An introduction to geometrical modelling and mesh generation with Gmsh

Christophe Geuzaine Jean-François Remacle

November 4, 2008

Contents

Chapter 1

Introduction

When we started the Gmsh project in the summer of 1996, our goal was to develop a *fast*, *light* and *user-friendly* software to easily create geometries and meshes that could be used in our three-dimensional finite element solvers [**?**], and then visualize and export the computational results with maximum flexibility. At the time, no open-source software combining a CAD engine, a mesh generator and a post-processor was available: the existing integrated tools were expensive commercial packages [**?**], and the freeware or shareware tools were limited to either CAD [**?**], two-dimensional mesh generation [**?**], three-dimensional mesh generation [**?**, **?**, **?**], or post-processing [**?**]. The need for a free integrated solution was conspicuous, and several projects similar in spirit to Gmsh were also born around the same time—some of them still actively developed today [**?**, **?**]. Gmsh however was unique in its design: it consisted of a very small kernel with four modules (geometry, mesh, solver and post-processing), not tied to any particular computational solver, and designed from the start to be driven both using a user-friendly graphical interface (GUI) and its own scripting language.

The first public release of Gmsh occurred in 1998. This version was Unix-only, distributed over the internet in binary form, with the graphics layer based on OpenGL [**?**] and the user interface written in Motif [**?**]. After several updates and a short-lived Windows-only fork in 2000, the whole user interface was rewritten using FLTK [**?**] in early 2001, and the code, still in binary-only form, was released for Windows and a variety of Unix operating systems. In 2003 the full source code was released under the GNU General Public License [**?**], and it was modified to provide native support for all major operating systems: Windows, MacOS and all Unix/X11 variants. In the

summer of 2006 Gmsh underwent a major rewrite, which led to the release of version 2 of the software in February 2007. About 50% of the code in version 2 is new: an abstract geometrical and post-processing layer has been introduced, the mesh data structures and algorithms have been rewritten from scratch, and the graphics layer has also been completely overhauled.

Today Gmsh enjoys a thriving community of several hundred users and developers worldwide. It is driven by the need of researchers and engineers in academia and industry alike for a small, open-source pre- and post-processing solution for grid-based numerical methods. The aim of this paper is not to be a user's guide or a reference manual—see [**?**] instead. Rather, it is to present the philosophy and the original features of Gmsh which make it stand out from its free and commercial alternatives.

The paper is structured as follows. In Section 1.1 we outline the overall philosophy and design goals of Gmsh, as well as the main technical choices that were made in order to achieve these goals. Sections **??**, **??**, **??** and **??** then respectively describe the geometry, mesh, solver and post-processing modules. The paper is concluded in Section **??** with perspectives for future developments.

1.1 The Design of Gmsh

Gmsh is built around four modules: geometry, mesh, solver and post-processing. Each module can be controlled either interactively using the GUI or using the scripting language.

The design of all four modules relies on a simple philosophy—be *fast*, *light* and *user-friendly*.

Fast: on a standard personal computer at any given point in time Gmsh should launch instantaneously, be able to generate a "larger than average" mesh (compared to the standards of the finite element community; say, one million tetrahedra in 2008) in less than a minute, and be able to visualize such a mesh together with associated post-processing datasets at interactive speeds.

Light: the memory footprint of the application should be minimal and the source code should be small enough so that a single developer can understand it. Installing or running the software should not depend on any non-widely available third-party software package.

User-friendly: the graphical user interface should be designed in such a way that a new user can create simple meshes in a matter of minutes. In addition, the code should be robust, portable, scriptable, extensible and thoroughly documented—all features contributing to a user-friendly experience.

In the following sections we describe the technical choices that we made to achieve these sometimes conflicting design objectives. Although major parts of the code have been rewritten over the years, the overall initial architecture and design from 1996 have always stayed the same.

1.1.1 Fast and Light

In order to be fast and light in the sense just described above, Gmsh is entirely written in standard C++ [**?**]—both the kernel and the user interface.

The kernel uses BLAS [**?**] (through the GSL, the GNU Scientific Library [**?**]) for most of the basic linear algebra. To keep them easy to understand the algorithms have not been overly optimized for speed or memory usage, yet Gmsh currently generates about a million tetrahedra per minute and per 150 Mb of RAM on a standard personal computer, which makes it powerful enough for many academic and engineering applications.

The graphical interface is built using FLTK [**?**] and OpenGL [**?**]. Using FLTK instead of a larger or more complex widget toolkit, like for example Java, TCL/TK, GTK or QT, allows to link Gmsh *statically* with the toolkit. This tremendously reduces the launch time, memory footprint and installation complexity (installing Gmsh requires copying a *single* executable file), as well as the build time—a statically linked, ready to use executable is produced in a few minutes on a standard personal computer. Analogously, directly using OpenGL instead of a more complex graphics library like Inventor [**?**] or VTK [**?**] makes Gmsh lightweight, without sacrificing rendering performance (Gmsh makes extensive use of OpenGL vertex arrays).

The design of the solver and scripting interfaces follows the same strategy: the solver interface is written using standard Unix and TCP/IP sockets instead of, e.g., Corba [**?**], and the scripting interface is built using Lex/Flex and Yacc/Bison [**?**] instead of using an external scripting language like, e.g., Python.

1.1.2 User-friendly

Although Gmsh can be built as a library (which can then be linked with other software tools), it is usually distributed as a stand-alone software, ready to be used by end users. This stand-alone version can be run either interactively using the graphical user interface or in a non-interactive mode—either from the command line or via the scripting language.

Achieving "user-friendliness" has been an important driving factor behind key technical choices. We detail some of these choices hereafter, focusing on the list of desirable features given at the beginning of the section.

Robustness and Portability

To achieve robustness, i.e., working for the largest possible range of input data and being as tolerant as possible to erroneous user input, we use robust geometrical predicates [**?**] in critical portions of the algorithms, and strive to provide useful error messages when an unmanageable exception is triggered.

In order to easily produce a native version of the code on all major operating systems, Gmsh is written entirely in standard C++, and uses portable toolkits for its GUI (FLTK) and graphics rendering (OpenGL). Open-sourcing Gmsh under the GNU General Public License [**?**] also helped portability, as the code was made part of several official Linux distributions (most notably Debian [**?**]), and thus benefited from their extensive automated testing infrastructure. Either with the graphical user interface or in batch mode, the same version of Gmsh now runs on most computers, from laptops to workstations and large HPC clusters.

Scriptability

Gmsh is scriptable so that all input data can be parametrized, and so that Gmsh can be easily inserted as a component inside a larger computational chain. As mentioned above, scripting is implemented in Gmsh using Lex and Yacc. The tight integration with the resulting language means that full access to internal capabilities is provided, including bidirectional access to more than 500 internal options fine-tuning the behaviour of the four modules. The scripting language allows for example to fully parametrize all geometrical entities, to interface external solvers without modifying the source code, or to automate all post-processing operations, e.g., to create complex animations or perform off-screen rendering [**?**].

Extensibility

We tried to ease the modification and addition of features by users and developers. Such extensibility takes different forms for each of the four modules:

Geometry: the abstract, object-oriented geometry layer permits to write all the algorithms independently of the underlying CAD representation. At the source code level Gmsh is thus easily extensible by adding support for additional CAD engines. Currently two engines are interfaced: the native Gmsh CAD engine and OpenCascade [**?**]. Adding support for other engines like, e.g., Parasolid [**?**], can be done simply by deriving four abstract classes—see Section **??**. At the scripting level users can then transparently mix and match geometrical parts represented internally by different CAD engines. For example, it is possible to extend an OpenCascade model of an airplane with a terrain model defined in the scripting language.

Mesh: using the abstract geometrical interface it is also possible to interface additional meshing kernels. Currently, in addition to its own meshing algorithms (see Section **??**), Gmsh is interfaced with Netgen [**?**] and Tetgen [**?**].

Solver: a socket-based communication interface allows to interface Gmsh with various solvers without changing the source code; tailored graphical user interfaces can also easily be added when more fine-grained interactions are needed.

Post-processing: the post-processor can be extended with user-defined operations through dynamically loadable plug-ins. These plug-ins act on post-processing datasets (called *views*) in one of two ways: either destructively changing the contents of a view, or creating one or more views based on the current view.

All source code-level extensions can be enabled or disabled at compile time thanks to an autoconf-based build mechanism [**?**], which selects which parts of the code to include/exclude.

Documentation and Open File Formats

Documentation is provided both in the source code and in the form of a reference manual, including several hands-on tutorial examples and a com-

prehensive web site with several mailing lists and a wiki.

Another important feature contributing to user-friendliness is the availability of standard input and output file formats. Gmsh uses open or *de facto* standard formats whenever possible, from standard bitmap graphics formats (JPEG, GIF, PNG) and vector formats [**?**] (SVG, PostScript, PDF) to mesh formats (Ideas UNV, Nastran BDF). Combined with the open-source release of the code this greatly facilitates the integration of Gmsh with other computational tools.

Chapter 2

Computational Geometry Toolbox

2.1 Delaunay and Voronoï

2.1.1 The Voronoï diagram

Let $\mathbf{p}_1$ and $\mathbf{p}_2$ be two points of R^2. The mediator $\mathcal{M}(\mathbf{p}_1, \mathbf{p}_2)$ is the locus of all the points which are equidistant to $\mathbf{p}_1$ and $\mathbf{p}_2$:

$$\mathcal{M}(\mathbf{p}_1, \mathbf{p}_2) = \{\mathbf{p} \in R^2, d(\mathbf{p}, \mathbf{p}_1) = d(\mathbf{p}, \mathbf{p}_2)\}$$

where $d(.,.)$ is the euclidian distance between two points of R^2, i.e.

$$d^2(\mathbf{p}_1, \mathbf{p}_2) = (\mathbf{p}_2 - \mathbf{p}_1) \cdot (\mathbf{p}_2 - \mathbf{p}_1). \tag{2.1}$$

The equation of the mediator can be found out using this definition

$$(\mathbf{p}_1 - \mathbf{p}) \cdot (\mathbf{p}_1 - \mathbf{p}) = (\mathbf{p}_2 - \mathbf{p}) \cdot (\mathbf{p}_2 - \mathbf{p})$$

or

$$(\mathbf{p}_1 - \mathbf{p}_2) \cdot \left[\mathbf{p} - \underbrace{\frac{1}{2}(\mathbf{p}_1 + \mathbf{p}_2)}_{\mathbf{p}_m} \right] = 0$$

This shows that the mediator is the orthogonal bissector of the straight edge linking the two points. The mediator separates the plane into two regions.

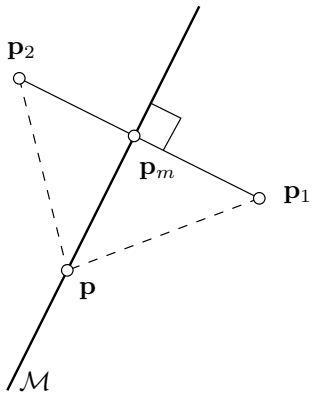


Figure 2.1: The mediator.

The first region contains all the points that are closer to $\mathbf{p}_1$, the second one contains the ones that are closer to $\mathbf{p}_2$. Any point of $R^2/\mathcal{M}$ can be associated to one of those two points.

Let us consider a set of N points $\mathcal{S} = \{\mathbf{p}_1, \ldots, \mathbf{p}_N\}$. More specifically, we assume that the points are in general position, by which we mean no four points are cocircular. The Voronoï cell $\mathcal{C}(\mathbf{p}_i)$ associated to point $\mathbf{p}_i$ is the locus of points of R^2 that are closer to $\mathbf{p}_i$ than any other point $\mathbf{p}_j$, $j = 1, \ldots, N$, $i \neq j$.

The Voronoï cell is constructed as follow. Consider all mediators $\mathcal{M}(\mathbf{p}_i, \mathbf{p}_j)$. Each of those mediators define a half plane. The Voronoï diagram is the part of the space that is always closer to $\mathbf{p}_i$, i.e. that always associate $\mathbf{p}_i$ to the point $\mathbf{p}$, for all possible mediators.(see Figure 2.2). The set of all Voronoï cells is called the Voronoï diagram of $\mathcal{S}$ (see Figure 2.2).

Let us present some remarkable properties of the Voronoï diagram.

Property 2.1.1 *Voronoï cells are convex polytopes.*

By definition, each Voronoi region $\mathcal{C}(\mathbf{p}_i)$ is the intersection of open half planes containing vertex $\mathbf{p}_i$. Therefore, $\mathcal{C}(\mathbf{p}_i)$ is open and convex. Different Voronoï regions are disjoint. Voronoï cells are polygons in 2D, polyhedra in 3D and d-polytopes in dD. Voronoï cells are either closed or open. They can only be open for points that are located on the convex hull of the 2D domain. A point $\mathbf{p}_i$ of $\mathcal{S}$ lies on the convex hull of $\mathcal{S}$ if and only if its Voronoï cell $\mathcal{C}(\mathbf{p}_i)$ is unbounded.

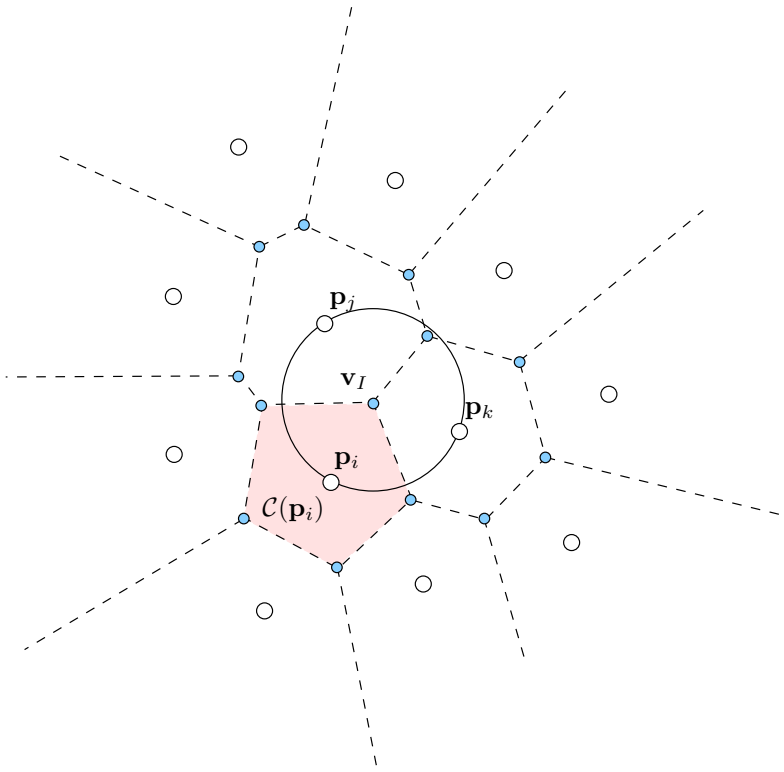


Figure 2.2: The Voronoï diagram. The Voronoï cell $\mathcal{C}(\mathbf{p}_i)$ relative to vertex $\mathbf{p}_i$ is coloured.

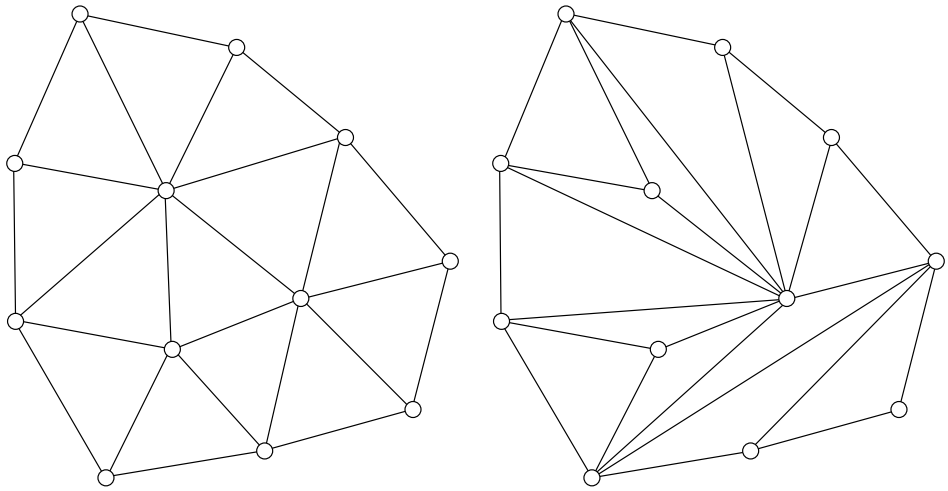


Figure 2.3: Two triangulations of $\mathcal{S}$, both containing $n_t = 13$ triangles.

Property 2.1.2 *Voronoï points $\mathbf{v}_I$ are always located at intersection of 3 mediators.*

Property 2.1.2 is only true if there exist no quadruplets of points in $\mathcal{S}$ that are cocircular. If such a set exists, it may happen that a vertex of the Voronoï is at the the intersection of more than 3 mediators. Consider the example of Figure 2.2 where $\mathbf{v}_i$ is at the intersecion of $\mathcal{C}(\mathbf{p}_i)$, $\mathcal{C}(\mathbf{p}_j)$ and $\mathcal{C}(\mathbf{p}_k)$. Voronoï point $\mathbf{v}_I$ is loacted at the center of the unique circle passing through $\mathbf{p}_i$, $\mathbf{p}_j$ and $\mathbf{p}_k$.

2.1.2 The Delaunay Triangulation

We first define what is a triangulation of $\mathcal{S}$: it is a planar subdivision whose bounded faces are triangles and whose vertices are the points $\mathbf{p}_i$ of $\mathcal{S}$. Note that all triangulations do not cover the same subset of R^2. In what follows, we consider that the domain to triangulate is the convex hull of $\mathcal{S}$.

Even with the same domain to cover, several triangulations are possible (see Figure 2.3). Yet, every triangulation has the same number of triangles and edges!

Property 2.1.3 *Let $\mathcal{S}$ be a set of N points in the plane, not all collinear, and let N_h denote the number of points in $\mathcal{S}$ that lie on the convex hull of*

$\mathcal{S}$. Then any triangulation of $\mathcal{S}$ has $n_t(N, N_h) = 2N - 2 - N_h$ triangles and $n_e(N, N_h) = 3N - 3 - N_h$ edges.

Proof Consider first that all the points in $\mathcal{S}$ are in the convex hull: $N = N_h$. A triangulation consist simply in triangles $t_I(\mathbf{p}_1, \mathbf{p}_i, \mathbf{p}_{i+1})$, $i = 2, \ldots, N-1$. The number of triangle is therefore $n_t = N - 2$ wich is consistent with the formula. The number of edges in the triangulation is calculated as follows: $N_h = N$ edges on the convex hull and $N - 3$ internal edges wich gives $n_e = N + N - 3 = 2N - 3$ which is again consistent with the proposition.

The rest of the proof works using a recurrence argument. Assume that formulas are true for N. Let us now consider one new vertex $\mathbf{p}_{N+1}$ that lies inside the convex hull. This new vertex lies inside one of the existing triangles, say $t_I(\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k)$. We remove t_I and replace it by three new triangles $t_J(\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_{N+1})$, $t_J(\mathbf{p}_j, \mathbf{p}_k, \mathbf{p}_{N+1})$, $t_J(\mathbf{p}_k, \mathbf{p}_i, \mathbf{p}_{N+1})$. We have therefore $n_t(N_h, N_h) = N_h - 2$ and $n_t(N + 1, N_h) = n_t(N, N_h) + 2$ which gives $n_t(N, N_h) = N_h - 2 + 2(N - N_h) = 2N - 2 - N_h$. Similarly, three new edges have been added in the process. Then, $n_e(N_h, N_h) = 2N_h - 3$ and $n_e(N+1, N_h) = n_e(N, N_h)+3$ which gives $n_e(N, N_h) = 2N_h - 3 + 3(N - N_h) = 3N - 3 - N_h$. ∎

Consider a triangulation $\mathcal{T}$ with n_t triangles. This triangulation has $3n_t$ internal angles. Consider the vector of angles $A(\mathcal{T}) = (\alpha_1, \ldots, \alpha_{3n_t})$ sorted by increasing values. We can define such a vector for any triangulation of the convex hull of the domain. Each of those vectors has the same length and it is therefore possible to compare them, e.g. lexicographically. We say that one given triangulation $\mathcal{T}$ is angle-optimal if $A(\mathcal{T}) \leq A(\mathcal{T}')$, $\forall \mathcal{T}'$. According to that criterion, left triangulation of Figure 2.3 is better than the right one.

Angle-optimal trianglations have interresting interpolation properties and it is therefore useful to find methods that allow to construct such triangulations.

For a given set of points, the (unique) angle-optimal triangulation is the triangulation that has the highest minimal angle. Let us see now how to build such a triangulation.

Consider now the edge $e(\mathbf{p}_4, \mathbf{p}_6)$ of Figure 2.4. This edge is surrounded by two triangles $t_1(\mathbf{p}_4, \mathbf{p}_6, \mathbf{p}_1)$ and $t_2(\mathbf{p}_4, \mathbf{p}_6, \mathbf{p}_2)$. An edge swap is a local mesh modification operator that consist in changing locally the triangulation by replacing edge $e(\mathbf{p}_4, \mathbf{p}_6)$ by $e'(\mathbf{p}_1, \mathbf{p}_2)$. Triangles $t_1(\mathbf{p}_4, \mathbf{p}_6, \mathbf{p}_1)$ and $t_2(\mathbf{p}_4, \mathbf{p}_6, \mathbf{p}_2)$ are replaced by $t'_1(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_4)$ and $t'_2(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_6)$. Figure 2.4 illustrate the edge swap operation.

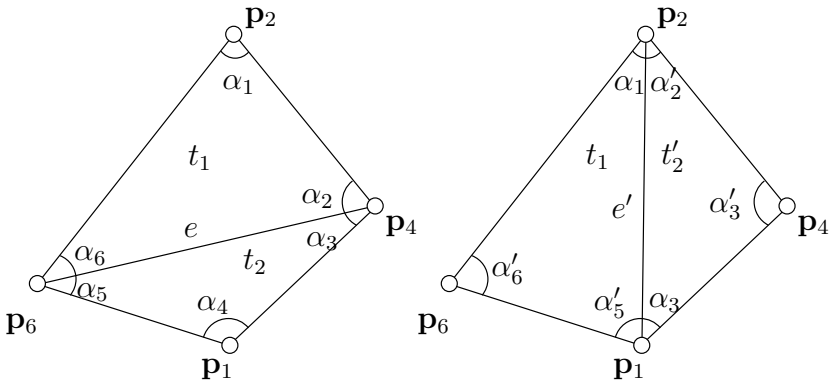


Figure 2.4: Edge swap.

It is indeed possible to use the edge swap operator in order to build angle-optimal triangulation. In that purpose, we can simply decide to swap edge e if

$$\min(\alpha_1, \ldots, \alpha_6) < \min(\alpha'_1, \ldots, \alpha'_6).$$

Such an edge is said *invalid*. Given a finite set of points, there is a finite number of possible triangulations. When the angle criterion is applied, every edge swap produce a new triangulation that is better than the actual one. Therefore, building the angle-optimal triangulation consist in looping over all the edges of the mesh and swap them until the optimal configuration is attained i.e. when no invalid edges remain in the triangulation. Figure 2.5 illustrate that procedure.

Yet, computing angles is a costly operation and it is possible to use a simpler and cheaper rule to verify the validity of an edge.

Property 2.1.4 *Let $\mathcal{C}$ be a circle, l a line intersecting $\mathcal{C}$ in points $\mathbf{p}_4$ and $\mathbf{p}_6$ and $\mathbf{p}_1$, and $\mathbf{p}_2$, $\mathbf{p}_3$ and $\mathbf{p}_5$ points lying on the same side of l. Suppose that $\mathbf{p}_2$ and $\mathbf{p}_3$ lie on $\mathcal{C}$, that $\mathbf{p}_5$ lies inside $\mathcal{C}$, and that $\mathbf{p}_1$ lies outside $\mathcal{C}$. Then (see Fig. 2.6):*

$$\alpha_1 < \alpha_2 = \alpha_3 < \alpha_5.$$

Property 2.1.5 *[Lawson's Criterion] Consider an edge e with its two neighboring triangles t_1 and t_2. Consider the circumcircle $\mathcal{C}$ of triangle t_1 and point $\mathbf{p}$ that belongs to t_2 but not to t_1. Edge e is invalid if $p \in \mathcal{C}$.*

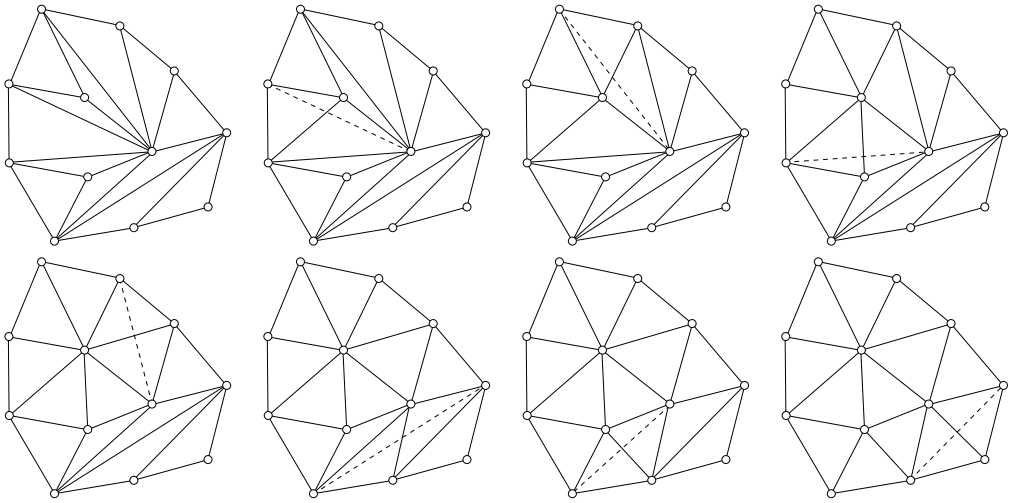


Figure 2.5: Building an angle-optimal triangulation using swaps.

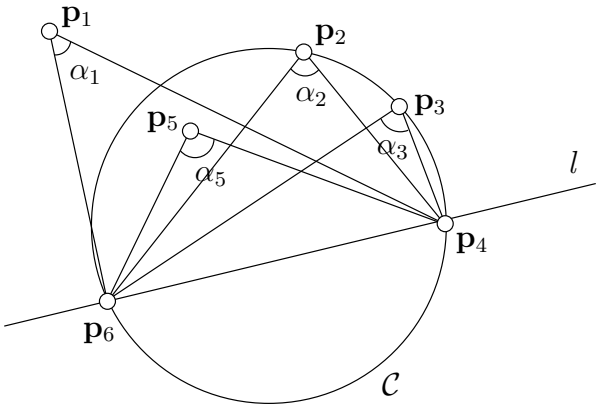


Figure 2.6: Theorem of Thalès

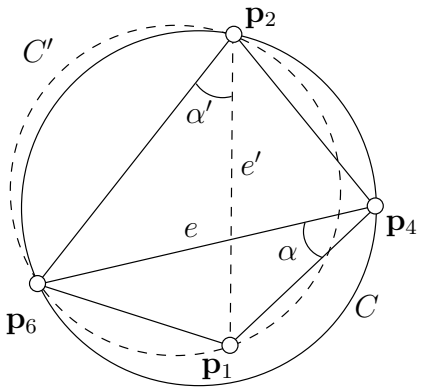


Figure 2.7: Lawson's criterion

Property 2.1.5 has been demonstrated by Sibson in [**?**]. Consider Figure 2.7. The proof consist in demonstrating that $\alpha' > \alpha$ if $\mathbf{p}_1 \in C$. The demonstration make use of property 2.1.4 (Thalès's Theorem).

Note that, when two neighboring triangles separated by an edge e form a concave quadrilateral, edge e is always valid.

Procedure described in Figure 2.5 allow to build angle-optimal triangulations. Yet, it can be shown that such an algorithm is slow in practice.

There exists a much faster manner to build angle optimal triangulations of $\mathcal{S}$ that is based on the Voronoï diagram. This triangulation $DT(\mathcal{S})$ is called the *Delaunay triangulation* and its construction works as follows.

First recall that each Voronoï cell $\mathcal{C}(\mathbf{p}_i)$ is associated to one only point $\mathbf{p}_I$ of $\mathcal{S}$.

Consider a Voronoï point $\mathbf{v}_I$ that at the meeting point of 3 Voronoï cells $\mathcal{C}(\mathbf{p}_i)$, $\mathcal{C}(\mathbf{p}_j)$, $\mathcal{C}(\mathbf{p}_k)$. Beacuse of the Voronoï property, point $\mathbf{v}$ is at the circumcenter of triangle $t_I(\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k)$. Triangle $t_I \in DT(\mathcal{S})$ is one of the triangles of the Delaunay triangulation.

The resulting figure is a triangulation (see Figure 2.8).

Property 2.1.6 *Let $\mathcal{S}$ be a set of points in the plane.*

(i) Three points $\mathbf{p}_i$, $\mathbf{p}_j$ and $\mathbf{p}_k \in \mathcal{S}$ are vertices of the same triangle t_I of the Delaunay triangulation if and only if the circumcircle $C(t_I)$ contains no point of $\mathcal{S}$ in its interior.

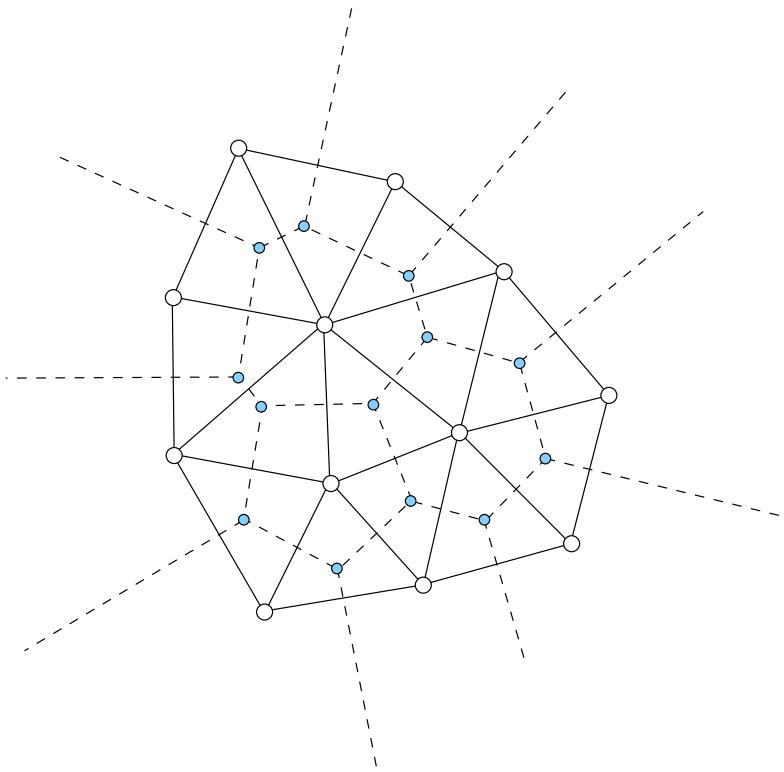


Figure 2.8: The Voronoï diagram and its associated Delaunay triangulation

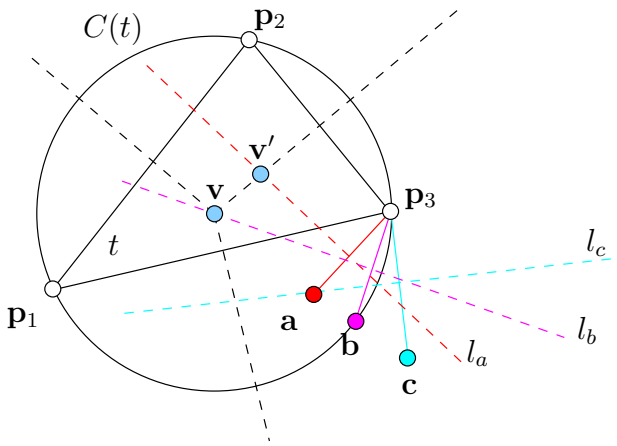


Figure 2.9: Illustration of why property (i) of 2.1.6 is true.

(ii) Two points $\mathbf{p}_i$ *and* $\mathbf{p}_j \in \mathcal{S}$ *form an edge* e_I *of the Delaunay triangulation if and only if there is a closed disc* C *that contains* $\mathbf{p}_i$ *and* $\mathbf{p}_j$ *on its boundary and does not contain any other point of* $\mathcal{S}$.

Proof Property (i) of 2.1.6, also called the incircle property, is a simple consequense of the properties of construction of the Voronoï diagram. Figure 2.11 show one triangle t and its circumcenter $\mathbf{v}$. If a point like $\mathbf{a}$ exist in $\mathcal{S}$, triangle t is cannot be in the Delaunay triangulation because point $\mathbf{a}$ is closer to $\mathbf{v}$ that at least one of the three points $\mathbf{p}_1$ $\mathbf{p}_2$ or $\mathbf{p}_3$. Therefore, as it is drawn in the Figure, mediator l_a crosses another mediator at point $\mathbf{v}'$ inside the triangle, which is impossible. ∎

Property 2.1.7 *Let* $\mathcal{S}$ *be a set of points in the plane in general position. A triangulation* $\mathcal{T}$ *of* $\mathcal{S}$ *is angle-optimal if and only if* $\mathcal{T}$ *is the Delaunay triangulation of* $\mathcal{S}$.

Proof We shall prove that the angle-optimal triangulation is the Delaunay triangulation by contradiction. So assume $\mathcal{T}$ is a legal triangulation of $\mathcal{S}$ that is not a Delaunay triangulation. By Property (ii) of 2.1.6, this means that there is a triangle $t(\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k)$ such that the circumcircle $C(t)$ contains a point $\mathbf{p}_l \in \mathcal{S}$ in its interior. Let $e(\mathbf{p}_i, \mathbf{p}_j)$ be the edge of $t'(\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_l)$ such that the triangle t' does not intersect t. Of all such pairs $(\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k, \mathbf{p}_l)$ in $\mathcal{T}$,

choose the one that maximizes the angle $\mathbf{p}_i, \mathbf{p}_l, \mathbf{p}_j$. Now look at the triangle $t''(\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_m)$ adjacent to t along e. Since $\mathcal{T}$ is angle-optimal, e is legal. By Property 2.1.5, this implies that $\mathbf{p}_m$ does not lie in the interior of $C(t)$. The circumcircle $C(t'')$ contains the part of $C(t)$ that is separated from t by e. Consequently, $\mathbf{p}_l \in C(t'')$ Assume that $e'(\mathbf{p}_j, \mathbf{p}_m)$ is the edge of t'' such that triangle $t'''(\mathbf{p}_j, \mathbf{p}_m, \mathbf{p}_l)$ does not intersect t''. But now angle $\mathbf{p}_j, \mathbf{p}_l, \mathbf{p}_m >$ angle $\mathbf{p}_i, \mathbf{p}_l, \mathbf{p}_j$ by Thalès's Theorem, contradicting the definition of the pair $(\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k, \mathbf{p}_l)$. ∎

Note that the unicity of the Delaunay triangulation is only guaranteed when there exists no quadruplets of points in the set $\mathcal{S}$ that are cocircular, i.e. for points in general position. When $\mathcal{S}$ is not in general position, some of the Delaunay triangulations may not be angle-optimal.

2.1.3 Construction of Delaunay Triangulations

There are two categories of algorithms that allow to create Delaunay triangulations.

Incremental Construction of Delaunay Triangulations: the Delaunay kernel

We consider a triangulation $\mathcal{T}_i$ and a point $\mathbf{p}_{i+1}$ inside $\mathcal{T}_i$. The cavity $\mathcal{C}(\mathcal{T}_i, \mathbf{p}_{i+1})$ associated to $\mathcal{T}_i$ and $\mathbf{p}_{i+1}$ is the set of all the triangles for which their circumsphere contains $\mathbf{p}_{i+1}$.

We consider a triangulation $\mathcal{T}_i$ and a point $\mathbf{p}_{i+1}$ outside $\mathcal{T}_i$. The cavity $\mathcal{C}_p(\mathcal{T}_i, \mathbf{p}_{i+1})$ associated to $\mathcal{T}_i$ and $\mathbf{p}_{i+1}$ is the set of all the triangles for which their circumsphere contains $\mathbf{p}_{i+1}$ completed by all triangles that can be formed by joining all the edges of $\mathcal{T}_i$ visible by $\mathbf{p}_{i+1}$.

Property 2.1.8 *The cavity $\mathcal{C}_p(\mathcal{T}_i, \mathbf{p}_{i+1})$ is star shaped and $\mathbf{p}_{i+1}$ belong to its Haddad kernel.*

A polygon is star-shaped with respect to $\mathbf{p}_{i+1}$ if, for each point $\mathbf{p}_k$, $k = 1 \ldots, N_c$ of the polygon the edge $e(\mathbf{p}_{i+1}, \mathbf{p}_k)$ lies entirely within the polygon. The set of all points $\mathbf{p}_{i+1}$ with the described property is called the kernel of the polygon, or its Haddad Kernel.

Figure **??** illustrate what we call a star shaped cavity. A convex polygons is star shaped, but the inverse is not true. Thanks to property 2.1.8, it is quite

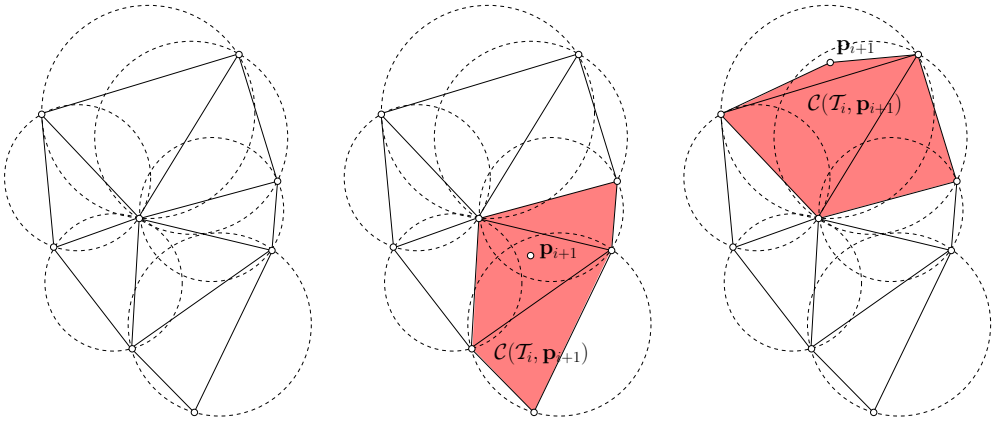


Figure 2.10: Delaunay triangulation $\mathcal{T}_i$ (left). The Delaunay cavity $\mathcal{C}_p(\mathcal{T}_i, \mathbf{p}_{i+1})$ is represented in both middle ($\mathbf{p}_{i+1}$ is inside $\mathcal{T}_i$) and right ($\mathbf{p}_{i+1}$ is outside $\mathcal{T}_i$) figures.

easy to build a triangulation of the cavity, simply by adding N_c triangles. This set of new triangles is called the ball $\mathcal{B}(\mathcal{T}_i, \mathbf{p}_{i+1})$.

Let us assume that we have build the Delaunay Triangulation $\mathcal{T}_i$ with the first i points of the set $\mathcal{S}$.

It is possible to build iteratively the delaunay triangulation $\mathcal{T}_{i+1}$, i.e. using $\mathcal{T}_i$ and $\mathbf{p}_{i+1}$. We define the Delaunay kernel as the following procedure

$$\mathcal{T}_{i+1} = \mathcal{T}_i - \mathcal{C}(\mathcal{T}_i, \mathbf{p}_{i+1}) + \mathcal{B}(\mathcal{T}_i, \mathbf{p}_{i+1})$$

that consist in removing from the triangulation elements of $\mathcal{C}(\mathcal{T}_i, \mathbf{p}_{i+1})$ that violate the incircle property (i) of 2.1.6 and subsequently to triangulate the cavity with the ball $\mathcal{B}(\mathcal{T}_i, \mathbf{p}_{i+1})$ Figure 2.11 illustrate the Delaunay kernel.

Property 2.1.9 *if $\mathcal{T}_i$ is the Delaunay triangulation of the convex hull of the i first points of a set $\mathcal{S}$, then $\mathcal{T}_{i+1}$, the triangulation constructed using the Delaunay kernel is a Delaunay triangulation*

Recursive Construction of Delaunay Triangulations: Divide and Conquer

There exists a "Divide and Conquer" type of algorithm for triangulating a known set of points. It allows a $\mathcal{O}(N \log N)$ complexity. Divide and conquer

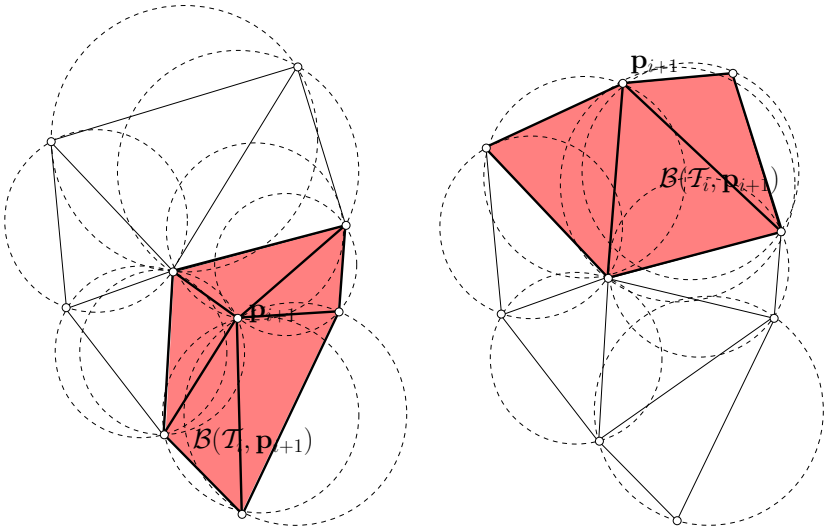


Figure 2.11: Delaunay triangulation $\mathcal{T}_{i+1}$ In the left Figure, $\mathbf{p}_{i+1}$ is inside $\mathcal{T}_i$ and, in the right Figure, $\mathbf{p}_{i+1}$ is outside $\mathcal{T}_i$.

(DC) is an important algorithm design paradigm. It works by recursively breaking down a problem into two or more sub-problems of the same (or related) type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.

Divide and Conquer algorithm for triangulations in two dimensions is due to Lee and Schachter which was improved by Guibas and Stolfi and later by Dwyer. In this algorithm, one recursively draws a line to split the vertices into two sets. The Delaunay triangulation is computed for each set, and then the two sets are merged along the splitting line

Algorithm 1 gives the pseudo code of the Divide and Conquer Delaunay triangulator of Gmsh.

The set of points is assumed to be sorted lexicographically, i.e. points are sorted form left to right and from bottom to top. Points are subsequently numbered using their lexicographic index (PointIndex). Figure 2.12 shows a set

$$\mathcal{S} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_9\}$$

of 9 points sorted lexicographically.

A doublely linked circular list is assigned to every point $\mathbf{p}_i$ of the set $\mathcal{S}$.

Algorithm 1 DelDC(PointIndex left, PointIndex right)

```
1: n = right - left + 1
2: if n = 2 then
3:    InsertInDList (left,right)
4: else if n = 3 then
5:    InsertInDList (left,right)
6:    InsertInDList (left,left+1)
7:    InsertInDList (left+1,right)
8: else if n > 3 then
9:    middle = (left + right) ≫ 1
10:   DelDC ( left, middle )
11:   DelDC ( middle+1, right )
12:   DelMerge ( left, middle , right )
13: end if
```

This data structure contains every point $\mathbf{p}_j$ that is connected through a mesh edge to point $\mathbf{p}_i$. Datas are stored in the list in the counter-clockwise sense, as it is pictured in Figure 2.13. The list is doubly-linked i.e. it is possible to advance forward and backward.

Four functions have to be implemented in order to use the doubly-linked lists. Function `InsertInDList(i,j)` adds edge `(i,j)` in the triangulation i.e. inserts point `j` in the list of adjacencies of `i` and inserts `i` in the list of adjacencies of `j`. Function `DeleteInDList(i,j)` deletes edge `(i,j)` from the triangulation. Functions `k = Predecessor(j)` and `k = Successor(i,j)` respectively computes the predecessor and the successor `k` to point `j` in the adjacency list of `i`.

The `DelDC` function (see algorithm 1) builds a Delaunay triangulation for the subset of points

$$\mathcal{S}_{subset} = \{\mathbf{p}_{left}, \mathbf{p}_{left+1}, \ldots, \mathbf{p}_{right}\}.$$

Three trivial cases are treated:

- If the number of points $n = right - left + 1$ in $\mathcal{S}_{subset}$ is less that $n = 2$, the triangulation contains no edge;
- If $n = 2$, one edge is added in the triangulation;
- If $n = 3$, three edges are created that form one triangle.

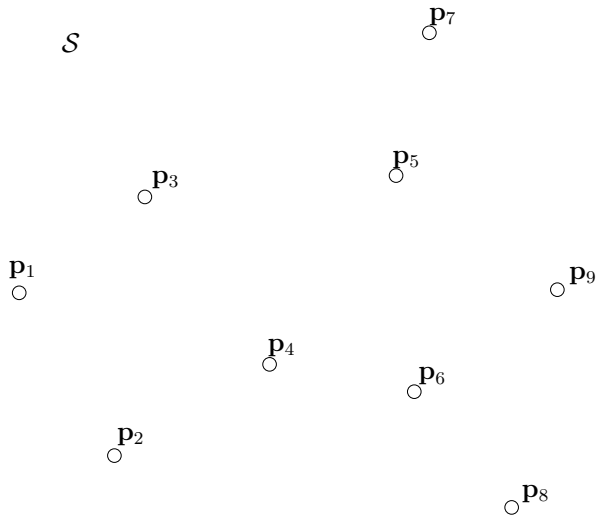


Figure 2.12: A set of points sorted lexicographically

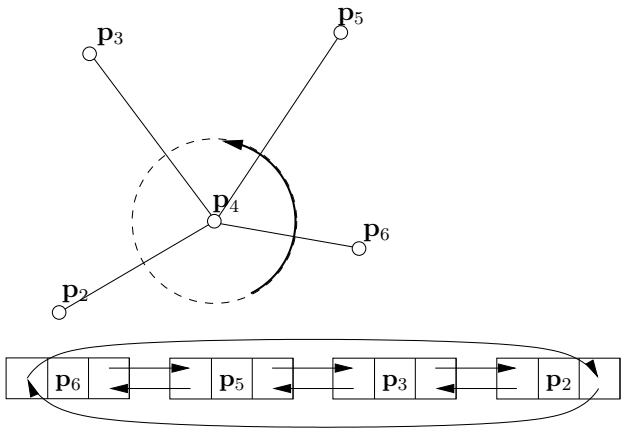


Figure 2.13: The doubly-linked circular list associated to point $\mathbf{p}_4$.

If $n > 3$, the `middle` point of $\mathcal{S}_{subset}$ is computed. The `DelDC` function is called twice recursively with ranges of half the initial size: `DelDC(left,middle)` and `DelDC(middle+1,right)`. When those two function calls are terminated, both sets of points going from `left` to `middle` and from `middle+1` to `right` are Delaunay triangulations. A procedure that we call DelMerge enables to merge the two disconnected Delaunay triangulationsto form a unique Delaunay triangulation of the whole range going from `left` to `right`

The first part of the `DelDC` procedure is illustrated graphically at Figure 2.14. The set of points is first split in two subsets $\mathcal{S}_1$ and $\mathcal{S}_2$ (part (a) of Figure 2.14). Then, each part is split again, giving four subsest $\mathcal{S}_{11}$, $\mathcal{S}_{12}$, $\mathcal{S}_{21}$ and $\mathcal{S}_{22}$ (part (b) of Figure 2.14). The recursion stops when sub-sets of points have at most three points. At this point, every subset can be processed trivially (part (c) of Figure 2.14).

The second part of the `DelDC` procedure is illustrated graphically at Figure 2.15. Delaunay triangulations $DT(\mathcal{S}_{11})$ and $DT(\mathcal{S}_{12})$ are merged as well as $DT(\mathcal{S}_{21})$ and $DT(\mathcal{S}_{22})$, forming two Delaunay triangulations $DT(\mathcal{S}_1)$ and $DT(\mathcal{S}_2)$ (part (a) of Figure 2.15). Then $DT(\mathcal{S}_1)$ and $DT(\mathcal{S}_2)$ are merged to form the final result $DT(\mathcal{S})$ (part (b) of Figure 2.15).

Let us now describe the most complex part of the algorithm, i.e. the `DelMerge` procedure. We will illustrate it using $DT(\mathcal{S}_1)$ and $DT(\mathcal{S}_2)$. The `DelMerge` procedure stars by computing the lower common tangent and the upper comman tangent of the union of both triangulations (Figure 2.16).

Edges UCT and LCT are edges of the Delaunay triangulation of the union of the two sets because they belong to the convex hull and Delaunay triangulations are triangulations of the convex hull. The merging process `MergeDC` aims at filling the empty gap between the two triangulations $DT(\mathcal{S}_1)$ and $DT(\mathcal{S}_2)$,

The `MergeDC` procedure starts from the LCT with its two points l and r. Even though triangles surrounding those two points are part of respectively $DT(\mathcal{S}_1)$ and $DT(\mathcal{S}_2)$, they may not be part of $DT(\mathcal{S})$.

Consider point r in Figure 2.16. Edge (r,r1) is the one the is the `Predecessor` of LCT in r's adjacency list and edge (r,r2) is the one the is the `Predecessor` of (r,r1) in r's adjacency list. We apply Lawson's criterion to edge (r,r1), i.e. look if point r2 lies inside the circum circle of triangle (l,r,r1). If it is not the case, then triangle (l,r,r1) cannot be excluded regarding to $DT(\mathcal{S}_2)$.

Similarly, consider point l in Figure 2.16. Edge (l,l1) is the one the is the `Successor` of LCT in l's adjacency list and edge (l,l2) is the one the is the `Successor` of (l,l1) in l's adjacency list. We apply Lawson's criterion to

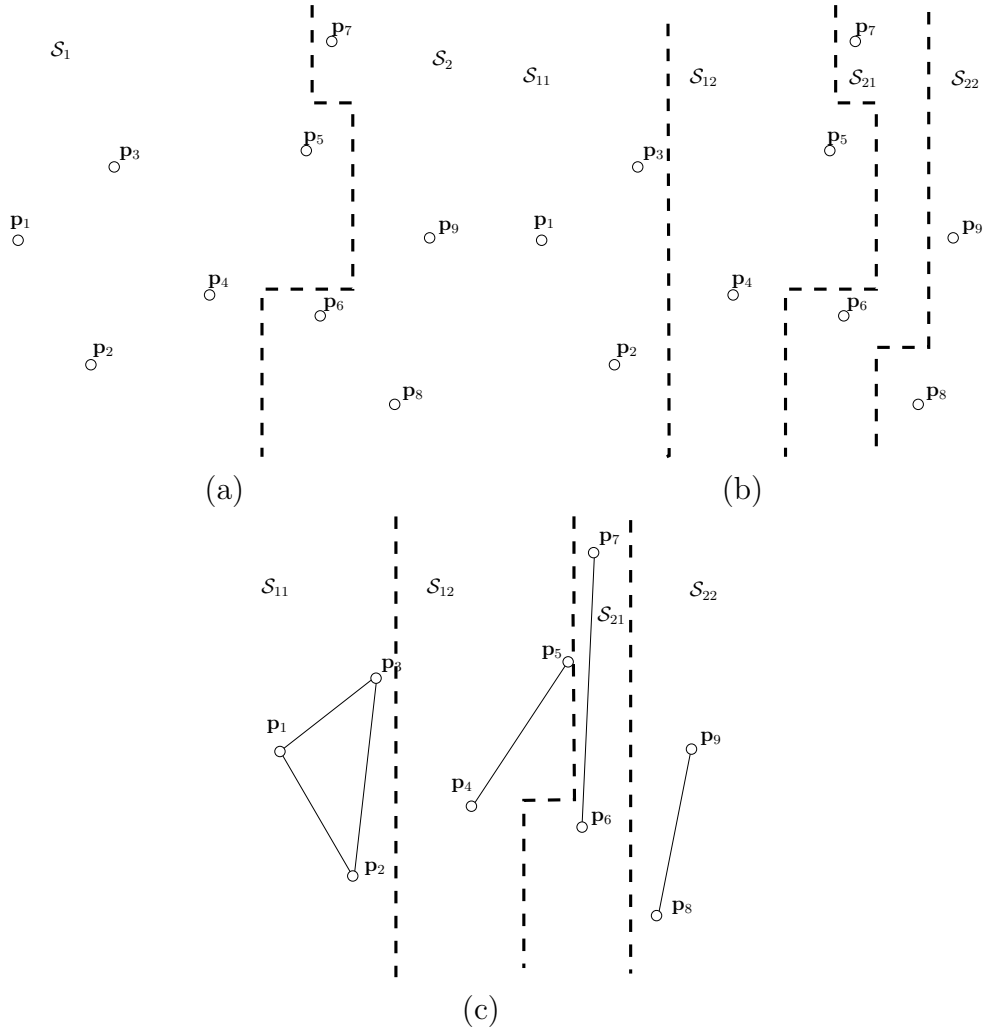


Figure 2.14: Illustration of the Divide and Conquer procedure.

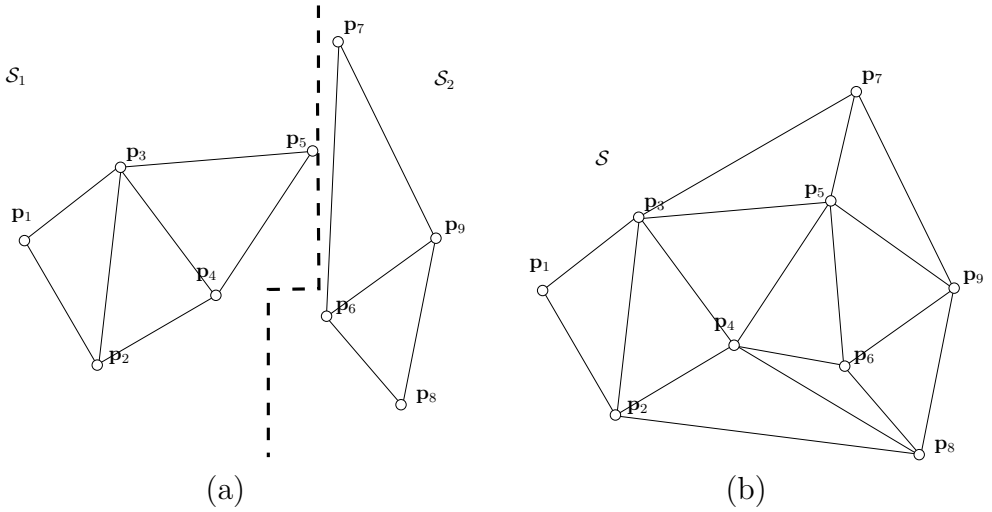


Figure 2.15: Illustration of the Divide and Conquer procedure.

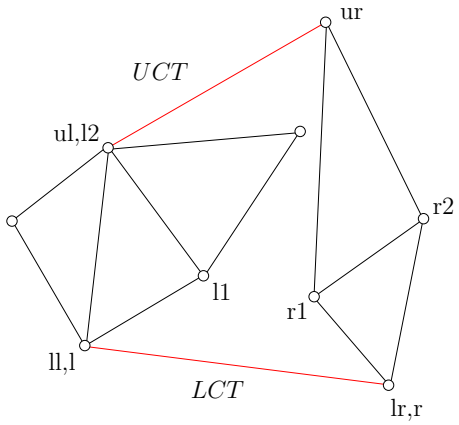


Figure 2.16: Computation of both lower common tangent (LCT) and the upper common tangent (UCT) of the union of both triangulations. Point indices like l,ll or l1 refer to the notations of algorithm 2

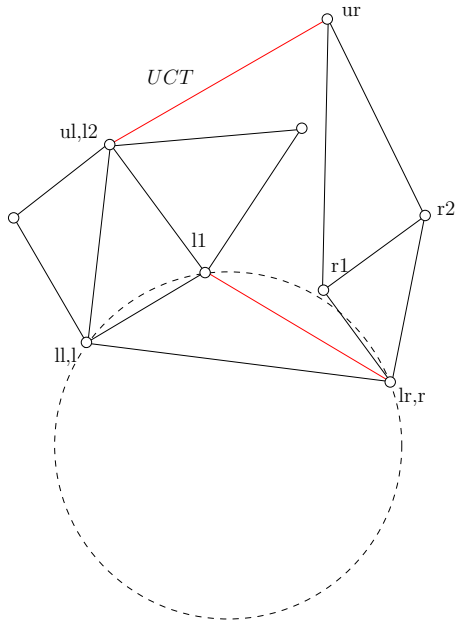


Figure 2.17: One step of the merging procedure.

edge (l,l1), i.e. look if point l2 lies inside the circum circle of triangle (l,l,l1). If it is not the case, then triangle (r,l,l1) cannot be excluded regarding to $DT(\mathcal{S}_1)$.

Among the two possible new edges (r,l1) and (l,r2) that may be added to the triangulation, edge (r,l1) is chosen because it satisfies the incircle property (see Figure 2.17).

In the particular case of Figure 2.16, the process can be continued two times (see Figure 2.1.3).

At the next step (Figure 2.19, part (a)), it is clear that edge (r,r1) cannot be part of the triangulation because triangle (l,r,r1) has its circumcircle that contains r2. Note that Lawson's criterion is symmetric, which means that, if the circumcircle of triangle (l,r,r1) contains r2, then the circumcircle of triangle (l,r,r2) contains r1. At this point, we simply replace edge (r,r1) by edge (l,r2) and continue the process.

An implementation of that algorithm is provided in Gmsh. Source code can be found in `gmsh/Mesh/DivideAndConquer.cpp`.

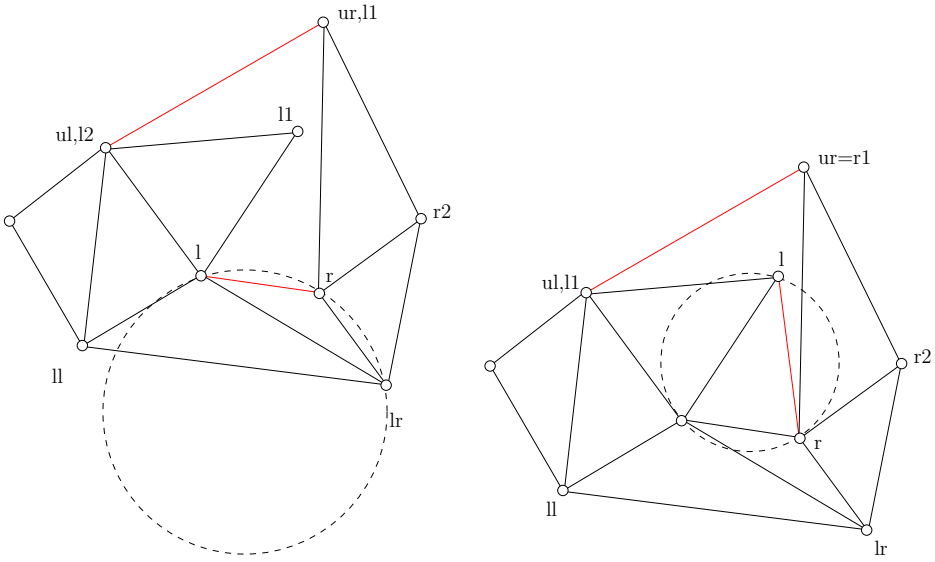


Figure 2.18: Two steps of the merging procedure.

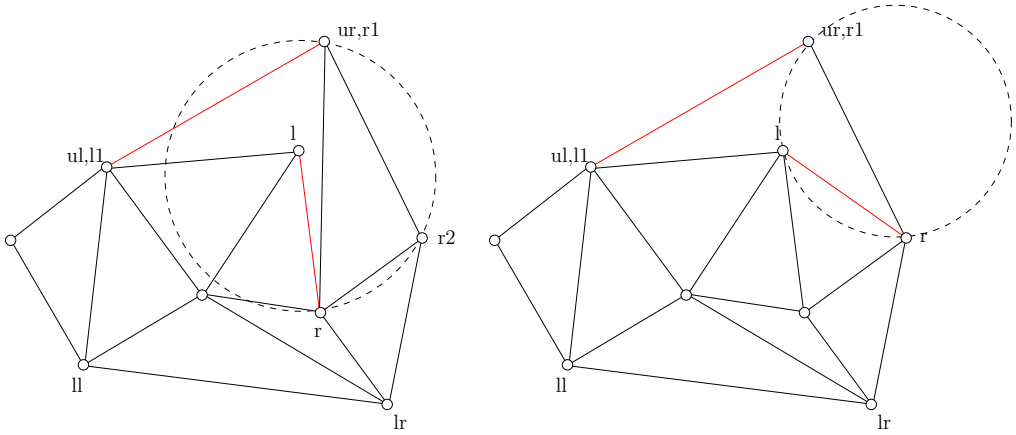


Figure 2.19: Three steps of the merging procedure.

Algorithm 2 MergeDC(PointIndex left, PointIndex mid, PointIndex right)

```
 1: [ll,lr] = LowerCommonTangent(left, mid, right) , l = ll, r = lr
 2: [ul,ur] = UpperCommonTangent(left, mid, right)
 3: while l ≠ ul or r ≠ ur  do
 4:    b = false
 5:    InsertInDList (l,r)
 6:    r1 = Predecessor (r,l)
 7:    while true do
 8:       r2 = Predecessor(r, r1)
 9:       if not InCirlce (l,r,r1,r2) then
10:          break
11:       else
12:          DeleteFromList (r,r1), r2=r1
13:       end if
14:    end while
15:    l1 = Successor (l,r)
16:    while true do
17:       l2 = Successor(l, l1)
18:       if not InCirlce (r,l,l1,l2) then
19:          break
20:       else
21:          DeleteFromList (l,l1), l2=l1
22:       end if
23:    end while
24:    if b or InCircle(l,r,r1,l1) then
          r = r1
25:    else
          l = l1
26:    end if
27: end while
28: InsertInDList (l,r)
```

2.2 Nearest neighbors

2.3 Point location

Chapter 3

Solid Models

A solid model is a computer model of a 3D solid. It is a virtual representation of the shape of a solid. Solid models can be simple parts (Figure 3.1, part (a)) or complex assemblies of multiple parts (Figure 3.1, part (b)).

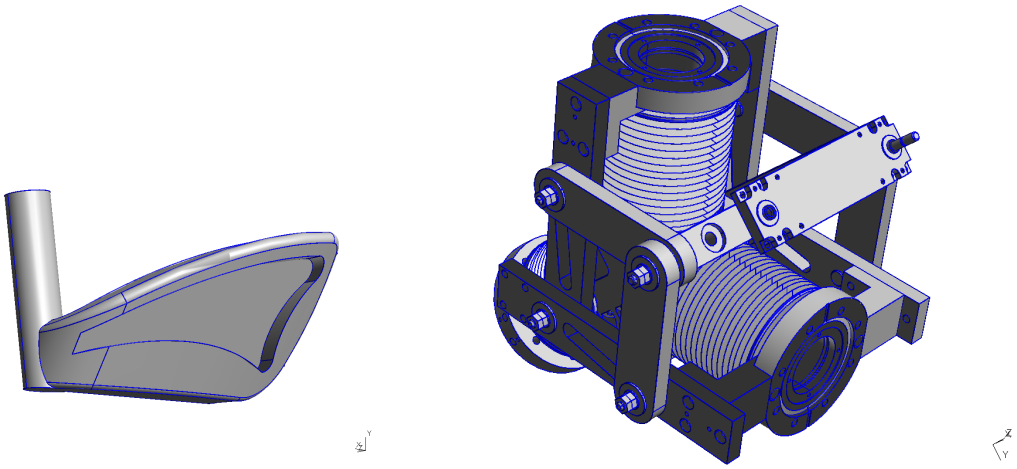
We aim here at explaining how such solids can be described on a computer. We will principally focus on the ability of such solid models to serve as input to numerical simulations. In particular, we will adress the issues related to finite element mesh generation.

3.1 Boundary Representation of Solids

Any 3-D model can be defined using its Boundary Representation (BRep): a volume (called *region*) is bounded by a set of surfaces, and a surface is bounded by a series of curves; a curve is bounded by two end points. Therefore, four kinds of *model entities* are defined:

1. Model Vertices G_i^0 that are topological entities of dimension 0,
2. Model Edges G_i^1 that are topological entities of dimension 1,
3. Model Faces G_i^2 that are topological entities of dimension 2,
4. Model Regions G_i^3 that are topological entities of dimension 3.

The boundary representation is purely topological, i.e., it only deals with adjacencies in the model. In a boundary representation of a model entity, each adjacency is signed: model entities are oriented.



(a) (b)

Figure 3.1: A simple part (a) and a complex assembly (b).

As a first illustration, let us look at the 2D model presented in Figure 3.2. This model has 1 model faces, 5 model edges and 5 model vertices.

Model edge G_1^1's boundary, ∂G_1^1 consist in two signed vertices:

$$\partial G_1^1 = -G_2^0 + G_3^0$$

which means that model edge G_1^1 goes from model vertex G_2^0 to model vertex G_3^0.

Model face G_1^2's boundary consist in two closed edge loops that consist of respectively four and one signed model edges:

$$\partial G_1^2 = \{-G_1^1 + G_4^1 + G_3^1 - G_2^1, G_1^5\}. \tag{3.1}$$

Model edge G_5^1 is periodic:

$$\partial G_1^5 = G_5^0 - G_5^0 = 0$$

which makes perfect sense: the boundary of a closed curve is zero. For being consistent, the boundary of a closed edge loop has also to be zero. In other words,

$$\begin{aligned} \partial\partial G_1^2 &= \{-\partial G_1^1 + \partial G_2^1 + \partial G_3^1 - \partial G_4^1, \partial G_1^5\} \\ &= \{-(G_3^0 - G_0^2) + (G_4^0 - G_3^2) + (G_4^0 - G_1^2) - (G_1^0 - G_2^2), G_5^0 - G_5^0\} \\ &= \{0, 0\}. \end{aligned}$$

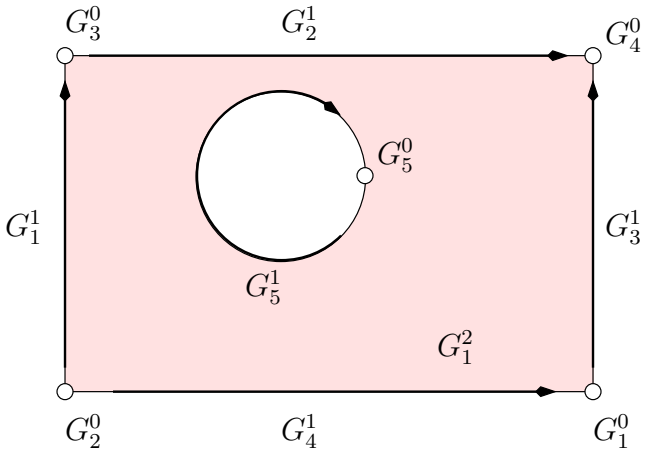


Figure 3.2: A simple 2D model.

Vertices that bound model curves of a model face have to appear once opsitively and once negatively. Two things have to be noted. First, $\partial G_i^0 =$ i.e. the boundary of a model vertex is zero. Then, changing the sign of the boundary representation of a model face

$$-\partial G_1^2 = -\{-G_1^1 + G_4^1 + G_3^1 - G_2^1, G_1^5\}. \tag{3.2}$$

gives the same topological model face, yet with a different sign i.e. a different orientation.

Then, a model region can also be described using its boundary representation. For example the cube of Figure 3.3 with one model region G_1^3 consist in one closed face loop

$$\partial G_1^3 = \{G_1^2 + G_2^2 + G_3^2 + G_4^2 + G_5^2 + G_6^2 + G_7^2 + G_8^2\}$$

with model face orientations that have to verify

$$\partial\partial G_1^3 = 0.$$

Let us now consider the cylindrical rod of Figure 3.4. The model region G_1^3 is bounded by three model faces

$$\partial G_1^3 = G_1^2 + G_2^2 + G_3^2.$$

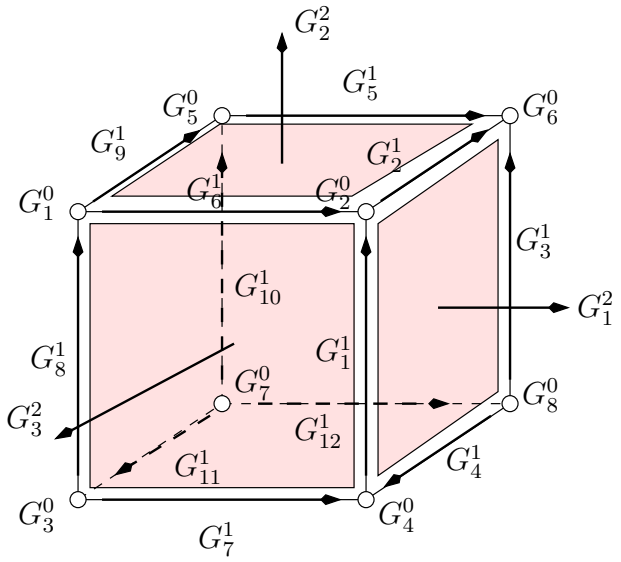


Figure 3.3: A cube.

Model faces G_1^2 and G_2^2 are usual model faces composed of one only closed edge loop:

$$\partial G_1^2 = G_1^1 \text{ and } \partial G_2^2 = -G_3^1.$$

Model face G_3^2 is different. It is a periodic model face and it has no interior holes. In order to be consistent with the boundary representation, its boundary representation should be one single closed loop. In order to achieve that goal, one seam edge, G_1^1, has to be added in the model. This seam edge acts like G_4^0 which is both the starting point and the ending point of G_3^1. We have then

$$\partial G_3^2 = G_1^1 + G_3^1 - G_1^1 - G_2^1.$$

The seam model edge G_1^1 appears twice in the same face loop and effectively acts as a seam that closes the periodic model face. Seam edges are present in any consistent boundary representation of solid models. It is very important to recognize such seam model edges in order to be able to perform any kind of mesh generation.

Historically, solid models were constructed using a direct approach i.e. each model entity was created using its boundary. Model vertices are initially created. Those vertices are used to define curves boundaries. Then, model

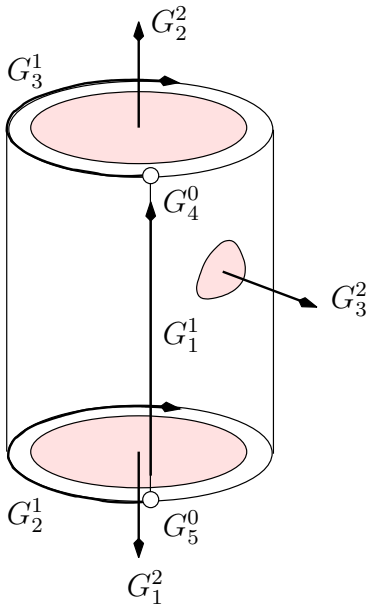


Figure 3.4: A cylindrical rod.

faces are created using model edges and model regions are created using model faces. This approach is still the one that is used in Gmsh's native CAD modeler. Yet, the direct approach for building solid models doest not allow to build complex models like the one presented in Figure 3.1, (b).

Since the 1990's, a new approach for building models has been developped. Constructive solid geometry (CSG) allows to create solid models by using boolean operators to combine objects. The simplest solid objects used for the representation are called primitives (sphere, cylinder, cube ...). Primitives are combined to form complex solid using boolean operators on sets: union, intersection and difference.

The development of robust solid modelers based on CSG has been the starting point of a true boost in engineering analysis. Yet, the development of a robust CSG implies the computation of intersections between complex curves and surfaces. We will briefly discuss that very complex topic.

The great difficulty of building robust solid modelers has had as consequense that very few commercial softwares are still on the market. To our best knowledge, ony one open source CSG modeler is available to date. OpenCascade is a complete CSG modeler that is truly open source. Parts of

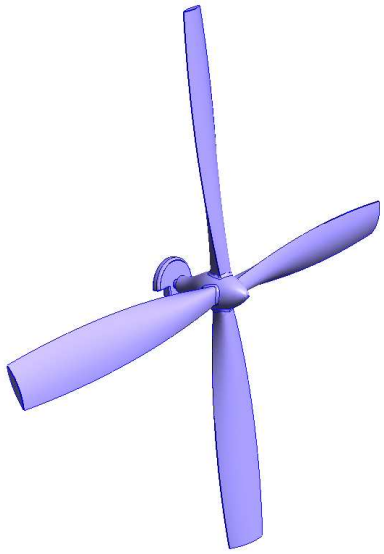


Figure 3.5: CAD model of a propeller (left) and its volume mesh (right)

OpenCascade have been interfaced in Gmsh so that Gmsh is now closer to modern CAD solutions than it was in the past.

As an example, let us consider the CAD model of a propeller presented in Figure 3.5. The model has been created with the OpenCascade solid modeler and has been loaded in Gmsh in its native format (brep). The model contains 101 model vertices, 170 model edges, 76 model faces and one model region.

Figure 3.6, which shows one of the 76 model faces of the propeller in the parametric space (left) and in real space (right). Three features of surface $\mathcal{S}$, common in CAD descriptions, make its meshing non-trivial:

1. $\mathcal{S}$ is periodic. The topology of the model face is modified in order to define its closure properly. A seam is present two times in the closure of the model face. These two occurrences are separated by one period in the parametric space.

2. $\mathcal{S}$ is trimmed: it contains four holes and one of them is crossed by the seam.

3. One of the model edges of $\mathcal{S}$ is degenerated. This is done for accounting of a singular point in the parametrization of the surface. This kind of

Figure 3.6: Geometry of a model face in parametric space (left) and in real space (right). Two seam edges are present in the face. The top model edge is degenerated in one point.

degeneracy is present in many shapes: spheres, cones and other surfaces of revolution.

3.1.1 Geometrical Description

Each model entity G_i^d has a shape, a geometry. More precisely, it is a manifold of dimension d that is embedded in 3-D space.

Solid modelers usually provide a parametrization of the shapes, i.e., a mapping $\mathbf{x} \in \mathcal{R}^d \mapsto \mathbf{x} \in \mathcal{R}^3$. The geometry of a model vertex G_i^0 is simply its 3-D location $\mathbf{x} = (x_1, x_2, x_3)$.

The geometry of a model edge G_i^1 is its underlying curve $\mathcal{C}_i$ with its parametrization

$$t \in [t_1, t_2] \mapsto \mathbf{x}(t) \in \mathcal{R}^3. \tag{3.3}$$

The geometry of a model face G_i^2 is its underlying surface $\mathcal{S}_i$ with its parametrization

$$(u, v) \in \mathcal{R}^2 \mapsto \mathbf{x}(u, v) \in \mathcal{R}^3.$$

The geometry associated to a model region is R^3.

If a curve is included within a surface, it is usually drawn on the parameter plane (u, v) of the surface:

$$t \in [t_1, t_2] \mapsto (u, v) \in \mathcal{R}^2 \mapsto \mathbf{x}\left(u(t), v(t)\right) \in \mathcal{R}^3. \tag{3.4}$$

Differential Geometry of Curves

Consider a segment of curve $\mathcal{C}$ defined by a range of parameter $t \in [t_a, t_b]$, $t_a \geq t_1$, $t_b \leq t_2$. The length of that segment can be computed as

$$\int_{\mathcal{C}} dl$$

with $dl = \sqrt{dx_1^2 + dx_2^2 + dx_3^2}$. Using $\mathcal{C}$'s parametrization (3.3), we have

$$\begin{aligned}\int_{\mathcal{C}} \sqrt{dx_1^2 + dx_2^2 + dx_3^2} &= \int_{t_a}^{t_b} \sqrt{x_{1,t}^2 + x_{2,t}^2 + x_{3,t}} \, dt \\ &= \int_{t_a}^{t_b} \|\mathbf{x}_{,t}\| \, dt\end{aligned}$$

This can be easily extended to the computation of integral quantities over model edges:

$$\int_{\mathcal{C}} f(x_1, x_2, x_3) dl = \int_{t_a}^{t_b} f(x_1(t), x_2(t), x_3(t)) \, \|\mathbf{x}_{,t}\| \, dt \tag{3.5}$$

The curvilinear abscissa $l(t)$ of a point $\mathbf{x}(t)$ of curve $\mathcal{C}$, is the length of the segment defined by parameter range $[t_1, t]$, i.e. the length of the curve from the origin $\mathbf{x}(t_1)$ to $\mathbf{x}(t)$:

$$l(t) = \int_{t_1}^{t} \|\mathbf{x}_{,t}\| \, dt \tag{3.6}$$

We have seen before that $dl = \|\mathbf{x}_{,t}\| \, dt$.

A parametrization of $\mathcal{C}$ is said to be regular if $\|\mathbf{x}_{,t}\| \neq 0$. For regular parametrizations, the unit tangent vector is defined as

$$\mathbf{t}(t) = \frac{\mathbf{x}_{,t}}{\|\mathbf{x}_{,t}\|} = \frac{d\mathbf{x}}{dl}.$$

The normal plane at point $\mathbf{x}(t)$ is the plane that contains $\mathbf{x}(t)$ and that has $\mathbf{t}(t)$ as normal vector (see Figure 3.7). The curvature of the curve at a point $\mathbf{x}$ can be defined as the amplitude of the variations of the unit tangent $\mathbf{t}$ along the curve. The vector $\mathbf{t}_{,l}$ is obviously orthogonal to $\mathbf{t}$ because $\mathbf{t}$'s amplitude is one along l. Recalling that

$$\frac{d}{dt}\frac{1}{\|\mathbf{x}\|} = -\frac{\mathbf{x}_{,t} \cdot \mathbf{x}}{\|\mathbf{x}\|^3}$$

we have

$$\begin{aligned}\mathbf{t}_{,l} &= \frac{1}{\|\mathbf{x}_{,t}\|}\mathbf{t}_{,t} \\ &= \frac{1}{\|\mathbf{x}_{,t}\|}\left(\frac{\mathbf{x}_{,tt}}{\|\mathbf{x}_{,t}\|} - \mathbf{x}_{,t}\frac{\mathbf{x}_{,t} \cdot \mathbf{x}_{,tt}}{\|\mathbf{x}_{,t}\|^3}\right) \\ &= \frac{1}{\|\mathbf{x}_{,t}\|^3}\left(\mathbf{x}_{,tt}\|\mathbf{x}_{,t}\| - \mathbf{x}_{,t}\frac{\mathbf{x}_{,t} \cdot \mathbf{x}_{,tt}}{\|\mathbf{x}_{,t}\|}\right).\end{aligned}$$

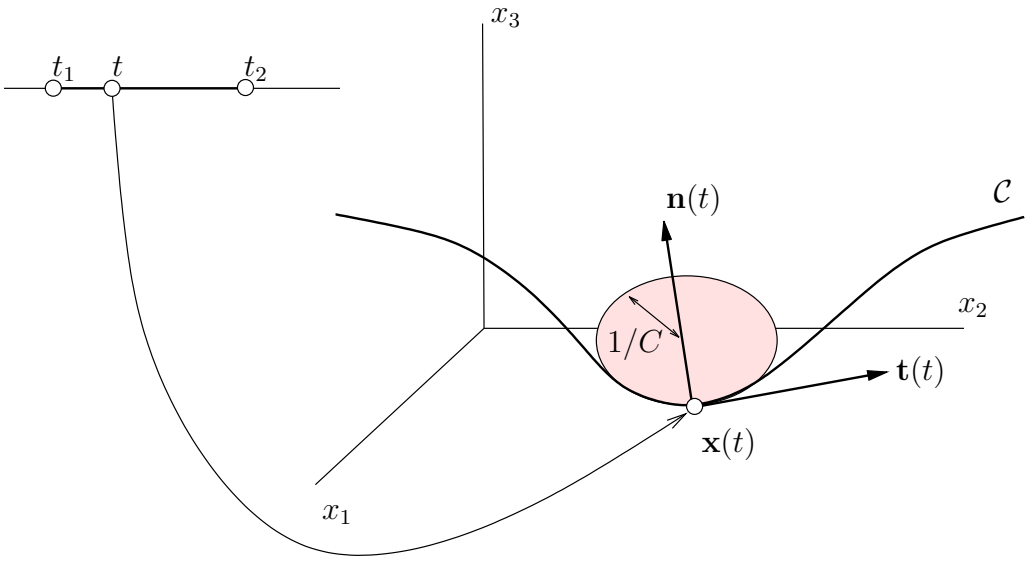


Figure 3.7: A curve $\mathcal{C}$.

Clearly, $\mathbf{t}_{,l} \cdot \mathbf{t} = 0$. Because we heve defined the curvature as the amplitude of the variations of the unit tangent $\mathbf{t}$ along the curve, we call rewrite

$$\mathbf{t}_{,l} = \|\mathbf{t}_{,l}\| \frac{\mathbf{t}_{,l}}{\|\mathbf{t}_{,l}\|} = C\mathbf{n}$$

with $\mathbf{n}$ a unit normal vector orthogonal to $\mathbf{t}$ and C the curvature that is the norm of $\mathbf{t}_l$. Remembering that

$$\|\mathbf{a}\times\mathbf{b}\|^2 = \|a\|^2\|b\|^2 \sin^2(\mathbf{a},\mathbf{b}) = \|\mathbf{a}\|^2\|\mathbf{b}\|^2 \left(1 - \frac{(\mathbf{a}\cdot\mathbf{b})^2}{\|\mathbf{a}\|^2\|\mathbf{b}\|^2}\right) = \|\mathbf{a}\|^2\|\mathbf{b}\|^2 - (\mathbf{a}\cdot\mathbf{b})^2,$$

it is easy to see that

$$C^2 = \frac{1}{\|\mathbf{x}_{,t}\|^6} \left(\|\mathbf{x}_{,t}\|^2\|\mathbf{x}_{,tt}\|^2 - (\mathbf{x}_{,tt}\cdot\mathbf{x}_{,t})^2\right) = \frac{1}{\|\mathbf{x}_{,t}\|^6}\|\mathbf{x}_t \times \mathbf{x}_{,tt}\|^2$$

and we get the classical formula

$$C = \frac{\|\mathbf{x}_t \times \mathbf{x}_{,tt}\|}{\|\mathbf{x}_{,t}\|^3}.$$

It is possible to define a local system of coordinates at any point $\mathbf{x}$ of the curve

$$(\mathbf{t}, \mathbf{n}, \mathbf{b})$$

with $\mathbf{b} = \mathbf{t} \times \mathbf{n}$. This system of coordinates is usually called the Frenet frame. The osculating plane of the curve at point $\mathbf{x}$ can be defined as the plane containing $\mathbf{x}$ and normal to $\mathbf{b}$. The curvature $C(t)$ is the inverse of the radius of the osculating circle at point $\mathbf{x}$ i.e. the circle which most closely approximates the curve near $\mathbf{x}$:

$$R(t) = \frac{1}{C(t)}.$$

This gives an interresting intuitive interpretation of the curvature.

Differential Geometry of Surfaces

A parameterization of a surface is a one-to-one mapping from a suitable domain to the surface. In CAD modelers, surfaces have explicit parametrizations i.e. their parametrization

$$(u, v) \in \mathcal{R}^2 \mapsto \mathbf{x}(u, v) \in \mathcal{R}^3.$$

is given explicitely as a continuous and differentiable function.

At any point $\mathbf{x}(u, v)$ of a surface $\mathcal{S}$, one can define two tangent vectors $\mathbf{x}_{,u}$ and $\mathbf{x}_{,v}$ and a normal vector $\mathbf{x}_{,u} \times \mathbf{x}_{,v}$ (see Figure 3.8).

Consider a curve that is included in surface $\mathcal{S}$. It is easy to extend the integration formula (3.5) as

$$\begin{aligned}
\int_{\mathcal{C}} \; & f(x_1, x_2, x_3) \quad \sqrt{dx_1^2 + dx_2^2 + dx_3^2} \\
&= \int_{\mathcal{C}} f(x_1, x_2, x_3) \sqrt{\|\mathbf{x}_{,u}\|^2 \, du^2 + 2 \; \mathbf{x}_{,u} \cdot \mathbf{x}_{,v} du \; dv + \|\mathbf{x}_{,v}\|^2 \, dv^2} \\
&= \int_{\mathcal{C}} f(x_1, x_2, x_3) \sqrt{\begin{bmatrix} du \\ dv \end{bmatrix}^T \begin{bmatrix} \mathbf{x}_{,u} \cdot \mathbf{x}_{,u} & \mathbf{x}_{,u} \cdot \mathbf{x}_{,v} \\ \mathbf{x}_{,v} \cdot \mathbf{x}_{,u} & \mathbf{x}_{,v} \cdot \mathbf{x}_{,v} \end{bmatrix} \begin{bmatrix} du \\ dv \end{bmatrix}} \\
&= \int_{\mathcal{C}} f(x, y, z) \sqrt{\mathbf{du}^T \mathbf{M} \; \mathbf{du}} \qquad (3.7)
\end{aligned}$$

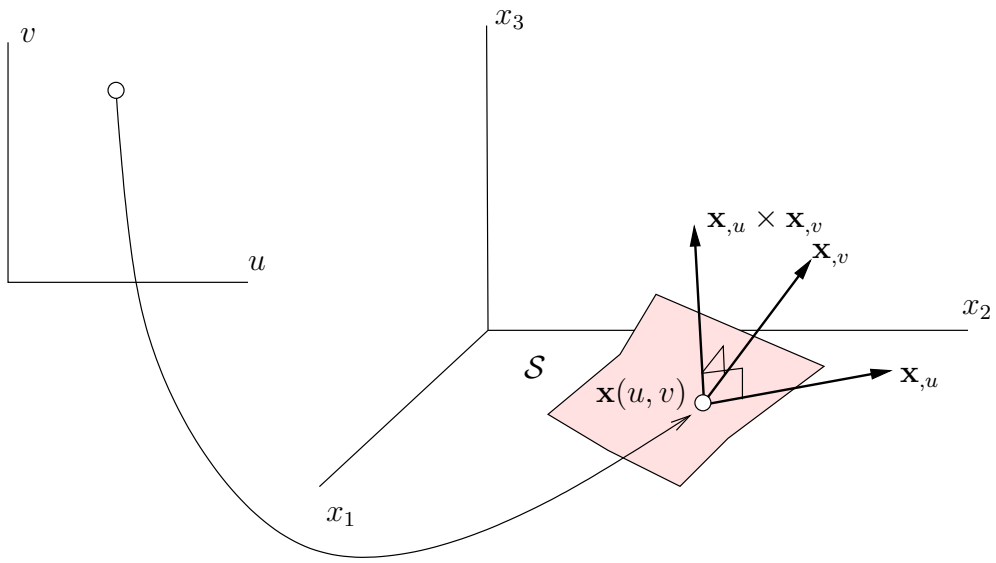


Figure 3.8: A surface $\mathcal{S}$.

In (3.7),

$$\mathbf{M} = \begin{bmatrix} \mathbf{x}_{,u} \cdot \mathbf{x}_{,u} & \mathbf{x}_{,u} \cdot \mathbf{x}_{,v} \\ \mathbf{x}_{,v} \cdot \mathbf{x}_{,u} & \mathbf{x}_{,v} \cdot \mathbf{x}_{,v} \end{bmatrix} = \begin{bmatrix} E & F \\ F & G \end{bmatrix}$$

is called the metric tensor defined on the surface. The metric tensor is defined, in general, on a d-dimensional manifold. It is a symmetric definite positive second order tensor that varies smoothly over the manifold.

We have just seen that the tensor metric enables to measure curve lengths drawn in the parametric plane. It also allows to generalizes many familiar properties of the dot product of vectors in Euclidean space. In particular, it allows to compute the angle between two tangent vectors to the surface. Any tangent vector at a point of the parametric surface can be written in the form

$$\mathbf{t} = a\mathbf{x}_{,u} + b\mathbf{x}_{,v}$$

with $a, b \in \mathcal{R}$. Let us consider two tangent vectors

$$\mathbf{t}_1 = a_1\mathbf{x}_{,u} + b_1\mathbf{x}_{,v} \quad \text{and} \quad \mathbf{t}_2 = a_2\mathbf{x}_{,u} + b_2\mathbf{x}_{,v}.$$

Coordinates $\mathbf{a} = (a_1, a_2)$ and $\mathbf{b} = (b_1, b_2)$ are called covariant coordinates of

$\mathbf{t}_1$ and $\mathbf{t}_2$. We have

$$\mathbf{t}_1 \cdot \mathbf{t}_2 = a_1 a_2 \mathbf{x}_{,u} \cdot \mathbf{x}_{,u} + (a_1 b_2 + a_2 b_1)\mathbf{x}_{,u} \cdot \mathbf{x}_{,v} + b_1 b_2 \mathbf{x}_{,v} \cdot \mathbf{x}_{,v} = \mathbf{a}^T \mathbf{M} \mathbf{b}.$$

Consider a small rectangle $du\ dv$ at a point on the parametric plane. Its area is

$$s = \|\mathbf{x}_{,u} du \times \mathbf{x}_{,v} dv\| = du\ dv \sqrt{\det \mathbf{M}}.$$

Note again that the value of the area only depends on the metric.

Lengths, angles and areas are fundamental quantities that are independant of the system of coordinates. The quadratic form $I(\mathbf{a}, \mathbf{b}) = \mathbf{a}^T \mathbf{M} \mathbf{b}$ is called the first fundamental form of the surface. It is the inner product on the tangent space of a surface in three-dimensional Euclidean space.

A mapping is isometric or length-preserving if the length of any arc is preserved. Such a mapping is called an isometry. For example, the mapping of a cylinder into the plane that transforms cylindrical coordinates into cartesian coordinates is isometric.

A mapping is conformal or angle-preserving if the angle of intersection of every pair of intersecting is the same on the parametric plane and on the surface. For example, the stereographic and Mercator projections are conformal maps from the sphere to the plane

A mapping is equiareal if surfaces are conserved by the mapping. For example, the Lambert projection is an equiareal mapping from the sphere to the plane.

Every isometric mapping is conformal and equiareal, and every conformal and equiareal mapping is isometric, i.e.,

$$\text{isometric} \iff \text{conformal} + \text{equiareal}.$$

In terms of the mertic tensor, we have

1. isometric mappings: $\mathbf{M} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$,
2. conformal mappings: $\mathbf{M} = \begin{bmatrix} \eta & 0 \\ 0 & \eta \end{bmatrix}$,
3. equiareal mappings: $\det \mathbf{M} = 1$.

We can thus view an isometric mapping as ideal, in the sense that it preserves just about everything we could ask for: angles, areas, and lengths.

However, as is well known, isometric mappings only exist in very special cases. When mapping into the plane, the surface $\mathcal{S}$would have to be developable, such as a cylinder. Many approaches to surface parameterization therefore attempt to find a mapping which either

1. is conformal, i.e., has no distortion in angles, or
2. is equiareal, i.e., has no distortion in areas, or
3. minimizes some combination of angle distortion and area distortion.

The surface unit normal $\mathbf{n}$ is orthogonal to both tangent vectors:

$$\mathbf{n} = \frac{\mathbf{x}_{,u} \times \mathbf{x}_{,v}}{\|\mathbf{x}_{,u} \times \mathbf{x}_{,v}\|} = \frac{\mathbf{x}_{,u} \times \mathbf{x}_{,v}}{\sqrt{\det \mathbf{M}}} = \frac{\mathbf{x}_{,u} \times \mathbf{x}_{,v}}{\sqrt{EG - F^2}}. \tag{3.8}$$

Vectors

$$(\mathbf{x}_{,u}, \mathbf{x}_{,v}, \mathbf{n})$$

form at each point of the surface a local system of coordinates usually called the *local frame*. It is easy to orthonormalize the local frame, i.e. by choosing

$$\mathbf{t}_1 = \frac{\mathbf{x}_{,u}}{\|\mathbf{x}_{,u}\|}, \quad \mathbf{t}_2 = \mathbf{n} \times \mathbf{t}_1$$

so that vectors

$$(\mathbf{t}_1, \mathbf{t}_2, \mathbf{n})$$

form an orthonormal system of coordinates usually called the *Darboux frame*.

Another fundamental quantity related to the shape of surfaces is curvature. To study the curvature of the surface at a point $\mathbf{x}$, one can examine the variations of the unit normal $\mathbf{n}$ around $\mathbf{x}$. In particular, one can derivate $\mathbf{n}$ in the direction specified by the tangent vectors at $\mathbf{x}$: this is called the Weingarten map. Note that $\mathbf{n}_{,u}$ and $\mathbf{n}_{,v}$ are both tangent vectors because $\mathbf{n}$ is a unit vector:

$$\frac{\partial}{\partial u}\frac{\mathbf{v}}{\|\mathbf{v}\|} = \frac{\mathbf{v}_{,u}}{\|\mathbf{v}\|} - \frac{\mathbf{v}}{\|\mathbf{v}\|^3}(\mathbf{v} \cdot \mathbf{v}_{,u})$$

Applying that formula to Equation (3.8) (an after very tedious calculations), we obtain the Weingarten equations that express the derivatives of the normal to a surface using derivatives of the position vector $\mathbf{x}$

$$\mathbf{n}_{,u} = \frac{fF - eG}{EG - F^2}\mathbf{x}_{,u} + \frac{eF - fE}{EG - F^2}\mathbf{x}_{,v}$$

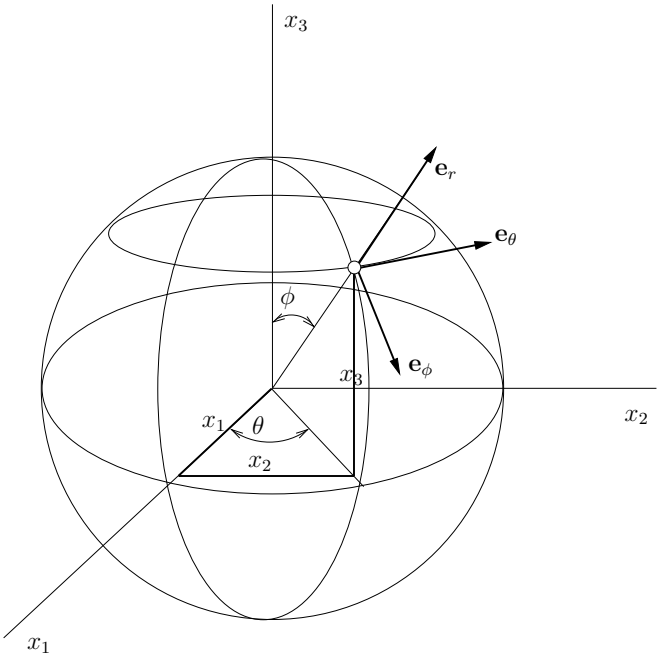


Figure 3.9: Spherical coordinates.

$$\mathbf{n}_{,v} = \frac{gF - fG}{EG - F^2}\mathbf{x}_{,u} + \frac{fF - gE}{EG - F^2}\mathbf{x}_{,v}$$

where

$$e = \mathbf{n} \cdot \mathbf{x}_{,uu}, \quad f = \mathbf{n} \cdot \mathbf{x}_{,uv} \quad \text{and} \quad g = \mathbf{n} \cdot \mathbf{x}_{,vv}.$$

Tensor

$$\mathbf{M}_2 = \begin{bmatrix} e & f \\ f & g \end{bmatrix}$$

is called the second fundamental tensor of the surface.

Parametrizations of the sphere

Let us consider, as an example, the parametrization of a sphere of radius R using spherical coordinates (Figure 3.9):

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} R\cos\theta\sin\phi \\ R\sin\theta\sin\phi \\ R\cos\phi \end{bmatrix}.$$

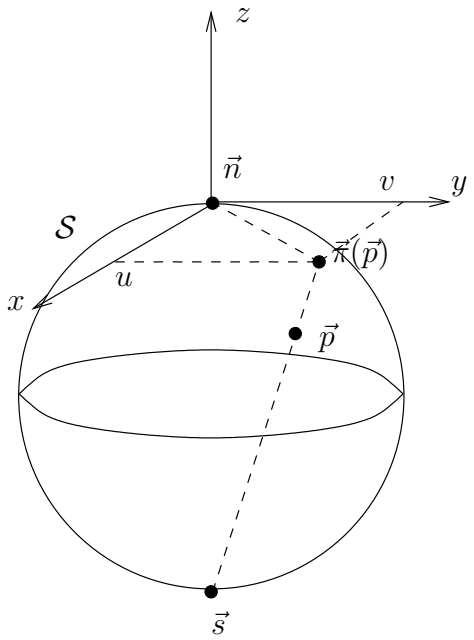


Figure 3.10: Stereographic projection.

We have

$$\mathbf{x}_{,\phi} = \begin{bmatrix} R\cos\theta\cos\phi \\ R\sin\theta\cos\phi \\ -R\sin\phi \end{bmatrix}, \quad \mathbf{x}_{,\theta} = \begin{bmatrix} -R\sin\theta\sin\phi \\ R\cos\theta\sin\phi \\ 0 \end{bmatrix}$$

and the metric tensor relative to that parametrization is

$$\mathbf{M} = R^2 \begin{bmatrix} 1 & 0 \\ 0 & \sin^2\phi \end{bmatrix}. \tag{3.9}$$

This parametrization is neither equiareal, neither conformal.

Let us consider the same sphere $\mathcal{S}$ centered at the origin and of radius R, and one point s. This point lies on the surface and will be the only singular point of the mapping.

Here, we choose $s = \{0, 0, -R\}$. It corresponds to the "South Pole" ot the sphere. The stereographic projection consists in projecting points $\vec{p}$ of the sphere on the plane $z = R$.

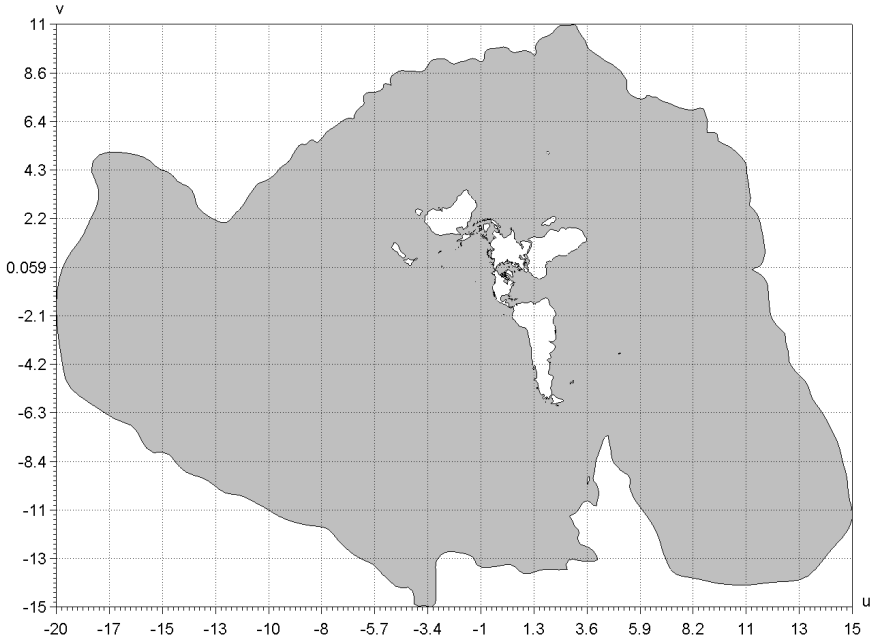


Figure 3.11: The World Ocean in stereographic coordinates.

The stereographic projection $\vec{u}(\vec{x}) = \{u, v\}$ of a point $\vec{x} = \{x, y, z\}$ is the intersection of vector $\vec{q} - \vec{p}$ with $z = R$:

$$\begin{aligned} \vec{u} &= \{u, v\} = \left\{ \frac{2R}{R+z} x, \frac{2R}{R+z} y \right\}, \\ \vec{x} &= \{x, y, z\} = \frac{4R^2}{u^2 + v^2 + 4R^2} \left\{ u, v, R(4R^2 - u^2 + v^2) \right\}. \end{aligned}$$

Figure 3.11 shows the World Ocean in stereographic coordinates $\{u, v\}$. The outside loop surrounding the domain is the stereographic projection of the Antarctica. The radius of the Earth is chosen arbitrarily to $R = 1$. No seam is required to define the overall domain and no singular point exists in the domain of interest.

The metric tensor $\mathbf{M}$ for the stereographic projection is

$$\mathbf{M} = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}. \tag{3.10}$$

with

$$\lambda(u,v) = \left(\frac{4R^2}{u^2+v^2+4R^2}\right).$$

Those two eigenvalues are equal: the stereographic mapping is therefore conformal. Remind that a conformal mapping will conserve angle at which curves cross each other.

3.2 Discrete Representation of the Geometry

In various applications, the only available representation of a surface $\mathcal{S}$ is a conforming triangular mesh, i.e. the union of a set

$$\mathsf{M} = \{M_1^2, ..., M_N^2\}$$

such that the triangles intersect only at common vertices or edges (See Figure 3.12). If M has a boundary, then the boundary will be polygonal and we denote it by $\partial\mathsf{M}$.

The output of a segmentation tool that extracts geometrical data from medical imaging is typically a triangulation. In computer graphics, most of the surfaces that are used in computer games e.g. are triangulations. Even in engineering, it is very common to use stereolitography (STL) triangulations as the geometrical input for analysis.

There are techniques that allow to reparametrize a discrete surfaces. In this text, we present one of these tachnique that is actually implemented in gmsh.

3.2.1 Harmonic maps

A very common way of parametrizing a triangulation is to use harmonic maps. Conformal mappings have many nice properties, not least of which is their connection to complex function theory. Consider for the moment the case of mappings from a planar region $\mathcal{S}$ to the plane. Such a mapping can be viewed as a function of a complex variable, $\omega = f(z)$. Locally, a conformal map is simply any function f which is analytic in a neighbourhood of a point z and such that $f'(z) = 0$. A conformal mapping f thus satisfies the Cauchy-Riemann equations, which, with $z = x + iy$ and $\omega = u + iv$, are

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y}, \quad \frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x}.$$

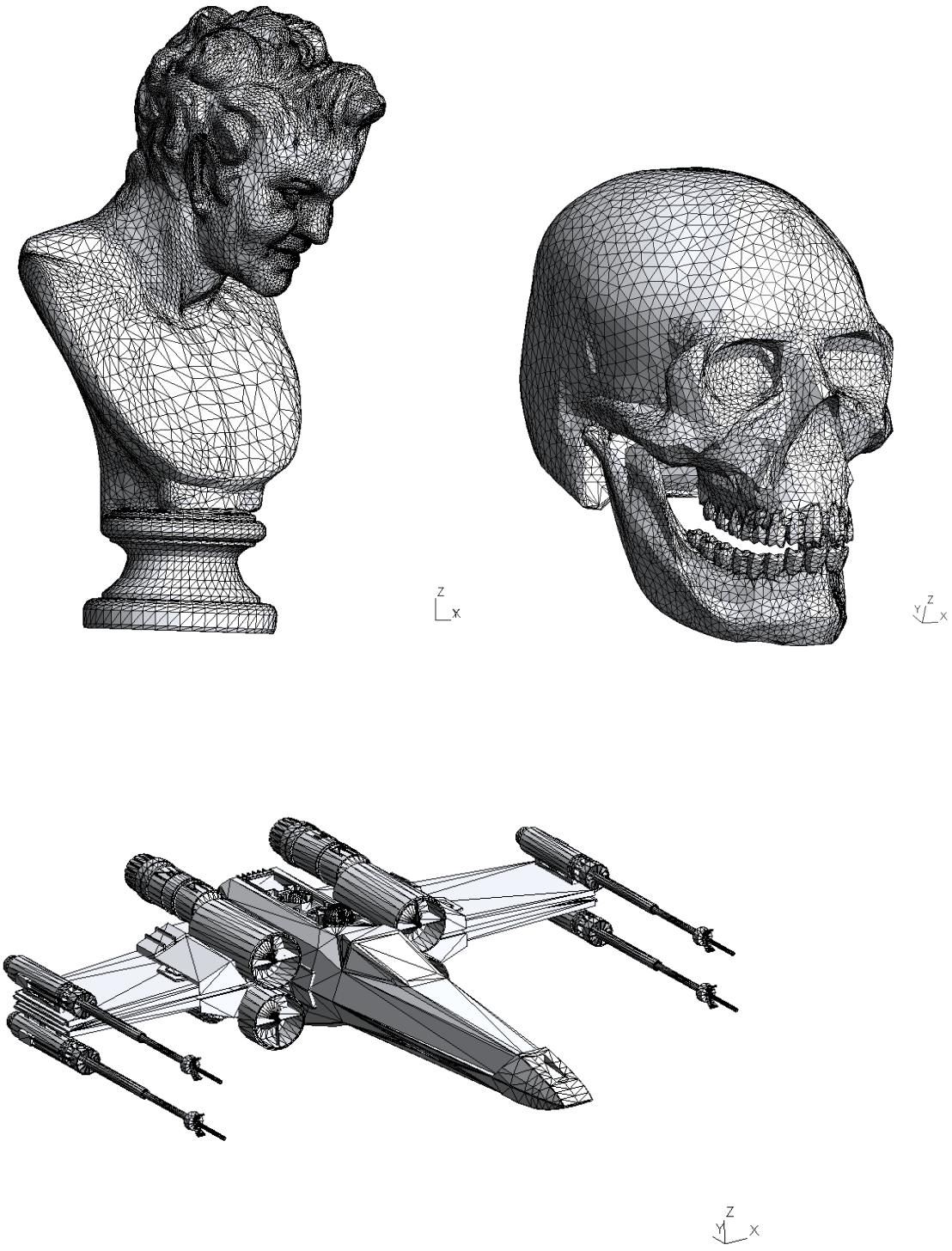


Figure 3.12: Examples of geometries that are defined as triangulations

Now notice that by differentiating one of these equations with respect to x and the other with respect to y, we obtain the two Laplace equations

$$\nabla^2 u = 0, \quad \nabla^2 v = 0.$$

Any mapping $(u(x,y), v(x,y))$ which satisfiies these two Laplace equations is called a harmonic mapping. Thus a conformal mapping is also harmonic, and we have the implications

$$\text{isometric} \Longrightarrow \text{conformal} \Longrightarrow \text{harmonic}.$$

The advantage over conformal maps is the ease with which they can be computed, at least approximately. After choosing suitable Dirichlet boundary conditions, each of the functions u and v is the solution to a linear elliptic partial differential equation which can be approximated by finite elements. Harmonic maps are also guaranteed to be one-to-one for convex regions. On the downside, harmonic maps are not in general conformal and do not preserve angles. Another weakness of harmonic mappings is their "one-sidedness". The inverse of a harmonic mapping is not necessarily harmonic.

Harmonic mappings can be computed to parametrize non planar surfaces, without increasing the complexity of their computation. Let us explain in details how to reparametrize a triangulated surface using finite elements. We indeed need to compute the two fields of coordinates $u(\mathbf{x})$ and $v(\mathbf{x})$. We'll only detail the way we compute u, the computation of v being of course absolutely similar.

Consider the following boundary value problem

$$\nabla^2 u = 0 \quad \text{on} \quad \Omega, \tag{3.11}$$

$$u = f(\mathbf{x}) \quad \text{on} \quad \partial\Omega. \tag{3.12}$$

It is easy to prove that (3.12) and (3.12) and is equivalent to the following quadratic minimization problem:

$$\min_{u \in U(\mathcal{S})} J(u) = \frac{1}{2} \int_{\mathcal{S}} \left| \nabla^2 u \right| ds \tag{3.13}$$

with

$$U(\mathcal{S}) = \{u \in H^1(\mathcal{S}),\ u = f(\mathbf{x}) \ \text{ on } \ \partial\mathcal{S}.\}$$

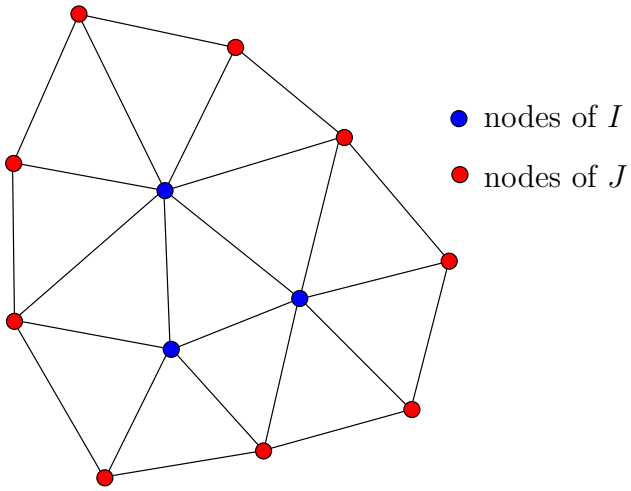


Figure 3.13: Sets of nodes.

Assume the following finite expansions for u

$$u_h(\mathbf{x}) = \sum_{i \in I} u_i \phi_i(\mathbf{x}) + \sum_{i \in J} f(\mathbf{x}_i)\phi_i(\mathbf{x}) \tag{3.14}$$

where I denotes the set of nodes of M that do not belong to the Dirichlet boundary, J denotes the set of nodes of M that belong to the Dirichlet boundary (Figure 3.13) and where ϕ_i are the nodal shape functions associated to the nodes of the mesh. We assume here that nodal shape function ϕ_i is equal to 1 on vertex $\mathbf{x}_i$ and 0 on any other vertex: $\phi_i(\mathbf{x}_j) = \delta_{ij}$.

Thanks to expansion (3.14), functional J of (3.13) can be written as

$$\begin{aligned} J(u_1, \ldots, u_N) \;=\; & \frac{1}{2} \sum_{i \in I} \sum_{j \in I} u_i u_j \int_{\mathsf{M}} \nabla \phi_i(\mathbf{x}) \cdot \nabla \phi_j(\mathbf{x}) ds + \\ & \sum_{i \in I} \sum_{j \in J} u_i f(\mathbf{x}_j) \int_{\mathsf{M}} \nabla \phi_i(\mathbf{x}) \cdot \nabla \phi_j(\mathbf{x}) ds + \\ & \frac{1}{2} \sum_{i \in J} \sum_{j \in J} f(\mathbf{x}_i) f(\mathbf{x}_j) \int_{\mathsf{M}} \nabla \phi_i(\mathbf{x}) \cdot \nabla \phi_j(\mathbf{x}) ds. \end{aligned}$$

In order to minimize J, we can simply cancel the derivative of J with

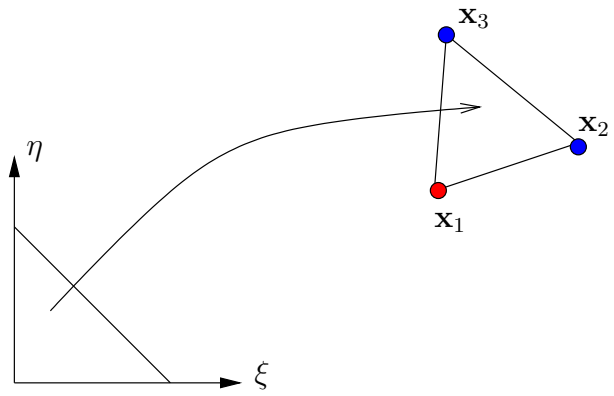


Figure 3.14: Unit triangle and its mapping.

respect to u_k

$$\begin{aligned}\frac{\partial J}{\partial u_k} &= \sum_{j\in I} u_j \int_{\mathsf{M}} \nabla\phi_j(\mathbf{x})\cdot\nabla\phi_k(\mathbf{x})ds + \\ &\quad \sum_{j\in J} f(\mathbf{x}_j)\int_{\mathsf{M}} \nabla\phi_k(\mathbf{x})\cdot\nabla\phi_j(\mathbf{x})ds \\ &= 0\ , \quad \forall k\in I. \end{aligned} \tag{3.15}$$

There are as many equations (3.15) as there are nodes in I. This system of equations can be proven to be symmetric positive definite so that it can be solved easily, e.g. using preconditioned conjugate gradients.

Consider one triangle M_j^2 with three vertices of coordinates $\mathbf{x}_i = \{x_i, y_i, z_i\}$, $i = 1, 2, 3$ (Figure 3.14). The triangle can itself be parametrized i.e. the unit triangle

$$\xi \in [0,1] \quad , \quad \eta \in [0, 1-\xi]$$

can be mapped to the 3D triangle using linear finite element shape functions

$$\mathbf{x} = \underbrace{(1-\xi-\eta)}_{\phi_1}\mathbf{x}_1 + \underbrace{\xi}_{\phi_2}\mathbf{x}_2 + \underbrace{\eta}_{\phi_3}\mathbf{x}_3.$$

It is easy to compute the metric of that mapping, as we've done it in (3.7). We have

$$\mathbf{M} = \begin{bmatrix} \mathbf{x}_{,\xi} \cdot \mathbf{x}_{,\xi} & \mathbf{x}_{,\xi} \cdot \mathbf{x}_{,\eta} \\ \mathbf{x}_{,\eta} \cdot \mathbf{x}_{,\xi} & \mathbf{x}_{,\eta} \cdot \mathbf{x}_{,\eta} \end{bmatrix} = \begin{bmatrix} \|\mathbf{x}_2 - \mathbf{x}_1\|^2 & (\mathbf{x}_2 - \mathbf{x}_1) \cdot (\mathbf{x}_3 - \mathbf{x}_1) \\ (\mathbf{x}_2 - \mathbf{x}_1) \cdot (\mathbf{x}_3 - \mathbf{x}_1) & \|\mathbf{x}_3 - \mathbf{x}_1\|^2 \end{bmatrix}$$

It is easy to see then that

$$\int_{M_j^2} \nabla\phi_i \cdot \nabla\phi_j ds = \int_0^1 \int_0^{1-\xi} \nabla^{\xi,\eta}\phi_i \ \mathbf{M}^{-1} \ \nabla^{\xi,\eta}\phi_j \sqrt{\det \mathbf{M}} \ d\xi d\eta.$$

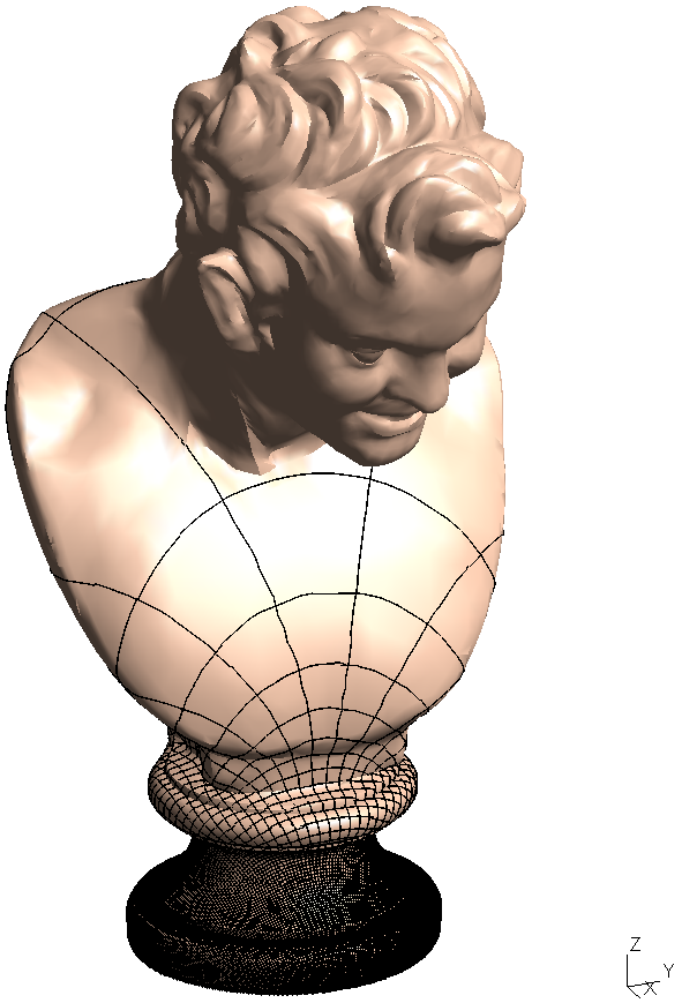


Figure 3.15: Reparametrization of the satyre statue

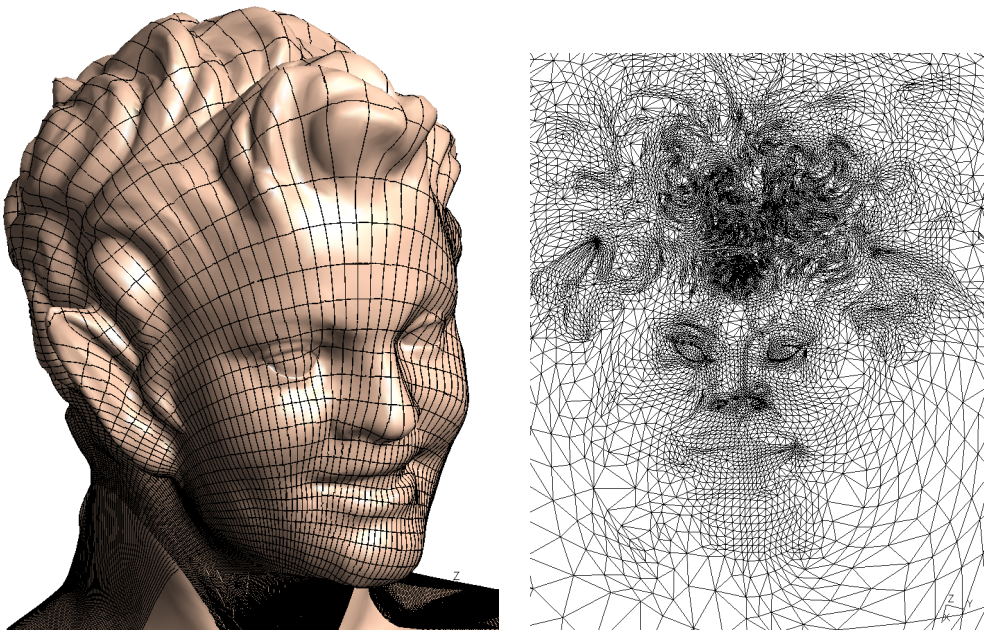


Figure 3.16: Reparametrization of the satyre statue

Chapter 4

Mesh Generation

4.1 Generalities

The goal of an analysis is to solve a set of partial differential equations over a geometrical domain G. The most common way to describe G is to use a boundary-based scheme where the geometric domain is represented as a set of topological types together with adjacencies. This has already be decribed in the sections of Chapter 3.

The mesh M is a discrete version of the domain. In some sense, it is similar to what we've defined for the geometric model: it consists of

- A collection of mesh entities M_i^d of controlled size and distribution;
- Topological relationships or adjacencies forming the graph of the mesh.

We have

1. Mesh Vertices M_i^0 that are topological entities of dimension 0,
2. Mesh Edges M_i^1 that are topological entities of dimension 1,
3. Mesh Faces M_i^2 that are topological entities of dimension 2,
4. Mesh Regions M_i^3 that are topological entities of dimension 3.

What differs is that mesh entities are more numerous than model entities but have limited complexity.

Mesh entities are topologically equivalent to the unit d-dimensional sphere $S^d = \{x \in \mathcal{R}^d; \|x\|_2 < 1\}$: they are made of one part, they are simply

connected (a manifold is said to be simply connected if every closed curve can be smoothly shrunk to a point) and they have no holes.

Mesh entities have simple shapes: they are are lines in 1D, triangles and quadrangles in 2D and tetrahedra, hexaedra and prisms in 3D.

Any mesh entity M_i^d is a piece of the discretization of a geometric entity G_j^q, $d \leq q$ (a mesh entity must have dimensionality less than or equal to the geometry it is associated with). We call this association a *classification* of a mesh entity to a geometrical entity and we note it as $M_i^d \sqsubset G_j^q$ [**?**, **?**, **?**].

Simulation attributes like boundary conditions or material properties are naturally related to model entities and not to mesh entities. In mesh generation and mesh enrichment procedures, the classification information is critical for ensuring that the mesh is constructed so that it improves the geometric approximation of the domain when it is is refined. It is therefore necessary to maintain the classification of mesh entities through all our algorithms.

A mesh M is composed of a collection of mesh entities together with their adjacencies. Any mesh entity bounds and/or is bounded by other ones of higher and/or lower dimension. This adjacency information represents the graph of a mesh.

It is interesting at this point to gather some statistics about the average number of adjacencies per entity that occurs in usual three dimensional tetrahedral and hexahedral meshes.

4.1.1 The Euler-Poincaré Formula

The Euler-Poincaré formula describes the relationship of the number of vertices, the number of edges and the number of faces of the *cellular decomposition* (a mesh) of a manifold. It has been generalized to include potholes and holes that penetrate the solid. To state the Euler-Poincaré formula, we need the following definitions:

- $\#V$ is the number of vertices in the cellular decomposition,
- $\#E$ is the number of edges in the cellular decomposition,
- $\#F$ is the number of faces in the cellular decomposition.
- G is the number of holes that penetrate the solid, usually referred to as *genus* in topology

- S is the number of shells. A shell is an internal void of a solid. A shell is bounded by a 2-manifold surface, which can have its own genus value. Note that the solid itself is counted as a shell. Therefore, the value for $\#S$ is at least 1.

- L is the number of loops. All outer and inner loops of faces are counted. The Euler-Poincaré formula is

$$\#V - \#E + \#F - (L - \#F) - 2(S - G) = 0.$$

Consider a cube. It has eight vertices ($\#V = 8$), 12 edges ($\#E = 12$) and six faces ($\#F = 6$), no holes and one shell ($S = 1$); but $\#L = \#F$ since each face has only one outer loop. Therefore, we have

$$\#V-\#E+\#F-(L-\#F)-2(S-G) = 8-12+6-(6-6)-2(1-0) = 0.$$

Consider now M, a 2D mesh of a domain Ω. The Euler-Poincaré relation gives the following relation between those quantities:

$$\#V - \#E + \#F - \chi(\Omega) = 0$$

where $\chi(\Omega)$ is the Euler-Poincaré characteristic of the surface. The Euler-Poincaré characteristic of different surfaces is given in the following bullet list:

  - for the sphere, $\chi = 2$ ($\#V - \#E + \#F = 2 - 4 + 4$),
  - for the torus, $\chi = 0$ ($\#V - \#E + \#F = 4 - 8 + 4$),
  - for the disk, $\chi = 1$,
  - for the Klein bootle, $\chi = 1$.

Another form of the relation, more useful for general domains:

$$\chi = \#V - \#E + \#F = 2 - 2g + b$$

where

  - b is the number of boundaries (1 for the plane or 0 for a torus or a sphere),

– g is the *genus* of the surface. The genus is the largest number of nonintersecting simple closed curves that can be drawn on the surface without separating it. Roughly speaking, it is the number of holes in a surface.

Property 4.1.1 *We consider a mesh of a 2D domain that is isomorph to a disk. Then, the following relations holds*

$$\#F - 2(\#V - 1) + \#V_b = 0 \quad (4.1)$$
$$\#E - 3(\#V - 1) + \#V_b = 0 \quad (4.2)$$

where $\#V_b$ is the number of vertices on b.

This is an important relation that gives a relation between the number of triangles and the number of vertices in the triangular mesh of a "disk-like" surface. In order to proove this, let us first remark that relations (4.1) and (4.2) are true for one triangle alone: $\#F = 1$, $\#V = \#V_b = 3$.

Swapping an edge of the mesh does not modify $\#V$, $\#E$, $\#F$ or $\#V_b$. All triangulations with $\#N$ given are therefore equivalent: edge swaps allow to transform any given triangulation to any other.

Inserting a point inside a triangle adds one vertex, 2 triangles and 3 edges to the mesh, leaving $\#V_b$ unchanged:

$$(\#F + 2) - 2((\#V + 1) - 1) + \#V_b = \#F - 2(\#V - 1) + \#V_b = 0$$
$$(\#E + 3) - 3((\#V + 1) - 1) + \#V_b = \#E - 3(\#V - 1) + \#V_b = 0.$$

Inserting a point on the domain boundary b adds one vertex, one triangles and two edges to the mesh. Relations (4.1) and (4.2) are still true:

$$(\#F + 1) - 2((\#V + 1) - 1) + (\#V_b + 1) = \#F - 2(\#V - 1) + \#V_b = 0$$
$$(\#E + 2) - 3((\#V + 1) - 1) + (\#V_b + 1) = \#E - 3(\#V - 1) + \#V_b = 0.$$

Property 4.1.2 *We consider a mesh of a 3D domain that is isomorphic to a sphere. The following relation holds*

$$\#E - \#R = \#V + \#V_b - 3$$

where $\#V_b$ is the number of vertices on the boundary.

Note that this relation is true for one tatrehadron only, as well as for any other 3D element.

In a 3D tetrahedral mesh, there exist no relation between the number of tetrahedra and the number of nodes, as it exists in 2D. As an example, it is possible to define the following "face swap" transformation: 2 tets that have a face in common can be transformed in 3 tets without changing the number of vertices. Yet one additional edge has to be added to the mesh in order to verify the relation, verifying the Euler-Poincaré formula.

Asymptotically, the 3D Euler-Poincaré gives:

$$\#V - \#E + \#F - \#R \simeq 0.$$

In the following Tables 4.1 and 4.2, we present some statistics about 3-D meshes. Those are "asymptotically" correct for sufficently large meshes but are patently wrong for coarse meshes. The average number of mesh entities in tetrahedral and hexahedral meshes are presented in Table 4.1.

Tetrahedral Mesh M	Hexahedral Mesh M
$\#R = 6\#V$	$\#R = \#V$
$\#F = 12\#V$	$\#F = 3\#V$
$\#E = 7\#V$	$\#E = 3\#V$

Table 4.1: Relation between number of entities in a mesh.

A second interesting set of statistics concerns the average number of mesh entities of dimension d adjacent to a mesh entity of dimension q. We call this $N^d(M^q)$. These statistics are represented in Table 4.2.

Tetrahedral Mesh M					Hexahedral Mesh M				
d	3	2	1	0	d	3	2	1	0
$N^3(M^d)$	1	2	5	23	$N^3(M^d)$	1	2	4	8
$N^2(M^d)$	4	1	5	35	$N^2(M^d)$	6	1	4	12
$N^1(M^d)$	6	3	1	14	$N^1(M^d)$	12	4	1	6
$N^0(M^d)$	4	3	2	1	$N^0(M^d)$	8	4	2	1

Table 4.2: Average number of adjacencies per entity

We read these tables as follow: most of the tetrahedron meshes will contain, when they are sufficiently big, 6 times more tetrahedron than vertices. Every edge is connected, on average, to 5 tetrahedron.

The information contained in these two tables are important when designing mesh data structures or when choosing one finite element interpolation scheme. In a tetrahedral mesh for example, it is important to figure out that there are 12 times more faces than vertices and that any scheme that imposes to store in memory the faces of the mesh will be expensive. Moreover, building finite element interpolations on tetrahedral meshes based on any other entity than vertices will generate large number of degrees of freedom.

4.1.2 Mesh generation procedure

Usual mesh generation procedures work as follows

- Curves are discretized first i.e. are subdivided into line elements,
- Then, surfaces are triangulated using the discretizations of the curves as boundaries,
- Finally, regions are tetrahedralized using surface meshes.

Some authors have proposed an alternative procedure that consist in building the 3D mesh at first [**?**, **?**]. This kind or approach is not used in Gmsh and will not be discussed here. Figure 4.1 illustrate this three steps procedure.

At the end, some pre- and post-processing steps can be plugged in the general mesh generation "three steps" flow.

- High order curvilinear meshed are usually build starting from a valid 3D straight sided mesh.
- Quadrilateral meshes generation procedures usually use a valid triangulation as their starting point.
- Boundary layer meshes are often build using an extrusion of the triangulation.

All the stages of those mesh generation procedures are presented in the next sections of the chapter.

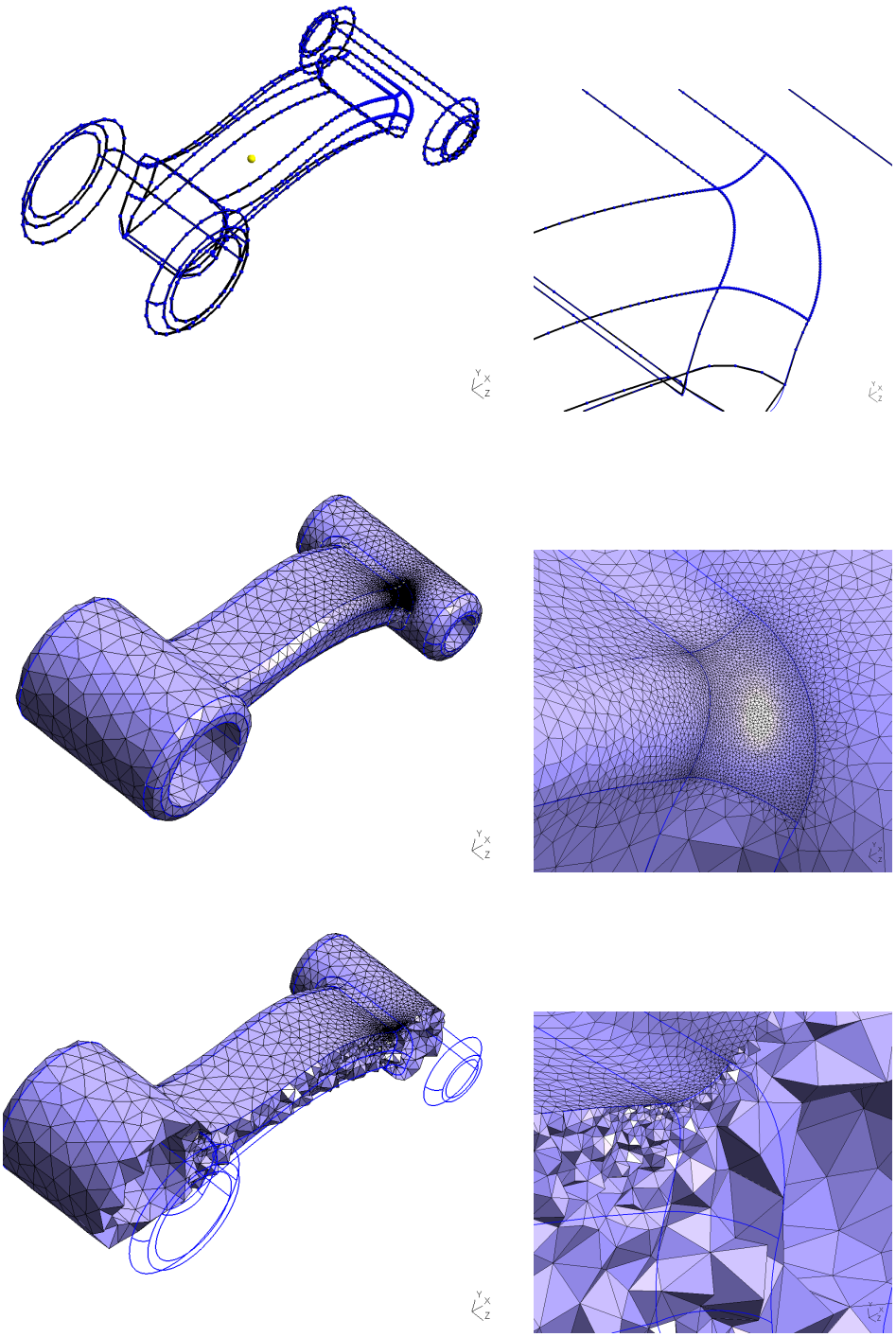


Figure 4.1: Example of a 3D meshing procedure in Gmsh. From top, 1D meshing procedure, surface meshing and volume meshing.

4.2 Mesh size field and quality measures

The aim of mesh generation is twofold

1. Generating elements of the right size,
2. Generating elements of the right shape.

For adressing those aims, we have to clarify what is right for an element, both in terms of size and shape.

4.2.1 Mesh size field

We define the mesh size function $\delta(x, y, z)$ as a function the defines at every point of the domain a target size for the elements at the point. The aim of the mesh generation procedure is to be able to build a mesh that complies with this mesh size field.

The present ways of defining such a mesh size field in Gmsh are:

1. mesh sizes prescribed at model vertices and interpolated linearly on model edges;
2. prescribed mesh gradings on model edges (geometrical progressions, ...);
3. mesh sizes defined on another mesh (a background mesh) of the domain;
4. mesh sizes that adapt to the principal curvature of model entities.

These size fields can then be acted on by functionals that may depend, for example, on the distance to model entities or on user-prescribed analytical functions; and when several size fields are provided, Gmsh uses the minimum of all fields. Thanks to that mechanism, Gmsh allows for a mesh size field defined on a given model entity to extend in higher dimensional entities. For example, using a distance function, a refinement based on the curvature of a model edge can extend on any surface adjacent to it.

Let us now consider an edge e of the mesh. We define the adimensional length of the edge with respect to the size field δ as

$$l_e = \int_e \frac{1}{\delta(x,y,z)} dl. \tag{4.5}$$

The aim of the mesh generation process is twofold:

1. Generate a mesh for which each mesh edge e is of size close to $l_e = 1$,
2. Generate a mesh for which each element K is *well shaped*.

In other words, the aim of the mesh generation procedure is to be able to build a good quality mesh that complies with the mesh size field.

To quickly evaluate the adequation between the mesh and the prescribed mesh size field, we defined an efficiency index τ [**?**] as

$$\tau = \exp\left(\frac{1}{ne}\sum_{e=1}^{ne}\tau_e\right) \tag{4.6}$$

with $\tau_e = l_e - 1$ if $l_e < 1$ and $\tau_e = \dfrac{1}{l_e} - 1$ if $l_e \geq 1$. The efficiency index ranges in $\tau \in [0,1]$ and should be as close as possible to $\tau = 1$.

For measuring the quality of elements, various element shape measures are available in the literature [**?**, **?**]. Here, we choose a measure based on the element radii ratio, i.e. the ratio between the inscribed and the circumcircles.

If K is a triangle, we have the following formula

$$\gamma_K = 4\frac{\sin\hat{a}\sin\hat{b}\sin\hat{c}}{\sin\hat{a}+\sin\hat{b}+\sin\hat{c}},$$

$\hat{a}$, $\hat{b}$ and $\hat{c}$ being the three inner angles of the triangle. With this definition, the equilateral triangle has a $\gamma_K = 1$ and degenerated (zero surface) triangles have a $\gamma_K = 0$.

For a tetrahedron, we have the following formula:

$$\gamma_K = \frac{6\sqrt{6}\,V_k}{\left(\sum_{i=1}^{4} a(f_i)\right)\max\limits_{i=1,\ldots,6} l(e_i)},$$

with V_K the volume of K, $a(f_i)$ the area of the i^{th} face of K and $l(e_i)$ the dimensional length of the i^{th} edge of K. This quality measurement lies in the interval $[0, 1]$, an element with $\gamma_K = 0$ being a sliver (zero volume).

4.3 One Dimensional Meshing

Let us consider a point $\mathbf{p}(t)$ on a curve $\mathcal{C}$, $t \in [t_1, t_2]$. The number of subdivisions N of the curve is its adimensional length:

$$\int_{t_1}^{t_2} \frac{1}{\delta(x, y, z)} \|\partial_t \vec{p}(t)\| dt = N. \tag{4.7}$$

The $N+1$ mesh points on the curve are located at coordinates $\{T_0, \ldots, T_N\}$, where T_i is computed with the following rule:

$$\int_{t_1}^{T_i} \frac{1}{\delta(x, y, z)} \|\partial_t \vec{p}(t)\| dt = i. \tag{4.8}$$

With this choice, each subdivision of the curve is exactly of adimensional size 1, and the 1-D mesh exactly satisfies the size field δ. In Gmsh, (4.8) is evaluated with a recursive numerical integration rule.

4.4 Planar Meshing

4.5 Surface Meshing

4.6 Quadrilateral Meshing

4.7 Tetrahedral Meshing

4.8 Hexaedral Meshing

4.9 Mesh Data Structures

4.10 Structured and Hybrid Mesh Generation

4.10.1 Hyperbolic mesh generation

4.10.2 Elliptic mesh generation

4.10.3 Boundary Layer Meshing

4.11 Mesh Adaptation

4.11.1 Local Mesh Modifications

4.11.2 Interactions with a solver

4.12 High Order Mesh Generation

Bibliography

[1] Debian. Debian linux. `http://www.debian.org`.

[2] G. Dhondt and K. Wittig. Calculix: A free software three-dimensional structural finite element program, 1998. `http://www.calculix.de`.

[3] J. Dongarra. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software (TOMS)*, 16(1):1–17, 1990.

[4] P. Dular and C. Geuzaine. GetDP: a general environment for the treatment of discrete problems, 1997. `http://www.geuz.org/getdp/`.

[5] M. Galassi, J. Davies, J. Theiler, B. Gough, and G. Jungman. Gnu scientific library, 2001. `http://www.gnu.org/software/gsl/`.

[6] P.-L. George and P. Frey. *Mesh Generation*. Hermes, 2000.

[7] C. Geuzaine. GL2PS: an OpenGL to PostScript printing library, 2000. `http://www.geuz.org/gl2ps/`.

[8] C. Geuzaine and J.-F. Remacle. Gmsh: a finite element mesh generator with built-in pre- and post-processing facilities, 1996. `http://www.geuz.org/gmsh/`.

[9] GNU. The GNU general public license, 1988. `http://www.gnu.org/licenses/gpl.html`.

[10] D. Heller, P. M. Ferguson, and D. Brennan. *Motif Programming Manual*, volume 6A. O'Reilly, second edition, 1994.

[11] B. Joe. GEOMPACK – a software package for the generation of meshes using geometric algorithms. *Adv. Eng. Software*, 13:325–331, 1991.

[12] J. R. Levine, T. Mason, and D. Brown. *Lex & Yacc*. O'Reilly, 1992.

[13] A. Liu and B. Joe. Relationship between tetrahedron shape measures. *BIT Numerical Mathematics*, 34(2):268–287, 1994.

[14] M. Muuss. BRL-CAD, 1984. Army Research Laboratory.

[15] C. F. Ollivier-Gooch. GRUMMP — generation and refinement of unstructured, mixed-element meshes in parallel, 1998. `http://tetra.mech.ubc.ca/GRUMMP/`.

[16] F. Ortega. GMV: The general mesh viewer, 1996. `http://www-xdiv.lanl.gov/XCM/gmv/GMVHome.html`.

[17] B. Paul. The Mesa 3D graphics library, 1995. `http://www.mesa3d.org/`.

[18] P. P. Pebay and T. J. Baker. Analysis of triangle quality measures. *Mathematics of Computation*, 72(244):1817–1839, 2003.

[19] Open CASCADE S.A.S. Open cascade. `http://www.opencascade.org`.

[20] J. Schöberl. Netgen, an advancing front 2d/3d-mesh generator based on abstract rules. *Comput. Visual. Sci.*, 1:41–52, 1997.

[21] W. Schroeder, K. Martin, and B. Lorensen. *The visualization toolkit*. Prentice Hall, 1998.

[22] SDRC. I-DEAS master series, 1993.

[23] M. Segal and K. Akeley. The OpenGL graphics system: A specification. Technical report, Silicon Graphics Computer Systems, 1992.

[24] J. R. Shewchuk. Robust Adaptive Floating-Point Geometric Predicates, May 1996.

[25] J. R. Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, May 1996.

[26] H. Si. Tetgen a quality tetrahedral mesh generator and three-dimensional delaunay triangulator, 2004. `http://tetgen.berlios.de/`.

[27] Siemens PLM Software. Parasolid. `http://www.parasolid.com`.

[28] B. Spitzak. FLTK, the fast light tool kit, 1998. `http://www.fltk.org`.

[29] Paul S. Strauss. IRIS Inventor, a 3D graphics toolkit. *ACM SIGPLAN Notices*, 28(10):192–200, 1993.

[30] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, 1997.

[31] G. V. Vaughan and T. Tromey. *GNU Autoconf, Automake and Libtool*. New Riders Publishing, 2000.

[32] S. Vavasis. QMG: mesh generation and related software, 1995. `http://www.cs.cornell.edu/home/vavasis/qmg-home.html`.

[33] S. Vinoski. CORBA: integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, 35(2):46–55, 1997.