The Generation of Valid Curvilinear Meshes

Christophe Geuzaine[1], Amaury Johnen[1],
Jonathan Lambrechts[2,3], Jean-François Remacle[2], Thomas Toulorge[2,3]

[1] Université de Liège, Dept. of Electrical Engineering and Computer Science, B-4000 Liège, Belgium
{a.johnen,cgeuzaine}@ulg.ac.be,
[2] Université catholique de Louvain, Institute of Mechanics, Materials and Civil Engineering (iMMC), B-1348 Louvain-la-Neuve, Belgium
{thomas.toulorge,jonathan.lambrechts,jean-francois.remacle}@uclouvain.be,
[3] Fonds National de la Recherche Scientifique, B-1000 Brussels, Belgium

Abstract. It is now well-known that a curvilinear discretization of the geometry is most often required to benefit from the computational efficiency of high-order numerical schemes in simulations. In this article, we explain how appropriate curvilinear meshes can be generated. We pay particular attention to the problem of invalid (tangled) mesh parts created by curving the domain boundaries. An efficient technique that computes provable bounds on the element Jacobian determinant is used to characterize the mesh validity, and we describe fast and robust techniques to regularize the mesh. The methods presented in this article are thoroughly discussed in Ref. [1,2], and implemented in the free mesh generation software Gmsh [3,4].

Keywords: High-order mesh, curvilinear mesh, geometry discretization, mesh validity, element Jacobian

1 Introduction

There is a growing consensus in the computational mechanics community that state of the art solver technology requires, and will continue to require too extensive computational resources to provide the necessary resolution for a broad range of demanding applications, even at the rate that computational power increases. The requirement for high resolution naturally leads us to consider methods which have a higher order of grid convergence than the classical (formal) 2^{nd} order provided by most industrial grade codes. This indicates that higher-order discretization methods will replace at some point the current finite volume and finite element solvers, at least for part of their applications.

The development of high-order numerical technologies for engineering analysis has been underway for many years now. For example, Discontinuous Galerkin methods (DGM) have been thoroughly studied in the literature, initially in a theoretical context [5], and now from the application point of view [6]. In many contributions, it is shown that the accuracy of the method strongly depends on the accuracy of the geometrical discretization (see for instance Ref. [7]). It is thus

necessary to address the problem of generating high-order, curvilinear meshes. This article focuses on unstructured meshes, that make it possible to deal with the complex geometries involved industrial-grade problems. It is mainly dedicated to the theoretical and practical aspects of high-order mesh generation as they are implemented in the Gmsh mesh generator [3,4], that is distributed as free software. An in-depth discussion of the techniques described hereafter, as well as numerous examples and applications, can be found in Ref. [1,2].

The rest of the article is organized as follows. Section 2 gives an overview of high-order mesh generation methods available in the literature. In Section 3, we describe how high-order nodes are created, ordered in elements and placed on the curved boundaries. In Section 4, we address the topic of mesh validity and the reliable evaluation of element Jacobian determinants. Section 5 details the mesh regularization methods available in Gmsh. Finally, we shortly draw conclusions in Section 6.

2 Overview of high-order mesh generation methods

Although high-order numerical schemes have been a subject of intense research for many years, a comparatively small number of research groups have worked on high-order mesh generation methods, despite the necessity pointed out above. So far, efforts in this field have been focused on the problem of creating curvilinear meshes (i.e. meshes that follow the curvature of the domain boundaries defined by CAD models) that fulfill the validity requirements imposed by the dominant numerical methods (namely finite volumes and finite elements).

Two approaches can be considered for the generation of high-order meshes [8]: one can either try to create directly a valid high-order mesh through a high-order version of a classical meshing algorithm, or generate a first-order (i.e. straight-sided) mesh that is subsequently curved and regularized if invalidities are found. Both approaches are actually related, as they share the necessity of detecting and repairing the invalid parts of the mesh.

The procedures for detection and regularization of invalidities are much more complex and computationally expensive in the high-order case than for usual straight-sided meshes, but the curvilinear character of a high-order mesh is usually localized to the vicinity of curved boundaries. This is why the direct approach, that involves operations on high-order meshes in the whole domain, has been disregarded by most if not all authors in favor of the indirect approach that takes advantage of existing technologies.

2.1 Curving

After generation of the first-order mesh, the following step in the indirect approach is to curve the boundaries of the mesh according to the geometry of the problem. Model entities like model faces or model edges are supposed to be smooth manifolds that are defined through a parametrization. In engineering applications, the geometry of model entities is most often defined by a CAD

model, but discrete geometries that are common in biomedical applications can also be smoothly parametrized.

The mesh elements (or cells in the finite volume terminology) are usually defined in a Lagrangian manner by a set of nodes (or vertices). Thus, the curving procedure normally comes down to placing the high-order nodes corresponding to boundary edges and boundary faces on the exact geometry.

This part of the high-order mesh generation process is generally not considered as a crucial point, and little information can be found about it in the literature. Simple techniques for obtaining high-order boundary nodes include interpolating them between the first-order boundary nodes in the parametrization describing the corresponding CAD entity [8,9] and projecting them on the geometry from their location on the straight-sided element.

More sophisticated procedures have also been proposed. In Ref. [10], the high-order nodes on boundary edges are interpolated in the physical space through a numerical procedure involving either the CAD parametrization (in the case of a mesh edge corresponding to an edge of the geometric model), or an approximation of the geodesic connecting the two first-order vertices (in the case of an edge located on a 3D surface). Nodes located within surface elements are obtained through a more sophisticated version of this procedure. Instead of interpolation, Sherwin and Peiró [11] use a mechanical analogy with chains of springs in equilibrium that yields the optimal node distribution along geometric curves and geodesics for edge nodes. Two-dimensional nets of springs provide the optimal distribution of surface element nodes.

These procedures generally aim to obtain a given node distribution on the boundary of the computational domain. However, it may be more interesting to search for the node locations that minimize the error in the approximation of the exact geometric by the high-order mesh boundary, for instance in terms of the Hausdorff distance. To our knowledge, this topic point has not yet been explored in the context of high-order mesh generation for scientific computation.

The creation and placement of high-order nodes is treated in detail in Section 3.

2.2 Detection of invalidities

As illustrated in Section 3, curving the mesh boundaries while leaving other mesh entities unchanged may result in an invalid mesh with tangled elements. From the point of view of numerical schemes, a mesh is generally considered as valid if it is possible to integrate a quantity over each element, and if elements do not overlap. Both requirements can be evaluated through the Jacobian determinant of a transformation between a canonical reference element and the actual element of the mesh, as in the finite element framework (see Section 4.1). The validity depends on the sign of the Jacobian.

In order to characterize the quality of a high-order mesh independently of the size of its elements, authors have used different *distortion* measures based on the Jacobian determinant. Most of them involve the ratio of minimum to maximum Jacobian in each element [8,10,11,12,13,14], the ratio of the minimum Jacobian

to the Jacobian of the corresponding straight-sided element [1] or the integral of the Jacobian divided by the Jacobian of the straight-sided element [15].

It is important to note that the Jacobian determinant of an element is a higher-order polynomial of the coordinates in the reference space. As such, its oscillatory nature makes it difficult to evaluate its extrema (and thus its sign) by sampling. Several authors have developed procedures based on the convex hull properties of Béziers polynomials for the reliable evaluation of Jacobian bounds [1,13,16]. This point is the object of Section 4.

2.3 Regularization of invalid meshes

In case invalid elements are detected in a high-order mesh, a procedure to regularize the mesh (i.e. untangle the broken elements) has to be applied. This is arguably the most important and problematic part of high-order mesh generation. On one hand, the procedure has to be robust, because a single invalid element can make the mesh inappropriate for computation. On the other hand, the regularization procedure shall not be much more computationally intensive than the rest of the meshing process, in order to remain suitable for practical applications. Several types of methodologies have been proposed in the literature.

A first possibility is to apply local refinement to the mesh in the regions where the boundary curvature is considered as large enough to possibly create invalid elements [11]. It has also been proposed to move the high-order nodes along specific lines depending on geometric constraints and neighbouring elements [13]. More sophisticated methodologies combine a prescribed deformation of elements near curved boundaries (by moving the control points of their Bézier representation) with topological modifications in order to untangle all invalid elements [8,16,17].

A second class of regularization methods are based on mechanical analogies: the mesh is considered as embedded in an elastic medium, and the imposition on the boundaries of a displacement corresponding to the curving results in a problem that can be solved using finite elements. No topological modification is made. The simple linear elastic analogy has been employed [10], but it may prove unreliable. A more robust method may be obtained by using non-uniform material properties, that is by using stiffer material in small elements near curved boundaries. More successful strategies have been proposed, including a non-linear mechanics formulation [14] and an iterative method working on the Bézier control points of the elements instead of the Lagrangian nodes [12].

The last approach for regularization methods is based on optimization algorithms. The point is then to find the mesh node locations which minimize an objective function that characterizes the validity of the mesh, using the measures described in Section 2.2. In Ref. [9], such a method is applied with a extended distortion measure on triangular surface meshes. In a recent paper [2], we have taken advantage of the control allowed by this approach on the validity of the mesh to propose a robust untangling method, that is detailed in Section 5.2.

3 Creation of high-order meshes

3.1 High-order elements

When dealing with high order elements, it is important to define how element nodes are numbered for a given element shape and for a given order p. Our convention is very general with respect to both those parameters. For all mesh and post-processing purposes, reference elements are defined as in a classical finite element framework.

We start by considering linear elements (lines, triangles, quads, tets, prisms, hexes and pyramids), that contain only corner vertices. In 2D elements, the nodes are numbered in such a way that each edge connects two consecutive nodes, the normal to the oriented edge pointing outwards. In 3D elements, the edge-vertex connectivity does not follow a general rule. Instead, the node ordering is such that of the normal to each face of the element points outward.

The node ordering of a higher order (possibly curved) element is a generalization of the one used in the first-order element. We number nodes in the following order:

- the element corner vertices (i.e. the first-order vertices);
- the internal nodes for each edge;
- the internal nodes for each face;
- the volume internal nodes.

The numbering for internal nodes is recursive, ie. the numbering follows that of the nodes of an embedded edge/face/volume of lower order. The higher order nodes are assumed to be equispaced.

A visual description of the node ordering used in the Gmsh mesh generation software can be found in its documentation [4].

3.2 Generating high order meshes based on CAD data

Modern mesh generation procedures take as input CAD models composed of *model entities*: vertices G_i^0, edges G_i^1, faces G_i^2 or regions G_i^3. Each model entity G_i^d has a geometry (or shape) [18,3] for which solid modelers usually provide a parametrization, that is, a mapping $\boldsymbol{\xi} \in R^d \mapsto \boldsymbol{x} \in R^3$. There are also four kind of mesh entities M_i^d that are said to be classified on model entities. Each mesh entity is classified on the model entity of the smallest dimension that contains it. The way of building a high order mesh is to first generate a straight sided mesh. Then, mesh entities (edges, faces and regions) are curved according to the geometry of the CAD entity it is classified on.

In the p-version of finite elements, high order nodes are added to edges, faces and regions of the element with the aim of creating curvilinear elements with their shape based on high order (Lagrangian or not) polynomial bases (see Figure 1). Other authors [19,20] would rather use the exact mappings of the geometry and build a so-called isogeometric mesh.

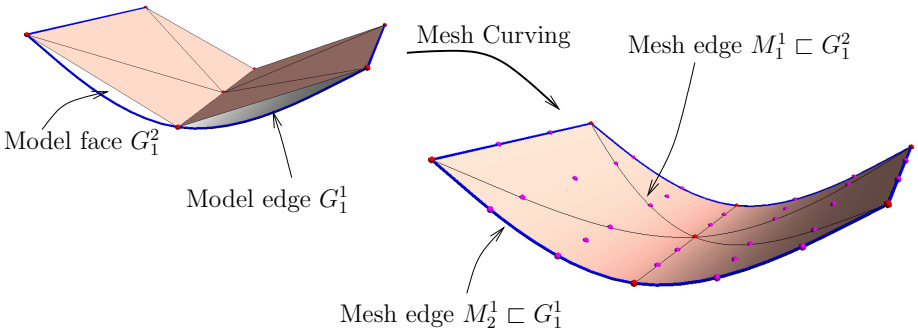

Fig. 1. Straight sided mesh (left) and curvilinear (cubic) mesh (right).

The naive curving procedures described above does not ensure that all the elements of the final curved mesh are valid. Figure 2 gives an illustration of this important issue: some of the curved triangles are tangled after having been curved. It is important to note that this problem is not related to the accuracy of the geometrical discretization: in Figure 2, the mesh would not be valid in the iso-geometric case i.e. if the curved edge was assigned the exact geometry (blue curve).

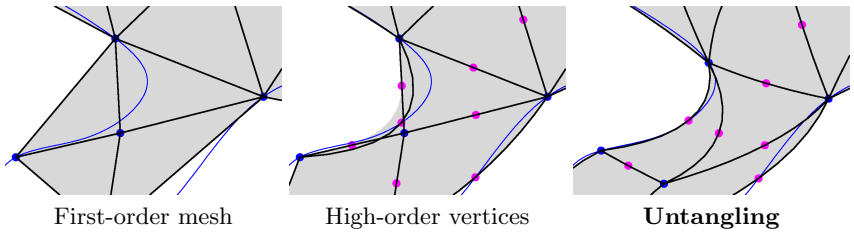

Fig. 2. Straight sided mesh (left) basic curvilinear (quadratic) mesh with tangled elements (center) and untangled mesh (right).

Gmsh implements the polynomial version of high order meshes. There, the choice of the position of high order nodes is a crucial parameter for the validity of the high order elements.

The position of high order nodes can be chosen by linear interpolation between the first-order nodes in the parametric space provided by the solid modeler for the corresponding model entity. This procedure has the advantage of being simple, fast and geometrically robust (no projection on surfaces is needed). Yet, ill conditioned mappings are very common in CAD systems. Such a procedure will generate very bad elements in the vicinity of singularities (e.g. the two poles of a sphere described using spherical coordinates).

Another option would be to place new nodes in such a way that the geometrical error, i.e. the distance between the CAD model and the mesh, is minimized. Note that the definition of such a distance is not trivial and that the non linear procedure involved in such a minimization is time consuming.

The way Gmsh generates high order nodes (order p) on a mesh made of simplices (triangles and tets) is the following one:

- For every model edge G_i^1, add $p-1$ high order nodes on mesh edges M_j^1 that are classified on G_i^1,
- For every model face G_i^2, add $p-1$ high order nodes on mesh edges M_j^1 that are classified on G_i^2,
- For every model region G_i^3, add $p-1$ high order nodes on mesh edges M_j^1 that are classified on G_i^3,
- For every model face G_i^2, add $(p-1)\times(p-2)/2$ high order nodes on mesh faces M_j^2 that are classified on G_i^2,
- For every model region G_i^3, add $(p-1)\times(p-2)/2$ high order nodes on mesh faces M_j^2 that are classified on G_i^3,
- For every model region G_i^3, add $(p-1)\times(p-2)\times(p-3)/6$ high order nodes on mesh regions M_j^3 that are classified on G_i^3.

When an edge going from $\boldsymbol{x}_a$ to $\boldsymbol{x}_b$ is saturated with its $p-1$ high order points $\boldsymbol{x}_i$, those are initially placed on a straight line segment between $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ in an equispaced manner. Then, points $\boldsymbol{x}_i$ are orthogonally projected on the real geometry, using the functions provided by CAD systems. When an internal edge of a region is saturated with high order points, no projection is needed.

When high order points are added on mesh faces classified on model faces, a different procedure is applied. An incomplete element that contains corner nodes and edge nodes is constructed. Those elements are sometimes called serendipity (or incomplete) elements [21]. Gmsh's way of numbering nodes allows us to easily generate such elements. Incomplete elements are not converging optimally to the exact geometry when the mesh is refined. Internal nodes have to be added to achieve optimal convergence. Shape functions of the incomplete elements are used to place the internal nodes on the geometry of the curvilinear incomplete element. Then, those nodes are projected on the model face. The same procedure is used for inserting mesh vertices inside high order regions. Yet, no projection is needed in this case. The use of incomplete elements to predict the position of high order nodes before projection has experimentally given better results than a simple projection. Yet, there is no guarantee that such a procedure produces valid high order meshes.

4 Validity of high-order meshes

4.1 Curvilinear Meshes, Distortion and Bounds on Jacobian Determinants

Let us consider a mesh that consists of a set of *straight-sided* elements of order p. Each element is defined geometrically through its nodes $\mathbf{x}_i$, $i = 1, \ldots, N_p$ and

a set of Lagrange shape functions $\mathcal{L}_i^{(p)}(\boldsymbol{\xi}),\ i = 1, \ldots, N_p$. The Lagrange shape functions (of order p) are based on the nodes $\mathbf{x}_i$ and allow to map a reference unit element onto the real one:

$$\mathbf{x}(\boldsymbol{\xi}) = \sum_{i=1}^{N_p} \mathcal{L}_i^{(p)}(\boldsymbol{\xi})\ \mathbf{x}_i. \tag{1}$$

The mapping $\mathbf{x}(\boldsymbol{\xi})$ should be injective, which means that it should admit an inverse. This implies that the Jacobian determinant $\det \mathbf{x}_{,\boldsymbol{\xi}}$ has to be strictly positive. In all what follows we will always assume that the straight-sided mesh is composed of well-shaped elements, so that the positivity of $\det \mathbf{x}_{,\boldsymbol{\xi}}$ is guaranteed. This standard setting is presented on Figure 3 (left) for the quadratic triangle.

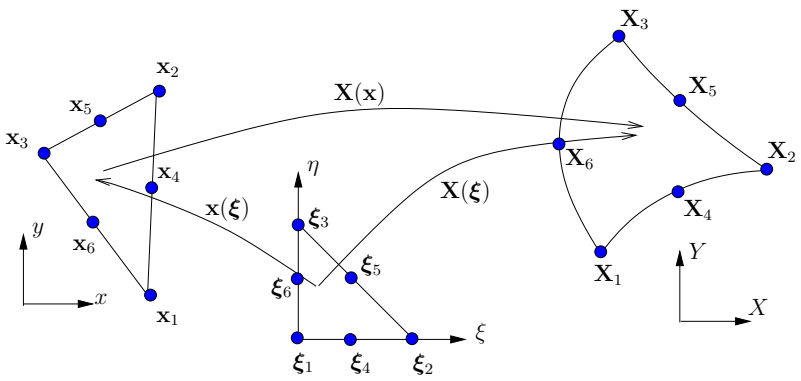

Fig. 3. Reference unit triangle in local coordinates $\boldsymbol{\xi} = (\xi, \eta)$ and the mappings $\mathbf{x}(\boldsymbol{\xi})$, $\mathbf{X}(\boldsymbol{\xi})$ and $\mathbf{X}(\mathbf{x})$.

Let us now consider a *curved* element obtained after application of the curvilinear meshing procedure, i.e., after moving some or all of the nodes of the straight-sided element. The nodes of the deformed element are called $\mathbf{X}_i$, $i = 1 \ldots N_p$, and we have

$$\mathbf{X}(\boldsymbol{\xi}) = \sum_{i=1}^{N_p} \mathcal{L}_i^{(p)}(\boldsymbol{\xi})\ \mathbf{X}_i. \tag{2}$$

Again, the deformed element is assumed to be valid if and only if the Jacobian determinant $J(\boldsymbol{\xi}) := \det \mathbf{X}_{,\boldsymbol{\xi}}$ is strictly positive everywhere over the $\boldsymbol{\xi}$ reference domain. The Jacobian determinant J, however, is not constant over the reference domain, and computing $J_{\min} := \min_{\boldsymbol{\xi}} J(\boldsymbol{\xi})$ is necessary to ensure positivity.

The approach that is commonly used is to sample the Jacobian determinant on a very large number of points. Such a technique is however both expensive and not fully robust since we only get a necessary condition.

It is possible to follow a different approach: because the Jacobian determinant J is a polynomial in $\boldsymbol{\xi}$, J can be interpolated exactly as a linear combination of specific polynomial basis functions over the element. We would then like to obtain provable bounds on $J_{\min}$ by using the properties of these basis functions.

In addition to guaranteeing the geometrical validity of the curvilinear element, we are also interested in quantifying its *distortion*, i.e., the deformation induced by the curving. To this end, let us consider the transformation $\mathbf{X}(\mathbf{x})$ that maps straight sided elements onto curvilinear elements (see Figure 1). It is possible to write the determinant of this mapping in terms of the $\boldsymbol{\xi}$ coordinates as:

$$\det \mathbf{X}_{,\mathbf{x}} = \frac{\det \mathbf{X}_{,\boldsymbol{\xi}}}{\det \mathbf{x}_{,\boldsymbol{\xi}}} = \frac{J(\boldsymbol{\xi})}{\det \mathbf{x}_{,\boldsymbol{\xi}}}. \tag{3}$$

We call $\mathbf{X}(\mathbf{x})$ the distortion mapping and its determinant $\delta(\boldsymbol{\xi}) := \det \mathbf{X}_{,\mathbf{x}}$ the distortion. The distortion δ should be as close to $\delta = 1$ as possible in order not to degrade the quality of the straight sided element. Elements that have negative distortions are of course invalid but elements that have distortions $\delta \ll 1$ or $\delta \gg 1$ lead to some alteration of the conditioning of the finite element problem. In order to guarantee a reasonable distortion it is thus necessary to find a reliable bound on $J_{\min}$ and $J_{\max} := \max_{\boldsymbol{\xi}} J(\boldsymbol{\xi})$ over the whole element.

Note that many different quality measures can be defined based on the Jacobian determinant J. For example, one could look at the Jacobian determinant divided by its average over the element instead of looking at the distortion. Obtaining bounds on $J_{\min}$ and $J_{\max}$ is thus still the main underlying challenge. Other interesting choices are presented and analyzed in [15].

4.2 Adaptive Bounds for Arbitrary Curvilinear Finite Elements

In order to explain the adaptive bound computation let us first focus on the one-dimensional case, for "line" finite elements. Since Bézier functions can be generated for all types of common elements (triangles, quadrangles, tetrahedra, hexahedra and prisms), the generalization to 2D and 3D elements will be straightforward.

The One-Dimensional Case In 1D the Bézier functions are the Bernstein polynomials:

$$\mathcal{B}_k^{(n)}(\xi) = \binom{n}{k} (1-\xi)^{n-k}\, \xi^k \qquad (\xi \in [0,1]\ ;\ k = 0, ..., n) \tag{4}$$

where $\binom{n}{k} = \frac{n!}{n!(n-k)!}$ is the binomial coefficient. The Bézier interpolation requires $n+1$ control values b_i. We have

$$J(\xi) = \sum_{k=0}^{N_n} \mathcal{B}_k^{(n)}(\xi)\, b_k. \tag{5}$$

Bernstein-Bézier functions have the nice following properties : (i) they form a partition of unity which means that $\sum_{k=0}^{n} \mathcal{B}_k^{(n)}(\xi) = 1$ for all $\xi \in [0,1]$ and (ii) they are positive which means that $\mathcal{B}_k^{(n)}(\xi) \geq 0$ for all $\xi \in [0,1]$. This leads to the well known property of Bézier interpolations:

$$\min_{\xi\in[0,1]} J(\xi) \geq b_{\min} = \min_i b_i \quad \text{and} \quad \max_{\xi\in[0,1]} J(\xi) \leq b_{\max} = \max_i b_i. \tag{6}$$

Moreover, the control values related to the corner nodes of the element are equal to the values of the interpolated function. In what follows we assume that these "corner" control values are always ordered at the K_f first indices. We then have

$$\min_{\xi\in[0,1]} J(\xi) \leq \min_{i<K_f} b_i \quad \text{and} \quad \max_{\xi\in[0,1]} J(\xi) \geq \max_{i<K_f} b_i. \tag{7}$$

Since Lagrange and Bézier functions span the same function space, computation of the Bézier values b_i from the nodal values J_i (and convertly) is done by a transformation matrix. The tranformation matrix $\boldsymbol{T}^{(n)}_{\mathcal{B}\to\mathcal{L}}$, which computes nodal values from control values, is created by evaluating Bézier functions at sampling points:

$$\boldsymbol{T}^{(n)}_{\mathcal{B}\to\mathcal{L}} = \begin{bmatrix} \mathcal{B}_0^{(n)}(\xi_0) & \dots & \mathcal{B}_n^{(n)}(\xi_0) \\ \mathcal{B}_0^{(n)}(\xi_1) & \dots & \mathcal{B}_n^{(n)}(\xi_1) \\ \vdots & \ddots & \vdots \\ \mathcal{B}_0^{(n)}(\xi_n) & \dots & \mathcal{B}_n^{(n)}(\xi_n) \end{bmatrix}.$$

Those sampling points are taken uniformly, *i.e.* at the location of the nodes of the element of order n. The inverse transformation is $\boldsymbol{T}^{(n)}_{\mathcal{L}\to\mathcal{B}} = {\boldsymbol{T}^{(n)}_{\mathcal{B}\to\mathcal{L}}}^{-1}$ and from the expression of the interpolation of the Jacobian determinant (5), we can write

$$\begin{aligned} J &= \boldsymbol{T}^{(n)}_{\mathcal{B}\to\mathcal{L}}\, B \\ B &= \boldsymbol{T}^{(n)}_{\mathcal{L}\to\mathcal{B}}\, J\,, \end{aligned} \tag{8}$$

where B and J are the vectors containing respectively the b_i's and the J_i's.

Adaptive Subdivision Let us assume that the domain $[0,1]$ is subdivided into Q parts. The interpolation $J^{[q]}(\xi^{[q]})$ on the q^{th} subdomain $[a,b]$ $(0 \leq a < b \leq 1)$ must verify

$$J^{[q]}(\xi^{[q]}) = \sum_{k=0}^{N_n} \mathcal{B}_k^{(n)}(\xi^{[q]})\; b_k^{[q]} = \sum_{k=0}^{N_n} \mathcal{B}_k^{(n)}(\xi(\xi^{[q]}))\; b_k \qquad (\xi^{[q]} \in [0,1]), \tag{9}$$

with $\xi(\xi^{[q]}) = a + (b-a)\;\xi^{[q]}$.

Considering the nodes $\xi_k^{[q]}$ such that $\xi_k^{[q]} = \xi_k$ $(k = 0, \ldots, n)$ (i.e., such that they are ordered like the sampling points), the expression (9) reads

$$\boldsymbol{T}^{(n)}_{\mathcal{B}\to\mathcal{L}}\, B^{[q]} \;=\; \begin{bmatrix} \mathcal{B}_0^{(n)}(a+(b-a)\,\xi_0) & \ldots & \mathcal{B}_n^{(n)}(a+(b-a)\,\xi_0) \\ \mathcal{B}_0^{(n)}(a+(b-a)\,\xi_1) & \ldots & \mathcal{B}_n^{(n)}(a+(b-a)\,\xi_1) \\ \vdots & \ddots & \vdots \\ \mathcal{B}_0^{(n)}(a+(b-a)\,\xi_n) & \ldots & \mathcal{B}_n^{(n)}(a+(b-a)\,\xi_n) \end{bmatrix} B \;=\; \boldsymbol{T}^{(n)}_{\mathcal{B}\to\mathcal{L}}{}^{[q]}\, B,$$

where $B^{[q]}$ is the vector containing the control values of the q^{th} subdomain. This implies that

$$B^{[q]} = \left[\boldsymbol{T}^{(n)}_{\mathcal{L}\to\mathcal{B}}\, \boldsymbol{T}^{(n)}_{\mathcal{B}\to\mathcal{L}}{}^{[q]}\right] B = \boldsymbol{M}^{[q]} B. \tag{10}$$

Each set of new control values bounds the Jacobian determinant on its own subdomain and we have:

$$b'_{\min} = \min_{i,q} b_i^{[q]} \;\le\; J_{\min} \;\le\; \min_{i<K_f,q} b_i^{[q]} \tag{11}$$

and

$$\max_{i<K_f,q} b_i^{[q]} \;\le\; J_{\max} \;\le\; b'_{\max} = \max_{i,q} b_i^{[q]}. \tag{12}$$

If an estimate is not sufficiently sharp, we can thus simply subdivide the appropriate parts of the element. This leads to a simple adaptive algorithm, exemplified in Figure 4. In this particular case the original estimate (6)-(7) is not sharp enough ($J_{\min} \in [-3, 1]$). After one subdivision, the Jacobian determinant is proved to be positive on the second subdomain. The first subdomain is thus subdivided once more, which proves the validity. In practice, a few levels of refinement lead to the desired accuracy. The subdivision has quadratic speed of convergence [22,23].

Note that in a practical implementation (with finite precision arithmetic), we must take care of a problematic situation. If the minimum of the Jacobian determinant is too close to zero but positive, then the upper bound is positive while the lower bound might never get positive. In order to avoid this situation, we limit the number of consecutive subdivisions that can be applied. The undetermined elements are then considered as invalid. Another way of getting rid of this issue is to relax the condition of rejection.

Extension to Higher Dimensions The extension of the method to higher dimensions is straightforward, provided that Bézier functions can be generated and that a subdivision scheme is available. Jacobian determinants J are polynomials of ξ, η in 2D and of ξ, η, ζ in 3D.

For high order triangles, the Bézier triangular polynomials are defined as

$$\mathcal{T}^{(p)}_{i,j}(\xi,\eta) = \binom{p}{i}\binom{p-i}{j}\, \xi^i\, \eta^j\, (1-\xi-\eta)^{p-i-j} \qquad (i+j \le p).$$

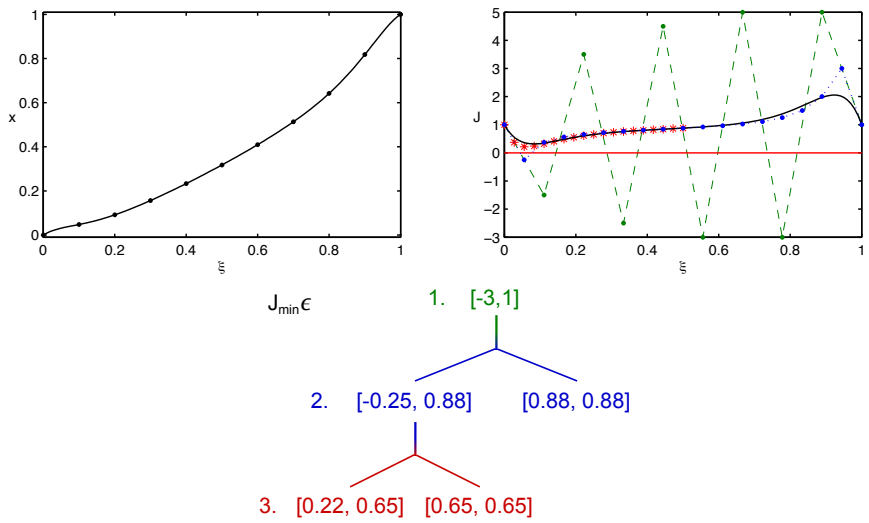

Fig. 4. Top left: One-dimensional element mapping $x(\xi)$. Top right: Exact Jacobian determinant $J(\xi)$ (black), control values on the original control points (green) and two adaptive subdivisions (blue and red). Bottom: Estimates of $J_{\min}$ at each step in the adaptive subdivision process.

It is possible to interpolate any polynomial function of order at most p on the unit triangle $\xi > 0, \eta > 0, \xi + \eta < 1$ as an expansion into Bézier triangular polynomials. Recalling that, for a triangle at order p, its Jacobian determinant $J(\xi, \eta)$ is a polynomial in ξ and η at order at most $n = 2(p-1)$, we can write

$$J(\xi, \eta) = \sum_{i+j \leq n} b_{ij} \mathcal{T}_{i,j}^{(n)}(\xi, \eta).$$

It is also possible to compute J in terms of Lagrange polynomials

$$J(\xi, \eta) = \sum_{i} J_i \mathcal{L}_i^{(n)}(\xi, \eta)$$

where the J_i are the Jacobian determinants calculated at Lagrange points. It is then easy to find a transformation matrix $T_{\mathcal{LB}}^n$ such that

$$B = T_{\mathcal{LB}}^n J,$$

where B and J are the vectors containing respectively the control values of the Jacobian determinant b_{ij} and the J_i's.

Other element shapes can be treated similarly. For quadrangles, tetrahedra, prisms and hexahedra, the Bézier are functions respectively:

$$\mathcal{Q}_{i,j}^{(p)}(\xi, \eta) = \mathcal{B}_i^{(p)}(\xi)\ \mathcal{B}_j^{(p)}(\eta) \qquad (i \leq p,\ j \leq p),$$

$$\mathcal{T}^{(p)}_{i,j,k}(\xi,\eta,\zeta) = \binom{p}{i}\binom{p-i}{j}\binom{p-i-j}{k}\ \xi^i\ \eta^j\ \zeta^k\ (1-\xi-\eta-\zeta)^{p-i-j-k} \qquad (i+j+k \leq p),$$

$$\mathcal{P}^{(p)}_{i,j,k}(\xi,\eta,\zeta) = \mathcal{T}^{(p)}_{i,j}(\xi,\eta)\ \mathcal{B}^{(p)}_k(\zeta) \qquad (i+j \leq p,\ k \leq p)$$

and

$$\mathcal{H}^{(p)}_{i,j,k}(\xi,\eta,\zeta) = \mathcal{B}^{(p)}_i(\xi)\ \mathcal{B}^{(p)}_j(\eta)\ \mathcal{B}^{(p)}_k(\zeta) \qquad (i \leq p,\ j \leq p,\ k \leq p).$$

Matrices of change of coordinates can then be computed inline for every polynomial order, and bounds of Jacobian determinants computed accordingly. Table 1 summarizes the order of the Jacobian determinant and the number of coefficients in its expansion for all common element types. In all cases the subdivision scheme works exactly in the same way as for lines. Figure 5 shows the first level of subdivision for a third-order triangle.

	Order (n) of J	Number of coefficients
Line	$p-1$	$n+1$
Triangle	$2(p-1)$	$(n+1)(n+2)/2$
Quadrangle	$2p-1$	$(n+1)^2$
Tetrahedron	$3(p-1)$	$(n+1)(n+2)(n+3)/6$
Prism	$3p-1$	$(n+1)^2(n+2)/2$
Hexahedron	$3p-1$	$(n+1)^3$

Table 1. Order of the Jacobian determinant and number of coefficients in the expansion for an element of order p.

Implementation As mentioned in Section 4.1, the bounds on the Jacobian determinant can be used to either make the distinction between valid and invalid elements with respect to a condition on $J_{\min}$, or to measure the quality of the elements by systematically computing $J_{\min}$ and $J_{\max}$ with a prescribed accuracy.

In both cases the same operations are executed on each element. First, the Jacobian determinant is sampled on a determined number of points N_s, equal to the dimension of the Jacobian determinant space, and so to the number of Bézier functions. Second, Bézier values are computed. Then adaptive subdivision is executed if necessary. Algorithms 1 and 2 show in pseudo-code the algorithm used to determine whether the Jacobian determinant of the element is everywhere positive or not.

Algorithm 2 could be further improved by optimzing the loop on line 5, by first selecting q for which we have the best chance to have a negative Jacobian determinant (line 4, algo 2). In practice this improvement is not significant since the only case for which we can save calculation is for invalid elements—and the proportion of them which require subdivision in order to be detected is usually

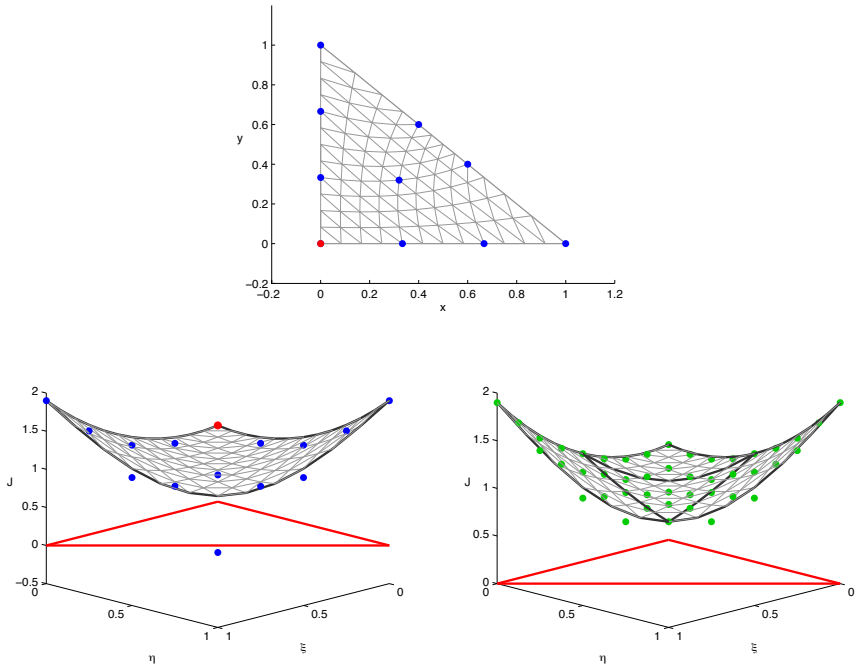

Fig. 5. Top: third-order planar triangle. Bottom left: exact Jacobian determinant and control values (dots) on the original control points; the validity of the element cannot be asserted. Bottom right: exact Jacobian determinant and control values (dots) after one subdivision; the element is provably correct.

small. Note that we may also want to find, for example, all the elements for which the Jacobian determinant is somewhere smaller than 20% of its average. We then just have to compute this average and replace the related lines (4 and 7 for algorithm 1).

Another possible improvement is to relax the condition of rejection. We could accept elements for which all control values are positive but reject an element as soon as we find a Jacobian determinant smaller than a defined percent of the average Jacobian determinant. The computational gain can be significant, since elements that were classified as good and which needed a lot of subdivisions (and have a Jacobian determinant close to zero) will be instead rapidly be detected as invalid.

More interestingly, the computation of sampled Jacobian determinants and the computation of Bézier control values in algorithm 1 can easily be executed for a whole groups of elements at the same time. This allows to use efficient BLAS 3 (matrix-matrix product) functions, which significantly speeds up the computations.

Algorithm 1: Check if an element is valid or invalid

Input: a pointer to an element.
Output: *true* if the element is valid, *false* if the element is invalid

set sampling points P_i, $i = 1, \ldots, N_s$;
compute Jacobian determinants J_i at points P_i;
for $i = 1$ **to** N_s **do**
    if $J_i <= 0$ **then return** *false*;
compute Bézier coefficients b_i, $i = 1, \ldots, N_s$ using (8);
$i = 1$;
while $i \leq N_s$ **and** $b_i > 0$ **do**
    $i = i + 1$;
if $i > N_s$ **then return** *true*;
call algorithm 2 with b_i as arguments and return *output*;

Algorithm 2: Compute the control values of the subdivisions

Input: Bézier coefficients b_i, $i = 1, \ldots, N_s$
Output: *true* if the Jacobian determinant on the domain is everywhere positive, *false* if not

compute new Bézier coefficients $b_i^{[q]}$, $q = 1, \ldots, Q$ as in equation (10);
for $q = 1$ **to** Q **do**
    for $i = 1$ **to** K_f **do**
        if $b_i^{[q]} <= 0$ **then return** *false*;
for $q = 1$ **to** Q **do**
    $i = 1$;
    while $i \leq N_s$ **and** $b_i^{[q]} > 0$ **do**
        $i = i + 1$;
    if $i \leq N_s$ **then**
        call algorithm 2 with $b_i^{[q]}$ as arguments and store *output*;
        if *output* = *false* **then return** *false*;
return *true*;

5 Untangling of high-order meshes

5.1 Strategy

Applying an untangling procedure to the entire mesh would result in a robust methodology. In most practical cases however, it represents an unnecessary expense of computational resources. Indeed, only the vertices and elements in the vicinity of the curved geometry really need to be modified in order to untangle the mesh, while the mesh entities located far from the boundary remain unchanged. In order to optimize the computational efficiency, the untangling procedure can thus be applied locally. Subdomains are first defined around each invalid element. Then, two strategies can be adopted in the framework of an optimization procedure.

Primary subdomains The first step in the untangling procedure consists in identifying all invalid elements in the mesh, for instance by means of the methods described in Section 4. A subdomain of N layers of elements can then be built around each of the invalid elements. Although these subdomains are usually appropriate in isotropic meshes, they may not be adequate for boundary layer-type meshes, where elements are highly anisotropic. In this case, the boundary curvature responsible for the mesh tangling is typically large compared to the normal size of the tangled element, but small compared to its tangential size (see Figure 6). Thus, untangling such a mesh requires several layers to be curved in the direction normal to the boundary, whereas there is no need to modify the elements that are adjacent in the tangential direction. Therefore, a geometrical criterion is introduced in the construction of the subdomains, as illustrated in Figure 6: among the elements contained in the N layers surrounding the invalid element, only those located within a certain distance are retained. This distance is defined by multiplying the distance between the straight-sided and high-order boundaries by a user-defined factor. The boundaries of the subdomain are fixed, in order to ensure that elements lying outside do not get invalidated.

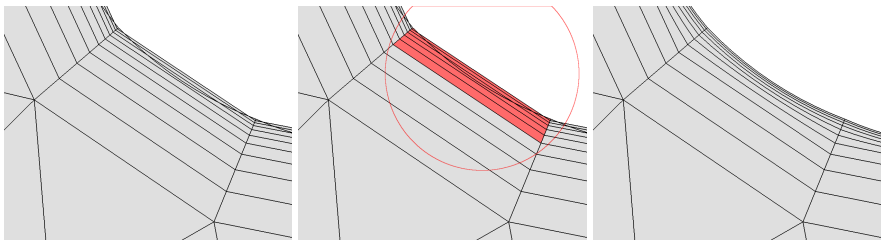

Fig. 6. Detail of a boundary-layer mesh on a curved geometry: tangled second-order mesh (left), subdomain definition with a circle representing the geometrical criterion (center), and untangled mesh (right).

The optimal values for the number of layers N and the distance factor are those that encompass the lowest number of elements while still leaving the untangling procedure enough freedom to untangle the mesh (or reach a target value for the Jacobian determinant). They are case-dependent. Note that in 3D, it is desirable to make sure that the subdomain includes at least one layer of elements around all faces of the invalid element, because the latter is likely to have tangled faces that cannot be repaired if they lie on the boundary of the subdomain. For the optimization procedure described in Section 5.2, a fair trade-off between computation cost and robustness can be obtained with $N = 6$ layers of quadrangles (or $N = 12$ layers of triangles) and a distance factor of 12 in many applications involving boundary-layer meshes.

In complex cases, a unique value of these parameters for the whole mesh may lead to subdomains that are too large for some invalid elements, but too small for others. It is then beneficial to use the untangling procedure in an adaptive loop where the subdomain size is progressively increased in case the procedure fails to untangle the mesh.

Overlapping subdomains A potential problem with the subdomains created as described above is that they may overlap. There is no guarantee that the untangling procedure can be efficiently applied to each of these subdomains. Let us consider the case of a subdomain built around an invalid element, that contains another invalid element close to its boundary: this subdomain may not provide the necessary degrees of freedom to fix the second invalid element, thus the untangling procedure can fail in this subdomain. In order to avoid such problems, two strategies can be applied:

Disjoint subdomains Overlapping subdomains are detected and merged. This strategy ensures that each invalid element in a subdomain has enough surrounding elements for the untangling procedure to be successful, provided the primary subdomains are large enough. It is also well-suited for a parallel setting in which each subdomain would be treated by a processor. However, it may lead to large subdomains, which can prove inappropriate if the cost of the untangling procedure does not scale well with the number of nodes and elements involved.

"One-by-one" strategy The untangling procedure is applied sequentially to each subdomain, with the objective of fixing only the invalid element around which it is built, while allowing other invalid elements in the same subdomain to remain broken. This strategy lets the untangling procedure work on small-size subdomains, but is less appropriate for parallel operation at subdomain level, because it is not possible to untangle concurrently subdomains that overlap.

5.2 Untangling through optimization

We chose to apply to each subdomain an untangling procedure that preserves the mesh topology. Its role is to move the mesh vertices in such a way that the

tangled elements become valid, while the other elements do not become invalid. The measure of the scaled Jacobian determinant characterizes the element validity in a continuously derivable manner with respect to the position of the vertices, which allows us to use an optimization algorithm. In this section, we derive an appropriate objective function relying on the robust Jacobian evaluation technique presented in Section 4, and explain how to efficiently compute its gradient. We then describe the method used to solve the optimization problem.

Computation of Bézier coefficients and their derivatives The general goal of our method is to untangle both surface and volume meshes. To this end, we consider that all points have three-dimensional coordinates $\boldsymbol{x} = \{x, y, z\}$. We also use three local coordinates $\boldsymbol{\xi} = \{\xi, \eta, \zeta\}$. For surface meshes however, where only two parametric coordinates are defined, we assume that the vector $\mathbf{n} = \partial \boldsymbol{x}/\partial\zeta$ is the constant unit normal vector to the straight sided element, with the same orientation convention used for the low-order mesh. This assumption makes it possible to compute the Jacobian determinant at every Lagrange node $\boldsymbol{\xi}_k = (\xi_k, \eta_k, \zeta_k)$ at order q:

$$J_k = J(\boldsymbol{\xi}_k) = \frac{\partial x}{\partial \xi}\frac{\partial y}{\partial \eta}\frac{\partial z}{\partial \zeta} + \frac{\partial z}{\partial \xi}\frac{\partial x}{\partial \eta}\frac{\partial y}{\partial \zeta} + \frac{\partial y}{\partial \xi}\frac{\partial z}{\partial \eta}\frac{\partial x}{\partial \zeta} - \frac{\partial z}{\partial \xi}\frac{\partial y}{\partial \eta}\frac{\partial x}{\partial \zeta} - \frac{\partial x}{\partial \xi}\frac{\partial z}{\partial \eta}\frac{\partial y}{\partial \zeta} - \frac{\partial y}{\partial \xi}\frac{\partial x}{\partial \eta}\frac{\partial z}{\partial \zeta}. \tag{13}$$

As

$$x = \sum_{i=1}^{N_p} x_i^e \mathcal{L}_i^{(p)}(\boldsymbol{\xi}_k),$$

we can easily express the sensitivity $\partial J_k/\partial x_i^e$ of the Jacobian determinant at point k with respect to x_i^e. The quantities $\partial J_k/\partial y_i^e$ and $\partial J_k/\partial z_i^e$ can be calculated in the same manner. In practice, we compute for each element e the matrix $\mathbf{J}$ of size $N_q \times (3N_p + 1)$ as:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial J_1}{\partial x_1^e} & \cdots & \frac{\partial J_1}{\partial x_{N_p}^e} & \frac{\partial J_1}{\partial y_1^e} & \cdots & \frac{\partial J_1}{\partial y_{N_p}^e} & \frac{\partial J_1}{\partial z_1^e} & \cdots & \frac{\partial J_1}{\partial z_{N_p}^e} & J_1 \\ \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots \\ \frac{\partial J_{N_q}}{\partial x_1^e} & \cdots & \frac{\partial J_{N_q}}{\partial x_{N_p}^e} & \frac{\partial J_{N_q}}{\partial y_1^e} & \cdots & \frac{\partial J_{N_q}}{\partial y_{N_p}^e} & \frac{\partial J_{N_q}}{\partial z_1^e} & \cdots & \frac{\partial J_{N_q}}{\partial z_{N_p}^e} & J_{N_q} \end{bmatrix}$$

Let $\mathbf{B}$ be the matrix

$$\mathbf{B} = \begin{bmatrix} \frac{\partial B_1}{\partial x_1^e} & \cdots & \frac{\partial B_1}{\partial x_{N_p}^e} & \frac{\partial B_1}{\partial y_1^e} & \cdots & \frac{\partial B_1}{\partial y_{N_p}^e} & \frac{\partial B_1}{\partial z_1^e} & \cdots & \frac{\partial B_1}{\partial z_{N_p}^e} & B_1 \\ \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots \\ \frac{\partial B_{N_q}}{\partial x_1^e} & \cdots & \frac{\partial B_{N_q}}{\partial x_{N_p}^e} & \frac{\partial B_{N_q}}{\partial y_1^e} & \cdots & \frac{\partial B_{N_q}}{\partial y_{N_p}^e} & \frac{\partial B_{N_q}}{\partial z_1^e} & \cdots & \frac{\partial B_{N_q}}{\partial z_{N_p}^e} & B_{N_q} \end{bmatrix}, \tag{14}$$

that contains both the Bézier coefficients B_l as well as their gradients with respect to the position of the nodes of element e. Defining the constant transformation matrix $T_{lk}^q = \mathcal{B}_l^{(q)}(\boldsymbol{\xi}_k)$ that gives the Bézier coefficients B_l from the

Lagrange coefficients J_k, $\mathbf{B}$ can be calculated through a single matrix-matrix product: $B_{lj} = T^q_{lk} J_{kj}$.

We can then make use of the B_i's and their gradients in a gradient-based optimization procedure. To this end, an objective function that allows us to control the quality of elements based on the coefficients B_i needs to be constructed.

Objective function In what follows, we detail the objective function $f(\boldsymbol{x}_i^e)$ that is employed in an unconstrained optimization procedure to untangle invalid curved elements. The function $f = \mathcal{E} + \mathcal{F}$ is composed of two parts $\mathcal{E}$ and $\mathcal{F}$.

The first part $\mathcal{E}$ is based on the assumption is that the method is provided with a straight-sided mesh of high quality. This mesh has potentially been defined to satisfy multiple criteria, such as a predetermined size field, or anisotropic adaptation. The conversion such meshes to high order is expected to preserve as much as possible all these features. Therefore, the nodes shall be kept as close as possible to their initial location in the straight sided mesh.

To this end, we define the function $\mathcal{E}$ by analogy with an *energy* associated with the displacement $\boldsymbol{x}_i^e - \boldsymbol{X}_i^e$ of the nodes, i.e. as a positive quadratic form that is a measure of the distance between the straight sided nodes $\boldsymbol{X}_i^e$ and their position $\boldsymbol{x}_i^e$ in the curved mesh:

$$\mathcal{E}(\boldsymbol{x}_i, \mathbf{K}) = \frac{1}{2} \sum_e \sum_{i=1}^{N_p} \sum_{j=1}^{N_p} (\boldsymbol{x}_i^e - \boldsymbol{X}_i^e) \mathbf{K}_{ij}^e (\boldsymbol{x}_j^e - \boldsymbol{X}_j^e) \geq 0 \tag{15}$$

where $\mathbf{K}$ is a symmetric positive matrix of size $3n_v \times 3n_v$ and $\mathbf{K}_{ij}^e$ is of size 3×3. Here, we define $\mathbf{K}$ as the diagonal matrix with entries w_i^e / L^2, where w_i^e are user-defined weights used to set the balance between the two parts $\mathcal{E}$ and $\mathcal{F}$ of the objective function f, and L is a length scale representative of the problem. We choose L as the maximum distance, among all vertices of the initial tangled mesh, between a node and its counterpart in the straight-sided mesh. Regarding the non-dimensional weights w_i^e, we usually set $w_i^e = 1000$ if the node i of element e lies on the boundary, and $w_i^e = 1$ otherwise. In practice, we observe that the exact value of w_i^e has a limited influence on the convergence of the optimization method. The part $\mathcal{E}$ in f prevents the problem from being under-determined, and it steers the optimization procedure towards a solution that preserves the features of the straight-sided mesh, but the term $\mathcal{F}$ dominates for the most problematic (tangled) elements that drive the mesh deformation.

The second part $\mathcal{F}$ of the functional controls the positivity of the Jacobian determinant. A log barrier[24] prevents Jacobians from becoming too small, and a quadratic function is used to penalize Jacobians that are too large:

$$\mathcal{F}(\boldsymbol{x}_i, \epsilon) = \sum_{e=1}^{n_e} \sum_{l=1}^{N_q} F_l^e(\boldsymbol{x}_i^e, \epsilon)$$

where

$$F_l^e(\boldsymbol{x}_i^e, \epsilon) = \left[\log \left(\frac{B_l^e(\boldsymbol{x}_i^e) - \epsilon J_0^e}{J_0^e - \epsilon J_0^e} \right) \right]^2 + \left(\frac{B_l^e(\boldsymbol{x}_i^e)}{J_0^e} - 1 \right)^2. \tag{16}$$

The function $\mathcal{F}$ is designed to blow up when $B_l^e = \epsilon J_0^e$, but still vanishes whenever $B_l^e = J_0^e$. Barrier methods are one of the most successful class of algorithms used for general nonlinear optimization problems. These techniques are very robust with respect to the convexity characteristics of the objective function and constraints[25], so they converge to at least a local minimum in most cases. Figure 7 shows a plot of the barrier function in Equation (16) for $\epsilon = 0.2$.

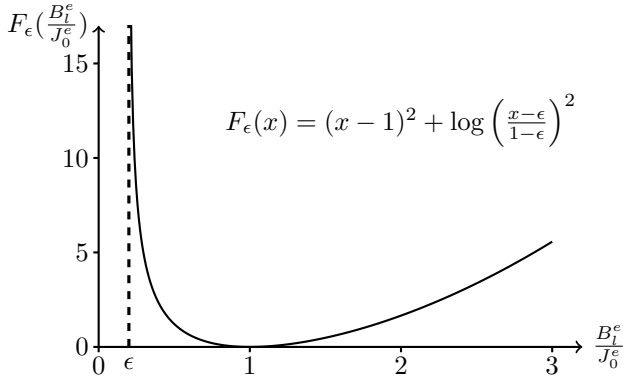

Fig. 7. Plot of the barrier function $F(J_l^e)$ for $\epsilon = 0.2$.

As the objective function f is smooth, we can compute its gradient ∇f with respect to the positions of the element vertices $\boldsymbol{x}_i^e$. This gradient can then be used in a gradient-based optimization procedure. As explained in Section 3.2, each vertex is classified on a given model entity, to which it is geometrically linked. In order for the vertices to remain on their model entity, the sensitivity of f is computed with respect to the location of vertices expressed in the parametric space of the model entities rather than in the physical space.

A mesh vertex M_i^0 classified on a model edge G_j^1 can only be moved along G_j^1, i.e. its position only depends on one curve parameter t. The corresponding component of the gradient will thus be computed as

$$\frac{df}{dt} = \frac{\partial f}{\partial \boldsymbol{x}_i^e} \cdot \frac{d\boldsymbol{x}_i^e}{dt}$$

with $d\boldsymbol{x}_i^e/dt$ the tangent vector to the curve at point t, that is obtained from the CAD model.

A mesh vertex M_i^0 classified on a model face G_j^2 can only be moved along the surface. Two parameters u and v are associated to such a vertex, and the corresponding sensitivities $\partial f/\partial u$ and $\partial f/\partial v$ depend respectively on $\partial \boldsymbol{x}_i^e/\partial u$ and $\partial \boldsymbol{x}_i^e/\partial v$ the two tangent vectors to the surface at point (u, v). Those can be computed using the CAD model.

A vertex that is classified on a model region has complete freedom to move in every direction of the 3D space, in which case the physical coordinates $\{x, y, z\}$

are used. Finally, a mesh vertex that is classified on a model vertex has no freedom to move, it is thus excluded from the optimization problem.

Optimization method The general optimization procedure, that involves a moving barrier for the objective function, is detailed in Algorithm 3.

Algorithm 3: Optimization method

Define subdomains $\mathcal{B}_k$, $k = 1 \dots N_{\mathcal{B}}$;
for $k = 1$ **to** $N_{\mathcal{B}}$ **do**
  repeat
    compute $\kappa = \min_e \min_l \frac{B_l^e}{J_0^e}, \quad e \in \mathcal{B}_i, \ l \ \in [1, N_q]$;
    if $\kappa < \bar{\epsilon}$ **then**
      set $\epsilon = \kappa - 0.1 \ |\kappa|$;
      solve $\min_{\boldsymbol{x}i} f(\boldsymbol{x}_i, \mathbf{K}, \epsilon)$ for all elements of subdomain $\mathcal{B}_k$;
      recompute $\kappa = \min_e \min_l \frac{B_l^e}{J_0^e}, \quad e \in \mathcal{B}_i, \ l \ \in [1, N_q]$;
  until $\kappa \geq \bar{\epsilon}$;

The untangling problem can be formally defined as

$$\min_{\boldsymbol{x}_i} f(\boldsymbol{x}_i, \mathbf{K}, \epsilon), \quad i = 1, \dots, n_v.$$

A broad range of methods can be used to solve such an unconstrained minimization problem. We have tested several alternatives: interior point methods implemented in the software package IPOPT [26], as well as L-BFGS [27] and conjugate gradients [28] algorithms provided by ALGLIB [29]. In our experience, the use of conjugate gradients seems to be the best choice in terms of computational efficiency.

A crucial aspect of the optimization procedure is the definition of an appropriate sequence of optimization problems. In a given subdomain, the value of the barrier ϵ must be lower than the worst scaled Jacobian determinant among all elements, so that the variables remain in the domain of definition of the barrier function $\mathcal{F}$. In particular, ϵ has to be negative for an initially tangled mesh. Therefore, we compute a sequence of optimization problems with "moving barriers": ϵ is increased between each optimization problem, until the desired barrier value $\bar{\epsilon}$ is reached. This procedure is illustrated in Figure 8.

In practical cases, it is most often necessary to apply preconditioning to the optimization problem, because the scale of the parametric or physical coordinates used for different mesh vertices can differ by orders of magnitude, depending on the model entities on which they are classified. We found that a simple diagonal preconditioner, based on the norm of the tangent vectors $d\boldsymbol{x}_i^e/dt$ for vertices classified on model edges (respectively $\partial \boldsymbol{x}_i^e/\partial u$ and $\partial \boldsymbol{x}_i^e/\partial v$ for vertices classified on model faces), and unity for vertices classified on model regions, allows the optimizer to converge in a fast and robust manner.

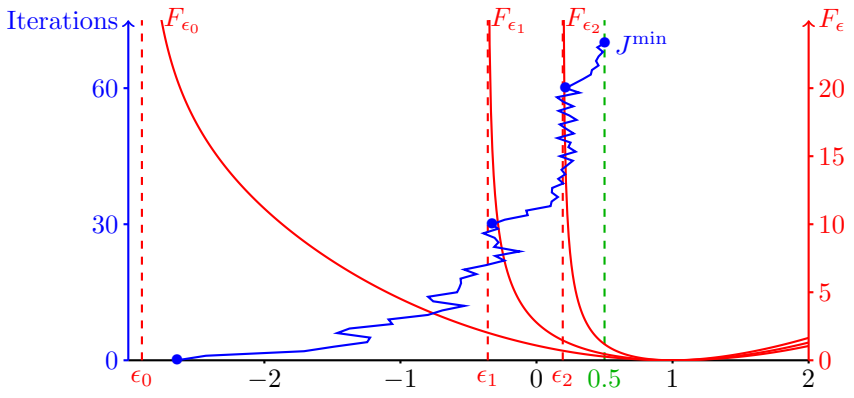

Fig. 8. Optimization procedure: three successive series of (maximum) 30 conjugate gradient iterations, with their respective log barriers.

In practice, the value of J_0^e appearing in Algorithm 3 represents the Jacobian determinant of the straight-sided version of element e in the original mesh. It is computed at initialization and held constant during the optimization process, which saves computational time and favors curved elements that resemble their first-order counterpart in the original mesh. However, all vertices of element e are allowed to move, so the value of J_0^e after optimization is different from the original one. The measure of the minimum scaled Jacobian determinant in the untangled mesh may thus yield a lower value than the barrier $\bar{\epsilon}$. Nevertheless, the positivity of the Jacobian determinant is not threatened as long as the optimization procedure is successful.

5.3 Untangling through analytical methods

In boundary layers meshes used in CFD, where elements are highly stretched along the boundary, it is relatively clear that the curvature should "propagate" in the direction normal to the boundary in order to untangle the mesh. In this case, it may be possible to apply simple techniques to determine in an analytical fashion the displacement that high-order nodes shall undergo for the mesh to become valid.

Such a method is illustrated in Figure 9. In a boundary layer mesh, the subdomains created following the methodology described in Section 5.1 typically consist of a "stack" of elements above the tangled element. It is then possible to define a "meta-element" that includes the whole subdomain under consideration, based on the curved edge or face of the tangled mesh element. In 2D, the meta-element is a quadrangle created from the curved boundary edge. In 3D, the meta-element is either a prism created from a curved boundary triangle, or an hexahedron created from a curved boundary quadrangle.

Let us consider the mapping between the straight-sided version and the curved version of the meta-element, as described in Section 4.1. If the meta-element is sufficiently large in the direction normal to the boundary, its curved version is valid in the sense of Section 4.1 and the mapping is injective. All mesh vertices in the subdomain are located within the straight meta-element. We can then apply the straight-to-curved meta-element mapping to them: their images are located in the curved subdomain in a way that untangle the mesh by distributing the element curvature along the direction normal to the boundary.

The straight-to-curved meta-element transformation is defined by the composition $\mathbf{X} \circ \mathbf{x}^{-1}$, following the notation of Section 4.1 illustrated in Figure 3. As the straight-to-reference meta-element mapping $\mathbf{x}(\boldsymbol{\xi})$ is bilinear, its inversion is computationally inexpensive. So is the forward high-order transformation $\mathbf{X}(\boldsymbol{\xi})$ from the reference to the curved meta-element. This interpolation technique is thus much more efficient than the iterative optimization procedure presented in Section 5.2.

This method is particularly interesting in boundary layer-type meshes, where most elements are created from columns of vertices that are aligned in the direction normal to the boundary. In this case, the meta-element can be chosen to have the same boundaries as the column of elements above the invalid one. Thus, all the elements in the column undergo a consistent deformation, and the untangling is successful as long as the meta-element is valid. When vertices are not aligned, as in boundary layer meshes near complex geometric features or fully unstructured meshes, different vertices of a same element can belong to different meta-elements, which may not lead to a valid element. Even then, it can be beneficial to apply the analytical method to correct most invalid elements, before using the optimization method described in Section 5.2 to untangle the rest of the mesh in a robust manner.

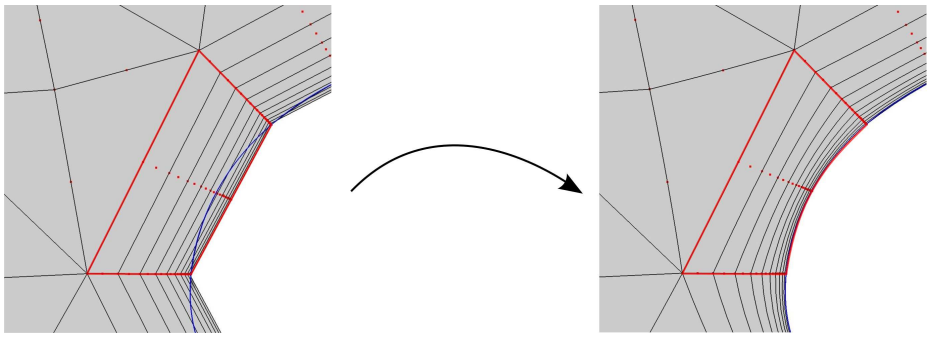

Fig. 9. Illustration of the analytical curving technique: the "meta-element" is shown with red lines and the high-order mesh nodes that are moved to untangle the mesh are marked in red.

6 Conclusion

The techniques presented in this article prove valuable for the creation, diagnostic and regularization of high-order meshes. They can be used as the basis of a robust, efficient and versatile high-order mesh generation framework. They have already been demonstrated to produce valid meshes of relatively high order for a wide variety of industrial and scientific problems. For a more thorough discussion of these methods and examples of applications, the reader is referred to Ref. [1,2]. An efficient and ready-to-use implementation can be found in the free software Gmsh [3,4].

Nevertheless, two topics have not been addressed in this article. The first one is the geometrical accuracy of the curvilinear mesh: the quality of the approximation of the CAD geometry by the mesh depends on the location of the high-order nodes on the boundary. The second one is the numerical properties of high-order meshes beyond the mere validity: their curvilinear nature affects numerical schemes in terms of interpolation properties and conditioning. These aspects should ideally be taken into account in the mesh generation process, but have been the subject of little interest from the scientific community so far. They should be investigated more intensely in the future.

References

1. Johnen, A., Remacle, J.F., Geuzaine, C.: Geometrical validity of curvilinear finite elements. Journal of Computational Physics **233** (2013) 359–372
2. Toulorge, T., Geuzaine, C., Remacle, J.F., Lambrechts, J.: Robust untangling of curvilinear meshes. Journal of Computational Physics **254** (2013) 8–26
3. Geuzaine, C., Remacle, J.F.: Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. International Journal for Numerical Methods in Engineering **79**(11) (2009) 1309–1331
4. Geuzaine, C., Remacle, J.F.: Gmsh website. `http://geuz.org/gmsh` (March 2014)
5. Cockburn, B., Karniadakis, G., Shu, C.W., eds.: Discontinuous Galerkin Methods. Volume 11 of Lecture Notes in Computational Science and Engineering. Springer, Berlin (2000)
6. Kroll, N., Bieler, H., Deconinck, H., Couaillier, V., Van Der Ven, H., Sorensen, K., eds.: ADIGMA – A European Initiative on the Development of Adaptive Higher-Order Variational Methods for Aerospace Applications. Notes on Numerical Fluid Mechanics and Multidisciplinary Design. Springer (2010)
7. Bassi, F., Rebay, S.: High-order accurate discontinuous finite element solution of the 2D Euler equations. Journal of Computational Physics **138**(2) (1997) 251–285
8. Dey, S., O'Bara, R.M., Shephard, M.S.: Curvilinear mesh generation in 3D. In: Proceedings of the 8th International Meshing Roundtable. John Wiley & Sons (1999) 407–417
9. Gargallo-Peiró, A., Roca, X., Peraire, J., Sarrate, J.: High-order mesh generation on CAD geometries. In: Proceedings of the VI International Conference on Adaptive Modeling and Simulation (ADMOS 2013). CIMNE, Barcelona, Spain (2013)
10. Xie, Z.Q., Sevilla, R., Hassan, O., Morgan, K.: The generation of arbitrary order curved meshes for 3D finite element analysis. Computational Mechanics **51**(3) (2013) 361–374

11. Sherwin, S.J., Peiró, J.: Mesh generation in curvilinear domains using high-order elements. International Journal for Numerical Methods in Engineering **53**(1) (2002) 207–223
12. Abgrall, R., Dobrzynski, C., Froehly, A.: A method for computing curved 2D and 3D meshes via the linear elasticity analogy: preliminary results. Rapport de recherche RR-8061, INRIA (September 2012)
13. Lu, Q., Shephard, M., Tendulkar, S., Beall, M.: Parallel curved mesh adaptation for large scale high-order finite element simulations. In: Proceedings of the 21st International Meshing Roundtable. Springer Berlin Heidelberg (2013) 419–436
14. Persson, P.O., Peraire, J.: Curved mesh generation and mesh refinement using lagrangian solid mechanics. In: Proceedings of the 47th AIAA Aerospace Sciences Meeting and Exhibit, Orlando (FL), USA, 5-9 January 2009. (2009)
15. Roca, X., Gargallo-Peiró, A., Sarrate, J.: Defining quality measures for high-order planar triangles and curved mesh generation. In Quadros, W., ed.: Proceedings of the 20th International Meshing Roundtable. Springer Berlin Heidelberg (2012) 365–383
16. Luo, X., Shephard, M., O'Bara, R., Nastasia, R., Beall, M.: Automatic p-version mesh generation for curved domains. Engineering with Computers **20**(3) (2004) 273–285
17. Sahni, O., Luo, X., Jansen, K., Shephard, M.: Curved boundary layer meshing for adaptive viscous flow simulations. Finite Elements in Analysis and Design **46**(1-2) (2010) 132–139
18. Remacle, J.F., Shephard, M.S.: An algorithm oriented mesh database. International Journal for Numerical Methods in Engineering **58**(2) (2003) 349–374
19. Dey, S., Shephard, M.S., Flaherty, J.E.: Geometry representation issues associated with p-version finite element computations. Computer methods in applied mechanics and engineering **150**(1) (1997) 39–55
20. Sevilla, R., Fernández-Méndez, S., Huerta, A.: NURBS-enhanced finite element method (NEFEM). International Journal for Numerical Methods in Engineering **76**(1) (2008) 56–83
21. Taylor, R.L.: On completeness of shape functions for finite element analysis. International Journal for Numerical Methods in Engineering **4**(1) (1972) 17–22
22. Lane, J.M., Riesenfeld, R.F.: A theoretical development for the computer generation and display of piecewise polynomial surfaces. IEEE Transactions on Pattern Analysis and Machine Intelligence **2**(1) (1980) 35–46
23. Cohen, E., Schumacker, L.L.: Rates of convergence of control polygons. Computer Aided Geometric Design **2** (1985) 229–235
24. Freitag, L., Knupp, P., Munson, T., Shontz, S.: A comparison of optimization software for mesh shape-quality improvement problems. Technical report, Argonne National Lab., IL (US) (2002)
25. Fiacco, A., McCormick, G.: Nonlinear programming: sequential unconstrained minimization techniques. Volume 4. Society for Industrial Mathematics (1990)
26. Waechter, A., Laird, C., Margot, F., Kawajir, Y.: Introduction to IPOPT: A tutorial for downloading, installing, and using IPOPT (2009)
27. Liu, D., Nocedal, J.: On the limited memory BFGS method for large scale optimization. Mathematical programming **45**(1) (1989) 503–528
28. Fletcher, R., Reeves, C.: Function minimization by conjugate gradients. The Computer Journal **7**(2) (1964) 149–154
29. Bochkanov, S.: ALGLIB. `http://www.alglib.net` (Dec. 2013)