edges by merging their endpoints,
- flip (swap) edges (in two dimensions) or flip edges and faces (generalized swap in three dimensions).

The quality of the final mesh is improved by successive iterations, a judicious combination of these operators allowing a gain in quality. Notice also, that this optimization stage is (almost) identical to the one involved in other simplicial mesh generation methods (Chapters 6 and 7).

**Remark 5.15** *If the filtering stage has not been carried out, the optimization stage can be more time consuming. In particular, one has to deal first with the mesh elements which have a size that is not compatible with the given size specification and resulting from the intersections with the cell sides.*

### 5.2.8 Computational issues.

As clearly appears in the general scheme, the efficiency of a spatial decomposition method is strongly related to the criteria used during the tree construction stage.

**Basic algorithms.** A careful analysis of the general scheme shows three key points. They are related to the tree construction and the tree structure management, the insertion of the boundary discretization (intersection problems) and the implementation of predefined patterns for triangulating terminal cells.



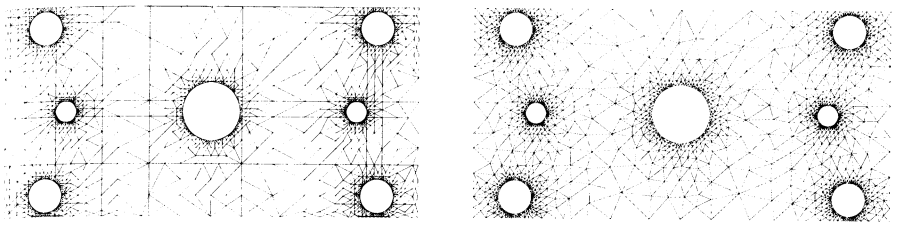


Figure 5.15: *Two dimensional mesh optimization. The initial raw quadtree mesh (left-hand side) and the final mesh after optimization (right-hand side).*

- Construction and management of the tree structure.

The tree structure is a dynamic structure that is updated during the insertion of the boundary discretization items and during the balancing stage. The use of a suitable data structure adapted to such operations (tree insertion, point searching operation, neighborhood, etc.) greatly facilitates the tree construction. One should favor a data structure allowing to have a direct access to the adjacent cells of a given cell so as to reduce tree traversals.

The search for a cell containing a given point is a common operation based on numerical checks. During tree construction, all cells have a similar shape (a square or a cube), thus allowing a bounding box test to be used to find the cell (the point coordinates being bounded by the cell extrema). However, if a filtering stage has been carried out, the shape of the cells is no longer preserved, thus making the search operation more tedious.

- Insertion of the boundary discretization.

The insertion of the boundary discretization involves numerical (metric) tests as well as topological tests (related to the neighborhood). Tree-like data structures are usually well suited to searching for items using adjacency relationships[5] However, the tree construction can lead to the division of a boundary item (for instance because of its proximity to another item). In this case, numerical information has to be propagated in all related cells. We have already mentioned that it could be useful to slightly modify the coordinates of a point so as to avoid numerical problems when inserting it into the tree. The intersection checks concern the pairs edge-cell (hence, edge-edge), edge-face and face-face.

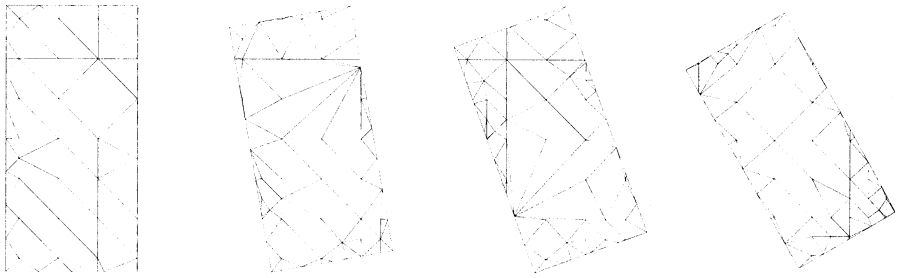
- Template implementation.

In order to improve the efficiency of the method, we have mentioned the use of predefined patterns to mesh the terminal cells (internal or external). From a practical point of view, each terminal cell is analyzed so as to determine its

[5] moreover they are used to this end in other mesh generation methods (cf. Chapters 6 and 7, for instance.

neighborhood and the result leads to a classification. A given pattern is associated with each class corresponding to a list of elements forming a conforming covering up of the cell.

**Decomposition-related issues.** We have already noticed that boundary integrity is usually not preserved with such methods. Moreover, the domain decomposition is related to its geometry as well as to its relative position in the classical frame of coordinates. In fact, each geometric transformation (translation, rotation) applied on the domain leads to different decompositions, thus to different meshes. Figure 5.16 shows different meshes obtained by rotating a rectangle, initially axis aligned, of angles of 10, 20, and 30 degrees respectively.



i) ii) iii) iv)

Figure 5.16: *Decompositions associated with a unique geometry : initial domain i), rotations of angles $\alpha = 10°$ ii), $\alpha = 20°$ iii) and $\alpha = 30°$ iv). Notice that the resulting meshes are dramatically different.*

In two dimensions, no convergence problem is faced. The only tedious problem is related to the proximity of two sides (almost tangent) that the tree construction will try to separate. This problem can be solved for instance by slightly distorting a cell so as to insert one of its corners in between the two items. The same strategy can be used in three dimensions to avoid convergence problems.

Notice finally that tree-based methods allow isotropic meshes to be created. The generation of anisotropic meshes (in arbitrary directions, Chapter 21) appears almost impossible (except in some specific cases, for instance in boundary layers, [McMorris,Kallinderis-1997]).

**Memory requirements and data structures.** Memory requirements correspond to the data structures needed to store the tree and the mesh. The main structures must allow access to :

- the list of vertex coordinates,
- the boundary items,
- the list of element vertices,
- the element adjacency relationships (in the optimization operations),



- the cell adjacency relationships,
- the list of boundary items per cell,
- other local resources.

## 5.3 Governed tree-based method

In numerous applications, the resulting meshes must satisfy specific properties, for instance related to the local element sizes, to the density of the elements, etc. The construction scheme then involves using a so-called *governed* or *controlled* mesh generation method that allows to match the desired requirements.

Such a control can be defined using an element size distribution function. This input data is in fact common to most unstructured mesh generation methods. A widely used technique consists of using a grid or a background mesh in which the elements (cells) keep track of the information related to desired local element size. From the discrete information (associated with the vertices of this covering up), one can use an interpolation scheme to construct a continuous size distribution function. This approach consists in defining a control space (see Chapter 1). In our case, the control structure is obviously a tree.

### 5.3.1 Control space

Suppose the size distribution function is known in a discrete manner, for instance at the vertices of a background mesh. From this discrete specification, a continuous size map is constructed using an interpolation scheme.

**Tree-based construction of the size distribution function.** Let $P$ be an arbitrary point in the domain. The size $h(P)$ at this point can be interpolated from the sizes $h(P_i)$, $i = 1, ..d$ (with $d$ the space dimension) at the vertices $P_i$ of the element enclosing $P$. Suppose that a decomposition tree is available (generated, for instance, according to the classical method described in the previous section). The points of the structure (the cell corners) will be assigned the size values. More precisely :

- For a given corner $Q$ of a tree cell,
  - find the element $K$ of the background mesh enclosing $Q$,
  - calculate $h(Q)$ using a $P^1$-interpolation of the sizes $h(P_i)$ at the vertices of $K$.

With this technique, we obtain a value $h(Q_i)$ associated with each corner of the tree cells. Then, to find the size at any point $P$ in the domain, we can simply use a $Q^1$ interpolation based on these sizes :

- For a given a point $P$,
  - find the cell containing $P$,

- calculate $h(P)$ using a $Q^1$ interpolation of the sizes $h(Q_i)$ at the cell corners.

When the background mesh is an empty mesh (with no internal point), these techniques result in a size distribution function that is exclusively related to the boundary discretization. However, if no background mesh is provided, it is still possible to construct a continuous size distribution function after the tree construction stage. In this case, the average value of the side lengths of the cell surrounding a corner is associated with each cell corner. A so-called *implicit* size map is then defined, which takes into account the boundary discretization and the domain geometry.

**Remark 5.16** *Notice that the discrete evaluation of the sizes as described above may result in an undesirable filtering effect if the cell sizes are not well adjusted.*

### 5.3.2 Governed tree construction

Once the control space has been defined, it is possible to modify the classical method so as to create size compatible elements with respect to the given specifications. As the elements sizes are related to the cell sizes they are created in, the basic idea is to modify the classical tree construction procedure to take these specifications into account.

The first stage of the construction remains unchanged. The boundary discretization items are successively inserted into the current tree. The resulting tree is thus adapted to the domain geometry (more refined in high curvature regions or when two sides are close and, conversely, less refined far from the boundary). Then, the tree is used together with the vertices of the background mesh to control the cell sizes.

At this point two approaches are possible. The first one consists in constructing the size distribution function as described above. One has to check that the size of each terminal cell (a square in two dimensions and a cube in three dimensions) is compatible with the average sizes associated with its corners. If not, the cell is recursively refined until the length of each sub-cell becomes less than or equal to the desired average length.

An alternative approach consists in initializing the sizes at each cell corner with an average value between the length of the adjacent cells (implicit map). Then, each vertex of the background mesh is embedded into the tree. Given the size $h(P)$ at a mesh vertex $P$, this size is compared with the size $h'(P)$ obtained using a $Q^1$ interpolation at the cell corners. If $h(P) < h'(P)$, the cell is refined and the process is repeated on the sub-cell enclosing the point $P$. Notice that the values associated with the cell corners are updated during the cell refinement.

In both cases, the tree is balanced according to the [2:1] rule.

### 5.3.3 Optimization

As mentioned in the classical case, it is usually very useful to optimize the mesh resulting from a governed spatial decomposition method.



**Edge length.** Let $AB$ be a mesh edge and let $h(t)$ be the size variation along the edge, such that $h(0) = h(A)$ and $h(1) = h(B)$. The normalized edge length $l_{AB}$ is defined as (cf. Chapters 1 and 10) :

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

with $d_{AB}$ the Euclidean distance between the two endpoints $A$ and $B$. Then , the edge $AB$ is said to conform with the size specification if its length is such that :

$$\frac{1}{\sqrt{2}} \leq l_{AB} \leq \sqrt{2}\,. \tag{5.3}$$

The coefficient $\sqrt{2}$ can be retrieved when comparing the length of the initial edge with the lengths of the two sub-edges after splitting the original edge. We are looking for the configuration that minimizes the error in distance to the unit length.

**Local modifications.** As for the classical case, the mesh quality is improved iteratively. The goal is to enforce unit edge lengths with respect to the specified metric (that is locally conforming to the size specification) for all mesh elements. Moreover, well-shaped elements are still aimed at. Therefore, a two-step optimization method is carried out, that first attempts to remove ill-shaped elements before optimizing the mesh in a global fashion. The second stage tends to improve shape quality as well as size quality using mesh modification operators (Chapters 17 and 18).

For practical reasons, the mesh elements are first optimized based on a size criterion and then the mesh shape quality is improved.

### 5.3.4 Application examples

In this section, several mesh examples obtained using a spatial decomposition method (classical or governed case) are proposed, in two and three dimensions. Figures 5.17 and 5.18 illustrate these examples. Table 5.1 gives some characteristic numerical values for different computational domains.

| - | $np$ | $nf$ | $ne$ | $\mathcal{Q}_M$ | $t$ (sec) |
|---|---|---|---|---|---|
| Example 1 | 465 | 678 | 1,629 | 4.3 | 12. |
| Example 2 | 2,548 | 4,028 | 8,761 | 8.2 | 174.3 |
| Example 3 | 5,792 | 7,506 | 22,681 | 8.2 | 346.5 |
| Example 4 | 10,705 | 9,412 | 48,667 | 6. | 115.8 |

Table 5.1: Statistics for different meshes obtained using a spatial decomposition method (the boundary discretization is the only data available to construct these meshes).

In Table 5.1, $np$, $nf$ and $ne$ denote respectively the number of vertices, the number of surface triangles and the total number of mesh elements, $\mathcal{Q}_M$ is the quality of the worst shaped element (after optimization) and $t$ is the CPU time ( Sun Ultra-1 Sparc workstation) required to mesh the domain. Table 5.2 shows the efficiency of the method, $v$ is the average speed of the method (*i.e.*, the number of elements created per minute). After the tree construction, the terminal (internal and external) cells are triangulated using templates and the boundary cells are meshed in a fully automatic manner. In this table, $t_B$ and $t_T$ denote the computational times devoted to tree construction (insertion of the boundary discretization) and the triangulation of non-boundary leaves, respectively.

The results clearly show that the speed $v$ ranges from 8,500 to 27,000 elements per minute. The difference between the lower and upper bound can be explained by noticing that as the number of mesh elements increases, the part devoted to tree construction (boundary discretization) becomes dominant, but the triangulation of the non-boundary leaves remains quasi-constant in times. The variations can also be explained by the geometric complexity of the computational domains that affect the tree construction stage.

As a conclusion, one could observe that the theoretical complexity of tree-based mesh generation methods is not easy to calculate. However, one can notice that the computational cost of the internal cells meshing process is proportional to the number of elements created in these cells and so to the number of internal cells in the tree decomposition. The latter number is related to the geometry of the domain, which is not the case for advancing-front or Delaunay-type mesh generation methods. The complexity of this category of methods is theoretically $\mathcal{O}(n \log(n))$, with $n$ being the number of mesh elements [Schroeder,Shephard-1989]. The term in $\log(n)$ can be explained by the use of a tree structure (in neighborhood and searching operations). In practice, the speeds observed are even closer to $\mathcal{O}(n)$ for the overall process.

## 5.4 Other approaches

In this section, we briefly mention several specific applications of the tree-based mesh generation method. In particular, we emphasize combined approaches (like Delaunay-octree, advancing-front-octree).

| - | $np$ | $ne$ | $t$ (sec) | $v$ | $t_B$ | $t_T$ |
|---|---|---|---|---|---|---|
| Example 1 | 1,060 | 3,558 | 24.9 | 8,580 | 12. | 2. |
| Example 2 | 4,089 | 15,947 | 74.9 | 12,780 | 30.1 | 2.2 |
| Example 3 | 25,887 | 124,213 | 278.5 | 26,760 | 99.8 | 13. |
| Example 4 | 27,686 | 134,826 | 375.4 | 21,540 | 115.8 | 42. |
| Example 5 | 79,820 | 380,183 | 823.8 | 27,689 | 305.7 | 36.5 |
| Example 6 | 104,015 | 498,509 | 1,100 | 27,191 | 400.6 | 47.5 |

Table 5.2: Efficiency and scalability of a spatial decomposition method, in three dimensions.



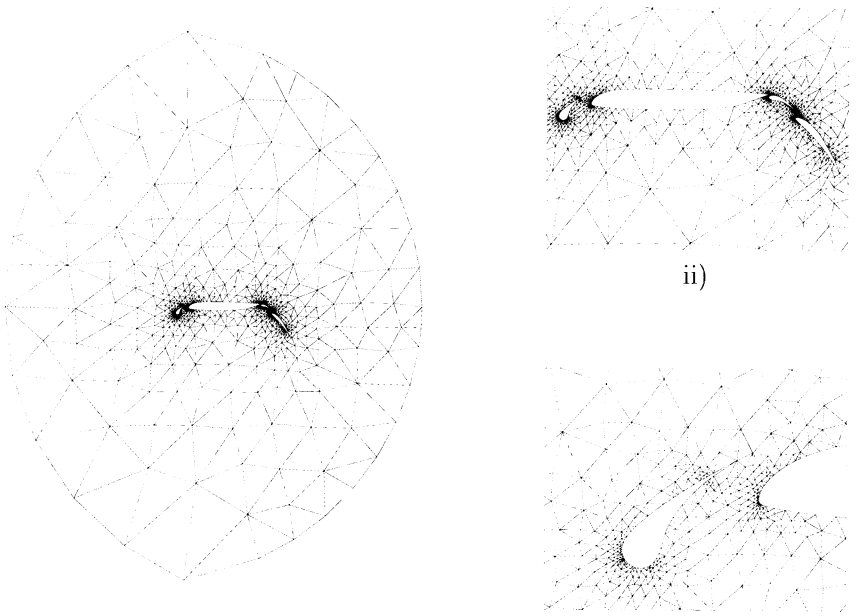



Figure 5.17: *Two-dimensional mesh of a computational domain and close-ups around the wing profile. The meshes have been generated using a quadtree-based method.*

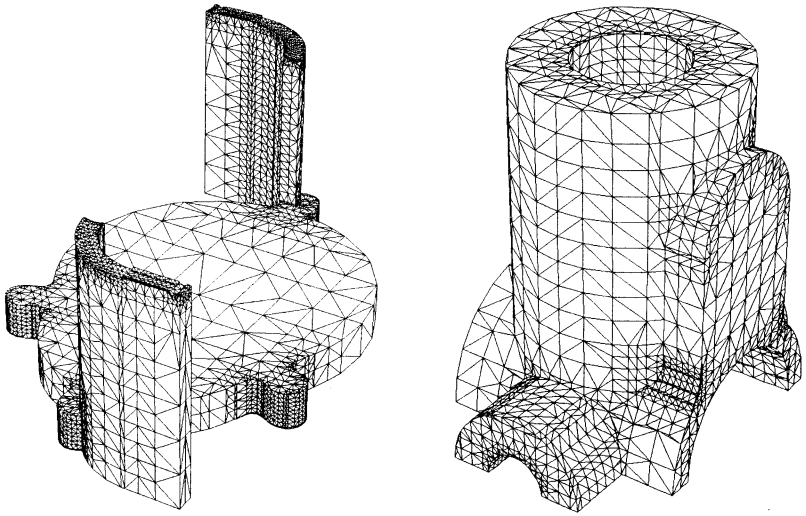


Figure 5.18: *Examples of meshes obtained using an octree-based method in three dimensions (data courtesy of the Mac Neal-Schwendler Corp.).*

### 5.4.1 Combined approaches

The strong point with tree-based decomposition methods is related to their ability to combine a control space and a neighborhood space in a single data structure. On the other hand, the main drawback of such methods is the lack of a real strategy for internal point creation based explicitly on the element (shape) quality. Indeed, one has to recall that the mesh vertices correspond to the terminal cell corners, which actually tends to rigidify the resulting mesh[6]. Moreover, numerous numerical problems are related to intersection computations in the vicinity of boundary items. This situation is due to the fact that the boundary is somehow embedded into the tree structure without any specific geometric consideration.

This is why some authors have found it useful to use the tree structure as a control space (to get the element sizes information) as well as a neighborhood space (to improve the searching operations), while using other approaches to create internal points and to connect them (*i.e.*, the mesh generation, strictly speaking).

- Delaunay/octree method.

Mesh generation based on a Delaunay-type algorithm generates a mesh point by point, each point possibly allowing the construction of several elements simultaneously. More precisely, the Delaunay kernel (cf. Chapter 7) is a procedure to connect the mesh vertices together (*i.e.*, to generate the mesh topology). First, the decomposition tree is built as in the governed case described above. The element sizes are then associated with the cell corners, the cell sizes being locally compatible with the desired element sizes. The cell corners as well as possible internal points (whenever the size variations are too great) are then inserted into the current mesh using the Delaunay kernel procedure.

**Remark 5.17** *Notice that the Delaunay algorithm used in this approach must be constrained so as to preserve the topology and the discretization of the boundary (cf. [Schroeder,Shephard-1988], [Sapidis,Perucchio-1993]).*

- Advancing-front/octree method.

In this case, the internal points are created and inserted into the domain using an advancing-front technique (Chapter 6). The optimal location of a candidate point is determined from the local sizes associated with the cell corners (these possibly being corrected so as to enforce a given mesh gradation). This technique has been successfully applied in mesh generation for viscous flow simulations in computational fluid dynamics [McMorris,Kallinderis-1997].

[6] In fact, it can be stated that this type of method is not really a mesh generation method [Schroeder,Shephard-1990].



### 5.4.2 Other approaches

The spatial decomposition techniques can be applied to the mesh generation with quadrilateral or hexahedral elements. In the approach suggested notably by [Schneiders *et al.* 1996] in three dimensions, the root of the tree is subdivided into 27 cells[7] which are then recursively subdivided until a given size criterion has been satisfied.

## 5.5 Extensions

In this section, we will mention the curve and surface mesh generation issues and make some comments on the use of a spatial decomposition method in the context of mesh adaptation.

### 5.5.1 Curve and surface mesh generation

In the previous sections, we have described the general scheme of the tree-based mesh generation method, under the assumption that the boundary discretization is provided. It is however possible to modify this scheme so as to construct the boundary discretization as well, this stage possibly being followed by the classical mesh generation procedure. This approach assumes that an adequate representation of the domain geometry is available (for instance provided by a geometric modeling system).

**Problem statement.** Surface mesh generation is a renowned complex problem notably related to the nature of the surfaces. Two approaches are commonly used to generate surface meshes. Whichever approach is used, the goal is to obtain a so-called geometric mesh such that the gap between the mesh edges and the mesh elements and the underlying surface is bounded. This control can be expressed in terms of size specifications (*i.e.*, the edge lengths are proportional to the principal radii of curvature, Chapters 14 and 15).

The indirect approach is used for parametric surfaces. It consists in meshing the surface via a parametric space using a classical mesh generation technique (for instance an advancing-front or a Delaunay-type method). The characteristic and the advantage of this approach is that the problem is reduced to a purely two-dimensional problem. The surface can be meshed using a two-dimensional mesh generation technique in the parametric space, then projecting the resulting mesh onto $\mathbb{R}^3$, via a (sufficiently smooth) transformation. However, to obtain a suitable surface mesh, one has to use a mesh generation method able to generate anisotropic meshes, the metric map definition being based on the intrinsic properties of the surface. Unfortunately, this feature immediately excludes the tree-based method.

[7] this 27-cell pattern has been chosen because of the relative ability with which a hexahedral mesh can be extracted from the tree structure.

On the other hand, if the surface is not considered via a parametric representation, the following tree-based technique can be used to create a surface mesh directly, without any transformation ([Grice *et al.* 1988], [Shephard,Georges-1991]).

**Control of the geometric approximation.** The depth $p$ of the tree can be related to the edge length $h$ and to the size $b$ of the bounding box $\mathcal{B}(\Omega)$ by the relation $p = \log_2(b/h)$. As the edge size $h$ is proportional to the minimum of the principal radii of curvature, denoted $\rho$ (cf. Chapter 15), we thus obtain a lower bound for the tree depth :

$$p \geq \log_2 \left( \frac{b}{\rho \, \alpha} \right) ,$$

where the coefficient of proportionality $\alpha$ is such that $\alpha = 2\sqrt{\varepsilon(2-\varepsilon)}$, with $\varepsilon$ is a given tolerance value. More precisely, let $h$ be the distance from a point to the supporting edge, and let $d$ be the length of this edge, one then has to enforce the relation :

$$\frac{h}{d} < \varepsilon ,$$

$\varepsilon$ being the given relative tolerance value (cf. Figure 5.14, left-hand side).

**Tree construction.** The proposed method follows the classical general scheme, with the difference that the tree construction stage is now based on an iterative algorithm for inserting the items of the geometric model (and not the items of the boundary discretization). The points (in fact, the singular points and more generally all points of $C^0$ discontinuity) are first inserted into the tree from the initial bounding box decomposition, according to the classical construction rules. Then, at each cell level, the geometric model is questioned (using a system of geometric queries) so as to determine the intersections between the model edges and the cell sides. Depending on the number of such intersections, the current cell may be refined. The model faces are in turn inserted into the tree.

The local intrinsic properties of the surface (principal radii of curvature, vertex normals, etc.) are used to construct the tree. In practice, the cells are refined until the geometric approximation criterion is satisfied.

**Main difficulties.** The difficulties related to this kind of technique are essentially the numerical problems encountered during the calculation of the intersections between the geometric model entities and the tree cells.

The following example emphasizes the complexity of the tree construction stage. Consider a cell intersected by at least two model edges. In Figure 5.19, the curved edge is inserted into the current cell, thus leading to the creation of a segment $AB$. Then, the edge $PQ$ is inserted and results in the creation of the straight segment $PQ$, that partially overlaps segment $AB$. This kind of problem can be solved either by subdividing the two curved edges into several straight segments (Figure 5.19, middle) or by refining directly the cell (same figure, right-hand side).



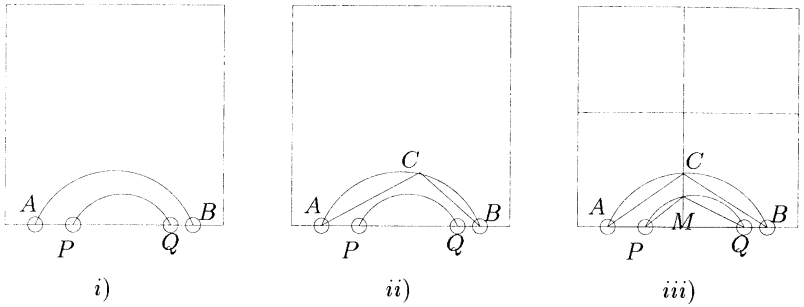



Figure 5.19: *Insertion of two curved edges into the same tree cell. i), problem of overlapping straight segments coming from the discretization. ii), the problem is solved by splitting the curved edge AB into two sub-segments. iii), the problem is solved by refining the quadrant.*

The difficulty is even greater in three dimensions. The intersections between the model face edges and the cell sides are calculated. The resulting intersection points are then connected, each octant face being processed separately, so as to approximate the intersection curves [Shephard *et al.* 1996].

**Remark 5.18** Finite Octree *meshing method, for instance, is a complete representation of the geometric model that has been partitioned. It maintains a hierarchical understanding of the portions of the model intersected by an octant. However, problems may occur, mostly related to the geometric tolerance, that may result in ambiguous configurations, Figure 5.20 shows such a problem : the sides intersected by the curve and that intersected by the discretization are not the same. Therefore, if the boundary discretization is built at the same time, this configuration can lead to consistency problems when the intersections between the edges and the quadrants are propagated into the tree structure.*

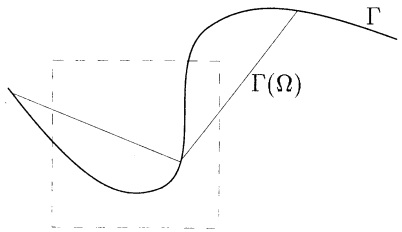



Figure 5.20: *Potential problem related to too great a tolerance value, in two dimensions. The sides of a quadrant intersected by the discretization and that resulting from the intersection of the underlying curve are not the same.*

### 5.5.2 Mesh adaptation

The objective of the computational adaptation scheme is to ensure the reliability of the solutions in the numerical simulation based on finite element methods. From a theoretical point of view, adapting the meshes to the physical properties of the considered problem should improve the accuracy of the numerical solutions and should guarantee the convergence of the computational scheme. The adaptation is based on a governed mesh generation method, in which the size map is provided by an *a posteriori* error estimate. Quadtree-octree based methods can thus be used in this context.

The mesh adaptation problem will be thoroughly reviewed in Chapter 21.

### A brief conclusion

Mesh generation methods based on a spatial tree decomposition technique are robust, efficient and flexible. Such methods allow the construction of classical as well as governed meshes (conforming to a prescribed size map). The use of the spatial decomposition structure as a control space is also an idea common to other methods. Moreover, the boundary discretization of the computational domain can be created using this approach. However, this kind of technique is not able to create anisotropic meshes (in which the stretching directions of the elements are arbitrary). In addition, the use of the cell corners to define the mesh vertices rigidify the resulting meshes (the optimization stage is usually more computationally expensive than with advancing-front or Delaunay-type mesh generation methods, Chapters 6 and 7).

Chapter 6

# Advancing-front technique for mesh generation

## Introduction

The advancing-front technique for mesh generation has been investigated for almost thirty years since the pioneering work of [A.George-1971] who studied a two-dimensional case. The classical advancing-front method, in its current form, was first described by [Lo-1985] and [Peraire *et al.* 1987]. Numerous improvements in this technique have been proposed over the years, first by [Löhner,Parikh-1988], [Golgolab-1989] and more recently by [Mavriplis-1992] and [Shostko,Löhner-1992]. This approach is now a very powerful and mature technique for generating high-quality unstructured meshes composed of simplices (triangles or tetrahedra) for domains of arbitrary shape. Variants of this technique have even been proposed to generate quadrilaterals or hexahedra in two and three dimensions (cf. [Blacker,Stephenson-1991], [Blacker,Meyers-1993]). To some extent, advancing-front techniques are often presented as competing approaches for the Delaunay-based mesh generators described in the next chapter.

Classical advancing-front approaches start from a discretization of the domain boundaries as a set of edges in two dimensions or a set of triangular faces in three dimensions (*i.e.*, the domains are considered as bounded regions). The name of this class of methods refers to a strategy that consists of creating the mesh sequentially, element by element, creating new points and connecting them with previously created elements, thus marching into as yet unmeshed space and sweeping a *front* across the domain. The process stops when the front is empty, *i.e.*, when the domain is entirely meshed. The *front* is the region(s) separating the part (or parts) of the domain already meshed from those that are still unmeshed. Hence, the front can have multiple connected components (cf. Figure 6.1).

One of the critical features of any advancing-front approach is the creation of internal points. This procedure should result in elements that, usually, satisfy size and shape quality criteria. Such a size criterion, and thus the corresponding

element-size distribution function prescribes the desired element shapes and sizes (review the notion of control space in Chapter 1).

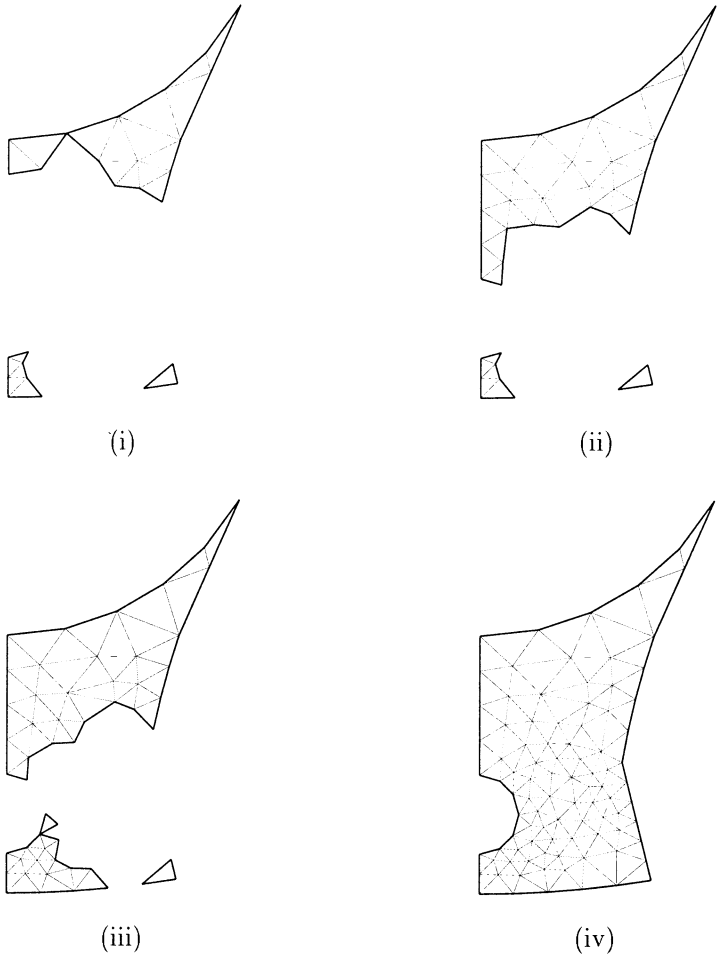



Figure 6.1: *Various stages (i - iv) illustrating the multiple components of an evolving front in two dimensions.*

The main advantage of current advancing-front techniques is their heuristic mesh generation strategy, which tends to produce high-quality elements and nicely graded meshes. Moreover, in contrast to some other automatic methods, boundary integrity is always preserved, as the discretization of the domain boundary constitutes the initial front, which is not the case for some other mesh generation methods (for instance, see Chapters 5 and 7). On the other hand, convergence problems can occur, especially in three dimensions, as it is not always clear how to complete a mesh of the entire domain. In other words, one can find some regions which are not easy to fill up with elements (mostly in three dimensions).



★
★ ★

In the first section of this chapter, the framework of a classical advancing-front method is outlined, which takes into account most features of this class of techniques. The critical features of the approach are described with emphasis on specific topics such as optimal point creation, geometric checks and mesh gradation. Some indications are also given about convergence issues. The second section explains how robustness and reliability issues can be addressed using a control space. Then, in the third section, some details are discussed about a governed method where the control corresponds to the data of a control space. The coupling of the advancing-front method with a Delaunay technique is discussed in the fourth section and the capabilities of combined or hybrid approaches are raised. Finally, possible extensions of the method to anisotropic mesh generation, surface mesh generation and mesh adaptation (for the mesh adaptativity problems discussed in Chapter 21) are introduced in the fifth section.

## 6.1 A classical advancing-front technique

An advancing-front technique is generally part of a mesh generation procedure including three successive steps :

Step 1 : mesh parameterization (boundary description, specification or construction of a function defining element size distribution, etc.),

Step 2 : boundary discretization,

Step 3 : the creation of the field vertices and elements, in other words, the advancing-front method itself.

This general scheme that will be found in other methods such as the Delaunay type method (Chapter 7), is slightly different to that of a *quadtree*-based method (Chapter 5). Indeed, in contrast to the latter case, here the boundary mesh is an input data that must be preserved.

Following a classical advancing-front approach (Step 3 of the above scheme), the mesh vertices are created as part of the procedure of mesh element creation. Numerous variants of this technique have been proposed by [Bykat-1976], [Peraire *et al.* 1988], [Löhner,Parikh-1988] [Dannelongue,Tanguy-1991] and also [Jin,Tanner-1993] and [Farestam,Simpson-1993]. Despite several differences between these approaches, a classical advancing-front method is, inherently, based on a local insertion mesh generation strategy relying on geometric criteria.

Although the fundamental steps of mesh generation are the same in two and three dimensions, the development of robust, reliable and efficient algorithms in three dimensions is much more tedious than that in two dimensions, as is shown below.

### 6.1.1 General scheme

Formally speaking, the advancing-front procedure is an iterative procedure, starting from a given boundary discretization, which attempts to fill the as yet unmeshed region of the domain with simplices. At each step, a front entity is selected and a new (optimal) vertex is created and possibly inserted in the current mesh should it form a new well-shaped simplex. The main steps of the classical advancing-front algorithm can be summarized as follows :

Step 1 : Preliminary definitions

- data input of the domain boundaries (*i.e.*, the boundary mesh $\mathcal{T}$)
- specification of an element-size distribution function (governed mesh generation) or the best possible construction[1] of such a function (classical problem).

Step 2 : Initialization of the front $\mathcal{F}$ with $\mathcal{T}$

Step 3 : Analysis of the front $\mathcal{F}$ (as long as $\mathcal{F}$ is not empty) :

(a) select a front entity $f$ (based on a specific criterion),

(b) determine a best-point position $P_{opt}$ for this entity,

(c) identify if a point $P$ exists in the current mesh that should be used in preference to $P_{opt}$. If such a point exists, consider $P$ as $P_{opt}$,

(d) form an element $K$ with $f$ and $P_{opt}$,

(e) check if the element $K$ intersects any mesh entity. If this check fails, pick a new point $P$ (if any) and return to 3.*d*.

Step 4 : Update the front and the current mesh :

- remove entity $f$ from the front $\mathcal{F}$ (and any entity of $\mathcal{F}$ used to form $K$),
- add the entities of the newly created element $K$ members of the new front,
- update the current mesh $\mathcal{T}$.

Step 5 : If the front $\mathcal{F}$ is not empty, return to Step 3.

Step 6 : Mesh optimization.

In the following sections, we give some details about these steps while discussing the main difficulties involved in the process.

[1] See Sections 6.2.1 and 1.6.



**Main difficulties.** The above scheme gives the principles of the method but does not reveal a series of delicate issues that must be addressed. In particular, there is no guarantee of robustness or convergence in three dimensions[2]. In fact, the tedious steps of the method are primarily concerned with :

- the selection of one entity in the front,
- the front analysis and the various possibilities for defining the optimal points,
- the element validation at the time an element is constructed,
- the use of appropriate data structures.

The three-dimensional implementation of this approach requires a great deal of attention to ensure a robust algorithm (especially regarding convergence, *i.e.*, the guarantee that the method really completes the mesh) with a certain extent of efficiency.

However, the main difficulties of an advancing-front type meshing process are relatively easy to identify. They are primarily related to

- the proper identification of the local situation at some neighborhood of a point or a region,
- the adequate intersection identification (for edges, faces, etc.) used, for instance, to validate one element before inserting it or to detect if a point falls outside the domain, etc.
- the definition of an optimal location for a point or the selection of an existing point when elements must be created based on a given front entity.

Clearly, any newly created mesh point must result in valid and well-shaped element(s), and must prevent, as much as possible, any configuration from further meshing difficulties.

**Remark 6.1** *The difficulties are then related to the methodology (point optimal definition) or to numerical aspects (intersection checks, efficient access to some neighborhood of an entity).*

### 6.1.2 Preliminary requirements

The proper achievement of an advancing-front method relies on a certain number of assumptions (about what the data input are) and on a well-suited choice about the necessary data structures.

[2] Although no specific difficulty is encountered in two dimensions.

**About the domain boundary mesh.** The boundary mesh is supplied together with an orientation convention. In a manifold case, this means, in two dimensions, that the polygonal discretization (the boundary mesh) consists of segments defined and traversed in a given direction (based on an orientation convention, for instance, the domain is in the region "right" to the given boundary). Similarly, in three dimensions, the faces members of the domain boundaries must be orientated in an adequate way, the domain being still defined by its position with respect to these faces. This requirement is used furthermore to identify the relative position of the domain with respect to a segment (a face) of the boundary (indeed, with respect to the half-spaces separated by this entity[3]).

The non-manifold case is more subtle in the sense that two situations may arise. There is an edge (a series of edges) which is not a boundary of a connected component of this domain (such an edge could be an isolated edge or may have only one endpoint located in a boundary) or, conversely, such an edge belongs to the interface between two connected component. In the first case, there is no longer an orientation problem, the domain is on one side of the entity and is also on the other side of it. In the second case, the entity separates two components and therefore has a different orientation for each of these components. Figure 6.2 depicts a case where the boundary discretization includes several connected components.

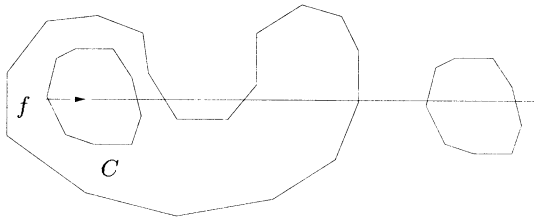



Figure 6.2: *Orientation of a boundary connected component C. Entity f must be re-orientated in the case where the region to be meshed corresponds to the shaded part.*

**Data structures.** Based on the nature of the operations needed in an advancing-front method, it is relevant to define some specialized data structures, [Löhner-1988], [Bonet,Peraire-1991]. The objective is twofold : to make the management of the front fast and easy once the process progresses and, on the other hand, to provide easy access to all the entities present in some neighborhood of a given point or a given region.

The front management requires a topological data structures that allow the suppression or the insertion of an entity (an edge in two dimensions, a face in three dimensions, cf. Chapter 2).

Access to some neighborhood of a point is made using a data structure like a neighborhood space (cf. Chapter 1).

[3] Actually, one can retain an orientation convention such that the face normals are orientated towards the interior of the domain.



### 6.1.3 Analysis of the front

The mesh elements are created based on the edges (faces) of the current front. The way to to this is related to the way in which the front is analyzed and thus to the choice of such or such a front entity.

The identification of a front entity from the generation front is an non-negligible task that can be performed in various ways depending on what benefit is expected. In general, the mesh generation does not proceed locally, but rather bounces randomly from one side of the domain to another. It seems obvious that this chaotic behavior has several undesirable side effects [Frykestig-1994] :

- unnecessary complexification of the front,
- loss of the possibility of using the local information previously collected,
- overall degradation of the performances (due notably to an increase in cache memory management (Chapter 2)).

One can always object that these types of problems should not arise with a smooth element-size distribution function. In three dimensions however, this is not really the case for various reasons. First of all, the front generally presents a chaotic aspect (as previously mentioned), even in simple geometric context and for smooth size functions (such as when a constant size is specified). Moreover, the desired size distribution can be tedious to follow (for instance, in CFD problems where size chocks and/or boundary layers must be considered).

More schematically, according to [George,Seveno-1994], the front entity can be chosen in a set of mesh entities (a subset of all front entities) corresponding to :

- all entities satisfying specific properties (angles, areas, lengths, etc.),
- an offset of a part of the initial front (at the beginning) and of the current front after,
- all the front entities.

An ordering can be defined over this set using several criteria. For instance, the selected front entity can be the entity leading to the smallest element, in order to avoid large elements crossing over regions of small elements [Löhner-1996b]. This can be useful for instance in the case of two merging fronts, if the element size varies rapidly over the region between these fronts (as illustrated in Figure 6.3). Other approaches consist of using the shortest edge (of a face, in three dimensions) [Peraire *et al.* 1992], the smallest entity [Jin,Tanner-1993] or even ordering the entities with respect to their areas [Möller,Hansbo-1995]. The selection of the front entity can also take the relative position of this entity with respect to its neighbors (incident entities). These approaches are usually carried out using a heap structure [Löhner-1988], for instance, based on binary trees (Chapter 2).

As the objective is to generate a good-quality mesh, another technique aims at creating locally the best element possible based on the worst front entity. This approach clearly makes sense in three dimensions since in two dimensions it is *a*

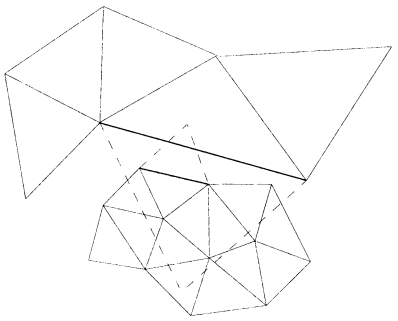


Figure 6.3: *Merging fronts of very dissimilar sizes and possible case of failure. The dashed lines illustrate the optimal elements that can be formed based on the selected front entities.*

*priori* always possible to form an equilateral triangle based on an arbitrary edge (provided the point to be defined is inside the domain). This is true, specifically, only in the configurations with corners where, in two dimensions, it is impossible to connect two adjacent edges having a rather acute angle one with the other [Jin,Tanner-1993]. Finally, the approach suggested by [Rassineux-1997] and [Seveno-1997] tends to minimize the size of the front and to take advantage of the (local) information previously collected. As a consequence, the number of intersection checks is also reduced, thus impacting favorably the overall meshing time. This approach is claimed to make mesh generation easier as it seems to reduce the number of degenerate cases. In practice, the example illustrated in Figure 6.7, becomes highly unlikely to happen as the algorithm tends to *convexify* the front. The approach starts from a given front entity and deals successively with any neighboring front entity that has not already been considered, thus resulting in a "spiral-like" progression.

**Exercise 6.1** *Implement the spiral procedure using a coloring algorithm and a FIFO (first in-first out) data structure, Chapter 2. Show that the previous algorithm proceeds by connected components.*

**Remark 6.2** *Regardless of the approach selected, the main concern should always be to design and implement a robust algorithm (*i.e.*, for which convergence can be guaranteed) resulting in high-quality meshes.*

### 6.1.4 Field point creation

Central to the advancing-front technique, the creation and insertion of an optimal point from a front entity requires a lot of attention. Let us assume that a front entity has been selected and that the desired element size can be known locally[4]. The objective is to discretize the domain in a set of well-shaped elements of arbitrary gradation. This can be carried out in the following steps :

[4]See Chapter 1 for the use of a control space to determine locally the desired element size.



- a local stepsize, denoted $h$, (e.g. a node spacing function or an element size value) is associated with each mesh vertex, based on an average value of the edge lengths (surface areas) sharing the vertex for a boundary vertex and computed using function $h$ for the mesh vertices after their creation.
- for a given entity, a point is defined (see hereafter) which is then connected with this entity.
- this choice is validated in accordance with the geometry and the size function. In the case of a point rejection, the previous step is repeated while picking another point.

Three types of different points are tried as candidates for element vertices :

- the optimal point with respect to the given front entity,
- some already existing points in the current mesh which are close (in some sense) to the front entity under analysis,
- some (guest) points created in a given neighborhood of the above optimal point.

As will be seen, these points will be the possible *candidates* from which the element vertices will be chosen.

**Optimal point.** Several construction schemes have been proposed for determining the location of the optimal point associated with a front entity. In two dimensions, a purely geometric idea defines as an optimal point that which allows for the construction of an equilateral triangle when combined with the given front edge. In three dimensions, the optimal point is that which allows the most regular tet possible to be built from a given front face (see Figure 6.4). An approach that takes advantage of the desired local size $h(P)$ (*i.e.*, the edge length) in any point $P$ is also a widely used solution. Specifically, this is a *natural* solution for a governed mesh generation method (as will be seen later). Thus, in three dimensions, the classical technique advocates positioning the point along the normal direction passing through the centroid of the front entity at a distance related to the magnitude of the desired element size (*i.e.*, the average distance from this point to the vertices of the front entity leads to the desired element size) [Peraire *et al.* 1992], [Jin,Tanner-1993].

Let $K$ be a triangle of the front, whose centroid is denoted $G$, and let $h(G)$ be the desired element size at the point $G$. If $\vec{n}_K$ represents the inward normal to $K$, the position of the optimal point $P_{opt}$ can be calculated as follows :

$$\overrightarrow{GP_{opt}} = \alpha h(G).\vec{n}_K \,, \tag{6.1}$$

where $\alpha$ is a normalization coefficient such that if $K$ is an equilateral triangle, the resulting tetrahedron will be regular (if the size map $h$ is constant).

**Exercise 6.2** *Determine the value of the coefficient $\alpha$ (Hint : look at the specific above configuration).*

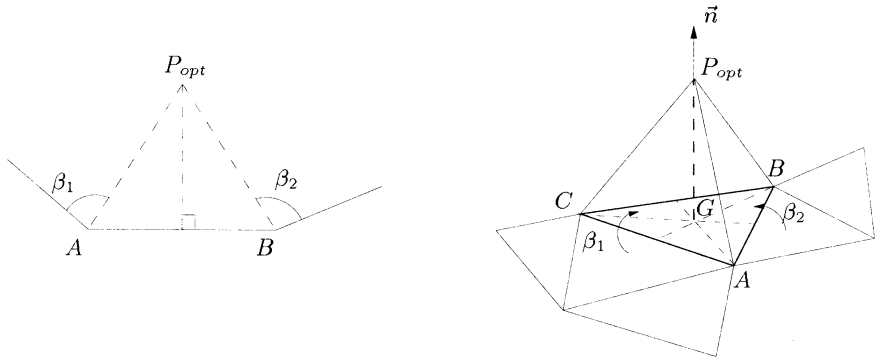



Figure 6.4: *Optimal point creation in two dimensions (left-hand side) and three dimensions (right-hand side).*

**Remark 6.3** *This construction assumes that the selected front entity is well-shaped. Nevertheless, a correction procedure has been proposed to account for distorted entities [Frykestig-1994], that basically consists in performing the computation in a* normalized *space, obtained by scaling the coordinates by the inverse of the desired mesh size value locally.*

**Remark 6.4** *Once the optimal point $P_{opt}$ has been located, the element size $h(P_{opt})$ at $P_{opt}$ can be used to relocate $P_{opt}$ so that the element size matches the desired element-size specification (cf. Section 6.2).*

The optimal point $P_{opt}$ (with respect to a given front entity) is not directly inserted in the mesh. Indeed, it is usually preferable to use an existing vertex as the optimal point rather than introducing a new one. Therefore, the other possible candidates are identified, collected and possibly ordered according to their suitability to form well-shaped elements with the selected front entity.

**Potential candidates.** The identification of other candidate points is carried out according to a distance criterion. The closeness of a point is related to the local element size specification $h(P_{opt})$ at the optimal point $P_{opt}$. A point $P$ can be considered as a candidate (*i.e.*, is sufficiently close) if it belongs to the circle (sphere) of radius $h(P_{opt})$ centered at $P_{opt}$, thus satisfying the following relationship :

$$\|\overrightarrow{P_{opt}P}\| \leq h(P_{opt})\,. \tag{6.2}$$

**Remark 6.5** *Given a front entity $K$, the points of the adjacent front entities $K_i$ are automatically added to the set of admissible points, if the angles $\beta_i$ (the dihedral angles in three dimensions) between $K$ and $K_i$ are less than a given threshold value (cf. Figures 6.4 and 6.5) irrespective of their distances to $P_{opt}$.*

**Exercise 6.3** *Find adequate values for the angle and dihedral angle bounds in two and three dimensions.*



**Exercise 6.4** *Explain how a neighborhood space (cf. Chapter 1) makes it possible to efficiently identify the mesh entities closely spaced to the given optimal point.*

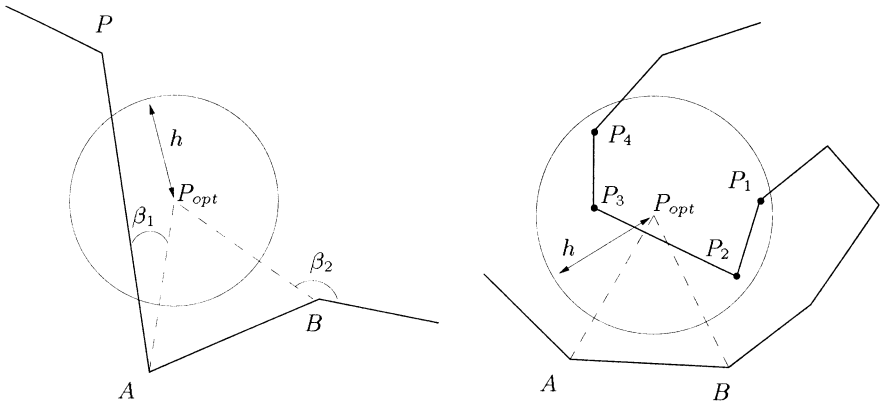



Figure 6.5: *Identification of all potential candidates for optimal point creation in two dimensions. Left : no candidate other than $P_{opt}$ exists. Right : the $P_i$'s represent all the possible candidates.*

**Remark 6.6** *Several authors [Jin, Tanner-1993], [Frykestig-1994] advocate the use of* tree points *that are constructed on predetermined positions (close to $P_{opt}$) to resolve complex front configurations.*

**Remark 6.7** *The input of specified points at the beginning of the construction (added to the boundary discretization) is also a case frequently encountered that induces some slight modifications of the algorithm.*

The set of potential candidates $P_i$ is ordered according to the increasing distance from the optimal point $P_{opt}$. They will be considered for element generation with this ordering. The optimal point is also inserted in this set and will be considered prior to some of these points, according to the distance criterion.

**Filtering.** As the computation of the optimal point location $P_{opt}$ does not explicitly account for existing vertices, an existing mesh vertex $P$ can incidentally be closely spaced to $P_{opt}$ which must then be discarded to the advantage of the existing vertex $P$.

### 6.1.5 Validation

The candidate points, including the optimal point, must of course satisfy certain *validity criteria* prior to being considered as potential mesh vertices. The purpose of the criteria is to ensure that the resulting mesh :

- is topologically and geometrically valid,
- is of good-quality,
- conforms to the specified element-size distribution function.

**Definition 6.1** *A mesh is topologically and geometrically valid if all mesh elements are conforming (Definition 1.2) and have positive surfaces (volumes).*

In practice, for a given candidate point $P_i$, conformity can be ensured by checking that none of the edges (faces) of the element intersects the front and that no existing mesh element contains the point $P_i$.

**Exercise 6.5** *Show that this geometric check can be used to determine whether a point $P_i$ belongs to the domain (to the relevant connected component) or not.*

**Definition 6.2** *A candidate element is a valid element if none of its edges (faces in three dimensions) intersect any front entity and if it does not contain any mesh entity (for instance a vertex or an element).*

Notice that this latter case (where an element exists) is a somewhat specious case. For instance, in two dimensions, we meet one element (or a series of elements) fully included in the element under analysis.

All this leads to performing a certain number of validation checks which reduce to intersection checks. The intersection checks are relatively straightforward to carry out (as the elements are supposed straight-sided). However, if the geometry checks are not carefully implemented, they can be impressively time consuming. For instance, [Löhner,Parikh-1988] reported that more than 80% of the meshing time can be spent checking the intersections. Hence, the overall meshing speed is strongly related to the algorithm used to check whether the elements intersect or not. Therefore, several efficient solutions have been proposed reducing the time to a mere 25% of the total meshing time [Löhner-1988], [Jin,Tanner-1993]. In short, one uses data structures (binary tree or quadtree/octree, etc.) and filters so as to reduce the number of entities that must be checked (cf. Figure 6.6).

**Remark 6.8** *The complexity of the intersection checks is almost constant (*i.e.*, does not depend on the mesh size). Indeed, the complexity is proportional to the number of neighboring elements collected, which is itself related to the size of the neighborhood.*

**Additional checks.** From the computational point of view, the notion of validity as envisaged so far is not sufficiently reliable. Indeed, it is certainly not enough to consider exclusively the new element to be created, as its insertion may affect the front configuration in such a way that further insertions may be virtually impossible[5]. Therefore, a practical approach consists in analyzing the distances between the edges of the element to be inserted and the front edges. This check makes it possible to control the space left after the element has been inserted.

[5]This situation is especially acute in three dimensions.



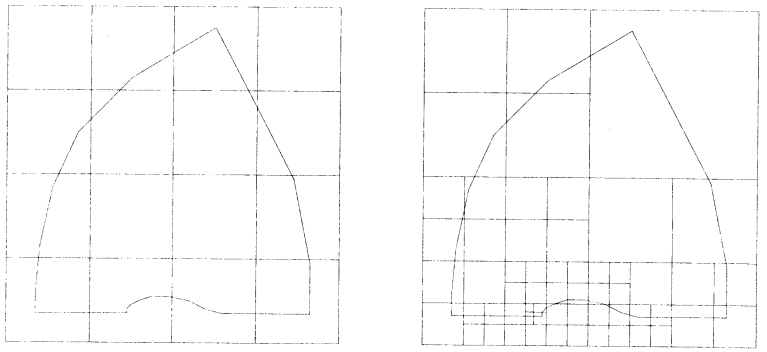


Figure 6.6: *For this two-dimensional example, a PR-quadtree structure (right-hand side) is better fitted to the domain geometry than an uniform grid (left-hand side) as a neighborhood space.*

In three dimensions exclusively, the worst problem encountered involves a configuration in which all front faces delimit a polyhedron that cannot be meshed without introducing a point. In this case, the problem comes down to determining a possible location for the internal point (cf. Figure 6.7). Convex polyhedra are easy to triangulate, but additional points may be needed to mesh a non-convex polyhedron [Lennes-1911], [Schönhardt-1928].

This naturally leads us to convergence issues of advancing-front algorithms.

So far, to continue the discussion, we assume that such a problem is not an issue, *i.e.*, we assume that in the set of the candidate points there is always a non empty set of *a priori* admissible points (for topologic and geometric validity).

**Quality checks.** At this stage of the evaluation procedure, a candidate element has been declared acceptable based only on geometric considerations. As the aim is to create well-shaped elements (as regular as possible), the introduction of a quality check may be desirable. A candidate point, that successfully passed the geometric checks, is nevertheless discarded if the resulting element shape quality is poor. Several quality measures have been proposed (cf. Chapter 18), a *natural* measure for a simplex $K$ being :

$$\mathcal{Q}_K = \alpha \frac{h_{max}}{\rho_K}, \tag{6.3}$$

where $h_{max}$ is the length of the longest edge of $K$, $\rho_K$ is the in-radius of $K$ and $\alpha$ a normalization coefficient (such that $\mathcal{Q}_K = 1$ if $K$ is the regular element, the quality function $\mathcal{Q}_K$ ranging from 1 to $\infty$).

A candidate element $K$ is thus discarded if its quality exceeds a user-specified threshold value $\eta$, *i.e.*,

$$\mathcal{Q}_K > \eta \,. \tag{6.4}$$

The $\eta$ value is not necessarily fixed during the process, which leads to different strategies. It is in fact possible to modify this threshold value (and degrade the

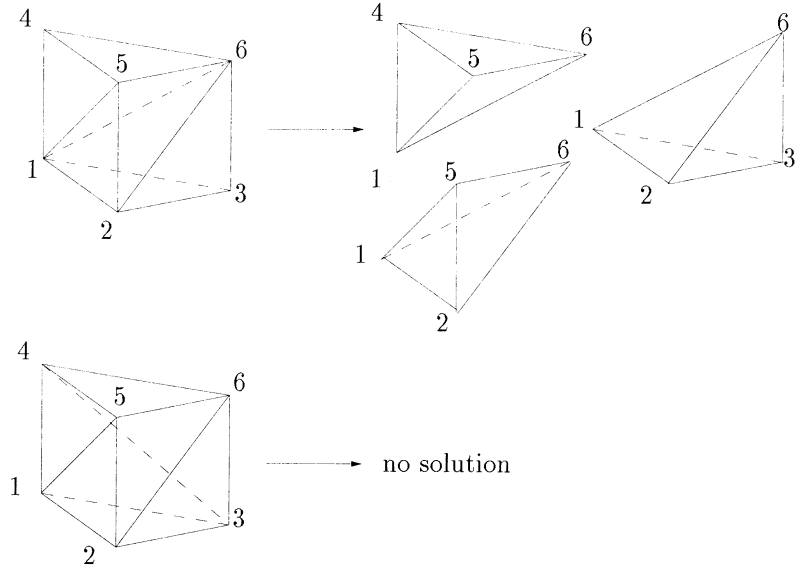



Figure 6.7: *Schönhardt polyhedron : valid and non-decomposable (without adding an internal point) constrained triangulation of a regular prism.*

shape quality) just to find a valid solution. It has been pointed out that a too strict a value leads to good-quality elements but, on the other hand, may have some negative effect on the convergence (and, in any case, is not always possible).

**An experimental analysis.** As seen at the beginning of this discussion, the principle of any advancing-front method is relatively simple (and admitted) but several strategies can be used in an actual implementation, all of them being based on heuristics. Thus, it seems to be of interest, since no theoretical issues can be involved, to look more closely at the behavior of such or such a strategy. To this end, it is necessary to have several implementations available so as to be able to make some comparisons. This fact is clearly unlikely to be realistic. Nevertheless, after [Seveno-1997], some statistics can be provided that give some interesting ideas. The method used relies on the above scheme and the element validation is made using the tools previously described. A large enough set of examples has been obtained and the nature of the retained connections for element creation has been reported. This appears to be a useful piece of information , obtained *a posteriori*, to give information about which strategy could be adopted (and also on how to optimize its implementation).

Table 6.1 shows, in three dimensions, the frequency of the various connections between a front face and an entity when forming an element. In this table, $np$, $ne$ and $nf$ represent the numbers of vertices, edges and faces, respectively created for a given front face.



| Face $K$ connected with | $np$ | $ne$ | $nf$ | frequency % |
|---|---|---|---|---|
| 1 adjacent face | 0 | 1 | 2 | 59.4 |
| 2 adjacent faces | 0 | 0 | 1 | 23.7 |
| optimal point | 1 | 3 | 3 | 13.4 |
| other cases | 0 | 1 | 3 | 3.5 |

Table 6.1: Frequencies of the connections with adjacent mesh entities, given a front face $K$ in three dimensions.

It appears that the front face is directly connected to an existing mesh vertex from an adjacent entity in about 80% of the cases. Hence, point creation represents at most 20% of the whole element creation process. Moreover, the table shows that, on average, two faces are created when constructing one new element.

These statistics give us some indications. The first leads us to prefer, particularly in three dimensions, an existing vertex instead of a new point. Another issue related to convergence argues again for this choice that, in practice, reduces the number of delicate situations. The second indication, as proposed in [Golgolab-1989], leads to ordering the candidate points based on their probability of being connected with a given front face.

### 6.1.6 Convergence issues

The notion of a convergence for a mesh generation method concerns its capability to complete a mesh in all situations. Here, it reduces simply to entirely "filling up" the domain. As previously indicated, this point is not really relevant in two dimensions. Indeed, the following theorem holds (as proven in [George,Borouchaki-1997] (Section 3.3).

**Theorem 6.1** *In two dimensions, any arbitrary domain defined by a polygonal non-crossing contour can be triangulated without adding an (internal) point.*

This interesting result has a direct consequence on the convergence of the iterative advancing-front scheme.

**Corollary 6.1** *In two dimensions, an iterative advancing-front mesh generation technique always converges in a finite number of steps.*

Actually, at each time, the iterations of the method can be stopped and the remaining unmeshed region(s) can be filled up.

In three dimensions, there is no similar issue. Indeed, irrespective of the numerical problems and provided an adequate strategy is used, the existence of a local configuration like that depicted in Figure 6.7 indicates that there is no guarantee of convergence.

**Remark 6.9** *It is worth noticing that this negative issue is still present in a Delaunay type method where it appears in a different way. With an advancing-front method, the point is to complete the mesh while with a Delaunay approach, the*

*delicate point is related to the enforcement of a given boundary (or not) entity (edge or face).*

Nonetheless, there are numerous examples (in three dimensions) of advancing-front based software which converge in most cases. As indicated, the strategies used try to minimize the number of delicate situations. However, some authors propose, when faced with bottle-necked situation, removing some previously created elements in some neighborhood so as to suppress the tedious pattern (while taking care not to produce the same pathology again or to provoke a cyclic process of creation-removal).

### 6.1.7 Point insertion and front updating

Given a front entity $f$, once a point has been identified (among $P_{opt}$, the candidate points, the trying points, etc.), and having successfully passed the different tests, it can be inserted in the current mesh. This stage consists of creating *one element* based on $f$ and this point.

**Remark 6.10** *As an optimization, specific cases allow multiple elements to be created during the insertion of a single optimal point (cf. Figure 6.8, in two dimensions).*

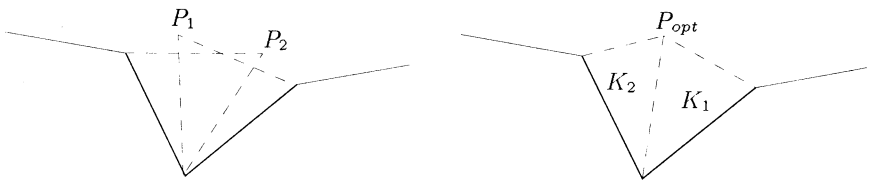



Figure 6.8: *Creation and insertion of a single optimal point $P_{opt}$ resulting in the creation of two elements $K_1$ and $K_2$ in two dimensions (right-hand side).*

Updating the mesh simply involves adding this (these) element(s) into the list of elements already constructed.

The last stage of the iterative procedure corresponds to the front update. In practice, this procedure :

- removes the (current) front entity from the front which is now a member of the element constructed, if a new point has been introduced or
- adds to the current front the edges (faces) of the constructed element not shared by any other element.

We meet here the fact that the data structure devoted to front storage must be such that adding or suppressing any entity is made easy and fast.



### 6.1.8 Optimization

Once all the points have been inserted and once the corresponding elements have been constructed, we have a mesh that could be optimized to some extent. The purpose is then to optimize a quality criterion related to the future application (a finite element style computation for instance).

As previously mentioned, a well-suited quality measure, for an element $K$ in the mesh, is :

$$\mathcal{Q}_K = \alpha \frac{h_{max}}{\rho_K}, \tag{6.5}$$

which, for the whole mesh, leads to a global value like :

$$\mathcal{Q}_{\mathcal{T}} = \max_{K \in \mathcal{T}} \mathcal{Q}_K. \tag{6.6}$$

The aim is then to minimize these values (including the global value). In fact, it could be observed that the method retained when choosing the points and thus when constructing (validating) the mesh elements has already considered a quality criterion. Nevertheless, for some convergence reasons, this criterion may have been violated in some cases. Also, the strategy generally leads to meshes with a majority of good quality elements (as a function of the threshold value $\eta$). Hence, in principle, the optimization stage is mostly of interest for a reduced number of elements.

**Optimization procedures.** The current mesh is optimized by means of local modification tools. Two local techniques can be used (see Chapter 18). In the first, the point positions are maintained and their connections are affected. In the other, the connections are unchanged and the node locations are improved.

By using one or the other of these techniques, we can use local optimization operators that allow to suppress some edges (by merging their endpoints), to swap (in two dimensions) an edge or to swap some faces (this is the generalized swapping procedure in three dimensions).

A subtle use of these ingredients has proved to optimize a mesh with some degree of efficiency. Notice again that this optimization phase is common to all methods resulting in simplicial meshes (Chapters 5 and 7).

### 6.1.9 Practical issues

As is now clear, any advancing-front method includes a large number of heuristics.

**About basic algorithms.** A precise examination of the general scheme indicates that there are three points that must be carefully handled. They are concerned with the front management (initialization and updating), the visit of a given neighborhood of the front entity under examination (to collect the neighboring entities and find the corresponding sizing information) and the validation checks, basically based on intersection checks. They need to find the neighboring entities and, from a numerical point of view, to minimize the necessary CPU cost while guaranteeing valid results.

- Front management and selection of a front entity.

The front is a list of edges (faces) that is updated at each step by inserting or removing some edges (faces). Its management is then made easier by using a data structure well-suited to such operations (structures like a table, a dynamic linked list, see Chapter 2).

Depending on the strategy used, choosing a front entity is related to a metric type criterion (thus a sort) or a topological style criterion (neighborhood). In the first case, the criterion (length, surface area, etc.) must be considered when dealing with the front which is easy using a heap or a tree type structure.

In the second case, an easy access to the neighbors (edges or faces) of a given front entity is rather useful and we return to the notion of a neighborhood space that allows for this. Notice, in three dimensions, that the initial front is a triangular mesh whose vicinity relationships (in terms of edge adjacencies) are easy to obtain (Chapter 2).

- Neighborhood searching.

Already mentioned in the previous paragraph, the search for the entities (in the mesh and not only in the front) in some neighborhood of a given entity $f$ is necessary in order to find the points and validate the elements resulting from the connection of $f$ with these points. As a consequence, data structures like a tree (quadtree/octree) or a uniform grid (in view of *bucket sorting*) allow easy access to the entities close (in terms of distance) to the given entity (the edges (resp. faces) in the mesh are encoded in this structure to make this searching task easy).

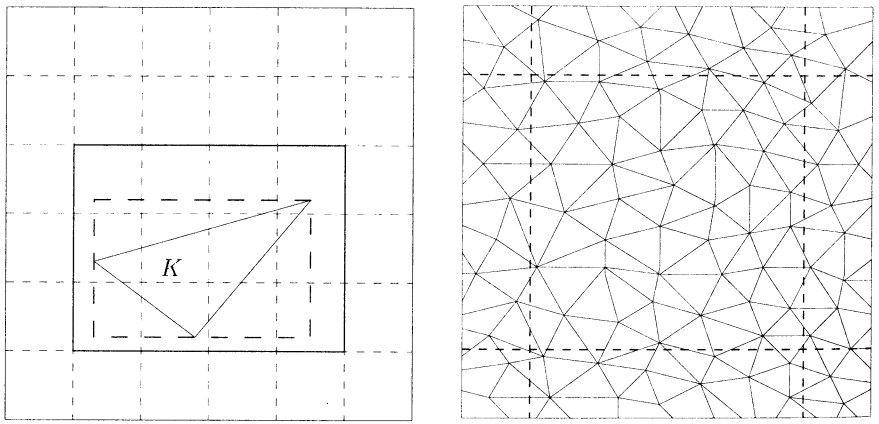



Figure 6.9: *Illustration of how to store and to retrieve a face in a grid. Range searching. Left-hand side : face $K$ is coded in each cell it intersects. Right-hand side : a common problem with a regular grid, the size of the grid cell is not compatible with the size of faces $K_i$.*



- Remarks about intersection checks.

The above neighborhood space (or any equivalent structure) is the key support for fast access to the entities close to the region under analysis. Then, the use of filters makes it possible to quickly reject the entities that are clearly not relevant (in terms of intersection). Indeed, it is obvious that a segment $[AB]$ such that $x_A < x_B$ does not intersect the segment $[CD]$ such that $x_C < x_D$ if, for instance, $x_C > x_B$. Thus, the intersection checks related to some adequate enclosing boxes show whether the corresponding entities may intersect or not. In the first case, an exact intersection check is made to give the decision, otherwise a rapid reject leads to a reduction in the cost.

Intersection checks and tests are concerned with the following pairs :

- box - box,
- edge - edge,
- edge - face,
- point (inside an) - element.

**Remark 6.11** *A case where there is not really an intersection but where the two entities in comparison are rather close to each other one must, in general, be considered as an intersection case. In fact, constructing an element in such a configuration will lead, for instance, to defining some edges very close to this element and then a rather small yet unmeshed space will be formed which will furthermore be rather tedious to fill up.*

**Convergence issues.** In two dimensions, a theoretical convergence result holds (Theorem (6.1)). From a practical point of view, the only trouble that is expected is when two fronts meet and the edge sizes are rather different. This potential drawback can be avoided if the sizes are globally considered which means that such a situation is avoided.

In three dimensions, the convergence results from a strategy which tends to minimize the number of impossible cases and, in the case of such a pattern, a procedure able to remove some elements. This consists in going backward so as to modify the local context and, in this way, remove the bottleneck.

**Memory resources and data structures.** Necessary memory resources correspond to the data structure used to store the useful values (vertices, mesh elements, etc.) and to the implementation of the basic algorithms (front management, vicinity, etc.).

The main internal data structures must be suitable to contain the following information[6] :

[6] their internal organization depends on the programming language used.

- point coordinates,
- point sizes,
- the vertices of the front entities,
- element vertices,
- adjacent elements to a given element (in terms of edge or face adjacencies) and
- some other resources, *a priori* with a small memory requirements as compared with the previous ones (for instance to encode the neighborhood space).

## 6.2 Governed advancing-front method

Constructing meshes that satisfy some specific properties (variable mesh density from one region to another, *i.e.*, variable element sizes) is required in numerous applications. The construction scheme leads then to using a governed mesh generation method where the desired control is specified as described below.

The specification of the desired element-size distribution function is indeed the desired information. Notice that this data, common to most mesh generation algorithms (see, for instance, Chapters 5 and 7) can be rather delicate to provide. The earliest approaches used a grid or a background mesh whose elements encode the desired sizing values. This data is then known everywhere in the domain. In fact, the purpose of this grid (mesh) was to provide a covering-up of the domain with which a continuous element-size distribution function is associated. This is exactly how a control space can be defined (cf. Chapter 1). In terms of the spatial aspect, the control structure may be defined in various way : using a regular grid, a quad- and octree decomposition or a background mesh (usual mesh or a mesh with no internal point, for instance, as seen in Chapter 7). In the coming sections, we briefly recall what a control space is.

### 6.2.1 Control space

Let us consider a background mesh of the domain as a control space[7]. The element-size function is known at the vertices of this background mesh. From the discrete size specification map, a continuous size function can be obtained using an interpolation scheme.

**Direct use of a background mesh.** Let $P$ be an arbitrary point in the domain, the size $h(P)$ at $P$ is computed as the $P_1$-interpolate of the sizes $h(P_i)$, $i = 1,..d$ ($d$ being the space dimension) at the vertices $P_i$ of the mesh element containing $P$ (cf. Figure 6.10). Schematically, the sizes $h(P_i)$ are computed as follows :

[7]For instance, a background mesh can be obtained using a Delaunay-based mesh generation algorithm, after the insertion of the boundary points and the boundary recovery (cf. Chapter 7)



- For all grid vertices $P_i$
  - find the element enclosing the cell vertices $P_i$,
  - compute $h(P_i)$ as the $P_1$-interpolate of the vertices of the element.

**Using a grid built on the background mesh.** Again, let $P$ be an arbitrary point in the domain (cf. Figure 6.10). We are given a uniform grid and we note $P_j$ its vertices. Schematically, the size value $h(P)$ is found after two steps. At first :

- For a given point $P_j$, a grid corner :
  - find the element within which $P_j$ falls,
  - compute $h(P_j)$ as the $P^1$-interpolate of the sizes $h(P_i)$ at the vertices $P_i$ of this element,

which reduces to using the above algorithm (so as to equip the grid corners with a size value). Then, for a given point $P$ :

- find the grid box within which $P$ falls,
- compute $h(P)$ as the $Q^1$-interpolate of the sizes $h(P_j)$ at the corners $P_j$ of this cell.

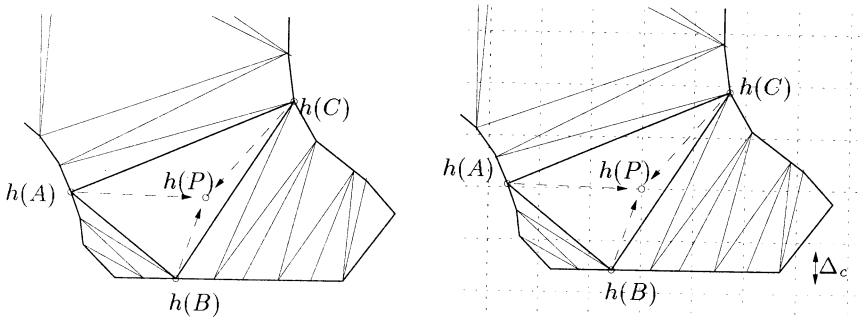



Figure 6.10: *Construction of the control space. Left : The size $h(P)$ at $P$ is obtained by interpolation between the sizes $h(A)$, $h(B)$ and $h(C)$ at $A$, $B$ and $C$, respectively. Right : a regular grid is combined with the background mesh, which can be discarded later.*

**Remark 6.12** *In such an approach, the size of the grid cells must be in accordance with the size of the elements in the background mesh. If not, a filtering effect can arise leading to a lack (dilution) of information, cf. Chapter 5.*

**Remark 6.13** *Notice also that the grid can be used as a neighborhood space only. For a given point $P$, one finds the cell enclosing it. This cell acts as a* pointer *on an vertex in the background mesh, thus making it easy to quickly find an element enclosing $P$. The algorithm given at first can be then used to evaluate the desired size at $P$.*

**Remark 6.14** *In the case where the background mesh is an "empty" mesh, these approaches result in a size distribution function strongly that is related to the boundary discretization. Indeed, the h's (size values) are "guested" from this sole data.*

In place of a uniform grid, one can use a decomposition like a PR-quadtree that gives *a priori* a way to have as the spatial part of the control space a covering up that is better adapted to the geometry (cf. Figure 6.11). The size of the tree cells is then related to the values of the size function and to local information (source points, etc.) if any. Therefore, the control structure is finer in high curvature regions and coarser elsewhere, in specific, inside the domain (say, far away from its boundaries) [Shephard,Georges-1991], [Rassineux-1995]. Thus, the above filtering effect is avoided to some extent. Other methods for size prescriptions can be found in [Löhner-1996b] which combine a general aspect and finer information.

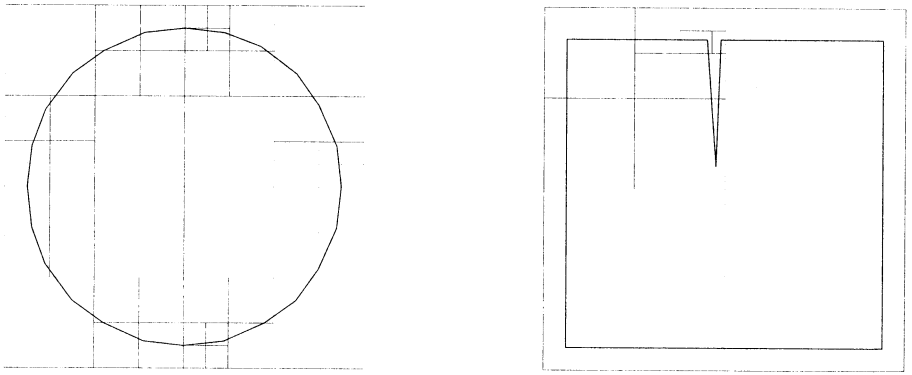


Figure 6.11: *Computing an implicit size map using a quadtree type decomposition. Left-hand side, the cell sizes increase as the distance from the boundary becomes greater. Right-hand side, The cell sizes is related to the domain geometry. The small cells make it possible to separate two entities belonging to two close boundary sides.*

### 6.2.2 Field point creation

Once the control space is ready, it is possible to develop a robust and flexible procedure for the field point creation designed in accordance with the information encoded in this space. Recall that for a given front entity, the goal is to create an optimal point such that the resulting element is well-shaped and its size conforms to the desired size. One can see (cf. Chapter 1) that such conditions can be satisfied if the edges of the optimal element are unit edge (close to one, in terms of length) with respect to the size map. A mesh whose edges are unit edges is said to be a *unit mesh.*



**Edge length.** Let $AB$ be a mesh edge, let $h(t)$ be the size specification along $AB$ such that $h(0) = h(A)$ and $h(1) = h(B)$, the normalized length of $AB$, denoted $l_{AB}$, with respect to $h(t)$ has been defined as (cf. Chapters 1 and 10)

$$l_{AB} = d_{AB} \int_0^1 \frac{1}{h(t)}\, dt\,, \tag{6.7}$$

where $d_{AB}$ stands for the Euclidean distance between point $A$ and point $B$. By definition[8], the edge $AB$ is said to have a length conforming to the size specification if :

$$\frac{1}{\sqrt{2}} \leq l_{AB} \leq \sqrt{2}\,, \tag{6.8}$$

and the goal is then naturally to create an optimal element (based on the front entity and the optimal candidate point) having all its edges conforming to the desired size.

**Optimal point creation.** The optimal point creation strategy is very similar to the classical case. In addition to the quality check, the algorithm attempts to create unit edge lengths for this element. In practice, an iterative algorithm is designed (both in two and three dimensions), which can be summarized as follows.

Consider a front face $ABC$ in three dimensions, let $\overrightarrow{n_{ABC}}$ be the normal to $ABC$ and let $G$ be the centroid of the face $ABC$,

1. Compute the optimal point location $P_{opt}$ (via the standard procedure) as

$$\overrightarrow{GP_{opt}} = \alpha h(G).\overrightarrow{n_{ABC}}\,,$$

   where $\alpha$ is a normalization coefficient leading to a regular tetrahedron. Set $it = 1$, the number of iterations.

2. Query the desired size $h(P_{opt})$ at $P_{opt}$ via the control space (an interpolation scheme can be used to find $h(P)$ based on the size specifications at the vertices of the control space element enclosing $P_{opt}$).

3. Compute the normalized edge lengths $l_{AP_{opt}}$, $l_{BP_{opt}}$ and $l_{CP_{opt}}$ of the optimal element.

4. Compute the locations of the *pseudo-optimal* points $P'_A$, $P'_C$, $P'_C$ along each edge $AP_{opt}$, $BP_{opt}$ and $CP_{opt}$ such that $l_{AP'_A} = 1$, $l_{BP'_B} = 1$, $l_{CP'_C} = 1$ (according to Relation (6.8)).

5. Compute $P'_{opt}$ the weighted barycenter of points $P'_A$, $P'_B$, $P'_C$.

[8] The coefficient $\sqrt{2}$ is related to the fact that an edge can be split if the lengths of the two sub-edges minimize the error distance to the unit length as compared with the length of the initial edge.

6. Move $P_{opt}$ towards point $P'_{opt}$, using a relaxation coefficient $\omega$ (cf. Figure 6.12),

$$P'_{opt} = P_{opt} + \omega \overrightarrow{P_{opt} P'_{opt}}.$$

7. Set $P_{opt} = P'_{opt}$, $it = it + 1$.

8. If $it < itmax$[9] then return to Step 2.

**Remark 6.15** *The procedure converges rapidly (practically $it < 5$) toward an optimal point $P_{opt}$. The purpose of using an iterative approach is to take into account the potential gradation shocks ahead of the front face (for instance, when merging two fronts of widely differing sizes).*

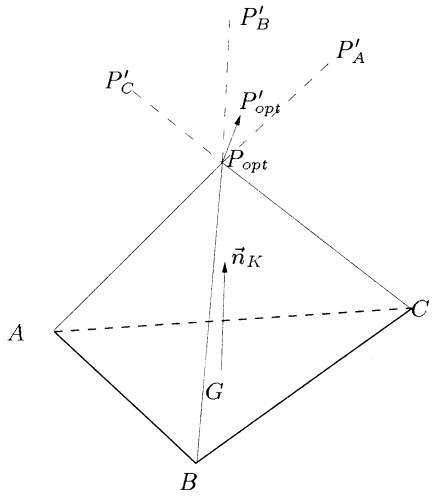



Figure 6.12: *Optimal point $P_{opt}$ creation, given a front face ABC the resulting tetrahedron K is considered optimal as it has conforming edge lengths.*

**Remark 6.16** *The field point creation algorithm must also check the shape quality of the new element. In fact, the method tends to create conforming edges with respect to the control space. In three dimensions however, this condition is not sufficient (e.g., the volume of a* sliver *is close to 0, even though all edge lengths are conforming, see Chapter 18).*

### 6.2.3 Optimization

As seen in the classical mesh generation problem and *a fortiori* in a governed case, it is usually useful to optimize the mesh resulting from the advancing-front governed algorithm. The goal is to produce the best regular element possible in accordance with the control space. Notice that the assumption of optimal field

[9] Actually, $itmax = 5$ seems adequate in most cases.



point generation has governed the entire procedure. Nevertheless, a two-stage optimization is applied to remove the worst elements in the mesh. This consists of improving the element quality (the shape as well as the size quality) using local topological mesh modifications (cf. Chapter 17, for more details). It appears efficient to optimize at first the size criterion and then to turn to the shape aspect (considering both criteria is probably elegant but tedious to do and is unlikely to be more efficient).

## 6.3 Application examples

In this section, several application examples of meshes in two and three dimensions, obtained using classical as well as governed advancing-front methods are proposed. Figures 6.13 and 6.14 illustrate a few meshes in two and three dimensions and Table 6.2 reports some characteristic values related to these examples.

| - | $np$ | $ne$ | $\mathcal{Q}_{th}$ | $\mathcal{Q}_M$ | $1-2$ | $t$ (sec) |
|---|---|---|---|---|---|---|
| case 1 | 1,955 | 7,254 | 10.2 | 14.3 | 75 | 11 |
| case 2 | 4,028 | 15,205 | 4.5 | 1.6 | 83 | 16 |
| case 3 | 8,125 | 36,517 | 7.7 | 9.1 | 84 | 47 |
| case 4 | 21,942 | 95,740 | 4. | 5.3 | 91 | 139 |
| case 5 | 33,373 | 170,330 | 3.2 | 7.7 | 93 | 212 |

Table 6.2: Statistics related to the selected meshes (the boundary discretization is the sole data used to generate the isotropic meshes).

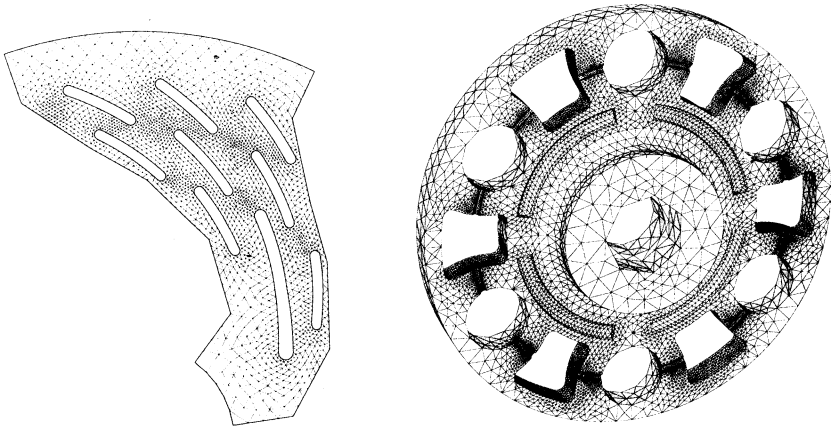


Figure 6.13: *Two- and three-dimensional advancing-front meshes (left : data courtesy of TELMA, right : data courtesy of ANSYS, Corp.).*

The statistics relative to the different meshes are provided in Table 6.2, where $np$ and $ne$ represent the number of mesh vertices and mesh elements respectively,

$\mathcal{Q}_M$ denotes the shape quality of the worst-shaped element in the mesh, after the optimization stage. In two dimensions, the mesh quality should be close to one, regardless of the domain boundary discretization[10]. In three dimensions however, the mesh quality must be compared with $\mathcal{Q}_{th}$ which represents the quality of the best element that can be theoretically created based on the worst face of the boundary mesh. The row $1-2$ represents the percentage of elements having a quality ranging between 1 and 2 (according to Relation (6.3)) and $t$ is the CPU time (DEC ALPHA 2100/500 workstation) required to complete the final mesh (including the i/o).

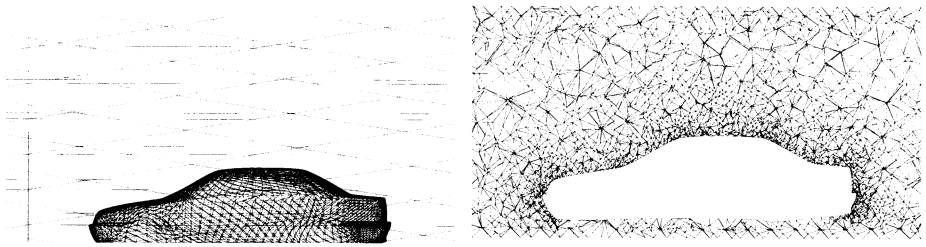


Figure 6.14: *Three-dimensional mesh of a car used for computational fluid dynamics (CFD) computations (data courtesy PSA). Left : computational domain (boundary mesh, local enlargement). Right : mesh in the vicinity of the car (section).*

In addition, Table 6.3 is given so as to appreciate the efficiency and the scalability of the advancing-front algorithm, $v$ represents the number of elements created per minute.

| - | $np$ | $ne$ | $t$ (sec., DEC 2100/500) | $v$ |
|---|---|---|---|---|
| case 1 | 20,674 | 107,085 | 192 | 33,464 |
| case 2 | 36,856 | 194,663 | 422 | 27.677 |
| case 3 | 72,861 | 421,444 | 807 | 31,334 |
| case 4 | 220,177 | 1,234,457 | 1,682 | 44,035 |
| case 5 | 327,092 | 1,832,038 | 2,153 | 51,055 |

Table 6.3: Efficiency and scalability of the advancing-front meshing algorithm.

The results show that the speed-up ranges from 25,000 to around 50,000 elements per minute. This difference can be explained by two factors :

- the size of the underlying data structure used as neighborhood space (here a regular grid has been used) which may be inappropriate with regard to the element sizes (for instance in case 2), or
- the small part devoted to intersection checking in some examples (for instance, when the computational domain is geometrically simple as in case 5 or for domains with high volume to surface area).

[10] provided the optimal point belongs to the domain !



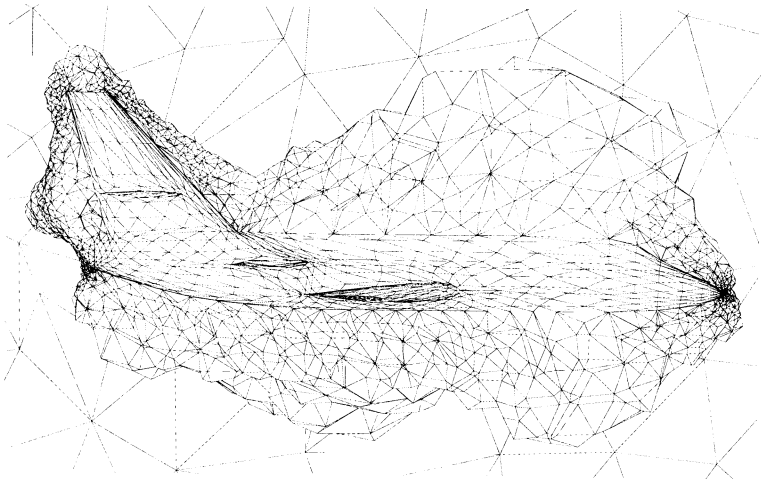


Figure 6.15: *Front evolution during the construction (cut through a three dimensional mesh after a few iterations).*

To conclude, notice that the theoretical complexity of any advancing-front method is not so obvious to investigate. We have already observed that the computational cost of the intersection checks is proportional to the number of entities close (in some sense) to a given entity and, for this reason, does not *a priori* depend on the mesh size. Also, we have pointed out that the geometrical aspect of the domain is not a parameter in terms of cost (since, after a few fronts, the remaining region is an arbitrarily shaped region, except for very specific cases where, in addition, a regular gradation is possible (and reached !).

Nevertheless, for instance in [Peraire *et al.* 1988] and [Bonet,Peraire-1991], a complexity in $\mathcal{O}(n\log(n))$ is reported where $n$ is the number of elements in the mesh. In these references, the factor in $\log(n)$ results from the use of a tree structure (an alternated digital tree). However, if the generation of a single element could be made in a constant time, then the triangulation of the whole domain will be linear in time. [Krysl-1996] also showed that a linear complexity is obtained in two specific problems, namely front management (using a hashing technique) and the search for the entities neighboring a given entity (using a neighborhood space, Chapter 1).

## 6.4 Combined approaches

The main drawback of the (standard or governed) advancing-front approach mainly relates to its efficiency (especially when compared to a Delaunay method). The intersection checking routine is a relatively expensive procedure for ensuring the acceptability of a new element.

In addition, advancing-front approaches often have problems with robustness, mainly because of the lack of a correct definition of the region of interest (*i.e.*, the neighboring elements of a mesh entity)

An approach combining the Delaunay criterion (Chapters 1 and 7) and the advancing-front technique is a solution that has proved especially useful to overcome some difficulties. Using it, it is possible to construct several elements at a time and, moreover, to allow the proper merging of two fronts of different length scales, which represents a classical type of failure in the standard algorithm (especially in three dimensions).

### 6.4.1 Advancing-front Delaunay approach

The Delaunay based methods are regarded as faster than the advancing-front style methods mainly because the mesh is constructed point by point and that each point allows for the creation of several elements. From a local point of view, the number of elements constructed from a point is a function of the local situations, it runs from a few elements to several tens of elements (see, in Chapter 7, the notion of a cavity). However, since the elements are constructed and removed during the algorithm, a more precise analysis is needed to make sure that the global efficiency is related to this point.

The idea suggested by [Mavriplis-1992] leads to defining an advancing-front strategy that automatically locates the new points and constructs as elements those which conform to the Delaunay criterion so as to achieve the efficiency of these methods[11]. The algorithm mainly consists in identifying the three following configurations :

- some neighborhood of $P_{opt}$ is empty (no point is included inside this region) or not,
- this zone is empty but there are some circles (spheres) circumscribing some elements of the current mesh that enclose the selected candidate point,
- this zone is empty and such circles (spheres) do not exist.

Then, for each of these patterns, the collected information allows us to anticipate the nature of the local context.

**Remark 6.17** *The inverse coupling, of Delaunay advancing-front type, is also a possible solution. The Delaunay method is used to connect the vertices while an advancing-front strategy is used to locate the field points, see [Frey* et al. *1998] and Chapter 7.*

**Computational issues.** The efficiency of the combined method is related to the fact that additional information is available, provided by the Delaunay criterion. In particular, this means that we automatically have certain knowledge of the neighborhood of the considered area, which, in the classical advancing-front approach, can only be obtained by a tremendous computational effort and through relatively complex data structures. The robustness of the method is also improved following this approach, as the Delaunay criterion makes it possible to anticipate

[11] This point of view is also suggested in [Merriam-1991], [Muller *et al.* 1992] and [Rebay-1993].



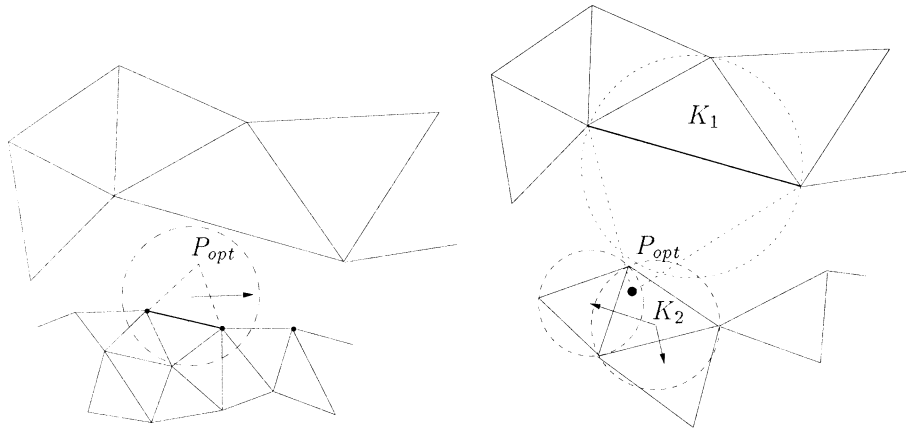



Figure 6.16: *Insertion of a new point $P_{opt}$ and element creation. Left-hand side : classical approach, failure to locate the end points of a large edge (the face is not coded in all cells of the background space). Right-hand side : the circumcircles of $K_1$ and $K_2$ are intersected, neighbors of $K_2$ are searched and one is found to have its circumcenter intersected.*

and to avoid potential collisions between two fronts, to use a criterion to decide the deletion of conflicting elements and, finally, to handle and create several elements simultaneously. Moreover, if the internal points are well located, the Delaunay criterion offers a certain guarantee of the resulting element (shape) quality.

### 6.4.2 Advancing-front with preplaced interior points

As the advancing-front methods work from the boundary towards the interior of the domain, special care must be taken regarding the generation of mesh elements in the vicinity of the boundary. However, for domains with high volume to surface area, the internal point creation stage can be improved.

The approach suggested by [Rassineux-1997] consists in combining the advancing-front method, used to create internal points close to the boundary, with a quadtree-octree based spatial decomposition method to generate the internal elements (cf. Chapter 5). More precisely, the general scheme of this technique consists in :

- building the spatial decomposition of the domain using a tree structure (quadtree in two dimensions and octree in three dimensions),
- marking the internal cells and the boundary cells,
- meshing the internal cells using predefined patterns (*templates*)
- meshing the boundary cells using an advancing-front approach.

An optimization step is then carried out to improve element (shape) quality, especially in the transition zones, between the regions where different meshing techniques have been used.

Notice that in this approach, the mesh gradation is governed by the tree balancing rule (element sizes are directly related to cell sizes). Moreover, the cell corners give the resulting mesh vertices, thus providing the final mesh with a certain rigidity.

### 6.4.3 Hybrid approaches

It is often desirable to generate stretched meshes for viscous flow computations, particularly in boundary layers and wake regions. This can be achieved using a hybrid technique which consists of creating a structured mesh in the boundary layers and wake regions combined with an unstructured mesh elsewhere.

The structured mesh consists of prisms (that can later be subdivided into tetrahedra, if necessary) obtained by offsetting the surface triangles along the normal directions at the vertices, thus defining a sort of homothetic surface to the initial one. The thickness of the layers is related to the vertex normals, the distance between the nodes being related to the Reynolds number and to the other parameters of the given problem. However, the stretching of the elements is that of the initial mesh.

The unstructured mesh is then created using a classical advancing-front technique. The internal points are calculated so as to generate well-shaped elements, that are as regular as possible. The mesh gradation is controlled so as to achieve a smooth transition of element sizes between the structured and unstructured meshes.

The tedious part of this approach concerns the proper definition of the boundary layer features and other regions where the elements have to be stretched, as well as the definition of the parameters (thickness, number of nodes, node stretching function). It is also necessary to ensure a smooth transition between the structured and unstructured meshes.

The kind of approach has been primarily studied in computational fluid dynamic applications, for instance by [Muller *et al.* 1992], [Johnston,Sullivan-1993], [Pirzadeh-1994], [Hassan *et al.* 1996] and [Löhner *et al.* 1992].

## 6.5 Extensions

In this section, we mention some peculiar applications of the advancing-front method, that allow it to increase its potential field of application to non-conventional meshing problems. In particular, we mention anisotropic mesh generation, which is then extended to surface mesh generation. Finally, we comment briefly on the use of an advancing-front technique within an adaptive mesh generation scheme.

### 6.5.1 Anisotropic mesh generation

So far, the general scheme of the advancing-front method has been designed to handle isotropic meshes. However, most of the proposed approaches have been conceived for isotropic meshing purposes and are not able to handle high-aspect



ratio element generation (such as those involved in Navier-Stokes computations, for example).

The general classical scheme of the advancing-front method can be modified to generate such elements. More precisely, the internal point creation stage is adapted to the creation of anisotropic elements.

**Edge length in an anisotropic metric.** We introduce here a more general formulation of the normalized edge length calculation, cf. Relation (6.7). Let $AB$ be an edge, defined using a parameterization such that $AB(t) = A + t\overrightarrow{AB}$, $t \in [0,1]$ and let $\mathcal{M}(M(t))$ be the $2 \times 2$ matrix (in two dimensions) defined by :

$$\mathcal{M}(M(t)) = \begin{pmatrix} \frac{1}{h^2(t)} & 0 \\ 0 & \frac{1}{h^2(t)} \end{pmatrix}, \tag{6.9}$$

(resp. a $3 \times 3$ matrix in three dimensions), where $h(t)$ denotes the expected size at point $M(t)$. Then, the length $l_{\mathcal{M}}(AB)$ of the segment $AB$ in the metric corresponding to the matrix $\mathcal{M}$ is defined by (Chapter 10) :

$$l_{\mathcal{M}}(AB) = \int_0^1 \sqrt{{}^t\overrightarrow{AB}\,\mathcal{M}(A + t\,\overrightarrow{AB})\,\overrightarrow{AB}}\,dt\,. \tag{6.10}$$

**Optimal point creation.** The optimal point creation based on a given front item in order to create an anisotropic element is carried out by replacing the edge length computation in the classical scheme by an edge length calculation in the anisotropic metric. More precisely, we use the previous definition of the normalized edge length, replacing the matrix $\mathcal{M}(M(t))$ by the matrix (in two dimensions) :

$${}^t\mathcal{R}(t) \begin{pmatrix} \frac{1}{h_1^2(t)} & 0 \\ 0 & \frac{1}{h_2^2(t)} \end{pmatrix} \mathcal{R}(t)\,. \tag{6.11}$$

The matrix $\mathcal{R}(t)$ makes it possible to specify two privileged directions at point $M(t)$ and $h_1(t)$ (resp. $h_2(t)$) indicates the desired size in the first (resp. second) direction at this point. The adjusting strategy of the optimal point location is then identical to that of the governed isotropic case.

**Optimizations.** As for the governed isotropic case, an optimization stage is carried out to improve the size and the shape qualities of the resulting anisotropic mesh. This stage is based on local topological and geometrical modifications.

### 6.5.2 Surface mesh generation

The generation of finite element surface meshes is a topic that has received a lot of attention over the last few years. The reason is manifold. At first, surface meshes

are important because of their effect on the accuracy of the numerical solutions (partly related to the boundary conditions) and the convergence of the computational scheme in numerical simulations based on finite (or boundary) element methods. On the other hand, three-dimensional meshing techniques often rely on surface (boundary) meshes. This is especially the case for quadtree-octree (Chapter 5), advancing-front (see above) and Delaunay-type methods (Chapter 7). Two approaches, direct and indirect, can be envisaged for surface meshing.

**Direct surface meshing.** This approach consists of applying a governed meshing technique (here the advancing-front) directly to the body of the surface, without using any kind of mapping related to any arbitrary parameterization. The mesh element sizes and shapes can be controlled by monitoring the surface variations. The approach proceeds by first discretizing the curves representing the surface boundary, then it triangulates the surface. The reader is referred to [Nakahashi,Sharov-1995] and [Chan,Anatasiou-1997], among others, for more details.

The general scheme of surface meshing using a direct approach is based on the same steps as the classical advancing-front technique. The main difference lies in the iterative algorithm used to find an optimal point location given a front edge.

More precisely, given an edge $AB$, the optimal point $P$ is computed to construct a triangle determined by an angle $\alpha$ between the stretching direction and the tangent at the midpoint $M$ of $AB$ (or using an average tangent plane and setting $\alpha$ to zero). The point $P$ is then projected onto the surface, its location being obtained through a query to a geometric modeler [Steger,Sorenson-1980]. Candidate points are identified as those which lie in the circle of center at $P$ and radius $\kappa \times \delta$, where $\kappa$ is a positive coefficient (e.g. 0.7 according to [Nakahashi,Sharov-1995]). The size of the triangle is locally adapted to the surface curvature (see Chapter 15). An optimization stage is required to globally improve the element shapes (cf. Chapter 19).

Notice that the edge lengths are calculated based on straight segments, however points are further moved onto the true surface, the computed length can thus be quite different from the true edge length. It is therefore quite a tedious approach to implement, especially when the surface contains large curvature variations.

**Parametric surface meshing.** The indirect approaches essentially concern parametric surface meshing. The generation of a mesh for such a surface is obtained through a parametric space, which in fact is equivalent to a purely two-dimensional problem (although it uses surface related information). Let $\Omega$ be a domain of $\mathbb{R}^2$ and$\sigma$ a sufficiently smooth function, then the surface $\Sigma$ defined by the application $\sigma : \Omega \longrightarrow \mathbb{R}^3$, $(u,v) \longmapsto \sigma(u,v)$ can be meshed using a two-dimensional meshing technique in $\Omega$, then mapping this mesh via $\sigma$ onto $\mathbb{R}^3$. To this end, one has to use a mesh generation technique with anisotropic features, the metric map being based on the intrinsic properties of the surface. The general scheme of this approach is based on three successive steps : the parameterization of curves and surfaces, the unstructured mesh generation in $\Omega$ and, finally, the mapping of this mesh from the parametric space to the real space so as to obtain the final mesh. An optimization stage can complete the mesh generation process.



Several authors have used an advancing-front for parametric surface meshing (see, for instance, [Peraire *et al.* 1987], [Samareh-Abolhassani,Stewart-1994], and also [Rypl,Krysl-1994] [Möller,Hansbo-1995], [Löhner-1996b] and [Marcum-1996]). Notice that the intersection calculations can be directly performed in the parametric space [Frykestig-1994].

### 6.5.3 Mesh adaptation

The basic idea of mesh adaptation is to improve the accuracy of the numerical solutions as well as to reduce the computational cost of the numerical operations. It requires a quasi-optimal node distribution at each iteration. The mesh adaptation problem will be discussed in greater detail in Chapter 21.

The mesh adaptation is based on a governed mesh generation technique, the size map is supplied by an *a posteriori* error estimate. The governed advancing-front strategy can be applied, almost without modification, to the creation of adapted meshes[12]

## A brief conclusion

Advancing-front type methods are efficient, robust and polyvalent. They allow classical or governed meshes (*i.e.,* conforming a prescribed size map) to be constructed. They are also of interest in the context of adaptation problems (Chapter 21) and, in addition, allow surface meshes to be constructed (taking advantage of the anisotropic features).

[12] The adaptation is based on the regeneration of the whole mesh and not on a mesh optimization.