Universitat Politècnica de Catalunya
Programa de Doctorat de Matemàtica Aplicada

Departament de Matemàtica Aplicada III

Paving the path towards automatic hexahedral mesh generation

by Xevi Roca Navarro

Doctoral thesis
Advisor: Josep Sarrate

Barcelona, June 8, 2009

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Spatial discretizations, represented by a *mesh*, have been related with computational methods since the appearance of the first electronic computers. Later, the evolution of computer graphics capabilities has induced to use meshes in new areas of application such as medical imaging, scientific visualization and 3D animation. However, automatic mesh generation was first prompted by many applications of computational methods in applied science and engineering. Specifically, the generation of a mesh is a pre-requisite for the application of several numerical techniques including the finite difference method (FDM), the finite element method (FEM), and the finite volume method (FVM).

The application of the above numerical techniques to 3D simulations leads to meshes composed by polyhedral *elements*. In these applications the most common types are the *tetrahedral* (four triangular faces) and *hexahedral* (six quadrilateral faces) elements. Three fast and robust approaches have been developed to automatically generate tetrahedral meshes for arbitrary domains: the advancing front technique (Lohner et al., 1985; Peraire et al., 1987, 1988; Lohner and Parikh, 1988), Delaunay based methods (Baker, 1987; George et al., 1988), and the Octree approach (Shephard and Georges, 1991; Yerry and Shephard, 1984). In these methods local connectivity modifications are a crucial step. However, in hexahedral meshes the connectivity modifications propagate through the mesh. Therefore, hexahedral meshes are more constrained and only a limited type of geometries can be automatically meshed with high-quality hexahedral elements.

In industrial applications, the semi-automatic process for obtaining a hexahedral mesh, see Section 1.2, is often the most time-consuming task of the whole analysis. Nevertheless, hexahedral elements are preferred in a wide range of applications, for instance:

- In elastic and elasto-plastic analysis, eigenvalues of stiffness matrix for hexahedra are smaller than those for tetrahedra. Specifically, the tetrahedron is too stiff and locks in bending tests (Benzley et al., 1995).

- In structural analysis empirical studies have shown that hexahedral elements provide more accuracy than tetrahedral elements for the same computational cost (Cifuentes and Kalbag, 1992; Weingarten, 1994). To obtain similar accuracy a tetrahedral mesh usually requires between four and ten times more elements than a hexahedral mesh.

- In Navier-Stokes computations elements with high aspect ratio are required at boundary layers. Stretched hexahedra perform better than stretched tetrahedra capturing the anisotropy of the flow field over such viscous regions.

- In structural dynamics simulations of composite materials elements strictly aligned with material features are required. Hexahedral elements reproduce better than tetrahedra the anisotropic properties of composite materials.

These advantages hold when comparing trilinear hexahedra versus linear tetrahedra but not in the case of their high-order versions. Quadratic tetrahedra require less computational effort than quadratic hexahedra to obtain similar accuracy (Cifuentes and Kalbag, 1992; Weingarten, 1994). However, hexahedra can also be suitable for high-order applications of the FEM. For instance, the *spectral element method* (SEM) is an accurate and efficient technique that explodes the characteristics of hexahedral elements (Patera, 1984). Efficiency is achieved by using the tensor product of Gauss-Lobatto points to determine the location of both interpolation and integration points. This results in diagonal mass matrices and the adequacy to parallel implementations. Therefore, special attention is focused on developing automatic algorithms to generate hexahedral meshes.

1.2 Generating hexahedral meshes

During the last decades the FEM, the FVM, and the FDM have become essential tools in applied science and engineering. However, in industrial applications, they are often hampered by the conversion of the *computer aided design* (CAD) model into a mesh adapted to the details of the geometry and to the prescribed distribution of the element size. Generating a mesh of the model can represent the ninety per cent of the analysis time (Halpern, 1997). In this process the most time-consuming tasks are:

- *Geometry healing.* The CAD model is repaired in order to obtain a water-tight geometry of the domain to analyze: rebuilding missing curves and surfaces; eliminating duplicated entities; and matching points, curves and surfaces properly.

- *Geometry de-featuring.* The details not relevant to the analysis being performed are removed.

- *Geometry decomposition.* It could be required to decompose the healed and de-featured geometry into simpler sub-volumes.

- *Generating meshes.* Finally, the user sets the meshing parameters and assigns the desired mesh algorithm to each one of the sub-volumes.

For tetrahedral meshes, once the geometry is healed and de-featured the user can generate meshes without performing geometry decomposition. The mesh generation process is fast, robust and automatic. However, for hexahedral meshes, the geometry decomposition task is the bottleneck of process.

1.2.1 Geometry representation

Mesh generation begins with a proper representation of the geometry to be meshed. The three most usual methodologies to represent geometry are:

- *Constructive solid geometry* (CSG). The domains are defined by means of Boolean unions, differences and intersections of a set of primitive geometric shapes including boxes, spheres, ellipsoids, and cones.

- *Faceted representations.* The geometry of the boundaries is described with polygonal meshes, usually composed by triangular and quadrilateral elements.
- *Boundary representation* (B-rep). The geometry is defined in terms of parameterized curves and surfaces that determine the boundaries of the domain. In CAD applications these parameterizations are usually determined by *non-uniform rational B-splines* (NURBS), see Piegl and Tiller (1997).

In this dissertation we assume that we have a B-rep of a CAD model. Hence, the model is determined by a set of topological and geometrical entities. On the one hand, the connectivity between different model entities is determined by a set of topological entities: *vertices* (0D), *edges* (1D), *faces* (2D), and *solids* (3D). On the other hand, the shape of the model is represented by the following geometrical entities: *point* (0D), *curve* (1D), *surface* (2D), and *volume* (3D).

1.2.2 Geometry decomposition

To generate a hexahedral mesh, experienced engineers have to manually decompose the healed and de-featured geometry into simpler sub-volumes. This process is repeated until each sub-volume can be meshed with existent automatic hexahedral mesh generators. There are two main decomposition strategies: the divide and conquer approach, and the multi-block approach. To apply these approaches powerful graphical mesh generation environments have been developed, see Chapter 6. These packages provide a set of graphical tools for manual and semi-automatic decomposition of domains.

Divide and conquer approach. In this approach the initial geometry is decomposed until each sub-volume can be meshed with existent automatic hexahedral mesh generators, such as mapping (Cook and Oakes, 1982), submapping (Whiteley et al., 1996; Ruiz-Gironès and Sarrate, 2009, 2008), and sweeping (Knupp, 1998, 1999; Blacker, 1996; Mingwu and Benzley, 1996; Staten et al., 1999; Scott et al., 2005). In general a large number of the resulting sub-volumes are extrusion geometries that can be meshed with the *sweeping* technique. Hence, it is of the major importance to improve the robustness and the mesh quality of this technique, see Chapters 2, 3, and 4.

The decomposition is mainly a geometrical and interactive process, where the engineer has to decide how to divide a given domain. To this end, several mesh generation environments have been developed during the last decades (Sandia National Labs, 2009; ANSYS, 2009b; Simulia, 2009). These environments provide a set of specific geometric operations that are not usually available in the CAD packages. These operations allow performing the decomposition by cutting existing entities with: planar and cylindrical surfaces, extensions of surfaces, surfaces filling holes between curves, and extrusions of curves.

Multi-block approach. In this approach the domain is manually divided in an ultra-coarse hexahedral mesh, *blocks*. Then, structured hexahedral meshes can be generated within each block. The regularity of the meshes inside each block allows storing them in data structures with a low memory cost. Furthermore, the numerical solvers for these meshes have also a simple and fast implementation.

For complex geometries a large number of blocks have to be generated. Therefore, an experienced user is needed to perform this task with the help of multi-block graphical user interfaces (Program Development Company, 2009; Pointwise, 2009; ANSYS, 2009a). Nevertheless, this is the preferred mesh generation method for computational fluid dynamics (CFD) simulations. This methodology provides the required control and layering of the mesh near the viscous boundaries.

1.3 Automatic hexahedral mesh generation

During the last decades several general-purpose algorithms for fully automatic hexahedral mesh generation have been proposed. These algorithms are described and classified in the surveys by Owen (1998), Blacker (2001), Tautges (2001), Baker (2005), and Shepherd (2007). Note that none of the existent algorithms is robust, automatic and generates high-quality meshes for any initial geometry. On the one hand, *grid-based* methods (Yerry and Shephard, 1984; Shephard and Georges, 1991; Schneiders and Bünten, 1995; Schneiders, 1996; Zhang and Bajaj, 2006; Zhang et al., 2005) are the only family of robust and fully automatic hexahedral mesh generation algorithms. However, the meshes obtained with grid-based methods have several disadvantages, low quality hexahedra can be generated near the boundary, the mesh depends on the coordinate axes orientation, and it is difficult to obtain a conformal transition of the element size. On the other hand, the rest of the hexahedral mesh generation algorithms can generate high-quality meshes for several configurations but are not robust and fully automatic.

Several authors have proposed hexahedral-meshing algorithms based on the dual. The *whisker weaving* algorithm proposed by Tautges et al. (1996) builds the topology, not the geometry, of the dual of a hexahedral mesh using an advancing front method (Folwell and Mitchell, 1999; Ledoux and Weill, 2007). Mueller-Hannemann (1999) proposed a purely topological method to decompose a geometry bounded by a quadrilateral surface mesh into hexahedra. In their work, the decomposition is guided by the topology of the closed dual curves of the surface mesh. Calvo and Idelsohn (2000) presented a decomposition approach where closed dual curves are used to recursively divide the topology of the dual of the surface mesh into two topological balls. All of these dual algorithms use the combinatorial information contained in the topology of the dual but do not construct its geometric representation.

All the above dual approaches to hexahedral mesh generation start from a prescribed quadrilateral mesh that over-constrains the problem. Hence, these methods usually require modifying the prescribed boundary mesh. In order to relax hexahedral meshing problem and avoid these modifications, Staten et al. (2005, 2006) recently presented the *unconstrained plastering* approach. This primal approach consists in generating hexahedral elements in an advancing front manner without the constraint of respecting a bounding quadrilateral mesh. Note that this method is an unconstrained version of the *plastering* method proposed by Blacker and Meyers (1993). The unconstrained paradigm is also used by the *medial axis decomposition* methods (Price et al., 1995; Price and Armstrong, 1997; Sheffer et al., 1999; Sheffer and Bercovier, 2000) based on the work by Tam and Armstrong (1991). These methods use the *medial axis*, the locus of the points that are roughly along the middle of a domain, as a reference to decompose the domain in a coarse hexahedral mesh.

Unconstrained dual surface insertion algorithms, where both the topology and the geometry of the dual are determined, have only been sketched in the literature. In particular, Murdoch et al. (1997) briefly introduce the *twist plane insertion* algorithm, and Calvo (2005) speculates about the *free stroking*[1] method. However these authors did not implement these methods because a methodology to represent, intersect and insert *continuous* dual surfaces is not available.

[1]Named “libre trazado” in the original work in Spanish.

1.4 Goals and outline

Hexahedral mesh generation has been an important and active research area over the last decades. However, a fully automatic hexahedral mesh generation algorithm for any geometry is still not available. Therefore, further research has still to be developed in order to obtain a general-purpose algorithm that generates a high quality hexahedral mesh for any volume.

This dissertation is devoted to develop several techniques that facilitate the generation of high quality hexahedral meshes for a wide range of geometries. We believe that industry will benefit from any improvement in the available hexahedral meshing techniques as well as any novel approach to automatically generate hexahedral meshes. On the one hand, advances in technology such as graphical meshing environments or the sweeping technique have shown being indispensable to semi-automatically generate hexahedral meshes in industrial applications. On the other hand, previous attempts to automatically obtain hexahedral meshes have provided more insight in the understanding of the hexahedral mesh generation problem. Hence, in this dissertation we consider five goals:

- **Proposing a new affine method for sweeping that leads to a set of normal equations with full rank**. Sweep methods are one of the most robust techniques to generate hexahedral meshes in extrusion volumes. The main issue in sweep algorithms is the projection of cap surface meshes along the sweep path. From the computational point of view, the most competitive technique to determine this projection is to find a least-squares approximation of an affine mapping. Several functional formulations have been defined to carry out this least-squares approximation. Different versions of this technique are known as *affine methods*. However, the minimization of these functionals may lead to a rank deficient set of normal equations with singular system matrix. To overcome this drawback, in Chapter 2 we propose a new functional that depends on two vector parameters. Moreover, we present a comparative analysis between the new and the classical functionals. In this analysis we prove the equivalence between the functionals. In addition, we prove under which conditions the minimization of the analyzed functionals leads to a full rank linear system. The proofs of the analysis presented in Chapter 2 are included in Appendix A.

- **Developing an automatic projection algorithm for sweeping that preserves offset data**. In Chapter 3 we present a new and automatic node projection algorithm to generate hexahedral meshes in extrusion geometries. It is designed to preserve the shape of the cap surfaces in the inner layers of a sweeping mesh. The algorithm is based on the new functional presented in Chapter 2. We first report that the functionals that have been traditionally used to compute the affine mapping generate four undesired effects on the inner layers of nodes. To overcome these drawbacks we introduce the concept of the pseudo-area and pseudo-normal vectors defined by a loop of nodes. In addition, we prove several geometrical properties of these two vectors. Based on the properties of the new functional and of the pseudo-normal vector, we detail a new projection algorithm that automatically selects the functional vector parameters. The aim of Chapter 3 is to provide the implementation details to developers, although we also present the theoretical background of the algorithm. Finally, several mesh examples are discussed to assess the properties of the proposed algorithm. The theoretical details of the algorithm implementation are included in Appendix B.

- **Presenting a sweeping scheme based on the proposed affine method**. Chapter 4 presents a new algorithm to map a given mesh over a source surface onto a target surface. This projection is carried out by means of a least-squares approximation of an affine mapping defined between the parametric spaces of the surfaces. Once the new mesh is obtained on the parametric space of the target surface, it is mapped up according to the target surface parameterization. Therefore, the developed algorithm does not require solving any root finding problem to ensure that the projected nodes are on the target surface. Afterwards, the projection algorithm presented in Chapter 3 is included in a sweep-meshing tool. To generate inner layers of nodes both cap surface meshes are directly projected to the current layer. This new scheme allows parallelizing the generation of the inner layers of nodes.

- **Proposing a novel approach for block meshing by representing the geometry and the topology of a hexahedral mesh dual**. Chapter 5 is devoted to obtaining a valid topological decomposition in blocks of a given domain without a previous discretization of the boundary. To this end, we propose an approach to directly construct a valid dual of the block mesh. The proposed algorithm is composed by two main steps: first, we create a *reference mesh*, a tetrahedral mesh, of the domain to decompose into blocks; and second, we create the dual of the block mesh by using the new concept of *local dual contributions*. These contributions are planar surfaces inside the tetrahedra of the reference mesh. The union of these contributions defines a discretized version of the dual surfaces arrangement. To this end, we hierarchically add local dual contributions following a set of matching rules. These rules ensure that the local dual contributions match with the proper multiplicity and do not present gaps between them. Practical results suggest that the proposed types of local dual contributions and the set of matching rules provide enough freedom to describe the dual arrangement of a block mesh.

- **Starting-up and developing a new mesh generation framework**. At the time of starting this dissertation several non-commercial frameworks that integrate a CAD engine and meshing algorithms were available. However, these environments did not fulfill our needs in quadrilateral and hexahedral mesh generation. Hence, in Chapter 6 we present an overview of a new mesh generation framework designed and developed from zero to provide the required technology. Specifically, this framework has been used to develop and implement: the projection method presented in Chapter 3, the sweeping scheme proposed in Chapter 4, and the block meshing approach of Chapter 5. Moreover, the tool incorporates several meshing tools not available in similar non-commercial packages.

Chapter 2

A new affine method for sweeping: fundamentals and comparative analysis

2.1 Introduction

A fully automatic hexahedral mesh generation algorithm for any arbitrary geometry is still not available. To generate hexahedral meshes the domains are decomposed into several simpler-subvolumes, see Section 1.2 of Chapter 1. It is common that a large number of extrusion volumes result from this decomposition. Therefore, special attention has been focused on algorithms to mesh this specific type of geometries. These extrusion or sweep volumes are obtained by extruding a surface along a curved path. That is, they are defined by a *source surface*, a *target surface* and a series of *linking sides*, see Figure 2.1. In order to ensure that a one-to-one geometry is a sweep volume, the following requirements have to be accomplished:

(i) Source and target surfaces must be topologically equivalent (they must have the same number of holes and logical sides). However, they may have different areas and curvatures.

(ii) Linking-sides must be mappable, or equivalently, defined by four logical sides.

(iii) Sweep volume has one source surface and one target surface.

(iv) Sweep volume must be defined by only one axis.

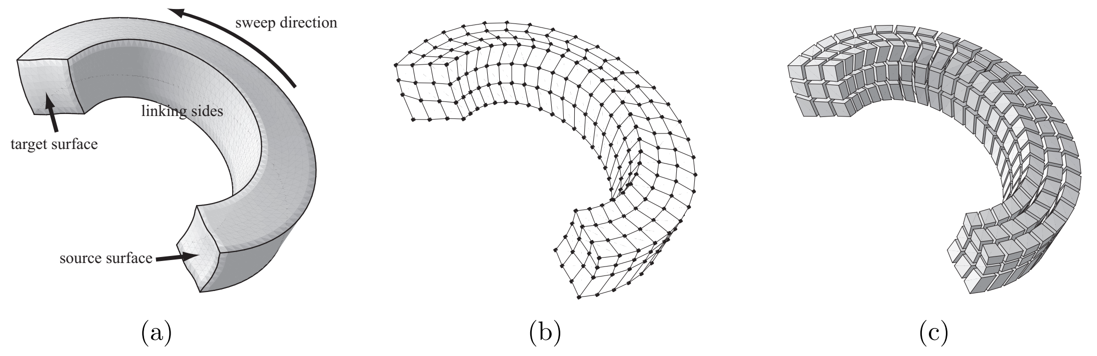


Figure 2.1: Sweeping process of an extrusion volume (a) geometry, (b) available data, i.e. surface meshes, and (c) shrunk hexahedral mesh.

A detailed presentation on constraints which must be met for a volume to be sweepable, in a generic sense, are presented in White and Tautges (2000). Based on the definition of extrusion geometry, the traditional procedure to generate an all–hexahedral mesh by sweeping consists in the following five steps:

(i) To generate a structured or unstructured quadrilateral mesh on the source surface. This mesh determines the topology of the obtained mesh along the sweep direction.

(ii) To map the source mesh into the target surface. Hence, the target surface mesh has the same topology as the source surface mesh.

(iii) To generate a structured quadrilateral mesh over the linking sides. This mesh defines the ribs that join each node on the boundary of the source surface with its corresponding node on the boundary of the target mesh. Moreover, it also defines the boundary of the inner layers of nodes.

(iv) To generate the inner layers of nodes. The topology of the source surface mesh is copied along the sweeping direction. The node location has to capture the transition between the curvatures of both cap surfaces.

(v) To generate the hexahedral elements by connecting the nodes of two consecutive layers.

Several robust quadrilateral surface mesh generation algorithms have been developed which greatly simplify the surface meshing process involved in the first step (Cass et al., 1996; Sarrate and Huerta, 2000b,a). The gridding of the linking sides involved in the third step must be generated using any standard structured quadrilateral surface mesh generator (Thompson et al., 1999). Hence, the two main issues to be dealt by any sweep algorithm are the second and fourth step. In both steps, the meshes to be generated (i.e. the mesh over the target surface and the inner layer meshes) must be topologically equivalent to the source surface mesh.

2.1.1 Affine methods: projecting surface meshes

We have seen above that projecting surface meshes between loops of nodes is the main issue of sweeping algorithms. To generate inner layers of hexahedra an initial mesh is extruded along the sweep path. These inner layers are delimited by several loops of nodes that belong to the structured meshes of the linking sides, see Figure 2.1(b). Specifically, for every layer there is one outer loop, and one inner loop for each hole in the sweep volume. Several algorithms have been developed in order to generate the inner layer of nodes. Mingwu and Benzley (1996) presented a method that geometrically determines the position of inner nodes. Staten et al. (1999) developed the BMSweep algorithm which uses barycentric co-ordinates in a background mesh to locate inner nodes. Blacker (1996) first developed a projection algorithm based on a least-squares weighted residual functional which does not require the creation of a background mesh. Knupp (1998) detailed a projection method in which the inner nodes are located using a least-squares approximation of a linear transformation between the boundary nodes of the source surfaces and the boundary nodes of the inner layers.

Although all the previous algorithms to generate the inner layer of nodes have their strengths and weaknesses, projection algorithms based on a least-squares approximation of an affine mapping are the fastest option, see Scott et al. (2005) for a comparative analysis. These projections are usually referred as *affine methods* and they generate high-quality meshes, especially in translatable, rotatable, scalable, and shearable geometries. That is, between loops of points that are almost affine. If the loops of nodes are not affine, then an additional smoothing step may be needed. However, this smoothing step is also needed in other methods (Mingwu and Benzley, 1996; Goodrich, 1997). Two steps compose the standard procedure of a projection algorithm. First, starting at the meshes of the cap surfaces, the location of a new layer of nodes is computed from the previous one in an *advancing front manner*, see Knupp (1998); Mingwu and Benzley (1996). Second, to increase the robustness of the projection algorithm, the final location of the inner nodes is computed averaging the nodal position obtained starting from the meshes of both cap surfaces.

2.1.2 Contributions and outline

Several functionals have been defined to compute this least-squares approximation, see Section 2.2. However, the minimization of some of these functionals does not always generate an acceptable projected mesh. For several usual geometric configurations the minimization of the proposed functionals may lead to a set of normal equations with a singular system matrix or may introduce several undesired geometrical effects on the final mesh. These previously unknown drawbacks were first reported in Roca et al. (2005b). In this chapter we extend this work. Here we develop an in depth theoretical study on the least-squares fitting of affine transformations defined between two finite sets of points. This analysis provides the necessary theoretical background for the projection algorithms presented in Knupp (1998), and in Chapters 3 and 4. The main contributions presented in this chapter are:

- To propose a new functional formulation that maintains the performance of standard affine methods, see Section 2.2.
- To develop a theoretical and comparative analysis between standard affine methods and the new formulation, see Section 2.3. In addition, we prove under which conditions standard formulations fail.
- To prove that our formulation overcomes the shortcomings of classical formulations. That is, our formulation leads to a set of normal equations with full rank, see Section 2.4.

2.2 Problem statement

2.2.1 The original formulation

Let $X = \{\mathbf{x}^i\}_{i=1,\dots,m} \subset \mathbb{R}^n$ be a set of source points, and $Y = \{\mathbf{y}^i\}_{i=1,\dots,m} \subset \mathbb{R}^n$ be a set of target points with $m \geq n$. From the practical point of view, two cases are important: $n = 2$ used to sweep curves on a plane, and $n = 3$ used to project meshes along a given path in a sweep algorithm. In a 3D sweep application the source layer is a quadrilateral mesh. This mesh is composed by a set of boundary points, X, and a set of inner nodes, see figure 2.2(a). The initial layer is the source surface mesh. Our goal is to map this quadrilateral mesh onto a target layer bounded by the set of points Y. Note that there is not exist an underlying surface defining this target layer. That is, the set of boundary points Y is the only available data for generating the new inner nodes of the quadrilateral mesh of the target layer. Therefore, we are looking for a mapping $\boldsymbol{\phi} : \mathbb{R}^n \to \mathbb{R}^n$ such that $\mathbf{y}^i = \boldsymbol{\phi}(\mathbf{x}^i)$ for $i = 1, \ldots, m$. Once this mapping is obtained, we will use it to map the inner nodes of the source quadrilateral mesh to the target layer.

The fastest method to sweep a mesh is to approximate $\boldsymbol{\phi}$ by an affine mapping $\boldsymbol{\varphi}$ from $\mathbb{R}^n$ to $\mathbb{R}^n$. This affine mapping is determined by a least-squares fitting of the given data. Thus, we want to find $\boldsymbol{\varphi}$ such that it minimizes the functional

$$E(\boldsymbol{\varphi}) = \sum_{i=1}^{m} \left\|\mathbf{y}^i - \boldsymbol{\varphi}(\mathbf{x}^i)\right\|^2. \tag{2.1}$$

From the geometrical point of view, the minimization of (2.1) means that the optimal affine mapping is such that the sum of the squares of the distances between the target points and images of the source points is minimized, see figure 2.2(b). We also define

$$\mathbf{c}^X := \frac{1}{m}\sum_{i=1}^{m} \mathbf{x}^i \quad \text{and} \quad \mathbf{c}^Y := \frac{1}{m}\sum_{i=1}^{m} \mathbf{y}^i, \tag{2.2}$$

as the geometric centers of the sets X and Y, respectively.

Remark 2.1. Any affine mapping $\boldsymbol{\varphi}$ from $\mathbb{R}^n$ to $\mathbb{R}^n$ can be written as $\boldsymbol{\varphi}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}'$, where $\mathbf{A} \in \mathcal{L}(\mathbb{R}^n)$ is a linear transformation, and $\mathbf{b}' \in \mathbb{R}^n$ is the affine part. If we define

$$\mathbf{b} := \mathbf{b}' + \mathbf{A}\mathbf{c}^X, \tag{2.3}$$

then we can write $\boldsymbol{\varphi}$ as

$$\boldsymbol{\varphi}(\mathbf{x}) = \mathbf{A}(\mathbf{x} - \mathbf{c}^X) + \mathbf{b}. \tag{2.4}$$

Taking into account Remark 2.1 we can write, without loss of generality, the initial least-squares problem (2.1) as the minimization of the functional

$$E(\mathbf{A}, \mathbf{b}) := \sum_{i=1}^{m} \left\|\mathbf{y}^i - \mathbf{A}(\mathbf{x}^i - \mathbf{c}^X) - \mathbf{b}\right\|^2, \tag{2.5}$$

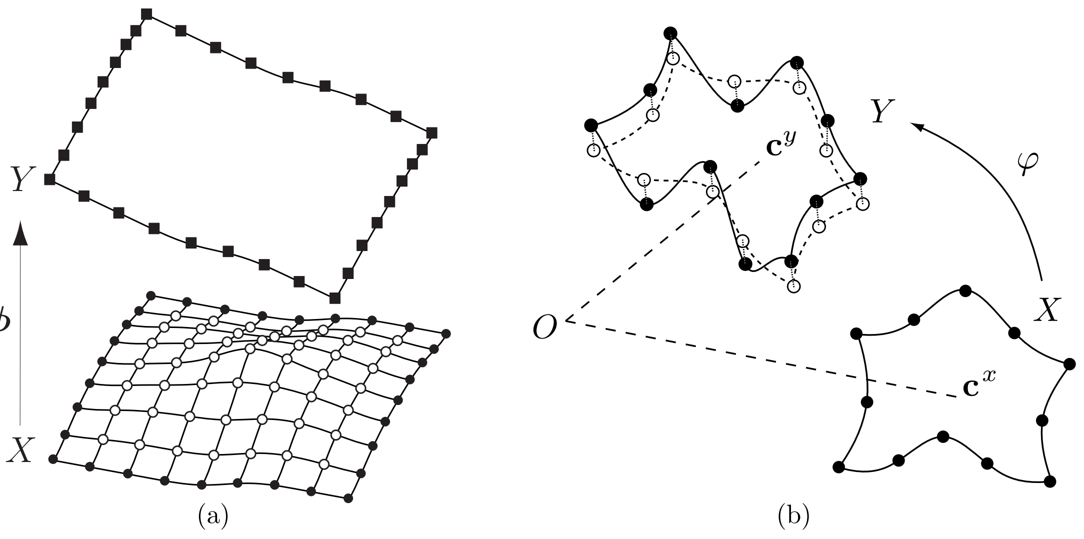


Figure 2.2: (a) The inner nodes of the source quadrilateral mesh (marked with ∘) are mapped according to map $\boldsymbol{\phi}$. Available data is: the set X (marked with •) and the set Y (marked with ■). (b) The image of set X by $\boldsymbol{\varphi}$, marked with ∘, approximates the set Y according to E.

where $\mathbf{A} \in \mathcal{L}(\mathbb{R}^n)$ and $\mathbf{b} \in \mathbb{R}^n$.

Proposition 2.2. *If* $(\mathbf{A}^E, \mathbf{b}^E) \in \mathcal{L}(\mathbb{R}^n) \times \mathbb{R}^n$ *is such that*

$$E(\mathbf{A}^E, \mathbf{b}^E) = \min_{(\mathbf{A},\mathbf{b}) \in \mathcal{L}(\mathbb{R}^n) \times \mathbb{R}^n} E(\mathbf{A}, \mathbf{b}),$$

then $\mathbf{b}^E = \mathbf{c}^Y$.

Proof. The minimization of functional E is equivalent to solving the overdetermined linear system

$$\mathbf{A}(\mathbf{x}^i - \mathbf{c}^X) + \mathbf{b} = \mathbf{y}^i, \qquad i = 1, \cdots, m. \tag{2.6}$$

Note that the unknowns of the previous overdetermined linear system are the coefficients of the $n \times n$ matrix, $\mathbf{A}$, and the coefficients of the n-dimensional vector, $\mathbf{b}$. According to Equation (2.3) we set $\mathbf{b}' = \mathbf{b} - \mathbf{A}\mathbf{c}^X$. Hence, Equation (2.6) is equivalent to $\mathbf{A}\mathbf{x}^i + \mathbf{b}' = \mathbf{y}^i$, for $i = 1, \cdots, m$. By applying the well known normal equations to the previous least-squares problem we obtain

$$\mathbf{A}\sum_{i=1}^{m} \mathbf{x}^i + m\mathbf{b}' = \sum_{i=1}^{m} \mathbf{y}^i.$$

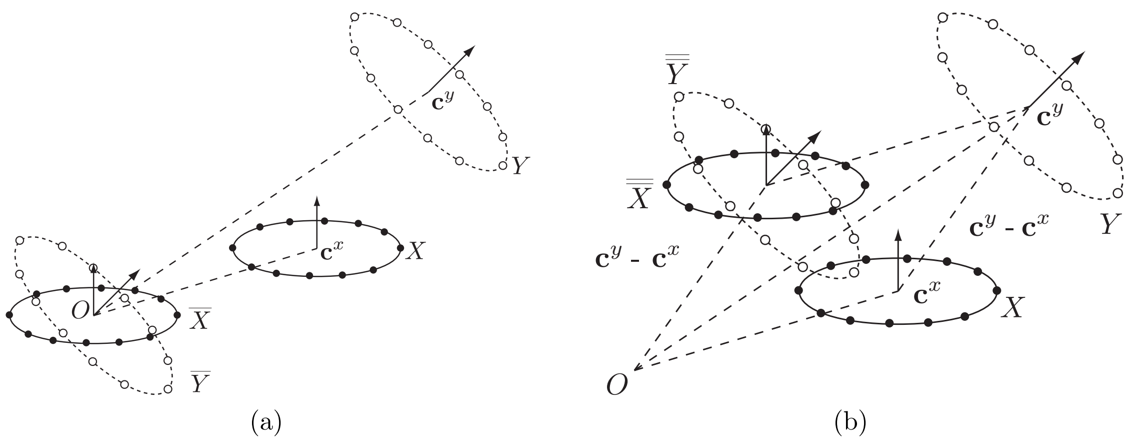

$\mathbf{c}^y$
Y
$\mathbf{c}^x$
X
O
$\overline{X}$
$\overline{Y}$
(a)
$\overline{\overline{Y}}$
$\overline{\overline{X}}$
$\mathbf{c}^y$ - $\mathbf{c}^x$
$\mathbf{c}^y$
$\mathbf{c}^y$ - $\mathbf{c}^x$
Y
$\mathbf{c}^x$
X
O
(b)

Figure 2.3: Geometric representation of the translation of the sets of points X and Y **(a)** to the origin and **(b)** to $\mathbf{c}^Y - \mathbf{c}^X$.

Taking into account the definitions of $\mathbf{c}^X$ and $\mathbf{c}^Y$ we obtain $\mathbf{b}' = \mathbf{c}^Y - \mathbf{A}\mathbf{c}^X$. Finally, according to Equation (2.3) we conclude that $\mathbf{b} = \mathbf{c}^Y$. $\square$

Remark 2.3. Proposition 2.2 has a geometrical meaning. It states that the optimal solution $(\mathbf{A}^E, \mathbf{b}^E)$ maps the center $\mathbf{c}^X$ into the center $\mathbf{c}^Y$. This result follows from Proposition 2.2 and Equation (2.4).

2.2.2 Alternative formulations

According to Remark 2.3, the new coordinates $\overline{\mathbf{x}} = \mathbf{x} - \mathbf{c}^X$ and $\overline{\mathbf{y}} = \mathbf{y} - \mathbf{c}^Y$ are introduced, see Knupp (1998) for details. These new coordinates can be interpreted as translating the sets of point X and Y to the origin, see Figure 2.3(a). Using these new coordinates the following functional is also introduced in Knupp (1998):

$$F(\mathbf{A}) := \sum_{i=1}^{m} \left\| \mathbf{y}^i - \mathbf{c}^Y - \mathbf{A}(\mathbf{x}^i - \mathbf{c}^X) \right\|^2 = \sum_{i=1}^{m} \left\| \overline{\mathbf{y}}^i - \mathbf{A}\overline{\mathbf{x}}^i \right\|^2. \tag{2.7}$$

Therefore, we are looking for a linear mapping $\mathbf{A}$ such that it transforms, in the least-squares sense, $\overline{X} = \{\overline{\mathbf{x}}^i\}_{i=1,\ldots,m}$ into $\overline{Y} = \{\overline{\mathbf{y}}^i\}_{i=1,\ldots,m}$. Functional (2.7) is used in Knupp (1998) in order to simplify the statement of the least-squares approximation of the affine mapping between the source and target points of a sweep volume. However, functional F has an important drawback: if the set of source points determines a hyperplane (for instance, a plane for 3D problems or a straight line for 2D problems),

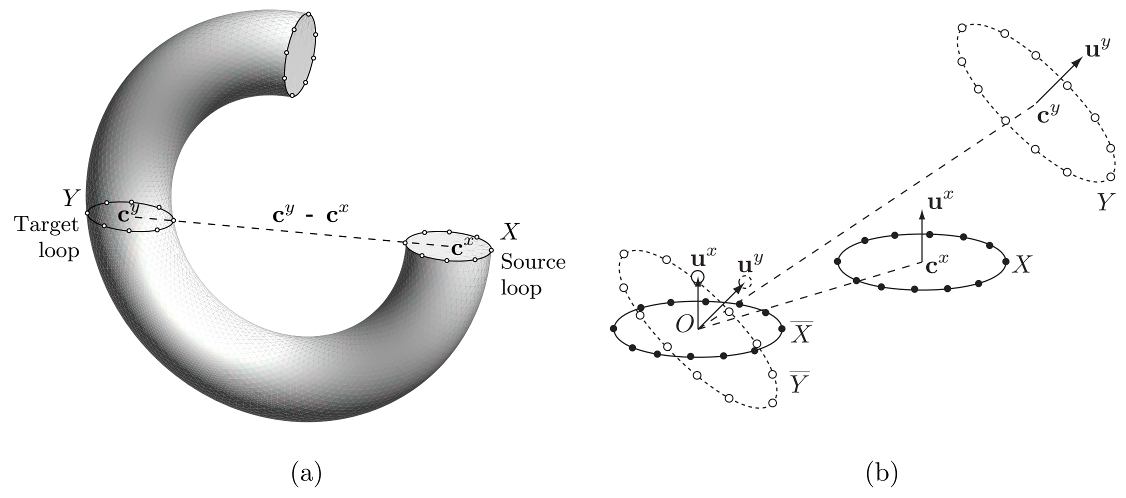


Figure 2.4: **(a)** Example of a geometry where $\mathbf{c}^Y - \mathbf{c}^X$ lies in the same plane than the source surface and the boundary of the inner layer. **(b)** Geometric representation of the translation of the sets of points X and Y to the origin and additional vectors $\mathbf{u}^X$ and $\mathbf{u}^Y$.

then the minimization of (2.7) induces a set of normal equations with a singular system matrix. Note that this geometrical configuration is usual in practical CAD models.

In order to avoid this drawback, the functional

$$G(\mathbf{A}) := \sum_{i=1}^{m} \left\| \mathbf{y}^i - \mathbf{c}^X - \mathbf{A}(\mathbf{x}^i - \mathbf{c}^X + \mathbf{c}^Y - \mathbf{c}^X) \right\|^2 = \sum_{i=1}^{m} \left\| \overline{\overline{\mathbf{y}}}^i - \mathbf{A}\overline{\overline{\mathbf{x}}}^i \right\|^2 \qquad (2.8)$$

is also proposed in Knupp (1998), where the new coordinates $\overline{\overline{\mathbf{x}}} = \mathbf{x} - \mathbf{c}^X + \mathbf{c}^Y - \mathbf{c}^X$ and $\overline{\overline{\mathbf{y}}} = \mathbf{y} - \mathbf{c}^X$ are introduced. These new coordinates have a clear geometric interpretation: the sets of points X and Y are translated to $\mathbf{c}^Y - \mathbf{c}^X$, see Figure 2.3(b). Therefore, by minimizing functional G we are looking for a linear mapping $\mathbf{A}$ such that it approximately transforms, in the least-squares sense, $\overline{\overline{X}} = \{\overline{\overline{\mathbf{x}}}^i\}_{i=1,\ldots,m}$ into $\overline{\overline{Y}} = \{\overline{\overline{\mathbf{y}}}^i\}_{i=1,\ldots,m}$. From the practical point of view, the minimization of functional G is only used for source surfaces with planar boundaries. However, we will prove that functional (2.8) also leads to a set of normal equations with singular matrix if the vector $\mathbf{c}^Y - \mathbf{c}^X$ lies in the hyperplane determined by the source points, see Figure 2.4(a).

2.2.3 A new functional formulation

In order to overcome the shortcomings presented by functionals (2.7) and (2.8), while maintaining the performance of these methods we propose the new functional

$$\begin{aligned} H(\mathbf{A};\mathbf{u}^X,\mathbf{u}^Y) &:= \sum_{i=1}^{m}\left\|\mathbf{y}^i-\mathbf{c}^Y-\mathbf{A}(\mathbf{x}^i-\mathbf{c}^X)\right\|^2+\left\|\mathbf{u}^Y-\mathbf{A}\mathbf{u}^X\right\|^2 \qquad (2.9)\\ &= \sum_{i=1}^{m}\left\|\overline{\mathbf{y}}^i-\mathbf{A}\overline{\mathbf{x}}^i\right\|^2+\left\|\mathbf{u}^Y-\mathbf{A}\mathbf{u}^X\right\|^2, \end{aligned}$$

where $\mathbf{u}^X$ and $\mathbf{u}^Y$ are two vector parameters that belong to $\mathbb{R}^n$. Note that vectors $\mathbf{u}^X$ and $\mathbf{u}^Y$ in (2.9) can be properly selected in order to obtain several desired properties of functional H. Therefore, we are looking for a linear mapping $\mathbf{A}$ such that it approximately transforms, in the least-squares sense, $\overline{X} = \{\overline{\mathbf{x}}^i\}_{i=1,\ldots,m}$ into $\overline{Y} = \{\overline{\mathbf{y}}^i\}_{i=1,\ldots,m}$, and $\mathbf{u}^X$ to $\mathbf{u}^Y$, see Figure 2.4(b).

2.3 Theoretical and comparative analysis

Functionals E (2.5), F (2.7), G (2.8), and H (2.9) have been defined in order to find an optimal affine mapping, in the least-squares sense, between the set of points X and Y. The aim of this section is to analyze their relationships and to prove that functional H is preferable.

2.3.1 Preliminary definitions and results

Linear algebra

In this analysis we only consider sets of points X of dimension $n-1$ and n. That is, we do not consider sets of points X that generate linear varieties of dimension less than $n-1$. For instance, in $\mathbb{R}^3$ we do not consider source surfaces that degenerate to lines or points, because it does not make sense to sweep them in practical applications.

Definition 2.4 (Hyperplanar set)**.** A set of points $X = \{\mathbf{x}^i\}_{i=1,\ldots,m}$ is hyperplanar if there exists only one hyperplane through all the points in X. In other words, if we take any point of X, the differences between the rest of points of X and the selected point determine a subspace of dimension $n-1$.

Definition 2.5 (Unitary normal vector)**.** Let X be a set of points and $\mathbf{0}$ the origin. A unitary normal vector to X is a vector $\mathbf{n}^X \in \mathbb{R}^n$ with $\|\mathbf{n}^X\| = 1$ such that $< \mathbf{n}^X, \mathbf{x}^i - \mathbf{0} >= c$, for $i = 1, \ldots, m$ and $c \in \mathbb{R}$.

Definition 2.6 (Homogeneous hyperplane)**.** Let X be a hyperplanar set of points. The homogeneous hyperplane of X is the subspace of vectors

$$\mathbb{H} = \{\mathbf{v} \in \mathbb{R}^n | < \mathbf{n}^X, \mathbf{v} >= 0\},$$

where $\mathbf{n}^X \in \mathbb{R}^n$ is a unitary normal vector to X.

Remark 2.7. If X is a hyperplanar set of points and $\mathbf{u}^X \notin \mathbb{H}$, then $\mathbb{R}^n = \text{span}(\mathbf{u}^X) \oplus \mathbb{H}$. Thus, for every $\mathbf{v} \in \mathbb{R}^n$ there exist $\lambda \in \mathbb{R}$ and $\mathbf{v}_{\mathbb{H}} \in \mathbb{H}$ such that $\mathbf{v} = \mathbf{v}_{\mathbb{H}} + \lambda \mathbf{u}^X$.

Lemma 2.8. *Let X be a hyperplanar set of points. Assume that $\mathbf{u}^X \notin \mathbb{H}$, $\mathbf{u}^Y \in \mathbb{R}^n$, and $\mathbf{A} \in \mathcal{L}(\mathbb{R}^n)$ are given. Then, the mapping $\mathbf{\Theta}[\mathbf{A}, \mathbf{u}^X, \mathbf{u}^Y] : \mathbb{R}^n \to \mathbb{R}^n$ defined by*

$$\mathbf{\Theta}[\mathbf{A}, \mathbf{u}^X, \mathbf{u}^Y](\mathbf{v}) := \mathbf{A}\mathbf{v}_{\mathbb{H}} + \lambda \mathbf{u}^Y \tag{2.10}$$

is such that:

(i) $\mathbf{\Theta}[\mathbf{A}, \mathbf{u}^X, \mathbf{u}^Y] \in \mathcal{L}(\mathbb{R}^n)$

(ii) $\mathbf{\Theta}[\mathbf{A}, \mathbf{u}^X, \mathbf{u}^Y](\mathbf{v}_{\mathbb{H}}) = \mathbf{A}\mathbf{v}_{\mathbb{H}}, \quad \forall\, \mathbf{v}_{\mathbb{H}} \in \mathbb{H}$

(iii) $\mathbf{\Theta}[\mathbf{A}, \mathbf{u}^X, \mathbf{u}^Y](\mathbf{u}^X) = \mathbf{u}^Y$.

Proof. Mapping $\mathbf{\Theta}[\mathbf{A}, \mathbf{u}^X, \mathbf{u}^Y]$ accomplishes the properties by construction. □

Remark 2.9. From the geometrical point of view Lemma 2.8 states that the linear mapping $\mathbf{\Theta}[\mathbf{A}, \mathbf{u}^X, \mathbf{u}^Y]$ can be used to take into account the offset data of a non-planar surface mesh delimited by a planar boundary, i.e. planar loop of nodes. On one hand it states that the linear mapping $\mathbf{\Theta}[\mathbf{A}, \mathbf{u}^X, \mathbf{u}^Y]$ maps any vector that belongs to the homogeneous hyperplane according to $\mathbf{A}$. On the other hand it maps the first parameter vector $\mathbf{u}^X$ (that does not belong to $\mathbb{H}$) onto the second parameter vector $\mathbf{u}^Y$.

Remark 2.10. Given a matrix $\mathbf{Z}$ it is well known that $\mathbf{Z}\,\mathbf{Z}^T$ has full rank if and only if $\mathbf{Z}$ has full rank, see Gill et al. (1991). On the contrary, we can prove that $\mathbf{Z}\,\mathbf{Z}^T$ is rank deficient if we prove that $\mathbf{Z}$ is rank deficient, too.

2.3.2 Rank analysis

In this section we prove that the minimization of functionals F and G could lead to a rank deficient set of normal equations. On the contrary, the minimization of functionals H can always lead to a full rank set of normal equations.

Proposition 2.11. *Let $X = \{\mathbf{x}^i\}_{i=1,\ldots,m} \subset \mathbb{R}^n$ be a hyperplanar set of points. Then, the minimization of functional F is equivalent to solving n uncoupled overdetermined linear systems of rank $n-1$.*

Proof. The minimization of functional F is equivalent to imposing the following m constraints:

$$\mathbf{A}(\mathbf{x}^i - \mathbf{c}^X) = \mathbf{y}^i - \mathbf{c}^Y, \quad i = 1, \cdots, m. \tag{2.11}$$

Defining

$$\mathbf{A} := \begin{pmatrix} a_{1,1} & \ldots & a_{1,n} \\ \vdots & & \vdots \\ a_{n,1} & \ldots & a_{n,n} \end{pmatrix}, \quad \overline{\mathbf{X}} := \begin{pmatrix} x_1^1 - c_1^X & \ldots & x_1^m - c_1^X \\ \vdots & & \vdots \\ x_n^1 - c_n^X & \ldots & x_n^m - c_n^X \end{pmatrix},$$

and

$$\overline{\mathbf{Y}} := \begin{pmatrix} y_1^1 - c_1^Y & \ldots & y_1^m - c_1^Y \\ \vdots & & \vdots \\ y_n^1 - c_n^Y & \ldots & y_n^m - c_n^Y \end{pmatrix} \tag{2.12}$$

we can write (2.11) as $\mathbf{A}\overline{\mathbf{X}} = \overline{\mathbf{Y}}$. Hence, the minimization of F is equivalent to solving the following n uncoupled overdetermined linear systems

$$\overline{\mathbf{X}}^T \mathbf{a}_k = \overline{\mathbf{y}}_k, \quad k = 1, \cdots, n, \tag{2.13}$$

where $\mathbf{a}_k := (a_{k,j})$ for $j = 1, \cdots, n$ and $\overline{\mathbf{y}}_k = (y_k^l - c_k^Y)$, for $l = 1, \cdots, m$. To conclude we have to prove that $\overline{\mathbf{X}}^T$ has rank $n-1$. By Lemma A.1, and taking into account that $\dim \mathbb{H} = n - 1$, we have that

$$\operatorname{rank} \overline{\mathbf{X}}^T = \dim\big\{\operatorname{span}(\mathbf{x}^1 - \mathbf{c}^X, \ldots, \mathbf{x}^m - \mathbf{c}^X)\big\} = \dim \mathbb{H} = n - 1. \qquad \square$$

Remark 2.12. The set of normal equations corresponding to the minimization of functional F can be obtained from equation (2.13) as

$$\overline{\mathbf{X}}\,\overline{\mathbf{X}}^T \mathbf{a}_k = \overline{\mathbf{X}}\,\overline{\mathbf{y}}_k, \quad k = 1, \cdots, n, \tag{2.14}$$

According to Remark 2.10 the system matrix $\overline{\mathbf{X}}\,\overline{\mathbf{X}}^T$ is singular if the set of points X is hyperplanar, and it is regular if the set of points X spans $\mathbb{R}^n$.

Proposition 2.13. *Let X be a hyperplanar set of points. If $\mathbf{c}^Y - \mathbf{c}^X \in \mathbb{H}$ then the minimization of functional G is equivalent to solving n uncoupled overdetermined linear systems of rank $n-1$. Otherwise, the rank is n.*

Proof. The proof is detailed in Section A.3 of Appendix A. □

Remark 2.14. If X is hyperplanar and $\mathbf{c}^Y - \mathbf{c}^X \in \mathbb{H}$ then the minimization of functional G leads to a set of normal equations with singular system matrix. However, if X is hyperplanar and $\mathbf{c}^Y - \mathbf{c}^X \notin \mathbb{H}$ then the system matrix is regular.

Proposition 2.15. *Let X be a hyperplanar set of points and assume that $\mathbf{u}^Y \in \mathbb{R}^n$. If $\mathbf{u}^X \in \mathbb{H}$ then the minimization of functional H is equivalent to solving n uncoupled overdetermined linear systems of rank $n-1$. Otherwise, the rank is n.*

Proof. The proof is detailed in Section A.3 of Appendix A. □

Remark 2.16. In sweeping applications if X is hyperplanar we can always select $\mathbf{u}^X \notin \mathbb{H}$. Therefore, the minimization of H leads to a set of normal equations with regular system matrix.

2.3.3 Equivalences between functionals

In this section we prove under which conditions the minimization of functionals F, G and H are equivalent to minimizing functional E.

Proposition 2.17. *Let $\mathbf{A}^E \in \mathcal{L}(\mathbb{R}^n)$, $\mathbf{b}^E \in \mathbb{R}^n$ and $\mathbf{A}^F \in \mathcal{L}(\mathbb{R}^n)$ be such that*

$$E(\mathbf{A}^E, \mathbf{b}^E) = \min_{(\mathbf{A},\mathbf{b}) \in \mathcal{L}(\mathbb{R}^n) \times \mathbb{R}^n} E(\mathbf{A}, \mathbf{b})$$

$$F(\mathbf{A}^F) = \min_{\mathbf{A} \in \mathcal{L}(\mathbb{R}^n)} F(\mathbf{A}).$$

Then:

(i) $\min_{(\mathbf{A},\mathbf{b}) \in \mathcal{L}(\mathbb{R}^n) \times \mathbb{R}^n} E(\mathbf{A}, \mathbf{b}) = \min_{\mathbf{A} \in \mathcal{L}(\mathbb{R}^n)} F(\mathbf{A})$

(ii) $E(\mathbf{A}^F, \mathbf{c}^Y) = E(\mathbf{A}^E, \mathbf{c}^Y)$

(iii) $F(\mathbf{A}^E) = F(\mathbf{A}^F)$.

Proof. The proof is detailed in Section A.4 of Appendix A. □

Remark 2.18. The number of degrees of freedom involved in the minimization of F is smaller than in the minimization of E. Hence, it is preferable to minimize F because it simplifies the projection algorithm.

Proposition 2.19. *Let X be a hyperplanar set of points, and assume that $\mathbf{c}^Y - \mathbf{c}^X \notin \mathbb{H}$. If $\mathbf{A}^F \in \mathcal{L}(\mathbb{R}^n)$ and $\mathbf{A}^G \in \mathcal{L}(\mathbb{R}^n)$ are such that*

$$F(\mathbf{A}^F) = \min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} F(\mathbf{A})$$

$$G(\mathbf{A}^G) = \min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} G(\mathbf{A}).$$

Then:

(i) $\min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} G(\mathbf{A})$ *has one and only one solution*

(ii) $\min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} F(\mathbf{A}) = \min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} G(\mathbf{A})$

(iii) $\mathbf{A}^G = \boldsymbol{\Theta}[\mathbf{A}^F, \mathbf{c}^Y - \mathbf{c}^X, \mathbf{c}^Y - \mathbf{c}^X]$

(iv) $F(\mathbf{A}^G) = F(\mathbf{A}^F)$.

Proof. The proof is detailed in Section A.4 of Appendix A. □

Remark 2.20. Property *(iii)* of Proposition 2.19 states that the optimal solution of the minimization of functional G, $\mathbf{A}^G$, can be computed from the optimal solution of the minimization of functional F, $\mathbf{A}^F$, when $\mathbf{c}^Y - \mathbf{c}^X \notin \mathbb{H}$ (note that this property is not used in the original work by Knupp (1998)). In this case we have that $\mathbf{A}^G = \boldsymbol{\Theta}[\mathbf{A}^F, \mathbf{c}^Y - \mathbf{c}^X, \mathbf{c}^Y - \mathbf{c}^X]$. Moreover, by Lemma 2.8 if $\mathbf{c}^Y - \mathbf{c}^X \notin \mathbb{H}$ and taking $\mathbf{u}^X = \mathbf{u}^Y = \mathbf{c}^Y - \mathbf{c}^X$, we have that $\mathbf{c}^Y - \mathbf{c}^X = \mathbf{A}^G(\mathbf{c}^Y - \mathbf{c}^X)$. That is, $\mathbf{c}^Y - \mathbf{c}^X$ is a fixed vector of $\mathbf{A}^G$.

Proposition 2.21. *Let X be a hyperplanar set of points, and assume that $\mathbf{u}^X \notin \mathbb{H}$ and $\mathbf{u}^Y \in \mathbb{R}^n$. If $\mathbf{A}^F \in \mathcal{L}(\mathbb{R}^n)$ and $\mathbf{A}^H \in \mathcal{L}(\mathbb{R}^n)$ are such that*

$$F(\mathbf{A}^F) = \min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} F(\mathbf{A})$$

$$H(\mathbf{A}^H; \mathbf{u}^X, \mathbf{u}^Y) = \min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} H(\mathbf{A}; \mathbf{u}^X, \mathbf{u}^Y).$$

Then:

(i) $\min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} H(\mathbf{A}; \mathbf{u}^X, \mathbf{u}^Y)$ *has one and only one solution*

(ii) $\min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} F(\mathbf{A}) = \min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} H(\mathbf{A};\mathbf{u}^X,\mathbf{u}^Y)$

(iii) $\mathbf{A}^H = \mathbf{\Theta}[\mathbf{A}^F,\mathbf{u}^X,\mathbf{u}^Y]$

(iv) $F(\mathbf{A}^H) = F(\mathbf{A}^F)$.

Proof. The proof is detailed in Section A.4 of Appendix A. □

Remark 2.22. Property *(iii)* of Proposition 2.21 states how to compute the optimal solution of the minimization of functional H, $\mathbf{A}^H$, from the optimal solution of the minimization of functional F, $\mathbf{A}^F$, when X is hyper-planar. In this case, $\mathbf{A}^H = \mathbf{\Theta}[\mathbf{A}^F,\mathbf{u}^X,\mathbf{u}^Y]$. Taking into account the basic properties of $\mathbf{\Theta}$, see Lemma 2.8, we have that $\mathbf{u}^Y = \mathbf{A}^H\mathbf{u}^X$ and $\mathbf{A}^H\mathbf{v}_{\mathbb{H}} = \mathbf{A}^F\mathbf{v}_{\mathbb{H}}$ for $\mathbf{v}_{\mathbb{H}} \in \mathbb{H}$. In other words, the optimal solution of the minimization of H maps $\mathbf{u}^X$ into $\mathbf{u}^Y$. Moreover, it is equal to the optimal solution of the minimization of F over $\mathbb{H}$.

Proposition 2.23. *Let X be a non-hyperplanar set of points and assume that $\mathbf{u}^Y \in \mathbb{R}^n$. If $\mathbf{A}^F \in \mathcal{L}(\mathbb{R}^n)$ and $\mathbf{A}^H \in \mathcal{L}(\mathbb{R}^n)$ are such that*

$$F(\mathbf{A}^F) = \min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} F(\mathbf{A})$$

$$H(\mathbf{A}^H;\mathbf{0},\mathbf{u}^Y) = \min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} H(\mathbf{A};\mathbf{0},\mathbf{u}^Y).$$

Then:

(i) $\min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} F(\mathbf{A}) = \min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} H(\mathbf{A};\mathbf{0},\mathbf{u}^Y) - \|\mathbf{u}^Y\|^2$

(ii) $F(\mathbf{A}^H) = F(\mathbf{A}^F)$.

Proof. The proof is detailed in Section A.4 of Appendix A. □

Remark 2.24. Proposition 2.23 extends the equivalence between the minimization of F and H to non-hyperplanar sets of points. The requirement is that $\mathbf{u}^X = \mathbf{0}$.

2.3.4 Exact mapping characterization

When the least-squares fitting of affine mappings is applied in a sweep algorithm, it is important to characterize under which conditions we can exactly map the set of points X to the set of points Y by means of an affine mapping. The proofs of the following exact mapping characterizations are detailed in Section A.5 of Appendix A.

Proposition 2.25. *There exists an affine mapping $\boldsymbol{\varphi}$ such that $\boldsymbol{\varphi}(\mathbf{x}^i) = \mathbf{y}^i$, for $i = 1, \ldots, m$, if and only if*

$$\min_{\mathbf{A} \in \mathcal{L}(\mathbb{R}^n)} F(\mathbf{A}) = 0.$$

Proposition 2.26. *Assume that X is hyperplanar and $\mathbf{c}^Y - \mathbf{c}^X \notin \mathbb{H}$. Then, there exists an affine mapping $\boldsymbol{\varphi}$ such that $\boldsymbol{\varphi}(\mathbf{x}^i) = \mathbf{y}^i$, for $i = 1, \ldots, m$, if and only if*

$$\min_{\mathbf{A} \in \mathcal{L}(\mathbb{R}^n)} G(\mathbf{A}) = 0.$$

Proposition 2.27. *There exists an affine mapping $\boldsymbol{\varphi}$ such that $\boldsymbol{\varphi}(\mathbf{x}^i) = \mathbf{y}^i$, for $i = 1, \ldots, m$, if and only if there exist $\mathbf{u}^X, \mathbf{u}^Y \in \mathbb{R}^n$ such that*

$$\min_{\mathbf{A} \in \mathcal{L}(\mathbb{R}^n)} H(\mathbf{A}; \mathbf{u}^X, \mathbf{u}^Y) = 0.$$

2.4 Minimizing functional H is preferable

In this section we summarize the shortcomings of functionals F and G. In addition, we highlight that it is always possible to select a vector $\mathbf{u}^X$ such that the minimization of H overcomes these drawbacks.

Flattening. We have seen that the minimization of functional F has two main shortcomings:

- If the set of source points, X, is hyperplanar (for instance a source surface mesh with planar boundary), then the minimization of F leads to a set of normal equations with singular system matrix, see Proposition 2.11. In practice singular value decomposition, SVD, may be used to solve the set of normal equations. In this case, the inner part of the projected mesh will be planar. Hence, a flattening effect will be introduced and the shape of the inner part of the source surface mesh will be lost, see Figures 2.5(a) and 2.5(b).
- If a given mesh is projected over an inner layer with a hyperplanar boundary by minimizing F, then the projected mesh will always have a planar inner part, see Figure 2.5(c). Thus, a flattening effect is also introduced.

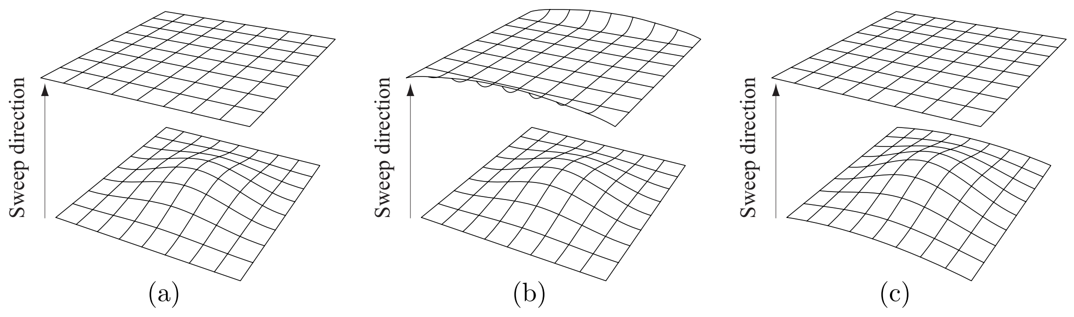


Figure 2.5: Illustration of the flattening effect: (a) projecting a non-planar mesh with planar boundary into a planar boundary loop; (b) projecting a non-planar mesh with planar boundary into a non-planar boundary loop; and (c) projecting a non-planar mesh into a planar boundary loop.

Functional G was introduced to overcome the first drawback of F. In particular, Proposition 2.19 states the equivalence between both functionals. However, if X is hyperplanar and $\mathbf{c}^Y - \mathbf{c}^X \in \mathbb{H}$ then the minimization of G also leads to a set of normal equations with a singular system matrix, see Proposition 2.13. Hence, the minimization of G also introduces the flattening effect.

Note that, on the one hand Proposition 2.21 holds in the case that the source points are hyperplanar. On the other hand, Proposition 2.21 holds even in the case of $\mathbf{c}^Y - \mathbf{c}^X \in \mathbb{H}$. Recall that in these cases the minimization of F and G lead to a set of normal equations with a singular system matrix. Thus, the minimization of H is always preferable in these two cases because it is always possible to choose a vector $\mathbf{u}^X \notin \mathbb{H}$ such that a set of normal equations with a regular system matrix is obtained.

Skewing. Functional G has an additional drawback when the projection algorithm is applied to planar sets of points, even in the case of $\mathbf{c}^Y - \mathbf{c}^X \notin \mathbb{H}$. Consider a non-planar source surface with planar boundary, see Figure 2.6(a). Assume that we want to project this source surface mesh into an inner layer (of a sweep volume) defined by a planar boundary, but non-parallel to the source surface. Figure 2.6(b) shows a cross-section of the source surface (thick solid line), the cross-section of the obtained solution minimizing functionals G (dotted line) and the cross-section of the obtained solution minimizing functionals H (thin solid line). Since $\mathbf{c}^Y - \mathbf{c}^X$ is a fixed vector of $\mathbf{A}^G$, see Remark 2.20, the cross-section obtained with $\mathbf{A}^G$ (dotted line in the top cross section of Figure 2.6(b)) does not preserve the shape of the original surface. On the contrary, Remark 2.22 states the optimal solution of the minimization of H,

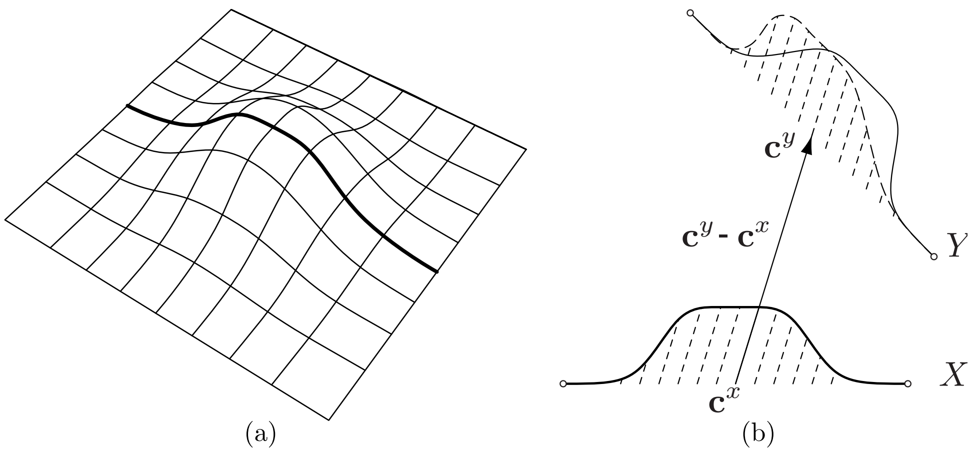


Figure 2.6: (a) A non-planar surface with planar boundary; (b) cross-section view of the surface and its image minimizing G (dotted line) and H (thin solid line).

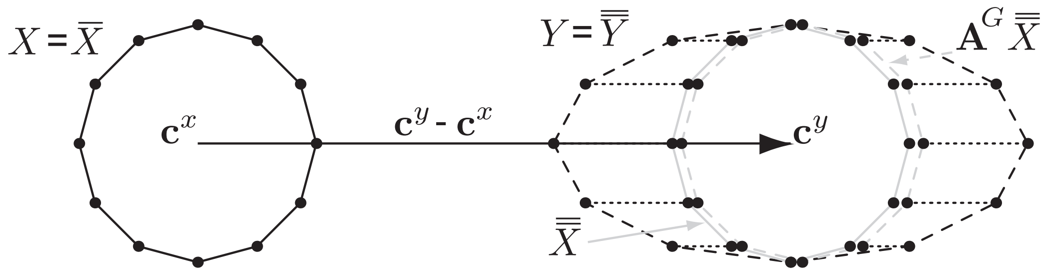

$X = \overline{X}$
$\mathbf{c}^x$
$\mathbf{c}^y$ - $\mathbf{c}^x$
$Y = \overline{\overline{Y}}$
$\mathbf{c}^y$
$\mathbf{A}^G \overline{\overline{X}}$
$\overline{\overline{X}}$

Figure 2.7: Least-squares approximation of an affine mapping between a circular shaped set of points X and an elliptical shaped set of points Y.

$\mathbf{A}^H$, maps $\mathbf{u}^X$ into $\mathbf{u}^Y$. If we select $\mathbf{u}^X = \mathbf{n}^X$ and $\mathbf{u}^Y = \mathbf{n}^Y$, then the normal of the source boundary is mapped into the normal of the target boundary. Therefore, its shape is preserved (thin solid line in the top cross section of Figure 2.6(b)). This example illustrates that the minimization of functional H is preferable since its optimal solution is not affected by the *skewing* introduced by the minimization of functional G and tends to preserve the shape of the original surface. This property was first reported in Roca et al. (2005b).

Not capturing affinities. If X is hyperplanar and $\mathbf{c}^Y - \mathbf{c}^X \in \mathbb{H}$ then Proposition 2.26 does not hold. In this case, if the set X and Y are affine, then the minimization of functional G could provide an affine mapping that does not exactly map X to Y. To illustrate this we consider two affine and coplanar sets of points $X = \{\mathbf{x}^i := (\cos t_i, \sin t_i, 0)\}_{i=0,\ldots,11} \subset \mathbb{R}^3$ and $Y = \{\mathbf{y}^i := (5 + 2\cos t_i, \sin t_i, 0)\}_{i=0,\ldots,11} \subset \mathbb{R}^3$, where $t_i = i\,2\pi/12$ for $i = 0, \ldots, 11$, see Figure 2.7. In this case $\mathbf{c}^X = (0,0,0)$ and $\mathbf{c}^Y = \mathbf{c}^Y - \mathbf{c}^X = (5,0,0)$. Note that in this example $\mathbf{c}^Y - \mathbf{c}^X \in \mathbb{H}$. We solve the corresponding sets of normal equations by means of the SVD which supplies the solution with the smallest norm, and we obtain the following linear transformations:

$$\mathbf{A}^F = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \qquad \mathbf{A}^G = \begin{pmatrix} 52/51 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \text{and} \quad \mathbf{A}^H = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

These linear transformations verify that $F(\mathbf{A}^F) = 0$, $G(\mathbf{A}^G) = 100/17$ and $H(\mathbf{A}^H) = 0$. Thus, $\mathbf{A}^F$ and $\mathbf{A}^H$ exactly map the circular shaped set $\overline{X}$ (black solid line in Figure 2.7) into an elliptical shaped set $\overline{Y}$ (black dotted line in Figure 2.7). On the contrary, $\mathbf{A}^G$ maps the circular shaped set $\overline{\overline{X}}$ (grey solid line in Figure 2.7) into an almost circular set of points $\mathbf{A}^G\overline{\overline{X}}$ (grey dotted line in Figure 2.7), whereas the set of target points $\overline{\overline{Y}}$ is elliptical. Note that in this example both solutions, $\mathbf{A}^F$ and $\mathbf{A}^G$, map the normal component of $\mathbb{H}$ into zero. Thus, all the offset information that the inner part of the source loop of points may contain will be lost (volumetric distributions of points are mapped into planar distributions). Hence, the minimization of H is preferable.

2.5 Concluding remarks

In this chapter we present a new theoretical and comparative analysis of several functionals that are extensively used in sweeping tools. We prove that minimizing F leads to a set of normal equations with a singular system matrix if the set of source points X is hyperplanar. To avoid this drawback we prove the equivalence between minimizing F and G. However, we prove that minimizing G also leads to a set of normal equations with a singular system matrix if, in addition, $\mathbf{c}^Y - \mathbf{c}^X$ lies in the same hyperplane that X. To overcome these drawbacks we introduce functional H and we prove the equivalence between minimizing F and H. Note that this result does not depend on where the vector $\mathbf{c}^Y - \mathbf{c}^X$ lays. Moreover, we also prove that we can always enforce that minimizing H leads to a set of normal equations with a regular system matrix.

Minimizing H has two additional advantages over minimizing F and G. On the one hand, if the source surface is non-planar but has a planar boundary, then the numerical solution obtained from the minimization of F generates planar inner layers of nodes. However, minimizing the new functional H we obtain non-planar inner layers of nodes. On the other hand, if the source surface is non-planar but has a planar boundary and $\mathbf{c}^Y - \mathbf{c}^X \notin \mathbb{H}$, then the minimization of H avoids the skewing effect introduced by G. Therefore, the minimization of H tends to preserve the shapes of cap surfaces on inner layers.

The source and the target loops of nodes are not affine in a wide range of applications. Therefore, it is not possible to obtain an affine transformation that exactly maps the source to the target. In these situations an additional smoothing step may be required in order to improve the quality of the final mesh. We claim that the minimization of functional H provides better initial node location than the minimization of functionals F and G. Moreover, this projection algorithm may provide an excellent initial guess for morphing procedures.

Chapter 3

A new affine method for sweeping: preserving offset data and implementation

3.1 Introduction

In Chapter 2 we have presented a theoretical and comparative analysis of several affine methods that have been extensively used to compute the inner layers of nodes in a sweep procedure. In spite of the computational efficiency of standard affine methods (both in terms of cpu time and memory), they present several drawbacks. For instance, the application of these methods may lead to a set of normal equations with a singular system matrix for very common geometrical configurations. In addition, the obtained mesh may present undesired effects such as the *flattening* and the *skewing* of the shape of the inner layers.

In order to overcome these shortcomings, in Chapter 2 we have introduced a new functional, H, that depends on two vector parameters, $\mathbf{u}^X$ and $\mathbf{u}^Y$, that can be selected by the user. However, only a feasible selection of these parameters, without a theoretical analysis, was provided. Moreover, we have proved the relationship between the optimal solution of the standard functionals, E, F and G, and the optimal solution of the functional H. Based on this relationship and on the definition of the pseudo-normal vector, we develop the main contributions of this chapter:

- **To report two new drawbacks of standard affine methods**. Two additional undesired effects haven been observed in hexahedral meshes that are obtained by minimizing standard functionals. We denote these effects as *flip-*

ping and *offset scaling*.

- **To define a measure of the direction of offset data**. At this point, we have reported four undesired effects of standard affine methods: flattening, skewing, flipping and offset scaling. We claim that these four effects are due to the inability of standard functionals to preserve the length, direction and orientation of offset data. In order to overcome these drawbacks, we propose a definition of a measure of the area enclosed by a given loop of nodes and a measure of the vector normal to this loop. We denote these vectors as *pseudo-area* and *pseudo-normal*, respectively. Note that, in sweeping applications, the loops of nodes that define the inner layers are not supported by an underlying surface. Hence, the area or the normal of a loop of nodes cannot be defined in the classical geometrical form. In addition, we also prove several geometrical properties of these vectors.

- **To implement a new projection algorithm for sweeping**. We propose an algorithm that automatically selects the vector parameters, $\mathbf{u}^X$ and $\mathbf{u}^Y$, of the new functional formulation. These parameters are selected in order to preserve the offset data. Moreover, to increase the computational efficiency of the proposed algorithm, the minimization of the new functional adequately reuses the optimal solution of the classical functional and the related singular value decomposition.

The rest of the chapter is organized as follows. In Section 3.2 we report the undesired effects that can be observed on hexahedral meshes that are obtained by means of a least-squares approximation of an affine mapping using functionals F and G. In order to preserve offset data in Section 3.3 we introduce the pseudo-area and pseudo-normal vectors and analyze their properties. Section 3.4 deals with the theoretical background of the projection algorithm and its implementation details.

3.2 Drawbacks of standard affine methods

Meshes generated by minimizing functional F, see Equation (2.7), may present three main problems that are illustrated in Figure 3.1:

- *Flattening.* Given a non-planar mesh the projection algorithm generates a planar mesh in the following two cases:
  - If a non-planar mesh with planar boundary, see Figure 3.1(a), is projected to another loop of nodes (planar or not), then the minimization of functional F leads to a set of normal equations with singular system matrix, see Chapter 2. In practice, the singular value decomposition is used to solve the set of normal equations. In this case the inner part of the projected mesh will be planar. Hence, the shape of the source surface mesh will be lost.
  - If a non-planar mesh is projected to an inner layer with a planar boundary by minimizing F, see Figure 3.1(b), then the projected mesh will always be planar, see Chapter 2.
- *Flipping.* If a loop of nodes is curved towards one direction of the sweep path and it is projected to another loop curved in the opposite direction, see Figure 3.1(c), then the solution of the minimization of F projects offset data inversely to the expected orientation. This may lead to tangled meshes or to distorted hexahedral elements.
- *Offset scaling.* Figure 3.1(d) shows a curved loop that is projected to another loop that is less curved in the same direction. In this case offset data is proportionally scaled. This increment in the scale of offset data may also lead to distorted hexahedral meshes.

Functional G, see Equation (2.8), also presents two important shortcomings:

- If the set of source points, X, is hyperplanar and $\mathbf{c}^Y - \mathbf{c}^X \in \mathbb{H}$, then the minimization of functional G leads to a set of normal equations with singular system matrix, see Chapter 2.
- *Skewing.* If a non-planar surface mesh with planar boundary is projected to an inner layer which is non-parallel to the boundary of the source surface, see Figure 3.2, then the projected nodes do not preserve the shape of the original surface mesh and a skewing effect is introduced, see Section 2.4 of Chapter 2 for details.

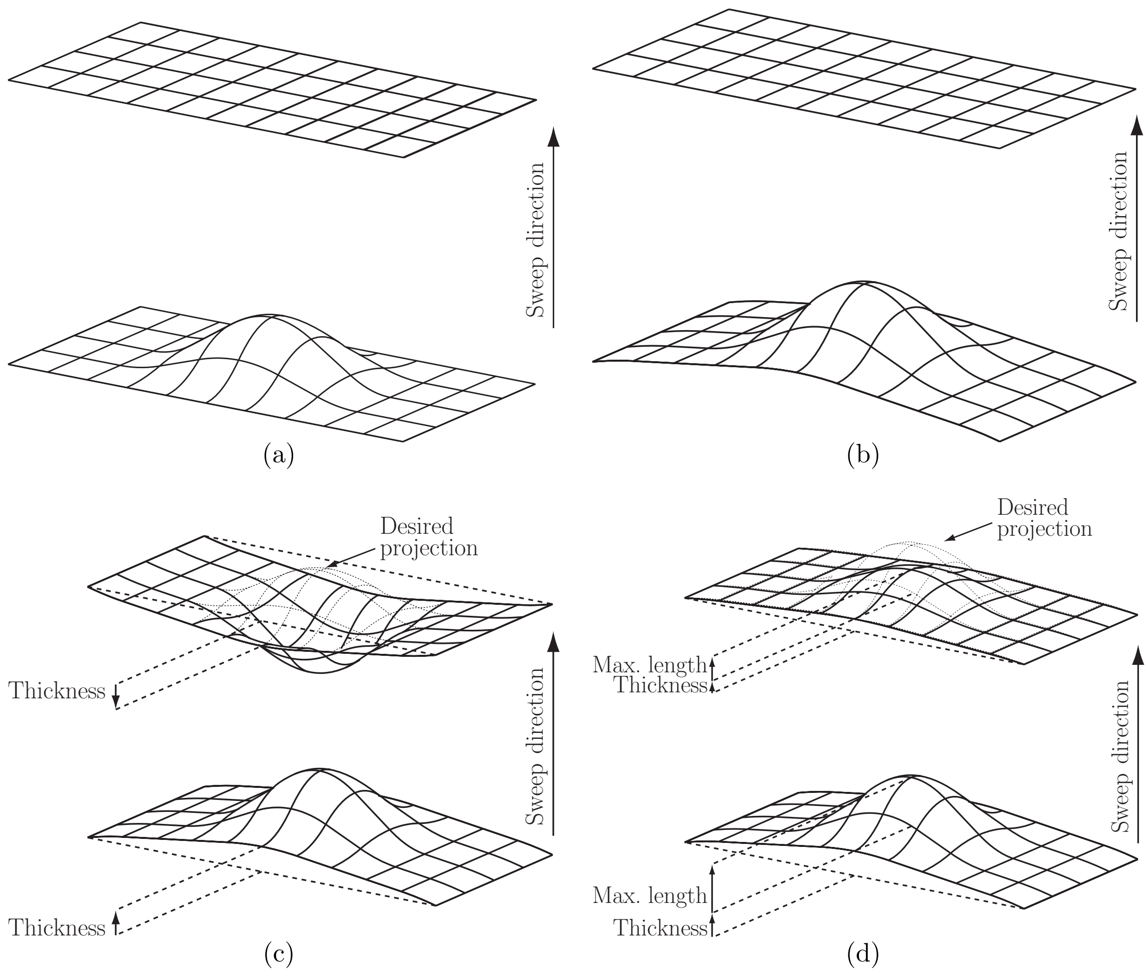


Figure 3.1: Graphical representation of the undesired effects. (a) A non-planar mesh with planar boundary is projected to a loop. A planar mesh is obtained; (b) a non-planar mesh is projected to a planar loop. A planar mesh is also obtained; (c) a non-planar loop is projected to another non-planar loop curved on the opposite direction. Offset data direction is inverted; and (d) a non-planar loop is projected to another non-planar loop. Offset data length is scaled.

There are two main strategies to project meshes using the previous functionals. The first one always projects from the cap surfaces mesh to inner layers. The second one projects, starting from the cap surfaces, from one layer to the next one in an *advancing front manner*. Note that the previous flattening, flipping, offset scaling and skewing effects do not depend on the strategy used to project meshes. However, for particular geometrical configurations some of them can be magnified if the first

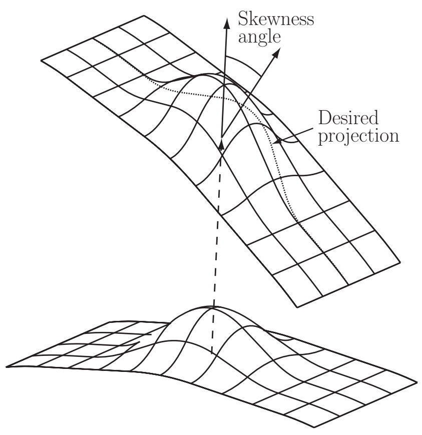


Figure 3.2: A non-planar mesh is projected to a non-parallel loop A skewed mesh is obtained. The desired profile is depicted with a dotted curve.

option is used.

In order to overcome the drawbacks arising from the minimization of functionals F and G, in Chapter 2 we have introduced the functional H (2.9) that depends on two vectors $\mathbf{u}^X$ and $\mathbf{u}^Y$.

In Proposition 2.21, see Chapter 2, we prove that if the set of source points is planar it is always possible to select a vector $\mathbf{u}^X$ such that the minimization of H leads to a set of normal equations with a full rank matrix. However, given any arbitrary geometry, no algorithm has already been established to properly define vectors $\mathbf{u}^X$ and $\mathbf{u}^Y$ in order to preserve offset data of projected meshes. One of the goals of this chapter is to determine how to select vectors $\mathbf{u}^X$ and $\mathbf{u}^Y$ in order to define an automatic and robust algorithm to sweep meshes in a one-to-one volume. Hence, these two vector parameters can be properly selected in such a way that the minimization of functional H generates inner layer meshes with the desired geometrical properties.

3.3 Preserving offset data

3.3.1 Pseudo-normal of a loop of nodes

This section is devoted to the definition of a *measure* of the vector normal to a given loop of nodes. Recall that in projection algorithms the inner layers are described by a loop of nodes. That is, there is not an underlying surface carrying any geometrical information. Moreover, in a wide range of applications the loops of nodes are not planar. Therefore the normal vector to this kind of loops is not defined. However, given a loop of nodes we will define the pseudo-normal vector and we will relate it to the preservation of the shape of the inner part of the projected mesh, the offset data.

Definition 3.1 (Loop)**.** Given a set of points $X = \{\mathbf{x}^i\}_{i=1,\dots,m} \subset \mathbb{R}^3$, a loop is the closed poly-line constructed by joining $\mathbf{x}^i$ with $\mathbf{x}^{i+1}$ for $i = 1, \dots, m$. We consider that $\mathbf{x}^{m+1} \equiv \mathbf{x}^1$.

In several applications it is necessary to sweep a non-simple connected surface along the extrusion path. These surfaces are defined by one outer boundary and as many inner boundaries as holes they have. Therefore, we need to consider sets of points composed by several loops. Specifically, one counter-clockwise oriented loop corresponding to the outer boundary, and several clockwise oriented loops corresponding to the inner holes.

Definition 3.2 (Multi-loop)**.** A set of points $X = \{\mathbf{x}^i\}_{i=1,\dots,m} \subset \mathbb{R}^3$ is a multi-loop if it is organized in p loops $X_1, \dots, X_p$.

Definition 3.3 (Pseudo-area)**.** Given a point $\mathbf{c} \in \mathbb{R}^3$, the pseudo-area of a loop $X = \{\mathbf{x}^i\}_{i=1,\dots,m} \subset \mathbb{R}^3$ is

$$\mathbf{a}^X_{pseudo} := \frac{1}{2}\sum_{i=1}^{m}(\mathbf{x}^i - \mathbf{c}) \times (\mathbf{x}^{i+1} - \mathbf{c}).$$

The pseudo-area of a multi-loop $X = X_1 \cup \dots \cup X_p$ organized in p loops is

$$\mathbf{a}^X_{pseudo} := \mathbf{a}^X_{pseudo,1} + \dots + \mathbf{a}^X_{pseudo,p},$$

where $\mathbf{a}^X_{pseudo,1}, \dots, \mathbf{a}^X_{pseudo,p}$ are the pseudo-areas of loops $X_1, \dots X_p$, respectively.

Note that $\|(\mathbf{x}^i - \mathbf{c}) \times (\mathbf{x}^{i+1} - \mathbf{c})\|$ is twice the area of the triangle $\widehat{\mathbf{x}^i\mathbf{x}^{i+1}\mathbf{c}}$, see Figure 3.3. Moreover, if X is a planar multi-loop, then the pseudo-area, $\mathbf{a}^X_{pseudo}$, is equal to the area enclosed by X.

In order to prove that pseudo-area is well defined, the next proposition states that the pseudo-area vector does not depend on the selected $\mathbf{c} \in \mathbb{R}^3$. Moreover, the pseudo-area is invariant under translations, and its norm is also invariant under orthogonal transformations.

Proposition 3.4 (Invariance of pseudo-area)**.** *Let $X = \{\mathbf{x}^i\}_{i=1,\dots,m} \subset \mathbb{R}^3$ be a set of points. The pseudo-area vector verifies:*

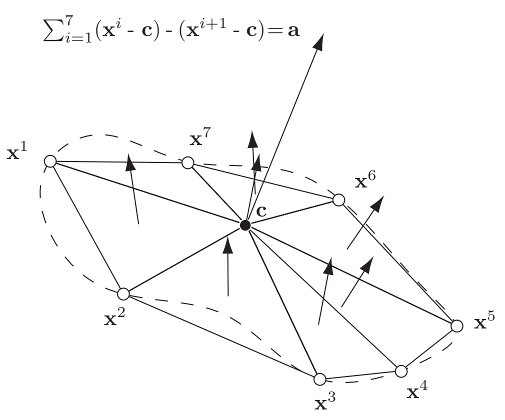


Figure 3.3: Geometrical interpretation of the pseudo-area vector.

(i) Given $\mathbf{c} \in \mathbb{R}^3$ *then*

$$\mathbf{a}^X_{pseudo} = \frac{1}{2}\sum_{i=1}^{m}(\mathbf{x}^i - \mathbf{c}) \times (\mathbf{x}^{i+1} - \mathbf{c}) = \frac{1}{2}\sum_{i=1}^{m}\mathbf{x}^i \times \mathbf{x}^{i+1}.$$

(ii) Given $\mathbf{t} \in \mathbb{R}^3$ *the pseudo-area of* X *is equal to the pseudo-area of* $X + \mathbf{t} = \{\mathbf{x}^i + \mathbf{t}\}_{i=1,\dots,m}$.

(iii) Given an orthogonal transformation $\mathbf{N}$*, then the pseudo-area of* $\mathbf{N}X = \{\mathbf{N}\mathbf{x}^i\}_{i=1,\dots,m}$ *is* $\mathbf{N}\mathbf{a}^X_{pseudo}$.

Proof. The proof is detailed in Section B.1 of Appendix B. □

Thus, given a loop X, we have proved that the norm of the pseudo-area vector is a geometrical invariant associated to the loop. Furthermore, it only depends on the ordering and the relative geometrical location of the points.

Proposition 3.5 (Projected area)**.** *If a multi-loop* X *is projected on an orthogonal plane to its pseudo-area vector,* $\mathbf{a}^X_{pseudo}$*, then the obtained polygon has area equal to* $||\mathbf{a}^X_{pseudo}||$.

Proof. The proof is detailed in Section B.1 of Appendix B. □

Hence, the view of the loop from the direction of the pseudo-area has an area equal to the norm of the pseudo-area of X. Therefore, we can interpret the direction of the pseudo-area as a vector *normal* to the loop. Moreover, according to Proposition 3.7 the pseudo-area vector is the direction that provides the view of the loop with maximum area.

Definition 3.6 (Pseudo-normal)**.** The pseudo-normal of a multi-loop X is the unitary vector

$$\mathbf{n}^X_{pseudo} := \mathbf{a}^X_{pseudo}/||\mathbf{a}^X_{pseudo}||,$$

where $\mathbf{a}^X_{pseudo}$ is the pseudo-area of X.

Proposition 3.7 (Maximum projected area)**.** *Let* $\mathbf{v} \in \mathbb{R}^3$ *be a unitary vector. Consider the signed area enclosed by the projection of a multi-loop* X *to the orthogonal plane to* $\mathbf{v}$*. Then, this area is maximized when* $\mathbf{v} = \mathbf{n}^X_{pseudo}$*. Moreover, the value of the enclosed area is* $||\mathbf{a}^X_{pseudo}||$*.*

Proof. The proof is detailed in Section B.1 of Appendix B. □

3.3.2 Offset data

Note that if X is a planar loop, the pseudo-normal $\mathbf{n}^X_{pseudo}$ is equal to the unitary normal $\mathbf{n}^X$ to X. The pseudo-normal provides a measure of the *normal* direction when there is not an underlying surface, and only the loop of points is available. All the information along the direction of the pseudo-normal is understood as *offset data.* We claim that in order to avoid flattening, flipping, offset scaling and skewing effects we have to obtain affine mappings that preserve the length, direction and orientation of offset data.

3.4 Algorithm implementation

In this section we detail the algorithm that we have developed in order to *(i)* properly select the parameters $\mathbf{u}^X$ and $\mathbf{u}^Y$, and *(ii)* obtain the affine projection. The basic idea is that we can efficiently use the minimization of functional F to minimize H. The key issue is to realize that $\mathbf{A}^H = \mathbf{\Theta}[\mathbf{A}^F, \mathbf{u}^X, \mathbf{u}^Y]$ when X is hyperplanar, see Proposition 2.21 of Chapter 2. Therefore, the optimal solution of functional H can be computed from one of the optimal solutions of functional F if a proper criterion to select the vectors $\mathbf{u}^X$ and $\mathbf{u}^Y$ is defined.

The general algorithm, for hyperplanar and non-hyperplanar set of points X, consists of four steps. First, the optimal solution $\mathbf{A}^F$ is computed. Second, we find the singular value decomposition of $\mathbf{A}^F$. Third, we define a criterion to select the vectors $\mathbf{u}^X$ and $\mathbf{u}^Y$ taking into account the singular value decomposition of $\mathbf{A}^F$. In

Algorithm 3.1: Obtain the affine projection.

```
Input: Coordinate matrix for source loop, $\mathbf{X}$
Input: Coordinate matrix for target loop, $\mathbf{Y}$
Output: Affine mapping $\varphi$
  1. Compute the optimal solution of minimizing $H$: $\mathbf{A}^F := \overline{\mathbf{Y}}\,\overline{\mathbf{U}}\,\overline{\mathbf{W}}^+\,\overline{\mathbf{V}}^T$.
  2. Compute the SVD of the optimal solution: $\mathbf{A}^F = \mathbf{U}\mathbf{W}\mathbf{V}^T$.
  3. Set the values of $\mathbf{u}^X$ and $\mathbf{u}^Y$:
     3.a If $w_i > 0$, for $i = 1, \ldots, n$.
         Set $\mathbf{u}^X = \mathbf{n}^X_{pseudo}$ and $\mathbf{u}^Y = \mathbf{n}^Y_{pseudo}$.
     3.b If $w_i > 0$, for $i = 1, \ldots, n-1$, and $w_n = 0$.
         Set $\mathbf{u}^X = \mathbf{v}_n$ and $\mathbf{u}^Y = \mathbf{u_n}$.
     3.c If $w_i \geq 0$, for $i = 1, \ldots, n-2$ and $w_{n-1} = w_n = 0$.
         Degenerate case. Stop the algorithm.
  4. For any $\overline{\mathbf{x}} \in \mathbb{R}^n$ compute the linear part of the affine projection as
         $\mathbf{A}(\overline{\mathbf{x}}) = \mathbf{A}^F(\overline{\mathbf{x}} - <\overline{\mathbf{x}}, \mathbf{u}^X> \mathbf{u}^X) + <\overline{\mathbf{x}}, \mathbf{u}^X> \mathbf{u}^Y.$
     Compute the desired affine mapping according to Equation (2.4)
         $\varphi(\mathbf{x}) := \mathbf{A}(\mathbf{x} - \mathbf{c}^X) + \mathbf{c}^Y.$
```

addition, if the set of points X and/or Y are hyperplanar (a planar source and/or target surfaces in 3D applications) a geometrical interpretation of the chosen vectors $\mathbf{u}^X$ and $\mathbf{u}^Y$ is also presented. Finally, in the fourth step we compute the affine mapping by first computing its linear part. Algorithm 3.1 summarizes the proposed implementation.

3.4.1 First step: computation of the optimal solution $\mathbf{A}^F$

In order to minimize functional F we compute the minimum norm solution of Equation (2.14). To this end, we use the singular value decomposition of the system matrix

$$\overline{\mathbf{X}}^T = \overline{\mathbf{U}}\,\overline{\mathbf{W}}\,\overline{\mathbf{V}}^T, \tag{3.1}$$

where $\overline{\mathbf{U}}$ is an $m \times n$ matrix with orthogonal columns, $\overline{\mathbf{W}}$ is a $n \times n$ diagonal matrix with positive or zero entries (the singular values)

$$\overline{\mathbf{W}} := \begin{pmatrix} \overline{w}_1 & & \\ & \ddots & \\ & & \overline{w}_n \end{pmatrix},$$

such that $\overline{w}_1 \geq \overline{w}_2 \geq \cdots \geq \overline{w}_{n-1} \geq \overline{w}_n \geq 0$, and $\overline{\mathbf{V}}$ is an $n \times n$ orthogonal matrix. We denote by $\overline{\mathbf{v}}_i \in \mathbb{R}^n$, for $i = 1, \ldots, n$, the columns of matrix $\overline{\mathbf{V}}$.

We minimize functional F by computing the minimum norm solution of Equation (2.14). To this end, we use the singular value decomposition of matrix $\overline{\mathbf{X}}^T$ presented in (3.1). Specifically, we compute $\mathbf{A}^F$ as

$$\mathbf{A}^F = \overline{\mathbf{Y}}\,\overline{\mathbf{U}}\,\overline{\mathbf{W}}^{+}\overline{\mathbf{V}}^T, \tag{3.2}$$

where $\overline{\mathbf{Y}}$ is the matrix of the target coordinates centered at point $\mathbf{c}^Y$, see Equation (2.12) of Chapter 2, and

$$\overline{\mathbf{W}}^{+} := \begin{pmatrix} \overline{w}_1^{+} & & \\ & \ddots & \\ & & \overline{w}_n^{+} \end{pmatrix} \quad \text{and} \quad \overline{w}_i^{+} = \begin{cases} 0 & \text{if } \overline{w}_i = 0 \\ \dfrac{1}{\overline{w}_i} & \text{if } \overline{w}_i \neq 0 \end{cases} \quad \text{for } i = 1, \ldots, n.$$

3.4.2 Second step: singular value decomposition of $\mathbf{A}^F$

Once we have computed the optimal solution $\mathbf{A}^F$ according to (3.2), we compute its singular value decomposition

$$\mathbf{A}^F = \mathbf{U}\mathbf{W}\mathbf{V}^T, \tag{3.3}$$

where $\mathbf{U}$ and $\mathbf{V}$ are two $n \times n$ orthogonal matrices, and $\mathbf{W}$ is a $n \times n$ diagonal matrix with positive or zero entries (the singular values)

$$\mathbf{W} := \begin{pmatrix} w_1 & & \\ & \ddots & \\ & & w_n \end{pmatrix},$$

such that $w_1 \geq w_2 \geq \cdots \geq w_{n-1} \geq w_n \geq 0$. We denote by $\mathbf{u}_i \in \mathbb{R}^n$ and $\mathbf{v}_i \in \mathbb{R}^n$, for $i = 1, \ldots, n$, the columns of matrices $\mathbf{U}$ and $\mathbf{V}$ respectively.

	Y hyperplanar	Y non-hyperplanar
X hyperplanar	$\dim \operatorname{Ker} \mathbf{A}^F = 1$ $\mathbf{u}^X = \mathbf{n}^X = \mathbf{n}^X_{pseudo}$ $\mathbf{u}^Y = \mathbf{n}^Y = \mathbf{n}^Y_{pseudo}$	$\dim \operatorname{Ker} \mathbf{A}^F = 1$ $\mathbf{u}^X = \mathbf{n}^X = \mathbf{n}^X_{pseudo}$ $\mathbf{u}^Y = \mathbf{u}_n$
X non-hyperplanar	$\dim \operatorname{Ker} \mathbf{A}^F = 1$ $\mathbf{u}^X = \mathbf{v}_n$ $\mathbf{u}^Y = \mathbf{n}^Y = \mathbf{n}^Y_{pseudo}$	$\dim \operatorname{Ker} \mathbf{A}^F = 0$ $\mathbf{u}^X = \mathbf{n}^X_{pseudo}$ $\mathbf{u}^Y = \mathbf{n}^Y_{pseudo}$

Table 3.1: Definition of vectors $\mathbf{u}^X$ and $\mathbf{u}^Y$ according to the sets X and Y.

3.4.3 Third step: selection of vectors $\mathbf{u}^X$ and $\mathbf{u}^Y$ and geometrical interpretation

From Equation (3.1) we realize that when the set of points X is hyperplanar the diagonal matrix $\overline{\mathbf{W}}$ has a null singular value: $\overline{w}_n = 0$. In this case, the singular value decomposition of the optimal solution $\mathbf{A}^F$ will also have a null singular value: $w_n = 0$. Therefore, to properly select $\mathbf{u}^X$ and $\mathbf{u}^Y$ we have to analyze $\operatorname{Ker} \mathbf{A}^F$ and $\operatorname{Range} \mathbf{A}^F$.

Remark 3.8. Let $\mathbf{M}$ be an $m \times n$ matrix, and $\mathbf{M} = \mathbf{U_M W_M V_M^{\mathit{T}}}$ its singular value decomposition. On one hand, the columns of the orthogonal matrix $\mathbf{V_M}$ with an associated singular value equal to zero span $\operatorname{Ker} \mathbf{M}$. On the other hand, the columns of the orthogonal matrix $\mathbf{U_M}$ with an associated positive singular value span $\operatorname{Range} \mathbf{M}$, see references Gill et al. (1991); Lawson and Hanson (1974).

Lemma 3.9. *Let X be a hyperplanar set of points and $\mathbf{A}^F$ the optimal solution of functional F computed according to Equation* (3.2)*. If $\mathbf{n}^X$ is a unitary normal vector to X and* $\dim \operatorname{Ker} \mathbf{A}^F = 1$*, then*

$$\operatorname{Ker} \mathbf{A}^F = \operatorname{Ker} \overline{\mathbf{X}}^T = \operatorname{span}(\overline{\mathbf{v}}_n) = \operatorname{span}(\mathbf{v}_n) = \operatorname{span}(\mathbf{n}^X).$$

Proof. The proof is detailed in Section B.2 of Appendix B. □

Lemma 3.10. *If Y is a hyperplanar set of points,* $\dim \operatorname{Ker} \mathbf{A}^F = 1$*, and $\mathbf{n}^Y$ is a unitary normal vector to Y, then* $(\operatorname{Rank} \mathbf{A}^F)^\perp = \mathit{span}(\mathbf{u}_n) = \mathit{span}(\mathbf{n}^Y)$.

Proof. The proof is detailed in Section B.2 of Appendix B. □

On one hand, in our algorithm we select $\mathbf{u}^X = \mathbf{v}_n$. That is, we choose $\mathbf{u}^X$ as the vector that generates $\operatorname{Ker} \mathbf{A}^F$ when $\dim \operatorname{Ker} \mathbf{A}^F = 1$. On the other hand, we also propose to select $\mathbf{u}^Y = \mathbf{u}_n$. In other words, we select $\mathbf{u}^Y$ as the vector that generates the orthogonal space to $\operatorname{Range} \mathbf{A}^F$ when $\dim \operatorname{Ker} \mathbf{A}^F = 1$. Therefore, Lemma 3.9 states that if X is a hyperplanar set of points, then our algorithm selects $\mathbf{u}^X$ as the unitary normal vector to X: $\mathbf{u}^X = \mathbf{n}^X$, which is in fact the natural choice. Moreover, Lemma 3.10 states that if Y is a hyperplanar set of points, then our algorithm selects $\mathbf{u}^Y$ as the unitary normal vector to Y: $\mathbf{u}^Y = \mathbf{n}^Y$, which is also the obvious choice. Table 3.1 presents the geometrical interpretation of the proposed selection of vectors $\mathbf{u}^X$ and $\mathbf{u}^Y$.

Note that, according to Table 3.1, when both loops of nodes X and Y are non-hyperplanar we select $\mathbf{u}^X$ and $\mathbf{u}^Y$ as $\mathbf{n}^X_{pseudo}$ and $\mathbf{n}^Y_{pseudo}$, respectively. That is, we select $\mathbf{u}^X$ and $\mathbf{u}^Y$ as *a measure* of the normal directions to the loops X and Y. In this case, and according to property *(iii)* of Lemma 2.8 of Chapter 2, we are mapping the component in the direction of the pseudo-normal of loop X to the pseudo-normal of loop Y.

3.4.4 Fourth step: computation of the affine mapping

Once we have selected vectors $\mathbf{u}^X$ and $\mathbf{u}^Y$, for any centered vector $\overline{\mathbf{x}} \in \mathbb{R}^n$ we compute the linear part of the affine projection as

$$\mathbf{A}(\overline{\mathbf{x}}) := \mathbf{A}^F(\overline{\mathbf{x}} - <\overline{\mathbf{x}}, \mathbf{u}^X> \mathbf{u}^X) + <\overline{\mathbf{x}}, \mathbf{u}^X> \mathbf{u}^Y.$$

In the case that X is a hyperplanar set we can decompose, by Lemma 2.8 of Chapter 2, $\overline{\mathbf{x}}$ as $\overline{\mathbf{x}}_{\mathbb{H}} + \lambda \mathbf{u}^{\mathbf{X}}$. Therefore, the obtained linear transformation maps $\overline{\mathbf{x}}_{\mathbb{H}}$ into $\mathbf{A}^F \overline{\mathbf{x}}_{\mathbb{H}}$ and $\lambda \mathbf{u}^{\mathbf{X}}$ to $\lambda \mathbf{u}^{\mathbf{Y}}$. Hence, by Proposition 2.21 of Chapter 2 we know that this linear mapping is the optimal solution of the minimization of functional H, obtained by means of minimizing F.

Finally, we compute the desired affine mapping according to Equation (2.4)

$$\varphi(\mathbf{x}) = \mathbf{A}(\overline{\mathbf{x}}) + \mathbf{c}^Y = \mathbf{A}(\mathbf{x} - \mathbf{c}^X) + \mathbf{c}^Y.$$

That is, to obtain the optimal solution $\mathbf{A}^H$, we first find the optimal solution $\mathbf{A}^F$, and based on its singular value decomposition we select the vectors $\mathbf{u}^X$ and $\mathbf{u}^Y$.

3.5 Testing offset data preservation

In this section we test the capability of the projection algorithm to reproduce the shape of the inner part of the projected mesh. Specifically, we show that the proposed algorithm avoids the flattening, flipping, offset scaling, and skewing effects. To highlight the analyzed capabilities we have selected four simple geometries and we have discretized them with a coarse mesh, as suggested in Tautges et al. (2004). In all examples we first project the source surface onto the target surface, see Chapter 4. Second, we obtain a structured mesh on the linking sides using a transfinite interpolation algorithm (TFI) (Thompson et al., 1999). Third, we compute the inner node location starting from each cap surface and computing the position of the new layer from the previous one in an *advancing front manner*. Finally, we compute the final inner node position by weighting the position obtained from both cap surfaces, see Chapter 4. Note that in these examples neither the boundary error correction (Blacker, 1996; White et al., 2004; Scott et al., 2005) nor any smoothing procedure is applied to the generated meshes. It is important to point out that in the four examples all the boundary loops are affine. That is, given two boundary loops an affine mapping exists than exactly maps one into the other. Therefore, functionals F and G become null in each projection (see Chapter 2 for details) and the boundary error correction will not improve the initial mesh since there is not error in the projection of the boundaries. The inner layers are numbered starting at the inner level next to the bottom cap surface. In order to measure the quality of the hexahedral mesh we use the hexahedron shape metric, f_{shape}, defined in Knupp (2001). Note that f_{shape} is a normalized measure. Therefore, it always lies in the range $[0, 1]$.

Flattening test. The goal of the first example is to illustrate that the flattening effect introduced by the minimization of functional F can be avoided using the proposed algorithm. In this example, the extrusion volume has a straight sweep path and two non-planar surfaces that have the inner part curved in the same orientation, see Figures 3.4(a) and 3.4(b). A constant element size is prescribed and 12 inner layers are generated along the extrusion path, see Figure 3.4(c).

Figures 3.5(a) and 3.5(b) show the central cross-section and the mesh quality of the obtained meshes minimizing functionals F and using the proposed algorithm, respectively. The boundary loops of the cap surfaces are such that while the sweeping process advances from one layer to the next one, the boundary loops become flatter and flatter until a planar boundary loop is reached at one quarter of the sweep path,

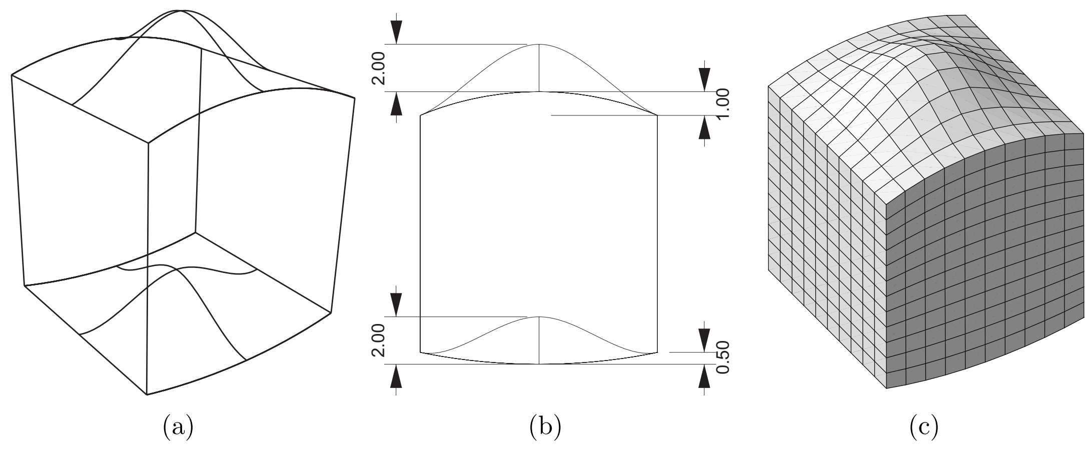


Figure 3.4: Flattening test. (a) Perspective view of the wire-frame model; (b) front view of the wire-frame model; and (c) surface mesh.

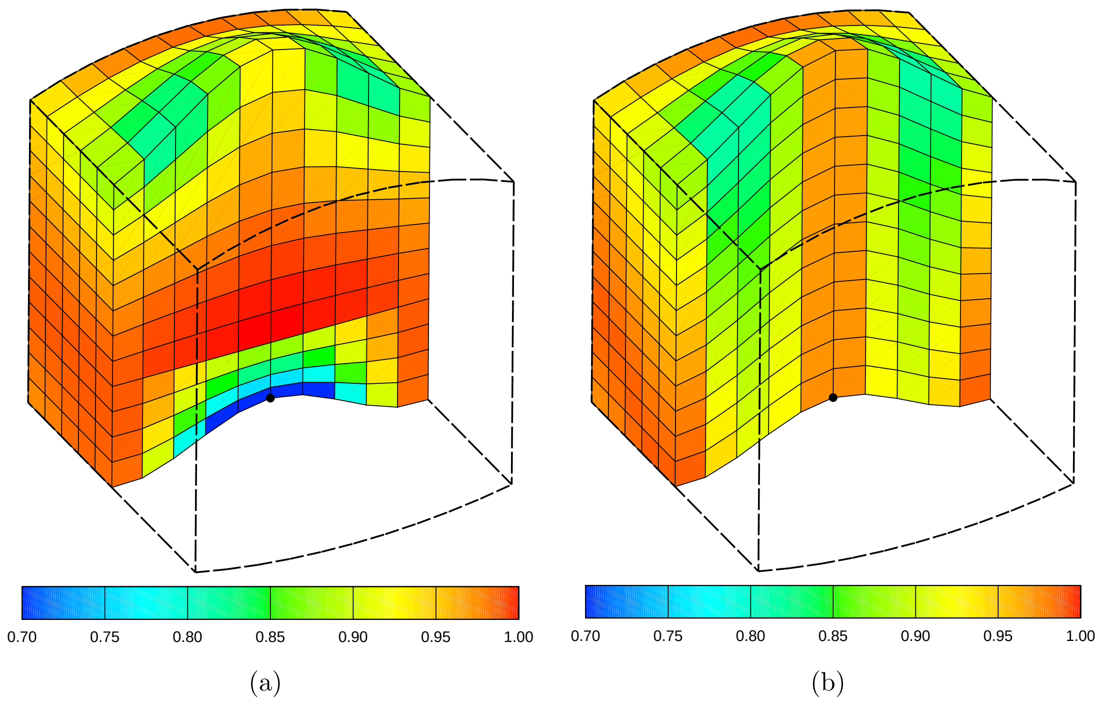
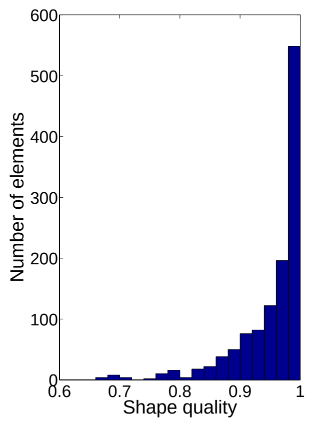
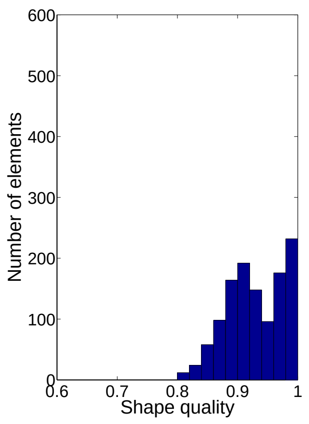


Figure 3.5: Central cross-section and shape quality values of the obtained mesh by (a) minimizing functional F; and (b) using the proposed algorithm.

see Figure 3.4(c). Therefore, the flattening and offset scaling effects produced by the minimization of functional F appear, being the flattening effect the most important. That is, in each projection the inner shape of the projected mesh becomes more



(a)

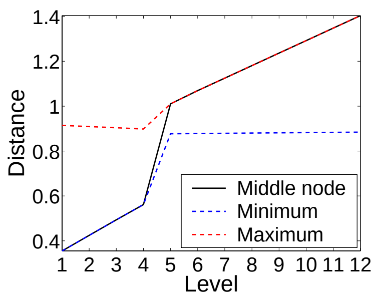


(b)

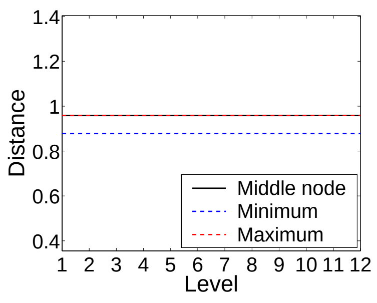
Functional	F	H
Minimum	0.6785	0.8145
Maximum	0.9995	0.9921
Mean	0.9516	0.9273
Std. dev.	0.0581	0.0458

(c)

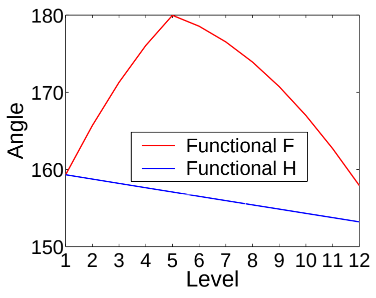
Figure 3.6: Distribution of the elements according to the shape quality measure: (a) histogram for functional F; (b) histogram for functional H; and (c) statistical values for both functionals.



(a)



(b)



(c)

Figure 3.7: Measures of the shape of the inner layers. Distances between two consecutive layers for (a) the minimization of functional F and (b) the minimization of functional H. In addition, (c) angle between two adjacent edges at the middle node.

planar. When the planar loop at one quarter of the extrusion path is reached, see Figure 3.5(a), a planar projected mesh is obtained and the offset data of the cap surfaces is completely lost. Nevertheless, the proposed algorithm also imposes that the optimal solution has to map $\mathbf{u}^{\mathbf{X}}$ into $\mathbf{u}^{\mathbf{Y}}$. According to our selection of these vectors, we take into account the offset data of the surface meshes, and in each projection, the location of the inner nodes of the new layer resemble the shape of the cap meshes, see Figure 3.5(b).

Figures 3.6(a) and 3.6(b) show the histograms of the distribution of the elements according to the shape quality measure for the generated meshes. Note that using the proposed algorithm we are able to increase the minimum quality value, $\min(f_{shape})$. However, the minimization of F generates elements with a higher value of $\max(f_{shape})$. The general behavior, which we have also observed in other examples, is that the proposed algorithm tends to increase the minimum quality value and to concentrate the quality of the elements around the mean value. Figure 3.6(c) details the statistical values of the quality of the mesh obtained minimizing both functionals.

For each inner layer of the generated meshes we compute the minimum and the maximum distance between two consecutive nodes in the extrusion direction. Note that the node at which these distances are reached may be different for each pair of two consecutive inner layers. We denote by *middle node* the mesh node located at the center of the cap surfaces, marked with a $\bullet$ in Figure 3.5. In addition, we compute the distance between the middle node locations for each two consecutive layers. Figures 3.7(a) and 3.7(b) plot the obtained values. Note that the minimization of functional F generates inner layers with a wide range of variation for these distances. On the contrary, the proposed algorithm maintains these three distances almost constant in all layers. Figure 3.7(c) presents for each layer the angle defined between the two mesh edges adjacent to the middle node and contained in the cutting plane presented in Figure 3.5. The bottom cap surface corresponds to the first level and the top cap surface corresponds to the twelfth level. Note that for the mesh obtained by the minimization of functional F the value of this angle first increases from 160° to 180° where a planar inner layer is generated. Then, it decreases again to 160°. On the contrary, for the mesh obtained by the proposed algorithm the value of the angle monotonically decreases from 160° to 155°.

Flipping test. In the second example we illustrate the ability of the proposed algorithm to avoid the flipping effect introduced by the minimization of functional F. In this example we discretize an extrusion volume defined by two non-planar cap surfaces and a straight sweep path, see Figures 3.8(a) and 3.8(b). Due to the shape of the boundary loops of the cap surfaces, the loops of nodes that define the boundary of the inner layers are curved towards the top surface except the first one which is curved towards the bottom surface, see Figure 3.8(c). Therefore, the minimization of functional F introduces the flipping effect. Figure 3.9(a) shows that all inner layers ranging from the second to the eighth layer are curved towards the bottom cap surface, whereas their boundary loops are curved towards the top cap surface. However, according to the proposed algorithm we are able to detect the proper direction of

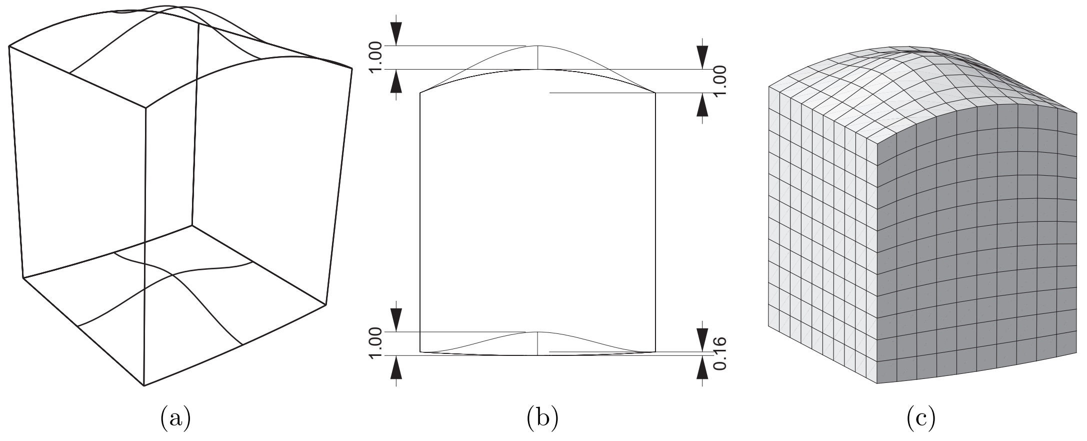


Figure 3.8: Flipping test. (a) Perspective view of the wire-frame model; (b) front view of the wire-frame model; and (c) surface mesh.

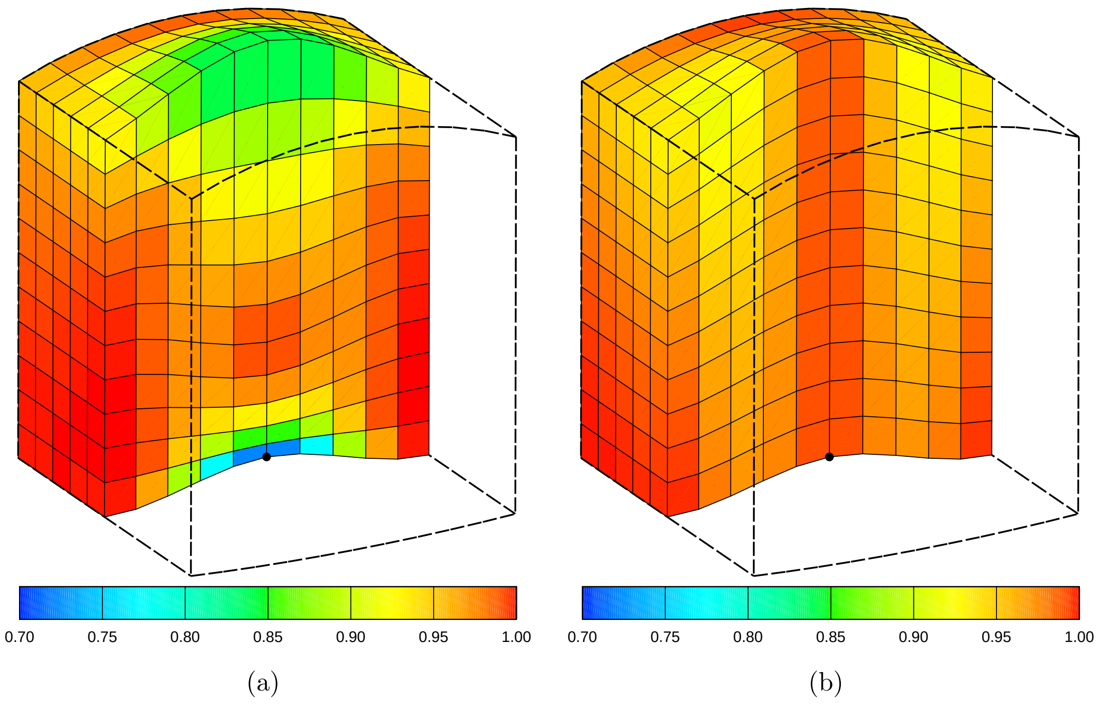
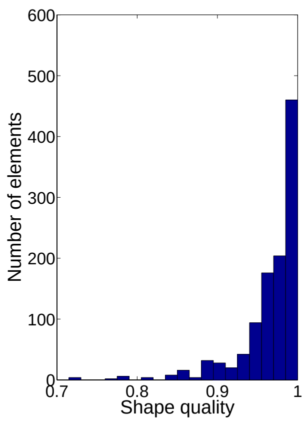


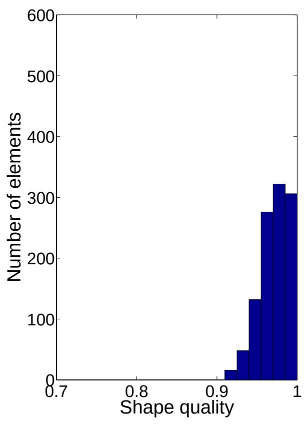
Figure 3.9: Central cross-section and shape quality values of the obtained mesh by (a) minimizing functional F; and (b) using the proposed algorithm.

vectors $\mathbf{u}^X$ and $\mathbf{u}^Y$. Hence, the shape of the cap surfaces is properly reproduced in the inner layers of nodes and a high quality mesh is generated, see Figure 3.9(b).

Figure 3.10 presents the distribution of the elements according to their shape



(a)



(b)

Functional	F	H
Minimum	0.7214	0.9156
Maximum	0.9994	0.9982
Mean	0.9656	0.971
Std. dev.	0.0401	0.0177

(c)

Figure 3.10: Distribution of the elements according to the shape quality measure: (a) histogram for functional F; (b) histogram for functional H; and (c) statistical values for both functionals.

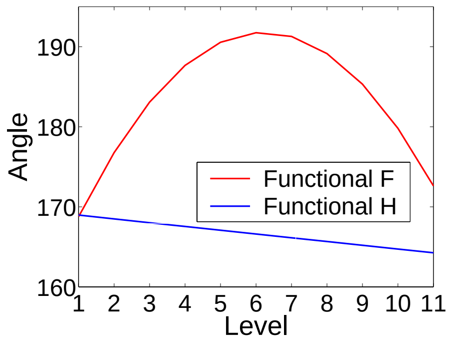


Figure 3.11: Angle between two adjacent mesh edges at middle node.

quality. Note that the proposed algorithm increases the minimum quality value, $\min(f_{shape})$, of the shape quality measure and generates a mesh with a better quality distribution. Moreover, the distribution of the elements is reduced to the interval $f_{shape} \in [0.91, 0.99]$. Similar to the first example we mark the middle node with a $\bullet$ in Figure 3.9. Figure 3.11 presents, for each layer, the angle defined between the two mesh edges adjacent at the middle node and contained in the cross-section presented in Figure 3.9. It clearly shows that the flipping effect appears when functional F is minimized since the angle increases from 170° in the first inner layer to 190° in the sixth inner layer, and then decreases to 170° in the eleventh inner layer. On the contrary, for the mesh obtained by the proposed algorithm the value of the angle monotonically decreases from 169° to 164°.

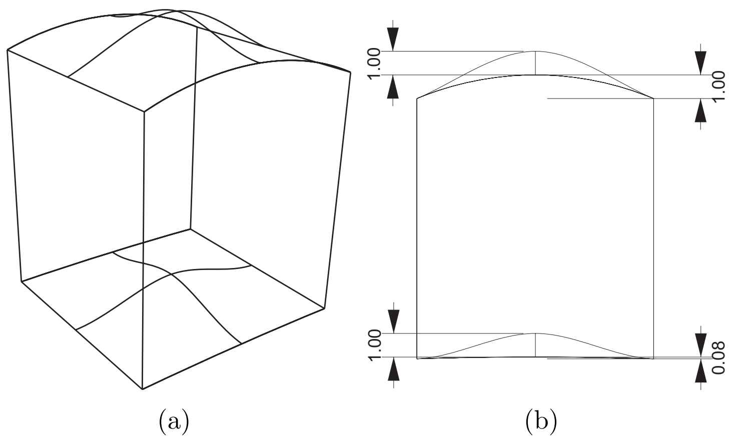


Figure 3.12: Wire-frame model of the geometry used for the offset scaling test: (a) perspective view; and (b) front view.

Offset scaling test. The goal of the third example is to illustrate how the proposed algorithm reduces the offset scaling effect. Figure 3.12 presents two views of the wire-frame model of the geometry corresponding to this example. Two non-planar cap surfaces and a straight extrusion path define the geometry. Since the thickness of the top boundary is higher than the thickness of the bottom boundary, see Figure 3.12(b), the offset scaling effect will appear when the mesh is obtained minimizing functional F. Figure 3.13(a) shows the cross-section and the quality of the obtained mesh when functional F is minimized. Note that the inner layers of elements are more curved at the bottom of the extrusion path leading to highly distorted hexahedral elements at the top of the sweep path. On the contrary, Figure 3.13(a) presents the cross-section and the quality of the generated mesh when the new algorithm is used. Note that in this case a more graded distribution of the element size is obtained along the sweep path, and the inner layers of elements reproduce the shape of the cap surfaces.

Figures 3.14(a) and 3.14(b) show the element distribution according to the shape quality of the element for the meshes obtained by means of functional F and H, respectively. Figure 3.6(c) details the statistical values of the quality for both meshes. Note that using the proposed algorithm the minimum value of the quality measure is more than two times higher than the minimum value obtained minimizing functional F.

For each one of the generated meshes, Figures 3.15(a) and 3.15(b) plot the maximum and the minimum distance between two consecutive nodes in the extrusion

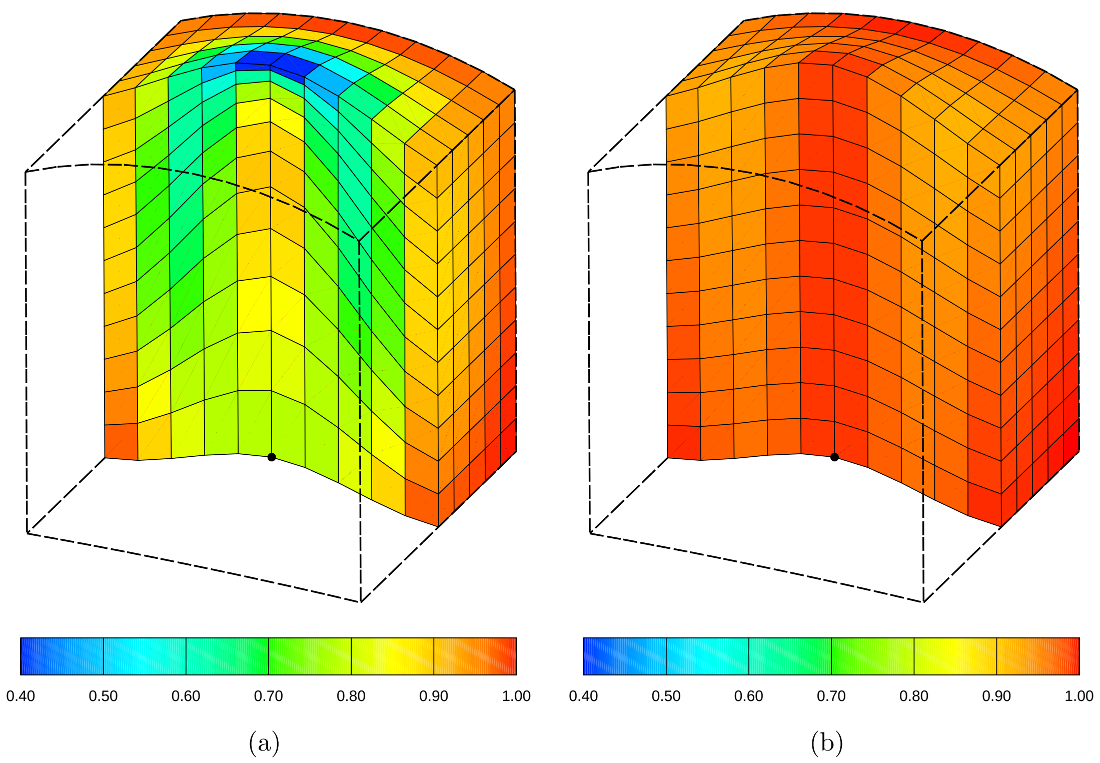
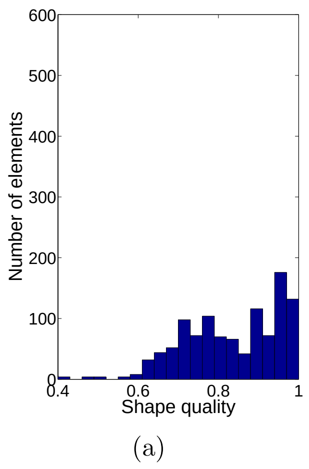
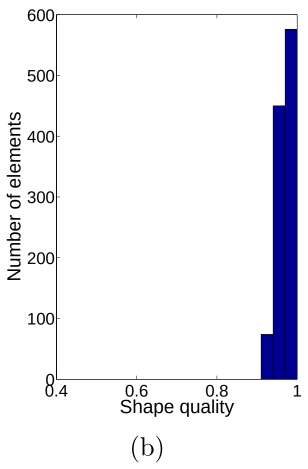


Figure 3.13: Central cross-section and shape quality values of the obtained mesh by (a) minimizing functional F; and (b) using the proposed algorithm.





Functional	F	H
Minimum	0.3949	0.9163
Maximum	0.9972	0.9981
Mean	0.8345	0.9700
Std. dev.	0.1190	0.0178

(c)

Figure 3.14: Distribution of the elements according to the shape quality measure: (a) histogram for functional F; (b) histogram for functional H; and (c) statistical values for both functionals.

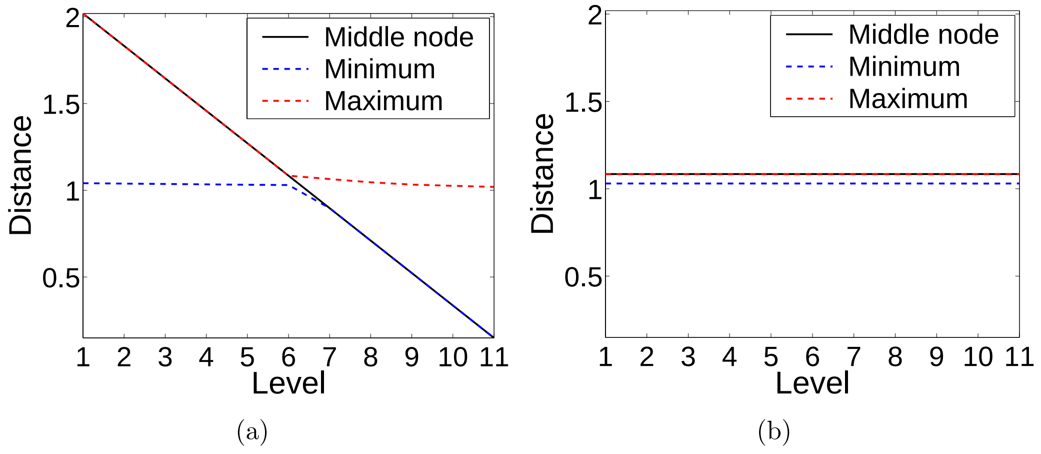


Figure 3.15: Minimum, maximum and middle node distances from one level to next one: (a) for the minimization of functional F; and (b) for the minimization of functional H.

direction. In addition, they also plot the distance between the middle node locations in two consecutive layers. Similar to the first example, the proposed algorithm maintains these three distances almost constant in all layers.

Skewing test. In the fourth example we illustrate the ability of the proposed algorithm to avoid the skewing effect introduced by the minimization of functional G. In this example a non-planar surface with a planar boundary is swept along a circular extrusion path. Figure 3.16 presents a wire-frame model of the geometry. Since the boundary loop of the cap surfaces is planar we know that the minimization of functional F will generate planar inner layers. Therefore, we do not present the meshes obtained by minimizing F. The minimization of functional G generates non-planar inner layers, see Figure 3.17(a). Since the planar loop of nodes that define each layer are non-parallel and the vector $\mathbf{c}^Y - \mathbf{c}^X$ is a fixed vector of the optimal solution of the minimization of G (see Chapter 2 for details), the skewing effect appears on the generated mesh. Figure 3.17(a) presents a cross-section and the quality of the obtained mesh minimizing functional G. Note that the shape of the inner layers is skewed towards the center of the extrusion path. Moreover, distorted hexahedra are generated in the layers located at one half of the extrusion path. On the contrary, Figure 3.17(b) shows a cross-section and the quality of the mesh generated using the proposed algorithm. In this case, the inner layers are not skewed and reproduce the shape of the cap surfaces.

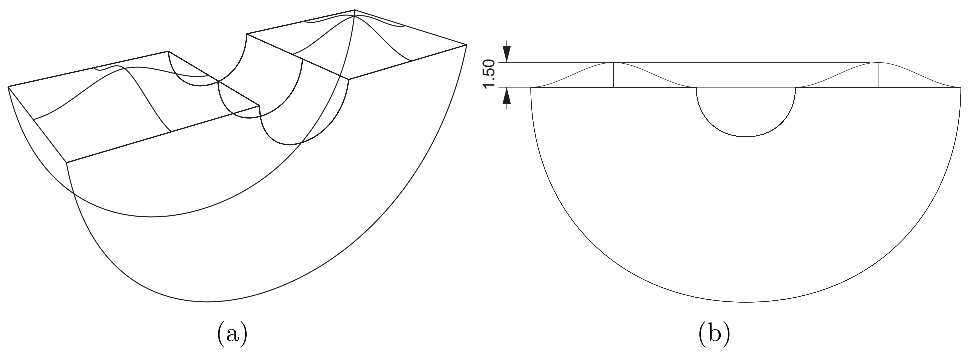


Figure 3.16: Wire-frame model of the geometry used for the skewing test: (a) perspective view; and (b) front view.

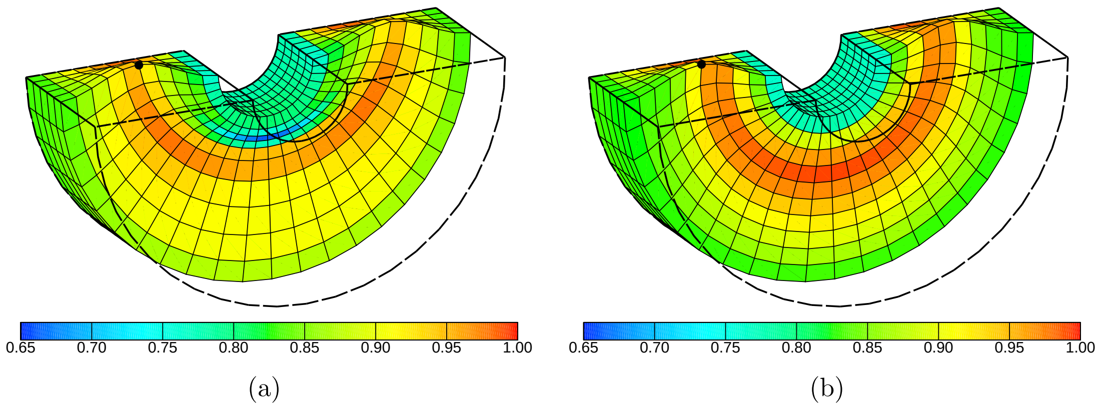
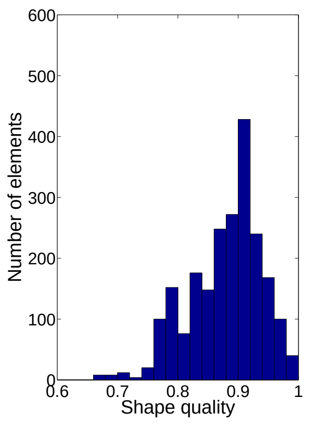
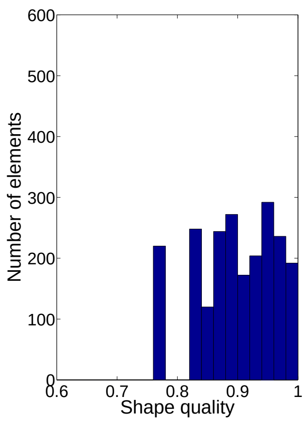


Figure 3.17: Central cross-section and shape quality values of the obtained meshes (a) minimizing functional G; and (b) using the proposed algorithm.

Figures 3.18(a) and 3.18(b) present the distribution of the elements according to the shape quality measure for the meshes generated minimizing functional G and using the proposed algorithm. Note that the minimum value of the shape quality is increased if the proposed algorithm is used. Figure 3.6(c) details the statistical values of the quality of the mesh obtained minimizing both functionals.

Figure 3.19 shows the minimum, maximum and middle node distances for each pair of consecutive inner layers corresponding to the meshes generated by means of functionals G and H. Both functionals generate meshes with the same values for the minimum and the maximum distances. However, the distance between consecutive locations of the middle point reach a minimum value at the middle of the extrusion path whereas the proposed algorithm maintains almost constant this distance.

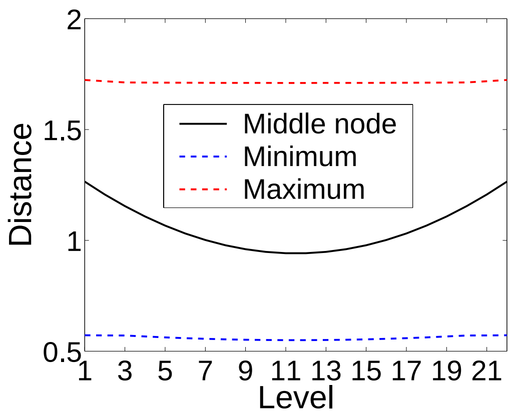

(a)


(b)

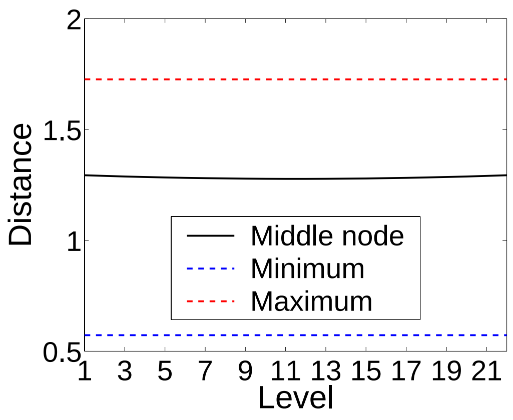
Functional	G	H
Minimum	0.6621	0.7648
Maximum	0.9828	0.9920
Mean	0.8792	0.8973
Std. dev.	0.0595	0.0629

(c)

Figure 3.18: Distribution of the elements according to the shape quality measure: (a) histogram for functional G; (b) histogram for functional H; and (c) statistical values for both functionals.



(a)



(b)

Figure 3.19: Minimum, maximum and middle node distances from one level to next one for: (a) the minimization of functional G; and (b) the minimization of functional H.

3.6 Concluding remarks

In this chapter we have proposed and detailed a node projection algorithm to obtain hexahedral meshes in one-to-one sweep geometries. Note that one-to-one projection procedures belong to the core of many-to-many sweep algorithms. The main target of the developed algorithm is to preserve non-planar shape of the cap surfaces in the inner layers of the hexahedral mesh. To this end, we first have introduced the pseudo-area and pseudo-normal vectors. In addition, we have also proved several useful geometrical properties of these vectors. Then, we have detailed an algorithm that allows us to select the two vector parameters of functional H. We claim that the presented algorithm generates inner layers that preserve the shape of the cap surfaces. Moreover, several test geometries show that this algorithm overcomes flattening, skewing, offset scaling, and flipping effects introduced by the minimization of the traditional functionals.

Chapter 4

A new sweeping scheme based on affine methods

4.1 Introduction

Taking into account the definition of extrusion geometry, in Chapter 2 we have presented the traditional procedure to generate an all–hexahedral mesh by sweeping: *(i)* to generate a structured or unstructured quadrilateral mesh on the source surface; *(ii)* to map the source mesh into the target surface; *(iii)* to generate a structured quadrilateral mesh over the linking sides; *(iv)* to generate the inner layers of nodes; and *(v)* to generate the hexahedral elements by connecting the nodes of two consecutive layers.

In this chapter we present an update of the new sweeping scheme introduced in (Roca et al., 2006). Specifically, we detail the implementation for the second and the fourth steps. In both steps, a method for projecting surface meshes along the sweep path is required. The meshes over the target surface and the inner layer meshes must be topologically equivalent to the source surface mesh. The main contributions of this chapter are:

- **To propose a fast surface mesh projection between parameterized surfaces**. Several algorithms have been developed to map meshes between surfaces (Goodrich, 1997). Most of them involve an orthogonal projection of nodes onto the target surface. These projections are expensive from a computational point of view since it is necessary to solve as many root finding problems as internal points are on the grid of the source surface. In order to overcome this shortcoming, in this chapter we present a new and efficient algorithm to map a given mesh over the source surface onto the target surface. This projection is determined by means of a least-squares approximation of an affine mapping defined between the parametric representation of the loops of boundary nodes of the cap surfaces. Once the new mesh is obtained on the parametric space of the target surface, it is mapped up according to the target surface parameterization.

- **To generate the inner layers of nodes by directly projecting from the source and the target**. The classical strategy to generate inner layers is to project the cap meshes from one layer to the next one in an *advancing front manner*. Therefore, the node location of an inner layer of nodes depends on the location of the previous and next layers. We propose a new scheme where the inner layers are generated by projecting directly from both cap surfaces. Hence, the location of the inner nodes only depends on the cap surface meshes and the procedure can be parallelized. Note that the developed algorithm to map meshes between cap surfaces can not be directly applied in order to generate the inner layer of nodes, since these layers are not defined by parametric surfaces. In fact, the available data to determine the position of the inner layers of nodes is the loops of nodes on the linking surfaces, and the cap surface meshes. Therefore, the projection algorithm detailed in Chapter 3 is used. Then, the inner nodes are located using a weighted least-squares approximation of the transformation between the boundary nodes of the cap surfaces and the boundary nodes of the layer as in (Blacker, 1996).

The remainder of this chapter is organized as follows. In Section 4.2 we explain how the surfaces of the extrusion volume are discretized. Specifically, we present the fast projection between parameterized surfaces. To finalize the implementation of the proposed sweeping method, in Section 4.3 we detail the generation of the inner layers of nodes. In Section 4.4 we present several examples to show the capabilities of the proposed sweep method.

4.2 Generation of surface meshes

Sweep volumes are usually defined by commercial CAD packages in industrial applications. Hence, source and target surfaces might be trimmed surfaces. We assume that a geometrical kernel is available, such that surfaces are parameterized. In fact, this kernel must provide query functions to obtain the physical coordinates of a parametric point.

4.2.1 Meshing the source

The source surface mesh can be generated with any unstructured quadrilateral mesh generator that deals with trimmed and parameterized surfaces. Specifically, we use an extended version to parametric surfaces of a previously developed unstructured quadrilateral mesh generator by Sarrate and Huerta (2000b,a). However, other quadrilateral mesh generators can be used such as Blacker and Stephenson (1991); Cass et al. (1996).

4.2.2 Meshing the target: fast projection of surface meshes

Once the source surface, S, is meshed the next step in the sweep algorithm is to map it onto the target surface, T. As it has been previously noted, most of the developed algorithms to map meshes between surfaces have to solve several root finding problems. In order to overcome this drawback, a new and efficient algorithm is devised to map meshes between trimmed surfaces. In fact, the mapping is defined between the parametric spaces of the surfaces, D_S and D_T. Then, the obtained mesh is mapped up according to the target surface parameterization. For some surfaces, it might be necessary to smooth the new surface mesh. Note that this smoothing is also needed in other methods (Goodrich, 1997).

First of all, we will show that determining a projection between trimmed surfaces is equivalent to finding out a projection between their parametric spaces. To this end, assume that the source and target surfaces are trimmed surfaces. Let

$$\begin{aligned}\boldsymbol{\psi}_S :&\ \ V_S \subset \mathbb{R}^2 \to \mathbb{R}^3\\ \boldsymbol{\psi}_T :&\ \ V_T \subset \mathbb{R}^2 \to \mathbb{R}^3\end{aligned}$$

be their extended parameterization, where V_S and V_T are two open and bounded sets of $\mathbb{R}^2$. Note that the domain of a trimmed surface, in general, is not a rectangle $[a,b]\times[c,d] \subset \mathbb{R}^2$ in the parametric space. We assume that $\boldsymbol{\psi}_S$ and $\boldsymbol{\psi}_T$ are continuous and injective, then the Brouwer's theorem on invariance of domain (Hatcher, 2002) states that they are also open mappings. Therefore, their restrictions (the definition of the trimmed surfaces)

$$\boldsymbol{\psi}_S|_{D_S} : D_S \subset V_S \to S \subset \mathbb{R}^3 \qquad \boldsymbol{\psi}_T|_{D_T} : D_T \subset V_T \to T \subset \mathbb{R}^3,$$

are homeomorphisms in D_S and D_T respectively. Hence, we have

$$S = \boldsymbol{\psi}_S(D_S) \qquad T = \boldsymbol{\psi}_T(D_T).$$

Recall that our aim is to determine a mapping $\widetilde{\boldsymbol{\phi}} : S \to T$ such that, given a mesh, $\mathcal{M}_S$, over the source surface, it yields a mesh, $\mathcal{M}_T$, onto the target surface with the same connectivities. Since S and T have the same topology we can assume that $\widetilde{\boldsymbol{\phi}}$ is also a homeomorphism.

Taking into account that $\boldsymbol{\psi}_S|_{D_S}$, $\boldsymbol{\psi}_T|_{D_T}$ and $\widetilde{\boldsymbol{\phi}}$ are homeomorphisms, it is possible to define

$$\boldsymbol{\phi} := \boldsymbol{\psi}_T|_{D_T}^{-1} \circ \widetilde{\boldsymbol{\phi}} \circ \boldsymbol{\psi}_S|_{D_S},$$

such that

$$\begin{tikzcd} S \subset \mathbb{R}^3 \arrow[r, "\widetilde{\boldsymbol{\phi}}"] & T \subset \mathbb{R}^3 \\ D_S \subset \mathbb{R}^2 \arrow[u, "\boldsymbol{\psi}_S|_{D_S}"] \arrow[r, "\boldsymbol{\phi}"] & D_T \subset \mathbb{R}^2 \arrow[u, "\boldsymbol{\psi}_T|_{D_T}"'] \end{tikzcd} \tag{4.1}$$

Under these conditions, the diagram of mappings (4.1) is a commutative diagram. Hence, it is feasible to find first the projection $\boldsymbol{\phi}$, homeomorphism between the parametric domains D_S and D_T, and then, mapping up the new mesh onto the target surface, T, according to its parameterization, $\boldsymbol{\psi}_T|_{D_T}$, as

$$\widetilde{\boldsymbol{\phi}} = \boldsymbol{\psi}_T|_{D_T} \circ \boldsymbol{\phi} \circ \boldsymbol{\psi}_S|_{D_S}{}^{-1}. \tag{4.2}$$

Note that it is not required to deduce the analytical expression of the inverse function $\boldsymbol{\psi}_S|_{D_S}{}^{-1}$. It suffices to store the pre-images of nodal coordinates of $\mathcal{M}_S$ by $\boldsymbol{\psi}_S|_{D_S}$. This can be achieved if the application stores both the physical and the parametric coordinates of each mesh node.

Therefore, special attention has to be focused on to settle the projection between D_S and D_T from the available data. Let m, with $m \geq 3$, be the number of nodes on all the boundary loops of the cap surfaces. We assume that each cap surface is delimited by one outer boundary and one inner boundary for each hole. These boundaries are previously meshed, and a series of loops of nodes on the boundary of the surface are obtained (see Figure 4.1(a)). Let $U_S = \{\mathbf{u}_S^i\}_{i=1,\ldots,m} \subset \mathbb{R}^2$ and

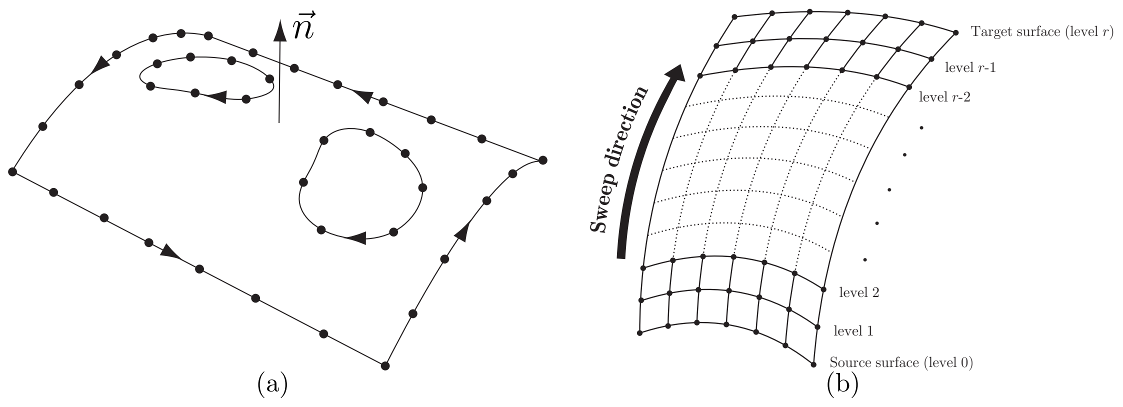


Figure 4.1: **(a)** Boundary nodes of a non simple connected surface; **(b)** discretization of a linking side using $r-1$ inner levels.

$U_T = \{\mathbf{u}_T^i\}_{i=1,\dots,m} \subset \mathbb{R}^2$ be the parametric coordinates of all boundary nodes of the source and target surfaces, respectively. It is important to point out that the physical coordinates of these points (i.e. their images by $\boldsymbol{\psi}_S|_{D_S}$ and $\boldsymbol{\psi}_T|_{D_T}$) do not necessarily determine planar loops. The goal is to find a function $\boldsymbol{\phi}$ such that

$$\boldsymbol{\phi}(\mathbf{u}_S^i) = \mathbf{u}_T^i, \qquad i = 1, \dots, m. \tag{4.3}$$

In this algorithm, the homeomorphism $\boldsymbol{\phi}$ is approximated by an affine mapping

$$\mathbf{u}_T \;=\; \boldsymbol{\phi}(\mathbf{u}_S) \;\approx\; \mathbf{A}(\mathbf{u}_S - \mathbf{c}^{U_S}) + \mathbf{c}^{U_T}, \tag{4.4}$$

where $\mathbf{u}_S$ and $\mathbf{u}_T$ are points on D_S and D_T respectively, $\mathbf{A}$ is a linear transformation, and

$$\mathbf{c}^{U_S} := \frac{1}{m}\sum_{i=1}^{m} \mathbf{u}_S^i, \text{ and } \mathbf{c}^{U_T} := \frac{1}{m}\sum_{i=1}^{m} \mathbf{u}_T^i.$$

Note that in practical applications the loop of source points are not aligned because they lay on the boundary of a non-degenerated surface. Under these conditions Proposition 2.11 states that the minimization of functional F leads to a full rank system of normal equations. Therefore, the optimal solution A^F of the minimization of functional F, see implementation details in Section 3.4 of Chapter 3, determines the affine approximation

$$\boldsymbol{\varphi}^F(\mathbf{u}_S) := \mathbf{A}^F(\mathbf{u}_S - \mathbf{c}^{U_S}) + \mathbf{c}^{U_T}. \tag{4.5}$$

In conclusion, an affine mapping (4.5) between parametric spaces has been found that fits, in the least-squares sense, the loops of boundary data. This transformation $\boldsymbol{\varphi}^F$ is used to approximate the map $\boldsymbol{\phi}$ between meshes from D_S to D_T.

Finally, to obtain the mesh $\mathcal{M}_T$ it is only needed to map up the nodes onto the target surface T. To this end, according to (4.2) we approximate $\widetilde{\boldsymbol{\phi}}$ by

$$\widetilde{\boldsymbol{\varphi}}^F(\mathbf{p}) := \boldsymbol{\psi}_T|_{D_T}\Big(\boldsymbol{\varphi}^F\big(\boldsymbol{\psi}_S|_{D_S}{}^{-1}(\mathbf{p})\big)\Big), \qquad \mathbf{p} \in S. \tag{4.6}$$

Note that, since the values of $\boldsymbol{\psi}_S|_{D_S}{}^{-1}$ are known in all nodes of $\mathcal{M}_S$, the new mesh on the target surface can be defined as

$$\mathcal{M}_T := \widetilde{\boldsymbol{\varphi}}^F(\mathcal{M}_S).$$

4.2.3 Linking-sides mesh generation

Since linking-sides are always defined by four logical sides (each logical side can be composed by several edges), they can be meshed using any standard structured quadrilateral mesh algorithm, for instance, transfinite mapping (TFI) (Haber et al., 1981). In order to apply the TFI method it is required that opposite logical sides will have the same number of nodes. It is important to have high quality structured meshes on the linking-sides. Note that these meshes determine the loops of nodes that are used later to generate the inner volume nodes in a layer by layer procedure (see Figure 2.1). Thus, if these surface meshes contains folded or low quality elements, then tangled meshes, reverse oriented or low quality hexahedral elements are obtained.

A hard test for a sweep algorithm is to mesh an extrusion volume with an S-shaped changing sweep direction, see Figure 4.2. It is well known that obtaining a good structured mesh over an S-shaped surface is non–trivial. If nodes are generated equidistant along the edges of the S-shaped surface, then some segments of the structured surface mesh cross over each other, see Figure 4.2(a). Thus, folded quadrilateral elements are obtained in the middle part of the surface mesh. Therefore, tangled hexahedral elements are generated inside of the sweep volume.

To solve this drawback, we have implemented an edge mesher procedure that is able to "follow" nodes on the opposite edge of the S-shaped surface. To this end, the distance between two opposite nodes across the surface is minimized. Using this procedure, consecutive joining segments will not cross each other at the middle part of the surface, see Figure 4.2(b). The details of this edge mesher are out of the scope

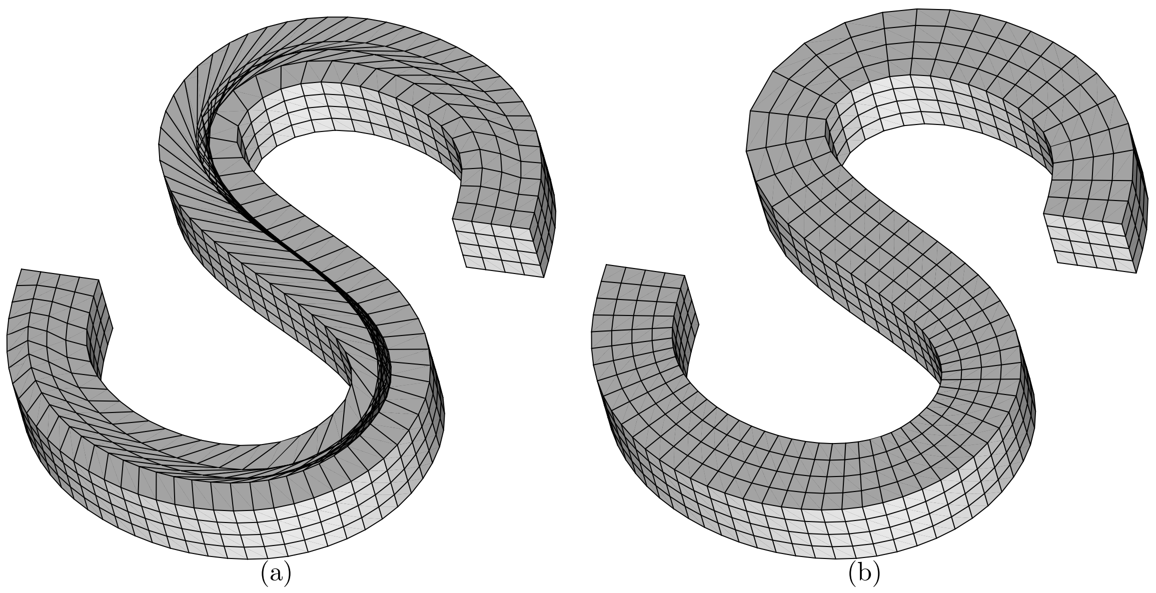


Figure 4.2: S-shaped sweep volume. **(a)** Equidistant nodes on the edges and folded elements; **(b)** well positioned edge nodes for the linking-sides structured mesh generation.

of this work. However, it is important to note that no surface mesh smoothing is required in this procedure.

4.3 Generation of inner nodes and elements

Once all boundaries are meshed, inner nodes of the extrusion volume have to be generated. These nodes have to be placed, layer by layer, along the sweep direction. Each layer is delimited by several loops of nodes that belongs to the structured meshes of the linking-sides (see Figure 2.1). In fact, for every layer there is one outer loop, and one inner loop for each hole in the sweep volume. Note that the method developed in section 4.2.2 can not be used here because no surface parameterization, $\boldsymbol{\varphi}_T|_{D_T}$, is available for these layers. Hence, the projection algorithm presented in Section 3.4 of Chapter 3 is used. Note that we can project from source level and from target level. Thus, a weighted least-squares approximation is developed, similar to those proposed in Knupp (1998, 1999); Blacker (1996).

Assume that $r-1$ inner levels of nodes have been generated on the linking-sides along the sweep direction (see Figure 4.1(b)). Then, $r-1$ layers of inner nodes have to be generated. Let $X_0 = \{\mathbf{x}_0^i\}_{i=1,\ldots,m} \subset \mathbb{R}^3$, $X_r = \{\mathbf{x}_r^i\}_{i=1,\ldots,m} \subset \mathbb{R}^3$ and $X_k = \{\mathbf{x}_k^i\}_{i=1,\ldots,m} \subset \mathbb{R}^3$ with $k = 1, \ldots, r-1$ be the physical coordinates of the boundary nodes of: the source surface (level 0), the target surface (level r) and the k-th level, respectively.

For each level k, we look for a mapping $\boldsymbol{\phi}_k^0$ such that

$$\mathbf{x}_k^i = \boldsymbol{\phi}_k^0(\mathbf{x}_0^i), \qquad i = 1, \ldots, m. \tag{4.7}$$

We approximate $\boldsymbol{\phi}_k^0$ by an affine mapping $\boldsymbol{\varphi}_k^0$ that preserves offset data. To this end, we use the affine method detailed in Algorithm 3.1.

A similar process could be defined using the target surface as initial surface instead of the source surface. Hence, we can obtain an affine mapping $\boldsymbol{\varphi}_k^r$ that maps points of the target surface to the k-th level.

Let $Z_0 = \{\mathbf{z}_0^j\}_{j=1,\ldots,l} \subset \mathbb{R}^3$, $Z_r = \{\mathbf{z}_r^j\}_{j=1,\ldots,l} \subset \mathbb{R}^3$, and $Z_k = \{\mathbf{z}_k^j\}_{j=1,\ldots,l} \subset \mathbb{R}^3$ with $k = 1, \ldots, r-1$ be the physical coordinates of the inner nodes of: the source mesh $\mathcal{M}_S$, the target mesh $\mathcal{M}_T$, and the k-th level, respectively. Note that the points in Z_r are the projections of the points in Z_0 to the target surface, see Section 4.2.2. We have two affine mappings $\boldsymbol{\varphi}_k^0$ and $\boldsymbol{\varphi}_k^r$ that allow to map the source mesh $\mathcal{M}_S$ and the target mesh $\mathcal{M}_T$ to the k-th level. Thus, according to Blacker (1996) we consider the following weighted transformation

$$\mathbf{z}_k^j := \left(1 - \frac{k}{r}\right)\boldsymbol{\varphi}_k^0(\mathbf{z}_0^j) + \frac{k}{r}\boldsymbol{\varphi}_k^r(\mathbf{z}_r^j), \tag{4.8}$$

where $\mathbf{z}_0^j \in \mathcal{M}_S$, $\mathbf{z}_r^j \in \mathcal{M}_T$, $k = 1, \ldots, r-1$, and $j = 1, \ldots, l$.

Finally, hexahedral elements are generated by joining the corresponding nodes between adjacent layers of quadrilateral meshes.

Figure 4.3 presents the discretization of an extrusion volume defined by two non-planar cap surfaces with different curvature. Figure 4.3(a) shows the surface mesh, and Figure 4.3(b) shows a detail of the shape of the inner hexahedral elements. Note that the weighted function (4.8) generates layers of elements with a smooth transition of the curvature from the source surface to the target surface. No smoothing was applied to obtain this mesh.

4.3.1 Implementation

Algorithm 4.1 details an implementation of the above method for generating the inner layers of nodes. It is a parallel implementation that considers a modification

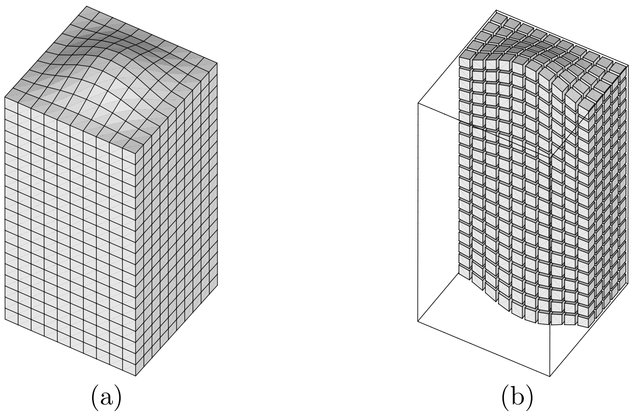


Figure 4.3: Extrusion volume with non planar cap surfaces and inner nodes placed according the weighted interpolation. **(a)** Surface mesh; **(b)** inner elements.

Algorithm 4.1: Generate inner nodes

```
   Input: number of levels, $r$
   Input: number of inner nodes per level, $l$
   Input: coordinates for boundary nodes at different levels, $X_0, \ldots, X_r$
   Output: coordinates for inner points at different levels, $Z_0, \ldots, Z_r$
1  parallel block
2      shared variables: $r$,$l$,$X_0, \ldots, X_r$,$Z_0, \ldots, Z_r$
3      private variables: $k, i, \boldsymbol{\varphi}_k^0, \boldsymbol{\varphi}_k^r, E_0, E_r$
4      parallel for $k = 1, \ldots, r-1$ do
5          $\boldsymbol{\varphi}_k^0$ :=obtain affine mapping($X_0, X_k$)
6          $\boldsymbol{\varphi}_k^r$ :=obtain affine mapping($X_r, X_k$)
7          $E_0$:=boundary error($X_0, X_k, Z_0$)
8          $E_r$:=boundary error($X_r, X_k, Z_r$)
9          for $j = 1, \ldots, l$ do
10             $\mathbf{z}_k^j := (1 - \frac{k}{r})(\boldsymbol{\varphi}_k^0(\mathbf{z}_0^j) + \mathbf{e}_0^j) + \frac{k}{r}(\boldsymbol{\varphi}_k^r(\mathbf{z}_r^j) + \mathbf{e}_r^j)$
11
```

of Equation (4.8) to include a boundary error correction procedure (Blacker, 1996; White et al., 2004):

- **Parallelization**. Standard methods for generating the inner layers of nodes proceed in an advancing front manner. To obtain the inner nodes of the k-th level a projection from previous and next level are performed. On the contrary, we propose a method to obtain the k-th level by means of a projection from the source and the target meshes. Therefore, the location of the inner nodes of k-th level do not depend on the inner nodes of the previous and the next levels. According to it, the main loop can be parallelized and the location of the inner nodes can be shared between different threads.

- **Boundary error**. Since the boundary nodes of the k-th level are known, we can calculate the error of the mappings at the boundary nodes

$$\begin{aligned}\varepsilon^i_{k,0} &= \mathbf{x}^i_k - \boldsymbol{\varphi}^0_k(\mathbf{x}^i_0)\\ \varepsilon^i_{k,r} &= \mathbf{x}^i_k - \boldsymbol{\varphi}^r_k(\mathbf{x}^i_r),\end{aligned}$$

where $i = 1, \ldots, m$. Then, these errors can be interpolated to obtain the error for the inner nodes. To this end, we consider the boundary error method proposed in Blacker (1996) and detailed in White et al. (2004). Applying this method we obtain the sets $E_{k,0} = \{\mathbf{e}^j_{k,0}\}_{j=1,\ldots,l}$ and $E_{k,r} = \{\mathbf{e}^j_{k,r}\}_{j=1,\ldots,l}$. These sets contain the error interpolation for the inner nodes projected from the source and the target respectively. Thus, we consider a modification of Equation (4.8)

$$\mathbf{z}^j_k := \left(1 - \frac{k}{r}\right)(\boldsymbol{\varphi}^0_k(\mathbf{z}^j_0) + \mathbf{e}^j_0) + \frac{k}{r}(\boldsymbol{\varphi}^r_k(\mathbf{z}^j_r) + \mathbf{e}^j_r) \tag{4.9}$$

where $\mathbf{z}^j_0 \in \mathcal{M}_S$, $\mathbf{z}^j_r \in \mathcal{M}_T$, $k = 1, \ldots, r-1$, and $j = 1, \ldots, l$.

4.4 Numerical examples and applications

4.4.1 Testing sweeping algorithm

In order to asses the quality of the sweep algorithm described in this chapter, five examples are presented. The geometry of the examples is defined using several commercial CAD softwares. The user assigns the element size, and the application automatically determines the cap surfaces and the linear or non linear sweeping direction. The first example shows the discretization of a mill head with linear sweeping axis and rotated cap surfaces, see Figure 4.4(a). It is composed by 1170 hexahedral elements. As it can be seen, an unstructured quadrilateral mesh is generated over the cap surfaces. Although the sweeping axis is twisted, high quality elements are generated without a posteriori mesh smoothing.

In the second example, a source surface defined by two squares centered at the same point with sides in a ratio of 0.4 and rotated 90^o is extruded following a twisted

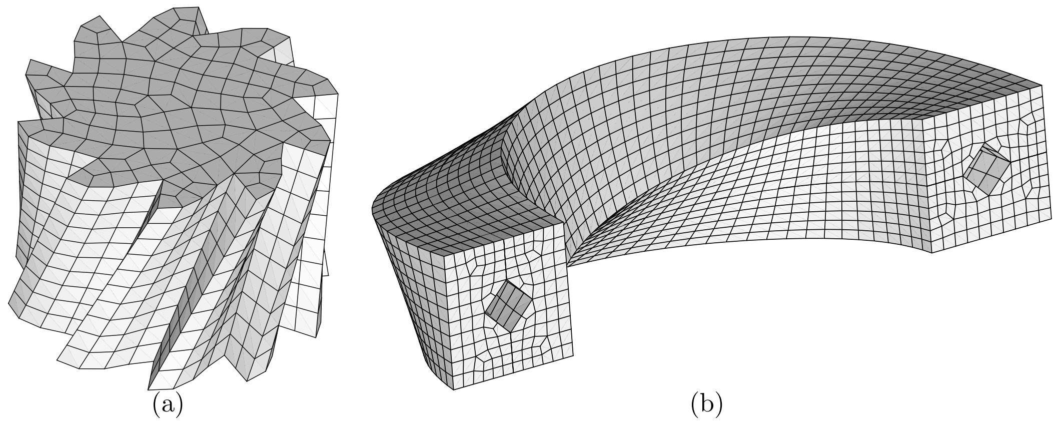


Figure 4.4: Examples of extrusion geometries meshed with the developed sweep algorithm. **(a)** Mill head with twisted sweep path; **(b)** one-hole extrusion volume with twisted and curved sweep path.

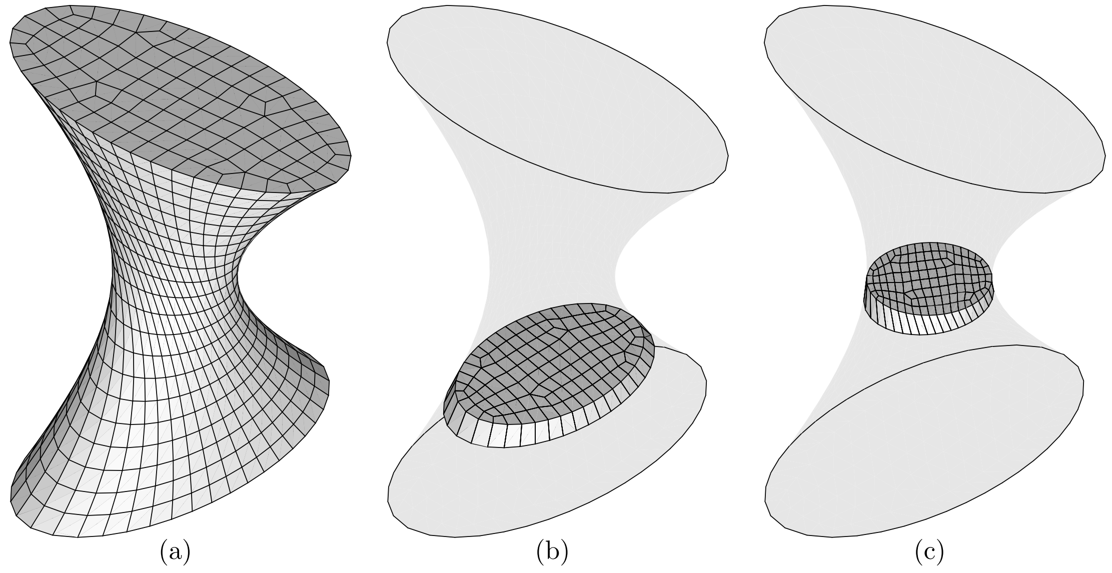


Figure 4.5: Volume with varying elliptical cross-sections along a twisted sweep path. **(a)** Final mesh; **(b)** layer of hexahedral elements at fourth-level; **(c)** middle layer of hexahedral elements.

and curved sweep path. The final mesh is presented in Figure 4.4(b). It is composed by 5104 hexahedral elements. As it can be seen, a non-structured quadrilateral mesh is generated over the cap surfaces. Note that, although the sweeping axis is twisted and curved, high quality elements are generated without a posteriori mesh smoothing.

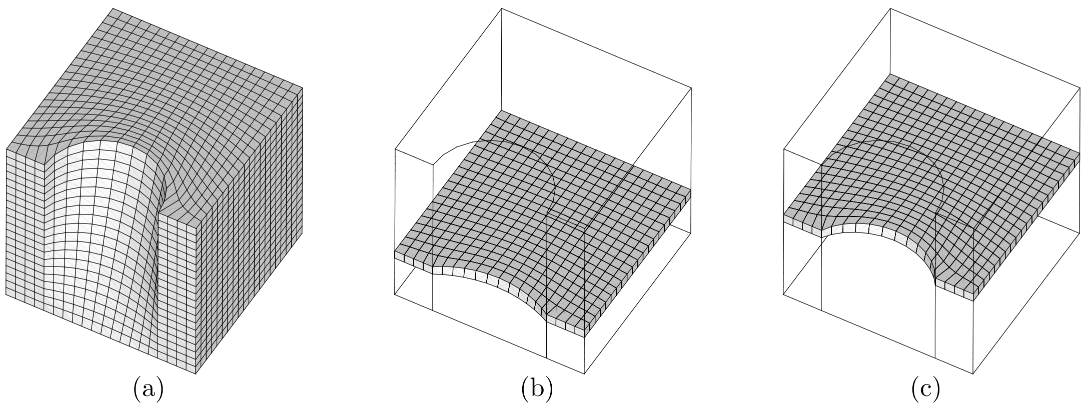


Figure 4.6: Sweep volume with convex source face and a target face with some concavities. **(a)** Whole mesh; **(b)** layer of hexahedral elements at level six; **(c)** layer of hexahedral elements at level eleven.

The third example shows the discretization of a extrusion volume defined by varying and rotating cross-sections along the sweep path. These cross-sections are elliptical-shaped with different size, and just on the middle of the extrusion path becomes circular. Moreover, the cap surfaces are rotated 90 degrees. The final mesh is composed by 2373 elements, see figure 4.5(a). It is important to point out that to obtain the final mesh no smoothing was applied on surface and inner nodes. In order to show the quality of elements inside the volume, figure 4.5(b) shows the fourth layer of hexahedral elements and figure 4.5(c) presents the middle layer with one circular bounding loop.

The fourth example shows an application of the developed algorithm to an extrusion volume defined by two non-affine cap surfaces. In this case, a smoothing algorithm had been applied on the target surface mesh. Note that in this example source surface is convex and target surface has concavities. Therefore, folded quadrilateral elements appear near the concavity of the target surface. In order to unfold these quadrilateral elements it is mandatory to smooth the target surface mesh. For this example we have used the smoothing technique presented in Sarrate and Huerta (2004). In figure 4.6(a) the whole mesh, composed by 8000 hexahedral elements, is presented. Two inner layers are showed in figures 4.6(b) and 4.6(c). Although the cap surfaces are not affine, no additional 3D global smoothing algorithm has been required in this case.

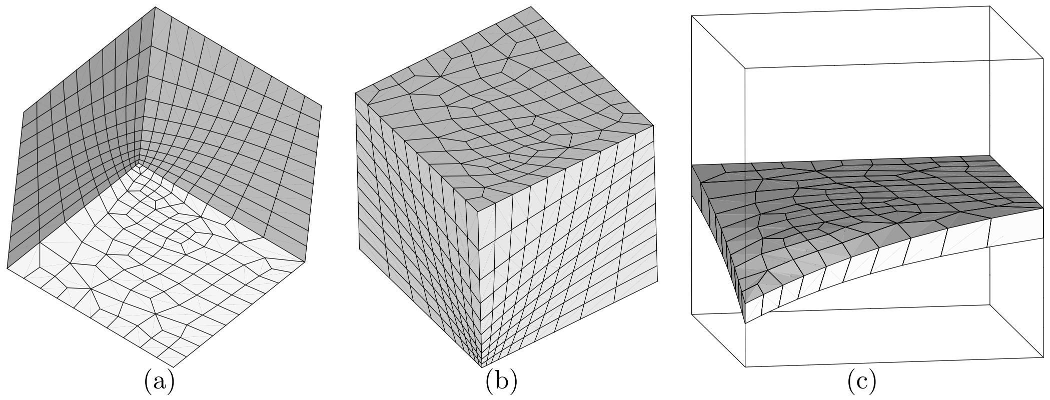


Figure 4.7: Cube with non uniform element size distribution. **(a)** Detail of the final mesh at the corner where high element concentration is prescribed; **(b)** view of target surface mesh; **(c)** inner layer of hexahedral elements.

In the fifth example, the discretization of a cube with a non–constant element size distribution is presented. A high element concentration is prescribed at one corner of the source surface. Hence, the boundary loops of nodes are not mutually affine. Figure 4.7 shows the final mesh composed by 1090 elements. Two smoothing steps were required: the first one to smooth the target surface mesh, and the second one to improve the overall quality of the hexahedral mesh.

4.4.2 Application to geometry decomposition

The aim of this section is to illustrate that the developed algorithm, coupled with volume decomposition, can be successfully used to mesh industrial CAD models. Other issues such as the application to skewed and twisted sweep paths, layers defined by non-affine or non-convex boundaries have been already addressed, in Section 4.4.1.

Power chain. The first example shows the application of the developed algorithm to an extrusion geometry composed by several sweep volumes. Figure 4.8(a) presents the geometry model of a power chain. The final mesh is composed by 33940 hexahedral elements. A detail of the obtained discretization is presented in figure 4.8(b). Note that a conformal mesh is generated over the shared surfaces that compose the pieces of this decomposed geometry.

Crank shaft. Figure 4.9(a) shows the mesh obtained for a crank shaft model. A conformal mesh is generated over the shared surfaces that define this model.

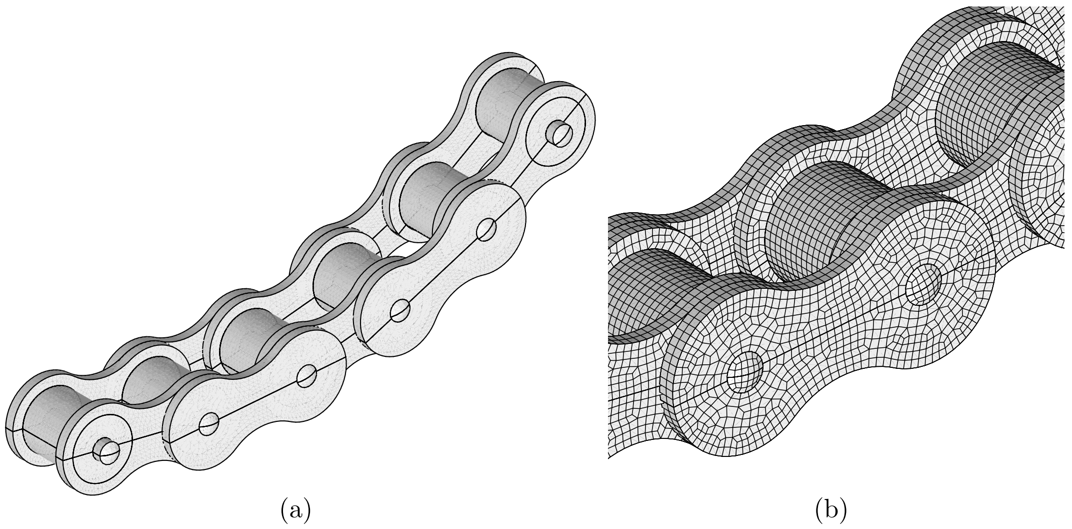

(a) (b)

Figure 4.8: Power chain discretization. **(a)** CAD model block decomposition; **(b)** hexahedral mesh.

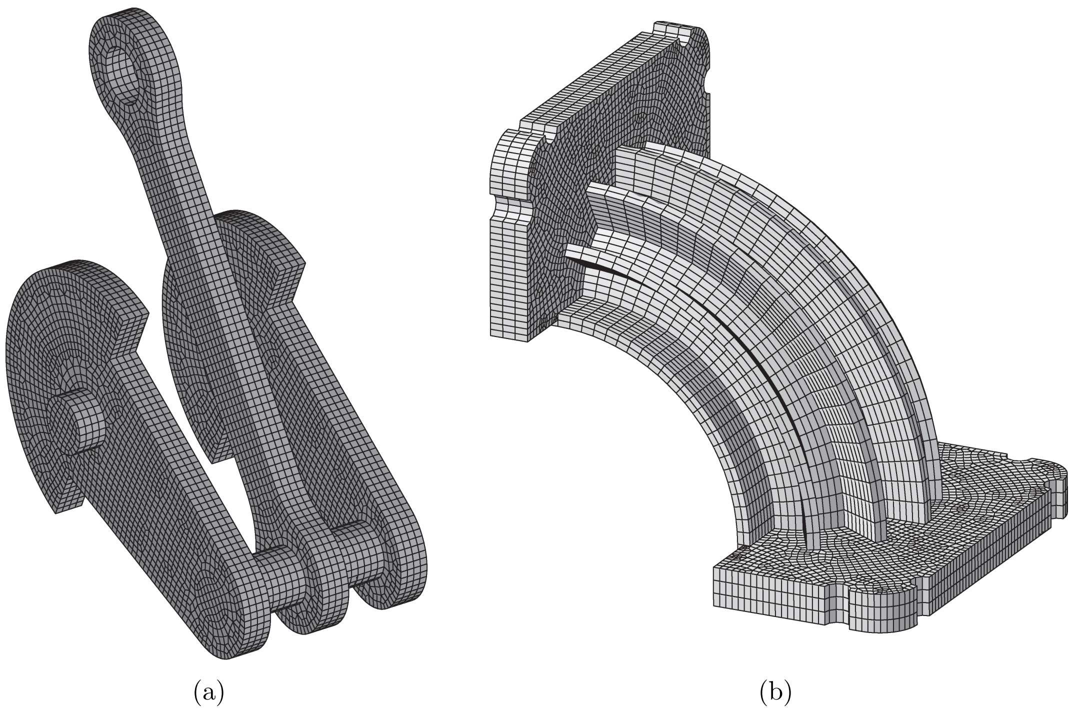

(a) (b)

Figure 4.9: Application of the proposed algorithm to the discretization of extrusion geometries defined by several one-to-one volumes: (a) crank shaft; and (b) heat sink.

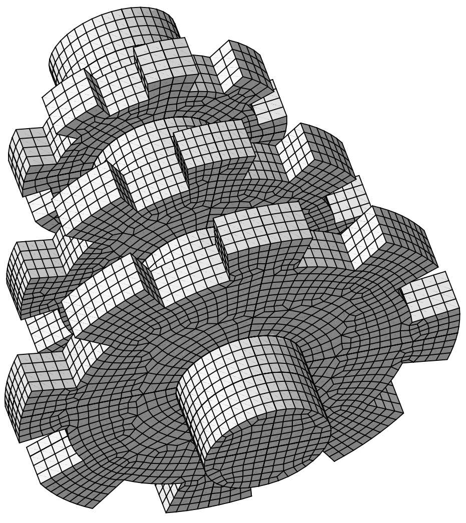


Figure 4.10: Application of the proposed algorithm to the discretization of a gear composed defined by several one-to-one volumes.

Heat sink. Figure 4.9(b) presents the mesh generated for a heat sink. Note that in this example the extrusion path is decomposed in three parts. The first and the last parts are straight whereas the second one has a circular sweep path. In both cases hexahedral elements of high quality are obtained.

Gear. The last example, Figure 4.4.2, shows the mesh generated for a gear model. A conformal mesh is generated over the shared surfaces that define this model.

4.5 Concluding remarks

In this chapter we have detailed the implementation of a sweeping tool based on affine method developed in Chapter 2 and Chapter 3. First, we have presented a new algorithm to project meshes between two topologically equivalent parametric surfaces has been presented. This projection is determined by means of a least-squares approximation of a transformation defined between the loops of boundary nodes of the cap surfaces in the parametric spaces. Once the new mesh is obtained in the parametric space, it is mapped up according the target surface parameterization. Second, we have detailed a new scheme for the generation of inner layers of nodes along the sweep path. To this end, we have proposed to obtain inner layers of nodes by directly projecting from both cap surface meshes. Specifically, the inner layers can be generated in parallel and the projections are obtained with the projection algorithm presented in Chapter 3. The resulting sweeping tool is able to mesh extrusion geometries defined by *(i)* any CAD application; *(ii)* non linear sweeping trajectories; *(iii)* non constant cross section along the sweep axis; *(iv)* non parallel cap surfaces; and *(v)* cap surfaces with different shape and curvature. The examples show that high quality hexahedral elements are generated, and that the layers of inner nodes are distributed in such a way that a smooth transition between the curvatures of cap surfaces is obtained. Moreover, they also illustrate that the developed algorithm, coupled with volume decomposition, can be successfully used to mesh a large class of three dimensional geometries.

Chapter 5

Local dual contributions: representing dual surfaces for block meshing

5.1 Introduction

In the introduction of this dissertation we have mentioned that hexahedral meshes are preferred in a wide range of applications. However, a fully automatic hexahedral mesh generation algorithm for any arbitrary geometry is still not available. In particular, special attention has been focused on the development of algorithms that automatically decompose the geometry into several simpler sub-volumes. This chapter is devoted to obtaining decompositions in block elements of a given domain without respecting a prescribed discretization of the boundary.

The work presented in this chapter has been prompted by the existence of a well-known and straightforward procedure to create unstructured quadrilateral or hexahedral meshes from 2D or 3D triangulations respectively. For instance, we can split each mesh triangle into three quadrilaterals by adding a node at the triangle barycenter and a middle node on each edge. Similarly, each mesh tetrahedron can be split into four hexahedra. However, meshes obtained with this procedure do not provide enough element quality to use them in a numerical simulation. Therefore one question arises: can we modify a mesh obtained with this procedure in order to obtain a high quality hexahedral mesh?

Inspired by the above question, we consider the dual of a quadrilateral or hexahedral mesh obtained by splitting a coarse initial triangulation. Can we "use" this initial dual in order to obtain another dual arrangement that leads to an ultra-coarse quadrilateral or hexahedral mesh? To this end, we propose to select part of the entities of the initial dual to obtain a discretized version of the final dual. This selection process can be interpreted as we were directly inserting dual surfaces. That is, as we were inserting layers of hexahedra to the primal. Then we can dualize this discretized dual to obtain a topological decomposition of the domain in *blocks* (ultra-coarse quadrilaterals or hexahedra).

Unconstrained dual surface insertion algorithms, where both the topology and the geometry of the dual are determined, have only been sketched in the literature. A methodology to represent, intersect and insert *continuous* dual surfaces is not available; see the literature review in Section 1.3 of the introduction. On the contrary, in this chapter we propose to obtain a *discretized* representation of a block mesh dual by insertion of *discretized* dual surfaces. The components of the discretized dual are selected from the initial dual configuration.

Our long-term goal is to use the above paradigm for automatically generating a decomposition in blocks of a given geometry. To this end in this chapter we consider two midterm goals:

1. To develop a tool that generates a representation of the final dual arrangement by using part of the entities composing an initial dual configuration. Moreover, this tool has to deal with two additional responsibilities. First, it has to ensure that dual surfaces *intersect* with proper multiplicity and therefore leading to valid dual representations. Second, it has to facilitate the *insertion* of dual surfaces by only selecting a few leading entities.

2. To automatically select a few leading entities in the initial dual configuration that must be in the final dual configuration. Now we can use this selection as the input for the previous representation tool.

In this chapter we develop the former and we also present an initial approach to the latter. Therefore, our main contribution is to propose a new tool that allows representing, intersecting and inserting discretized dual surfaces based on the local dual contributions concept. Moreover, this tool is used to implement an automatic decomposition tool that deals with blocky geometries. The block-meshing procedure is based in the proposed unconstrained dual approach for obtaining topological decompositions in block elements. Specifically, we use a tetrahedral mesh where we

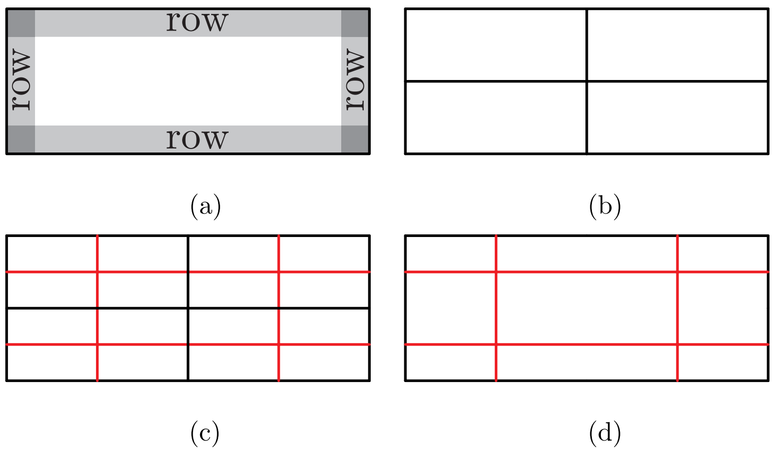


Figure 5.1: Desired dual curves for a rectangular domain: **(a)** required rows of quadrilaterals; **(b)** coarsest quadrilateral mesh; **(c)** connecting opposite edges; and **(d)** desired dual curves.

directly construct the representation of the block mesh dual to finally obtain the primal blocks from it.

5.1.1 Outline

The remainder of this chapter is organized as follows. First, we present a 2D motivation example in Section 5.2. According to this motivation, in Section 5.3 we propose a block meshing approach. To develop this approach we present the required 3D theory in Section 5.4. Specifically, we introduce the concept of local dual contributions. Then, in Section 5.5 we detail how to hierarchically add local dual contributions to represent a valid dual of a hexahedral mesh. The main application of these dual representations is the generation of meshes composed by block elements, see Section 5.6. In Section 5.7 we present several examples that show the capabilities of the proposed block meshing approach.

5.2 2D motivation

To illustrate and clarify the proposed block-meshing paradigm we present a 2D example. Specifically, we consider a rectangular domain to be meshed with ultra-coarse quadrilateral elements. A natural requirement is that high quality elements must be placed near the boundary. To this end we require an independent row of quadrilateral

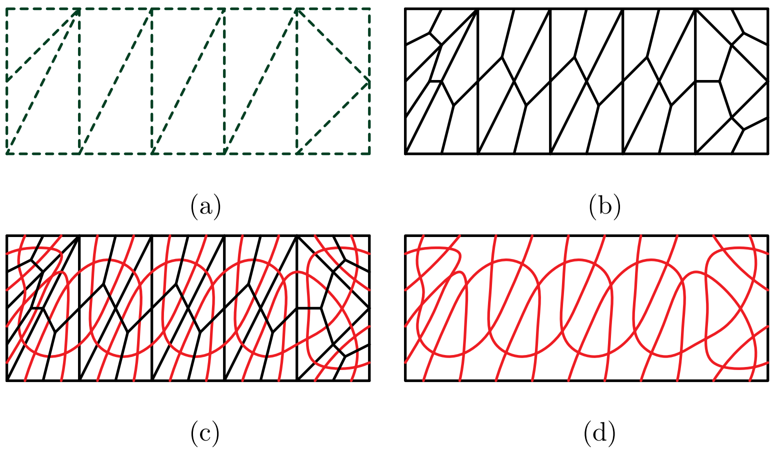


Figure 5.2: Dual curves for a Tri-to-Quad mesh: **(a)** reference mesh; **(b)** Tri-to-Quad mesh; **(c)** connecting opposite edges; and **(d)** dual curves.

elements that follows each boundary curve, see Figure 5.1(a). The coarsest mesh that fulfills this constraint is presented in Figure 5.1(b). Connecting each quadrilateral edge with the opposite one, see Figure 5.1(c), we obtain a set of curves known as the dual curves of the mesh. We realize that requiring a row of elements for each boundary curve of a 2D geometry, see Figure 5.1(a), is equivalent to requiring a dual curve parallel to each boundary curve, see Figure 5.1(b). Note that the configuration in Figure 5.1(d) is the desired configuration of dual curves because it leads to the desired quadrilateral mesh in Figure 5.1(b).

Our goal is to generate an initial dual configuration and use it to obtain the desired one. To create the initial dual configuration we first generate a coarse triangular mesh of the rectangular domain, see Figure 5.2(a), referred as *reference mesh*. We split each triangle in three quadrilaterals to obtain a *Tri-to-Quad* mesh, see Figure 5.2(b). To construct the dual curves of this Tri-to-Quad mesh we add for each element two segments of dual curves connecting opposite quadrilateral edges, see Figure 5.2(c). These dual edges define a set of dual curves that are in correspondence with rows of quadrilateral elements and define the initial dual configuration, see Figure 5.2(d).

Consider the initial dual configuration composed by curved dual entities, see Figure 5.3(a), and add to the foreground the reference mesh triangles, see Figure 5.3(b). With this representation we conclude that each reference mesh triangle is contributing to the dual with three pieces of dual curves, one for each edge, see Figure 5.3(c). Since the dual is a topological representation of the adjacency relations between pri-

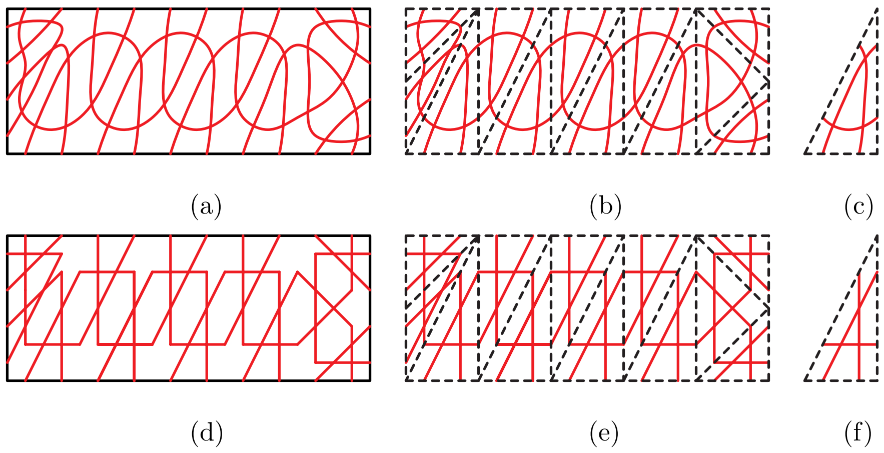


Figure 5.3: Two topologically equivalent representations of dual curves. Curved representation: **(a)** curved dual curves; **(b)** curved dual over the reference mesh; and **(c)** three curved contributions per triangle. Straight representation with poly-lines: **(d)** poly-line dual curves; **(e)** poly-line dual curves over the reference mesh; and **(f)** three segment contributions per triangle.

mal quadrilaterals, we can consider the following representation. Given a reference mesh triangle, we substitute each curved piece of dual curve by a straight segment that will represent a piece of dual curve, see Figure 5.3(f). Doing this substitution to all reference mesh triangles we obtain a dual configuration composed by the straight contributions to the dual of each triangle, see Figure 5.3(e). Specifically, we obtain a dual configuration composed by poly-line dual curves, see Figure 5.3(d), that is topologically equivalent to the initial dual representation, see Figure 5.3(a), and therefore leads to the same quadrilateral mesh.

We have represented with poly-lines the dual curves of the Tri-to-Quad mesh. Now we consider each straight segment of this representation as a potential contributor to the final dual, see Figure 5.4(a). That is, we want to select several of these *candidate* segments to obtain a final dual configuration. This configuration has to be topologically equivalent to the desired one, represented in Figure 5.1(d). As a first approach we can select all those candidate segments that are parallel to the boundary curves, later referred as *hard* contributions. Now we have a set of segments that contribute to the desired dual and lead to an invalid representation with gaps, see Figure 5.4(b). To fill these gaps we add several segments non-parallel to the boundary curves, later referred as *soft* contributions, as in Figure 5.4(c). Thus, we have a better representation without gaps. However, we still have a shortcoming.

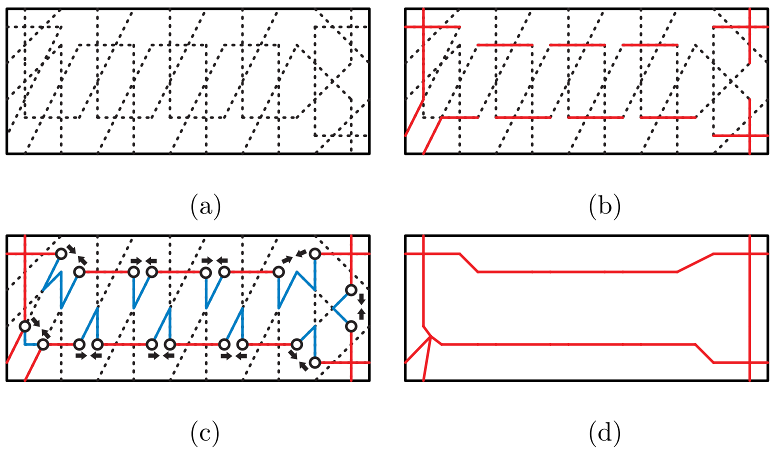


Figure 5.4: Selecting parts of the dual of a tri-quad mesh: **(a)** initial dual curves; **(b)** hard contributions; **(c)** soft contributions; **(d)** final dual curves.

Note that close to the bottom-left corner there are two intersection points, marked with bullets in Figure 5.4(c), with three incident segments. This configuration is not a valid dual representation because the dual of a quadrilateral mesh only has intersection points with four incident segments. To repair these intersection points we consider that each group of adjacent segments added at the second step, the soft contributions, are collapsed into one point, see Figure 5.4(c). As a result we obtain a new dual configuration, see Figure 5.4(d), which is topologically equivalent to the desired one, see Figure 5.1(d).

Finally the block mesh is obtained as the dual of the final dual configuration. To this end we consider the dual regions delimited by the discretized dual curves. Each dual region is in one-to-one correspondence with a primal node, and each change of region delimited by hard pieces of dual curves is in one-to-one correspondence with a primal edge. Therefore we create a primal node inside each dual region and we connect this node with primal nodes in adjacent regions.

5.3 Algorithm proposal

We propose a new approach to obtain a valid topological block decomposition of a given domain without a previous discretization of the boundary. The procedure is structured in three steps described in Algorithm 5.1. In the first step we generate a tetrahedral mesh, see Section 5.4.2. Then this mesh is used to add planar polygons

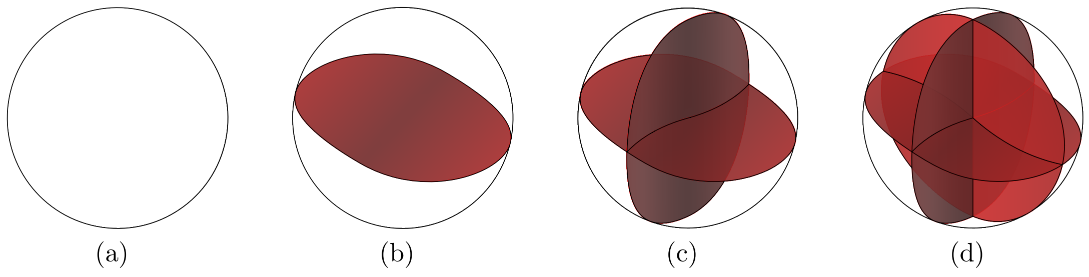


Figure 5.5: Locally valid dual configurations: **(a)** no dual surfaces; **(b)** one dual surface; **(c)** one dual curve; and **(d)** one dual point.

that define a discrete version of the dual surfaces, see Section 5.4.3 for definitions and Section 5.5 for details. This step is equivalent to inserting discrete representations of the dual surfaces. Finally we obtain the block mesh as the dual of the previously generated dual, see Section 5.4.4.

Algorithm 5.1: blockMesh

```
  Input: geometry to block mesh $\mathcal{G}$
  Output: a block mesh $\mathcal{M}$ of $\mathcal{G}$
1 $\mathcal{R} := obtainReferenceMesh(\mathcal{G})$
2 $\mathcal{D} := addLocalDualContributions(\mathcal{R})$
3 $\mathcal{M} := dualize(\mathcal{D})$
```

5.4 3D theory

In this section we formalize and extend to 3D the concepts presented in Section 5.2. Furthermore, the definitions follow the order of appearance in the motivation example.

5.4.1 Desired local dual

A valid hexahedral mesh has to fulfill several topological, geometrical and qualitative requirements in order to be used in a numerical simulation. A detailed exposition of these conditions for hexahedral meshes can be found in Blacker (2001); Tautges (2001). Our goal is to construct a representation of a hexahedral mesh dual that leads to a valid decomposition in blocks. Therefore, the dual representation has also to meet its own set of topological, geometrical and qualitative conditions. To this end, we propose to obtain a dual representation that is locally valid, captures the boundary features and fulfills the boundary constraints. Such a representation is said to be the *desired dual*.

Valid local dual

The full set of topological rules to be met by the dual are detailed in Mitchell (1996). Here we reformulate part of these rules to state that only four local dual configurations are possible. That is, a dual is *locally valid* if for each point there exists a ball that intersects either:

- *no dual entities*, the ball contains a piece of dual region and does not intersect with any dual surfaces, curves or points, see Figure 5.5(a); or
- *one dual surface*, the ball contains a piece of dual surface and does not intersect with any dual curves or points, see Figure 5.5(b); or
- *one dual curve*, the ball contains part of a dual curve that is obtained from the intersection of two dual surfaces and does not intersect with any dual point, see Figure 5.5(c); or
- *one dual point*, the ball contains a dual point that is obtained from the intersection of three dual surfaces, see Figure 5.5(d).

Boundary features

A hexahedral meshing procedure has to preserve the geometric features near the boundary curves and vertices. To this end, we consider that a geometry curve at the boundary is a *feature curve* if the user or an automatic procedure considers that at least one stack of hexahedra is required around it. To automatically detect the features, we consider the angle θ between the normal vectors of the two adjacent surfaces sharing the curve. According to the notation used for paving (Blacker and Stephenson, 1991) and submapping (Whiteley et al., 1996) we consider three types of feature curves:

- *End.* One stack of hexahedra is required. The user determines this requirement or it is automatically detected if θ is approximately $\pi/2$, see Figure 5.6(a).

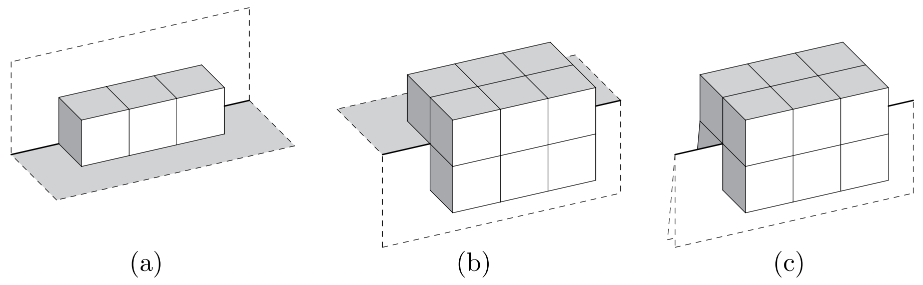


Figure 5.6: Required meshes around feature curves of type: **(a)** end, **(b)** corner, and **(c)** reversal.

- *Corner.* Three stacks of hexahedra are required. The user determines this requirement or it is automatically detected if θ is approximately $3\pi/2$, see Figure 5.6(b).

- *Reversal.* Four stacks of hexahedra are required. The user determines this requirement or it is automatically detected if θ is approximately 2π, see Figure 5.6(c).

Moreover, we consider that a geometry vertex is a *feature vertex* if it is adjacent to three or more feature curves.

Boundary constraints for the dual

Nowadays requirements on geometry and quality of the dual of a hexahedral mesh are less understood than the primal ones. Nevertheless, a detailed survey and analysis of dual constraints associated with hexahedral meshes is presented in Shepherd and Johnson (2008). For the purposes of this chapter we highlight the *boundary constraints*:

1. For each surface of the domain there exists at least one dual surface, with similar shape to the surface, but offset a certain distance. This ensures at least one layer of primal elements parallel to boundary surfaces.

2. For each feature curve of the domain there exists at least one dual curve, with similar shape to the curve, but offset a certain distance. This ensures at least one stack of hexahedra parallel to boundary curves.

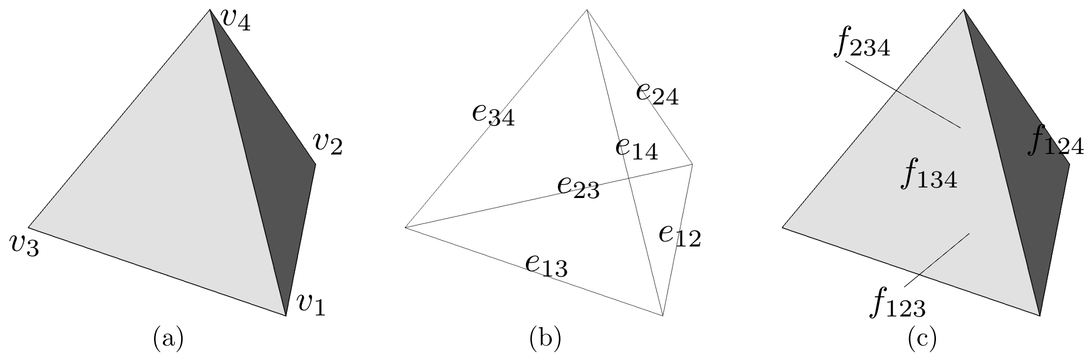


Figure 5.7: Reference element entities: **(a)** vertices; **(b)** edges; **(c)** faces.

3. For each feature vertex of the domain there exists at least one dual point offset a certain distance. This ensures at least one hexahedron on the vertices of the domain.

Note that the offset distance is a function of the desired primal element size.

5.4.2 Reference mesh

To obtain an initial dual configuration we first generate a tetrahedral mesh of the domain referred as *reference mesh*. A tetrahedron of the reference mesh is a *reference element*. Moreover, the vertices, edges and faces of a reference element are denoted by *reference vertices*, *reference edges* and *reference faces* respectively. A reference element t with vertex points v_1, v_2, v_3 and v_4 is denoted by $[v_1, v_2, v_3, v_4]$, see Figure 5.7(a). We univocally denote all reference element edges in t by $e_{ij} := [v_i, v_j]$ where $i < j$ and $i, j = 1, \ldots, 4$. Hence, the six reference edges in t are, see Figure 5.7(b):

$$
\begin{aligned}
e_{12} &:= [v_1, v_2], \quad e_{13} := [v_1, v_3], \quad e_{14} := [v_1, v_4], \\
e_{23} &:= [v_2, v_3], \quad e_{24} := [v_2, v_4], \quad e_{34} := [v_3, v_4].
\end{aligned}
$$

We univocally denote all reference element faces in t by $f_{ijk} := [v_i, v_j, v_k]$ where $i < j < k$ and $i, j, k = 1, \ldots, 4$. The four reference faces in t are, see Figure 5.7(c):

$$
\begin{aligned}
f_{123} &:= [v_1, v_2, v_3], \quad f_{124} := [v_1, v_2, v_4] \\
f_{134} &:= [v_1, v_3, v_4], \quad f_{234} := [v_2, v_3, v_4].
\end{aligned}
$$

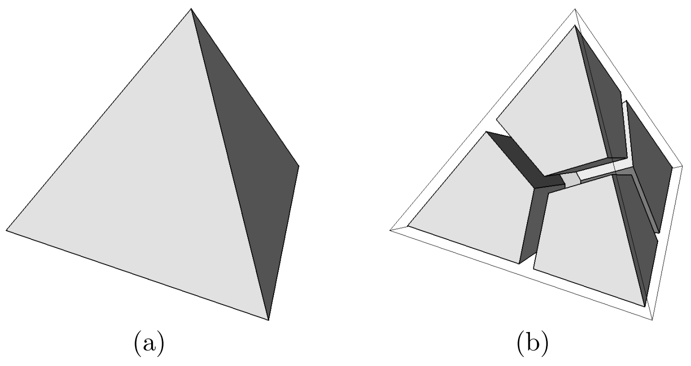


Figure 5.8: Splitting of a tetrahedron: **(a)** tetrahedron and **(b)** decomposition into four hexahedra.

5.4.3 Local dual contributions

The basic idea of the proposed method is to generate a valid dual of a decomposition in blocks by selecting pieces, called candidates, of an initial dual configuration. These candidates are planar polygons and determine a planar representation of the dual on a reference mesh.

Initial dual

From the reference mesh we obtain a hexahedral mesh of the domain by splitting each reference element, see Figure 5.8(a), in four hexahedra, see Figure 5.8(b). To this end we use the procedure *hexing the tet* detailed in Carey (2002). The resulting mesh is said to be a *Tet-to-hex mesh*.

Once we obtain the Tet-to-hex mesh, the *initial dual* configuration is obtained by assembling the dual contained in each split reference element, see Figure 5.9(a). Similar to the 2D case we substitute the part of the dual inside each reference element by four planar surfaces, see Figure 5.9(b). Both representations, curved and planar, are topologically equivalent. These planes are determined by four plane equations expressed in barycentric coordinates $(\alpha, \beta, \gamma, \delta)$:

$$\begin{aligned}
\alpha &= \mu, \text{ (plane opposite to vertex } v_1)\\
\beta &= \mu, \text{ (plane opposite to vertex } v_2)\\
\gamma &= \mu, \text{ (plane opposite to vertex } v_3)\\
\delta &= \mu, \text{ (plane opposite to vertex } v_4),
\end{aligned}$$

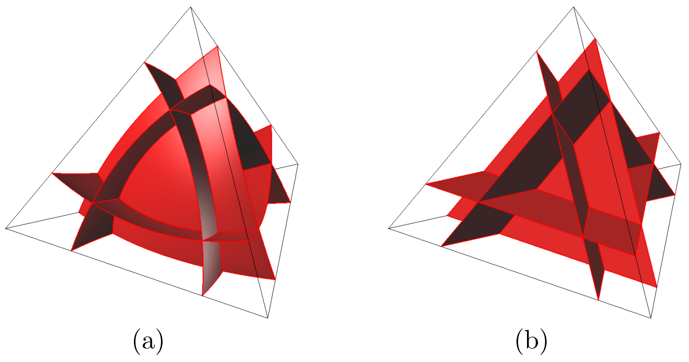


Figure 5.9: Topologically equivalent pieces of the dual in a reference element: **(a)** curved representation and **(b)** planar representation.

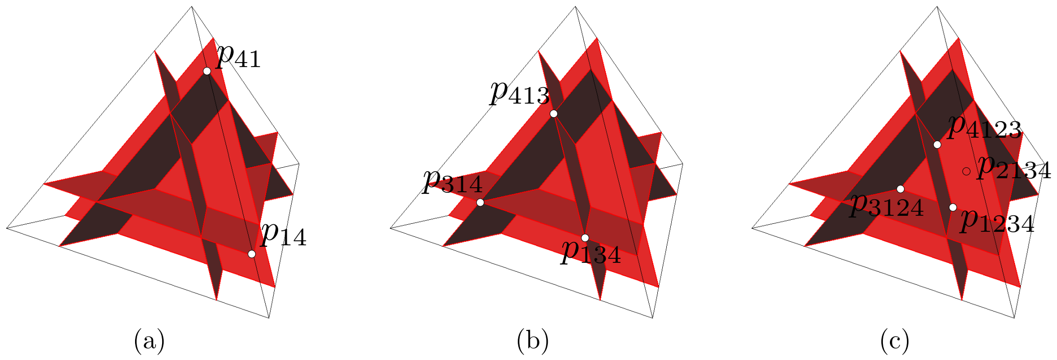


Figure 5.10: Intersection points of a planar dual representation inside a reference element: **(a)** points on edge e_{14}; **(b)** points on face f_{134}; and **(c)** points inside the reference element.

where μ is a given number such that $0 < \mu < \frac{1}{4}$. Note that we have to add to each plane equation the barycentric coordinates condition $\alpha + \beta + \gamma + \delta = 1$ where $\alpha, \beta, \gamma, \delta \geq 0$. Moreover, we fix the value of μ for the whole reference mesh to ensure that planes match properly between different reference elements. Hence, we obtain a topologically equivalent representation of the curved initial dual with planar surfaces. To obtain the figures in this section we have used $\mu = \frac{1}{5}$. In order to highlight local dual contributions from faces we have used $\mu = \frac{1}{10}$ in Section 5.7. The selection of these values is due to aesthetical criteria and does not have influence in the obtained primal mesh.

Definition and classification of candidates

The assembly of four planar surfaces for each reference element composes our initial dual configuration. Note that the location of these surfaces is determined by a fixed number μ. The planar surfaces intersect themselves inside each reference element. Thus, they can be split in 28 planar polygons. Accordingly, the union of planar polygons composes the initial dual surfaces. A planar polygon inside a reference element is a *candidate* to contribute because it can be selected to compose part of the desired final dual configuration. Note that these 28 planar polygons decompose the reference element into 15 sub-volumes.

The candidate polygons inside a reference element $t = [v_1, v_2, v_3, v_4]$ are determined by the intersection points between dual surfaces at reference edges, reference faces, and inside the reference element. For each edge $e_{i,j} = [v_i, v_j]$, where $i < j$ and $i, j = 1, \ldots, 4$, we have two intersection points denoted by p_{ij} and p_{ji}. The first index indicates the closest vertex to the intersection point. For instance, the closest vertex to p_{43} is the vertex v_4. In Figure 5.10(a) we present the two intersection points on edge e_{14}. For each face $[v_i, v_j, v_k]$ we have three intersection points denoted by p_{ijk}, p_{jik} and p_{kij}, where $i < j < k$ and $i, j, k = 1, \ldots, 4$. The first index indicates the closest vertex to the intersection point. Thus, the closest vertex to p_{314} is v_3. In Figure 5.10(b) we present the three points on face f_{134}. Finally, inside a reference element t we have four intersection points denoted by p_{1234}, p_{2134}, p_{3124}, and p_{4123}. The first index indicates the closest vertex to the intersection point. For instance, for p_{2134} we have that v_2 is the closest vertex. Figure 5.10(c) shows all the intersection points inside the reference element. The list of barycentric coordinates for all the intersection points in a reference element is included in Section C.1 of Appendix C.

We classify the 28 candidate planar polygons of a reference element in three categories. First, the *face candidates* are the candidate polygons that only have intersection points inside the reference element, see Figure 5.11(a). For each face f in a reference element t we have an associated face candidate denoted by $\mathbf{c}(f; t)$. In Figure 5.12(a) we present the candidate for face f_{134}. Second, the *edge candidates* are the candidate polygons that touch reference faces but not reference edges, see Figure 5.11(b). For each adjacent face f to an edge e in a reference element t we have an associated edge candidate denoted by $\mathbf{c}(e; f; t)$. The candidate for edge e_{13} and face f_{134} is presented in Figure 5.12(b). And last, the *vertex candidates* are the candidate polygons that touch reference edges, see Figure 5.11(c). For each adjacent edge e to

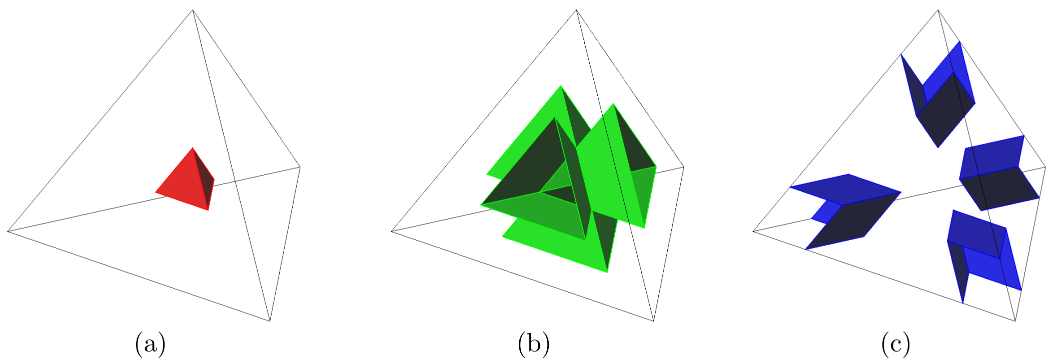


Figure 5.11: Classification of all possible candidates: **(a)** face candidates; **(b)** edge candidates; and **(c)** vertex candidates.

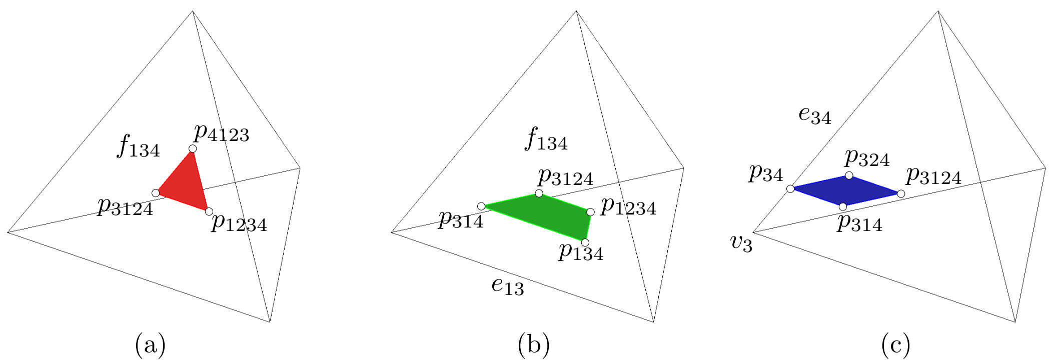


Figure 5.12: A detailed example for each type of contribution: **(a)** face candidate for face f_{1234}; **(b)** edge candidate for edge e_{13} and face f_{134}; and **(c)** vertex candidate for vertex v_3 and edge e_{34}.

a vertex v in a reference element t we have an associated vertex candidate denoted by $\mathbf{c}(v;e;t)$. Figure 5.12(b) shows the candidate for vertex v_3 and edge e_{34}. A list of the 28 candidate polygons for a reference element, with intersection points as vertices, is included in Section C.1.1 of Appendix C.

Definition and classification of local dual contributions

We have of three types of candidate polygons that can be used to contribute to the final dual configuration. A candidate polygon is considered a *local dual contribution* if it is selected to contribute to the final dual surface. To represent a valid desired dual we have to ensure that local dual contributions capture required local features of the desired dual, do not present gaps between them, and match properly. To this end we consider two types of local dual contribution. The first one is denoted by *hard* local dual contribution. This type is used to enforce the required local configurations of the desired dual. Hard local dual contributions are the selected candidates that have to be respected to obtain the desired primal mesh. The second type is called *soft* local dual contribution. It is used to fill gaps and to match properly the local dual contributions between them. Soft local dual contributions have to be collapsed, and therefore ignored, to obtain the primal mesh.

5.4.4 Obtaining the final block mesh

Once we have a valid representation of the dual we can obtain the block mesh as the dual of this dual. That is, we generate: a primal node for each dual region delimited by dual surfaces; a primal edge connecting primal nodes generated from adjacent dual regions; a primal quadrilateral for each piece of dual curve between two dual points; and a hexahedron for each dual point. However, we have to slightly modify this procedure to take into account hard and soft local dual contributions. By definition, hard contributions have to be respected and soft contributions have to be collapsed into one dual point. According to these definitions, we only generate primal edges between adjacent dual regions separated by at least one local dual contribution of type hard. On the contrary, we do not generate a primal edge between adjacent dual regions that are separated only by local dual contribution of type soft. The last is equivalent to collapsing each soft local dual contribution in one point. Then we can create the quadrilateral faces and finally the hexahedral elements that determine the blocks.

5.5 Adding local dual contributions

This section details how to add local dual contributions on a reference mesh to obtain the desired dual configuration. To this end, we propose a hierarchical scheme that takes into account a set of matching rules. These rules ensure that adjacent local dual contributions match properly and define locally valid dual configurations. Moreover, these rules depend on three functions that classify the reference entities. These functions enforce that the boundary constraints are fulfilled.

The call graph for the addition of local dual contributions is presented in Table 5.5. Algorithms on the left call algorithms on their right. This is also the order of presentation for the algorithms along this section. Hence, we present the process of adding local dual contributions going from less detail to more detail.

Depth 1		Depth 2		Depth 3		Depth 4
Algorithm 5.2	$\rightarrow$	Algorithm 5.3	$\rightarrow$	Algorithm 5.6	$\rightarrow$	Algorithm 5.9
	$\rightarrow$	Algorithm 5.4	$\rightarrow$	Algorithm 5.7	$\rightarrow$	Algorithm 5.10
	$\rightarrow$	Algorithm 5.5	$\rightarrow$	Algorithm 5.8	$\rightarrow$	Algorithm 5.11

Table 5.1: Call graph for adding local dual contributions.

5.5.1 Hierarchical scheme

We propose a hierarchical scheme to simplify the task of obtaining a valid dual by addition of local dual contributions. This scheme is detailed in Algorithm 5.2 and it is performed in three steps. First, we add local dual contributions from faces. Second, we add local dual contributions from edges taking into account the face candidates previously added. Third, we add local dual contributions from vertices taking into account the edge candidates added in the previous step. Figure 5.13 depicts these three steps for an L-shaped geometry.

Algorithm 5.2: addLocalDualContributions

```
Input: reference mesh $\mathcal{R}$
Output: dual represented by local dual contributions $\mathcal{C}_f$, $\mathcal{C}_e$ and $\mathcal{C}_v$
1 $\mathcal{C}_f := addLocalDualContributionsFromReferenceFaces(\mathcal{R})$
2 $\mathcal{C}_e := addLocalDualContributionsFromReferenceEdges(\mathcal{R}, \mathcal{C}_f)$
3 $\mathcal{C}_v := addLocalDualContributionsFromReferenceVertices(\mathcal{R}, \mathcal{C}_e)$
```

Add local dual contributions from reference faces. Algorithm 5.3 describes how to add local dual contributions from faces. Specifically, for each face f inside a reference element t the procedure checks if the face candidate $\mathbf{c}(f;t)$ has to be added. This checking is performed by the matching rule for faces described later in Algorithm 5.6. Figure 5.13(a) shows that several face candidates (triangles) are added as hard local dual contributions (red). These hard local dual contributions are determining pieces of the desired dual surfaces. Note that there are gaps between these contributions.

Add local dual contributions from reference edges. Algorithm 5.4 takes into account face candidates previously added to add local dual contributions from

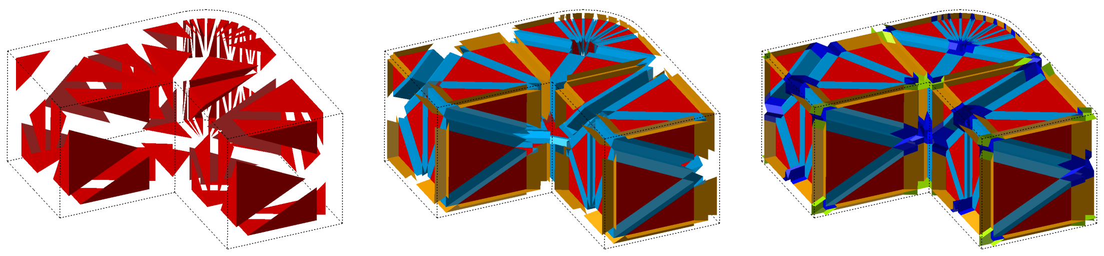


Figure 5.13: Hierarchical scheme for adding local dual contributions from: **(a)** faces (red); **(b)** edges (cyan and orange); and **(c)** vertices (blue and yellow).

Algorithm 5.3: addLocalDualContributionsFromReferenceFaces

```
  Input: reference mesh $\mathcal{R}$
  Output: local dual contributions from faces $\mathcal{C}_f$
1 foreach reference element $t$ do
2     foreach reference face $f$ in $t$ do
3         $type := matchingRuleForFace(f, t)$
4         if $type \neq none$ then add $\mathbf{c}(f;t)$ as $type$ in $\mathcal{C}_f$
```

edges. That is, given an edge e and an adjacent face f the procedure decides if the edge candidate $\mathbf{c}(e;f;t)$ has to be considered as part of the final dual. This decision is performed by the matching rule for edges, see Algorithm 5.7 presented later. For instance, Figure 5.13(b) shows that several edge candidates (long quadrilaterals) are added as soft local dual contributions (cyan) to fill the gaps between the added face candidates. Moreover, several edge candidates are added as hard local dual contributions (orange). These hard contributions determine locally the dual configuration for obtaining the dual curves. Again we still have gaps between added local dual contributions.

Algorithm 5.4: addLocalDualContributionsFromReferenceEdges

```
  Input: reference mesh $\mathcal{R}$
  Input: local dual contributions from faces $\mathcal{C}_f$
  Output: local dual contributions from edges $\mathcal{C}_e$
1 foreach reference element $t$ do
2     foreach reference edge $e$ in $t$ do
3         foreach adjacent reference face $f$ to $e$ in $t$ do
4             $type := matchingRuleForEdge(e, f, t, \mathcal{C}_f)$
5             if $type \neq none$ then add $\mathbf{c}(e;f;t)$ as $type$ in $\mathcal{C}_e$
```

Add local dual contributions from reference vertices. Taking into account previously added local dual contributions from edges, Algorithm 5.5 adds the required vertex candidates as local dual contributions. That is, given a vertex v and an adjacent face e we have to decide if the vertex candidate $\mathbf{c}(v;e;t)$ has to be considered as part of the final dual. To this end we use the matching rule for vertices detailed in Algorithm 5.8. Figure 5.13(c) shows that vertex candidates (small quadrilaterals) are added as soft local dual contributions (blue) to fill the gaps between the added face and edge candidates. Furthermore, several vertex candidates are added as hard local dual contributions (yellow). These hard contributions determine the proper local dual configuration for representing a dual point. At the end of this step there are not gaps between local dual contributions.

Algorithm 5.5: addLocalDualContributionsFromVertices

```
  Input: reference mesh $\mathcal{R}$
  Input: local dual contributions from edges $\mathcal{C}_e$
  Output: local dual contributions from vertices $\mathcal{C}_v$
1 foreach reference element $t$ do
2     foreach reference vertex $v$ in $t$ do
3         foreach adjacent reference edge $e$ to $v$ in $t$ do
4             $type := matchingRuleForVertex(v, e, t, \mathcal{C}_e)$
5             if $type \neq none$ then add $\mathbf{c}(v;e;t)$ as $type$ in $\mathcal{C}_v$
```

5.5.2 Classification of reference entities

To state the process of adding local dual contributions we have to classify all reference entities, both boundary and inner reference entities. We classify reference entities according to the local dual configuration to be captured around them. Specifically, using the functions presented in Section 5.5.4 a reference entity is classified as:

- *Free*, if dual surfaces, dual curves and dual points are not required around the reference entity. This type applies to reference faces, edges and vertices.

- *Layer*, if only dual surfaces are required around the reference entity. Note that dual curves and dual points are not required. This type applies to reference faces, edges and vertices. To respect the first boundary constraint presented in Section 5.4.1, all boundary faces have to be considered of this type.

- *Stack*, if only dual curves are required around the reference entity. Note that dual points are not required. This type only applies to reference edges and vertices. All the feature edges require at least one stack of hexahedra. Therefore they are also of type stack. This ensures the second boundary constraint presented in Section 5.4.1.
- *Hexahedron*, if at least one dual point is required around the reference entity. This type applies only to reference vertices. All the feature vertices require at least one hexahedron. Thus, they are also of type hexahedron. This ensures the third boundary constraint presented in Section 5.4.1.

Note that this classification does not take into account the number of dual entities required around a reference entity. In Section 5.5.4 we detail our implementation for the classification of reference: faces (Algorithm 5.9), edges (Algorithm 5.10) and vertices (Algorithm 5.11).

5.5.3 Matching rules

To ensure that we obtain the desired dual configuration we consider a set of matching rules. On the one hand, matching rules ensure that added candidates match properly and represent a dual without gaps. To this end, local dual contributions of type soft are added. On the other hand, matching rules enforce the desired dual configuration by adding hard local dual contributions.

Since the addition process is performed in a hierarchical manner there are three matching rules. That is, there are matching rules for adding local dual contributions from reference faces, edges and vertices. These rules are based on the classification of the reference entities and the previously added local dual contributions.

Matching rule for faces. Given a reference face f and its classification we decide to add local dual contributions from this face according to the rule described in Algorithm 5.6.

Matching rule for edges. Given an edge e and an adjacent face f we have to decide if the edge candidate $\mathbf{c}(e; f; t)$ has to be considered as part of the final dual. Moreover, we have to ensure that the addition of the candidate $\mathbf{c}(e; f; t)$ matches properly with previously added local dual contributions from faces. To this end we consider a matching rule determined by the classification of edge e and the type of local dual contribution added from face f, see Algorithm 5.7.

Algorithm 5.6: matchingRuleForFace

```
Input: reference face $f$, element $t$
Output: type of local dual contribution to add, type
1 switch $classify(f)$ do
2     case free
3         type:=none
4     case layer
5         type:=hard
```

Algorithm 5.7: matchingRuleForEdge

```
Input: reference edge $e$, face $f$ and element $t$
Input: local dual contributions from faces $\mathcal{C}_f$
Output: type of local dual contribution to add, type
1 switch $classify(e)$ do
2     case free
3         type:=none
4     case layer
5         if $\mathbf{c}(f;t)$ is not in $\mathcal{C}_f$ then type:=soft else type:=none
6     case stack
7         if $\mathbf{c}(f;t)$ is not in $\mathcal{C}_f$ then type:=soft else type:=hard
```

Matching rule for vertices. Given a vertex v and an adjacent face e we have to decide if the vertex candidate $\mathbf{c}(v;e;t)$ has to be considered as part of the final dual. Moreover, we have to ensure that the addition of the candidate $\mathbf{c}(v;e;t)$ matches properly with previously added local dual contributions from faces. Algorithm 5.8 specifies this matching rule according to the classification of vertex v and the type of local dual contributions added from adjacent edge e.

5.5.4 Classification functions for reference entities

According to the proposed hierarchical scheme, the classification functions for reference entities are implemented also in a hierarchical manner. Specifically, the classification function for vertices depends on the classification function for edges. Similarly, the classification of edges depends on the classification function for faces. The latter depends on an initial list $\mathcal{L}$ of reference faces. This list contains the inner reference faces to be considered of type layer. That is, these layer faces determine the inner cuts of the decomposition process, see details in Section 5.6.1.

Algorithm 5.8: matchingRuleForVertex

```
Input: reference vertex $v$, edge $e$ and element $t$
Input: local dual contributions from edges $\mathcal{C}_e$
Output: type of local dual contribution to add, $type$
{$f_1, f_2$}:=$adjacentFaces(e)$
$\mathbf{c}_1$:=$\mathbf{c}(e; f_1; t)$
$\mathbf{c}_2$:=$\mathbf{c}(e; f_2; t)$
switch $classify(v)$ do
    case $free$
        $type$:=none
    case $layer$
        if $\mathbf{c}_1$ is not in $\mathcal{C}_e$ and $\mathbf{c}_2$ is not in $\mathcal{C}_e$ then
            $type$:=soft
        else
            $type$:=none
    case $stack$
        if $\mathbf{c}_1$ is not in $\mathcal{C}_e$ and $\mathbf{c}_2$ is not in $\mathcal{C}_e$ then
            $type$:=soft
        else if neither $\mathbf{c}_1$ is hard nor $\mathbf{c}_2$ is hard then
            $type$:=soft
        else
            $type$:=none
    case $hexahedron$
        if $\mathbf{c}_1$ is not in $\mathcal{C}_e$ and $\mathbf{c}_2$ is not in $\mathcal{C}_e$ then
            $type$:=soft
        else if neither $\mathbf{c}_1$ is hard nor $\mathbf{c}_2$ is hard then
            $type$:=soft
        else
            $type$:=hard
```

Classification of faces. The implementation details for classifying reference faces are presented in Algorithm 5.9. Note that this function classifies as layer those faces being on a geometry surface. That is, the reference faces at the boundary are determining that at least one layer of hexahedra has to be generated parallel to boundary surfaces. Therefore, this function enforces the first boundary constraint. The rest of the dual surfaces are determined by the election of the initial list $\mathcal{L}$. Moreover, this list determines the classification of reference edges and vertices.

Classification of edges. Algorithm 5.10 presents the implementation for classifying reference edges. This function classifies as stack those edges being on a feature

Algorithm 5.9: classify

```
   Input: reference face $f$
   Input: a list of inner reference faces to be considered as layer faces $\mathcal{L}$
   Output: classification of $f$, $classification$
 1 if $f$ is on a geometry surface or in $\mathcal{L}$ then
 2     $classification$:=layer
 3 else
 4     $classification$:=free
```

curve. These stack edges are determining that at least one stack of hexahedra has to be generated parallel to the boundary curves. Hence, this function ensures the second boundary constraint. The rest of the classification is done according to the number of adjacent faces of type layer, see line 4. Note that to obtain this number we have to call the classification function for faces.

Algorithm 5.10: classify

```
   Input: reference edge $e$
   Output: classification of $e$, $classification$
 1 if $e$ is on a feature curve then
 2     $classification$:=stack
 3 else
 4     $nOfAdjacentLayerFaces$:=number of adjacent faces of type layer
 5     switch do
 6         case $nOfAdjacentLayerFaces = 0$
 7             $classification$:=free
 8         case $0 < nOfAdjacentLayerFaces < 3$
 9             $classification$:=layer
10         case $3 \leq nOfAdjacentLayerFaces$
11             $classification$:=stack
```

Classification of vertices. The implementation of the classification function for vertices is detailed in Algorithm 5.11. This implementation classifies reference vertices coincident with geometry vertices as being of hexahedron type. That is, these vertices are determining that at least one hexahedron has to be generated close to the boundary vertices. Therefore, this function ensures the third boundary constraint. The rest of the classification is done according to the number of adjacent edges of type layer and stack, see lines 4 and 5 respectively. Note that to obtain these numbers we have to call the classification function for edges.

Algorithm 5.11: classify

```
Input: reference vertex $v$
Output: classification of $v$, $classification$
if $v$ is on a feature vertex then
    $classification$:=hexahedron
else
    $nOfAdjacentLayerEdges$:=number of adjacent edges of type layer
    $nOfAdjacentStackEdges$:=number of adjacent edges of type stack
    switch do
        case $nOfAdjacentStackEdges = 0$
            if $nOfAdjacentLayerEdges = 0$ then
                $classification$:=free
            else
                $classification$:=layer
        case $0 < nOfAdjacentStackEdges < 3$
            $classification$:=stack
        case $3 \leq nOfAdjacentStackEdges$
            $classification$:=hexahedron
```

5.6 Application to block-meshing

The addition of local dual contributions is performed according to the classification functions for faces, edges and vertices. These functions depend on an initial list $\mathcal{L}$ of inner reference faces. This list determines the inner reference faces that are considered to be of layer type. Hence, to automatically generate block meshes with the proposed algorithm, see Section 5.3, we have to automatically select the layer faces. To illustrate this, Algorithm 5.12 details a simple procedure to mark reference faces as layers based on geometrical criteria. Note that we detail an algorithm that does not deal with reversal curves because the logic is simpler. However, Algorithm 5.12 can be extended to decompose blocky geometries with reversal curves.

5.6.1 Marking faces to determine required layers

The process described in Algorithm 5.12 is performed in three stages:

Creating lists and maps. From lines 1 to 3, the procedure creates an empty list of faces $\mathcal{L}$ and two empty maps $\mathcal{E}_c$ and $\mathcal{E}_n$. The list $\mathcal{L}$ is used during the execution to mark the faces as layers. Both structures, $\mathcal{E}_c$ and $\mathcal{E}_n$, are used to map an indexed edge to a list of adjacent faces. Specifically, $\mathcal{E}_c$ and $\mathcal{E}_n$ store the current and the new

Algorithm 5.12: Mark layers

```
   Input: reference mesh $\mathcal{R}$
   Output: faces marked as layers $\mathcal{L}$
1  $\mathcal{L} := \emptyset$
2  $\mathcal{E}_c := \emptyset$
3  $\mathcal{E}_n := \emptyset$
4  foreach feature edge e in $\mathcal{R}$ do
5      if e is on a corner or touches a vertex with valence $> 3$ then
6          add to $\mathcal{E}_c[e]$ all boundary faces adjacent to $e$
7  $oldNOfLayers := 0$
8  $newNOfLayers := 1$
9  while $oldNOfLayers < newNOfLayers$ do
10     $oldNOfLayers :=$ size of $\mathcal{L}$
11     foreach edge e indexed in $\mathcal{E}_c$ do
12         foreach face f in list $\mathcal{E}_c[e]$ do
13             $candidates$ :=list of inner faces adjacent to $e$ and different to $f$
14             $\mathbf{n}$ :=direction of normal vector to $f$
15             $layer$ :=select in $candidates$ a face with a normal direction similar
               to $\mathbf{n}$
16             add to $\mathcal{L}$ face $layer$
17             foreach edge $e_f \neq e$ in face layer do
18                 if $e_f$ is inner edge then add $layer$ to $\mathcal{E}_n[e_f]$
19     $newNOfLayers :=$ size of $\mathcal{L}$
20     $\mathcal{E}_c := \mathcal{E}_n$
21     clear $\mathcal{E}_n$
```

fronts respectively.

Creating initial fronts. From lines 4 to 6, the algorithm indexes the boundary edges that are on a corner or touch a vertex with valence greater than three. For each indexed edge e its adjacent boundary faces are stored in list $\mathcal{E}_c[e]$. Thus, the initial front is determined by indexed boundary edges and their adjacent boundary faces stored in $\mathcal{E}_c$.

Advancing fronts. From lines 9 to 21, new faces are marked as layers in an advancing front manner. The fronts are advanced using the inner faces adjacent to the indexed edges in $\mathcal{E}_c$. Those inner faces that have similar normal directions to the faces stored in indexed lists of $\mathcal{E}_c$ are marked as layers. These new layer faces and its inner edges not in $\mathcal{E}_c$ determine a new front stored in $\mathcal{E}_n$. The fronts are advanced while new faces are marked as layers. That is, the procedure stops when all fronts

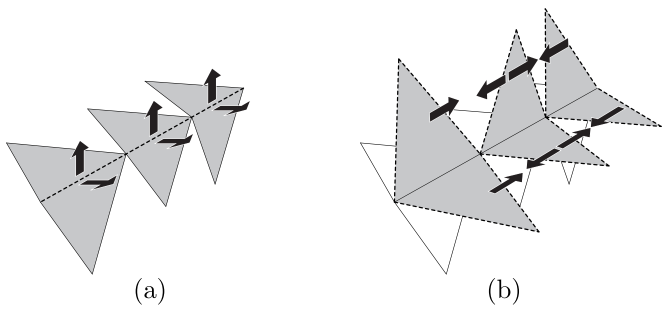


Figure 5.14: Advancing fronts of edges and faces from a corner curve: **(a)** initial front; **(b)** new front at first iteration; and **(c)** new front at second iteration.

reach again the boundaries.

To illustrate this advancing front procedure, in Figure 5.14 we present the three first steps around a corner curve. In Figure 5.14(a), each one of the initial front edges, dashed lines, has a list of two adjacent boundary faces, grey triangles. The arrows determine the preferred advancing directions. Figure 5.14(b) shows the optimal faces, grey triangles, adjacent to initial front edges and the new front edges, dashed lines. These new faces and edges compose the new front. The next iteration of the advancing front loop is presented in Figure 5.14(c). A new front is determined by new optimal inner faces, grey triangles, and inner edges not in the previous front, dashed lines.

5.6.2 Dealing with high-valence vertices

It is well known that discretizing with hexahedra the region around a feature vertex with valence greater than three is a difficult task. However, it is straightforward to automatically discretize these regions with tetrahedra. This is one of the advantages of using a tetrahedral mesh as a reference mesh for adding local dual contributions. Note that all nodes in a tetrahedron and in a hexahedron are 3-valent. Therefore, for each feature vertex with valence greater than three the reference mesh is naturally decomposed in 3-valent vertices. Therefore, we propose to add to the initial front those edges that touch a vertex with valence greater than three, see Algorithm 5.12. Then, the front advances through the reference mesh and the obtained layer faces naturally divide the feature vertex in several 3-valent vertices.

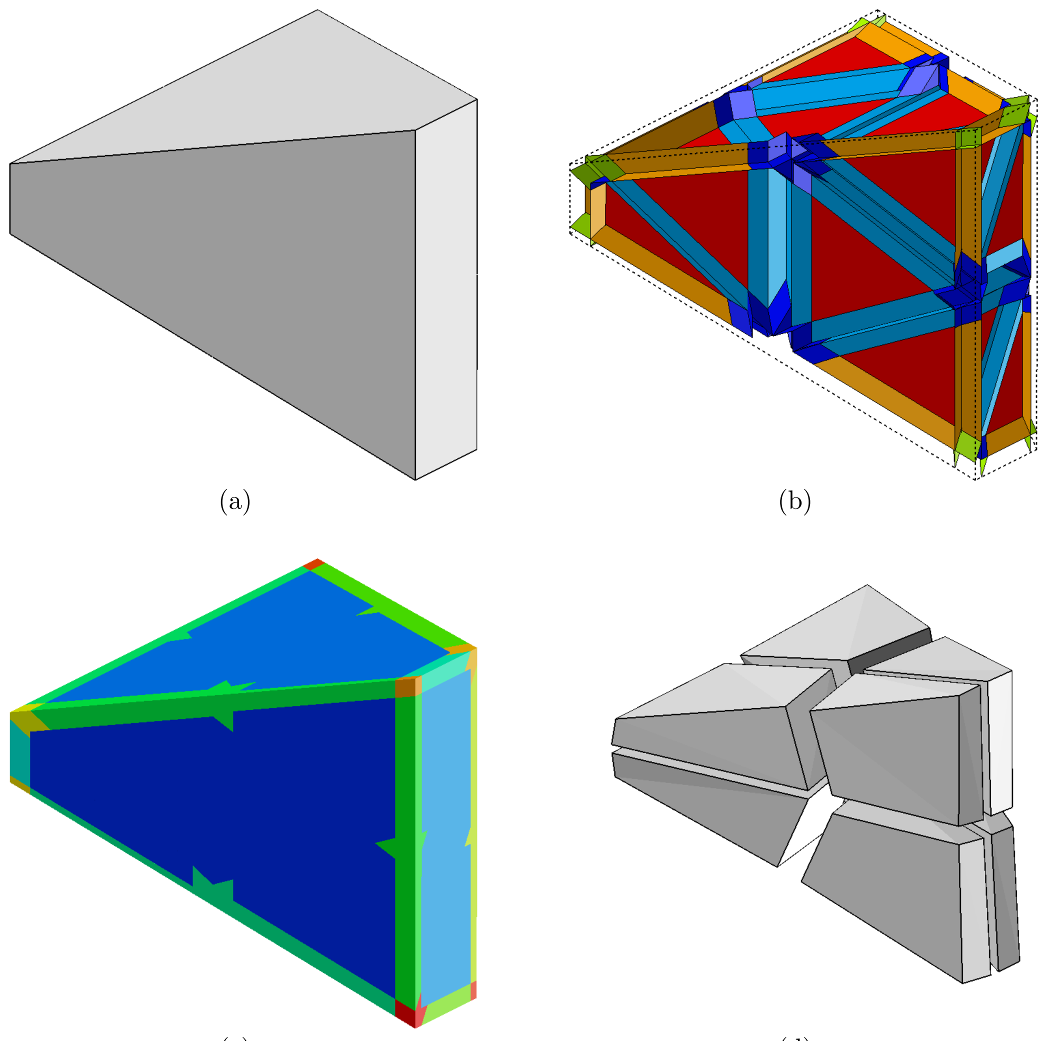
5.7 Examples

In this section we present several examples to illustrate the capabilities of the proposed approach. The local dual contributions allow capturing the dual of the desired block mesh for a wide range of convex and non-convex domains. The examples present geometric features including planar surfaces, curved surfaces, vertices with valence greater than three, thin configurations, and holes.

We use our implementation to decompose these geometries in coarse blocks. Notice that the proposed algorithm focuses on the generation of valid topological decomposition in blocks. Then, the final node location can be improved using a relaxation procedure. In addition, a finer mesh can be obtained by meshing each block separately while keeping the compatibility of the mesh between them.

For the first three examples, we present four figures showing the geometry to decompose, the representation of the dual surfaces with local dual contributions, the dual regions delimited by the dual surfaces, and the obtained block mesh. For the last four examples, which contain a large number of local dual contributions, we only include the dual regions and the associated block mesh. In all the examples, the local dual contributions are colored according to their type: hard face local dual contributions (triangles) are colored in red; edge candidates (long quadrilaterals) are colored in orange and cyan to indicate hard and soft local dual contributions respectively; and vertex candidates (small quadrilaterals) are colored in yellow and blue to indicate hard and soft local dual contributions respectively. The dual regions are also depicted in different colors: each colored region is associated with a different primal node of the block mesh. Furthermore, for each example we include a table with the number of mesh nodes and elements that compose the reference mesh, the dual surfaces represented with local dual contributions, the dual regions, and the final block mesh. Note that the number of polygons that define the dual surfaces is at most 28 times the number of reference elements. On the contrary, the number of polyhedra that determine the dual regions is always 15 times the number of reference elements.

Brick-shaped domain. This geometry is a convex domain that is tapered in two orthogonal directions, see Figure 5.15(a). All the dual surfaces follow the boundary of the domain and the possible gaps between local dual contributions are filled with soft local dual contributions, see Figure 5.15(b). Hard local dual contributions from edges and vertices determine the desired mesh close to feature curves and vertices.



(a) (b) (c) (d)

Figure 5.15: Block-meshing process for the brick-shaped domain: **(a)** domain; **(b)** local dual contributions; **(c)** dual regions; and **(d)** unstructured block mesh.

Resulting local dual contributions are a discrete representation of the dual surfaces and divide the domain in several dual regions represented in different colors in Figure 5.15(c). Each one of the dual regions corresponds to a primal node of the final block mesh, see Figure 5.15(d). In this example, all the vertices are 3-valent. Hence, it can be meshed with submapping or midpoint subdivision. Specifically, the obtained decomposition is equivalent to a coarse mesh obtained with these methods. Table 5.2 shows the number of nodes and elements that compose the meshes used to obtain the final block mesh.

	Nodes	Elements	Type
Reference mesh	16	21	tetrahedra
Dual surfaces	336	406	polygons
Dual regions	368	315	polyhedra
Block mesh	27	8	blocks

Table 5.2: Mesh entities used for the brick-shaped domain.

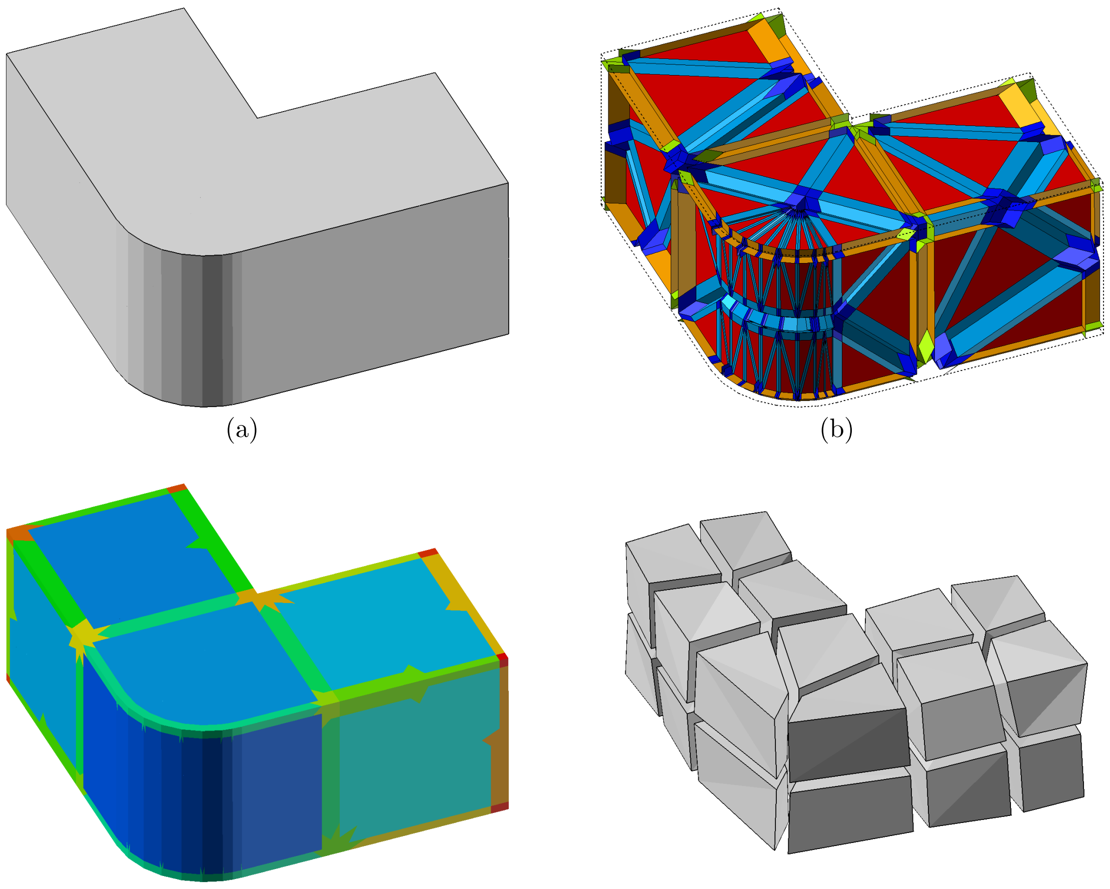


Figure 5.16: Block-meshing for the L-shaped domain: **(a)** domain; **(b)** local dual contributions; **(c)** dual regions; and **(d)** unstructured block mesh.

L-shaped domain. This geometry is non-convex and presents a curved surface determined by a rounding, see Figure 5.16(a). Our meshing tool automatically adds local dual contributions that follow the surface boundaries and that cut the domain along the corner curve, see Figure 5.16(b). These local dual contributions divide the domain in several dual regions represented in different colors in Figure 5.16(c). Each

	Nodes	Elements	Type
Reference mesh	52	88	tetrahedra
Dual surfaces	1198	1374	polygons
Dual regions	1460	1320	polyhedra
Block mesh	50	22	blocks

Table 5.3: Mesh entities used for the L-Shaped domain.

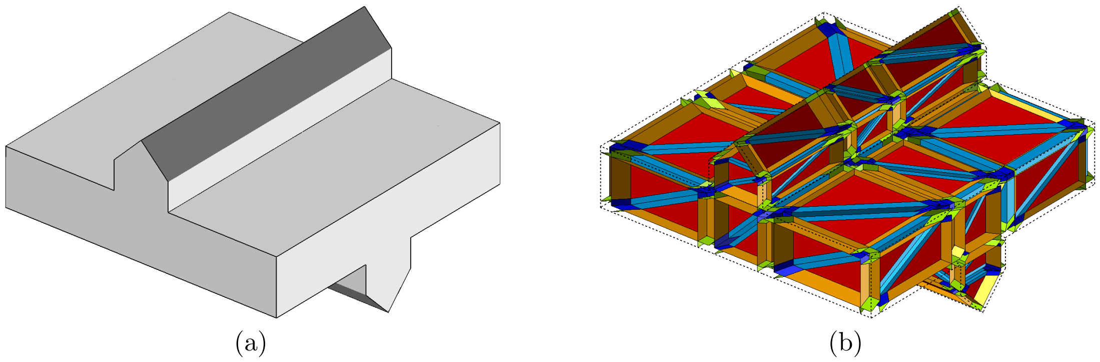
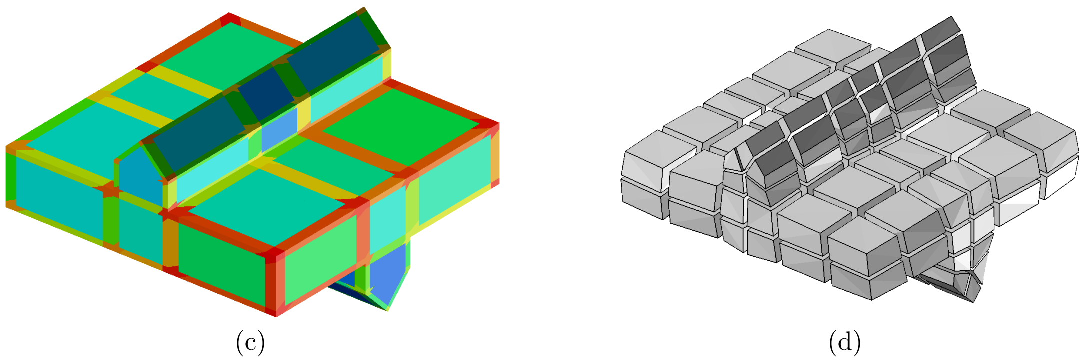



Figure 5.17: Block-meshing for the domain with non-sweepable protrusions: **(a)** domain; **(b)** local dual contributions; **(c)** dual regions; and **(d)** unstructured block mesh.

one of the dual regions corresponds to a primal node of the final block mesh, see Figure 5.16(d). Note that to generate a hexahedral mesh for this domain we can use sweeping but not submapping. Table 5.3 shows the number of nodes and elements that compose the meshes used to obtain the final block mesh.

Domain with non-sweepable protrusions. This non-convex domain presents two transversal and non-sweepable protrusions. The block meshing tool can decompose the domain with the desired unstructured blocks, see Figure 5.17(a). Note that neither submapping nor sweeping techniques can generate this decomposition. The

	Nodes	Elements	Type
Reference mesh	57	114	tetrahedra
Dual surfaces	1750	1374	polygons
Dual regions	1807	1710	polyhedra
Block mesh	260	132	blocks

Table 5.4: Mesh entities used for the domain with non-sweepable protrusions.

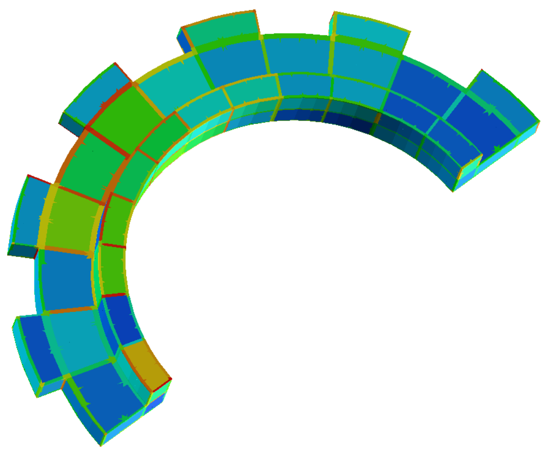
mesher adds local dual contributions to the reference mesh, see Figure 5.17(b), to determine a valid arrangement of dual surfaces. This arrangement delimits several dual regions, see Figure 5.17(c), that determine the final block mesh, see Figure 5.17(d). Table 5.4 shows the number of entities used in the meshing process.

Half of a gear. This non-convex domain presents a large number of corner curves and protrusions around the revolution axis. The shape and the dual regions are depicted in Figure 5.18(a). The different dual regions split the domain around non-convex curves. The resulting block mesh, see Figure 5.18(b), is similar to a coarse mesh obtained with the sweeping and submapping techniques. Owing to the number of corner curves we obtain a great amount of local dual contributions, see Table 5.5.

Thin domain with holes. This non-convex domain is curved and has two square-shaped holes. The dual regions, see Figure 5.19(a), determine two stretched layers of hexahedra through its thickness, see Figure 5.19(b). Since it is a curved domain a large number of boundary faces are needed to represent the shape. Hence, a large number of local dual contributions are needed, see Table 5.6.

Closed loop. This example presents a non-convex domain that can be meshed with neither submapping nor sweeping. Moreover, this example requires a fully unstructured hexahedral generator. Figure 5.21(a) shows the obtained dual regions that lead to the final unstructured block mesh, see Figure 5.21(b). The number of mesh entities used by the algorithm is presented in Table 5.7.

4-valent-protrusion domain.A parallelepiped base with a pyramidal protrusion composes the last example. Herein the difficulty to mesh with hexahedral elements is that the top vertex is 4-valent. Our block mesher splits the space around this feature vertex by following the reference mesh in an advancing front manner. The resulting division allows inserting the 3-valent hexahedral elements. Note that the cuts are propagated along the whole of the domain. The number of entities used along the meshing process is summarized in Table 5.8.


(a)

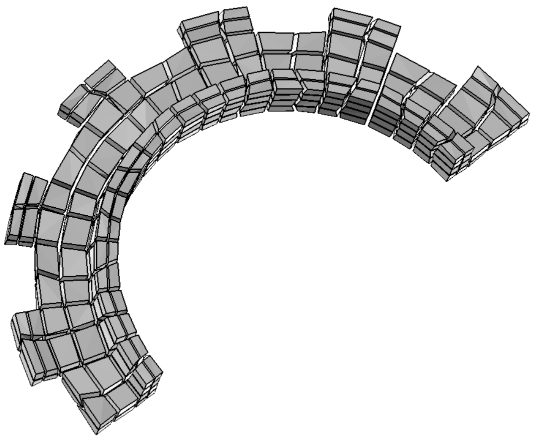
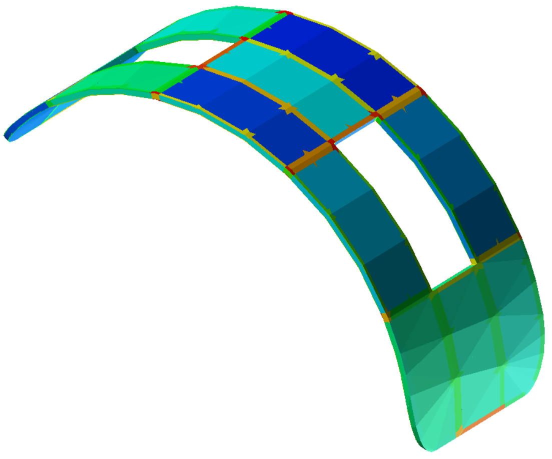

(b)

Figure 5.18: Block-meshing for the half of a gear domain: **(a)** dual regions and **(b)** unstructured block mesh.

	Nodes	Elements	Type
Reference mesh	513	1229	tetrahedra
Dual surfaces	16697	20966	polygons
Dual regions	18840	18435	polyhedra
Block mesh	633	336	blocks

Table 5.5: Mesh entities used for the half of a gear example.


(a)

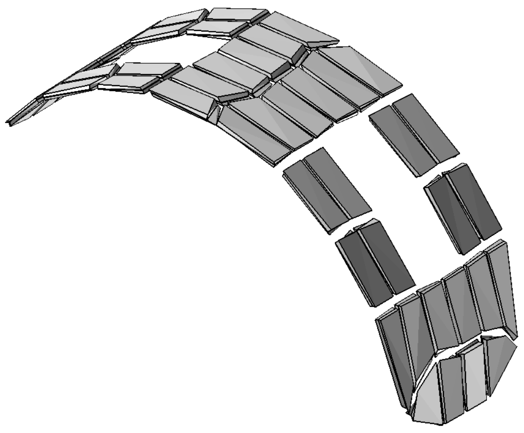

(b)

Figure 5.19: Block-meshing for the thin domain with holes: **(a)** dual regions and **(b)** unstructured block mesh.

	Nodes	Elements	Type
Reference mesh	391	1052	tetrahedra
Dual surfaces	13445	15657	polygons
Dual regions	15764	15780	polyhedra
Block mesh	201	96	blocks

Table 5.6: Mesh entities used for the thin domain with holes.

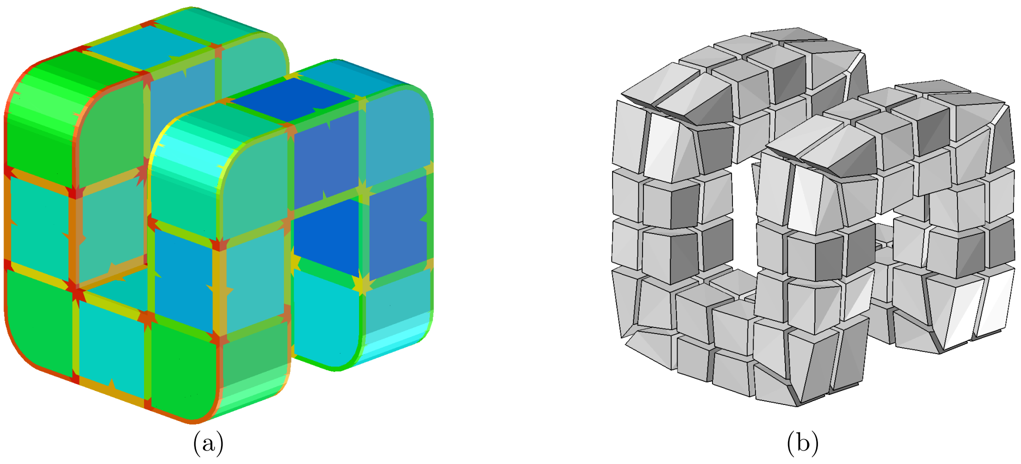


Figure 5.20: A rounded and closed loop geometry: **(a)** dual regions and **(b)** unstructured block mesh.

	Nodes	Elements	Type
Reference mesh	303	579	tetrahedra
Dual surfaces	7527	8626	polygons
Dual regions	9372	8685	polyhedra
Block mesh	240	112	blocks

Table 5.7: Mesh entities used for the rounded loop example.

	Nodes	Elements	Type
Reference mesh	33	56	tetrahedra
Dual surfaces	891	1312	polygons
Dual regions	924	840	polyhedra
Block mesh	177	92	blocks

Table 5.8: Mesh entities used for the 4-valent protrusion example.

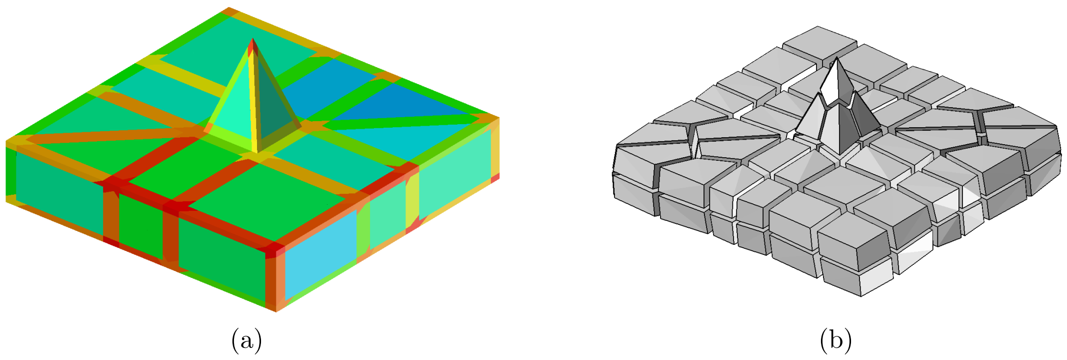


Figure 5.21: Geometry with a 4-valent protrusion (pyramid): **(a)** dual regions and **(b)** unstructured block mesh.

5.8 Concluding remarks

In this chapter we have proposed and detailed a new tool, the local dual contributions, for directly representing the topology and the geometry of a block mesh dual. Specifically, the main contribution of this work is to detail how to add local dual contributions on a reference mesh to represent the desired dual configurations. To this end, we present a hierarchical scheme and a set of matching rules to ensure that we obtain a valid dual configuration. That is, the local dual contributions define a dual of the block mesh with intersections of the proper multiplicity, without gaps and that reproduces the boundary features of the domain.

Practical results suggest that the proposed types of local dual contributions, hard and soft, can capture the dual of the desired block mesh for a wide range of geometries. In particular, soft local dual contributions allow obtaining the same dual configuration for different reference meshes of the domain.

The computational cost of adding local dual contributions is the time of creating a tetrahedral reference mesh plus the time of checking all possible candidates to contribute to the dual. Note that tetrahedral mesh generators are efficient and the number of candidates is 28 times the number of reference elements. Hence, the local dual contributions tool has an asymptotic behavior equivalent to the used tetrahedral mesh generator. Therefore, no effort has been made to optimize the performance of our implementation. Nevertheless, the performance of the current implementation can be improved through code optimization.

This tool allows us to state a novel approach to generate block decompositions of a given domain when there is not a prescribed boundary mesh. The proposed algorithm is developed in three steps: *i)* the generation of a coarse reference mesh composed by tetrahedral elements; *ii)* the insertion of dual surfaces by addition of local dual contributions; and *iii)* the dual regions are dualized to obtained the final block mesh. We have implemented a block mesh generator to show a first application of this scheme. To this end, we have presented an automatic procedure to mark inner reference faces as layers. Several examples show that the current implementation generates the expected decomposition for several convex and non-convex blocky geometries. Moreover, we obtain the desired unstructured decomposition in blocks even in those examples where submapping and sweeping techniques fail. These examples present different geometrical characteristics such as planar surfaces, curved surfaces, thin configurations, holes, and vertices with valence greater than three. Specifically, the tool deals with high-valence vertices due to the use of a reference mesh composed by tetrahedra. The tetrahedral elements provide a natural splitting of the high-valence vertices in 3-valent nodes.

Chapter 6

EZ4U: developing a new mesh generation framework

6.1 Introduction

In this chapter we present an overview of a new mesh generation framework for geometry and mesh based numerical methods. It is a multi-platform framework that integrates a CAD engine and state of the art quadrilateral and hexahedral mesh generation algorithms. The development of this framework, called EZ4U, has been driven by the meshing needs of the members of our research group. Moreover, the tool has evolved with the mesh requirements determined by the techniques presented in this thesis. At the time of starting the development of EZ4U, several integrated environments were available. However, these environments did not fulfill our requirements. The existing tools with quadrilateral and hexahedral meshing capabilities were not open source (Sandia National Labs, 2009; ANSYS, 2009b; Program Development Company, 2009; Pointwise, 2009; ANSYS, 2009a; Simulia, 2009). On the contrary, the available open source projects were oriented to triangular and tetrahedral mesh generation (Geuzaine and Remacle, 2009; Schorbel, 2009) or were not multi-platform (Salome, 2009). Hence, we decided to start from zero the development of EZ4U: an *easy*, mesh generation environment, *for you*.

6.1.1 Contributions

The major contribution of EZ4U is to provide the technology required to implementing the hexahedral mesh generation algorithms and data structures presented in this thesis. That is, the framework incorporates the affine method of Chapter 3, the sweeping scheme of Chapter 4, and the block meshing algorithm of Chapter 5. Moreover, EZ4U is showing to be a productive framework where other researchers can implement mesh generation algorithms (Ruiz-Gironès and Sarrate, 2009, 2008; Ruiz-Gironès et al., 2009) or integrate existing codes for triangular and tetrahedral mesh generation (Shewchuk, 1996; Si, 2007).

6.1.2 Outline

The rest of the chapter is organized as follows. First in Section 6.2 we state the functional and non-functional requirements of the framework. In order to fulfill these requirements EZ4U is structured in four layers outlined in Section 6.3 and detailed in the following sections. In Section 6.4 we describe the design and the data structure of the mesh generation framework. Moreover, we also present the low level modules of the framework: CAD interface, mesh database, meshes with exact boundary representations and high-order nodes, attributes, and hierarchical mesh generation. These features are encapsulated in an application programming interface (*API*) presented in Section 6.5. This API is used to implement all the filing, editing, modeling and meshing operations available in this framework. These operations and a session manager are included in the commands kernel described in Section 6.6. In Section 6.7 we overview the final layer of the framework. This layer is composed by the execution modes that mediate between the user and the commands kernel. In Section 6.8 we present several examples that show the meshing features of EZ4U.

During the management, design and development of the project we have used several software engineering techniques, open source tools and libraries. Details on this information are out of the scope of this chapter. Nevertheless, our election is included in Appendix D since it can be useful for managers, designers, and developers of similar projects.

6.2 Requirements

Our goal is to develop a mesh generation framework that minimizes the user input effort and expertise. On the one hand, it is designed to fulfill the following functional requirements:

- *Ease of use.* It has to provide an intuitive interface that allows users to generate meshes with few operations.

- *Simple and powerful geometry modeler.* The geometry modeler has to allow fast and easy modeling of middle complexity models. In addition, it has to allow importing complex CAD designs by means of standard exchange file formats as STEP and IGES.

- *Quadrilateral and hexahedral mesh generation.* The meshing module has to generate structured and unstructured quadrilateral meshes. Furthermore, it has to generate structured and semi-structured hexahedral meshes.

- *Meshes with exact boundary representation.* The meshing module has to allow converting the boundary entities of a linear mesh to curved edges and faces that define a mesh with exact boundary representation.

- *Generation of high-order nodes.* It has to allow exporting high-order nodes that follow the boundaries of the domain.

- *Triangles and tetrahedra.* It has to integrate existing triangular and tetrahedral mesh generators.

On the other hand, it has to provide a convenient developing framework that verifies the following non-functional requirements:

- *Unify mesh generation code.* All mesh generation code, legacy and research, has to be integrated in the mesh generation environment.

- *Software re-use.* In order to improve productivity we have to use available libraries, adapt legacy code, and implement new classes and methods for later code re-use.

- *Scalability.* We have to promote a design and implementation that facilitates the growing of the application.

- *Maintainability.* We have to apply several software engineering practices ensuring that a small group can maintain the application.

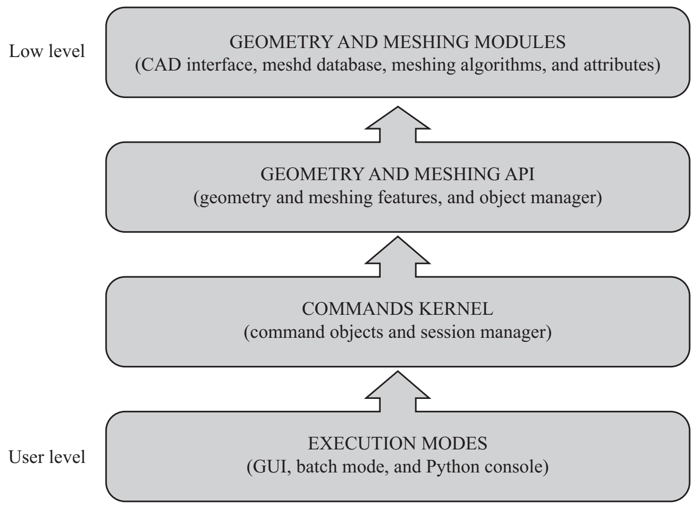

Low level
GEOMETRY AND MESHING MODULES
(CAD interface, meshd database, meshing algorithms, and attributes)
GEOMETRY AND MESHING API
(geometry and meshing features, and object manager)
COMMANDS KERNEL
(command objects and session manager)
User level
EXECUTION MODES
(GUI, batch mode, and Python console)

Figure 6.1: Architecture in layers of EZ4U.

- *Easy to code*. New developers should be productive in short-term. That is, they should be able to add new features without understanding the whole of the source code.

- *Cross-platform*. The environment has to work, at least, in Windows and Linux platforms.

6.3 Design and architecture

EZ4U is organized in four layers: geometry and meshing modules, application programming interface, commands kernel and the user interfaces. Each layer uses the features of the previous layer, except the geometry and meshing modules that provide the low-level features for implementing the application, see Figure 6.1.

- **Geometry and meshing modules**. This layer is responsible of providing the low level features required for the implementation of the mesh generation environment. Five modules compose it: CAD interface, mesh database, meshing algorithms, and attributes management.

- **Geometry and meshing application programming interface (API)**. This layer provides a high level interface to the geometry and meshing modules. The features of the API are the building blocks for implementing the commands kernel. To this end, an object manager focused in the topological entities is implemented. It controls the creation, modification and deletion of model objects and attributes.

- **Commands kernel**. This is the middle layer between the user interfaces and the geometry and meshing API. This kernel has a session history where the execution order of the command objects is stored. The commands depend on the input parameters and are implemented using the geometry and meshing API. Moreover, this layer provides undo and redo capabilities and the possibility of loading and saving session files.

- **Execution modes**. This layer is composed by the three possible execution modes of the environment: the graphical user interface (GUI), the batch mode and the python console. The user can set input parameters and execute command to obtain the desired result.

6.4 Geometry and meshing modules

Mesh generation environments require the specification, definition and implementation of a large amount of data types. Moreover, these data types have to collaborate in order to represent complex data structures as meshes, solid CAD models or mesh generation algorithms. To this end, we have adopted the programming paradigms and software engineering techniques presented in Appendix D. They allow us to define useful and scalable classes of objects. These objects store data and collaborate between them by means of data aggregation and inheritance. In this section we present the concepts related to these classes and an overview of their implementation.

6.4.1 CAD interface

This module includes the required features to represent the model to be meshed. Specifically, it implements: modeling operations, classes for geometrical and topological representation, and functions to query geometrical properties. To this end, this module is interfaced with the OpenCascade library (Open CASCADE, 2007).

Modeling features

EZ4U provides a set of basic modeling features that allow to define simple and middle complexity geometries. These features fulfill the necessities of our users in academia. However, EZ4U can also mesh industrial CAD models. To fulfill this goal, the environment allows importing exchange files in STEP format obtained in an external CAD modeler.

Geometrical and topological representation

Similar to the work by Athanasiadis and Deconinck (2003), a combined topological and geometrical model representation is adopted. In particular, two kinds of entities are considered:

- *Geometrical entities.* They are the geometric realizations used to define a domain. They are classified in four types: points, curves, surfaces, and volumes. Note that each realization may have several representations. For instance, a straight line, a B-spline or a NURB can represent a curve.
- *Topological entities.* They are the objects that are used to define the adjacency relationships between geometrical entities. They define the manner in which geometrical entities are composed and connected. Only one geometrical entity corresponds to each topological entity.

The data structures used to describe the geometrical and topological structure of the model follow the STEP representation (TC184 and SC, 2009). Hence, one basic entity is defined for each dimension. The basic topological entities are:

- *Vertex.* Topological entity of dimension 0, its geometrical representation is a *point.*
- *Edge.* Topological entity of dimension 1, its geometrical representation is a *curve.* Two adjacent edges share at least one vertex.
- *Face.* Topological entity of dimension 2, its geometrical representation is a *surface.* Its boundary is defined by one or more loops of edges. The first loop defines the outer boundary, and the other loops define the inner holes. Two adjacent faces share at least one edge.

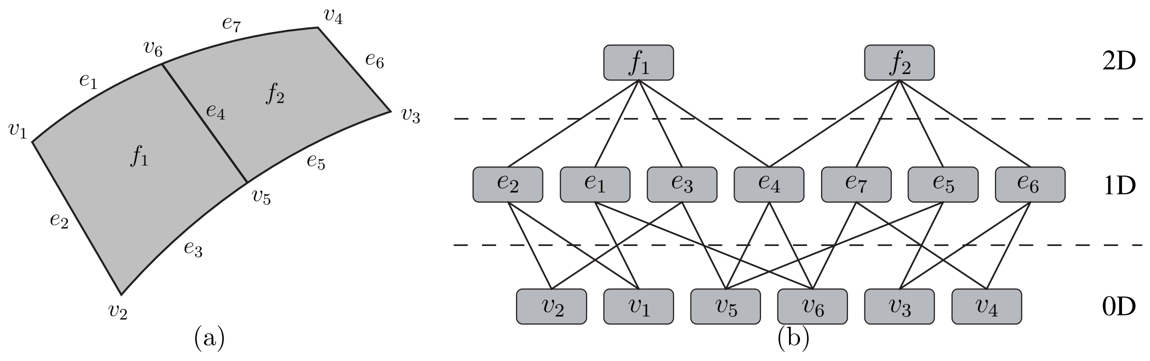


Figure 6.2: Topological representation of two surfaces: **(a)** A domain composed by two surfaces that share an edge; **(b)** hierarchical organization of the entities.

- *Solid.* Topological entity of dimension 3, its geometrical representation is a *volume* with or without holes. It is defined by one or more loops of faces joined by edges. Two adjacent solids share at least one face.

We have also defined two additional topological entities:

- *Wires.* Topological entity of dimension 1 that determines a closed loop of edges.
- *Shells.* Topological entity of dimension 2 that determines a closed loop of faces.
- *Compound solids.* Topological entity of dimension 3 that determines a composition of several adjacent solids.

In order to illustrate these definitions, Figure 6.2 shows the hierarchical representation of a surface defined by two faces, f_1 and f_2. These faces are defined by four edges, and share edge e_4. The face f_1 is defined by the set of edges $\{e_1, e_2, e_3, e_4\}$, while face f_2 is defined by the set $\{e_4, e_5, e_6, e_7\}$. Similarly, all the edges are defined by the set of vertices $\{v_1, v_2, v_3, v_4, v_5, v_6\}$. The solid lines in Figure 6.2(b) reveal the hierarchical relationships and the shared entities. For instance, the solid lines shows that edge e_4 is shared by faces f_1 and f_2, or that vertices v_5 and v_6 are shared by three edges since three solid lines are attached to them.

It is important to point out that for each topological entity there exists one geometrical representation:

- *Point.* The geometrical representation of a vertex is a point p in $\mathbb{R}^3$.

- *Curve.* The geometrical representation of an edge is the parameterization of NURBS curve
$$\boldsymbol{\gamma} : [t_0, t_1] \subset \mathbb{R} \rightarrow \mathbb{R}^3.$$

- *Surface.* The geometrical representation of a face is either the parameterization of a NURBS surface
$$\boldsymbol{\psi} : [u_0, u_1] \times [v_0, v_1] \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3,$$
or the restriction of a NURBS surface to a subset D of $[u_0, u_1] \times [v_0, v_1]$
$$\boldsymbol{\psi}|_D : D \subset [u_0, u_1] \times [v_0, v_1] \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3.$$
The latter representation is referred as *trimmed surface* and allows defining surfaces with holes and bounded by more than four curves.

- *Volume.* It is the region of the space determined by the boundary surfaces of the corresponding solid.

We use an object-oriented programming paradigm to build the interface between the topological and geometrical entities. Hence, we hide the realization of the geometrical entities (a straight line or a NURBS for curves) of its associated topological object (an edge). Specifically, in the implementation, the `Shape` class is abstract and the rest of topological entities are specializations of this one, see Figure 6.3. That is, all topological entities are used by means of the same function interface, determined by the abstract class `Shape`, and each particular class provides the required specialized behavior.

Geometrical queries

Finally, mesh algorithms need to query several geometrical properties of the model to be meshed. This module provides an abstract interface between the geometrical representation of the CAD model and the meshing module. Specifically, several query functions are provided for the geometrical entities:

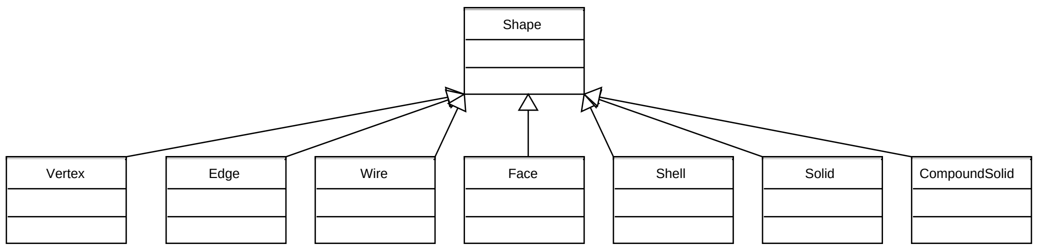


Figure 6.3: `Shape` class hierarchy diagram.

- To obtain the geometrical representation p of a given vertex v.
- Given a parameterized curve $\boldsymbol{\gamma}$ and a parameter t in $[t_0, t_1]$ it is required to obtain the coordinates of $\boldsymbol{\gamma}(t)$, the derivative value $\dot{\boldsymbol{\gamma}}(t)$, or the curvature k of $\boldsymbol{\gamma}$ at t. Moreover, it is required to obtain the parameter t in $[t_0, t_1]$ of a point p over the curve $\boldsymbol{\gamma}$.
- Given a parameterized surface $\boldsymbol{\psi}$ and the parameters (u, v) in $D \subset [u_0, u_1] \times [v_0, v_1] \subset \mathbb{R}^2$, it is required to obtain the coordinates of $\boldsymbol{\psi}(u, v)$, the directional derivative vectors $\boldsymbol{D}_u\boldsymbol{\psi}(u, v)$ and $\boldsymbol{D}_v\boldsymbol{\psi}(u, v)$, and the principal curvature values k_1 and k_2 with principal directions $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$. Moreover, it is required to obtain the parameters (u, v) in $D \subset [u_0, u_1] \times [v_0, v_1]$ of a point p over the surface $\boldsymbol{\psi}$, or a curve $\tilde{\boldsymbol{\gamma}}$ defined in the parameter space $D \subset [u_0, u_1] \times [v_0, v_1] \subset \mathbb{R}^2$ such that $\boldsymbol{\gamma} = \boldsymbol{\psi} \circ \tilde{\boldsymbol{\gamma}}$.

6.4.2 Mesh database

In mesh generation procedures it is required to add, remove and find mesh entities. Moreover, mesh data structure has to allow querying for adjacencies between the different mesh entities, i.e. query for the elements that surround a given node, or for the faces that share and edge. In Figure 6.4 we present the inheritance and collaboration diagram for the `Element` and `Mesh` database classes. Thus, different types of entities together with their adjacencies compose a mesh. The abstract class `Element` represents the common interface for the mesh entities. Each type of mesh entity is a specialization of this abstract class. Specifically, `Node`, `Edge`, `Face` and `Cell` represent the 0D, 1D, 2D and 3D entities of a mesh, respectively. Class `Mesh` has four dynamical containers to store nodes, edges, faces and cells.

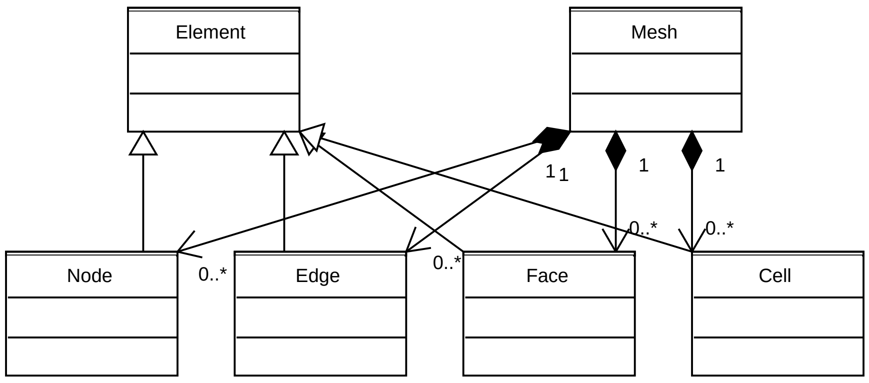


Figure 6.4: `Element` and `Mesh` class hierarchy and collaboration diagram.

This module supports the standard element shapes of the finite element method and the discontinuous Galerkin method: segments, triangles, quadrilaterals, tetrahedra, hexahedra, and prisms. These shapes are implemented as specializations of the classes `Edge`, `Face`, and `Cell`.

6.4.3 Exact boundary representation and high-order meshes

The mesh generation algorithms of EZ4U generate linear meshes. Nevertheless, these discretizations can be converted to meshes with exact boundary representation or high-order meshes. To this end, all the mesh entities store a pointer to the corresponding topological entity of the model. Via this pointer each mesh entity can access to the properties of the underlying geometrical representation:

- The topological vertices of the model are discretized with mesh nodes. These mesh nodes can access to the physical coordinates of the associated geometrical point.
- The topological edges of the model are discretized with mesh nodes and mesh edges. A mesh node can access to its parametric coordinate t on the associated parameterized curve $\boldsymbol{\gamma}$. In addition, a mesh edge can access to its associated parameterized curve $\boldsymbol{\gamma}$.
- The topological faces of the model are discretized with mesh nodes, mesh edges, and mesh faces. A mesh node can access to its parametric coordinates (u, v) on the associated geometric surface $\boldsymbol{\psi}$. A mesh edge is represented by a physical

curve over the parameterized surface $\boldsymbol{\psi}$. Specifically, this curve is the image by $\boldsymbol{\psi}$ of a linear segment on the parametric space. The limits of this linear segment are the parametric coordinates (u_p, v_p) and (u_q, v_q) of the limit nodes of the edge p and q respectively. A mesh face can access to its associated parameterized surface $\boldsymbol{\psi}$.

- The topological solids of the model are discretized with mesh nodes, mesh edges, mesh faces, and mesh cells. These mesh entities can access to the associated topological solid.

The association of the mesh entities with the model entities allows obtaining *meshes with exact boundary representation*. That is, the mesh entities of the boundary can be represented by means of the points, curves, and surfaces of the model boundary:

- **Mesh nodes of the boundary**. A mesh node of a parameterized curve $\boldsymbol{\gamma}$ is represented by its parametric coordinate t. In addition, a mesh node of a parameterized surface $\boldsymbol{\psi}$ is represented by its parametric coordinates (u, v).

- **Mesh edges of the boundary**. A mesh edge over a parameterized curve $\boldsymbol{\gamma}$ is represented by the restriction of this curve to $[t_p, t_q]$, where t_p and t_q are the parameters of the edge limiting nodes p and q. A mesh edge of a parameterized surface $\boldsymbol{\psi}$ is represented by a curve over the geometric surface. This curve is the image by $\boldsymbol{\psi}$ of the straight segment determined by the parametric coordinates (u_p, v_p) and (u_q, v_q) of the limiting nodes p and q of the edge.

- **Mesh faces of the boundary**. A mesh face over a parameterized surface $\boldsymbol{\psi}$ is represented by the restriction of this surface to the region delimited by its bounding mesh nodes and mesh edges.

A mesh with exact boundary representation can be used to obtain high-order meshes of degree p by generating additional inner nodes for the mesh edges, faces, and cells. To this end, we first generate a Gauss-Lobatto distribution of $p + 1$ points in the interval $[-1, 1]$. Second, this distribution is mapped to the straight and curved mesh edges according to the arch length parameter of the underlying geometrical curve. Then the high-order nodes in the interior of the faces are generated depending on the element type:

- **Quadrilateral**. A transfinite interpolation of the physical coordinates of the high-order nodes of the edges determines the location of the high-order nodes of the face interior.

- **Triangle**. The physical coordinates of the high-order nodes of the edges are used to obtain an iso-parametric mapping with a smooth Jacobian. This mapping is used to map a Fekette distribution of points on a reference element to obtain the location of the high-order nodes for the face interior. To obtain this mapping we use the procedure described in (Hesthaven and Warburton, 2008). Note that the obtained high-order nodes ensure the optimal convergence rate of the Lagrange interpolation error.

6.4.4 Attributes

Mesh generation environments have to deal with several properties assigned to the entities that characterize the model to simulate. These properties usually correspond to boundary conditions and to material properties of the model. We denote these properties as *model attributes* and we provide an abstract and generic procedure to deal with them. The user can mark with labels the topological entities that compose the model in order to properly assign boundary or material conditions. Each label can be applied to several entities of the same dimension. In addition, we can apply different labels to the same entity. Moreover, the user can choose if the labels are applied over the nodes or over the elements of the topological entities. In the first case, we can mark with the same label the nodes of a group of vertices, edges, faces or solids. In the second case, we can mark with the same label: the mesh edges over a group of model edges; the mesh faces over a group of model faces; or the mesh cells over a group of model solids.

In several mesh generation environments the attributes are assigned to mesh entities. Therefore, the users need to re-mesh the model each time that an attribute is modified. We have developed a data structure that overcomes this drawback. Specifically, mesh entities are contained in `Mesh` objects that are associated to geometrical entities. When a mesh entity asks for its attributes, the container mesh delegates the query to the corresponding geometrical entity. Not storing the value of the attributes in the mesh entities, and querying to the corresponding entity instead, allow avoiding the time consuming task of remeshing the model when a given attribute is changed.

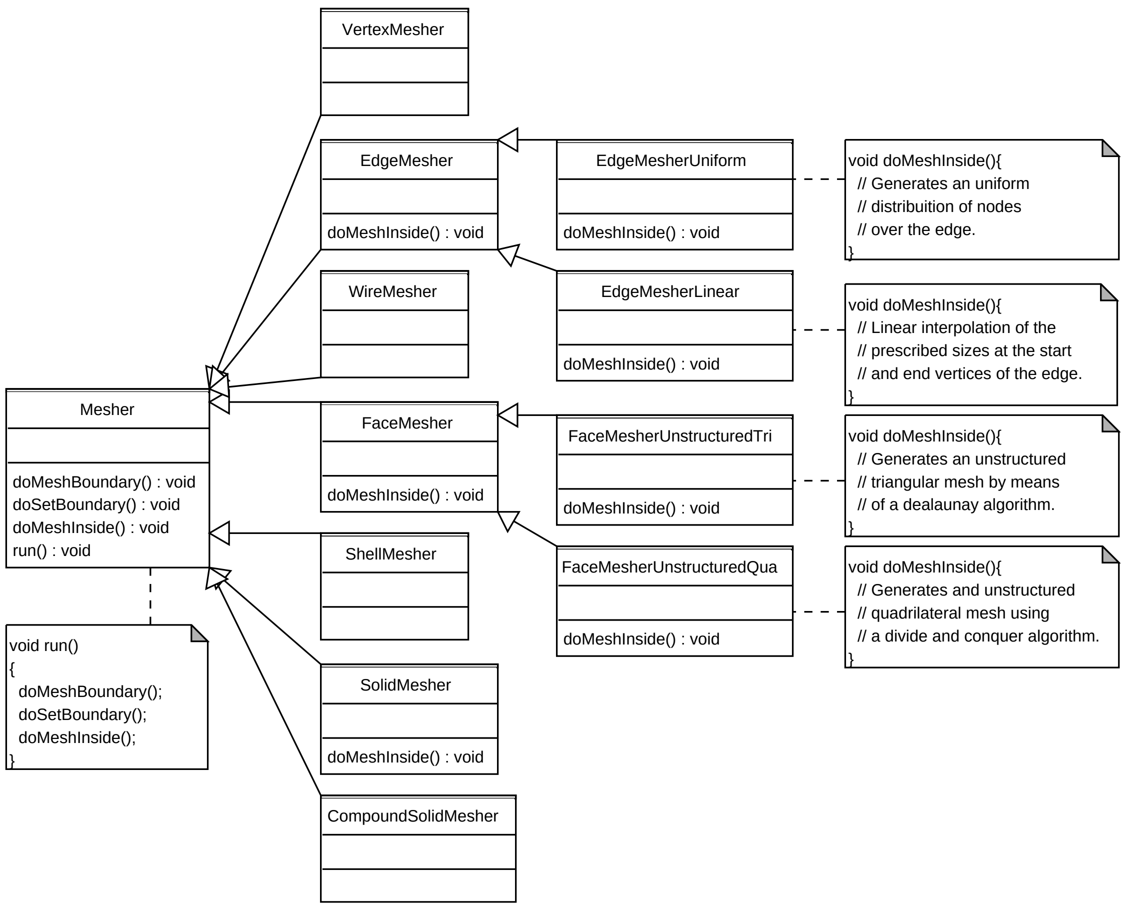


Figure 6.5: `Mesher` class hierarchy diagram.

6.4.5 Hierarchical mesh generation

Similar to the topological and geometrical description, the meshing algorithms are also implemented according to a hierarchical structure (Athanasiadis and Deconinck, 2003). Therefore, the meshing algorithms are adapted to the topological and geometrical representation in a natural manner. In this sense, current implementation allows to add a new mesh generation algorithm overloading only a particular set of functions. It is important to point out that this structure is essential to ensure consistency (conformity) between meshes corresponding to adjacent entities.

The basic idea is to start by meshing entities of dimension 0 (vertices), and then to proceed by meshing entities of dimension 1 (edges), dimension 2 (faces), and finally entities of dimension 3 (solids). That is, a mesh generation algorithm is assigned to each entity of dimension d. These algorithms use as boundary mesh one or several meshes corresponding to the discretization of the boundaries entities of dimension $d-1$. These boundary meshes have been previously obtained and may be shared by other entities of dimension d. Therefore, the process always begins by meshing all the entities of lower dimension. Taking into account this property, we define four abstract classes for mesh generation algorithms: `VertexMesher`, `EdgeMesher`, `FaceMesher`, and `SolidMesher` (one for each type of topological entities). In addition we consider the abstract classes `WireMesher`, `ShellMesher`, and `CompoundSolidMesher` that allow sewing meshes over groups of edges (wires), faces (shells), and solids (compound solids).

Figure 6.6 shows the hierarchical mesh generation process corresponding to the discretization of the domain presented in Figure 6.2. In the first step, all the entities of dimension 0 (vertices) are meshed, see Figure 6.6(a). Taking into account this discretization, in the second step the seven entities of dimension 1 (edges) are meshed, see Figure 6.6(b). Finally, the mesh corresponding to the two entities of dimension 2 (faces) are obtained from the discretization of the edges, see Figure 6.6(c).

Object oriented paradigm allows to define a natural class hierarchy for the meshers and their specializations, see Figure 6.5. The root class is `Mesher` that provides the common interface to all meshers. A mesher is executed by means of the `run` function, that calls sequentially the not already implemented functions: `doMeshBoundary`, `doSetBoundary` and `doMeshInside`. First function, `doMeshBoundary`, calls all the meshers associated to the boundary entities of the entity to be meshed. Second function, `doSetBoundary`, adds to the mesh of the current mesher the boundary nodes obtained with `doMeshBoundary`. Third function, `doMeshInside`, calls the code of the algorithm used to mesh the inner part of the selected entity. These three functions are pure virtual, i.e. they are not implemented in the `Mesher` class and have to be defined by the particular specializations of the `Mesher` class. The second level of classes in the hierarchy override the `doMeshBoundary` and `doSetBoundary` functions, but not `doMeshInside`. The latter is implemented by the specializations of `Mesher` that are on the third level. For instance, `EdgeMesherUniform` and `EdgeMesherLinear` override the `doMeshInside` function in order to specify a particular `EdgeMesher`. Similarly `FaceMesherUnstructuredTri` and `FaceMesherUnstructuredQua` are specializations of `FaceMesher` that allow to mesh the surface of a face with triangular or quadrilateral elements, respectively.

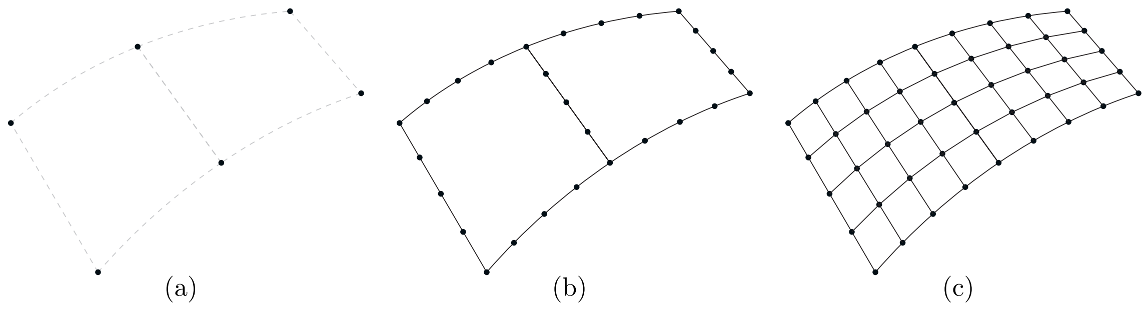


Figure 6.6: Hierarchical mesh generation process corresponding to the discretization of the domain presented in figure 6.2. **(a)** 0D mesh; **(b)** 1D mesh; and **(c)** 2D mesh.

6.5 Geometry and meshing API

This layer is responsible of integrating the features of the geometry and meshing modules. To this end the different classes have to collaborate together. This collaboration is represented by many connections between objects, which may reduce the reusability of the code. Adding more connections can lead to objects that cannot work without the support of the others. Moreover, little changes in one part of the source code might be propagated through the rest of the code. To solve these drawbacks we introduce `Key` and `KeyManager` classes that manage the collaboration between objects, see Figure 6.7. The first one is devoted to mediate collaborations between the geometry and topology, attributes and meshing functions. For instance, meshers are related to one shape, and can query for their geometrical properties, such as: point coordinates, distances, curvatures or tangent vectors. Thus, assigned attributes over shapes can be obtained for a particular mesh entity. Each key has one mesher and one mesh corresponding to a particular shape. Each key has at most one background mesh, and as many attributes as it is required. The second class, `KeyManager`, is used to manage the instances of objects. Basically, it allows adding, removing and finding `Key` class instances that collaborate to provide the geometry, topology, attributes and meshing features.

6.6 Commands kernel

This layer integrates a session manager and several command objects. On the one hand, the session manager controls the command execution history. Thus, the user

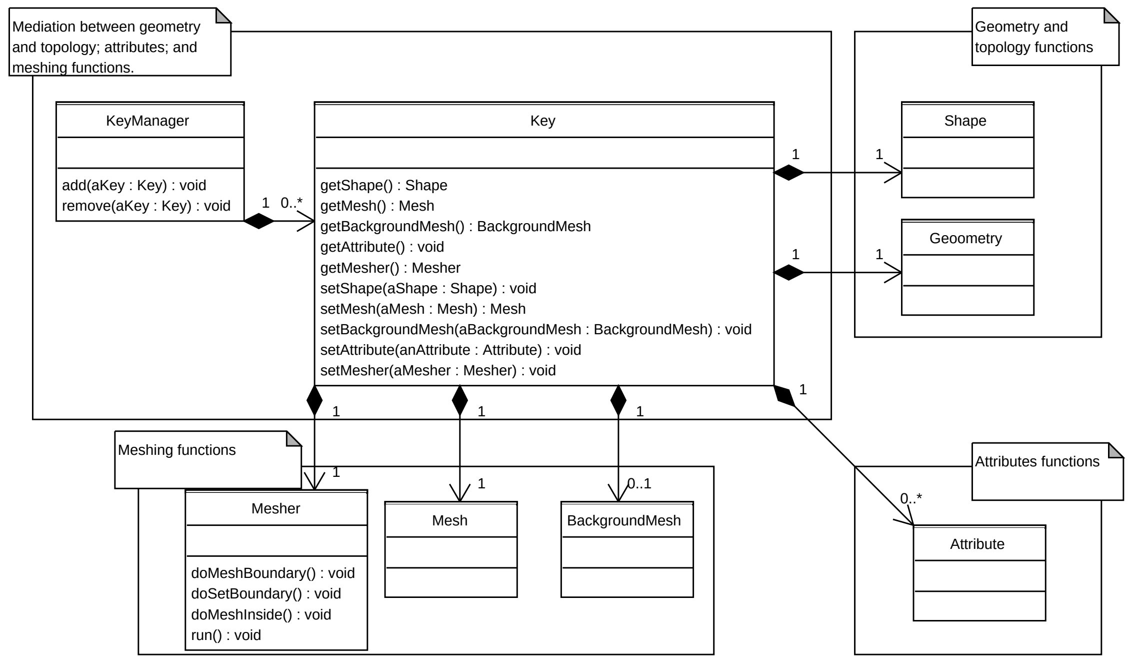

Mediation between geometry and topology; attributes; and meshing functions.
KeyManager
add(aKey : Key) : void
remove(aKey : Key) : void
1
0..*
Key
getShape() : Shape
getMesh() : Mesh
getBackgroundMesh() : BackgroundMesh
getAttribute() : void
getMesher() : Mesher
setShape(aShape : Shape) : void
setMesh(aMesh : Mesh) : Mesh
setBackgroundMesh(aBackgroundMesh : BackgroundMesh) : void
setAttribute(anAttribute : Attribute) : void
setMesher(aMesher : Mesher) : void
Geometry and topology functions
Shape
Geoometry
Meshing functions
Mesher
doMeshBoundary() : void
doSetBoundary() : void
doMeshInside() : void
run() : void
Mesh
BackgroundMesh
0..1
Attributes functions
Attribute
0..*

Figure 6.7: `KeyManager` and `Key` classes mediate collaboration between geometry/topology, attributes, and meshing functions.

interfaces can load and save geometry modeling and mesh generation sessions from a session file. On the other hand, the command objects are designed according to the command pattern presented in (Gamma et al., 1995). This design provide undo and redo capabilities to EZ4U. The command objects are called from the user interface and are grouped in five *contexts*:

- *File context.* It includes the entire file input and output commands. Using this context the user can import and export a CAD model or open/save a previously developed model.

- *Edit context.* It enfolds all the edition operations. For instance, the user can perform operations such as copy, move, rotate, scale, mirror, collapse or delete a given entity.

- *Topology and geometry context.* It provides access to commands that create or repair topological and geometrical entities. For instance, a user can create a geometrical domain from a set of geometrical primitives.

- *Attributes context.* It contains commands to assign attributes to a selected entity. For instance, the user can select an entity and assign to it a scalar or vector field related to some material properties or boundary conditions.

- *Mesh context.* It includes commands related to the mesh generation algorithms. From this context the user selects the mesh generation algorithm and assigns the required parameters such as the element size, the background mesh, or type of elements. Moreover, a quantitative analysis of the mesh quality can be conducted and displayed from this context.

6.7 Execution modes

The end user can access to EZ4U features by means of three execution modes: the graphical user interface (GUI), the batch mode, and the python console.

6.7.1 Graphical user interface (GUI)

The implemented GUI allows to visually monitoring the steps performed in the mesh generation process. The user can verify the results of their actions using standard visual steering tools (i.e. zoom, panning, rotation). The GUI of the mesh generation environment, see Figure 6.8, is composed by eight widgets:

- *Menu bar.* It provides access to the five contexts grouped in pop-up menus.

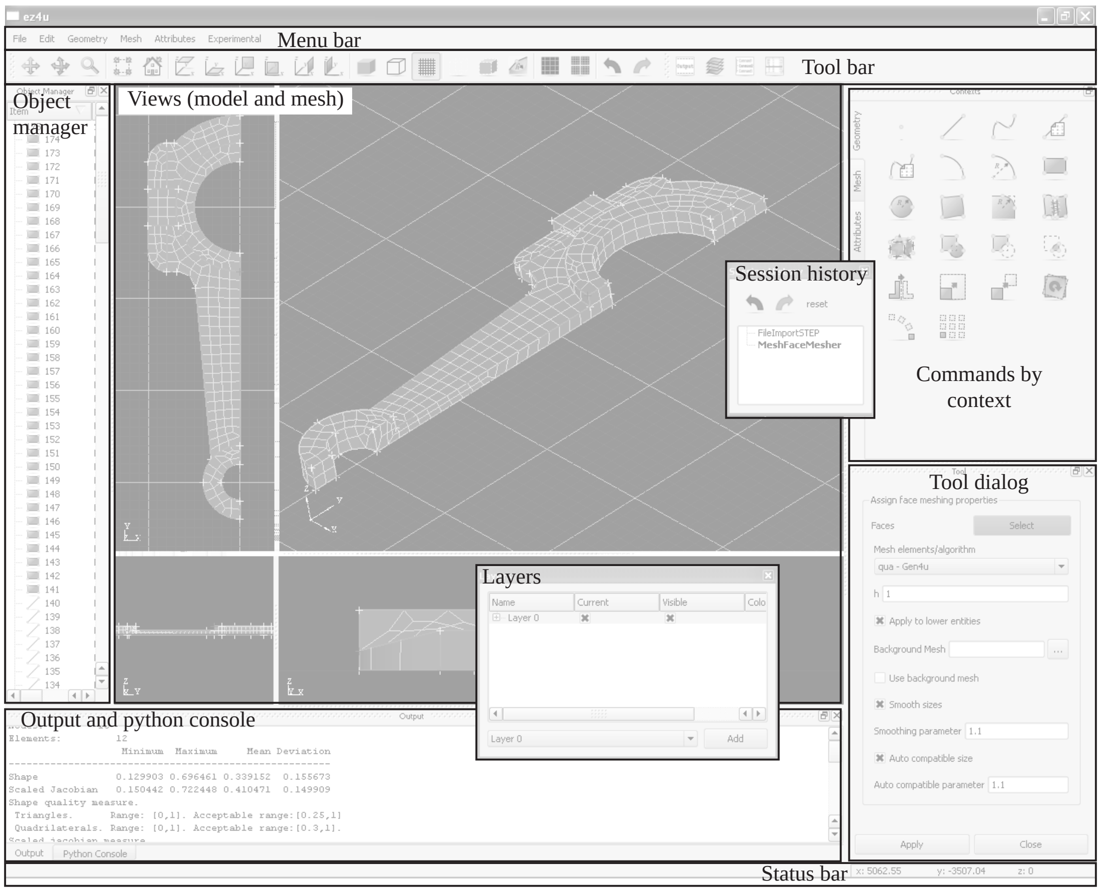
- *Tool bar.* It includes buttons for activating and deactivating steering tools, visualization modes and dialog windows.

- *Views.* It displays one or several views of the current model and meshes.

- *Commands by context.* It provides access to the commands of the geometry, attributes, and mesh contexts. The commands are grouped in pages accessible by means of a tab bar.

- *Tool dialog.* It shows the user interface for the current command. The user can set the parameters and execute the command.

- *Output and python console.* It is used to input commands, and to display log, warning and error messages.



```
Elements:         12
                  Minimum  Maximum     Mean Deviation
-------------------------------------------------------
Shape             0.129903 0.696461 0.339152  0.155673
Scaled Jacobian   0.150442 0.722448 0.410471  0.149909
Shape quality measure.
 Triangles.      Range: [0,1]. Acceptable range:[0.25,1]
 Quadrilaterals. Range: [0,1]. Acceptable range:[0.3,1].
```

Assign face meshing properties

Faces: Select

Mesh elements/algorithm: qua - Gen4u

h: 1

✓ Apply to lower entities

Background Mesh:

☐ Use background mesh

✓ Smooth sizes

Smoothing parameter: 1.1

✓ Auto compatible size

Auto compatible parameter: 1.1

Apply | Close

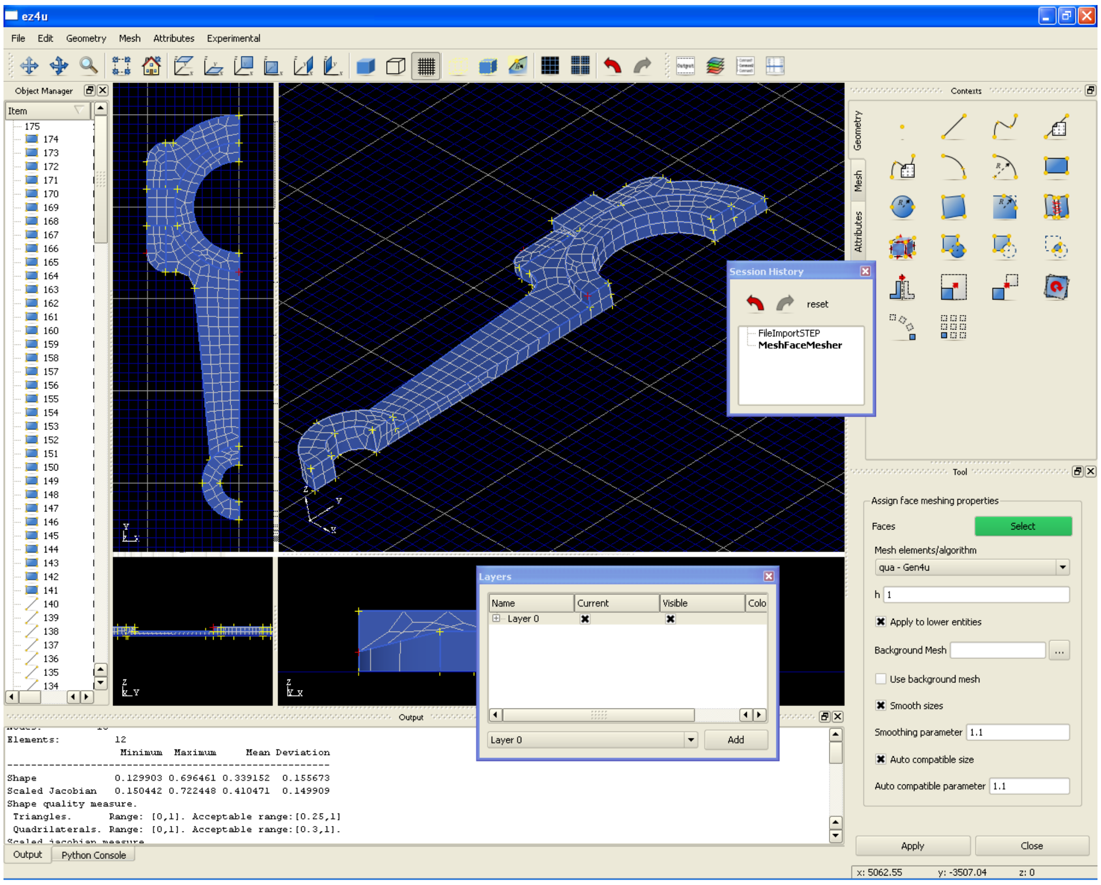
x: 5062.55 y: -3507.04 z: 0

Figure 6.8: A sketch of the GUI of the environment for geometry-based simulations.

- *Session history.* It shows a list of commands sorted according to the execution order. The user can directly access to a previous step, or undo and redo the last command.

- *Layers.* This dialog allows managing and organizing objects in independent groups (*layers*). Thus, the user can manipulate all the model entities in the same layers at once. For instance, we can turn off, change their display color, and select all of them with one operation.

- *Object manager.* It lists the components of the model according to its type. The selection of a list item is equivalent to select the model entity in the main view.



```
Elements:          12
                   Minimum  Maximum     Mean Deviation
-----------------------------------------------------
Shape              0.129903 0.696461 0.339152  0.155673
Scaled Jacobian    0.150442 0.722448 0.410471  0.149909
Shape quality measure.
 Triangles.       Range: [0,1]. Acceptable range:[0.25,1]
 Quadrilaterals. Range: [0,1]. Acceptable range:[0.3,1].
```

Assign face meshing properties
Faces: Select
Mesh elements/algorithm: qua - Gen4u
h: 1
☒ Apply to lower entities
Background Mesh:
☐ Use background mesh
☒ Smooth sizes
Smoothing parameter: 1.1
☒ Auto compatible size
Auto compatible parameter: 1.1

x: 5062.55 y: -3507.04 z: 0

Figure 6.9: Screenshot of the GUI of the mesh generation environment.

Figure 6.9 shows a screenshot of the GUI. The central widget shows four views of a mechanical piece model: one 3D image in perspective; and three 2D images obtained from the top, left and front of the object. The tool dialog shows the parameters of the unstructured quadrilateral mesh command. Moreover, this command appears as executed in the session history. The resulting mesh is shown in the different views and its quality is listed in the output window. The faces, edges and vertices that compose the model are listed in the object manager window on the left. Finally, the layers widget shows that all the model entities are in the same layer.

6.7.2 Batch mode

EZ4U comes with a so-called batch mode that allows the user to generate meshes from the command line. This mode allows to integrate EZ4U in mesh adaption procedures or to generate meshes without user interaction. The batch mode has a series of flags to determine the geometry file, element type, mesh algorithm, background mesh, the degree p of a high-order mesh, and the output file. Moreover, the user can run previously saved sessions in the GUI in a non-interactive mode.

6.7.3 Python console

The end user has full access from the Python console to all the commands contexts of EZ4U: file, edit, geometry and topology, attributes, and meshing. This interface allows writing Python scripts that glue the mesh environment with other calculations or to parameterize the input data of the EZ4U commands. For instance, the user can automate processes such as: creating parameterized geometries; generating a succession of finer meshes for convergence studies; or integrating the mesh generator in an adaptivity procedure.

6.8 Mesh generation features

In this section we present several mesh generation features that are available in EZ4U. Note that these features are not usually found in commercial and non-commercial mesh generation tools. They provide to our package an additional value and are of major importance to fulfill the mesh requirements presented in the introduction.

6.8.1 Quadrilateral mesh generation

In order to obtain a conformal quadrilateral mesh it is required that shared edges of adjacent surfaces have the same number of nodes, and surface boundaries have to be discretized with an even number of nodes. The former is satisfied by means of the adopted hierarchically based mesh paradigm (Athanasiadis and Deconinck, 2003). To satisfy the latter EZ4U automatically solves a linear integer problem subject to the parity constrain. The presented framework incorporates algorithms for generating unstructured and structured conform meshes of quadrilaterals.

Unstructured. To generate unstructured quadrilateral meshes we have implemented an extension to NURBS surfaces (Roca et al., 2005a) of an algorithm previously developed by Sarrate and Huerta (2000b,a). The algorithm is based on an automatic and recursive decomposition of the domain until quadrilateral elements are

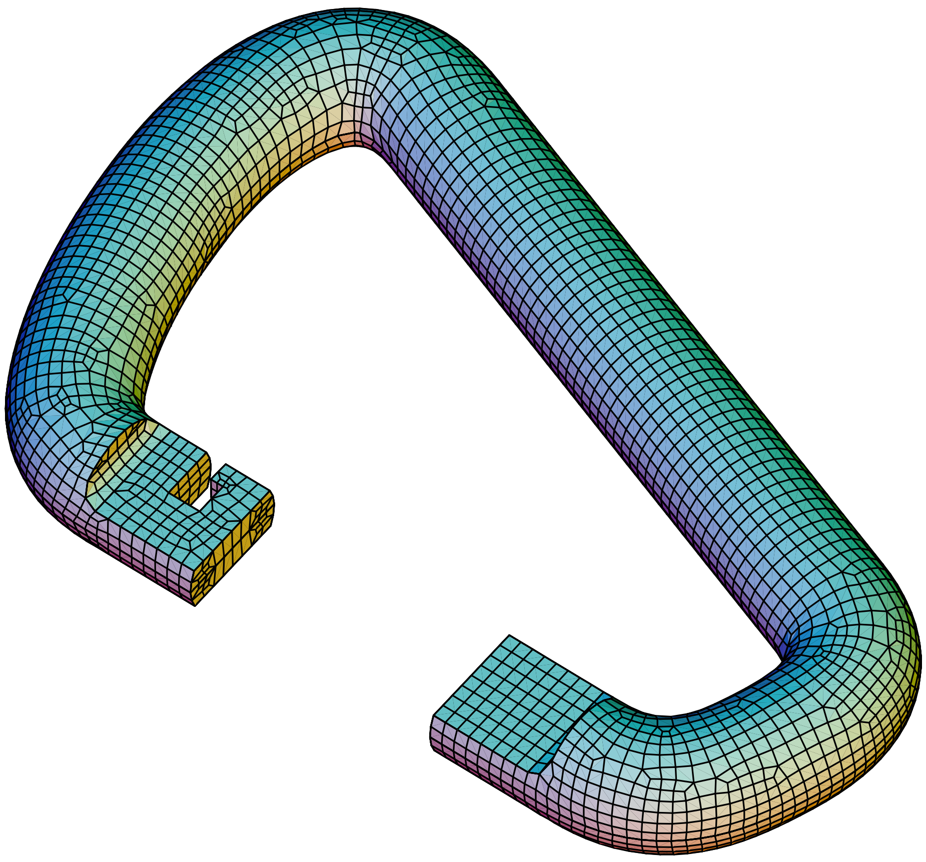


Figure 6.10: Unstructured quadrilateral mesh over the surface of a snap harness.

obtained. Desired element size can be prescribed by means of a background mesh. Figure 6.10 shows an unstructured quadrilateral mesh for a snap harness.

Structured. The generation of structured meshes is frequently required in several applications of mesh based numerical methods. To this end, EZ4U includes an implementation (Ruiz-Gironès and Sarrate, 2009) of the submapping 2D algorithm (White, 1996). This meshing procedure splits the surface into several patches, logically equivalent to a quadrilateral, and then meshes each patch separately. The submapping algorithm can only be applied to geometries where the angles between consecutive edges are, approximately, multiples of straight angles. User assigns desired size and the implemented procedure automatically classifies vertices angles, divides each edge with a compatible number of segments, and finally generates the structured mesh.

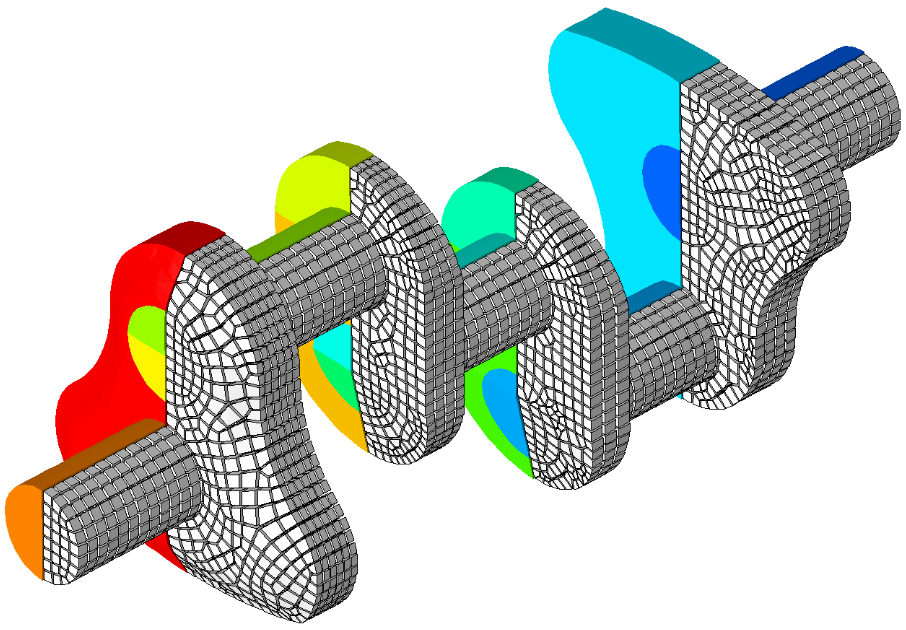
6.8.2 Hexahedral mesh generation

To obtain hexahedral meshes domains are decomposed into several simpler sub-volumes. Then these sub-volumes are discretized by means of specific hexahedral mesh generators such as sweeping or submapping, see Section 1.2 of Chapter 1. The presented framework incorporates both meshing techniques. Furthermore, it includes the block meshing technique presented in Chapter 5.

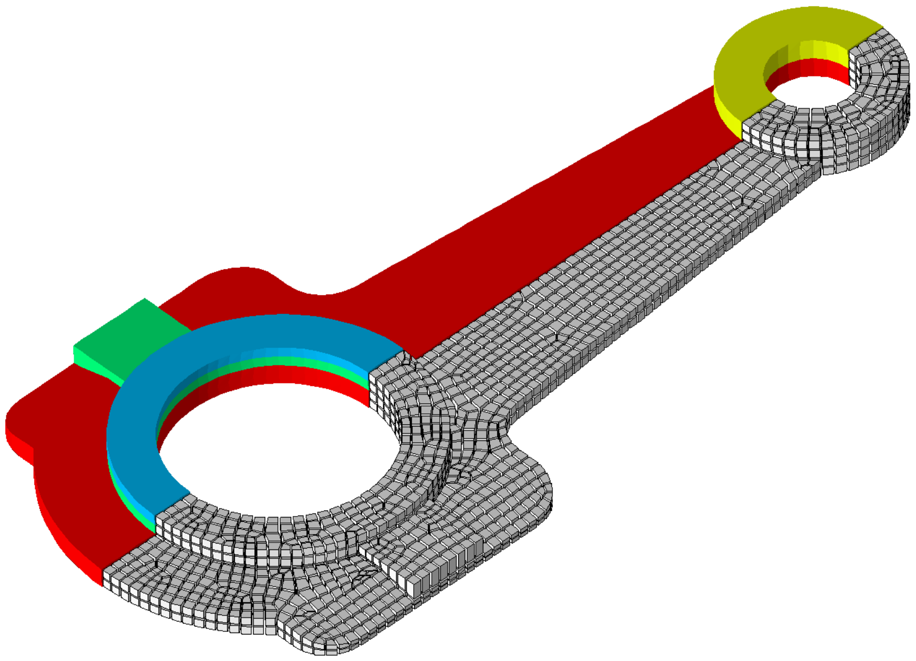
Sweeping. The sweeping technique allows to generate semi-structured hexahedral elements in extrusion volumes, see Chapter 2. The presented framework incorporates a fast and robust implementation of the sweeping technique detailed on chapters 3 and 4. To illustrate the capabilities of this implementation, we present several sweeping meshes in Section 4.4 of Chapter 4.

Moreover, EZ4U features a new implementation (Ruiz-Gironès, Roca, and Sarrate, 2009) of the many-to-many sweep technique (Blacker, 1996; Mingwu and Benzley, 1996; White et al., 2004). This technique automatically decomposes multi-source and multi-target extrusion volumes in several one-to-one extrusion volumes. Notice that in the decomposition process it is required to project inner nodes between loops of nodes without an underlying surface. To this end, the implemented multi-sweep technique uses the projection algorithm presented in Chapter 3. Once the decomposition is finished, the obtained extrusion volumes are meshed with the sweeping core presented in Chapter 4. To illustrate the multi-sweep approach, in Figure 6.11 we present the automatic decomposition in extrusion volumes and the resulting meshes for two initial geometries. Specifically, Figure 6.11(a) shows the mesh for a crankshaft model and Figure 6.11(b) for the model of a connecting rod. Note that different extrusion volumes are colored with different colors.

Submapping. The extension to 3D domains of the submapping technique allows to obtain structured hexahedral meshes in blocky geometries (White and Tautges, 2000; Whiteley et al., 1996). This method relies on a geometric decomposition of the domain into sub-volumes logically equivalent to an hexahedron. Then, each sub-volume is meshed separately using a standard structured mesh generation algorithm. EZ4U incorporates the submapping implementation detailed by Ruiz-Gironès and Sarrate (2008). Figure 6.12 shows a mesh generated for the half of a gear using the implemented submapping algorithm.



(a)



(b)

Figure 6.11: Multi-sweep meshes: **(a)** a crankshaft and **(b)** a connecting rod.

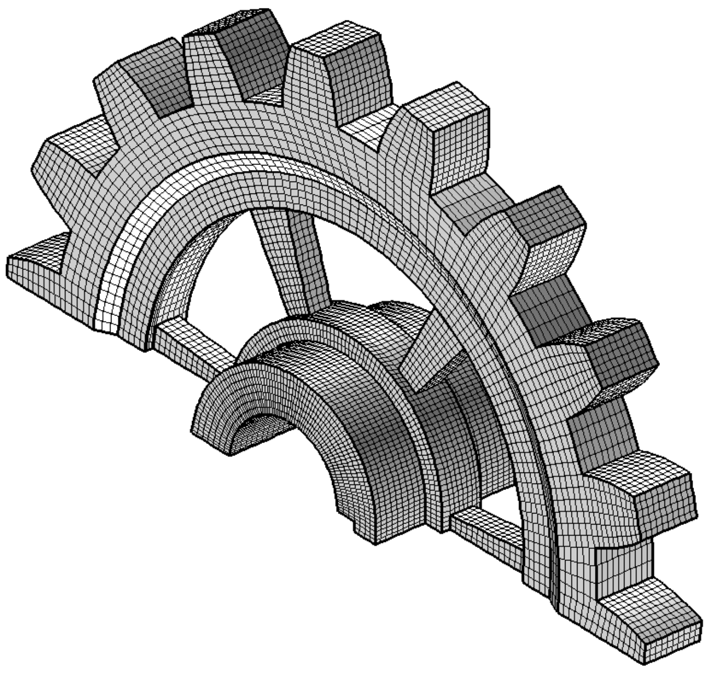


Figure 6.12: Hexahedral mesh for a half of a gear with the submapping technique by Ruiz-Gironès and Sarrate (2008).

6.8.3 Meshes with exact boundary representation and high-order meshes

The data structure of EZ4U allows converting a valid linear mesh of a domain to a mesh with exact boundary representation or with high-order nodes. If the initial linear mesh is not valid the resulting mesh can present mesh edges or mesh faces that cross the boundary surfaces of the domain.

Meshes with exact boundary representation. The main application of this type of meshes is the *NURBS-enhanced finite element method* (*NEFEM*), proposed by Sevilla et al. (2008); Sevilla (2009). This method requires a mesh where the first layer of boundary elements represents the exact CAD geometry. Thus, the NEFEM is able to reproduce with fidelity those features of the solution that depend on the geometrical properties of the domain. Figure 6.13 shows two discretizations of an aircraft model. The linear mesh approximates the model geometry, see Figure 6.13(a). On the contrary, the associated mesh with exact boundary representation reproduces the original CAD model, Figure 6.13(b).

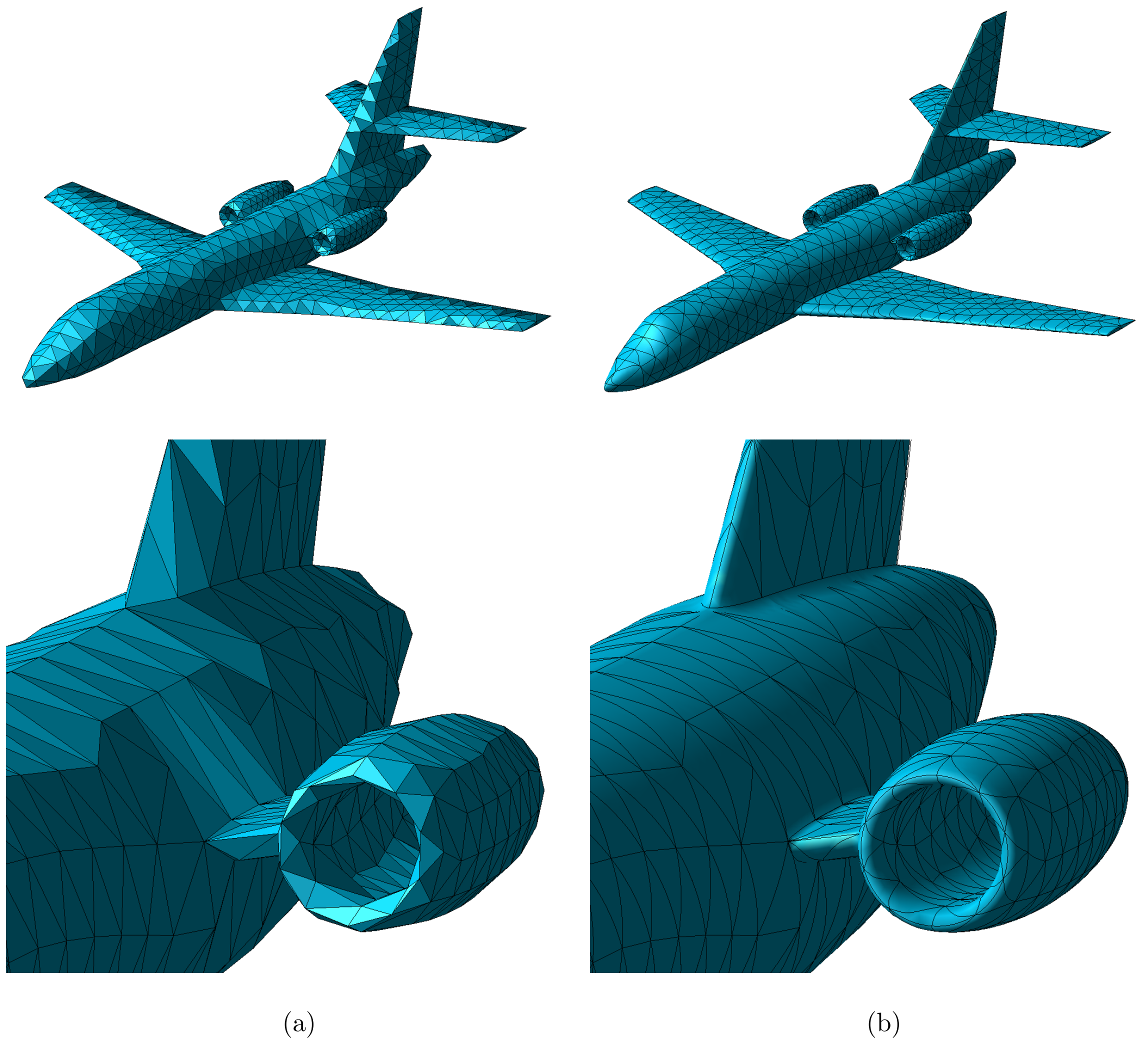


Figure 6.13: Two meshes of an airplane: **(a)** general view and detail of the linear mesh; **(b)** general view and detail of the mesh with exact boundary representation.

High-order meshes. Several numerical formulations, such as the FEM and the discontinous Galerkin (DG), need high-order discretizations. EZ4U can convert a valid linear mesh to a high-order mesh of degree $p \geq 2$. This feature generates high-order nodes according to the procedure presented in Section 6.4.3. In addition, the high-order nodes are renumbered to reduce the bandwidth of the linear system matrices of the numerical method. The implemented renumbering procedure is an extension to high-order meshes of the reverse Cuthill-McKee algorithm (Cuthill and McKee, 1969).

6.9 Concluding remarks

In this chapter we have presented an overview of the design, architecture and meshing features of a new non-commercial mesh generation framework. The design and implementation of EZ4U is one of the partial goals considered in the introduction of this dissertation. The project started in order to cover the mesh generation requirements of our research group. In addition, it is the development framework used to implement: the projection method presented in Chapter 3, the sweeping scheme proposed in Chapter 4, and the block meshing approach of Chapter 5.

The design and architecture of the framework have shown to provide code reuse, scalability, and maintainability. That is, EZ4U is a productive framework where other developers have implemented techniques not considered in this dissertation such as submapping 2D, submapping 3D, and many-to-many sweeping. Moreover, the developed data structure provides basic functionality for converting valid linear discretizations to meshes with exact boundary representation or with high-order nodes.

To summarize, we have presented a framework focused on quadrilateral and hexahedral mesh generation that provides features not available in similar non-commercial projects.

Chapter 7

Summary and future work

7.1 Summary and contributions

In this thesis we have accomplished five goals addressed to developing hexahedral mesh generation technology. We have detailed these goals in the central chapters of this dissertation. Therefore, we just summarize below the main contributions:

1. **We have proposed a new affine method for sweeping that leads to sets of normal equations of full rank**. In Chapter 2 we have presented a new functional formulation for affine methods that depends on two vector parameters. This new functional has been compared with the standard functional formulations in a detailed theoretical analysis. From this analysis, we have concluded that the new formulation overcomes the drawbacks of standard formulations. That is, the two vector parameters of the proposed functional can be chosen to obtain linear systems of full rank.

   Chapter 2 and Appendix A are an extension of the work presented in Roca, Sarrate, and Huerta (2005b). This work is accepted for publication in the *14th International Meshing Roundtable* special issue of *Engineering with Computers*.

2. **We have developed an automatic projection algorithm for sweeping that preserves offset data**. In Chapter 3 we have presented the theoretical background and the implementation of an automatic projection algorithm based on the new affine method presented in Chapter 2. The algorithm specifies the selection of the two vector parameters of the new formulation and ensures the preservation of the offset data. We have considered several tests that show

that the proposed projection algorithm overcomes all the reported drawbacks of standard affine methods.

Chapter 3 and Appendix B correspond to the work presented in Roca and Sarrate (2006). This work has been selected for publication in the *15th International Meshing Roundtable* special issue of *Engineering with Computers.*

3. **We have proposed a sweeping scheme based on affine methods**. To this end, in Chapter 4 we have detailed two projection procedures based on affine methods. First, we have proposed a fast surface mesh projection between parameterized surfaces. The projection is performed by an affine method that maps the parametric coordinates of the source surface mesh to the parametric representation of the target surface. Second, we have detailed how to generate inner layers of nodes by directly projecting from the source and target meshes. The projection from the source and the target is performed with the projection algorithm presented in Chapter 3. Notice that, the generation of the inner layers is parallelizable and preserves offset data. The numerical examples show that the presented sweeping tool can mesh a wide range of extrusion geometries.

   The contents of Chapter 4 are an update of the work by Roca, Sarrate, and Huerta (2006) published in *Communications in Numerical Methods in Engineering.*

4. **We have proposed a novel approach for block meshing by representing the geometry and the topology of a hexahedral mesh dual**. In Chapter 5 we have presented and detailed the new concept of local dual contributions. Then, we have stated a novel approach to block meshing based on the addition of local dual contributions. Given a domain without a prescribed boundary mesh, the block meshing is developed in three steps: *i)* generating a coarse reference mesh composed by tetrahedral elements; *ii)* inserting dual surfaces by addition of local dual contributions; and *iii)* obtaining final block mesh as the dual of the dual regions. The examples show that the block-meshing algorithm generates the expected block mesh for convex and non-convex blocky geometries.

   The contents of Chapter 5 are an update of the work presented in Roca and Sarrate (2008). This work has been presented in the *17th International Meshing Roundtable.*

5. **We have started and developed a new mesh generation framework**. In Chapter 6, we have presented an overview of the EZ4U framework. This framework has provided the meshing technology required to develop and implement the algorithms presented in Chapter 3, Chapter 4, and Chapter 5. In addition, EZ4U have shown to be a productive framework where developers can implement their own mesh generation algorithms.

7.2 Future work

The work carried out in this thesis leaves open some research lines that should be investigated in the near future.

1. **Sweeping**. The new affine method presented in Chapters 2 and 3 has been used to detail a new sweeping scheme in Chapter 4.

    - The proposed sweeping scheme generates inner layers of nodes by projecting directly from the cap surface meshes. On the contrary, the standard sweeping scheme computes the inner layers of nodes by projecting from the previous and next layers. Our sweeping scheme shows two advantages, it is parallelizable and it preserves offset data. However, it would be interesting to implement a standard sweeping scheme that uses our new affine method and compare it with our sweeping methodology.
    - It is planned to incorporate the proposed one-to-one sweeping scheme in a many-to-many sweeping tool (Blacker, 1996; Mingwu and Benzley, 1996; White et al., 2004). Moreover, it is also planned to develop a multi-axis sweep algorithm (Miyoshi and Blacker, 2000). First results in the many-to-many sweeping technique have been presented in Ruiz-Gironès, Roca, and Sarrate (2009). This technique automatically decomposes multi-source and multi-target extrusion volumes in several one-to-one extrusion volumes. Then the resulting volumes can be meshed with the sweeping core presented in Chapter 4.

2. **Local dual contributions and block meshing**. The local dual contributions tool is a recent concept first presented in Roca and Sarrate (2008). Moreover, obtaining a general-purpose block mesh generator is our long-term goal. In this

sense, the work presented in Chapter 5 present several issues that should be investigated and solved in the near future.

- Additional logic might be incorporated to apply the proposed approach to more complicated geometries: assembly models (contiguous domains with shared curves and surfaces) and non-blocky geometries (smooth domains without sharp features).
- We have to improve the advancing front procedure for marking faces as layers. Specifically, we have to reduce the propagation through the mesh of the cuts that start at high-valence vertices.
- It is scheduled to analyze and reduce the dependence of the algorithm from the reference mesh.
- We have to analyze if a fine reference mesh, adapted to a prescribed element size, can be used to generate hexahedral meshes by means of adding local dual contributions.
- The current approach adds local dual contributions on the potential set of all the polygon candidates of an initial dual arrangement. We have considered the possibility of first adding all the polygon candidates as hard. This is equivalent to obtain a tet-to-hex mesh. Then, this initial dual can be modified iteratively to obtain a better dual configuration according to a heuristic that controls the quality of the dual. To this end, several local dual contributions can be removed or converted to soft local dual contributions at each iteration.
- It would be interesting to develop a graphical user interface (GUI) that helps to add by hand dual surfaces represented with local dual contributions. This tool could be used to semi-automatically decompose geometries in hex-meshable sub-volumes.

3. **Mesh generation framework**. The mesh generation framework developed in this thesis, has a GUI that integrates a CAD module and several meshing algorithms. It is planned to solve several issues of the current version of the framework.

- It is scheduled to improve the usability of the current GUI.
- The current version of EZ4U is only interfaced with the Open Cascade library. This library provides boundary representation (B-rep) geometry. However, it is frequently required to mesh geometries described with a faceted representation. For instance, legacy geometries are described in the faceted STL format. Therefore, it would be interesting to incorporate a faceted geometry engine. To this end, we have planned to integrate a virtual geometry engine that encapsulates both boundary and faceted geometry representations.
- The current data structure allows converting a linear mesh to a mesh with exact boundary representation or with high-order nodes. The resulting mesh can present edges and faces that cross the model boundaries. To solve this drawback it is required to generate finer meshes. However, mesh based numerical methods that use these types of meshes fully advocate for the use of coarse meshes. Therefore, we have planned to investigate how to generate valid coarse meshes with exact boundary representation or with high-order nodes.
- The development of EZ4U has been focused on the implementation of quadrilateral and hexahedral mesh generation algorithms. Thus, it is scheduled to improve the performance and capabilities of the incorporated triangular and tetrahedral mesh generation features.

Appendix A

Proofs of Chapter 2

A.1 Linear algebra results

The next three lemmas are linear algebra results needed to develop the proofs of results in Chapter 2.

Lemma A.1. *Let* $X = \{\mathbf{x}^i\}_{i=1,\dots,m} \subset \mathbb{R}^n$ *be a hyperplanar set of points. There exists a set of indices* $\{i_1, \dots, i_n\} \subset \{1, \dots, m\}$ *such that*

$$\mathbb{H} = \operatorname{span}(\mathbf{x}^{i_1} - \mathbf{x}^{i_n}, \dots, \mathbf{x}^{i_{n-1}} - \mathbf{x}^{i_n}) = \operatorname{span}(\mathbf{x}^{i_1} - \mathbf{c}^X, \dots, \mathbf{x}^{i_{n-1}} - \mathbf{c}^X).$$

Proof. Since X is hyperplanar, the affine hyper-plane $\mathbf{x}^m + \mathbb{H}$ passes through all the points in X. Therefore, the homogeneous hyperplane verifies

$$\mathbb{H} = \operatorname{span}(\mathbf{x}^1 - \mathbf{x}^m, \dots, \mathbf{x}^{m-1} - \mathbf{x}^m).$$

Taking into account that $\mathbb{H}$ has dimension $n-1$ and that $\mathbf{c}^X$ lies in the hyperplane determined by the points in X, see Roca et al. (2005b), it is straightforward to show that there exists a set of indices $\{i_1, \dots, i_n\} \subset \{1, \dots, m\}$ such that

$$\begin{aligned}\mathbb{H} &= \operatorname{span}(\mathbf{x}^1 - \mathbf{x}^m, \dots, \mathbf{x}^{m-1} - \mathbf{x}^m)\\ &= \operatorname{span}(\mathbf{x}^{i_1} - \mathbf{x}^{i_n}, \dots, \mathbf{x}^{i_{n-1}} - \mathbf{x}^{i_n})\\ &= \operatorname{span}(\mathbf{x}^{i_1} - \mathbf{c}^X, \dots, \mathbf{x}^{i_{n-1}} - \mathbf{c}^X)\end{aligned}$$

$\square$

Lemma A.2. *Let $U = \{\mathbf{u}^i\}_{i=1,\dots,n}$ be a set of vectors in $\mathbb{R}^n$ such that:*

$$\dim\{\text{span}(\mathbf{u}^1,\dots,\mathbf{u}^{n-1})\} = \dim\{\text{span}(\mathbf{u}^1,\dots,\mathbf{u}^{n-1},\mathbf{u}^n)\} =$$
$$\dim\{\text{span}(\mathbf{u}^1-\mathbf{u}^n,\dots,\mathbf{u}^{n-1}-\mathbf{u}^n)\} = n-1^{\cdot}$$

If $\mathbf{v}\in\mathbb{R}^n$ is such that $\mathbf{v}\notin \text{span}(\mathbf{u}^1,\dots,\mathbf{u}^{n-1})$ then

$$\text{span}(\mathbf{u}^1,\dots,\mathbf{u}^{n-1},\mathbf{v}) = \text{span}(\mathbf{u}^1+\mathbf{v},\dots,\mathbf{u}^n+\mathbf{v}).$$

Proof. Since $\mathbf{u}^n \in \text{span}(\mathbf{u}^1,\dots,\mathbf{u}^{n-1})$, $\mathbf{u}_n$ can be written as linear combination of the other vectors. In other words,

$$\mathbf{u}^n = \sum_{i=1}^{n-1}\lambda^i\mathbf{u}^i, \qquad \lambda^i\in\mathbb{R}, \quad i=1,\dots,n-1. \tag{A.1}$$

For later use in this proof, we will first prove that $\sum_{i=1}^{n-1}\lambda^i \neq 1$. Assume that $\sum_{i=1}^{n-1}\lambda^i = 1$, then we can write

$$\sum_{i=1}^{n-1}\lambda^i\mathbf{u}^n = \sum_{i=1}^{n-1}\lambda^i\mathbf{u}^i,$$

grouping terms we obtain

$$\sum_{i=1}^{n-1}\lambda^i(\mathbf{u}^i-\mathbf{u}^n) = \mathbf{0}.$$

Hence, we have a non-trivial linear combination of $\mathbf{u}^i-\mathbf{u}^n$, $i=1,\dots,n-1$, which is the null vector. That is, $\dim\{\text{span}(\mathbf{u}^1-\mathbf{u}^n,\dots,\mathbf{u}^{n-1}-\mathbf{u}^n)\} < n-1$. On the contrary, by hypothesis of this Lemma, we have that $\dim\{\text{span}(\mathbf{u}^1-\mathbf{u}^n,\dots,\mathbf{u}^{n-1}-\mathbf{u}^n)\} = n-1$. Hence, we conclude that $\sum_{i=1}^{n-1}\lambda^i \neq 1$.

Taking into account this partial result, and starting with the $\text{span}(\mathbf{u}^1+\mathbf{v},\dots,\mathbf{u}^n+\mathbf{v})$, we will prove the lemma. If we subtract the linear combination of the first $n-1$ vectors of the span $\sum_{i=1}^{n-1}\lambda^i(\mathbf{u}^i+\mathbf{v})$ to the last term, then

$$\text{span}(\mathbf{u}^1+\mathbf{v},\dots,\mathbf{u}^n+\mathbf{v}) = \text{span}\Big(\mathbf{u}^1+\mathbf{v},\dots,\mathbf{u}^{n-1}+\mathbf{v},\mathbf{u}^n-\sum_{i=1}^{n-1}\lambda^i\mathbf{u}^i+\Big(1-\sum_{i=1}^{n-1}\lambda^i\Big)\mathbf{v}\Big).$$

Moreover, using equation (A.1) we have that

$$\text{span}(\mathbf{u}^1+\mathbf{v},\dots,\mathbf{u}^n+\mathbf{v}) = \text{span}\Big(\mathbf{u}^1+\mathbf{v},\dots,\mathbf{u}^{n-1}+\mathbf{v},\Big(1-\sum_{i=1}^{n-1}\lambda^i\Big)\mathbf{v}\Big).$$

Finally, dividing by $(1 - \sum_{i=1}^{n-1} \lambda^i) \neq 0$ the last term of the previous equation, and then subtracting $\mathbf{v}$ to the first $n-1$ terms we have

$$\operatorname{span}(\mathbf{u}^1 + \mathbf{v}, \dots, \mathbf{u}^n + \mathbf{v}) = \operatorname{span}(\mathbf{u}^1, \dots, \mathbf{u}^{n-1}, \mathbf{v}). \qquad \square$$

A.2 Properties of functionals

Lemma A.3 states several basic relationships between the functionals E, F, G, and H.

Lemma A.3. *For every* $\mathbf{A} \in \mathcal{L}(\mathbb{R}^n)$*:*

(i) $E(\mathbf{A}, \mathbf{c}^Y) = F(\mathbf{A})$

(ii) $E\big(\mathbf{A}, \mathbf{c}^X + \mathbf{A}(\mathbf{c}^Y - \mathbf{c}^X)\big) = G(\mathbf{A})$

(iii) $E(\mathbf{A}, \mathbf{c}^Y) = H(\mathbf{A}; \mathbf{u}^X, \mathbf{A}\mathbf{u}^X)$

Proof. The proof follows from the definitions of functionals E, F, G, and H. $\square$

The following three lemmas prove that functionals F, G and H can be written in terms of linear mapping $\mathbf{\Theta}$.

Lemma A.4. *Let X be a hyperplanar set of points, and assume that $\mathbf{u}^X \notin \mathbb{H}$ and $\mathbf{u}^Y \in \mathbb{R}^n$. If $\mathbf{A} \in \mathcal{L}(\mathbb{R}^n)$ then $F\big(\mathbf{\Theta}[\mathbf{A}, \mathbf{u}^X, \mathbf{u}^Y]\big) = F(\mathbf{A})$.*

Proof. Since X is hyperplanar, $\mathbf{x}^i - \mathbf{c}^X \in \mathbb{H}$, for $i = 1, \dots, m$. Therefore, by property *(ii)* of Lemma 2.8 we have that $\mathbf{\Theta}[\mathbf{A}, \mathbf{u}^X, \mathbf{u}^Y](\mathbf{x}^i - \mathbf{c}^X) = \mathbf{A}(\mathbf{x}^i - \mathbf{c}^X)$, for $i = 1, \dots, m$. The proof follows from the definition of the functional F, see Equation (2.7). $\square$

Lemma A.5. *Let X be a hyperplanar set of points, and assume that $\mathbf{c}^Y - \mathbf{c}^X \notin \mathbb{H}$. If $\mathbf{A} \in \mathcal{L}(\mathbb{R}^n)$ then:*

(i) $F\big(\mathbf{\Theta}[\mathbf{A}, \mathbf{c}^Y - \mathbf{c}^X, \mathbf{c}^Y - \mathbf{c}^X]\big) = F\big(\mathbf{A}\big)$

(ii) $G\big(\mathbf{\Theta}[\mathbf{A}, \mathbf{c}^Y - \mathbf{c}^X, \mathbf{c}^Y - \mathbf{c}^X]\big) = F\big(\mathbf{\Theta}[\mathbf{A}, \mathbf{c}^Y - \mathbf{c}^X, \mathbf{c}^Y - \mathbf{c}^X]\big)$

Proof. Assume that $\mathbf{u}^X = \mathbf{c}^Y - \mathbf{c}^X$ and $\mathbf{u}^Y = \mathbf{c}^Y - \mathbf{c}^X$. Then, property *(i)* is a particular case of Lemma A.4. Property *(ii)* follows from the definition of functional G, see Equation (2.8), and Lemma 2.8. $\square$

Lemma A.6. *Let X be a hyperplanar set of points, and assume that $\mathbf{u}^X \notin \mathbb{H}$ and $\mathbf{u}^Y \in \mathbb{R}^n$. Then $H\big(\mathbf{\Theta}[\mathbf{A}, \mathbf{u}^X, \mathbf{u}^Y]; \mathbf{u}^X, \mathbf{u}^Y\big) = F\big(\mathbf{\Theta}[\mathbf{A}, \mathbf{u}^X, \mathbf{u}^Y]\big)$.*

Proof. This result follows from the definitions of functionals F and H, and Lemma 2.8. □

A.3 Rank analysis

In this section we detail the proofs for propositions 2.13, and 2.15 of Section 2.3.2 in Chapter 2.

Proposition (2.13)**.** *Let X be a hyperplanar set of points. If $\mathbf{c}^Y - \mathbf{c}^X \in \mathbb{H}$ then the minimization of functional G is equivalent to solving n uncoupled overdetermined linear systems of rank $n-1$. Otherwise, the rank is n.*

Proof. Similar to Proposition 2.11, if we define

$$\overline{\overline{\mathbf{X}}} := \begin{pmatrix} x_1^1 - c_1^X + c_1^Y - c_1^X & \dots & x_1^m - c_1^X + c_1^Y - c_1^X \\ \vdots & & \vdots \\ x_n^1 - c_n^X + c_n^Y - c_n^X & \dots & x_n^m - c_n^X + c_n^Y - c_n^X \end{pmatrix},$$

then the minimization of functional G is equivalent to solving the following n uncoupled overdetermined linear systems

$$\overline{\overline{\mathbf{X}}}^T \mathbf{a}_k = \overline{\mathbf{y}}_k, \quad k = 1, \cdots, n, \tag{A.2}$$

where $\mathbf{a}_k := (a_{k,j})$ for $j = 1, \cdots, n$ and $\overline{\mathbf{y}}_k = (y_k^l - c_k^X)$, for $l = 1, \cdots, m$. Since the set of points X is hyperplanar, by Lemma A.1, we can assume that we have already reordered the points in X in such a way that $\dim\{\text{span}(\mathbf{x}^1 - \mathbf{c}^X, \dots, \mathbf{x}^{n-1} - \mathbf{c}^X)\} = \dim\{\text{span}(\mathbf{x}^1 - \mathbf{x}^n, \dots, \mathbf{x}^{n-1} - \mathbf{x}^n)\} = n - 1$. If we apply Lemma A.2 considering $\mathbf{v} = \mathbf{c}^Y - \mathbf{c}^X$, then

$$\begin{aligned} \text{rank}(\overline{\overline{\mathbf{X}}}) &= \dim\big\{\text{span}(\mathbf{x}^1 - \mathbf{c}^X + \mathbf{c}^Y - \mathbf{c}^X, \dots, \mathbf{x}^n - \mathbf{c}^X + \mathbf{c}^Y - \mathbf{c}^X)\big\} \\ &= \dim\big\{\text{span}(\mathbf{x}^1 - \mathbf{c}^X, \dots, \mathbf{x}^{n-1} - \mathbf{c}^X, \mathbf{c}^Y - \mathbf{c}^X)\big\}. \end{aligned}$$

Therefore, if $\mathbf{c}^Y - \mathbf{c}^X \in \mathbb{H}$ then $\text{rank}(\overline{\overline{\mathbf{X}}}) = n - 1$. Otherwise, if $\mathbf{c}^Y - \mathbf{c}^X \notin \mathbb{H}$ we have $\text{rank}(\overline{\overline{\mathbf{X}}}) = n$. □

Proposition (2.15)**.** *Let X be a hyperplanar set of points and assume that $\mathbf{u}^Y \in \mathbb{R}^n$. If $\mathbf{u}^X \in \mathbb{H}$ then the minimization of functional H is equivalent to solving n uncoupled overdetermined linear systems of rank $n-1$. Otherwise, the rank is n.*

Proof. Similar to Proposition 2.11, if we define

$$\hat{\mathbf{X}} := \begin{pmatrix} x_1^1 - c_1^X & \dots & x_1^m - c_1^X & u_1^X \\ \vdots & & \vdots & \vdots \\ x_n^1 - c_n^X & \dots & x_n^m - c_n^X & u_n^X \end{pmatrix}$$

and

$$\hat{\mathbf{Y}} := \begin{pmatrix} y_1^1 - c_1^Y & \dots & y_1^m - c_1^Y & u_1^Y \\ \vdots & & \vdots & \vdots \\ y_n^1 - c_n^Y & \dots & y_n^m - c_n^Y & u_n^Y \end{pmatrix},$$

then the minimization of functional H is equivalent to solving the following n uncoupled overdetermined linear systems

$$\hat{\mathbf{X}}^T \mathbf{a}_k = \hat{\mathbf{y}}_k, \quad k = 1, \cdots, n, \tag{A.3}$$

where $\mathbf{a}_k := (a_{k,j})$ for $j = 1, \cdots, n$ and $\hat{\mathbf{y}}_k = (y_k^1 - c_k^Y, \cdots, y_k^m - c_k^Y, u_k^Y)$. Since X is hyperplanar, by Lemma A.1, we can assume that we have already reordered the points in X in such a way that $\dim\{\text{span}(\mathbf{x}^1 - \mathbf{c}^X, \dots, \mathbf{x}^{n-1} - \mathbf{c}^X)\} = \dim\{\text{span}(\mathbf{x}^1 - \mathbf{x}^n, \dots, \mathbf{x}^{n-1} - \mathbf{x}^n)\} = n-1$. If $\mathbf{u}^X \in \mathbb{H}$ then $\text{rank}\,\hat{\mathbf{X}} = \dim\{\text{span}(\mathbf{x}^1 - \mathbf{c}^X, \dots, \mathbf{x}^{n-1} - \mathbf{c}^X, \mathbf{u}^X)\} = n-1$. Otherwise, $\mathbf{u}^X \notin \mathbb{H}$, we have that $\text{rank}\,\hat{\mathbf{X}} = \dim\{\text{span}(\mathbf{x}^1 - \mathbf{c}^X, \dots, \mathbf{x}^{n-1} - \mathbf{c}^X, \mathbf{u}^X)\} = n$. □

A.4 Equivalences between functionals

In this section we detail the proofs for propositions 2.17, 2.19, 2.21, and 2.23 of Section 2.3.3 in Chapter 2.

Proposition (2.17)**.** *Let $\mathbf{A}^E \in \mathcal{L}(\mathbb{R}^n)$, $\mathbf{b}^E \in \mathbb{R}^n$ and $\mathbf{A}^F \in \mathcal{L}(\mathbb{R}^n)$ be such that*

$$E(\mathbf{A}^E, \mathbf{b}^E) = \min_{(\mathbf{A},\mathbf{b}) \in \mathcal{L}(\mathbb{R}^n) \times \mathbb{R}^n} E(\mathbf{A}, \mathbf{b})$$

$$F(\mathbf{A}^F) = \min_{\mathbf{A} \in \mathcal{L}(\mathbb{R}^n)} F(\mathbf{A}).$$

Then:

(i) $\min_{(\mathbf{A},\mathbf{b})\in\mathcal{L}(\mathbb{R}^n)\times\mathbb{R}^n} E(\mathbf{A},\mathbf{b}) = \min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} F(\mathbf{A})$

(ii) $E(\mathbf{A}^F,\mathbf{c}^Y) = E(\mathbf{A}^E,\mathbf{c}^Y)$

(iii) $F(\mathbf{A}^E) = F(\mathbf{A}^F)$

Proof. We have that

$$\begin{aligned}
F(\mathbf{A}^F) &\le F(\mathbf{A}^E) && \text{since } \mathbf{A}^F \text{ minimizes } F\\
&= E(\mathbf{A}^E,\mathbf{c}^Y) && \text{by Lemma } A.3\\
&\le E(\mathbf{A}^F,\mathbf{c}^Y) && \text{since } (\mathbf{A}^E,\mathbf{c}^Y) \text{ minimizes } E \text{ by Proposition 2.2}\\
&= F(\mathbf{A}^F) && \text{by Lemma } A.3.
\end{aligned}$$

Note that the first and the last terms in the previous expression are the same. Then, the inequalities are in fact equalities. All properties follow from rewriting and reordering the chain of equalities. □

Proposition (2.19)**.** *Let X be a hyperplanar set of points, and assume that $\mathbf{c}^Y - \mathbf{c}^X \notin \mathbb{H}$. If $\mathbf{A}^F \in \mathcal{L}(\mathbb{R}^n)$ and $\mathbf{A}^G \in \mathcal{L}(\mathbb{R}^n)$ are such that*

$$F(\mathbf{A}^F) = \min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} F(\mathbf{A})$$

$$G(\mathbf{A}^G) = \min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} G(\mathbf{A}).$$

Then:

(i) $\min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} G(\mathbf{A})$ *has one and only one solution*

(ii) $\min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} F(\mathbf{A}) = \min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} G(\mathbf{A})$

(iii) $\mathbf{A}^G = \mathbf{\Theta}[\mathbf{A}^F, \mathbf{c}^Y - \mathbf{c}^X, \mathbf{c}^Y - \mathbf{c}^X]$

(iv) $F(\mathbf{A}^G) = F(\mathbf{A}^F)$

Proof. Property *(i)* follows from Proposition 2.13 and Remark 2.14. Assume we have $(\mathbf{A}^E,\mathbf{b}^E) \in \mathcal{L}(\mathbb{R}^n)\times\mathbb{R}^n$ such that $E(\mathbf{A}^E,\mathbf{b}^E) = \min_{(\mathbf{A},\mathbf{b})\in\mathcal{L}(\mathbb{R}^n)\times\mathbb{R}^n} E(\mathbf{A},\mathbf{b})$. Then:

$$\begin{aligned}
F(\mathbf{A}^F) &= E\big(\mathbf{A}^E,\mathbf{b}^E\big) && \text{by Proposition 2.17}\\
&\le E\big(\mathbf{A}^G, \mathbf{c}^X + \mathbf{A}^G(\mathbf{c}^Y - \mathbf{c}^X)\big) && (\mathbf{A}^E,\mathbf{b}^E) \text{ minimizes } E\\
&= G\big(\mathbf{A}^G\big) && \text{by Lemma A.3}\\
&\le G\big(\mathbf{\Theta}[\mathbf{A}^F, \mathbf{c}^Y - \mathbf{c}^X, \mathbf{c}^Y - \mathbf{c}^X]\big) && \text{since } \mathbf{A}^G \text{ minimizes } G\\
&= F\big(\mathbf{\Theta}[\mathbf{A}^F, \mathbf{c}^Y - \mathbf{c}^X, \mathbf{c}^Y - \mathbf{c}^X]\big) = F(\mathbf{A}^F) && \text{by Lemma } A.5.
\end{aligned}$$

Note that the first and the last terms in the previous expression are the same. Then, the inequalities are in fact equalities, and property *(ii)* follows. From the previous sequence of equalities, we have proved that $\mathbf{\Theta}[\mathbf{A}^F, \mathbf{c}^Y - \mathbf{c}^X, \mathbf{c}^Y - \mathbf{c}^X]$ and $\mathbf{A}^G$ minimize functional G. Thus, property *(iii)* is also proved since G has a unique solution. Property *(iv)* follows from Lemma A.5 and property *(iii)* of this Proposition. □

Proposition (2.21)**.** *Let X be a hyperplanar set of points, and assume that $\mathbf{u}^X \notin \mathbb{H}$ and $\mathbf{u}^Y \in \mathbb{R}^n$. If $\mathbf{A}^F \in \mathcal{L}(\mathbb{R}^n)$ and $\mathbf{A}^H \in \mathcal{L}(\mathbb{R}^n)$ are such that*

$$F(\mathbf{A}^F) = \min_{\mathbf{A} \in \mathcal{L}(\mathbb{R}^n)} F(\mathbf{A})$$

$$H(\mathbf{A}^H; \mathbf{u}^X, \mathbf{u}^Y) = \min_{\mathbf{A} \in \mathcal{L}(\mathbb{R}^n)} H(\mathbf{A}; \mathbf{u}^X, \mathbf{u}^Y).$$

Then:

(i) $\min_{\mathbf{A} \in \mathcal{L}(\mathbb{R}^n)} H(\mathbf{A}; \mathbf{u}^X, \mathbf{u}^Y)$ *has one and only one solution*

(ii) $\min_{\mathbf{A} \in \mathcal{L}(\mathbb{R}^n)} F(\mathbf{A}) = \min_{\mathbf{A} \in \mathcal{L}(\mathbb{R}^n)} H(\mathbf{A}; \mathbf{u}^X, \mathbf{u}^Y)$

(iii) $\mathbf{A}^H = \mathbf{\Theta}[\mathbf{A}^F, \mathbf{u}^X, \mathbf{u}^Y]$

(iv) $F(\mathbf{A}^H) = F(\mathbf{A}^F)$

Proof. Property *(i)* follows from Proposition 2.15 and Remark 2.16. We define $R(\mathbf{A}; \mathbf{u}^X, \mathbf{u}^Y) := \|\mathbf{u}^Y - \mathbf{A}\mathbf{u}^X\|^2$. Hence, $H(\mathbf{A}; \mathbf{u}^X, \mathbf{u}^Y) = F(\mathbf{A}) + R(\mathbf{A}; \mathbf{u}^X, \mathbf{u}^Y)$. We consider the following sequence of equalities and inequalities:

$$\begin{aligned}
F(\mathbf{A}^F) &\leq F(\mathbf{A}^H) && \text{since } \mathbf{A}^F \text{minimizes F} \\
&\leq F(\mathbf{A}^H) + R(\mathbf{A}^H; \mathbf{u}^X, \mathbf{u}^Y) && \text{since } R(\mathbf{A}; \mathbf{u}^X, \mathbf{u}^Y) \geq 0 \\
&= H(\mathbf{A}^H; \mathbf{u}^X, \mathbf{u}^Y) && \text{definitions of } R(\mathbf{A}; \mathbf{u}^X, \mathbf{u}^Y) \text{ and } H \\
&\leq H\big(\mathbf{\Theta}[\mathbf{A}^F, \mathbf{u}^X, \mathbf{u}^Y]; \mathbf{u}^X, \mathbf{u}^Y\big) && \text{since } \mathbf{A}^H \text{ minimizes } H \\
&= F\big(\mathbf{\Theta}[\mathbf{A}^F, \mathbf{u}^X, \mathbf{u}^Y]\big) = F(\mathbf{A}^F). && \text{by Lemmas } A.6 \text{ and } A.4.
\end{aligned}$$

Note that the first and the last terms are the same. Thus, all the inequalities are in fact equalities. Therefore, properties *(ii)* and *(iv)* hold. From the previous sequence of equalities, we have also proved that $\mathbf{\Theta}[\mathbf{A}^F, \mathbf{u}^X, \mathbf{u}^Y]$ and $\mathbf{A}^H$ minimize the functional H. Since the minimization of H has a unique solution we have that $\mathbf{A}^H = \mathbf{\Theta}[\mathbf{A}^F, \mathbf{u}^X, \mathbf{u}^Y]$, and property *(iii)* holds. □

Proposition (2.23)**.** *Let X be a non-hyperplanar set of points and assume that $\mathbf{u}^Y \in \mathbb{R}^n$. If $\mathbf{A}^F \in \mathcal{L}(\mathbb{R}^n)$ and $\mathbf{A}^H \in \mathcal{L}(\mathbb{R}^n)$ are such that*

$$F(\mathbf{A}^F) = \min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} F(\mathbf{A})$$

$$H(\mathbf{A}^H; \mathbf{0}, \mathbf{u}^Y) = \min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} H(\mathbf{A}; \mathbf{0}, \mathbf{u}^Y).$$

Then:

(i) $\min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} F(\mathbf{A}) = \min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} H(\mathbf{A}; \mathbf{0}, \mathbf{u}^Y) - \|\mathbf{u}^Y\|^2$

(ii) $F(\mathbf{A}^H) = F(\mathbf{A}^F)$

Proof. Observe that, from the definitions of functionals F and H we have that $H(\mathbf{A}; \mathbf{0}, \mathbf{u}^Y) = F(\mathbf{A}) + \|\mathbf{u}^Y\|^2$. In other words, H differs from F only by a constant that does not depend on $\mathbf{A}$. Hence, both properties follow. □

A.5 Exact mapping characterization

In this section we detail the proofs for propositions 2.25, 2.26, and 2.27 of Section 2.3.4 in Chapter 2.

Proposition (2.25)**.** *There exists an affine mapping $\boldsymbol{\varphi}$ such that $\boldsymbol{\varphi}(\mathbf{x}^i) = \mathbf{y}^i$, for $i = 1, \ldots, m$, if and only if*

$$\min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} F(\mathbf{A}) = 0.$$

Proof. Assume that there exists an affine mapping $\boldsymbol{\varphi}$ such that $\boldsymbol{\varphi}(\mathbf{x}^i) = \mathbf{y}^i$ for $i = 1, \ldots, m$. Taking into account Equation (2.4) and Proposition 2.2, this is equivalent to imposing $\mathbf{y}^i - \mathbf{c}^Y - \mathbf{A}(\mathbf{x}^i - \mathbf{c}^X) = \mathbf{0}$ for $i = 1, \ldots, m$. Thus $F(\mathbf{A}) = 0$.

On the other hand, assume that $\min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} F(\mathbf{A}) = 0$. In this case there exists a linear transformation $\mathbf{A}^F \in \mathcal{L}(\mathbb{R}^n)$ such that $F(\mathbf{A}^F) = 0$. From the definition of F we have that $\sum_{i=1}^m \left\|\mathbf{y}^i - \mathbf{c}^Y - \mathbf{A}^F(\mathbf{x}^i - \mathbf{c}^X)\right\|^2 = 0$. Since all the summation members are non-negative, and defining $\boldsymbol{\varphi}(\mathbf{x}) := \mathbf{A}^F(\mathbf{x} - \mathbf{c}^X) + \mathbf{c}^Y$, we have that $\boldsymbol{\varphi}(\mathbf{x}^i) = \mathbf{y}^i$, for $i = 1, \ldots, m$. □

Proposition (2.26)**.** *Assume that X is hyperplanar and $\mathbf{c}^Y - \mathbf{c}^X \notin \mathbb{H}$. Then, there exists an affine mapping $\boldsymbol{\varphi}$ such that $\boldsymbol{\varphi}(\mathbf{x}^i) = \mathbf{y}^i$, for $i = 1, \ldots, m$, if and only if*

$$\min_{\mathbf{A}\in\mathcal{L}(\mathbb{R}^n)} G(\mathbf{A}) = 0.$$

Proof. This result follows directly from property *(ii)* of Proposition 2.19 and Proposition 2.25. □

Proposition (2.27)**.** *There exists an affine mapping $\boldsymbol{\varphi}$ such that $\boldsymbol{\varphi}(\mathbf{x}^i) = \mathbf{y}^i$, for $i = 1, \ldots, m$, if and only if there exist $\mathbf{u}^X, \mathbf{u}^Y \in \mathbb{R}^n$ such that*

$$\min_{\mathbf{A} \in \mathcal{L}(\mathbb{R}^n)} H(\mathbf{A}; \mathbf{u}^X, \mathbf{u}^Y) = 0.$$

Proof. It is proved analogously to Proposition 2.25. □

Appendix B

Proofs of Chapter 3

B.1 Pseudo-normal of a loop of nodes

Proposition (3.4). *Let $X = \{\mathbf{x}^i\}_{i=1,\ldots,m} \subset \mathbb{R}^3$ be a set of points. The pseudo-area vector verifies:*

(i) Given $\mathbf{c} \in \mathbb{R}^3$ then

$$\mathbf{a}^X_{pseudo} = \frac{1}{2}\sum_{i=1}^{m}(\mathbf{x}^i - \mathbf{c}) \times (\mathbf{x}^{i+1} - \mathbf{c}) = \frac{1}{2}\sum_{i=1}^{m}\mathbf{x}^i \times \mathbf{x}^{i+1}.$$

(ii) Given $\mathbf{t} \in \mathbb{R}^3$ the pseudo-area of X is equal to the pseudo-area of $X + \mathbf{t} = \{\mathbf{x}^i + \mathbf{t}\}_{i=1,\ldots,m}$.

(iii) Given an orthogonal transformation $\mathbf{N}$, then the pseudo-area of $\mathbf{N}X = \{\mathbf{N}\mathbf{x}^i\}_{i=1,\ldots,m}$ is $\mathbf{N}\mathbf{a}^X_{pseudo}$.

Proof. Given $\mathbf{c} \in \mathbb{R}^3$, and taking into account that X is a loop, i.e. $\mathbf{x}^{m+1} \equiv \mathbf{x}^1$, then

$$\begin{aligned}
\mathbf{a}^X_{pseudo} &= \frac{1}{2}\sum_{i=1}^{m}(\mathbf{x}^i - \mathbf{c}) \times (\mathbf{x}^{i+1} - \mathbf{c}) \\
&= \frac{1}{2}\left[\sum_{i=1}^{m}\mathbf{x}^i \times \mathbf{x}^{i+1} + \sum_{i=1}^{m}\mathbf{c} \times (\mathbf{x}^i - \mathbf{x}^{i+1}) + \sum_{i=1}^{m}\mathbf{c} \times \mathbf{c}\right] \\
&= \frac{1}{2}\sum_{i=1}^{m}\mathbf{x}^i \times \mathbf{x}^{i+1}.
\end{aligned}$$

Given $\mathbf{t} \in \mathbb{R}^3$, property *(ii)* is a direct consequence of property *(i)* applied to $\mathbf{c} = -\mathbf{t}$.

By property *(i)* and taking into account that $\mathbf{N}$ is orthogonal we have that

$$\mathbf{a}^{\mathbf{N}X}_{pseudo} = \sum_{i=1}^{m} \mathbf{N}\mathbf{x}^i \times \mathbf{N}\mathbf{x}^{i+1} \;=\; \sum_{i=1}^{m} \mathbf{N}(\mathbf{x}^i \times \mathbf{x}^{i+1}) \;=\; \mathbf{N}\mathbf{a}^X_{pseudo}. \qquad \square$$

Proposition (3.5)**.** *If a multi-loop X is projected on an orthogonal plane to its pseudo-area vector, $\mathbf{a}^X_{pseudo}$, then the obtained polygon has area equal to $||\mathbf{a}^X_{pseudo}||$.*

Proof. By definition of pseudo-area of a multi-loop, it suffices to prove the result for a single loop. Given a loop X, the projection of the points $\{\mathbf{x}^i\}_{i=1,\dots,m}$ on the orthogonal plane to $\mathbf{a}^X_{pseudo}$ are the points $\mathbf{x}^i_{\mathbf{a}^\perp} := \mathbf{x}^i - \mathbf{x}^i_{\mathbf{a}}$, where

$$\mathbf{x}^i_{\mathbf{a}} := \frac{< \mathbf{x}^i, \mathbf{a}^X_{pseudo} >}{< \mathbf{a}^X_{pseudo}, \mathbf{a}^X_{pseudo} >} \mathbf{a}^X_{pseudo}.$$

Hence, we have that $\mathbf{x}^i = \mathbf{x}^i_{\mathbf{a}} + \mathbf{x}^i_{\mathbf{a}^\perp}$ for $i = 1, \dots, m$. The pseudo-area of X is

$$\begin{aligned}\mathbf{a}^X_{pseudo} &= \frac{1}{2}\sum_{i=1}^{m} \mathbf{x}^i \times \mathbf{x}^{i+1} \;=\; \frac{1}{2}\sum_{i=1}^{m} (\mathbf{x}^i_{\mathbf{a}} + \mathbf{x}^i_{\mathbf{a}^\perp}) \times (\mathbf{x}^{i+1}_{\mathbf{a}} + \mathbf{x}^{i+1}_{\mathbf{a}^\perp}) \\ &= \frac{1}{2}\sum_{i=1}^{m} \mathbf{x}^i_{\mathbf{a}} \times \mathbf{x}^{i+1}_{\mathbf{a}^\perp} + \mathbf{x}^i_{\mathbf{a}^\perp} \times \mathbf{x}^{i+1}_{\mathbf{a}} + \mathbf{x}^i_{\mathbf{a}^\perp} \times \mathbf{x}^{i+1}_{\mathbf{a}^\perp}\end{aligned}$$

Note that $\mathbf{x}^i_{\mathbf{a}} \times \mathbf{x}^{i+1}_{\mathbf{a}^\perp}$ and $\mathbf{x}^i_{\mathbf{a}^\perp} \times \mathbf{x}^{i+1}_{\mathbf{a}}$ are orthogonal to $\mathbf{a}^X_{pseudo}$. Thus, their sum is also orthogonal to $\mathbf{a}^X_{pseudo}$. Furthermore, $\mathbf{x}^i_{\mathbf{a}^\perp} \times \mathbf{x}^{i+1}_{\mathbf{a}^\perp}$ is parallel to $\mathbf{a}^X_{pseudo}$. Therefore, $\mathbf{x}^i_{\mathbf{a}} \times \mathbf{x}^{i+1}_{\mathbf{a}^\perp} + \mathbf{x}^i_{\mathbf{a}^\perp} \times \mathbf{x}^{i+1}_{\mathbf{a}}$ cannot contribute to the parallel component to $\mathbf{a}^X_{pseudo}$ and it has to be null. Hence, we have that

$$\mathbf{a}^X_{pseudo} = \frac{1}{2}\sum_{i=1}^{m} \mathbf{x}^i_{\mathbf{a}^\perp} \times \mathbf{x}^{i+1}_{\mathbf{a}^\perp},$$

which is the area of the polygon obtained from the projection of the loop X on the plane orthogonal to $\mathbf{a}^X_{pseudo}$. $\square$

Lemma B.1. *Let $\mathbf{v} \in \mathbb{R}^3$ be a given vector with $||\mathbf{v}|| = 1$. Let $\mathbf{x} \in \mathbb{R}^3$ and $\mathbf{y} \in \mathbb{R}^3$ be two arbitrary vectors. We decompose these vectors as $\mathbf{x} = \mathbf{x}_\mathbf{v} + \mathbf{x}_{\mathbf{v}^\perp}$ and $\mathbf{y} = \mathbf{y}_\mathbf{v} + \mathbf{y}_{\mathbf{v}^\perp}$, where $\mathbf{x}_\mathbf{v} :=< \mathbf{x}, \mathbf{v} > \mathbf{v}$, $\mathbf{x}_{\mathbf{v}^\perp} := \mathbf{x} - \mathbf{x}_\mathbf{v}$, $\mathbf{y}_\mathbf{v} :=< \mathbf{y}, \mathbf{v} > \mathbf{v}$, and $\mathbf{y}_{\mathbf{v}^\perp} := \mathbf{y} - \mathbf{y}_\mathbf{v}$. Then,*

$$< \mathbf{x} \times \mathbf{y}, \mathbf{v} >=< \mathbf{x}_{\mathbf{v}^\perp} \times \mathbf{y}_{\mathbf{v}^\perp}, \mathbf{v} > .$$

Proof. Taking into account the proposed decomposition of vectors $\mathbf{x}$ and $\mathbf{y}$, and that $< \cdot, \cdot >$ is bilinear we have that

$$\begin{aligned} < \mathbf{x} \times \mathbf{y}, \mathbf{v} > &= < \mathbf{x_v} \times \mathbf{y_v}, \mathbf{v} > + < \mathbf{x_v} \times \mathbf{y_{v^\perp}}, \mathbf{v} > \\ &+ < \mathbf{x_{v^\perp}} \times \mathbf{y_v}, \mathbf{v} > + < \mathbf{x_{v^\perp}} \times \mathbf{y_{v^\perp}}, \mathbf{v} > . \end{aligned}$$

Therefore, since $\mathbf{x_v} \times \mathbf{y_v}$, $\mathbf{x_v} \times \mathbf{y_{v^\perp}}$ and $\mathbf{x_{v^\perp}} \times \mathbf{y_v}$ are orthogonal to $\mathbf{v}$, we have that $< \mathbf{x} \times \mathbf{y}, \mathbf{v} >=< \mathbf{x_{v^\perp}} \times \mathbf{y_{v^\perp}}, \mathbf{v} >$. □

Proposition (3.7). *Let* $\mathbf{v} \in \mathbb{R}^3$ *with* $||\mathbf{v}|| = 1$ *be an arbitrary unitary vector. Consider the signed area enclosed by the projection of a multi-loop* X *to the orthogonal plane to* $\mathbf{v}$*. Then, this area is maximized when* $\mathbf{v} = \mathbf{n}^X_{pseudo}$*. Moreover, the value of the enclosed area is* $||\mathbf{a}^X_{pseudo}||$*.*

Proof. The value of the signed area of the projection of X onto a plane orthogonal to $\mathbf{v}$ is

$$a^X_{\mathbf{v}^\perp} := \frac{1}{2} \sum_{i=1}^{m} < \mathbf{x}^i_{\mathbf{v}^\perp} \times \mathbf{x}^{i+1}_{\mathbf{v}^\perp}, \mathbf{v} > .$$

Taking into account Lemma B.1 and that $< \cdot, \cdot >$ is bilinear we have that

$$a^X_{\mathbf{v}^\perp} =< \frac{1}{2} \sum_{i=1}^{m} \mathbf{x}^i \times \mathbf{x}^{i+1}, \mathbf{v} > .$$

By means of Proposition 3.4 we have that

$$a^X_{\mathbf{v}^\perp} =< \mathbf{a}^X_{pseudo}, \mathbf{v} > .$$

Then,

$$\max_{\substack{\mathbf{v} \in \mathbb{R}^3 \\ ||\mathbf{v}||=1}} \frac{1}{2} \sum_{i=1}^{m} < \mathbf{x}^i_{\mathbf{v}^\perp} \times \mathbf{x}^{i+1}_{\mathbf{v}^\perp}, \mathbf{v} >= \max_{\substack{\mathbf{v} \in \mathbb{R}^3 \\ ||\mathbf{v}||=1}} < \mathbf{a}^X_{pseudo}, \mathbf{v} > . \tag{B.1}$$

To finalize, the maximization problem (B.1) is solved for

$$\mathbf{v} = \frac{\mathbf{a}^X_{pseudo}}{||\mathbf{a}^X_{pseudo}||} = \mathbf{n}^X_{pseudo},$$

and the maximum value is

$$< \mathbf{a}^X_{pseudo}, \mathbf{n}^X_{pseudo} >=< \mathbf{a}^X_{pseudo}, \frac{\mathbf{a}^X_{pseudo}}{||\mathbf{a}^X_{pseudo}||} >= ||\mathbf{a}^X_{pseudo}||.$$ □

B.2 Algorithm implementation

Lemma B.2. *If* $\dim \operatorname{Ker} \mathbf{A}^F = 1$, *then* $\operatorname{Ker} \mathbf{A}^F = span(\mathbf{v}_n)$.

Proof. Since $\dim \operatorname{Ker} \mathbf{A}^F = 1$ and $w_1 \geq w_2 \geq \cdots \geq w_{n-1} \geq w_n \geq 0$ we have that $w_i > 0$ for $i = 1, \ldots, n-1$ and $w_n = 0$. To finalize, by Remark 3.8 we know that $\operatorname{Ker} \mathbf{A}^F = \operatorname{span}(\mathbf{v}_n)$. $\square$

Lemma B.3. *If* $\dim \operatorname{Ker} \mathbf{A}^F = 1$, *then* $(\operatorname{Range} \mathbf{A}^F)^\perp = span(\mathbf{u}_n)$, *where* $^\perp$ *denotes the orthogonal space.*

Proof. Since $\dim \operatorname{Ker} \mathbf{A}^F = 1$ and $w_1 \geq w_2 \geq \cdots \geq w_{n-1} \geq w_n \geq 0$ we have that $w_i > 0$ for $i = 1, \ldots, n-1$, and $w_n = 0$. Taking into account Remark 3.8, we know that $\operatorname{Range} \mathbf{A}^F = \operatorname{span}(\mathbf{u}_1, \ldots, \mathbf{u}_{n-1})$. To finalize, since $\mathbf{U}$ is orthogonal we have that $< \mathbf{u}_n, \mathbf{u}_i >= 0$, for $i = 1, ..., n-1$. Therefore $(\operatorname{Range} \mathbf{A}^F)^\perp = span(\mathbf{u}_n)$. $\square$

It is important to point out that if $\dim \operatorname{Ker} \mathbf{A}^F = 1$ then the vectors $\mathbf{u}_n$ and $\mathbf{v}_n$ have a geometrical interpretation. Specifically, Lemma 3.9 states that if X is a hyperplanar set of points, then $\mathbf{v}_n$ and $\mathbf{n}^X$ generate $\operatorname{Ker} \mathbf{A}^F$ (i.e. they are two parallel vectors). Moreover, if Y is a hyperplanar set of points, then $\mathbf{u}_n$ and $\mathbf{n}^Y$ generate $(\operatorname{Range} \mathbf{A}^F)^\perp$ (i.e. they are also two parallel vectors).

Lemma B.4. *If X is a hyperplanar set of points and* $\mathbf{n}^X$ *is a unitary normal vector to X, then* $\operatorname{Ker} \overline{\mathbf{X}}^T = \operatorname{span}(\overline{\mathbf{v}}_n) = \operatorname{span}(\mathbf{n}^X)$.

Proof. Since X is hyperplanar then $\operatorname{Rank} \overline{\mathbf{X}}^T = n-1$, see Chapter 2. That is, $\dim \operatorname{Ker} \overline{\mathbf{X}}^T = 1$. Therefore, $\operatorname{Ker} \overline{\mathbf{X}}^T = \operatorname{span}(\overline{\mathbf{v}}_n)$, see Remark 3.8. Since $\mathbf{n}^X$ is a unitary normal vector we have that $\overline{\mathbf{X}}^T \mathbf{n}^X = \mathbf{0}$, see definition (2.6). Hence, $\mathbf{n}^X \in \operatorname{Ker} \overline{\mathbf{X}}^T$. Thus, $\operatorname{span}(\mathbf{n}^X) = \operatorname{Ker} \overline{\mathbf{X}}^T = \operatorname{span}(\overline{\mathbf{v}}_n)$. $\square$

Lemma (3.9)**.** *Let X be a hyperplanar set of points and* $\mathbf{A}^F$ *the optimal solution of functional F computed according to Equation* (3.2)*. If* $\mathbf{n}^X$ *is a unitary normal vector to X and* $\dim \operatorname{Ker} \mathbf{A}^F = 1$, *then*

$$\operatorname{Ker} \mathbf{A}^F = \operatorname{Ker} \overline{\mathbf{X}}^T = \operatorname{span}(\overline{\mathbf{v}}_n) = \operatorname{span}(\mathbf{v}_n) = \operatorname{span}(\mathbf{n}^X).$$

Proof. Since $\overline{\mathbf{V}}$ is an orthogonal matrix we have that $\overline{\mathbf{V}}^T \overline{\mathbf{v}}_n = (0 \cdots 0\ 1)^T$. Moreover, since X is hyperplanar and $\overline{w}_1 \geq \overline{w}_2 \geq \cdots \geq \overline{w}_{n-1} \geq \overline{w}_n \geq 0$, we have that $\overline{w}_n = 0$. Therefore, $\overline{\mathbf{W}}^+ \overline{\mathbf{V}}^T \overline{\mathbf{v}}_n = \overline{\mathbf{W}}^+ (0 \cdots 0\ 1)^T = \mathbf{0}$. Hence

$$\mathbf{A}^F \overline{\mathbf{v}}_n = \overline{\mathbf{Y}}\, \overline{\mathbf{U}}\, \overline{\mathbf{W}}^+ \overline{\mathbf{V}}^T \overline{\mathbf{v}}_n = \mathbf{0}.$$

That is, $\overline{\mathbf{v}}_n \in \operatorname{Ker}\mathbf{A}^F$. Since $\dim\operatorname{Ker}\mathbf{A}^F = 1$ we have that $\operatorname{Ker}\mathbf{A}^F = \operatorname{span}(\overline{\mathbf{v}}_n)$. To finalize, we only have to apply Lemmas B.2 and B.4. $\square$

Lemma (3.10)**.** *If Y is a hyperplanar set of points,* $\dim\operatorname{Ker}\mathbf{A}^F = 1$*, and $\mathbf{n}^Y$ is a unitary normal vector to Y, then* $(\operatorname{Rank}\mathbf{A}^F)^\perp = span(\mathbf{u}_n) = span(\mathbf{n}^Y)$*.*

Proof. First, since Y is hyperplanar then $\overline{\mathbf{Y}}^T\mathbf{n}^Y = \mathbf{0}$, or equivalently

$$(\mathbf{n}^Y)^T\overline{\mathbf{Y}} = \mathbf{0}^T. \tag{B.2}$$

Next, we will prove that $(\mathbf{n}^Y)^T\mathbf{U}\mathbf{W}\mathbf{V}^T = \mathbf{0}^T$

$$\begin{aligned}
(\mathbf{n}^Y)^T\mathbf{U}\mathbf{W}\mathbf{V}^T &= (\mathbf{n}^Y)^T\mathbf{A}^F && \text{by Equation (3.3)}\\
&= (\mathbf{n}^Y)^T\overline{\mathbf{Y}}\,\overline{\mathbf{U}}\,\overline{\mathbf{W}}^{+}\overline{\mathbf{V}}^T && \text{by Equation (3.2)}\\
&= \mathbf{0}^T && \text{by Equation (B.2)}\,.
\end{aligned}$$

Since $\mathbf{V}$ is orthogonal it is invertible. Thus

$$(\mathbf{n}^Y)^T\mathbf{U}\mathbf{W} = \mathbf{0}^T,$$

which is equivalent to the following set of conditions:

$$\begin{aligned}
(\mathbf{n}^Y)^T\mathbf{u}_1 w_1 &= 0\\
&\vdots\\
(\mathbf{n}^Y)^T\mathbf{u}_{n-1} w_{n-1} &= 0\\
(\mathbf{n}^Y)^T\mathbf{u}_n w_n &= 0.
\end{aligned}$$

Since $\dim\operatorname{Ker}\mathbf{A}^F = 1$, then $w_i > 0$, for $i = 1, \ldots, n-1$, and w_n=0. Therefore, $\mathbf{n}^Y$ is orthogonal to $\mathbf{u}_1, \ldots, \mathbf{u}_{n-1}$ (the first $n-1$ columns of matrix $\mathbf{U}$). To finalize, using Lemma B.3, we have that $\operatorname{span}(\mathbf{u}_n) = (\operatorname{Range}\mathbf{A}^F)^\perp = \operatorname{span}(\mathbf{n}^Y)$. $\square$

Appendix C

Determining polygon candidates

C.1 Intersection points for planar dual surfaces

Here we list the barycentric coordinates of the intersection points for the planar dual surfaces in a reference element. Specifically, we include the intersection points at reference edges, reference faces an inside a reference element. To this end, let us consider a fixed number μ such that $0 < \mu < \frac{1}{4}$. This number determines the position of the planar dual surfaces inside a reference element. The twelve intersection points at reference edges are

$$\begin{aligned}
p_{12} &= (\lambda_e, \mu, 0, 0), & p_{21} &= (\mu, \lambda_e, 0, 0),\\
p_{13} &= (\lambda_e, 0, \mu, 0), & p_{31} &= (\mu, 0, \lambda_e, 0),\\
p_{14} &= (\lambda_e, 0, 0, \mu), & p_{41} &= (\mu, 0, 0, \lambda_e),\\
p_{23} &= (0, \lambda_e, \mu, 0), & p_{32} &= (0, \mu, \lambda_e, 0),\\
p_{24} &= (0, \lambda_e, 0, \mu), & p_{42} &= (0, \mu, 0, \lambda_e),\\
p_{34} &= (0, 0, \lambda_e, \mu), & p_{43} &= (0, 0, \mu, \lambda_e),
\end{aligned}$$

where $\lambda_e := 1 - \mu$. Note that the first index i of an intersection point p_{ij} determines the position of λ_e in the barycentric coordinates. The twelve intersection points at reference faces are

$$\begin{aligned}
p_{234} &= (0, \lambda_f, \mu, \mu), & p_{324} &= (0, \mu, \lambda_f, \mu), & p_{423} &= (0, \mu, \mu, \lambda_f),\\
p_{413} &= (\mu, 0, \mu, \lambda_f), & p_{314} &= (\mu, 0, \lambda_f, \mu), & p_{134} &= (\lambda_f, 0, \mu, \mu),\\
p_{142} &= (\lambda_f, \mu, 0, \mu), & p_{214} &= (\mu, \lambda_f, \mu, 0), & p_{412} &= (\mu, \mu, 0, \lambda_f),\\
p_{312} &= (\mu, \mu, \lambda_f, 0), & p_{213} &= (\mu, \lambda_f, \mu, 0), & p_{123} &= (\lambda_f, \mu, \mu, 0),
\end{aligned}$$

where $\lambda_f := 1-2\mu$. Note that the first index i of an intersection point p_{ijk} determines the position λ_f in the barycentric coordinates. The four inner intersection points are

$$\begin{aligned} p_{1234} &= (\lambda_t, \mu, \mu, \mu), \quad p_{2134} = (\mu, \lambda_t, \mu, \mu), \\ p_{3124} &= (\mu, \mu, \lambda_t, \mu), \quad p_{4123} = (\mu, \mu, \mu, \lambda_t), \end{aligned}$$

where $\lambda_t := 1-3\mu$. Note that the first index i of an intersection point p_{ijkl} determines the position λ_t in he barycentric coordinates.

C.1.1 All candidate polygons

In this section we list the 28 possible candidates inside a reference element: the face candidates, edge candidates and vertex candidates. All the candidates are expressed according to the intersection points of the planar dual surfaces, see Section C.1. The four possible face candidates are expressed in terms of the inner intersection points:

$$\begin{aligned} \mathbf{c}(f_{234}; c_{1234}; t) &= [p_{2134}, p_{3124}, p_{4123}] \\ \mathbf{c}(f_{134}; c_{1234}; t) &= [p_{4123}, p_{3124}, p_{1234}] \\ \mathbf{c}(f_{124}; c_{1234}; t) &= [p_{1234}, p_{2134}, p_{4123}] \\ \mathbf{c}(f_{123}; c_{1234}; t) &= [p_{3124}, p_{2134}, p_{1234}]. \end{aligned}$$

The twelve edge candidates are expressed in terms of inner and face intersection points:

$$\begin{aligned} \mathbf{c}(e_{12}; f_{123}; t) &= [p_{213}, p_{123}, p_{1234}, p_{2134}] \\ \mathbf{c}(e_{12}; f_{124}; t) &= [p_{124}, p_{214}, p_{2134}, p_{1234}] \\ \mathbf{c}(e_{13}; f_{134}; t) &= [p_{314}, p_{134}, p_{1234}, p_{3124}] \\ \mathbf{c}(e_{13}; f_{123}; t) &= [p_{123}, p_{312}, p_{3124}, p_{1234}] \\ \mathbf{c}(e_{14}; f_{134}; t) &= [p_{134}, p_{413}, p_{4123}, p_{1234}] \\ \mathbf{c}(e_{14}; f_{124}; t) &= [p_{412}, p_{124}, p_{1234}, p_{4123}] \\ \mathbf{c}(e_{23}; f_{234}; t) &= [p_{234}, p_{324}, p_{3124}, p_{2134}] \\ \mathbf{c}(e_{23}; f_{123}; t) &= [p_{312}, p_{213}, p_{2134}, p_{3124}] \\ \mathbf{c}(e_{24}; f_{234}; t) &= [p_{423}, p_{234}, p_{2134}, p_{4123}] \\ \mathbf{c}(e_{24}; f_{124}; t) &= [p_{214}, p_{412}, p_{4123}, p_{2134}] \\ \mathbf{c}(e_{34}; f_{234}; t) &= [p_{324}, p_{423}, p_{4123}, p_{3124}] \\ \mathbf{c}(e_{34}; f_{134}; t) &= [p_{413}, p_{314}, p_{3124}, p_{4123}]. \end{aligned}$$

The twelve vertex candidates are expressed in terms of inner, face and edge intersection points:

$$
\begin{aligned}
\mathbf{c}(v_1; e_{12}; t) &= [p_{12}, p_{123}, p_{1234}, p_{124}] \\
\mathbf{c}(v_1; e_{13}; t) &= [p_{13}, p_{123}, p_{1234}, p_{134}] \\
\mathbf{c}(v_1; e_{14}; t) &= [p_{14}, p_{124}, p_{1234}, p_{134}] \\
\mathbf{c}(v_2; e_{12}; t) &= [p_{21}, p_{213}, p_{2134}, p_{214}] \\
\mathbf{c}(v_2; e_{23}; t) &= [p_{23}, p_{213}, p_{2134}, p_{234}] \\
\mathbf{c}(v_2; e_{24}; t) &= [p_{24}, p_{214}, p_{2134}, p_{234}] \\
\mathbf{c}(v_3; e_{13}; t) &= [p_{31}, p_{312}, p_{3124}, p_{314}] \\
\mathbf{c}(v_3; e_{23}; t) &= [p_{32}, p_{312}, p_{3124}, p_{324}] \\
\mathbf{c}(v_3; e_{34}; t) &= [p_{34}, p_{314}, p_{3124}, p_{324}] \\
\mathbf{c}(v_4; e_{14}; t) &= [p_{41}, p_{412}, p_{4123}, p_{413}] \\
\mathbf{c}(v_4; e_{24}; t) &= [p_{42}, p_{412}, p_{4123}, p_{423}] \\
\mathbf{c}(v_4; e_{34}; t) &= [p_{43}, p_{413}, p_{4123}, p_{423}].
\end{aligned}
$$

Appendix D

Management and development of a mesh generation environment

The management and development process of a mesh generation environment requires the use of good software engineering practices. In this section we provide useful information for managers, designers and developers of similar projects. First, we discuss programming paradigms and techniques that have been followed to develop the mesh generation environment. Second, we present an overview of the project management process. Third, we summarize the Open Source tools that we have used to develop the environment. Finally, we detail our selection of Open Source Libraries that provide high-level features for modeling and Graphical User Interface (GUI) creation.

D.1 Development paradigms and techniques

In order to deal with software development complexity we have used several modern programming paradigms and techniques. On the one hand, they improve scalability and maintainability of the software project. On the other hand they are oriented to ensure that the development process is oriented to a clear and affordable goal.

Object oriented. We adopted the *Object Oriented programming* (OOP) paradigm to promote flexibility and scalability of the program. That is, we design and implement our application by means of objects that collaborate together. Each object is capable of receiving messages, processing data and sending messages to other objects. Several OOP languages are available. However, we use C++ (Stroustrup, 2000; Koenig and Moo, 2000) because it is: a mature language, faster than other

OOP languages, and widely used in software industry. Moreover, there are a large amount of available libraries.

Agile methodology. Several agile software development methodologies have appeared during last decade. They focus on development processes that are more responsive to customer needs, “agile”, than traditional methods. In particular, *eXtreme Programming* (Beck, 2000; Newkirk and Martin, 2001) provides traditional engineering practices and takes them to ‘extreme” levels. In our project we have adopted part of the eXtreme Programming practices. In particular:

- *Pair programming.* We frequently program in couples. Two programmers developing on the same machine create better quality code. The code is reviewed at the same time it is typed. Hence, the source code knowledge is shared.

- *Small steps.* Each developer is responsible of coding a particular and small feature. Developers try to implement only the required feature and not future features. We use the tickets concept of *Trac* (Trac, 2007) for assigning small tasks to developers.

- *Unit testing.* For each new-implemented feature, a test case is added to the unit tests. Once a new feature is added all the tests in the unit are run. Unit testing ensures that after a modification is performed, the program still works and is able to run the previously developed features. We have selected *CppUnit* (CppUnit, 2007) as unit testing library.

- *Refactoring.* Developers use refactoring (Fowler, 1999) in order to simplify and make clear the source code. Code duplication is not allowed, and each time it is found developers create adequate abstractions (Gamma et al., 1995; Kerievsky, 2004). Unit testing ensures that the program has the same functionality after refactoring.

- *Sharing the source code.* All the developers share source code of a centralized repository. The developers can verify and change any part of the source code. To properly manage code sharing we use *Subversion* (Subversion, 2007) as a version control system.

- *Continuous integration.* After a new feature is correctly implemented, it is committed to the source code repository. That is, new features are continuously integrated in the shared source code. Subversion and distributed compilation with *distcc* (distcc, 2007) facilitates new features integration.

Design patterns. The development of a mesh generation environment requires the design of a complex system, with a high number of classes collaborating together. These classes collaborate through inheritance and aggregation in order to represent complex data structures such as meshes and solid models. Several issues that appear during the design process of the environment are software engineering common problems. For this reason when a design problem is identified the adequate pattern (Gamma et al., 1995; Freeman et al., 2004; Shalloway and Trott, 2004) is used to solve the problem. In addition, we take into account design patterns during code refactoring (Kerievsky, 2004).

Source code guidelines and checking. Code guidelines (Sutter and Alexandrescu, 2004; KDE, 2007b) help developers to write cleaner code, simplify maintenance, improve code communication, reduce coding time, and improve quality. Moreover, code guidelines are a good resource for design and programming tips (Sutter and Alexandrescu, 2004; Meyers, 1997, 1995; Sutter, 2004, 2002). Code guidelines provide to developers rules on:

- *Organizational and policy issues.* Tools and techniques for writing solid code, such as: version control systems, compiler flags, code reviewing, and automatic building tools.

- *Design style.* Software engineering good design practices, such as: write simple classes, avoid premature optimization, reduce class dependencies and encapsulate data.

- *Coding style.* Coding issues, such as: how to use `#include` guards, avoid cyclic dependencies, always initialize variables, and prefer compiler and linker errors to run-time errors.

- *Implementation.* Tips and rules related to the implementation of the design concepts, such as: prefer composition to inheritance, when we have to use inheritance or templates, object construction and destruction, automatic memory management or error handling.

To verify that code guidelines are followed we use pair programming and code reviews of the new code. In addition, we automatically check part of the rules with the *Krazy* (Krazy, 2007) tool, which is part of *English Breakfast Network* (EBN, 2007). Krazy looks for some issues that should be fixed for reasons of policy, design, coding or implementation.

D.2 Project management

We adopt part of eXtreme Programming agile methodology using *Trac* (Trac, 2007). Trac is an Open Source tool for web-based software project management. Moreover, it provides wiki implementation, issue tracking system and an interface to Subversion.

The flow of the development can be driven with Trac using the *Milestone* concept. Each Milestone represents a group of common *Tickets* (enhancements, tasks and defects) that define a new version (or an iteration in the eXtreme Programming context) of the program. Users of the environment can report bugs and wish lists creating new tickets. In this sense, tickets provide issue-tracking functionality to Trac. The *Roadmap* shows a view of the current state of the project by means of the number of open and closed tickets per milestone. Trac allows to link tickets from: the wiki, other tickets, and commit messages of Subversion. Developers can access to the Subversion repository from a web interface. Finally, the wiki facilitates communication between developers. We use the wiki to dynamically add or change: guidelines, programming tips, comments on common programming errors, documentation and useful information for developers.

D.3 Tools

Several Open Source tools are used in order to: analyze, design, implement, manage, compile and document the source code of the environment. Below we provide our particular selection of Open Source tools.

- *Kubuntu* (Kubuntu, 2007) is a GNU/Linux distribution based on textitKDE, the K Desktop Environment (KDE, 2007a). KDE is built on the *Qt* library (Trolltech, 2007). Therefore, development with Qt library is natural in this platform.

- *KDevelop* (KDevelop, 2007) is an easy to use Integrated Development Environment (IDE) for developing KDE applications. Since KDE is based on Qt, KDevelop it is also a suitable IDE to develop with Qt. Among its features we highlight: C++ and Qt project management, debugger, profiler, and visual GUI designer.

- *GCC* (gnu, 2007a) is the GNU Compiler Collection. It provides compilers for several languages, in particular for C++. It is the standard compiler for development under GNU/Linux. In addition, it provides a good C++ standard implementation.

- *distcc* (distcc, 2007) accelerates source code builds by means of distributed compilation on several machines. Thus, it improves productivity since code, compile, test, and debug cycle time is reduced.

- *Subversion* (Subversion, 2007) is a version control system with similar interface and features to *CVS* (CVS, 2007). However, it also provides:

  - version control of directories, copies, and renames.
  - revision numbers are assigned in terms of per-commit and not in per-file terms.
  - efficient operations in terms of memory and CPU time.

  Moreover, several mature GUI front-ends are available for Subversion. These front-ends facilitate usual operations such as: commit, merge, blame, diff or patch. In particular, we have selected *TortoiseSVN* (TortoiseSVN, 2007) for Windows, and *KDESvn* (KDESvn, 2007) for Kubuntu.

- *CMake* (Kitware, 2007), the cross-platform make, is used to automate the build process. It creates the makefiles and workspaces for a particular platform and compiler.

- *GDB* (gnu, 2007b), the GNU project debugger, provides standard debugging features and can be called from KDevelop.

- *Valgrind* (Valgrind, 2007) is a complete tool for debugging and profiling Linux programs. KDevelop integrates Valgrind in the IDE. Current version provides four tools:

  - *Memory error detector.* This tool allows to automatically checking for memory errors such as: uninitialized memory, bad memory access, read and write out of limits of allocated memory, and memory leaks.
  - *cache (time) profiler.* It provides time cost analysis in order to improve the computational efficiency of the program.
  - *Call-graph profiler.* It analyzes the call relationships between functions of an application.
  - *Heap profiler.* It analyzes where, when and how much memory is allocated during the program execution.

- *Doxygen* (van Heesch, 2007) generates, from the source code, documentation for several languages, in particular for C++. The user can configure the output and obtain documentation in: HTML, LaTeX, RTF, MS-Word, PostScript, and Unix man pages. Moreover, it can extract the source code structure and create UML inheritance and collaboration class diagrams.

D.4 Libraries

In the development process we have used the following three open source libraries:

- *Open CASCADE* (Open CASCADE, 2007) is a powerful Open Source geometry and topology kernel that provides essential features for solid modeling, CAD data exchange, and rapid application development. It is used by several Open Source mesh generation environments such as: *GMSH* (Geuzaine and Remacle, 2009), *SALOME* (Salome, 2009), and *NetGen* (Schorbel, 2009).

- *Qt* (Trolltech, 2007) is a standard cross-platform library for rapid GUI development with C++. The GUI of our environment is fully implemented with Qt. Several tool bars, dock windows and a central widget with the current 3D view compose it.

- *GLPK* (gnu, 2007c), the GNU Linear Programming Kit, is a C library for the resolution of large scale Linear Programming (LP) and Mixed Integer Programming (MIP) problems. This library is used to ensure edge division compatibility between adjacent faces in unstructured quadrilateral and submapping mesh generation algorithms.

Bibliography

ANSYS (2009a). ANSYS ICEM CFD. `http://www.ansys.com/products/icemcfd.asp`.

ANSYS (2009b). Gambit. `http://www.ansys.com/products/workbench/meshing/`.

Athanasiadis, A. and H. Deconinck (2003). Object-oriented three-dimensional hybrid grid generation. *International Journal for Numerical Methods in Engineering 58*, 301–318.

Baker, T. J. (1987). Three dimensional mesh generation by triangulation of arbitrary point sets. In *8th Computational Fluid Dynamics Conference*, pp. 255–271.

Baker, T. J. (2005). Mesh generation: Art or science? *Progress in Aerospace Sciences 41*(1), 29–63.

Beck, K. (2000). *Extreme programming explained: embrace change.* Addison-Wesley Longman Publishing Co., Inc.

Benzley, S., E. Perry, K. Merkley, B. Clark, and G. Sjaardema (1995). A comparison of all-hexahedral and all-tetrahedral finite element meshes for elastic and elasto-plastic analysis. In *4th International Meshing Roundtable*, pp. 179–191. Sandia National Laboratories.

Blacker, T. D. (1996). The Cooper tool. In *5th International Meshing Roundtable.*

Blacker, T. D. (2001). Automated conformal hexahedral meshing constraints, challenges and opportunities. *Engineering with Computers 17*(3), 201–210.

Blacker, T. D. and R. J. Meyers (1993). Seams and wedges in Plastering: a 3-D hexahedral mesh generation algorithm. *Engineering with computers 9*(2), 83–93.

Blacker, T. D. and M. B. Stephenson (1991). Paving: A new approach to automated quadrilateral mesh generation. *International Journal for Numerical Methods in Engineering 32*(4), 811–847.

Calvo, N. A. (2005). *Generación de mallas tridimensionales por métodos duales.* Ph. D. thesis, Universidad Nacional del Litoral.

Calvo, N. A. and S. R. Idelsohn (2000). All-hexahedral element meshing: Generation of the dual mesh by recurrent subdivision. *Computer Methods in Applied Mechanics and Engineering 182*(3-4), 371–378.

Carey, G. F. (2002). Hexing the tet. *Communications in Numerical Methods in Engineering 18*(3), 223–227.

Cass, R. J., S. Benzley, R. J. Meyers, and T. D. Blacker (1996). Generalized 3-D paving: An automated quadrilateral surface mesh generation algorithm. *International Journal for Numerical Methods in Engineering 39*(9), 1475–1490.

Cifuentes, A. O. and A. Kalbag (1992). A performance study of tetrahedral and hexahedral elements in 3-D finite element structural analysis. *Finite Elements in Analysis and Design 12*(3-4), 313 – 318.

Cook, W. A. and W. R. Oakes (1982). Mapping methods for generating three-dimensional meshes. *Computers in mechanical engineering 8*, 67–72.

CppUnit (2007). CppUnit - C++ port of JUnit. `http://sourceforge.net/projects/cppunit`.

Cuthill, E. and J. McKee (1969). Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th national conference*, pp. 157–172. ACM Press New York, NY, USA.

CVS (2007). CVS - Concurrent Versions System. `http://www.nongnu.org/cvs`.

distcc (2007). distcc: a fast, free distributed c/c++ compiler. `http://distcc.samba.org`.

EBN (2007). English breakfast network. `http://www.englishbreakfastnetwork.org`.

Folwell, N. T. and S. A. Mitchell (1999). Reliable whisker weaving via curve contraction. *Engineering with Computers 15*(3), 292–302.

Fowler, M. (1999). *Refactoring: improving the design of existing code.* Addison-Wesley Longman Publishing Co., Inc.

Freeman, E., B. Bates, and K. Sierra (2004). *Head First Design Patterns.* O' Reilly & Associates, Inc.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides (1995). *Design patterns: elements of reusable object-oriented software.* Addison-Wesley Longman Publishing Co., Inc.

George, P. L., F. Hecht, and E. Saltel (1988). Constraint of the boundary and automatic mesh generation. In *Numerical grid generation in computational fluid mechanics*, pp. 589–597.

Geuzaine, C. and J. F. Remacle (2009). Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. `http://www.geuz.org/gmsh/`.

Gill, P. E., W. Murray, and M. H. Wright (1991). *Numerical Linear Algebra and Optimization.* Addison-Wesley.

gnu (2007a). Gcc, the gnu compiler collection. `http://gcc.gnu.org`.

gnu (2007b). Gdb - the gnu project debugger. `http://sourceware.org/gdb/`.

gnu (2007c). Glpk (gnu linear programming kit). `http://www.gnu.org/software/glpk/`.

Goodrich, D. (1997). Generation of all-quadrilateral surface meshes by mesh morphing. Master's thesis, Brigham Young University.

Haber, R., M. S. Shephard, J. F. Abel, R. H. Gallagher, and D. P. Greenberg (1981). A general two-dimensional, graphical finite element preprocessor utilizing discrete transfinite mappings. *International Journal for Numerical Methods in Engineering 17*(7), 1015–1044.

Halpern, M. (1997). Industrial requirements and practices in finite element meshing: a survey of trends. In *6th International Meshing Roundtable*, pp. 399–411.

Hatcher, A. (2002). *Algebraic Topology.* Cambridge University Press.

Hesthaven, J. S. and T. Warburton (2008). *Nodal Discontinuous Galerkin Methods.* Springer.

KDE (2007a). The k desktop environment. `http://www.kde.org`.

KDE (2007b). Kde developer's corner - howtos and faqs. `http://developer.kde.org/documentation/other/index.html`.

KDESvn (2007). Kdesvn. `http://www.alwins-world.de/wiki/programs/kdesvn/`.

KDevelop (2007). Kdevelop - an integrated development environment. `http://www.kdevelop.org`.

Kerievsky, J. (2004). *Refactoring to Patterns.* Pearson Higher Education.

Kitware (2007). Cmake - cross platform make. `http://www.cmake.org`.

Knupp, P. M. (1998). Next-generation sweep tool: a method for generating all-hex meshes on two-and-one-half dimensional geometries. In *7th International Meshing Roundtable*, pp. 505–513.

Knupp, P. M. (1999). Applications of mesh smoothing: copy, morph, and sweep on unstructured quadrilateral meshes. *International Journal for Numerical Methods in Engineering 45*(1), 37–46.

Knupp, P. M. (2001). Algebraic mesh quality metrics for unstructured initial meshes. *Finite Elements in Analysis and Design 39*(3), 217–241.

Koenig, A. and B. Moo (2000). *Accelerated C++: practical programming by example.* Addison-Wesley Longman Publishing Co., Inc.

Krazy (2007). Krazy. `http://techbase.kde.org/Development/Tutorials/Code_Checking`.

Kubuntu (2007). Kubuntu - the kde desktop. `http://www.kubuntu.org`.

Lawson, C. and R. Hanson (1974). *Solving Least Squares Problems.* Prentice-Hall.

Ledoux, F. and J. C. Weill (2007). An Extension of the Reliable Whisker Weaving Algorithm. In *16th International Meshing Roundtable*, pp. 215–232. Springer.

Lohner, R., K. Morgan, J. Peraire, and O. C. Zienkiewicz (1985). Finite element methods for high speed flows. In *7th Computational Fluid Dynamics Conference*, pp. 403–410. AIAA.

Lohner, R. and P. Parikh (1988). Generation of three-dimensional unstructured grids by the advancing-front method. *International Journal for Numerical Methods in Fluids 8*(10), 1135–1149.

Meyers, S. (1995). *More Effective C++: 35 New Ways to Improve Your Programs and Designs*. Addison-Wesley Longman Publishing Co., Inc.

Meyers, S. (1997). *Effective C++: 50 specific ways to improve your programs and designs*. Addison-Wesley Longman Publishing Co., Inc.

Mingwu, L. and S. E. Benzley (1996). A multiple source and target sweeping method for generating all-hexahedral finite element meshes. In *5th International Meshing Roundtable*, pp. 217–225.

Mitchell, S. A. (1996). A characterization of the quadrilateral meshes of a surface which admit a compatible hexahedral mesh of enclosed volume. *Lecture Notes in Computer Science 1046*, 465–478.

Miyoshi, K. and T. D. Blacker (2000). Hexahedral mesh generation using multi-axis cooper algorithm. In *9th International Meshing Roundtable*, pp. 89–97.

Mueller-Hannemann, M. (1999). Hexahedral mesh generation by successive dual cycle elimination. *Engineering with Computers 15*(3), 269–279.

Murdoch, P., S. Benzley, T. D. Blacker, and S. A. Mitchell (1997). The spatial twist continuum: A connectivity based method for representing all-hexahedral finite element meshes. *Finite Elements in Analysis and Design 28*(2), 137–149.

Newkirk, J. and R. Martin (2001). *Extreme Programming in Practice*. Addison-Wesley Longman Publishing Co., Inc.

Open CASCADE (2007). Open CASCADE Technology, 3D modeling & numerical simulation. `http://www.opencascade.org`.

Owen, S. J. (1998). A survey fo unstructured mesh generation technology. In *7th International Meshing Roundtable*, pp. 239–267.

Patera, A. T. (1984). A spectral element method for fluid dynamics-laminar flow in a channel expansion. *Journal of Computational Physics 54*(3), 468–488.

Peraire, J., J. Peiro, L. Formaggia, K. Morgan, and O. C. Zienkiewicz (1988). Finite element Euler computations in three dimensions. *International Journal for Numerical Methods in Engineering 26*(10), 2135–2159.

Peraire, J., M. Vahdati, K. Morgan, and O. C. Zienkiewicz (1987). Adaptive remeshing for compressible flow computations. *Journal of computational physics 72*(2), 449–466.

Piegl, L. A. and W. Tiller (1997). *The NURBS book*. Springer.

Pointwise (2009). Gridgen - Reliable CFD Meshing. `http://www.pointwise.com/gridgen/`.

Price, M., C. Armstrong, and M. Sabin (1995). Hexahedral mesh generation by medial surface subdivision: Part I. Solids with convex edges. *International Journal for Numerical Methods in Engineering 38*(19), 3335–3359.

Price, M. A. and C. G. Armstrong (1997). Hexahedral mesh generation by medial surface subdivision: Part II. Solids with flat and concave edges. *International Journal for Numerical Methods in Engineering 40*(1), 111–136.

Program Development Company (2009). GridPro - High quality grid generation. `http://www.gridpro.com/`.

Roca, X. and J. Sarrate (2006). An automatic and general least-squares projection procedure for sweep meshing. In *15th International Meshing Roundtable*, pp. 437–506.

Roca, X. and J. Sarrate (2008). Local dual contributions on simplices: A tool for block meshing. In *17th International Meshing Roundtable*, pp. 513–531.

Roca, X., J. Sarrate, and A. Huerta (2005a). Generación de mallas de cuadriláteros sobre superficies paramétricas. In *Congreso de Métodos Numéricos en Ingeniería*, Granada, Spain.

Roca, X., J. Sarrate, and A. Huerta (2005b). A new least–squares approximation of affine mappings for sweep algorithms. To appear in *Engineering with Computers.*

Roca, X., J. Sarrate, and A. Huerta (2006). Mesh projection between parametric surfaces. *Communications in Numerical Methods in Engineering 22*, 591–603.

Ruiz-Gironès, E., X. Roca, and J. Sarrate (2009). Automatic generation of hexahedral meshes using the multi-sweeping method. In *Congreso de Métodos Numéricos en Ingeniería*, Barcelona, Spain.

Ruiz-Gironès, E. and J. Sarrate (2008). Automatic generation of structured hexahedral meshes for non-simply connected geometries using submapping. In *The Sixth International Conference on Engineering Computational Technology.*

Ruiz-Gironès, E. and J. Sarrate (2009). Generation of structured meshes in multiply connected surfaces using submapping. To appear in *Advances in Engineering Software.*

Salome (2009). Salome: The open source integration platform for numerical simulation. `http://www.salome-platform.org`.

Sandia National Labs (2009). Cubit - geometry and mesh generation toolkit. `http://cubit.sandia.gov`.

Sarrate, J. and A. Huerta (2000a). Automatic mesh generation of nonstructured quadrilateral meshes over curved surfaces in $\mathbb{R}^3$. In *3th ECCOMAS*, Barcelona, Spain.

Sarrate, J. and A. Huerta (2000b). Efficient unstructured quadrilateral mesh generation. *International Journal for Numerical Methods in Engineering 49*(10), 1327–1350.

Sarrate, J. and A. Huerta (2004). A new approach to minimize the distortion of quadrilateral and hexahedral meshes. In *4th ECCOMAS*, Jyväskylä, Finland.

Schneiders, R. (1996). A grid-based algorithm for the generation of hexahedral element meshes. *Engineering with Computers 12*(3), 168–177.

Schneiders, R. and R. Bünten (1995). Automatic generation of hexahedral finite element meshes. *Computer Aided Geometric Design 12*(7), 693–707.

Schorbel, J. (2009). Netgen - automatic mesh generator. `http://www.hpfem.jku.at/netgen/`.

Scott, M. A., M. N. Earp, S. E. Benzley, and M. B. Stephenson (2005). Adaptive sweeping techniques. In *14th International Meshing Roundtable*, pp. 417–432.

Sevilla, R. (2009). *The NURBS-Enhanced Finite Element Method (NEFEM).* Ph. D. thesis, Universitat Politènica de Catalunya.

Sevilla, R., S. Fernández-Méndez, and A. Huerta (2008). NURBS-enhanced finite element method (NEFEM). *International Journal for Numerical Methods in Engineering 76*(1), 56–83.

Shalloway, A. and J. Trott (2004). *Design Patterns Explained: A New Perspective on Object-Oriented Design.* Addison-Wesley Professional.

Sheffer, A. and M. Bercovier (2000). Hexahedral meshing of non-linear volumes using Voronoi faces and edges. *International Journal for Numerical Methods in Engineering 49*, 329–351.

Sheffer, A., M. Etzion, A. Rappoport, and M. Bercovier (1999). Hexahedral mesh generation using the embedded Voronoi graph. *Engineering with Computers 15*(3), 248–262.

Shephard, M. S. and M. K. Georges (1991). Automatic three-dimensional mesh generation by the finite octree technique. *International Journal for Numerical Methods in Engineering 32*(4), 709–749.

Shepherd, J. F. (2007). *Topologic and geometric constraint-based hexahedral mesh generation.* Ph. D. thesis, The University of Utah.

Shepherd, J. F. and C. R. Johnson (2008). Hexahedral mesh generation constraints. *Engineering with Computers 24*(3), 195–213.

Shewchuk, J. R. (1996). Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. *Lecture Notes in Computer Science 1148*, 203–222.

Si, H. (2007). A quality tetrahedral mesh generator. `http://tetgen.berlios.de/`.

Simulia (2009). Abaqus/CAE. `http://www.simulia.com/products/abaqus_cae.html`.

Staten, M. L., S. A. Canann, and S. J. Owen (1999). BMSweep: locating interior nodes during sweeping. In *Engineering with computers*, pp. 212–218.

Staten, M. L., R. A. Kerr, S. J. Owen, and T. D. Blacker (2006). Unconstrained paving and plastering: Progress update. In *Proceedings, 15th International Meshing Roundtable*, pp. 469–486. Springer.

Staten, M. L., S. J. Owen, and T. D. Blacker (2005). Unconstrained paving and plastering: A new idea for all hexahedral mesh generation. In *14th International Meshing Roundtable*.

Stroustrup, B. (2000). *The C++ Programming Language*. Addison-Wesley Longman Publishing Co., Inc.

Subversion (2007). Subversion. `http://subversion.tigris.org`.

Sutter, H. (2002). *More exceptional C++: 40 new engineering puzzles, programming problems, and solutions*. Addison-Wesley Longman Publishing Co., Inc.

Sutter, H. (2004). *Exceptional C++ Style: 40 New Engineering Puzzles, Programming Problems, and Solutions*. Pearson Higher Education.

Sutter, H. and A. Alexandrescu (2004). *C++ Coding Standards: 101 Rules, Guidelines, and Best Practices*. Addison-Wesley Professional.

Tam, T. K. H. and C. G. Armstrong (1991). 2D finite element mesh generation by medical axis subdivision. *Advances in engineering software and workstations 13*(5-6), 313–324.

Tautges, T. J. (2001). The generation of hexahedral meshes for assembly geometry: survey and progress. *International Journal for Numerical Methods in Engineering 50*(12), 2617–2642.

Tautges, T. J., T. D. Blacker, and S. A. Mitchell (1996). The whisker weaving algorithm: A connectivity-based method for constructing all-hexahedral finite element meshes. *International Journal for Numerical Methods in Engineering 39*(19), 3327–3350.

Tautges, T. J., D. R. White, and R. W. Leland (2004). Twelve ways to fool the masses when describing mesh generation performance. In *13th International Meshing Roundtable*, pp. 181–190.

TC184, I. and I. SC (2009). 10303-42-part 42: Industrial automation systems and integration – product data representation and exchange – integrated generic resources: Geometric and topological representation. `http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=37846`.

Thompson, J. F., B. Soni, and N. Weatherill (1999). *Handbook of Grid Generation.* CRC Press.

TortoiseSVN (2007). Tortoisesvn. `http://tortoisesvn.tigris.org`.

Trac (2007). The trac project - trac. `http://trac.edgewall.org`.

Trolltech (2007). Qt - code less. create more. `http://trolltech.com/products/qt`.

Valgrind (2007). Valgrind. `http://valgrind.org`.

van Heesch, D. (2007). Doxygen. `http://www.stack.nl/~dimitri/doxygen/`.

Weingarten, V. (1994). The controversy over hex or tet meshing. *Machine design 66*(8), 74–76.

White, D. (1996). Automatic, quadrilateral and hexahedral meshing of pseudo-cartesian geometries using virtual subdivision. Master's thesis, Brigham Young University.

White, D. R., S. Saigal, and S. J. Owen (2004). CCSweep: automatic decomposition of multi-sweep volumes. *Engineering with Computers 20*(3), 222–236.

White, D. R. and T. J. Tautges (2000). Automatic scheme selection for toolkit hex meshing. *International Journal for Numerical Methods in Engineering 49*(1-2), 127–144.

Whiteley, M., D. R. White, S. Benzley, and T. D. Blacker (1996). Two and three-quarter dimensional meshing facilitators. *Engineering with Computers 12*(3-4), 144–154.

Yerry, M. A. and M. S. Shephard (1984). Automatic three-dimensional mesh generation by the modified-octree technique. *International Journal for Numerical Methods in Engineering 20*(11), 1965–1990.

Zhang, Y. and C. Bajaj (2006). Adaptive and quality quadrilateral/hexahedral meshing from volumetric data. *Computer Methods in Applied Mechanics and Engineering 195*(9-12), 942–960.

Zhang, Y., C. Bajaj, and B. S. Sohn (2005). 3D finite element meshing from imaging data. *Computer Methods in Applied Mechanics and Engineering 194*(48-49), 5083–5106.