result preserves the initial topology in a plausible way. It might well be imagined therefore that repairing a surface is, at present, a hot topic.

We must however mention that work regarding the automation of such processes, [Barequet *et al.* 1998], is being carried out. The idea is then to infer a possible topology of the surface and then to modify the patches accordingly.

Chapter 16

Meshing implicit curves and surfaces

Introduction

In practical terms and notably in applications related to geometric modeling and graphical visualization, the usual representation of curves and surfaces is the *parametric* one (cf. Chapters 12 and 13). Nevertheless, other representations exist and are employed to some extent depending on the applications envisaged. An *explicit* representation is based on functions of the form $z = f(x, y)$. This approach is quite limited in practice, as the surfaces defined in this way are usually rather "rudimentary" or do not correspond to concrete cases. A third approach consists in defining a surface as the set of points (x, y, z) in $\mathbb{R}^3$, that are solutions of an equation of the type $f(x, y, z) = 0$. The study of such surfaces, called *implicit surfaces* is the subject of this chapter.

Interest in implicit curves and surfaces has increased over the last few years, notably due to the emergence of discrete (sampled) data for modeling computational domains. *Discrete geometry* attempts to transpose the results of classical (affine and differential) geometry to the discrete field. However, as pointed out by [Hoffmann-1993], the application field of implicit functions seems to remain largely underestimated.

Given an implicit curve or surface representing the boundary of a computational domain, we focus here on the problem of meshing this boundary. In the first section, we recall some basic definitions and properties of implicit functions. Then, we deal with the mesh generation of implicitly defined planar curves. In the third section, we indicate how the meshing techniques for curves can be extended to the meshing of implicit surfaces. Application examples are shown at the end of this section to illustrate the various meshing techniques for these surfaces. The last section briefly presents the basic principles of constructive geometry for implicit domains before dealing with mesh generation of implicit domains (whose

boundaries are implicit curves or surfaces), presented here as a natural extension of meshing techniques for curves and surfaces.

16.1 Review of implicit functions

In this section, we recall the main definitions and properties related to implicit planar curves and surfaces. In particular, we see how the main results of differential geometry introduced in Chapter 11 are involved.

16.1.1 A preliminary remark

As meshing techniques suitable for parametric curves and surfaces have been extensively developed, as seen before, one might ask if it were not possible to convert an implicit function into one (or more) equivalent parametric representation(s), so as to find a known problem (Chapter 14).

Notice first that any parametric rational curve or surface assumes an implicit form, the way of obtaining it being widely known (see [Sederberg-1983], among others). However, obtaining the parametric form corresponding to a given implicit function is not trivial, for many reasons that we need not concern ourselves with here [Hoffmann-1993]. In practical terms, this old problem[1] turns out to be extremely difficult to solve in the general case, which justifies the development of direct meshing methods for implicit curves and surfaces.

16.1.2 Implicit planar curves

Let $f : \Omega \longrightarrow E$ be a function of class C^k ($k \geq 1$) on an open set Ω of the affine plane E (here $E = \mathbb{R}^2$). The set Γ of points $M(x, y) \in \Omega$ such that $f(M) = f(x, y) = 0$ is the so-called *implicit curve* defined by $f = 0$:

$$\Gamma = \{M(x, y) \in \Omega\,,\ f(M) = f((x, y) = 0\}\,.$$

Definition 16.1 *A point $M(x, y)$ of the curve Γ is said to be an* ordinary *(or* regular*) point if it is such that :*

$$\nabla f(M) = {}^t\left(\frac{\partial f(M)}{\partial x}, \frac{\partial f(M)}{\partial y}\right) \neq 0\,,$$

where the operator ∇ denotes the gradient. Thus, the curve Γ is said to be regular *if $\nabla f(M) \neq 0$ for each point M. The point M is said to be* singular *if $\nabla f(M) = 0$.*

Notice that the inverse image of a value k of $\mathbb{R}$ is the solution to the equation $f(M) = k$, at a point M of $\mathbb{R}^2$. Thus, the curve Γ, defined by $f = 0$, is $f^{-1}(0)$, the inverse image of 0. More generally, a curve Γ_k defined by $f(x, y) = k$ is called an *iso-value curve* (or *level curve*), k being the *level* of f.

The geometric interpretation of the theory of implicit functions makes it possible to deduce the following result :

[1] In the last century, [Salmon,1885] already proposed a way of getting rid of the parameters of parametric equations.

Theorem 16.1 (implicit functions) *Let M_0 be an ordinary point of the curve Γ, there exists a neighborhood $V(M_0)$ such that $\Gamma \cap V$ is the support of a parameterized arc Γ_0 of class C^k.*

Hence, it is important to bear in mind from this result that the implicit curve Γ admits local parameterizations at each of its regular points[2] Theorem (16.1) makes it possible to write in explicit form $y = y(x)$. Figure 16.1 illustrates a parameterization of Γ.

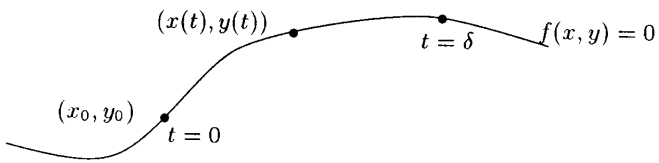

Figure 16.1: *A parameterization $x(t), y(t)$ of the implicit function $f(x, y) = 0$ passing through the ordinary point (x_0, y_0).*

Study of critical points. In the case where the implicit function theorem holds, the study of the extrema of f requires a preliminary study of its critical points.

Definition 16.2 *We say that a function f has a* local maximum *(resp.* local minimum*) at a point P, if there exists a neighborhood $\mathcal{V}$ of P such that :*

$$\forall M \in \mathcal{V}\,,\quad f(M) \leq f(P) \qquad (\textit{resp. } f(M) \geq f(P)\,)\,.$$

An extremum *is a maximum or a minimum.*

The curve Γ has a *critical point*, say $M(x, y)$, if :

$$\nabla f(M) = 0 \iff \frac{\partial f}{\partial x} = \frac{\partial f}{\partial y} = 0\,.$$

By extension, a *critical value* represents the value of the function at a critical point. The study of the Hessian H_f of f makes it possible to specify the nature of this critical point. More precisely, if $H_f(M) > 0$ we have a *minimum*, if $H_f(M) < 0$ we have a *maximum* and if $H_f(M) = 0$ we cannot conclude (we must then study the "sign" of the derivatives at order 3).

For an ordinary point $M(x, y)$ of an implicit curve Γ to be a *point of inflection*, its coordinates must satisfy the equation $H(x, y) = 0$, where the function H is defined by :

$$H = f'^2_y f''_{xx} - 2 f'_x f'_y f''_{xy} + f'^2_x f''_{yy}\,,$$

which can also be written as follows :

$$H = \begin{vmatrix} 0 & f'_x & f'_y \\ f'_x & f''_{xx} & f''_{xy} \\ f'_y & f''_{xy} & f''_{yy} \end{vmatrix}\,.$$

[2] From the numerical point of view, it is important to know that the value close to the regular values are regular as well.

Tangent vector. using the implicit function theorem, we deduce that the tangent at an ordinary point $M_0(x_0, y_0)$ to the parameterized arc Γ_0 is the line defined by the equation :

$$\langle \nabla f(M_0), \overrightarrow{MM_0} \rangle = 0 .$$

In fact, there exist two intervals I and J and an application φ of class C^k verifying $\varphi(x_0) = y_0$, such that the relations :

$$(x, y) \in I \times J \quad \text{and} \quad f(x, y) = 0$$

are equivalent to $x \in I$ and $y = \varphi(x)$. Thus, the set of corresponding points M is Γ_0, the arc of Cartesian equation $y = \varphi(x)$.

The tangent at M_0 to Γ_0 is the line of equation $y - y_0 = \varphi'(x_0)(x - x_0)$ and thus we have :

$$\varphi'(x_0) = -\frac{f'_x(x_o, y_0)}{f'_y(x_o, y_0)} .$$

Hence, the tangent is defined by :

$$(x - x_0) f'_x(x_0, y_0) + (y - y_0) f'_y(x_0, y_0) = 0 \tag{16.1}$$

Hence, we note $\vec{\tau}$ the unit tangent vector at an ordinary point M_0 to Γ.

Principal normal. The normal $\vec{\nu}$ to the curve Γ at a regular point $M_0(x_o, y_0)$ is parallel to the vector of coordinates $f'_x(x_0, y_0), f'_y(x_0, y_0)$, that is to the gradient vector ∇f. The vector

$$\vec{\nu}(M_0) = \frac{\nabla f(M_0)}{\|\nabla f(M_0)\|}$$

is called the *principal normal* to the curve at M_0 and verifies $\|\vec{\nu}(M_0)\| = 1$. Notice that, as expected, $\langle \vec{\tau}, \vec{\nu} \rangle = 0$, the two vectors being orthogonal.

Curvature. The calculation of the *local curvature* at a regular point $M_0(x_0, y_0)$ is based on the relation $\langle \vec{\tau}, \vec{\nu} \rangle = 0$. From a practical point of view, we again use the implicit functions theorem and we introduce the curvilinear abscissa s (considering this time the arc Γ_0) :

$$\frac{d(\langle \nabla f, \vec{\tau} \rangle)}{ds} = \left\langle \frac{d\nabla f}{ds}, \vec{\tau} \right\rangle + \left\langle \nabla f, \frac{d\vec{\tau}}{ds} \right\rangle = 0 ,$$

which will allow us to retrieve one of Frénet's formulas linking the local curvature at M_0 to the derivative of the tangent (Chapter 11) :

$$\frac{d\vec{\tau}}{ds} = C.\vec{\nu} = \frac{\nabla f}{\|\nabla f\|} . \tag{16.2}$$

using a similar approach, the other Frénet's formula would be written :

$$\frac{d\vec{\nu}}{ds} = -C.\vec{\tau} . \tag{16.3}$$

Distance from a point to a curve. In Chapter 14, we discussed the tedious problem of geometric mesh generation for (parametric) curves. In order for the mesh to follow the arc geometry, the maximal distance δ between the arc and the segment discretizing it must be bounded (*i.e.*, the length of the curve and the length of the chord are close to each other). Thus, if h is the length of the segment, it is desirable that, for a given value ε :

$$\delta < \varepsilon h .$$

One of the problems encountered consists of evaluating this distance δ. We will focus on calculating the Euclidean distance of a given point P to a planar curve Γ implicitly defined by $f = 0$.

If the function f is a polynomial, numerical techniques can be used to evaluate this distance [Kriegman,Ponce-1990]. However, in most cases, it is rather tricky to obtain an accurate answer to this question and usually a first order approximation of this distance is sufficient to decide whether or not to pursue the algorithm.

The distance d can be defined as the minimum of the distances from P to any other point Q of Γ :

$$d(P, \Gamma) = \min_{Q \in \Gamma} \|Q - P\| , \tag{16.4}$$

which leads to a minimization problem.

If P is a regular point, a Taylor series of f in the neighborhood of P makes it possible to write :

$$f(Q) = f(P) + \langle h, \nabla f(P) \rangle + \frac{1}{2}({}^t h H_f(P) h) + \dots ,$$

where $h = \overrightarrow{PQ}$ is sufficiently small to ensure the validity of this expansion. By truncating this expansion at the order 1, we have :

$$f(Q) = f(P) + \langle h, \nabla f(P) \rangle + \mathcal{O}(\|h\|^2) .$$

In practical terms, we simply consider the approximation :

$$f(Q) \approx f(P) + \langle h, \nabla f(P) \rangle .$$

The triangular inequality then makes it possible to write :

$$|f(P) + \langle h, \nabla f(P) \rangle| \geq |f(P)| - |\langle h, \nabla f(P) \rangle|$$

then, using Cauchy-Schwartz's inequality :

$$|f(Q)| \geq |f(P)| - \|h\| \, \|\nabla f(P)\| . \tag{16.5}$$

Finally, the distance $d(P, \Gamma)$, defined as the value of $\|h\|$ such that the right-hand term of the previous expression vanishes [Taubin-1992], is approached by the formula :

$$d(P, \Gamma) \approx \frac{|f(P)|}{\|\nabla f(P)\|} . \tag{16.6}$$

Practical aspects. In applications, to reduce the number of distance calculations, we can use the properties of the *lipschitz* functions. For such a function f, we have $|f(P) - f(Q)| \leq \lambda ||P - Q||$, for all P, Q, λ being a positive parameter characterizing f. In fact, Lipschitz's constant λ (the smallest λ satisfying the equation) defines the lower bound of the module of the derivative function. If f is a continuous function, then Lipschitz's constant is the maximal slope of the function, which is reached at one of the zeros of the second derivative of the function (*i.e.*, at a global minimum of f').

Hence, for a given point P, let us denote $Q \in f^{-1}(0)$ the point such that :

$$||P - Q|| = d(P, f^{-1}(0)) .$$

Then, we can write that :

$$|f(P)| < \lambda\, d(P, f^{-1}(0)) .$$

Hence, $\lambda^{-1} f(P)$ represents a distance bound for f (the sign is meaningful).

Remark 16.1 *This result makes it possible in practice to use a Newton's method (that locally converges at the order two) to find the coordinates of the root in only a few iterations using the formula :*

$$P^{k+1} = P^k - \frac{f(P^k)\nabla f(P^k)}{||\nabla f(P^k)||^2}$$

P^0 being a given starting point. We can thus search the first intersection between a given line and the function f.

We will go into more detail on how to use this result in the section related to the mesh generation of implicit planar curves.

16.1.3 Extension to implicit surfaces

We will now focus on implicit surfaces defined by a function $f(M) = f(x, y, z) = 0$. Most of the results on planar curves can be extended to the case of surfaces.

Normal, tangent. For a regular surface Σ, the vector ∇f defines a vector that is orthogonal to Σ. In fact, a regular surface is *orientable*. The unit normal vector to the surface at a regular point P is given by :

$$\vec{n}(P) = \frac{\nabla f(P)}{||\nabla f(P)||} . \tag{16.7}$$

Critical points. As for planar curves, a necessary condition to have a critical point is the following :

$$\nabla f = 0 \Longleftrightarrow \frac{\partial f}{\partial x} = \frac{\partial f}{\partial y} = \frac{\partial f}{\partial z} = 0 .$$

We introduce then the Hessian matrix :

$$H_f = \begin{bmatrix} f''_{xx} & f''_{xy} & f''_{xz} \\ f''_{yx} & f''_{yy} & f''_{yz} \\ f''_{zx} & f''_{zy} & f''_{zz} \end{bmatrix} , \tag{16.8}$$

which is a symmetric matrix if the function is of class C^2. The study of the determinant $Det(H_f)$ of the Hessian matrix makes it possible to conclude : if $Det(H_f) < 0$ we have a *saddle point*, if $Det(H_f) > 0$ we have an *extremum* (if $H_f > 0$ (resp. $H_f < 0$) it is a *minimum* (resp. *maximum*)) and if $Det(H_f) = 0$, we cannot conclude, we must then study the partial derivatives at the order 3.

Principal curvatures. To find the principal curvatures and the principal directions at a regular point $M_0(x_0, y_0)$, we use again the fact that $\langle \vec{\nu}, \vec{\tau} \rangle = 0$. Let $\vec{u}$ and $\vec{v}$ be two vectors forming an orthonormal basis of the tangent plane $\Pi(M_0)$ and let $\vec{\tau}$ be a unit tangent vector at M_0. We can write :

$$\vec{\tau} = \cos\theta \vec{u} + \sin\theta \vec{v} .$$

From the formula giving the curvature κ_τ in the direction $\vec{\tau}$:

$$\kappa_\tau = \frac{{}^t\vec{\tau}\, H_f\, \vec{\tau}}{||\nabla f||} ,$$

we can deduce the formulas of the principal curvatures κ_1 and κ_2 and the principal directions τ_1 et τ_2 (that are expressed in the basis $[\vec{u}, \vec{v}]$ [Monga,Benayoun-1995]) :

$$\kappa_1 = \frac{{}^t\vec{\tau}_1\, H_f\, \vec{\tau}_1}{||\nabla f||} , \qquad \kappa_2 = \frac{{}^t\vec{\tau}_2\, H_f\, \vec{\tau}_2}{||\nabla f||} . \tag{16.9}$$

Supplied with this theoretical background, we will now discuss the problem of meshing a domain defined by an implicit function. But, prior to focusing more closely on meshing techniques for implicit curves and surfaces, we will first introduce a general formulation of this problem which then makes it possible to find a general meshing scheme, in two and three dimensions.

16.2 Implicit function and meshing

We will first attempt to formulate the problem in a practical way, that is in view of the envisaged application, the geometric meshing of an implicit domain for a numerical simulation using finite element methods. For the sake of simplicity, we will limit ourselves to the two dimensional case only.

16.2.1 Problem statement

Let Γ be an implicitly defined curve, *i.e.*, the solution to an equation of the form $f(x, y) = 0$, f being a differentiable function admitting 0 as a regular value.

The problem of discretizing Γ can be seen as the search for a polygonal curve $\widetilde{\Gamma}$ of same topology as Γ that forms a sufficiently accurate approximation of Γ. This problem can be formally described using the following conditions (see [Velho *et al.* 1997]) :

- $\widetilde{\Gamma}$ is a piecewise linear approximation of Γ,
- there exists a homeomorphism[3] $h : \Gamma \longrightarrow \widetilde{\Gamma}$ such that :

$$\forall P \in \Gamma\,, \qquad d(P, h(P)) < \varepsilon\,, \tag{16.10}$$

where $\varepsilon > 0$ is a user-specified tolerance and d is the usual Euclidean distance in $\mathbb{R}^2$.

Remark 16.2 *The tolerance value ε corresponds to the quality of the geometric approximation of the curve geometry.*

In practice, we have seen that with a piecewise linear approximation, the lengths of the edges of the discretization are locally proportional to the radii of curvature (Chapter 14).

16.2.2 Geometric mesh

The problem we are interested in is to mesh an implicit curve so as to correctly approach its geometry. In other words, the metric to follow is of a purely geometric nature.

Desired properties and related problems. The first property desired is to assume a relative control over the curve, such that the length of the curve (the arc) and that of the chord locally approaching it (the underlying chord) are close. To this end, if h denotes the length of a segment and if δ measures the largest distance between this segment and the curve, then, for ε given, we want to have the following condition :

$$\delta \leq \varepsilon\, h\,.$$

Notice that with parametric curves, the length of the arc can be easily obtained using the curvilinear abscissa s. With implicit curves, such a feature does not exist (the implicit function theorem allowing only a local parameterization). One way of evaluating the distance between the segment and the curve consists in sampling the segment and in calculating the largest distance between the sampled points and Γ :

$$\delta = \max_{P_i \in I} d(P_i, \Gamma)\,,$$

where I represents here a small interval along the segment considered.

The second problem is related to the location of the points along the curve. From a given point M_0, one needs to find the series of points $M_i\,, i = 2, 3, \ldots$ such

[3] Thus ensuring that Γ and $\widetilde{\Gamma}$ have the same topology.

that the length of the segments $M_i M_{i+1}$ is compatible with the given accuracy ε. It is obvious that the points of discontinuity (the singular points) must be part of the mesh (Chapter 14). We still have to find the number and the location of the other points. If we consider a neighborhood that is sufficiently small around the point M_0, we can apply the local study introduced in Chapter 14. This leads to locating the point M_1 at a distance $\alpha\,\rho(M_0)$ from the point M_0, where the coefficient α is given by Relation (14.7) and $\rho(M_0)$ is the radius of curvature of Γ at M_0. The point M_1 is then defined as the intersection of the circle of radius $\alpha\,\rho(M_0)$ centered at M_0 and the curve Γ.

General scheme. A geometric mesh of an implicit curve can be obtained in the following manner :

- identify the extrema (of the radii of curvature) of the curve and the singular points,
- initialize the mesh with these points,
- approach the curve using sub-curves corresponding to the curves joining two such consecutive points,
- mesh each piece using the previous principle.

The main difficulty of this approach is related to the calculation of the lengths of the portions of curve. An easy way consists of using a sufficiently small (uniform) sampling to evaluate the desired length numerically. Obviously, this method can be costly. A "binary-searching" type method or a "divide-and-conquer" type method (Chapter 2) can also be envisaged. We will study different possible types of approaches below.

Before going in more detail about the meshing of implicit curves, notice that for a given accuracy threshold ε, it is possible to obtain different meshes of the same implicit curve, depending on the nature of the meshing algorithm employed. Hence, the notion of optimal result must be considered.

Optimal mesh. In Chapter 1, we already mentioned the notion of optimal mesh and indicated that this notion is related to the application envisaged (*i.e.*, a mesh is optimal with respect to a certain criterion and not necessarily optimal for another criterion). Here, we will consider an optimal mesh to be one which corresponds to a good geometric approximation (see below) and which contains a minimal number of vertices. In fact, we consider that :

Definition 16.3 *The* optimal *piecewise linear approximation is that which, among all possible solutions having the same quality of geometric approximation, minimizes the number of elements.*

However, this simple geometric criterion is not sufficient in practice to estimate the optimality of a solution (see also Chapter 18). Thus, for example, for a finite element computation, the number of elements of a mesh is an important factor

(as it conditions the size of the matrices). Moreover, the (shape and size) element qualities are also important as they relate to the numerical accuracy of the results and the convergence of some computational schemes [Ciarlet-1978]. It is thus important to avoid the creation of (poor quality) badly-shaped elements.

In practical terms, it is convenient to consider as optimal a mesh that achieves an acceptable compromise between the different criteria considered. Thus, for isotropic triangles in two dimensions, a quality close to 1 defines well-shaped elements. Such a quality indicates that the lengths of the edges are close to 1 (possibly in a given metric). We bring the notion of optimal mesh down to the notion of unit mesh :

Definition 16.4 *In two dimensions, a* unit mesh *is a mesh whose edge lengths are close to 1.*

Such a mesh is considered to be optimal.

16.2.3 General principle

Implicit curves and surfaces offer less flexibility than parametric curves and surfaces. In particular, it is rather tedious to determine the intrinsic properties of these curves and surfaces. In addition, tracking an implicit curve is a difficult problem (we will see that, from a given point, numerical techniques make it possible to locate a neighboring point on the curve). This is why, meshing techniques are largely inspired by heuristics. In particular, the classical approaches (see for instance [Allgower,Schmidt-1985] and [Allgower,Gnutzmann-1987]) consist in :

- sampling the domain of definition (using a covering-up of the domain),
- searching the roots of the function in the cells of the covering-up,
- constructing a topology (*i.e.*, a mesh) whose vertices are the root of the function.

In other words, a spatial partitioning (a set of disjoint and congruent cells enclosing the domain) of the domain is created and the implicit function is locally approached in each cell of this covering-up. These approaches usually involve numerical techniques to find the points along the curve (surface) in a given cell.

Remark 16.3 *Notice that creating a sampling makes the problem a discrete one, the value of the implicit function being known at the vertices of the sample. In fact, most of the envisaged methods are capable of dealing directly with discrete data (for instance, those obtained using a scanning device).*

Remark 16.4 *Notice that this problem is slightly different from that aiming at reconstructing a topology from a set of points all belonging to the boundary of the domain (see [Hoppe-1994] and [Boissonnat* et al. *1999], for example).*

In the next section, we examine different approaches to constructing a geometric mesh of an implicit curve in two dimensions (the meshing problem for implicit surfaces will be dealt with in the following section).

16.3 Implicit curve meshing

As we have already mentioned, meshing an (implicit) curve consists in discretizing it into a finite number of segments of suitable lengths. It is obvious that these lengths depend on the envisaged application (*i.e.*, on the constraints related to the application) and on the given metric information.

16.3.1 Construction of a spatial covering-up

Consider a given implicit curve Γ. The sampling stage consists in finding a set of points $P_i \in \Gamma$ sufficiently dense for the geometry of Γ to be approached within a tolerance of ε. The several techniques proposed can be classified into different classes, based on whether they proceed :

- by (exhaustive) enumeration,
- by continuation,
- by (adaptive) subdivision, etc.

To see this more clearly, we will first consider a naive approach to finding the points of Γ.

A "naive" approach (ray-tracing). To find the points on the curve, we can intersect Γ with a family $\mathcal{D}$ of lines (the domain is somehow sampled by a beam of lines).

Thus formulated, the problem consists in solving a set of one-variable equations of the type $f(x_i, y) = 0$ for a sample of uniformly distributed points $x_i \in \mathbb{R}$ (Figure 16.2).

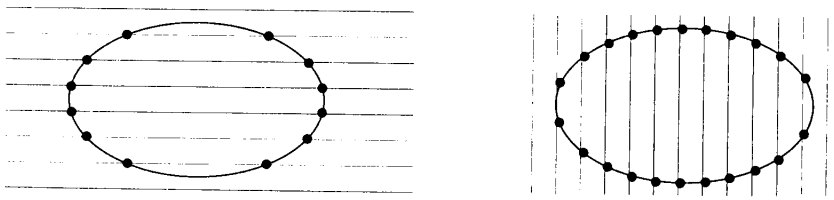

Figure 16.2: *Naïve method for an implicit curve. Notice that the same curve* Γ *leads to different samplings depending on whether the family of lines is horizontal (left-hand side) or vertical (right-hand side).*

This example illustrates simply the influence of the lines. Clearly, there does not exist any suitable criterion (*i.e.*, working in all cases) to fix the size and the density of the rays *a priori*. In fact, some lines do not contribute to the sampling while others can "miss" some intersections.

Close to the naive approach, we then find an approach of the exhaustive enumeration type.

Exhaustive enumeration approaches. The existence of discrete data, supplied by scanning devices for instance, leads to a problem for which the spatial covering-up is given. Usually, this covering-up is a regular grid (uniform and axis-aligned) or a Coxeter-Freudenthal triangulation (each cell of a regular grid is split into two triangles, [Freudenthal-1942], [Coxeter-1963]), the values of the implicit function being known at its vertices.

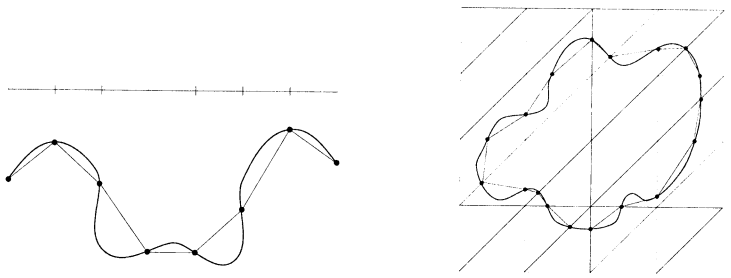

Figure 16.3: *Uniform approximation of a parametric curve (left-hand side) and of an implicit curve using a method of the exhaustive enumeration type (right-hand side).*

We can make here an analogy with the mesh of parametric curves : the segment representing the domain of the parameters is split into equally-sized sub-segments and the function f is used to find the vertices and to construct the polygonal approximation of the curve (Figure 16.3).

The principle of the *exhaustive enumeration* method consists in examining all cells of the partition and to process only those that are intersected by the curve. At the vertices of such a cell, the signs of the function are not constant. The curve Γ intersects a given cell side if the values of f at the two endpoints of this side are of opposite signs (Figure 16.4). The intersection points of Γ with a cell sides are either determined by a linear interpolation based on the values at the two edge endpoints, or using a procedure to find the roots (see below). We will see later that this technique allows us to approach the curve using a piecewise linear approximation $\widetilde{\Gamma}$.

As for the naive approach, the main difficulty of this type of approach (especially well-suited to discrete data) is to fix the resolution (*i.e.*, the cell size) so as to capture the local behavior of the curve as well as possible. Again, too coarse a sampling can lead to intersections being missed or to reconstructing a topology that is different from that of the curve[4].

This leads us to envisage methods that allow the curve to be *tracked* (in a sense that we will specify later).

Methods by continuation. The techniques by continuation aim at finding, from a given point $M_i \in \Gamma$, a point M_{i+1} close to it on the curve. Depending on

[4]However, too small a size is penalizing as it slightly increases the number of intersections checks and requires more accurate algorithms.

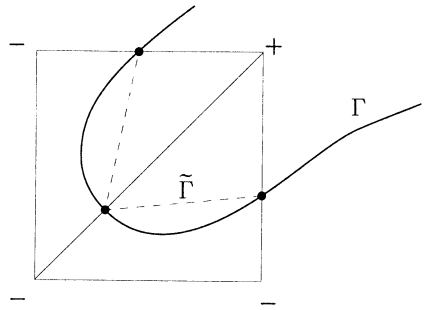

Figure 16.4: *Exhaustive enumeration method : intersection of the implicit curve Γ with two cells of the Freudenthal triangulation.*

the manner of predicting the position of M_{i+1}, the methods are of the *progression* type or of the *prediction-correction* type.

- Methods by progression.

This concerns incremental methods. From a basic cell (*i.e.*, containing a portion of the curve), the curve is approached by a set of cells, the partitioning being constructed "on the fly". Adding a cell can be performed by adjacency, the adjacency direction being determined by studying of the sign of the function at the vertices of the current cell (Figure 16.5). The neighboring cell is thus the cell sharing the edge of the current cell having its endpoints of opposite signs (the values of the implicit functions being evaluated at the cell vertices). The process stops as soon as the curve is fully enclosed in the cells. A coloring scheme can be used to avoid going back to cells that have already been analyzed. A stack (Chapter 2) is used to store the cells to be processed.

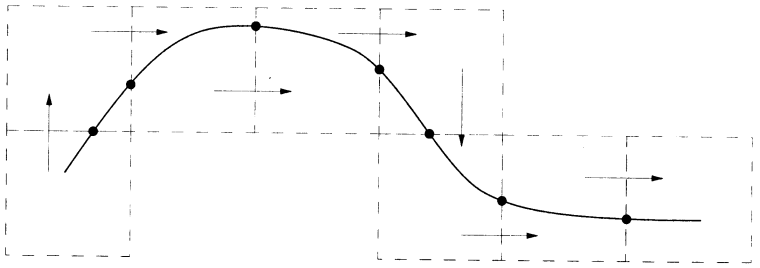

Figure 16.5: *Method by progression, the partitioning is constructed "on the fly" from a root cell.*

Such an approach is sensitive to the cell sizes which, if too coarse, do not allow the local behavior of the implicit curve to be captured (see the ambiguity problems below) and, if too fine, lead to a too great number of samples (thus penalizing further processing). Notice that, as for the naive approach, no criterion exists to fix the cell size *a priori*.

- Methods by "prediction-correction".

These methods allow us to calculate the position of a point on the curve from a known point $M_i \in \Gamma$ and a small displacement. The point M_i is "moved" along the tangent $\vec{\tau}(M_i)$ to the curve at M_i, to get a point M'_{i+1}. The position of this point (which does not belong to Γ) is then (iteratively) corrected, for example using a Newton-Raphson method (Chapter 11), to find a point $M_{i+1} \in \Gamma$ (Figure 16.6).

In this approach, the covering-up is virtual. One difficulty consists in fixing the stepsize of the displacement to avoid too fine a sampling or, on the other hand, to "miss" the curve (the correction method is not converging, especially in highly curved regions). Another problem consists in finding a starting point M_0 in each connected component of Γ.

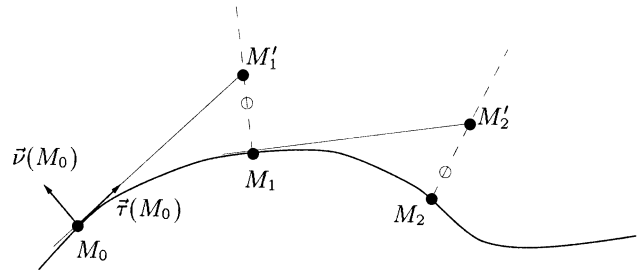

Figure 16.6: *Method by continuation of the prediction-correction type. From a position M_0, we determine a series of points $M_1,..,M_n$ along the curve Γ.*

This type of method is based on the property of orthogonality (in two dimensions) of the gradient vector ∇f and the vector $\vec{H} = (-\partial f/\partial y, \partial f/\partial x)$. The latter is thus tangent to the level curves[5] of f and, in particular, to $f(x,y) = 0$. Given a point $M_0 = (x_0, y_0)$, the points of the connected component containing M_0 are solutions to a system of the form :

$$\begin{cases} \dfrac{dx}{dt} = -\dfrac{\partial f}{\partial y} & \text{and } x(0) = x_0\,, \\ \dfrac{dy}{dt} = \dfrac{\partial f}{\partial x} & \text{and } y(0) = y_0\,. \end{cases} \tag{16.11}$$

The position of the new point M_1 is estimated as $M'_1 = M_0 + \partial\vec{H}(M_0)$. The point M'_1 does not belong in principle to Γ. A correction scheme of the Newton-Raphson type aims at moving M'_1 back to a point M_1 of Γ.

Method by recursive subdivision. From the study of the previous approaches, we can observe that controlling the geometric approximation of Γ is rather delicate. We have seen that there does not exist a general criterion to fix, *a priori*, the size of the cells of the partition (supposed to be uniform so far). In particular, the curves with discontinuities of the order G^1 are difficult to approach with such methods.

On the contrary, it seems reasonable to suppose that if the cell size is variable and, for example, related to the local curvature, the approximation of the curve would be better (even without talking about a reduced number of elements). That is the basic idea of adaptive subdivision methods.

[5] The level curves of f correspond to the curves defined by $f(x,y) = k$, $k \in \mathbb{R}$.

According to the principle of spatial decomposition methods (Chapter 5), from a bounding box of the domain, this type of approach constructs the partitioning of the domain in a recursive way. The bounding box is subdivided into four equally sized cells that can be organized hierarchically (using a *quadtree*, for example). The cell refinement is linked to a criterion related to the local curvature of Γ.

Among the useful criteria commonly used, we should mention :

- the *planarity* (*i.e.*, the angle between the normals at the intersection points),
- the variation of the radii of curvature at the intersection points,
- the number of intersection points, etc.

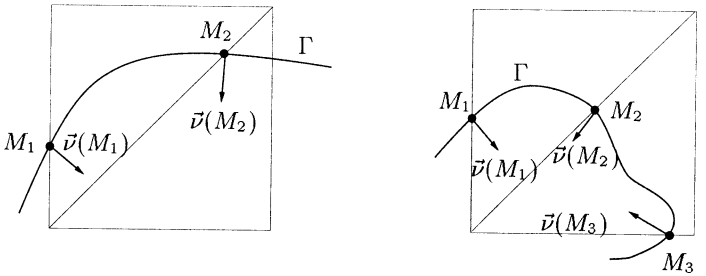

Figure 16.7: *Adaptive subdivision based on the evaluation of the local curvature. The normals $\vec{\nu}(M_i)$ at the intersection points M_i of Γ with the edges of the covering up are calculated and from their variation (gap), we deduce a possible refinement of the cells (right-hand side).*

This type of approach raises again the problems already discussed in Chapter 5 for the meshing of the domain boundaries. Notice here however that the tree structure is not necessarily balanced (using the [2:1] rule, for example). Actually, the aim of the decomposition is to provide a sample of points belonging to Γ and which is sufficiently representative of the behavior of the curve.

Once the sampling has been done (and independently of the approach chosen), we must connect the points of Γ together so as to obtain a mesh of the curve.

Remark 16.5 *Notice that if the objective is to visualize the curve Γ, it is sufficient to create a cloud of points that is dense enough to capture the variations of the curve and to provide its visual aspect.*

16.3.2 Meshing an implicit curve from a point cloud

From the cloud of points defined at the previous stage, we now try to extract a mesh of the given curve. Notice immediately that this problem is very similar to that of the search for a curve passing through a given set of points (see [Hoppe *et al.* 1991],

for example). However, unlike the latter[6], we have here some additional information provided by the spatial covering up.

Construction of a geometric mesh. Depending on the sampling technique adopted, the mesh creation can be more or less trivial. Thus, with a continuation method, the mesh is obtained by simply connecting the intersection points (the M_i's) two-by-two in the order of their creation. With the enumeration type methods, the task is more tedious. In this case, the discretization is obtained by analyzing each cell intersected by the curve and by connecting the intersection points belonging to the cell sides. It should be pointed out that some configurations can lead to topological ambiguities (Figure 16.8).

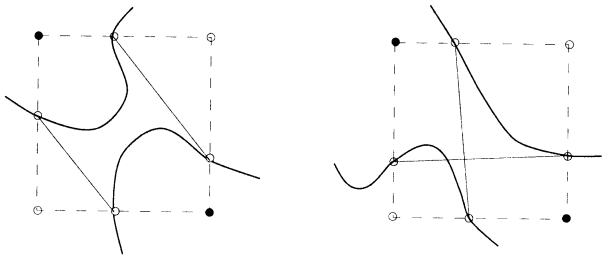

Figure 16.8: *Example of topological ambiguities : we illustrate here two ways of connecting the intersection points that violate the topology of the implicit curve.*

Exercise 16.1 *Identify the topological ambiguities possible with a enumeration type method and propose a simple way of getting rid of these ambiguities (Hint : examine the sign of the function at the vertices of the partition and use the intrinsic properties of the curve).*

From a practical point of view, the mesh edges are created by using predefined patterns (*templates*) based on the sign of the implicit function at the cell vertices. For a uniform decomposition or a quadtree type decomposition, only the terminal cells intersected by the curve need to be considered. We identify thus $2^4 = 16$ distinct patterns.

At completion of this stage, the implicit curve Γ is thus approached by a polygonal segment. However, the geometric mesh obtained is not necessarily correct with a view to numerical computation (the size variations between two edges incident at a vertex might be great). A control on the edge size variation can be imposed. Here, we come up against a well-known problem.

Mesh of an implicit curve defined by a discretization. We find here in a case where the metric to follow is not necessarily of a geometric nature. We must then intersect this metric (Chapter 10) with the geometric metric. The problem

[6] Which can sometimes be NP-hard, for example, when the goal is to find the closed polygon of minimal perimeter having the given points as vertices (traveling salesman problem).

can be identified with the rather difficult curve remeshing detailed in Chapter 14. We can thus adopt a technique which consists in reconstructing a geometric support (a parametric arc this time) and then remeshing this support using classical optimization tools (point insertion, vertex removal and node relocation). But we can also use the information extracted from this covering up (singular points, normals, tangents, etc.) to control the remeshing procedures. Notice that point insertion can be performed via the spatial covering up (to facilitate root searching).

In this section, we have mentioned root searching several times (*i.e.*, the intersections between the curve and a given edge). Before dealing with the surface meshing, we will go back to several practical aspects of curve meshing.

16.3.3 Computational aspects of implicit curve meshing

In this short section, we briefly mention several practical aspects of the implicit function meshing and especially the search for intersection points and the approximate calculation of the gradient of the function at a given point.

Root finding. The methods previously discussed all rely on the identification of the intersection points between an edge of the covering up and the curve Γ. When the implicit function is continuous and monotonic, the intermediate value theorem ensures the existence of (at least) one solution along the edge AB, if the values $f(x_A, y_A)$ and $f(x_B, y_B)$ of the function at these points are of opposite signs. If the derivative of the function is known exactly, a Newton type method can be used to find the root, that is the point $P(x, y)$ such that $f(x, y) = 0$. However, this method can be quite impredictable and may not converge for some functions.

In all cases, a binary searching method allows us to quickly find the root within a given tolerance ε. The tolerance is usually based on a fraction of the edge length or on a maximum number of iterations [Bloomenthal-1988]. In some special cases, the algorithm can be modified so as to improve the localization of the intersection point, notably when the function f vanishes on a sub-segment of AB (Figure 16.9 and see [Blinn-1982], [Frey,Borouchaki-1996], for example).

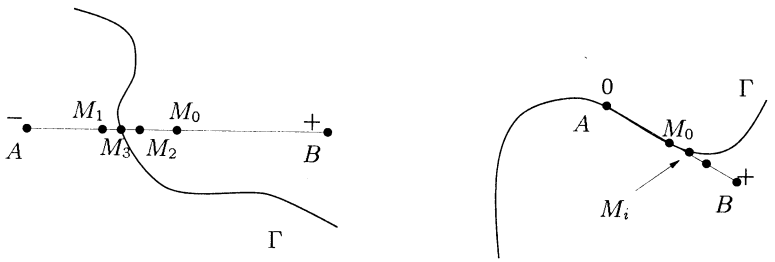

Figure 16.9: *Search for the intersection point using a binary searching method along the edge AB. Left-hand side, the intersection point is identified as the point M_3. Right-hand side, the vertex A is a root, but the modified algorithm makes it possible to find a point M_i as the intersection point.*

When the curve Γ is known in a discrete way, (*i.e.*, the corresponding implicit function is not known explicitly), the intersection points may be approached using a linear interpolation based on the given values of the function. Given an edge AB for which the values $f(x_A, y_A)$ and $f(x_B, y_B)$ are of opposite signs, a linear interpolation along AB makes it possible to find the value of f at $P(x, y)$ (corresponding to the parameter t) using the formula :

$$f(x, y) = (1 - t)\, f(x_A, y_A) + t\, f(x_B, y_B)\,, \tag{16.12}$$

where the function f varies linearly between $f(x_A, y_A)$ for $t = 0$ and $f(x_B, y_B)$ for $t = 1$.

Remark 16.6 *The detection and the identification of singular and critical points can be performed during the binary searching of the roots by modifying the corresponding algorithm so as to take into account information regarding the gradient of the function at each of the evaluation points [Attili-1997], [Schöberl-1997].*

Estimation of the gradient. At a regular point $P = (x, y)$, the normal $\vec{\nu}(P)$ to the curve Γ can be estimated as the unit gradient vector at this point. If the partial derivatives are not known analytically, the gradient can be approached numerically, using a finite difference scheme :

$$\langle \nabla f(P), \delta(P) \rangle \approx f(P + \delta(P)) - f(P)\,, \tag{16.13}$$

where $\delta(P)$ represents a small size (generally chosen as a fraction of the cell size of the covering up) vector. A centered difference scheme, which is also possible, then gives :

$$\langle \nabla f(P),\, 2\,\delta(P) \rangle \approx f(P + \delta(P)) - f(P - \delta(P))\,. \tag{16.14}$$

Posing $\delta(P)_i = h e_i$ where e_i is the unit vector related to the i-axis and h is a small value, we obtain, for the first approximation :

$$\nabla f(P) \approx \frac{1}{h} \begin{pmatrix} f(P_1) - f(P) \\ f(P_2) - f(P) \end{pmatrix},$$

where $P_i = P + h\, e_i$ and this approximation relates the error to h while the second approximation relates the error to h^2. Depending on the value of h, either of these approximations can be chosen.

Remark 16.7 *At singular points, where the gradient is not defined, the normal can be obtained as a mean value of the normals at the neighboring vertices.*

We will now show that mesh generation for implicit surfaces involves similar approaches to those used for meshing implicit curves.

16.4 Implicit surface meshing

The techniques used to mesh an implicit surface derive from the techniques developed for meshing implicit curves. However, several problems that are particular to surfaces will be dealt with in this section.

For the same reasons as in two dimensions, the meshing of implicit surfaces relies on a sampling stage and a mesh construction stage given a point cloud. Constructing a spatial covering up makes it possible to localize the processes. In other words, the goal is to capture locally the behavior of the surface in a small volume element. Hence, each element of the covering up intersected by the surface is analyzed and the roots of the function (*i.e.*, the points of Σ) are identified along the edges of this element.

16.4.1 Construction of a spatial covering-up

The objective is here to find a covering up of the domain of study that allows us to extract a set of points $P_i \in \Sigma$ which is sufficiently dense for the geometry of the surface Σ to be approached within a given tolerance ε. As in two dimensions, we have the following classification of spatial covering methods :

- methods by exhaustive enumeration,
- methods by continuation,
- methods by adaptive subdivision.

Methods by exhaustive enumeration. In this type of approach, the covering up $\mathcal{R}$ is usually an input of the problem and this covering up is then a regular grid or a triangulation (Coxeter-Freudenthal type, for example). This is notably the case when the implicit surface is an iso-surface defined in a discrete way using scanning devices (scanners). The implicit function is sampled and its value is known at the nodes of a lattice of (usually cubical) cells, that is both structured (for each point, the number of adjacent points is constant) and uniform (the distance being two points is constant). The principle of an *exhaustive enumeration* method consists in :

- identifying the elements of $\mathcal{R}$ intersected by the surface and
- locating the intersection points (of Σ with the edges of the elements of $\mathcal{R}$).

The surface Σ will then be discretized using a piecewise linear approximation $\widetilde{\Sigma}$ in each element of $\mathcal{R}$.

As in two dimensions, this type of approach is sensitive to the grid resolution (this being given or constructed). Figure 16.10 shows the influence of the element size on the accuracy of the geometric approximation.

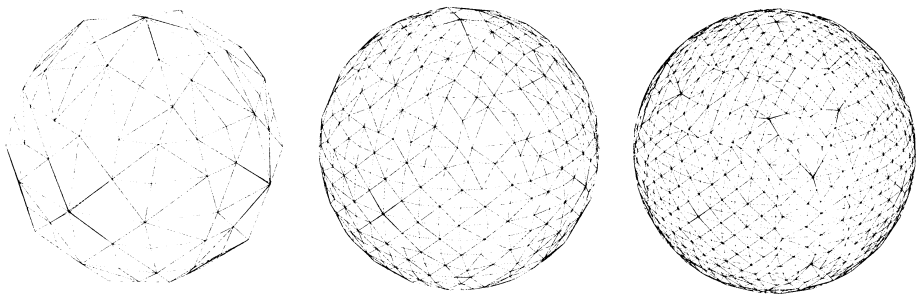

Figure 16.10: *Discretizations of a sphere by successive refinements. At each stage, the covering up is uniform (*i.e., *formed by equally-sized cells).*

Continuation methods. Continuation methods for surfaces are on all points similar to continuation methods for curves in two dimensions. We thus find methods by *progression* and methods by *prediction-correction.*

Recall just that a progression method consists in starting from an element of $\mathcal{R}$, the *seed* and progressing by adjacency towards the neighboring elements. The direction of the progression is defined by the sign of the values of the function at the vertices of the current element. A stack is used to store the elements to be processed. In principle, this method makes it possible to process a single connected component for a given seed.

As for implicit curves, prediction-correction type methods can be applied successfully to "track" an implicit surface. From an initial point M_0, we look for a point M_1' obtained by a small displacement of M_0 in the tangent plane $\Pi(M_0)$ associated with M_0. The point M_1' *predicted* is then iteratively relocated onto the surface to a point M_1, via a *correction* stage based on a Newton-Raphson procedure.

Remark 16.8 *The tedious part corresponds to the determination* a priori *of a seed for each connected component. Moreover, the set of vertices obtained is not intrinsically ordered which may result later (during the structuration stage) in overlapping elements.*

Adaptive subdivision methods. For the same reasons as in two dimensions, it is desirable to construct a covering up $\mathcal{R}$ that is representative of the intrinsic properties of the surface.

An adaptive subdivision method consists in including the domain of study into a bounding box which is recursively subdivided into identical elements[7]. We thus obtain a structure that is naturally organized hierarchically, an *octree* for example in the case where the basic element is a cube (Chapter 5).

[7]The requisite of getting identically shaped elements is justified by the need to avoid the degeneracy of volumes resulting from the subdivision.

Remark 16.9 *A covering up based on tetrahedral cells is also possible. Hence, the* Kuhn tetrahedron, *based on the points* $(0,0,0)$, $(1,0,0)$, $(1,1,0)$, $(1,1,1)$ *can be subdivided into similarly shaped tetrahedra. The subdivision procedure leads to eight sub-tetrahedra, four of which share a vertex of the initial tetrahedron and the four others being internal (Figure 16.11).*

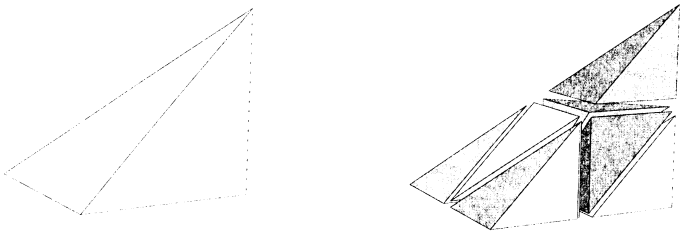

Figure 16.11: *Decomposition of Kuhn's tetrahedron (left-hand side) into eight tetrahedra (right-hand side).*

Exercise 16.2 *Enumerate the tetrahedra of Kuhn's decomposition. Show that the decomposition cannot be applied on the tetrahedron based on the points* $(0,0,0)$, $(1,0,0)$, $(0,1,0)$, $(0,0,1)$.

An element of $\mathcal{R}$ is subdivided whenever a specific criterion is not satisfied for this element. The criteria considered for refining an element of $\mathcal{R}$ are mainly based on [Schmidt-1993] :

- the variation between the normals at the vertices (or the faces) of the current element (*planarity* criterion) :

$$\max_{P_i} \langle \vec{v}_i, \vec{\nu}(P_i) \rangle < \cos(\varepsilon_1),$$

  where $\vec{v}_i$ represents the unit vector supported by the segment P_iP_{i+1}, $\vec{\nu}(P_i)$ is the unit normal at P_i and ε_1 is a given tolerance,

- the *divergence* of the normals at the vertices P_i with respect to the normal at the center of the element M, measured by the quantity :

$$1 - \min_{P_i}(\langle \vec{\nu}(P_i), \vec{\nu}(M) \rangle < \cos(\varepsilon_2),$$

- the changes between the signs of the values of the function at the vertices,
- the proximity of a singularity, etc.

We will see in Chapter 19 that some criteria can also be used to optimize the surface meshes.

With this type of approach, the subdivision leads to a partitioning for which the element density (thus that of the sampling points) is proportional to the local curvature (for a geometric triangulation, that is for which the geometric approximation to the surface is controlled by a given tolerance value ε). However, the discontinuities of order C^0 of the surfaces are more tedious to capture exactly and require more sophisticated algorithms. Figure 16.12 illustrates the problem of searching for the intersection points along the edges of the elements of $\mathcal{R}$.

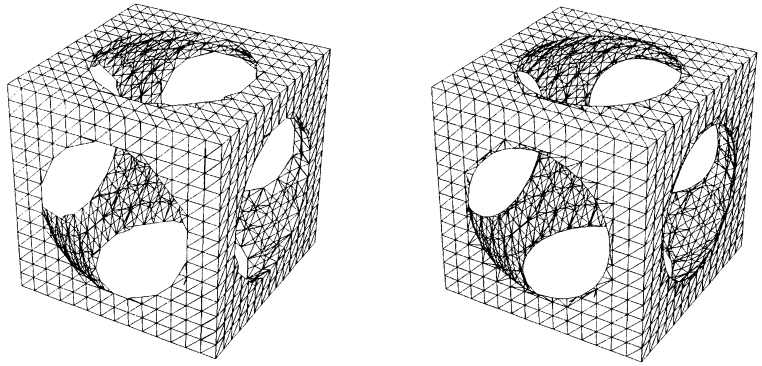

Figure 16.12: *Example of root finding, "*wiffle cube*". left-hand side, a linear interpolation is used to find a root along an edge of the decomposition. Right-hand side, a binary searching algorithm is used to find a change of sign of the function along the edge, thus improving the approximation of the sphere.*

Remark 16.10 *Another approach consists of starting from a coarse partition of the space into tetrahedra. The latter are analyzed to decide on a possible refinement, according to the criteria mentioned. The difference lies in the fact that when a tetrahedron is subdivided, its barycenter (or any other suitable point) is created and inserted in the current triangulation using the* Delaunay kernel *procedure (Chapter 7) [Frey,Borouchaki-1996]. This approach constructs, incrementally, a triangulation of the domain conforming to the Delaunay criterion.*

We now deal with the construction of a surface mesh from the point cloud obtained at completion of the sampling stage.

16.4.2 Mesh of an implicit surface defined by a point cloud

At this stage, we have a cloud of points located on the surface, as dense as the partitioning was fine. We now have to connect these points in order to obtain a geometric mesh of the surface. It seems obvious that the tedious aspect of this operation consists in making sure that the topology of the discretization conforms to that of the implicit surface it represents.

For the sake of convenience, we will detail here the construction of a mesh where the cloud of points comes from a enumeration type method[8] or an adaptive

[8]This choice is also justified by the fact that this type of technique is widely used in practice

subdivision method. We will leave it up to the reader to see how a mesh may be obtained from a continuation method [Bloomenthal-1988], [Wyvill *et al.* 1986].

Construction of a geometric mesh. When the covering up $\mathcal{R}$ is a uniform grid, the connectivity of $\mathcal{R}$ allows us to use a very simple (and nowaday very popular) algorithm to construct a surface mesh. This algorithm has also proved to be especially well-suited for processing discrete data [Lorensen,Cline-1987].

- "*Marching Cubes*" algorithm

Based on a *divide and conquer* approach, the method consists in analyzing any element of $\mathcal{R}$ intersected by Σ (the identification of an element K being based on the study of the signs of the function at the vertices of K). Each vertex can be either positive or negative (the case where the value is null is a peculiar case that can be resolved by *dilating* the surface locally), the eight vertices of a given element K allow us to construct an index with a value in $]0, 256[$, (as $256 = 2^8$).

In practice, with each value of the index is associated a list of polygons used to locally approach the surface. The 255 potential lists can be reduced to 15 representative cases, using symmetrical and rotational properties. These 15 cases lead to 15 predefined patterns (*templates*) that serve to define a piecewise linear approximation of the surface in each element.

Exercise 16.3 *Retrieve the cases representative in Figure 16.13 from the 256 possible cases.*

Such an algorithm constructs polygons that have 3 to 6 vertices (Figure 16.14). To get a mesh composed only of triangles, we have to subdivide the polygons of a higher degree than 3 into triangles. This involves a combinatorial procedure (to find all the possible topologies) as well as a geometrical procedure (to chose from all the possible ones, the one that leads to the best geometric approximation).

Theoretically, *Catalan's number* of order n

$$Cat(n) = \frac{(2n-2)!}{n!(n-1)!},$$

is a formula giving the number N_n of different triangulations of a polygon with n vertices : $N_n = Cat(n-1)$. Thus, when $n = 3, 4, 5$ or 6, we find respectively $N_n = 1, 2, 5, 14$ (Chapter 18).

The drawback of the method is related to the fact that topological ambiguities and/or non-closed surfaces can be created. This is the case when an element of $\mathcal{R}$ contains a face in which vertices of opposite signs are diagonally opposed two-by-two (Figure 16.15) [Dürst-1988], [vanGelder,Wilhelms-1992], [Montani *et al.* 1994].

Several techniques enable us to remedy this problem, for example using :

- the value of the function at the center of the face [Wyvill *et al.* 1986] (although this solution may fail [Matveyev-1994]),
- a bilinear representation of the function, a hyperbolic curve describing the intersection of Σ with an edge, the value of the function at the intersection points of the asymptotes of the hyperbola makes it possible to predict the topology [Nielson,Hamann-1991].

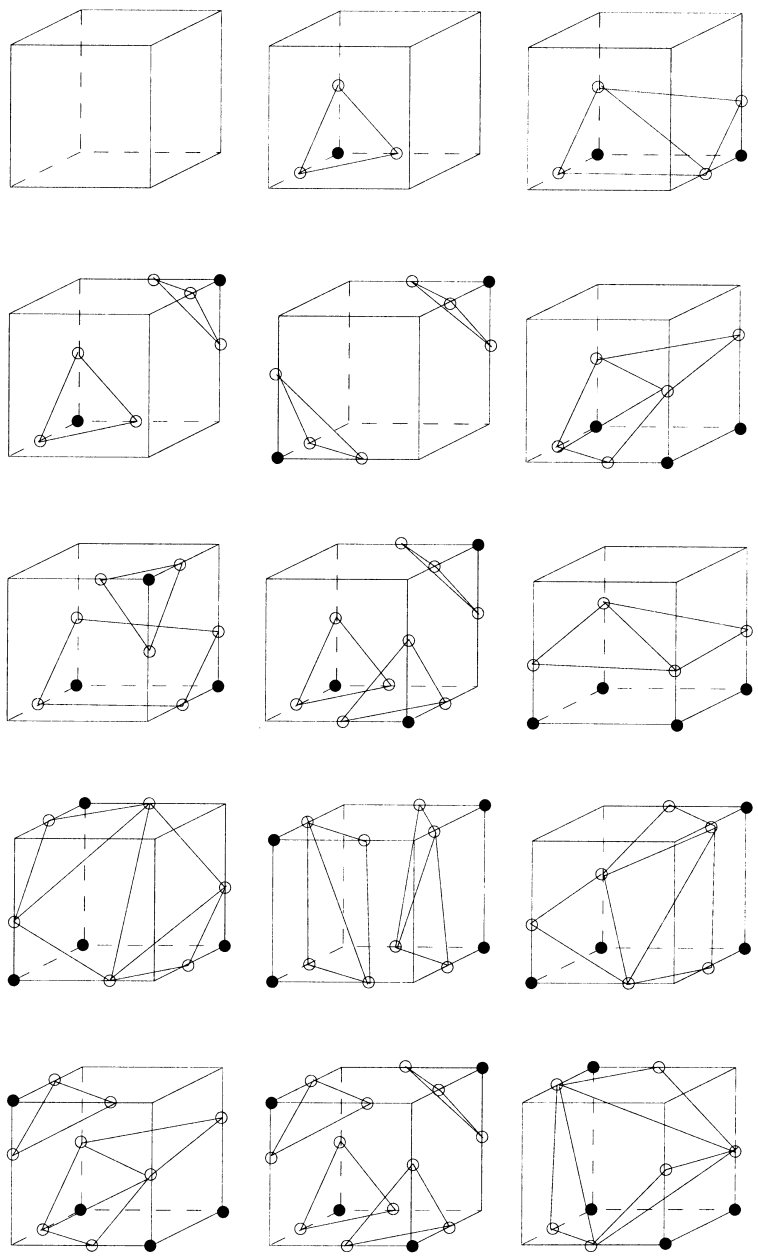

Figure 16.13: *"Marching Cubes" algorithm : intersections surface-cell. The* $2^8 = 256$ *possible patterns can be reduced to a set of 15 configurations using different properties of symmetry and rotation preserving the topology of the triangulated surface (notice here that the polygons having more than three vertices have been subdivided into triangles).*

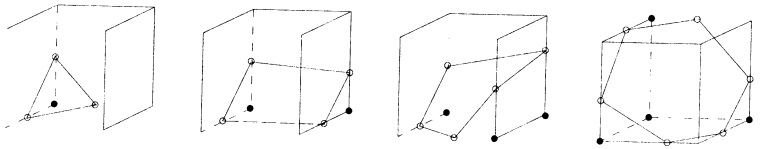

Figure 16.14: *Different types of polygons constructed using predefined patterns ("templates") by a "*Marching-Cubes*" type algorithm, if the numbers of vertices are 3,4,5,6, from left to right, respectively.*

Notice however, that these approaches cannot in practice be used for discrete data.

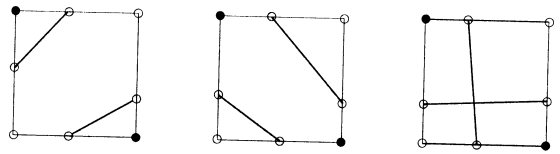

Figure 16.15: *Faces presenting a topological ambiguity. Several connections are possible between the vertices.*

Adaptive subdivision methods. In this type of approach, the elements of the spatial partitioning $\mathcal{R}$ are all of the same type (cubes or tetrahedra) although their shapes and sizes can vary locally, based on the local properties of the surface.

If the partitioning is represented by an octree, the construction of the geometric mesh is close to that used in the "*Marching Cubes*" type algorithm (see, for instance, [Wilhelms,vanGelder-1990], [Frey-1993]).

When the partitioning is simplicial, vertices of opposite signs can be separated by a single plane, thus leading to only $2^4 = 16$ possible configurations, the only polygons formed being triangles and quadrilaterals (Figure 16.16). The quadrilaterals obtained can then be subdivided into triangles according to geometric criteria (Chapter 19). The degenerate cases (a tetrahedron vertex belongs to the surface) can be avoided by locally *expanding* the surface [Frey,Borouchaki-1996].

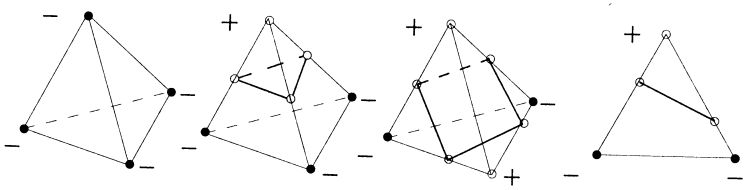

Figure 16.16: *Different triangulations possible, depending on the sign at the vertices, of a cell for a simplicial covering up.*

Meshes obtained by such approaches are geometric meshes. However the size variations between neighboring elements are not controlled, thus this type of mesh

may not be suitable for a finite element type calculation. Here we are confronted again with the problem of surface mesh optimization.

16.4.3 Surface mesh optimization

The purpose here is not to specify the optimization operations used for surface meshes which will be described in Chapter 19, but to present the context of such an optimization.

Shape quality optimization. The objective is to optimize a pertinent quality criterion with respect to the envisaged application (here a finite element type calculation). A shape quality measure for a triangle K is given by the formula :

$$\mathcal{Q}_K = \alpha \frac{h_{max}}{\rho_K}, \tag{16.15}$$

where h_{max} represents the diameter of the element and ρ_K the radius of the inscribed circle (α is a normalization coefficient so that $\mathcal{Q}_K = 1$ for an equilateral triangle). The goal is to obtain a quality value close to 1 for all mesh elements. To this end, topological modification operators are used (which preserve the point locations but modify their connections) as well as metric modifications (which modify the points locations while preserving their connections).

Mesh simplification. The number of elements in a geometric mesh is related to the gap between the element and the underlying surface. However, depending on the application envisaged, too great a number of elements can be penalizing. *Simplification* (or decimation) methods reduce the number of elements of a mesh while preserving the quality of the geometric approximation (Chapter 19). Such methods involve the same modification operators as optimization procedures.

16.4.4 Computational aspects of implicit surface meshing

In some applications (such as computer graphic visualization or when the surface mesh is the input of a volumetric meshing technique), the consistent orientation of the triangulation can be a very important requirement (Chapter 6).

Orientation of the triangles. The orientation of the triangles in the surface mesh can be performed either *a priori*, or *a posteriori*. In the first approach, the polygons of the patterns used to mesh are oriented counterclockwise (for example) based on a canonical orientation at the cell level (Figure 16.17 for a tetrahedron). In the second approach, the polygons are oriented by adjacency in a consistent way, in each connected component (see Chapters 1 and 2 for data structures and algorithms appropriate to this type of process).

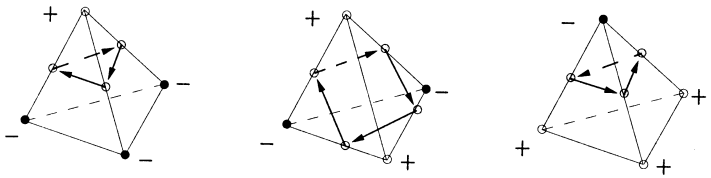

Figure 16.17: *Consistent orientation of polygons in a tetrahedron.*

Memory resources and data structures. Memory resources correspond to the data structures necessary to store the information related to the mesh (nodes, triangles, etc.) as well as the structures related to the spatial partitioning. The main internal data structures contain :

- an array of mesh vertices (coordinates),
- an array of triangles (list of vertices),
- an array for the neighboring elements (edge adjacency),
- an array for the vertices of the covering up,
- an array for the elements of the covering up,
- additional resources (for example to store the value of the function at the vertices of the covering up), etc.

16.4.5 Examples of surface meshes

To conclude this section, we provide several examples of implicit surface meshes created by some of the methods described above. To help the understanding of these meshes, Table 16.1 indicates some characteristic values, *np*, *ne* denote respectively the number of vertices and elements of the triangulation and $\mathcal{Q}_M$, $\mathcal{Q}_{pire}$ indicate the mesh quality and the quality of the worst element before optimization.

-	*np*	*ne*	$\mathcal{Q}_M$	$\mathcal{Q}_{pire}$
case 1	6,656	12,816	1.88	22.07
case 2	36,208	72,432	3.57	190.
case 3	122,940	244,431	4.02	925.

Table 16.1: Statistics related to several examples of implicit surface meshes.

The evaluation of (implicit) surface meshes is based on geometric criteria related to the quality of the geometric approximation (see [Frey,Borouchaki-1998] and Chapter 19). On the other hand, depending on the envisaged application, (*i.e.*, a finite element calculation), it is important to guarantee a good element shape quality.

The example in Figure 16.18 corresponds to the reconstruction of an iso-surface using an exhaustive enumeration method based on a regular grid.

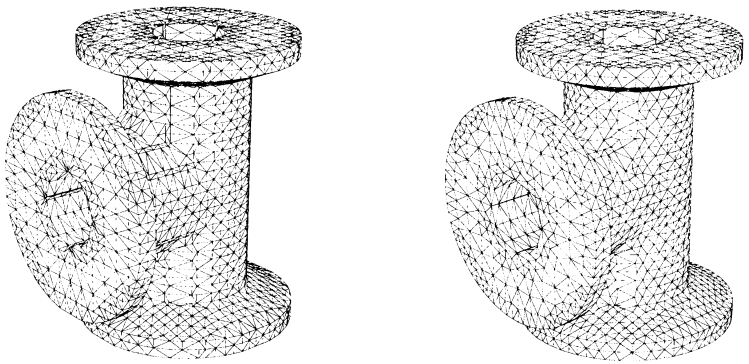

Figure 16.18: *Exhaustive enumeration method applied to the reconstruction of a surface from a regular grid. Initial mesh (left-hand side) and optimized mesh (right-hand side).*

The second example (Figure 16.19) presents different meshes of an implicit surface of degree six corresponding to the equation [Schmidt-1993] :

$$f(x,y,z) = (x^2+y^2-4)(x^2+z^2-4)(y^2+z^2-4) - 4.0078 = 0\,.$$

The initial mesh (left-hand side) has been created using an adaptive partition in tetrahedra. Notice that the geometric approximation of the surface is correct, although the shape quality of the mesh is not acceptable (at least for a finite element calculation). Hence, this mesh has been optimized (middle) and then simplified (right-hand side).

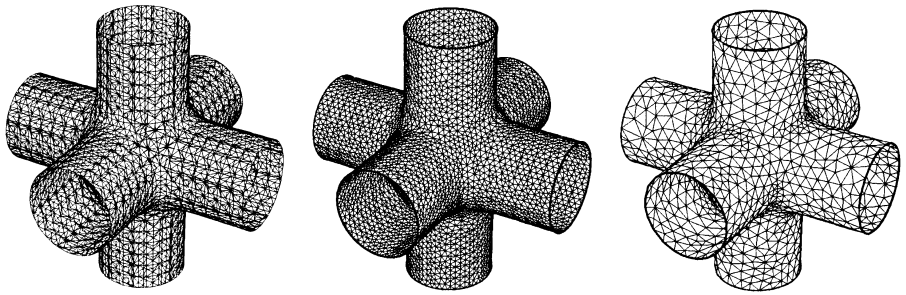

Figure 16.19: *Different meshes of an implicit surface : initial geometric mesh obtained by an adaptive method (left-hand side), mesh optimized with respect to the triangle shape quality (middle) and simplified geometric mesh (right-hand side).*

Finally, the last two examples (Figures 16.20 and 16.21) show iso-surface meshes for biomedical applications[9] created from discrete data.

[9]Here, the iso-surfaces are associated to anatomic structures.

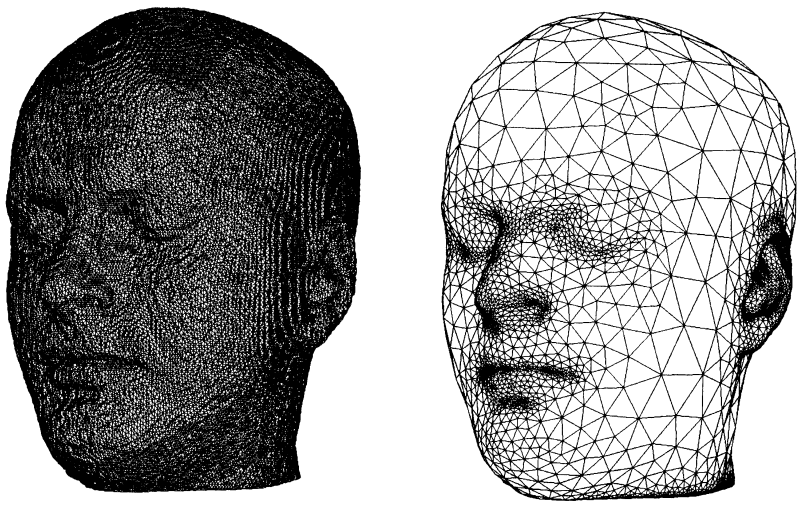

Figure 16.20: *Mesh of an iso-surface of a head from volumetric data (left-hand side, data courtesy of Mika Seppa, Low Temp. Lab., University of Technology, Helsinki, Finland) and simplified and optimized mesh (right-hand side).*

In the example in Figure 16.20 it seems obvious that the original mesh (left-hand side) contains too many elements to be numerically exploitable (67,106 vertices and 134,212 triangles). Therefore, a simplification procedure is applied to obtain a geometric mesh where the density of the elements is related to the local curvature of the surface [Frey,Borouchaki-1998].

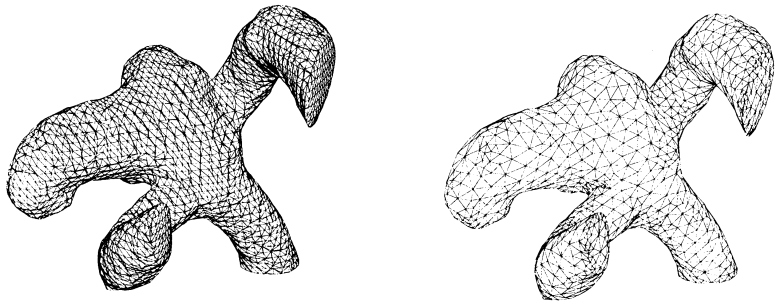

Figure 16.21: *Biomedical iso-surface meshes (an anevrism of the central cerebral artery) from discrete data. Left, initial mesh obtained using a* "Marching Lines" *type algorithm [Thirion-1993]. Right, optimized and simplified mesh.*

16.5 Extensions

To conclude this chapter, we will briefly evoke the possibility of using implicit functions for modeling purposes. We will then mention the construction of meshes for such domains.

16.5.1 Modeling based on implicit functions

Classically, we can distinguish three geometric modeling systems, depending on whether they :

- are based on a boundary representation of the solid (a *B-Rep*), the objects being represented by the union of their boundaries,
- use a spatial decomposition, the domain being approached by the union of the internal cells or the cells intersected by the domain boundary,
- involve constructive geometry (C.S.G.), the domain being represented by a tree structure in which the leaves are the primitives and the nodes correspond to boolean operations.

Constructive solid geometry. Constructive solid geometry (C.S.G.) stems from the work of [Rvachev-1963] related to R-functions, in the context of the numerical resolution of problems in complex domains. In this type of representation, the domain is defined via a unique function, which is at least C^0-continuous, with real values $f : \mathbb{R}^d \longrightarrow \mathbb{R}^+$, called the *representation function* (*F-Rep*). This function can be defined either algorithmically (that is encoded with an algorithm that returns the value of the function at a given point), or in a discrete manner, for instance at the nodes of a regular grid [Pasko *et al.* 1995]. With this approach, the topology of the domain is preserved both implicitly (within the tree structure) and explicitly (by the primitives).

A functional representation. Let us consider a closed domain defined by such a function f, we assume that :

$$\left\{ \begin{array}{rcl} P \in \mathring{\Omega} & \Longleftrightarrow & f(P) > 0 \\ P \in \partial\Omega & \Longleftrightarrow & f(P) = 0 \\ P \in \mathbb{R}^d \setminus \Omega & \Longleftrightarrow & f(P) < 0 \end{array} \right. \tag{16.16}$$

which introduces a *classification* of the points of $\mathbb{R}^d$. Hence, the points P internal to the domain are those verifying $f(P) > 0$, whereas the boundary points are such that $f(P) = 0$. We can see how this approach relates with implicit curves and surfaces introduced in the previous sections.

Primitives. In C.S.G., numerous geometric forms can be considered as primitives, from the simplest ones to the more "exotic" ones, from planes and quadrics (spheres, cones, cylinders, ellipsoids, etc.), to superquadrics (Figure 16.22).

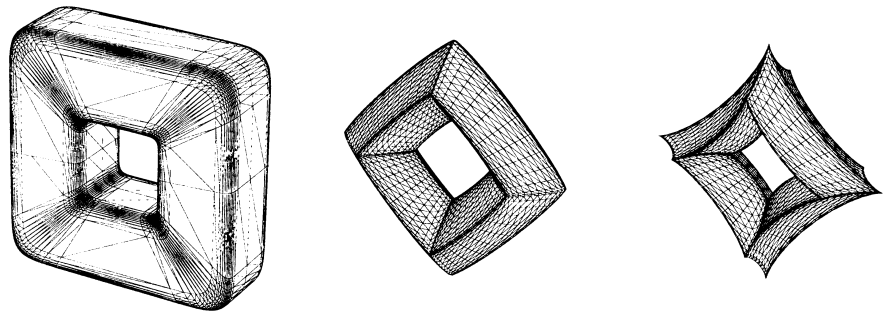

Figure 16.22: *Examples of superquadric surfaces (supertorus) that can be used as primitives in C.S.G.*

More generally, each *primitive* can be expressed as a special instance (occurrence) of a function chosen from a finite set of possible types. Hence, for instance, the equation :

$$f(x, y, z) = r^2 - \left((x - c_x)^2 + (y - c_y)^2 + (z - c_z)^2\right) = 0$$

defines, in three dimensions, the primitive corresponding to the representation of a sphere of radius r and centered at the point $C = (c_x, c_y, c_z)$.

Boolean operations. As we have already mentioned, constructive geometry is based on the combination of primitives using boolean unary and binary operations. In this context, the set operations is replaced by operations related to the corresponding functions.

For example, the *union* and the *intersection* of two primitives, defined by the function f_1 and f_2, can be written as follows[10] :

$$f_1 \cup f_2 = f_1 + f_2 + \sqrt{f_1^2 + f_2^2} \quad \text{and} \quad f_1 \cap f_2 = f_1 + f_2 - \sqrt{f_1^2 + f_2^2}\,. \tag{16.17}$$

Notice that these functions C^k-continuous ($k \geq 1$) present C^1 discontinuities when $f_1 = f_2 = 0$.

The reader is referred to [Pasko *et al.* 1995] to find the definition of several other operations.

16.5.2 Implicit domain meshing

We have seen that an implicit domain is one whose boundary is an implicit curve (surface). Given this remark, two approaches are possible to mesh such a domain :

- the *hierarchical* approach which consists in meshing the boundaries of the domain using one of the meshing techniques described here, prior to providing the resulting mesh to a "classical" meshing technique (Chapters 5 to 7),

[10] Using the symbols relative to the underlying sets.

- the *global* approach which considers the mesh generation for implicit domains as an extension of the meshing techniques for implicit curves and surfaces.

We will now briefly explain the global approach.

Global approach. Enumeration or adaptive subdivision methods are more adapted to this problem than continuation methods, mainly because the latter "track" the boundary and "forget" the elements not intersected by the boundary. The creation of the point cloud and then of the boundary mesh (curve or surface) is thus considered to be resolved.

The internal point creation and thus of the elements is related to the type of the spatial partitioning used. Hence, in three dimensions, when the covering up is of composed of *voxels* (uniform regular grid), or of the *octree* type, predefined patterns can be used [Frey *et al.* 1994]. For a simplicial covering up, the *texel* approach proposed by [Frey *et al.* 1996], consists in introducing a point in each simplex (this point is chosen so as to guarantee a suitable mesh gradation) and in using the *Delaunay kernel* as the element creation procedure (cf. Chapter 7).

Finally, an optimization procedure based on the element shape quality is applied, first on the boundary elements, then on the interior of the domain (refer to Chapters 18 and 19 for more details about the modifications used).

Chapter 17

Mesh modifications

Introduction

The aim of this chapter is to review some methods designed for mesh modification and manipulation (except for those concerning mesh optimization which will be discussed in the next two chapters). Provided with one or several meshes we are concerned with the methods that manipulate this (these) mesh(es) in various ways. Among the points of interest here, we first review mesh transformations with a geometric nature, the way in which two meshes can be merged into a single one, refinement techniques and various operations related to the attributes associated with the mesh in question.

Geometric mesh transformations are briefly reviewed together with transformations resulting in a global or a local mesh refinement and methods for geometric type change. In addition we discuss how to merge two meshes (sharing a common boundary part). Then, we give some information about node construction and node numbering for elements other than P^1 elements (namely where the nodes can be different from the vertices). Various representative finite elements are listed to illustrate how to construct (to number) the nodes. Then we focus on how to optimize, in some sense, both the vertex (node) and the element numbering in a mesh. To end the chapter, various other aspects are discussed (including how to manage the mesh physical attributes properly, how to use two different meshes, etc.).

17.1 Mesh (geometric) modifications

Depending on the geometry of the domain we want to mesh and the capabilities of the mesh generation method that is used, it is often tedious to construct a

mesh that satisfies certain repetitive properties (such as symmetry) enjoyed by the domain itself.

Thus, one way to achieve such repetitive features is to design the mesh generation process by defining explicitly what is expected. In other words, a domain with a symmetry (if we consider this example) is split into two parts. One of these is effectively meshed (using a mesh generation method), then a symmetry is applied to the resulting mesh and both meshes are merged so as to complete a mesh of the whole domain that obviously includes this symmetry.

17.1.1 Popular geometric transformations

For the sake of simplicity, we only consider P^1 meshes (the element nodes are the element vertices, the element edges are straight and the element faces (in three dimensions) are planar, see Definitions (17.1) and (17.2) below.

Given a mesh, the goal is to create a new mesh resulting from a classical geometric transformation (symmetry, translation, rotation, isotropic or anisotropic dilation or a more general transformation explicitly defined by a transformation matrix).

If the transformation corresponds to a *positive isometry*, it only affects the position of the mesh points; thus both the connectivity and the numbering of the elements remain unchanged.

Otherwise, a transformation corresponding to a *negative isometry*, acts on the position of the mesh points and, in our situation, on the numbering (or connectivity) of the element vertices, in such a way that the resulting elements maintain positive surface areas (in two dimensions), or positive volumes (in three dimensions). To achieve this, a permutation of the list of the element vertices must be carried out (for example, a triangle with vertices $(1, 2, 3)$ must be transformed into the triangle with vertices $(1, 3, 2)$). The different transformations of interest are :

- symmetries with respect to a line or a plane,
- translations or shifts of a given vector,
- isotropic or anisotropic dilations about a given center, whose dilation coefficients are given,
- rotations of a given angle around a point or an axis (in three dimensions),
- or general transformations (given through their explicit matrices).

In addition, any combination of these operators defines a new transformation.

In practice, a transformation can be defined using a matrix $\mathcal{T}_{ra}$. Thus, if P is a vertex in the initial mesh, the corresponding vertex, P', is obtained as :

$$P' = \mathcal{T}_{ra}(P)\,. \tag{17.1}$$

This simply means that we have, in $\mathbb{R}^2$:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = [\mathcal{T}_{ra}] \begin{pmatrix} x \\ y \end{pmatrix},$$

with $\mathcal{T}_{ra}$ a 2×2 matrix (or more generally, a $d \times d$ matrix, in d dimensions).

Nevertheless, in order to give an explicit and easy description of the transformations considered, we use a coordinate system where these coordinates are homogeneous (*i.e.*, vertex with coordinates (x, y) is seen as the triple $(x, y, 1)$). The matrix is then a $(d+1) \times (d+1)$ matrix (regarding these matrices, one can consult [Rogers,Adams-1989]). Hence, in two dimensions :

$$\mathcal{T}_{ra} = \begin{bmatrix} 1+A^2F & ABF & ACF \\ ABF & 1+B^2F & BCF \\ 0 & 0 & 1 \end{bmatrix} \text{ with } F = \frac{-2}{A^2+B^2}$$

corresponds to a symmetry with respect to the line $Ax + By + C = 0$. While

$$\mathcal{T}_{ra} = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix}$$

is a translation of vector (T_x, T_y). A dilation of coefficients (α_x, α_y) whose center is (C_x, C_y) is characterized by

$$\mathcal{T}_{ra} = \begin{bmatrix} \alpha_x & 0 & C_x(1-\alpha_x) \\ 0 & \alpha_y & C_y(1-\alpha_y) \\ 0 & 0 & 1 \end{bmatrix}$$

and a rotation of angle α about a point $P = (P_x, P_y)$ is defined by

$$\mathcal{T}_{ra} = \begin{bmatrix} \cos\alpha & -\sin\alpha & P_x \\ \sin\alpha & \cos\alpha & P_y \\ 0 & 0 & 1 \end{bmatrix}.$$

In three dimensions, we have respectively (with similar notations)

$$\mathcal{T}_{ra} = \begin{bmatrix} 1+A^2F & ABF & ACF & ADF \\ ABF & 1+B^2F & BCF & BDF \\ ACF & BCF & 1+C^2F & CDF \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ with } F = \frac{-2}{A^2+B^2+C^2}$$

for a symmetry.

$$\mathcal{T}_{ra} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

for a translation and

$$\mathcal{T}_{ra} = \begin{bmatrix} \alpha_x & 0 & 0 & C_x(1-\alpha_x) \\ 0 & \alpha_y & 0 & C_y(1-\alpha_y) \\ 0 & 0 & \alpha_z & C_z(1-\alpha_z) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

for a dilation. For a rotation about the x-axis, we have :

$$\mathcal{T}_{ra} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ or } \begin{bmatrix} 1 & 0 & 0 & P_x \\ 0 & \cos\alpha & -\sin\alpha & P_y \\ 0 & \sin\alpha & \cos\alpha & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

depending on whether the rotation is made around the origin or an arbitrary point P. For the rotations about the other axis, we find similar expressions, *i.e.*, (with P at origin) :

$$\mathcal{T}_{ra} = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ et } \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

about the y and the z-axis respectively. Matrix $\mathcal{T}_{ra}$ defined by :

$$\begin{bmatrix} a^2 - Sd + CS_2g & ab - Se + CS_2h & ac - Sf + CS_2i & 0 \\ SC_2a + Cd + SS_2g & SC_2b + Ce + SS_2h & SC_2c + Cf + SS_2i & 0 \\ -S_2a + C_2g & -S_2b + C_2h & -S_2c + C_2i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

corresponds to a rotation of angle α, of axis $A = (A_x, A_y, A_z)$ around point $P = (0,0,0)$. In this matrix (cf. Figure 17.1), we have :

- if $A_x \neq 0$, $\phi = \arctan\left(\dfrac{A_y}{A_x}\right)$, $\theta = \arctan\left(\dfrac{A_z}{\sqrt{A_x^2 + A_y^2}}\right)$, $C = \cos\phi$ and $S = \sin\phi$, $C_2 = \cos\theta$ and $S_2 = \sin\theta$,
- otherwise, if $A_y \neq 0$, $\theta = \arctan\left(\dfrac{A_z}{\sqrt{A_x^2 + A_y^2}}\right)$, $C = 0$, $S = 1$ and $C_2 = \cos\theta$, $S_2 = \sin\theta$,
- otherwise $C = 0$, $S = 1$ and $C_2 = 0$, $S_2 = 1$,

where

$$a = C_2C\,, \qquad b = C_2S\,, \qquad c = -S_2 \quad \text{and} \quad d = -\cos\alpha S - \sin\alpha S_2C\,,$$

$$e = \cos\alpha C - \sin\alpha SS_2 \quad \text{and} \quad f = -\sin\alpha C_2 \qquad \text{and finally}$$

$$g = -\sin\alpha S + \cos\alpha S_2C\,, \qquad h = \sin\alpha C + \cos\alpha SS_2 \quad \text{and} \quad i = \cos\alpha C_2\,.$$

As an exercise, we now check that the above matrix is nothing other than the combination of the matrices

$$\mathcal{T}_{ra}(\phi, \vec{Z}) \circ \mathcal{T}_{ra}(\theta, \vec{Y}) \circ \mathcal{T}_{ra}(\alpha, \vec{X}) \circ \mathcal{T}_{ra}(-\theta, \vec{Y}) \circ \mathcal{T}_{ra}(-\phi, \vec{Z})$$

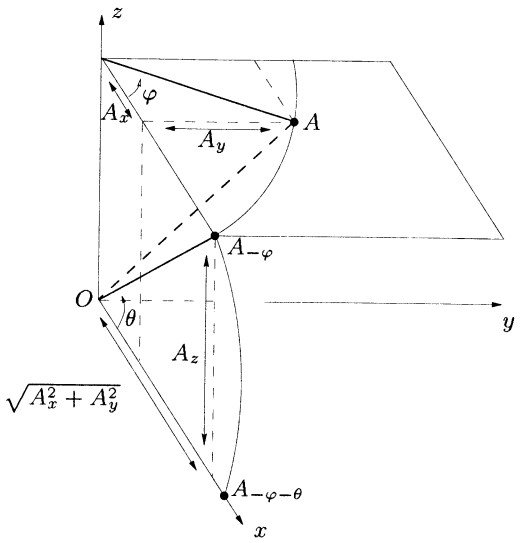

Figure 17.1: *Rotation about axis $A = (A_x, A_y, A_z)$. The angles ϕ and θ enable us to return to a rotation in terms of α about the x-axis.*

where $\mathcal{T}_{ra}(angle, vecteur)$ stands for the matrix related to the rotation of angle *angle* and axis *vecteur*.

We consider the case where $A_x \neq 0$. Then, writing only the sub-matrices corresponding to the rotation, the above expression leads to computing the product of the five following matrices :

$$\begin{bmatrix} C & -S & 0 \\ S & C & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C_2 & 0 & S_2 \\ 0 & 1 & 0 \\ -S_2 & 0 & C_2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\,\alpha & -s\,\alpha \\ 0 & s\,\alpha & c\,\alpha \end{bmatrix} \begin{bmatrix} C_2 & 0 & -S_2 \\ 0 & 1 & 0 \\ S_2 & 0 & C_2 \end{bmatrix} \begin{bmatrix} C & S & 0 \\ -S & C & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

using the above notations (where $c\,\alpha$ is short for $\cos\alpha$ and $s\,\alpha$ is short for $\sin\alpha$). It is then easy to write this operation as follows :

$$T_1\ T_2\ T_3\ T_4\ T_5\,.$$

Now, we compute the product $T_1\,T_2$, thus :

$$\begin{bmatrix} CC_2 & -S & CS_2 \\ SC_2 & C & SS_2 \\ -S_2 & 0 & C_2 \end{bmatrix}.$$

We note the product $T_3\,T_4\,T_5$ by :

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}.$$

And we return to the above matrix T_{ra} after completing the product of these two matrices. To end, we express $T_3\,T_4\,T_5$ as a function of T_3 and the product $T_4\,T_5$, thus leading to the coefficients a, b, ..., i. □

Remark 17.1 *The center of rotation is supposed to be at the origin; to return to a general situation, it corresponds to adding the translation which takes into account the adequate point $P = (P_x, P_y, P_z)$ This comes down to applying first a translation of vector ${}^t(-P_x, -P_y, -P_z)$ and, after applying this to the result of above matrix T_{ra}, to applying the opposite translation.*

Remark 17.2 *In the case of a rotation, the definition of angles ϕ and θ by an arctangent gives a determination at modulo π. When programming such an operator, this ambiguity must be removed using the sine and cosine of the angles in such a way that they are well determined. Thus, we compute the coefficients C, S, C_2 and S_2 directly according to A_x, A_y et A_z to avoid this indetermination.*

17.1.2 Local and global refinement

We briefly discuss methods resulting in a global or a local refinement in a mesh.

Global refinement method. Provided with a global subdivision parameter, n, a global refinement method involves a "uniform" partition of all elements in the mesh. Each element is then split into n^d (d being the spatial dimension) elements with the same geometric nature. One could note that splitting triangles, quads, as well as hexahedra or pentahedra results in *similar* elements. On the other hand, the same method results in *non similar* sub-meshes in the case of tetrahedral elements.

A global refinement method can be used both for mesh construction and for analyzing the quality of a solution. In the first case, a coarse mesh is constructed which is then subdivided so as to achieve a fine enough mesh. In the second case, repeated global refinements is one way to check some convergence issues about the solutions computed at the different levels. Note that such a method rapidly leads to large meshes (in terms of the number of elements). In this respect, the different refined meshes can be used to compare a mesh adaptation method with variable and local stepsizes with a "reference" solution computed on a uniform fine mesh.

Remark 17.3 *A particular process must be carried out when the element under partition has a boundary edge (face). This must be subdivided by taking into account the geometry of the boundary.*

Local refinement method. Unlike the previous global method, a local refinement method allows the creation of meshes with variable densities (in terms of

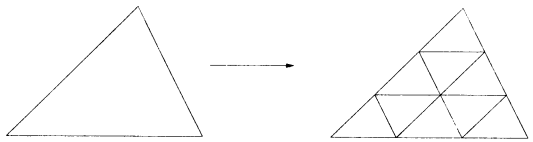

Figure 17.2: *Global partitioning of a triangle for $n = 3$.*

element or vertex spacing). Thus some extent of flexibility is obtained. Such a method can serve as a way to achieve some adaptive features and can therefore be used to deal with adaptive processes (as will be seen in Chapter 21 for example).

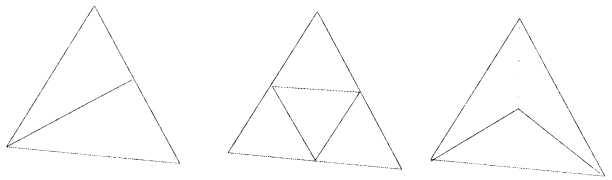

Figure 17.3: *Some local possibilities for splitting a triangle.*

In practice, a list of vertices is given around which a refinement is expected. The way in which this list is defined depends on the criteria used in the computational process. For instance, edges in the current mesh which are judged too long can be defined and their endpoints can be put into the above list of vertices. Then, the elements are split according to the local situation. Figure 17.3 shows an example where a point is created along one edge (left-hand side), three points are defined as edge midpoints (middle) and a point is inserted inside the initial triangle (right-hand side).

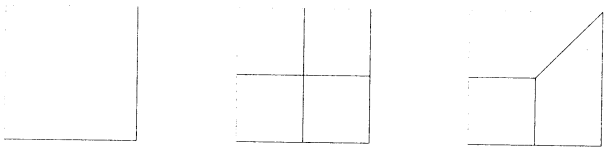

Figure 17.4: *Several local partitionings of a quad.*

Figure 17.4 considers the case of a quad. The initial quad (left-hand side) is split into four sub-quads by introducing the edge midpoints (middle) or into three sub-elements by introducing two points along two consecutive edges (right-hand side). Figure 17.5 illustrates a tet example. One point is created along one edge (left-hand side), one point is created in a face (middle) while a point is created inside the initial tet (right-hand side).

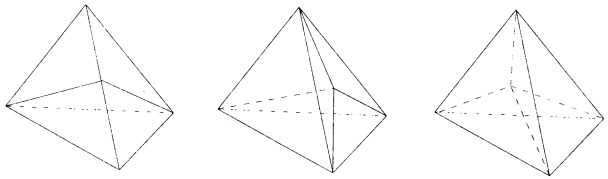

Figure 17.5: *Several local partitionings of a tet.*

Remark 17.4 *The important point is to ensure a good quality of the resulting elements, which means seeing whether the shape of the initial elements is preserved or altered by the local process.*

Remark 17.5 *Another topic to be addressed is that of maintaining a conforming mesh. To this end, elements in some neighborhood of the elements which are refined must be considered as candidates for some extent of refinement.*

Numerous theoretical works can be found on how angles, shapes, etc., are preserved or not when applying local splitting (see, for instance, [Rivara-1984b], [Rivara-1990], [Rivara-1997]).

The reader is also referred to Chapter 21 for more details about local mesh modification tools and to Chapter 8 where some other possibilities for local splitting are given.

17.1.3 Type conversion

Geometric type change of the elements in a mesh proves to be useful in various contexts. A quad element can be split by means of triangles using three different patterns (Figure 17.6). A triangle may be split into three quads (same figure) but the resulting quads may be poorly-shaped (see Chapter 18 regarding quality measures for both triangles and quads).

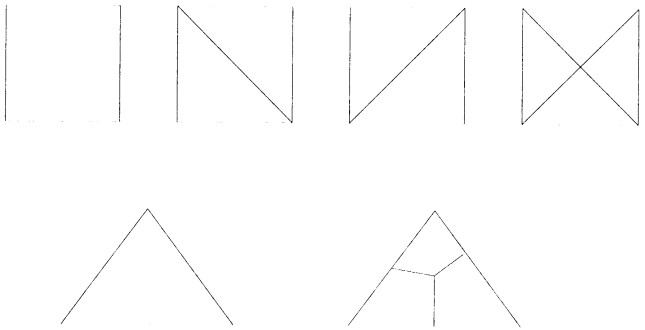

Figure 17.6: *A triangle leading to three quads (bottom) and a quad leading to two or four triangles (top).*

Similar change type operators can be formally defined for three-dimensional elements. The above transformation of a triangle into quads extends to three dimensions which allows a tet to be split by means of hexahedra. Nevertheless, as will be seen in Chapter 18, the degree of the vertices in the resulting mesh is very poor. Pentahedra can be split using tets (actually three tets are required). Hexahedra can be split into tets using five or six tets (Figures 18.3 and 18.4).

Remark 17.6 *There are 74 possibilities for remeshing a given hex into tets, with two solutions for a five-element splitting and 72 for a six-element pattern.*

Exercise 17.1 *Find this number of splitting (Hint : find all the types of tet one can construct based on the hex vertices (58 cases are encountered), then, examine all the possible valid pairs, Suppress the pairs of tets separated by a face plane, look at the case where a face plane is common to two tets and show the 72 cases where the face is really a common face).*

17.2 Merging two meshes

Various other mesh modification or mesh post-processing are also of interest. Among these is a process for merging two given meshes into a single one when these two initial meshes share a boundary region.

Merging two meshes into a single one is frequently necessary. Several points must be taken into account including how to find the common entities (vertices, edges and faces), in spite of the round-off errors, and how to complete this task quickly. Another question is to find a new numbering (labeling) for the vertices (resp. elements) in the resulting mesh.

Before going further, let us give a more precise formulation of the merging problem. Let Ω_1 and Ω_2 be two domains in two or three dimensions and let $\mathcal{T}_1$ and $\mathcal{T}_2$ be the two corresponding meshes. We assume that these two domains share a common boundary region. We also assume that these meshes have been produced by any of the suitable mesh generation methods in an independent manner. Nevertheless (at least irrespective of the round-off errors) the meshes of the interface regions are assumed to be strictly identical. This mesh is in $d-1$ dimensions and thus consists of segments ($d=2$) or of faces ($d=3$).

If $\mathcal{T}_1 \cap \mathcal{T}_2$ stands for the mesh of the above interface, the merging problem consists of combining the vertices and the elements in $\mathcal{T}_1$ and in $\mathcal{T}_2$ in such a way as to obtain a single mesh $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$. This question can be subdivided into three sub-problems :

- a problem of a metric (geometric) nature which consists of identifying the entities (vertices, edges and faces) common to the two meshes,
- a problem of vertex numbering that leads to finding an index (a label) for all the vertices in the mesh resulting from the merging process. This involves assigning, in a consistent manner, a numbering system to the three types of vertices that are met in the resulting mesh, say the former vertices in $\mathcal{T}_1$ not common with $\mathcal{T}_2$, those in $\mathcal{T}_2$ not common in $\mathcal{T}_1$ and, finally, those in $\mathcal{T}_1 \cap \mathcal{T}_2$, the interface,
- and a problem of element numbering that consists of finding a global numbering system in the resulting mesh.

As assumed, the points in the interface are supposed identical (*i.e.*, this interface is meshed in a similar way once the two initial meshes have been constructed). Nevertheless, in practice, and due to the potential round-off problems, these points which are assumed to be identical are in fact identical for a given tolerance threshold. Thus, a searching technique with an accuracy of ε must be used. The common entities being identified, it remains to find the numbering of both the vertices and the elements. Let $num(P)_{\mathcal{T}_i}$ be the index (the label or the number) of vertex P in $\mathcal{T}_i$ for $i = 1, 2$. The problem becomes one of finding these values for the entire set of vertices whereas the element labeling problem is obvious.

All these points will be discussed below, but first we discuss how to find the common entities and then we turn to the numbering issues.

Finding the common vertices (geometric point of view). Here, given a vertex in $\mathcal{T}_1$ also included in $\mathcal{T}_1 \cap \mathcal{T}_2$, we discuss how to find the corresponding vertex in $\mathcal{T}_2$ (and conversely). Using a hashing technique (Chapter 2) has proved very efficient for this task. This operation consists of several stages :

- Constructing a hashing table based on mesh $\mathcal{T}_1$, the first mesh.

The two initial meshes are enclosed[1] in a quadrilateral "box" (hexahedral box in three dimensions). The box dimensions Δ_x, Δ_y and, in three dimensions, Δ_z are computed as well as $\Delta = \max(\Delta_x, \Delta_y, \Delta_z)$, see Figure 17.7. Given ε, a threshold tolerance, we define a sampling stepzize δ as :

$$\delta = 2\,\varepsilon\,\Delta\,,$$

which is the *resolution power* of the method.

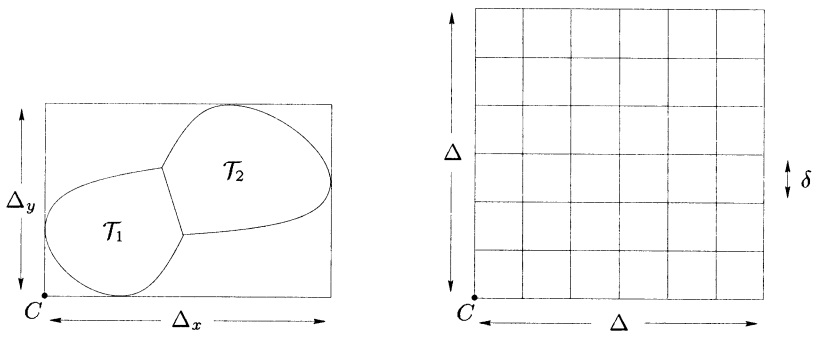

Figure 17.7: *The box enclosing the two initial meshes and the virtual grid with the parameter values associated with the merging algorithm.*

A grid with boxes (cells) of size δ is defined[2] which formally discretizes the space. The vertices in $\mathcal{T}_1$, the first mesh, are classified with respect to the grid boxes. To this end, if x, y and z are the coordinates of a given point P, we associated with this triple a triple of integers i, j and k defined as $i = \dfrac{x - x_{min}}{\delta}$, $j = \dfrac{y - y_{min}}{\delta}$ and $k = \dfrac{z - z_{min}}{\delta}$, where x_{min}, x_{min} and z_{min} are the coordinates of the bottom left corner of the introduced box.

Then, a hashing function is used which associates a hashing value with the triple i, j, k (and thus with point P). For example, $Hfunc(i, j, k) = i + j + k \;\; modulo(mod)$, where mod is a given integer, is a possible hashing function (see also Chapter 2).

This key serves as an entry point in a linked list containing the points of mesh $\mathcal{T}_1$. For example, let $Hfunc$ and $Link$ be the two above "tables", for a given point

[1] Note that it is also relevant to define the enclosing box as the box based on only one mesh (for instance, the smaller mesh, in terms of the number of vertices), this box being slightly dilated.

[2] In practice, this grid is not actually constructed.

P with coordinates x, y, z, we compute $key_{hach} = Hfunc(P) = Hfunc(i, j, k)$, then, if $Link(key_{hach}) \neq 0$, $Q_1 = Link(key_{hach})$ is the index of the first point related to P while the other points (Q_l, if any) related to P are obtained via the sequence :

```
l ← 1, key_hach ← Q_1
REPEAT key_hach = Link(key_hach)
   IF key_hach ≠ 0,
      l ← l + 1,
      Q_l ← key_hach
   OTHERWISE, END.
```

Then, this material[3] is used to find the vertices in the second mesh which are common to the vertices in the first mesh.

Remark 17.7 *In some geometric situations, the virtual enclosing grid can be defined as the intersection of the two enclosing boxes associated with the two initial meshes in which the interface necessarily falls. Depending on the relative positions of the initial meshes, the virtual grid is significantly reduced and thus the corresponding parameters are better adapted.*

Now, we can proceed to the next stage of the process.

- Analysis of mesh $\mathcal{T}_2$ with regard to the above hashing table.

Once all the points in $\mathcal{T}_1$ have been processed, we consider the vertices in $\mathcal{T}_2$. Given point P in $\mathcal{T}_2$, we compute[4] $key_{hach} = Hfunc(P)$, the value associated with P, *i.e.,* the equivalent of the box in which P falls. If x, y are the coordinates of P (for the sake of simplicity, in what follows, we assume a two-dimensional problem), we also compute the keys associated with the virtual eight points whose coordinates are $(x, y \pm shift)$, $(x \pm shift, y)$, $(x - shift, y \pm shift)$ and $(x + shift, y \pm shift)$ where $shift = \frac{\delta}{2} = \varepsilon\,\Delta$ (in fact, a value smaller than this theoretical value must be used, for instance, instead of $shift = 0.5\,\delta$, it is advisable to take $shift = 0.499\,\delta$).

Due to the value of $shift$, only four different keys can be found (Figure 17.8). Then, let key_{hach} be one of these entry points, we just have to visit table $Hfunc$ in parallel with table $Link$ to find the vertices of $\mathcal{M}_1$ which are "close" to point P under examination.

Let Q be such a point, then if $\max(|x_P - x_Q|, |y_P - y_Q|) \leq \delta$, the pair (P, Q) is a candidate pair of common points, otherwise, table $Link$ is visited to find any other possible candidates.

Remark 17.8 *In this algorithm, we can stop once the first candidate pair has been obtained or we can continue in order to check if there exist more than one point in $\mathcal{T}_2$ which forms a candidate pair. In such a case, a natural idea is to pick the closest point as the solution.*

[3] Another use of this hashing is to find the points in $\mathcal{T}_1$ which are "close" to a given point.

[4] If the grid is based on the first mesh only, it is necessary to check that P falls within the grid before computing its key, otherwise, point P is not considered.

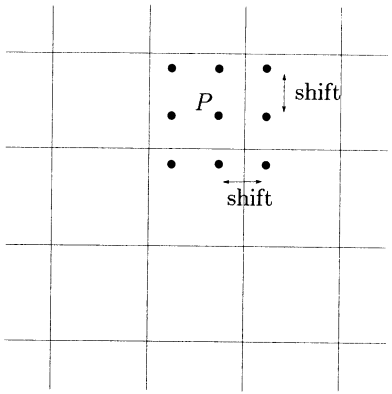

Figure 17.8: *The eight virtual points result in four different hashing keys.*

Remark 17.9 *Following the previous remark, due to round-off errors or due to the more or less strict coincidence of the "common" boundaries, it is possible to find several points in $\mathcal{T}_2$ which are identified with only one point in $\mathcal{T}_1$, see Figure 17.9. To prevent such a problem, we consider the pairs of "identical" points resulting from the merging of $\mathcal{T}_1$ with $\mathcal{T}_2$ along with those resulting from the merging of $\mathcal{T}_2$ with $\mathcal{T}_1$. Then, the pairs which are in both lists are solutions.*

Finding the common entities. Common geometric vertices are obtained using the previous algorithm. Nevertheless, merging two meshes requires finding not only the common vertices but the common edges and faces as well. In addition, the merge can be based on a more general definition of the common entities involving not only the geometric point of view.

- Pure geometric merging process.

In this case, the common vertices are determined using the above method. Then, common edges are those whose endpoints are common while common faces are those whose vertices are common. Exhibiting both the common edges and faces and not only the common vertices allows for an easy check of the Euler formula (Chapter 1) thus providing a way to guarantee the correctness of the resulting mesh.

- More sophisticated merging process.

In this case, we consider both the geometric aspect and the physics of the problem. Including the physics in a mesh (see Chapter 1) involves associating some physical attributes (integer values, for instance) with the mesh entities (vertices, edges and faces). Then two vertices are said to be common if they are geometrically common and, in addition, if their two physical attributes are identical. Similarly, common edges and faces are defined by taking into account the physics. For instance,

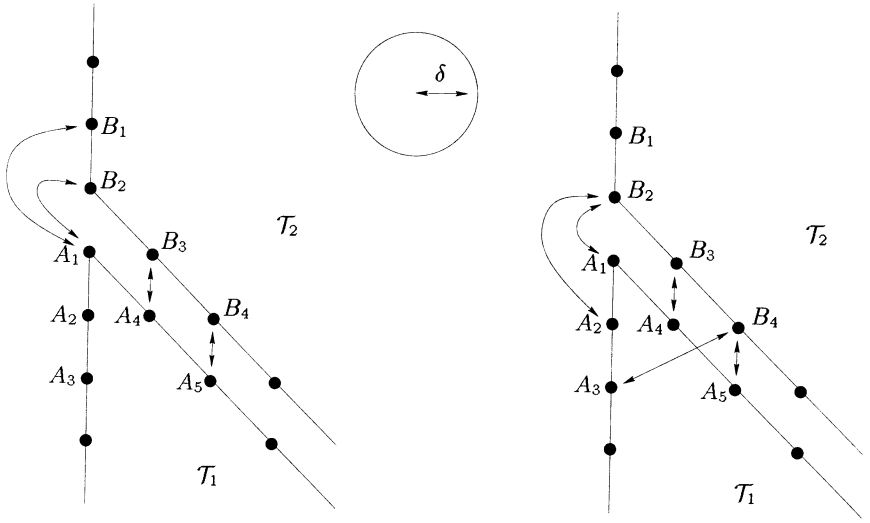

Figure 17.9: *Merging $\mathcal{T}_1$ and $\mathcal{T}_2$ results in the pairs $(B_1, A_1), (B_2, A_1), (B_3, A_4), (B_4, A_5), \ldots$ (left-hand side) while merging $\mathcal{T}_2$ and $\mathcal{T}_1$ produces the pairs $(A_1, B_2), (A_2, B_2), (A_3, B_4), (A_4, B_3), (A_5, B_4), \ldots$ (right-hand side). Then the points which are judged coincident are A_1 and B_2, A_4 and B_3, A_5 and B_5, etc. Note that, for clarity, the two meshes have been slightly shrunk, in fact the common boundaries are assumed to be coincident or, at least, very close.*

introducing physical considerations is a simple way to define cracks (in fact, two edges located at the same location exist at a given time and, due to the nature of the problem under consideration, are then clearly different : a crack appears.)

Numbering issues. This point may concern two aspects, element numbering and vertex numbering. Actually, depending on how the mesh data structure is defined, the mesh elements have an explicit numbering (or index) or are simply sequentially enumerated (meaning that the first element is that located at the beginning of the "table" which stores them). Whatever the case, the elements in mesh $\mathcal{T}$ are formed by the elements in mesh $\mathcal{T}_1$ sequentially followed by those of mesh $\mathcal{T}_2$. On the other hand, vertex numbering must receive some explicit attention.

Let n_1 (resp. n_2) be the number of vertices in $\mathcal{T}_1$ (resp. $\mathcal{T}_2$) and let $num(P)_{\mathcal{T}_1}$ the index[5] of vertex P in mesh $\mathcal{T}_1$, the aim is to define $num(P)_{\mathcal{T}}$ for all any point P in $\mathcal{T}$. The points in $\mathcal{T}$ are classified into three categories, those which are points of the former mesh $\mathcal{T}_1$, those which are common to $\mathcal{T}_1$ and $\mathcal{T}_2$, the interface, and, finally, those which are points of the former mesh $\mathcal{T}_2$ not common to the interface.

Once the common entities have been established, the common points are known.

[5] Indeed, in general, $num(P)_{\mathcal{T}_1} = P$.

In other words, we know a correspondence table like

$$P \in \mathcal{T}_2 \iff Q \in \mathcal{T}_1 ,$$

for all the points in the interface. Then, the following algorithm can be used to find the vertex numbering :

$num(P)_{\mathcal{T}} = 0$ for all $P \in \mathcal{T}_2$
$n \leftarrow n_1$,
IF $P \in \mathcal{T}_1$
  $num(P)_{\mathcal{T}} \leftarrow num(P)_{\mathcal{T}_1}$
IF $P \in \mathcal{T}_2$
  IF $P \in \mathcal{T}_1 \cap \mathcal{T}_2$
    $num(P)_{\mathcal{T}} \leftarrow num(Q)_{\mathcal{T}_1}$ where $P \in \mathcal{T}_2 \Longleftrightarrow Q \in \mathcal{T}_1$
  OTHERWISE IF $num(P)_{\mathcal{T}} = 0$
    $n \leftarrow n + 1$
    $num(P)_{\mathcal{T}} \leftarrow n$.
  END IF
END IF

In fact, initializing this table from $n_1 + 1$ to $n_1 + n_2$ ensures an adequate initialization (since $n_1 + n_2$ is larger than the number of vertices in the final mesh). Using the initialization stage of this algorithm allows us to number the vertices by visiting the elements in the global mesh. Indeed, $num(P)_{\mathcal{T}} = 0$ means that the index of vertex P must be defined while $num(P)_{\mathcal{T}} \neq 0$ means that the index of P has been already established (P is a member of the (former) first mesh or P is in the interface or, finally, P has already been labeled when processing an element prior to the current element).

Remark 17.10 *The vertex numbering is* a priori *not optimal (in any sense), especially, merging $\mathcal{T}_1$ and $\mathcal{T}_2$ results in a different numbering to merging $\mathcal{T}_2$ and $\mathcal{T}_1$. Thus, a renumbering algorithm can be necessary to optimize the final mesh.*

Remark 17.11 *The merging of two meshes, a very natural and frequent operation in mesh generation packages, can be seen in a very different way. In fact, based on the methodology used to define the mesh of the entire domain by partitioning this domain into regions where a mesh generator is used, the regions common to two (local) meshes are formed by a set of two lines or two surfaces that have been explicitly defined beforehand. Therefore, the interfaces are a priori known, i.e., not to be sought in one way or another. In fact, the storage of this information in a mesh data structure is not trivial, and consequently we must regenerate them by computation. This apparent weakness is compensated for, on the one hand, by the efficiency of the merging algorithm and, on the other hand, by the resulting modularity in the sense that merge formally becomes independent of the design of the local meshes.*

To conclude, one should note that, in terms of CPU, the hashing technique used to find the coincident vertices has proved to be rather inexpensive. Thus the computational effort necessary to merge two meshes is mainly due to the I/O steps of the process.

17.3 Node creation and node labeling

Both the creation and the adequate numbering of finite element nodes is a crucial task when finite elements other than P^1 are needed (a section, at the end of the chapter, will come back to this question which is also discussed in Chapter 20). In this section we discuss how to assign a number (an index) to the nodes of a given mesh. We briefly consider this numbering issue at the mesh generation step and we turn to the case of meshes whose nodes do not reduce to their vertices. Renumbering issues (for optimization purposes) will be discussed in the next section.

17.3.1 Vertex and element labeling

As discussed in many chapters, numerous mesh generation methods exist which complete a so-called P^1 mesh (see Definition (17.2), below). Apart from some specific methods which result in a specific vertex (element) numbering, there is no particular reason for the vertex (element) numbering obtained should be optimal. The creation of a mesh with nodes other than the mesh vertices leads to the same conclusion since the sole aim of the node numbering (discussed in the following of this section) is to assign a number to the mesh nodes without any concern about the optimality of this numbering.

17.3.2 Node creation

To introduce the problem, we discuss how to define P^2 finite elements. As previously seen, mesh generation algorithms implicitly construct P^1 meshes. To make this notion precise, we first review Definition (1.16) in Chapter 1 :

Definition 17.1 *A node is a point supporting one or several unknowns or degrees of freedom (d.o.f).*

A formal definition of a P^1 type mesh is as follows :

Definition 17.2 *A P^1 mesh if a mesh whose geometry is P^1, meaning that the element edges are straight and, in three dimensions, the element faces are planar, and whose nodes are reduced to its vertices.*

Every mesh vertex is then considered as a node and the geometric elements[6] are actually the finite elements (once the basis functions are defined, Chapter 20).

In this section, we assume that a mesh implicitly of a P^1-type is given and we wish to define a P^2-type mesh. We assume in addition that the given mesh is *compatible* in a sense that will made clear hereafter, with respect to a P^2 interpolation. The problem is then to define the "midpoint" nodes along the boundary edges (for a problem in two dimensions), Figure 17.10, as the extra nodes required for the interior edges are trivial to obtain. Boundary nodes must be properly located on the boundary and thus we need to know which edge is a boundary edge and, for

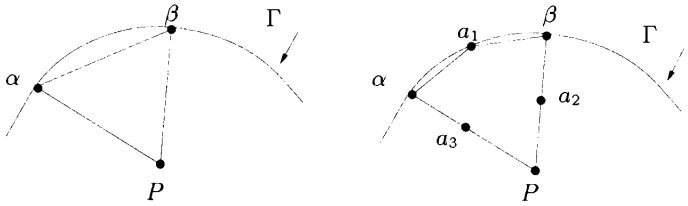

Figure 17.10: *Construction of a P^2 finite element. An element resulting from the mesh generation method previously used is shown on the left where Γ denotes the boundary domain. The corresponding P^2 element is depicted on the right-hand side. This six node triangle includes as nodes the three vertices P, α, β together with the above three edge "midpoints" a_i. Actually, for a straight six node triangle, the "midpoints" are the edge midpoints while for a curved (isoparametric) six node triangle, these a_i's must be properly located on Γ (for a boundary edge).*

a boundary edge, we also need to have a suitable representation of the geometry, so as to permit the proper location of the new node.

Hence, two problems must be addressed. First, how to find a proper location for the nodes (other than element vertices). Second, how to find a global numbering of the nodes.

17.3.3 Node construction and $p-$compatibility

The most used finite elements (see below) involve either edge midpoints as nodes, several points along every element edge and/or some points inside the element. In three dimensions, nodes may be defined along element edges, element faces or inside the elements. Note also that element vertices may or may not be nodes.

Definition 17.3 *A given mesh, assumed to be P^1 is termed p-compatible if finite elements of degree p can be constructed resulting in a valid mesh.*

Actually a mesh is p-compatible if all of its elements are p-compatible. The element depicted in Figure 17.10 is 2-compatible.

At this stage, the p-compatibility is assumed and remains purely intuitive. Let us simply consider that edges close enough to a curved boundary Γ result in a p-compatible mesh. This will be discussed with more detail in Chapters 20 and 22 where conditions ensuring this property will be developed. In fact, we have introduced this notion to make the node numbering problem consistent (we assume that creating these nodes poses no problem).

17.3.4 Node labeling

When the nodes are created, it is necessary to find a suitable numbering system for them both at the element level (*local numbering*) and at the mesh level (*global numbering*). At the element level, one needs to find a number (an index) for a given node. At the mesh level, one needs to associate a global index with the above local indices. Furthermore, it could be of great interest to optimize the resulting node numbering (see below).

[6] *I.e.*, the elements constructed by any mesh generation method.

Nodes are created along the mesh edges, in the mesh faces or inside the mesh elements based on the type of the finite element which must be defined. In the case where a constant number of nodes is defined for a given entity, the index of a node can be easily related to the index of the underlying entity. On the other hand, when the number of nodes varies for a given entity, finding a node index requires more subtle attention.

- Local numbering.

The definition of a finite element includes the definition of its nodes (see Σ_K hereafter). In other words the local numbering of the nodes is explicitly known. For example, the P^2 triangle consists of six nodes. The first three nodes are the three vertices while the mid-edge nodes form the nodes 4, 5 and 6 of the element. Usually, the first nodes are the vertices and they follow their local numbering while the others, if any, are sequentially numbered based on their supporting entities (the edges, then the faces and, finally, (the interior of) the element).

- Global numbering.

Given a finite element, *i.e.*, its node list, the aim is to find a global node numbering over the entire mesh. In practice, at the computational step, the elements are considered one at a time and the correspondence (let us consider again the six node triangle)

$$[1,2,3,4,5,6] \Longrightarrow [n_1, n_2, n_3, ..., n_6],$$

is used to perform the necessary calculation. Indeed, the labels n_1, n_2 and n_3 are known (as a result of the mesh generation step which completes a P^1 mesh) and the numbering problem reduces to finding the three remaining labels. To this end, we can construct the list of the mesh edges, assign a number to any edge while maintaining the relationship between this edge number and the element numbers. Then, fix $num = np$, where np is the number of mesh vertices and we use the following algorithm :

```
FOR i = 1, ne, ne being the number of elements
   One visits the edges of element i in the mesh.
   If the examined edge has not been previously visited
      (while visiting an element with index i less than the current index),
      one assigns the index of the known node to the edge node
      one proceeds to the next edge.
   OTHERWISE
      one sets num = num + 1
      one associates num with the edge and
      one takes num as the index of the node in the examined edge.
   END IF
END FOR
```

For a finite element with more than one node on a given edge, the previous scheme must be slightly modified. Let k_n be the (constant) number of nodes per edge, then the following algorithm must be used :

```
FOR i = 1, ne
One visits the edges in the mesh.
If the examined edge has not been previously visited
   one uses the index of the edge to find the various indices
   for the nodes related to this unique index
   one assigns these indices to the nodes of the edge when considering
   it as indicated above based on the indices of its endpoints.
   one proceeds to the next edge.
OTHERWISE, IF j is the edge index,
   then the values num_1 = np + k_n(j-1) + 1, ...,
   num_{k_n} = np + k_n(j-1) + k_n = np + k_n j are the node indices
   for this edge.
   one considers the endpoint indices, i_1 and i_2 and
   IF i_1 < i_2
      one assigns num_1, num_2, ..., num_{k_n} to the edge nodes
      while seeing it from i_1 to i_2
   IF i_1 > i_2
      one assigns num_1, num_2, ..., num_{k_n} to the edge nodes
      now seen from i_2 to i_1,
END FOR
```

For a finite element having some nodes on their faces (other than the nodes located on the face edges), the previous algorithm allows the face node numbering. An index is assigned to the face and one has just to take care how the corresponding node labels are assigned so as to insure a consistent numbering for a face shared by two elements. Again, comparing the labels of the face vertices is a solution to complete a consistent node numbering. Nodes inside an element are numbered according to the same principle.

Remark 17.12 *In this discussion we have assumed a constant number of nodes for each entity. If this number varies, the global index is not obtained by a simple formula as above but by adding one to the previous index used (note that the relationship between the entity index and the (first) node index is also required).*

17.3.5 Some popular finite elements

Due to node definition and numbering problems, we recall here some of the finite elements that are, for the most part, extensively used in the finite element computational processes commonly included in available software packages.

A finite element is fully characterized by a triple (see Chapter 20 for more details) :

$$[K, P_K, \Sigma_K],$$

where K is a mesh element, Σ_K is the set of degrees of freedom defined over K and P_K denotes the basis polynomials (commonly termed as the shape functions) on K.

In terms of geometry, we are mostly interested in what K should be. In terms of node definition, we have to focus on Σ_K. The other point, the basis polynomials, does not fall within the scoop of this book (the reader is referred to the ad-hoc literature).

About the geometry. Let us consider a triangle. The geometry of the three node triangle (namely the Lagrange P^1 finite element) is fully determined by the three vertex elements. Similarly, the geometry of the straight six node triangle (Lagrange P^2 element) is characterized by the three element vertices while the geometry of the curved (isoparametric) six node triangle is defined using the three element vertices along with the three edge "midpoints". Thus, in the defining triple, K is the triangle resulting from the mesh generator or this triangle enriched by three more points.

About the node definition. The nodes are the support of the degrees of freedom. A given node may support one or several degrees. There are various types of degrees. The most simple finite elements, namely *Lagrange type* elements, involve as degrees the value(s) of the unknown(s) which is (are) computed. For instance, depending on the physical problem, the pressure, the temperature, the displacements, the stresses, the velocity, the Mach number, etc., could be the unknown functions. Then, a temperature leads to one degree, its value, while a velocity is associated with two (three) degrees (one degree for one direction). More sophisticated finite elements, such as *Hermite type* elements, involve as degree the value of the unknown function together with some of its derivatives. Also *beam*, *plate* and *shell* elements use various derivatives as degrees of freedom.

Now, given an element, the nodes can be located at various places. "Poor" elements have only their vertices as nodes. Other elements use in addition some nodes located along their edges, on their faces or inside them. Note also that there exist elements whose vertices are not nodes.

In addition, finite elements exist where several types of nodes are encountered for which the number of degrees of freedom varies.

Examples of popular finite elements can be found in Chapter 20.

17.4 Renumbering issues

As a consequence of vertex, node or element numbering problems, another issue of interest concerns the vertex (node) or element renumbering methods that allow the minimization, in some sense, of the resulting matrices (*i.e.*, based on the created mesh) used at the solution step of a computational process.

Renumbering concerns the solution methods that will be used subsequently. The aim is both to minimize the memory resources needed to store the matrix associated with the current mesh and to allow an efficient solution step for some solution methods where a matrix system must be considered.

17.4.1 Vertex renumbering

Various methods exist for mesh vertex (element) renumbering purposes. In what follows, we discuss in some detail one of the possible methods, namely the so-called Gibbs method ([Gibbs *et al.* 1976a]), and a variation of it.

For the sake of simplicity, we consider P^1-type meshes and the case of arbitrary meshes is discussed afterwards. The basic scheme of this method can be summarized by the three following steps :

- the search for a good *starting point* to initialize the algorithm (following [A.George,Liu-1979]),
- the optimization of the numbering *descent*,
- the numbering itself based on the previous material using the Cuthill MacKee algorithm ([Cuthill,McKee-1969]).

In order to describe the method, we need to introduce some notations and definitions.

The mesh under consideration is $\mathcal{T}$, P_i is a mesh vertex and np and ne are the number of vertices and the number of elements, K_j is a mesh element. In terms of graphs, the vertices are also referred to as the *nodes*. Two such nodes are said to be neighbors if there exists an element in $\mathcal{T}$ with these two nodes as vertices (or nodes, in terms of finite element nodes). The *degree* of a node P, $deg(P)$, is the number of its neighboring nodes. The *graph*, Gr, associated with $\mathcal{T}$ indicates the connections between the nodes (Figure 17.11). Note that a graph may contain one or more connected components, the connected component number k being denoted C_k. The neighbors of node P constitute $\mathcal{N}_1(P)$ level one of its *descent*. The neighbors of the nodes in $\mathcal{N}_1(P)$, not yet referenced, form $\mathcal{N}_2(P)$, level two of the descent of P and so on. The *descent* of node P, $\mathcal{D}(P)$ is the collection of the levels $\mathcal{N}_k(P)$. The *depth* of P, $d(P)$, is the number of levels in $\mathcal{D}(P)$.

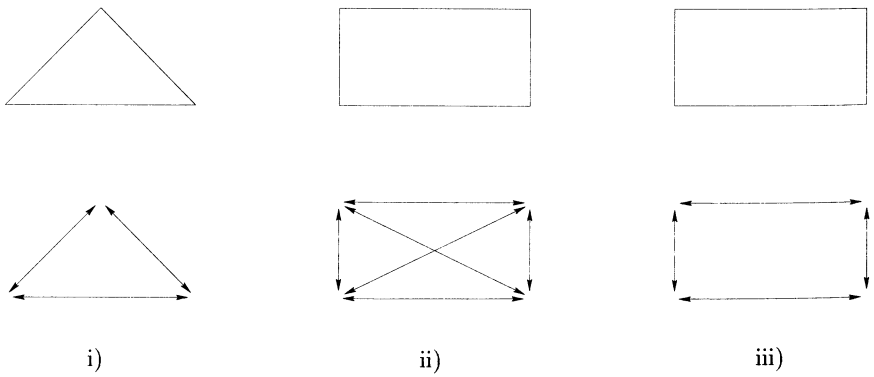

Figure 17.11: *Connections from nodes to nodes for a triangle i) and a quadrilateral ii). All the nodes of the element interact. For a quadrilateral iii), a reduced graph can be constructed by omitting the connections corresponding to the diagonals.*

First, we establish the neighboring relationships from element to element. At the same time, the degrees of the mesh nodes are found. Then, several steps are performed :

- Step 1 : an initialization step corresponding to the following algorithm :

```
Pick in T the node P of minimal degree deg(P)
(1) define D(P), the descent of P,
consider the last level in this set, i.e., N_k(P),
select the node Q of minimal degree.
form D(Q) and compare D(P) with D(Q),
IF d(Q) > d(P)
   replace P by Q
   RETURN to (1)
OTHERWISE
   the initialization is completed.
END IF.
```

This step merits some comments. Actually, the initial guess P is the point with minimal degree. Indeed, P must be a boundary point but this way of processing avoids explicitly establishing the list of the boundary points while leading to the desired result after very few iterations.

On completion of the initialization step, we have found PQ the *pseudo-diameter* of the renumbering graph which now serves to pursue the method.

- Step 2 :

Given the pseudo-diameter PQ together with $\mathcal{D}(P) = \{\mathcal{N}_1(P), \mathcal{N}_2(P), ..., \mathcal{N}_p(P)\}$ and $\mathcal{D}(Q) = \{\mathcal{N}_1(Q), \mathcal{N}_2(Q), ..., \mathcal{N}_p(Q)\}$ where p is the depth of both the descent of P and that of Q, we construct the mesh descent $\mathcal{D}$ and more precisely its level $\mathcal{N}_i$ which will enable us to find the desired renumbering of the nodes.

To this end, three stages are necessary. First, we construct the pairs (i, j) as follows :

```
for M a given point in T
find the level of D(P) and that of D(Q) where is M.
Let N_i(P) and N_k(Q) be these two levels,
form the pair (i,j) such that j = p + 1 - k.
```

Using this information, we can start the construction of the global levels :

```
If the pair (i,i) exists for point M
   put M in the level N_i
   suppress M from the list of nodes to be examined, i.e., from graph Gr.
```

If Gr is empty, we go to Step 3 of the global algorithm, otherwise, the remaining nodes must be classified in the different levels.

The current graph Gr can now consist of a set of one or more disjoint connected components. These are found and ordered following the decreasing number of nodes they have. Let C_k be the component of index k and let *ncomp* be the number of connected components, then we analyze the *ncomp* components :

```
FOR k = 1, ncomp
   FOR m = 1, p
      compute n_m the number of nodes in level N_m,
      compute h_m = n_m+ (the number of nodes) ∈ C_k ∩ N_m(P),
      compute l_m = n_m+ (the number of nodes) ∈ C_k ∩ N_m(Q),
   END FOR
   compute h_0 = max_{m=1,p} (h_m) where h_m - n_m > 0,
   compute l_0 = max_{m=1,p} (l_m) where l_m - n_m > 0,
      IF h_0 < l_0
         put all the nodes in C_k in level N_i
         where i is the first index in the pair (i, j) associated with
         the node under examination,
      OTHERWISE IF h_0 > l_0
         put all the nodes in C_k in level N_j
         where j is the second index in the pair (i, j) associated
         with the node,
      OTHERWISE IF h_0 = l_0
         put all the nodes in C_k in the level with the
         smallest dimension,
      END IF
END FOR
```

In this way, each vertex of Gr has been assigned a level in the descent. Moreover, a nice balancing is obtained in the different levels. Then the numbering, following Cuthill Mackee, can be processed. This is the purpose of the following step.

- Step 3 :

First if $deg(Q) < deg(P)$, we reverse P and Q and the levels obtained at the previous step to ensure that the numbering will start from the endpoint of the pseudo-diameter of lower degree. Reversing the levels leads to fixing $\mathcal{N}_i = \mathcal{N}_{p-i+1}$, for $i = 1, p$. Then we want to obtain $newnum(M)$, for $M \in \mathcal{M}$, the desired number of the vertex M. To this end :

```
Initialization, one starts from P, thus M = P
   n = 1
   newnum(M) = n
   N_0 = {P}
FOR k = 1
   If ∃ M ∈ N_k with adjacent vertices not yet renumbered
      construct the list C(M) = adj(M) ∩ N_k where adj(M) notes
      the vertices neighbor of M
   OTHERWISE IF ∃ S ∈ C(M) not yet renumbered (such points being
   considered following the increasing order of their degrees)
      n = n + 1
      newnum(M) = n.
   END IF
END FOR
```



```
FOR k = 2, p - 1
   WHILE ∃ M ∈ N_k with adjacent vertices not yet renumbered
      construct the list C(M) = adj(M) ∩ N_{k+1} where adj(M) notes
      the vertices neighbor of M
   S OTHERWISE IF ∃ S ∈ C(M) not yet renumbered (such points being
   considered following the increasing order of their degrees)
         n = n + 1
         newnum(M) = n.
END FOR
```

If some cases (in fact, if P and Q were reversed and if j was selected in the previous step, *i.e.*, $h_0 > l_0$ was found when component C_1 was analyzed or P and Q were not reversed but i was selected in the previous step) the previous renumbering is reversed :

```
FOR i = 1, np
   i ← np - i + 1
END FOR
```

This vertex renumbering method has proved very efficient, and is, moreover, fully automatic. It results in a significant minimization of both the *bandwidth* and the *profile* of the sparse matrix that will be associated with the mesh being processed.

Arbitrary meshes. Given an arbitrary mesh (in terms of the node definition), the previous material can be applied for node renumbering. Let us consider a simple example, a P^2 mesh where the nodes are the mesh vertices together with the edge midpoints. A possible node renumbering method could be as follows :

- apply the previous method to the P^1 mesh associated with the current mesh. This comes down to only considering the vertices of the P^2 mesh as nodes.
- use the resulting node numbering to complete the node numbering of the P^2 mesh. To this end, initialize $i = 1$ and $num = 1$, then
  - pick the node i in the P^1 mesh, find the *ball* of node i, assign numbers $num, num + 1, ...$ to the P^2 nodes of the elements in this ball that have not yet been renumbered, update num and repeat this process (with $i = i + 1$).

Remark 17.13 *This rather simple renumbering method is probably not optimal but appears to be reasonable on average. To obtain better results, one could perform the classical method by considering the full node graph of the mesh, which therefore increases the effort required.*

Remark 17.14 *The case of adapted meshes, in specific anisotropic meshes, is not really a suitable situation for the above numbering method.*

17.4.2 Element renumbering

A compact numbering of the elements may be a source of benefit at the solution step as this could minimize some cache default or some memory access default.

A simple idea to achieve this feature is to make the element numbering more compact around the mesh vertices. Thus, it could be of interest to renumber the mesh vertices first before processing the element renumbering step. A synthetic algorithm is then :

```
Initializations
   num = 0
   newnum(1 : ne) = 0
FOR i = 1, np
   FOR j = 1, ne
      IF P_i is a vertex in K_j and newnum(j) = 0
         num = num + 1
         newnum(j) = num
      END IF
   END FOR j
END FOR i
```

with the same notations as above (K_j denotes an element and P_i denotes a vertex). At a glance, the complexity of this algorithm is something like $\mathcal{O}(n^2)$ where n stands for ne (or np), thus, this algorithm is not really suitable in terms of computer implementation.

17.4.3 Application examples

To illustrate the efficiency of the above method (for vertex renumbering), we give some selected examples related to various meshes.

Table 17.1 presents various characteristics of the meshes before and after being renumbered. The first two meshes are in two dimensional space, the two others correspond to three dimensional cases. The values of interest are the *total profile*, the *mean profile*, the *bandwidth* and v the number of nodes dealt with per second as a function of the size of the expected matrix, assumed to be an n by n symmetric matrix (with $n = np$, the number of nodes of the mesh). The bandwidth, β, is defined by

$$\beta = \max_i |i - j_i| ,$$

where $i = 1, n$ and j_i is the index of the first row where an a priori non null coefficient may exist. Indeed the value $|i - j_i|$, denoted as δ_i, measures the distance between the first " non null" row and the diagonal of the resulting matrix (*i.e.*, in terms of vertex indices, the distance, given vertex index i, between the vertex, adjacent to vertex i, which is as far as possible from vertex i). The profile and the mean profile are then

$$pr = \sum_{i=1}^{n} \delta_i \quad \text{and} \quad pr_{mean} = \frac{pr}{n} \quad \text{with} \quad n = np$$

Mesh	1	2	3	4
np	36,624	59,313	43,434	92,051
ne	72,783	116,839	237,489	463,201
β (initial)	35,396	53,580	42,516	89,530
pr	364,616,352	827,498,368	483,132,896	2,194,965,760
pr_{mean}	9,955	13,951	11,123	23,845
β (final)	811	458	1,368	5,186
pr	11,115,036	9,166,162	39,944,464	172,534,912
pr_{mean}	303	154	919	1,874
$\frac{pr}{pr_{mean}}$	33	90	12	13
v	29,000	27,000	11,000	10,600

Table 17.1: Gibbs : bandwidth and profile before and after renumbering. Examples one and two concern two-dimensional triangle meshes while the two other examples are three-dimensional tetrahedral meshes.

The Gibbs method proves to be robust and efficient. In addition, it is fully automatic and does not require any directive. Comments on these statistics can be seen below (to allow the comparison with other renumbering methods).

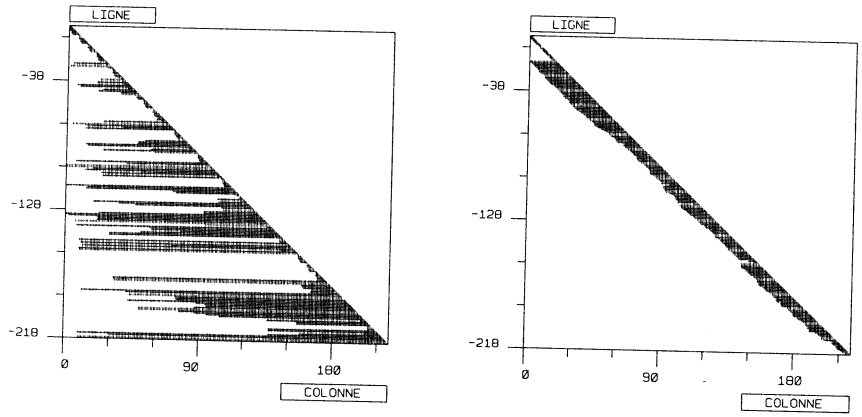

Figure 17.12: *Matrix occupation associated with a given mesh before (left-hand side) and after node renumbering (right-hand side). For the sake of clarity, this example concerns a "small" mesh different from those described in Table 17.1.*

Remark 17.15 *Any conclusion on the efficiency of a Gibbs based renumbering method must be made more carefully in the case of adapted meshes where the density of elements varies greatly from place to place or also when anisotropic elements made up the mesh. In such cases, other renumbering methods may be advocated (see below).*

17.4.4 Other renumbering methods

Among the various renumbering methods, the so-called *frontal method*[7] is now briefly discussed (to make some comparisons with the Gibbs method possible and to take into account the above remark).

In essence, this method consists of radiating from a starting front. Let $\mathcal{F}_0$ be a set reduced to one node or a collection of adjacent nodes (generally chosen on the domain boundary), then the algorithm can formally be written as follows :

```
k ← 0
(A) FOR i = 1, card(F_k) (the number of nodes in F_k)
   FOR all the elements sharing node i in F_k
      construct V_i, the set of indices of the nodes neighboring
      node i, not yet renumbered
      FOR all the nodes in V_i (let j be the index of such a node)
         compute nbneigh(j), the number of neighbors of node j
      END FOR
      renumber the nodes in V_i as a function of nbneigh(j),
      the number of neighbors (in increasing or decreasing order
      of nbneigh)
   END FOR
```

Thus, set $\mathcal{V}_i$ forms the new renumbering front, then :

```
k ← k + 1
F_k = V_i
IF card(F_k) > 0
   return to (A)
OTHERWISE, END.
   The process is completed (the numbering may be reversed).
```

Numerous variations exist. For instance, this method of node renumbering depends greatly on the choice of the initial front and could therefore require the user's directive. It also depends on the way in which the nodes of the $\mathcal{V}_i$'s are selected.

Finding the initial front can be the user's responsibility or can be obtained using the above technique for the construction of the *pseudo-diameter* of the mesh.

Choosing how to proceed the $\mathcal{V}_i$'s can be done :

- by selecting the node in $\mathcal{V}_i$ with minimum degree,
- by selecting the node in $\mathcal{V}_i$ with maximum degree,
- by selecting the node in $\mathcal{V}_i$ with minimum index,
- and so on.

[7] Also referred to as *greedy*.

Tables 17.2 and 17.3, similar to the previous table, give the statistics related to two different variations of a frontal method. Table 17.2 considers a method whose initial front is the point with minimal degree (presumably a boundary point) while Table 17.3 shows a method where the pseudo-diameter is used to initialize the front.

Mesh	1	2	3	4
np	36,624	59,313	43,434	92,051
ne	72,783	116,839	237,489	463,201
β (initial)	35,396	53,580	42,516	89,530
pr	364,616,352	827,498,368	483,132,896	2,194,965,760
pr_{mean}	9,955	13,951	11,123	23,845
β (final)	552	303	1,390	4,527
pr	16,145,040	12,647,528	39,873,528	254,601,584
pr_{mean}	440	213	918	2,765
$\frac{pr}{pr_{mean}}$	23	65	12	9
v	55,000	50,000	15,000	15,000

Table 17.2: Naive frontal method : bandwidth and profile before and after renumbering (the examples are the same as in Table 17.1).

Mesh	1	2	3	4
np	36,624	59,313	43,434	92,051
ne	72,783	116,839	237,489	463,201
β (initial)	35,396	53,580	42,516	89,530
pr	364,616,352	827,498,368	483,132,896	2,194,965,760
pr_{mean}	9,955	13,951	11,123	23,845
β (final)	630	357	1,425	5,172
pr	12,296,281	10,045,739	41,060,840	235,254,128
pr_{mean}	335	169	945	2,555
$\frac{pr}{pr_{mean}}$	30	82	12	9
v	39,000	37,000	13,000	12,300

Table 17.3: Frontal method starting from the pseudo-diameter : bandwidth and profile before and after renumbering.

Comparing the three tables leads to some comments. In terms of bandwidth, in our (limited) experience the two frontal methods give as the worst value one smaller (*i.e.*, better) than the Gibbs method. Nevertheless, the Gibbs method results in a better profile (and, thus the ratio $\frac{pr}{pr_{mean}}$ is larger for any of the examples we have tried). In terms of CPU, the naive method is less demanding than the other frontal method and the Gibbs method requires more effort. As a final remark, it could be observed that the ratio of improvement due to a renumbering method depends on the geometry of the domain. In this respect, thin regions connected

to large regions, loops (holes), etc., interact on the result.

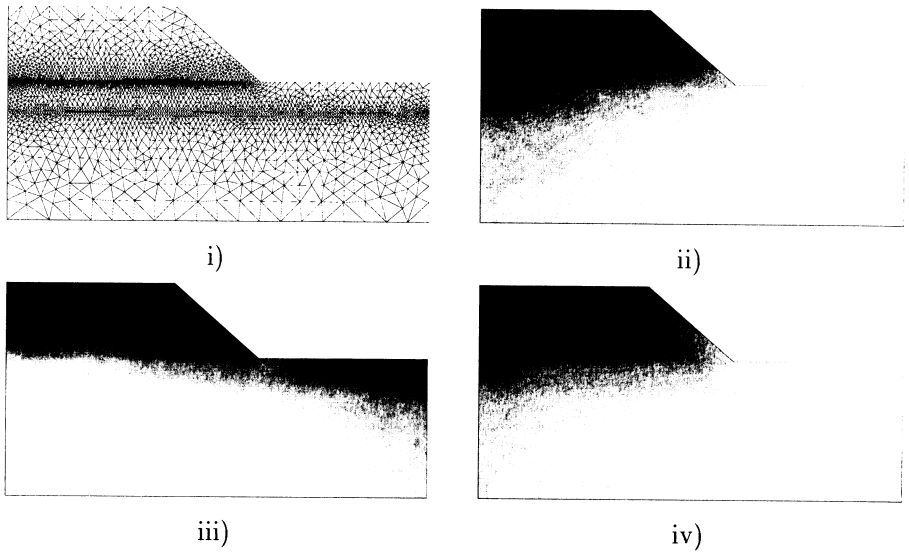

Figure 17.13: *Gibbs versus frontal methods. The original mesh i), Gibbs ii) naive frontal iii) and frontal with pseudo-diameter iv). Colors are associated to vertex indices (from white (vertex #1) to black (vertex #np). The color variation indicates the way the methods proceed.*

Element coloring. A method can be constructed to color the elements of a given mesh. The goal is to obtain several disjoint packets of elements, each of which corresponds to one color while two elements in two different neighboring packets have two different colors. Using ideas similar to the four colors theorem, this method consists of renumbering the elements of a mesh by creating packets of elements such that any neighbor of an element in a given packet is not located in the same packet. When only two packets are created, this method clearly looks like the "Red-Black" method [Melhem-1987] which permits separation of nodes into two disjoint sets. Put briefly, the algorithm is based on searching for the neighbors of an element with the aim of separating these elements. The main application of this method is the numerical solution to a problem on a vector computer, but the idea may be applied to various areas.

Other numbering (coloring) issues. Various methods can be developed based on particular numbering or coloring techniques.

17.5 Miscellaneous

In this section, we briefly mention some topics about mesh modification that have not already been covered.

Mesh manipulation and physical attributes. As mentioned earlier, a mesh devoted to a given numerical simulation does not include only a geometric aspect and must contain information about the physics of the problem. In this respect, the elements in the mesh are not fully described when the vertex coordinates and the vertex connection are known. Actually, physical attributes are also part of the mesh description. At the element level, it has proved to be useful to have some physical attributes associated with the element itself as well as with the element entities (vertex, edge, face). Therefore, any mesh modification includes two parts. The part concerned with the geometric aspect of the mesh manipulation has been already described while the way to manage the physical attributes of the given mesh and to derive the corresponding information for the resulting mesh must be now discussed.

In practice, the physical aspects of the resulting mesh inherit, in some way, the physical aspects included in the initial mesh. The issue is to make the desired correspondences precise, based on the type of mesh manipulation used. Figure 17.14 shows two basic examples and gives some idea of what must be done.

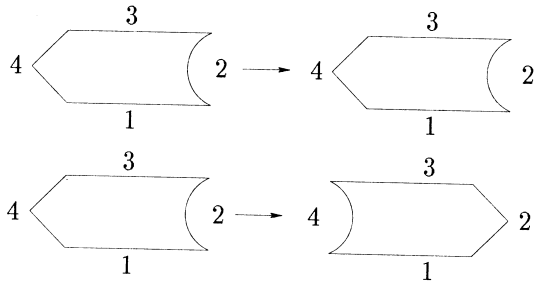

Figure 17.14: *According to the transformation (translation (top), symmetry (bottom)), attributes* $1, 2, 3$ *and* 4 *remain unchanged or must be permuted as* $1, 4, 3$ *and* 2.

Non obtuse mesh. See Chapter 18.

Crack or widthless region construction. Defining one or several cracks or regions with little thickness makes it possible to deal with some types of problems in solid or fluid mechanics.

A crack (a thin region) is a line (a surface) which has *a priori* two meshes. In the case of a crack, these two meshes are coincident, in some sense. After processing, the crack may open and become longer.

Constructing a thin zone makes it possible, for example, to introduce finite elements like line elements between the two initial elements sharing the common edge (in two dimensions).

Creating a crack of a zone with no thickness is not an easy task for a mesh generation method. The fact that there is at least one pair of strictly coincident points at the beginning implies that the classical methods (*quadtree-octree*, advancing-front

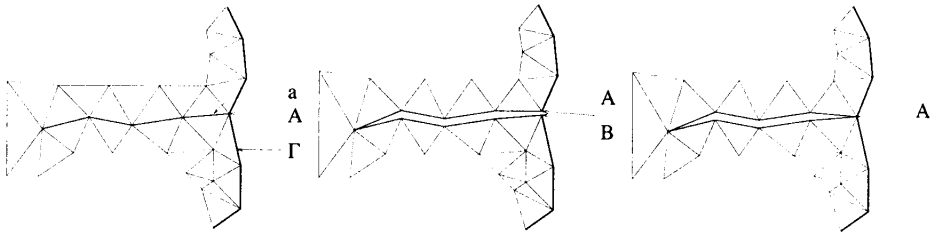

Figure 17.15: *Left-hand side : a part of the domain, a set of edges that must be duplicated (a), the boundary Γ and the initial endpoint of the crack (A). Middle : the crack appears and the initial point A is duplicated (in point A and another point B which is distinct). Right-hand side : initial point A is not duplicated.*

or Delaunay based) are not suitable, at least in their classical versions (Chapters 5, 6, 7). Such a situation is indeed seen as a cause of failure. Therefore, defining such a zone is not possible without using specific post-processing[8].

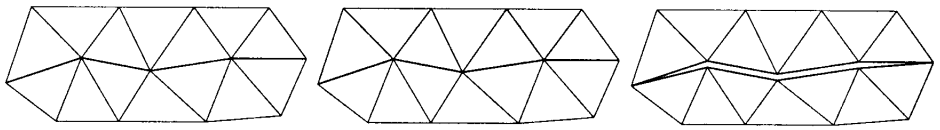

Figure 17.16: *Elements located on one of the side of the crack are identified, then the crack is defined by duplicating the edges that have been detected in this way.*

The first method consists in a suitable construction of two meshes which are then merged while avoiding the points and the edges previously identified (see above) to be merged. Thus, a merging operator is used to complete the desired result. A second method (Figure 17.16) consists of finding all the elements that share the edges to be duplicated, identifying the adjacencies and then suppressing these adjacencies by duplicating the entities concerned.

Periodic mesh. Some problems show periodic properties at the boundary condition level in some part of the domain boundary which, in turn, has the same periodic aspect (think about a translation or a rotation).

Some computational software then make it possible to reduce the analysis to only one portion of the domain provided the appropriate periodic conditions are defined in such a way as to relate some parts one with another. It is then possible to deduce the entire domain together with the solution in this entire domain whereas the computation has only been done in one portion of this domain (for example, an analysis of a single turbine blade allows us to obtain the result for the whole turbine).

In terms of how to construct a periodic mesh, it is sufficient to be sure that the two zones that must be in correspondence have been meshed in the same

[8] Or, at least, some degree of tinkering inside the mesh generation method !

way. Thus consistency will be ensured when the pair of nodes to be connected are identified.

Remark 17.16 *If the region to be processed is composed of quads (in three dimensions) and if the elements (hexs or pentahedra) are split into tets, it is necessary to constrain the element split in order to obtain a compatible mesh on both sides (if mesh conformity is required).*

In conclusion, mesh modification methods (with a view of mesh optimization) have not been discussed in this chapter. This point is discussed in Chapter 18 for planar and volumic meshes and Chapter 19 deals with surface meshes.