An introduction to Mesh Generation

Christophe Geuzaine Jean-François Remacle

Contents

CHAPTER 1

Combinatorial topology of meshes

1.1 Euler's formula

Consider a finite set $S = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^2$ of n distinct points in the plane. The convex hull $\Omega(S)$ is is the smallest convex set that contains S.

Definition: A triangulation $T(S)$ of S is a set of non overlapping triangles that exactly covers the convex hull $\Omega(S)$ with all points of S being among the vertices of the triangulation.

Different triangulations of the same point set S may exist (e.g. Figure 1.1), but we are going to show that they all have the same number of edges and of triangles.

Property 1.1.1 *Every triangulation $T(S)$ contains exacltly $n_f = 2(n-1) - n_h$ triangles and $n_e = 3(n-1) - n_h$ edges.*

Proof The proof uses a very well know result of Euler that he proved in 1758. Here is what Euler had to say: Consider any polyhedron and let n be the number of its vertices, n_f the number of its faces, and n_e be the number of its edges. Then

$$n + n_f - n_e = 2. \tag{1.1}$$

A commemorative stamp put out by the Swiss Post shows Euler together with that very famous formula (Figure 1.2). David Eppstein gives 20 different proofs of Euler's formula in [**?**]. Here is one that is quick and elegant. The skeleton of any convex polyhedron is a planar graph. This is geometrically easy to see: in order tu build such a planar graph, dispose the polyhedron on one on a plane and dig a hole on one of its face (an upper face). Then, enlarge this hole in order to unfold the polyhedron up to the point it is completely flattened. The upper face then becomes "infinite" and can be seen as the outer face of the graph. Figure 1.3 shows an example of such a flattening procedure: a cube is shown with its corresponding skelton that is actually a planar graph Γ. Let Γ' be the dual of Γ i.e. a graph with its 6 nodes that correspond to the faces of the cubes and its 12 edges that correspond to the edges of the cube. Edges of both graphs have a one-to-one correspondance.

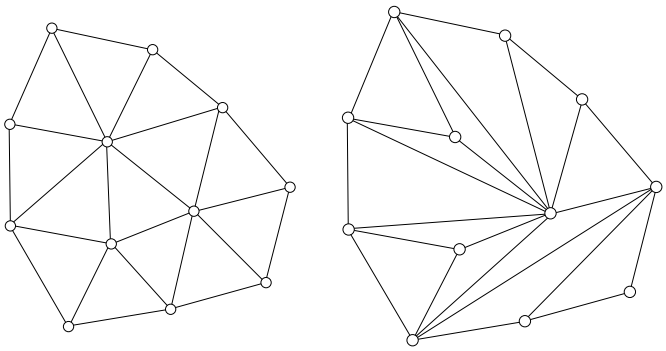


Figure 1.1: Two triangulations of the same point set, both containing $n_f = 13$ triangles defined by a total of $n = 12$ points with $n_h = 9$ points that lie on $\Omega(S)$.

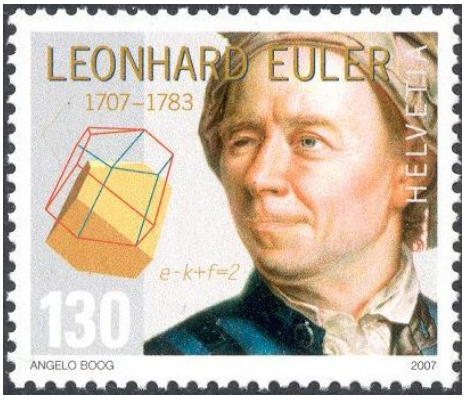


Figure 1.2: Commemorative stamp with Euler and his famous formula.

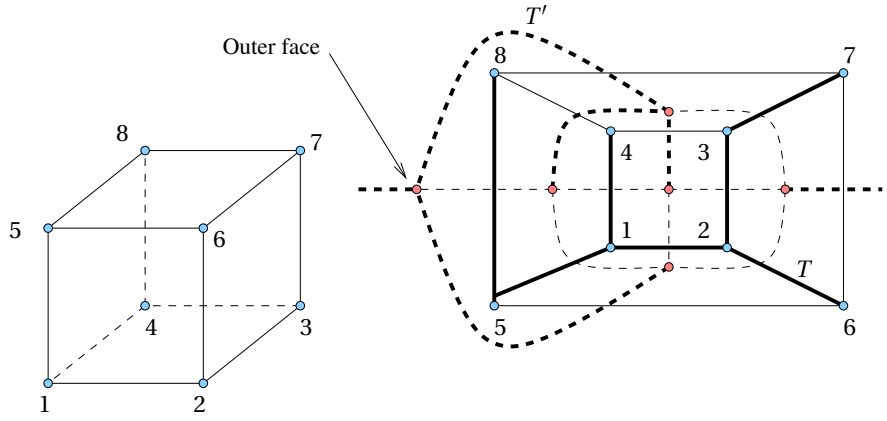


Figure 1.3: A cube (left) and its corresponding skelton Γ (right, plain lines) and the dual Γ' of Γ (right, dashed lines). Edges in bold correspond to a spanning tree T of Γ and edges in bold and dashed correspond to a spanning tree of Γ'.

Let T be a spanning tree of Γ i.e. a subgraph $T \subset \Gamma$ that includes all of the vertices of Γ and that is a tree i.e. that contains no cycles. T does not contain any cycles, so it does not disconnect the plane. The co-tree T^* of T is the set of edges of the graph that are not in T. Consider now the set of edges $T' \subset \Gamma'$ that correspond to T^*. Set T' contains no cycles: if one cycle exists in T', then the corresponding edges of Γ would create some isolated vertices in T, which is impossible because T is a spanning tree and it contains all vertices of Γ. T' contains all vertices (the faces of the polyhedron) of Γ' because T does not contain any cycles. Then, T' is a spanning tree of Γ'.

The number of edges on a spanning tree can be computed in a general fashion. Let's construct a spanning tree in the following way: start with one random edge e of Γ and add it to T. This first edge e connects 2 vertices that are inserted in a stack. While this stack is not empty, we take the vertex v at the top of the stack and look for all edges $e_i(v, v_i)$ that are incident to v. We add e_i to T if neither v or v_i is not yet in T. So, each edge of T correspond to one vertex of Γ, except the first one that correspond to two. Then, a spanning tree has exactly $n-1$ vertices.

So T has n vertices and $k \equiv n-1$ edges. Similarly, T' has n_f vertices and $k' \equiv n_f - 1$ edges. Since $k + k' = n_e$, we have $n - 1 + n_f - 1 = n_e$ and formula 1.1 follows.

1.1.1 Platonic solids

As a first example of use of Euler's formula, it is easy to prove that there exist only 5 platonic solids i.e. regular, convex polyhedron. Let m denote the number of edges and vertices of each face and k the degree of each vertex i.e. the number of faces adjacent to the vertex.

We require that k is the same for every vertex: each vertex has k adjacent face

and each face has m vertices so that so that $mn_f = kn$. Each edge has 2 adjacent faces and each face has m edges. This leads to

$$mn_f = kn = 2n_e.$$

Using Euler's formula $n - n_e + n_f = 2$, we have

$$n\left(1 + \left(\frac{k}{m} - \frac{k}{2}\right)\right) = 2$$

which leads to

$$(2m + 2k - mk)n = 4m.$$

Since $n > 0$ and $m > 0$, we have to have

$$2m + 2k - mk = -(k-2)(m-2) + 4 > 0$$

or

$$(k-2)(m-2) < 4.$$

Since $k \geq 3$ and $m \geq 3$, we have the following possible solutions for (m,k): $(3,3)$, $(4,3)$, $(5,3)$, $(3,4)$ and $(3,5)$ which correspond respectively to the tetrahedron , the cube, the octahedron, the dodecahedron and the icosahedron.

1.2 Euler-Poincaré characteristic

Euler's formula applies to polyhedron i.e. meshes that are topologically equivalent to a sphere. Assume a closed orientable surface $\mathscr{S}$. The genus g of $\mathscr{S}$ is an integer representing the maximum number of cuttings along non-intersecting closed simple curves without rendering the resultant manifold disconnected. Consider a sphere. There exist no closed curve on the sphere that does not divide it into two disconnected parts: its genus is $g = 0$. A simple torus has a genus of one. The genus of a surface $\mathscr{S}$ can be defined in terms of its Euler characteristic χ (see §1.1). Both g and χ carry the same topological information and their relationship (valid for orientable closed surfaces) is

$$\chi = 2 - 2g. \tag{1.2}$$

For a sphere, we have obviously $\chi = 2$. Figure 1.4 shows manifold meshes of different objects together with their Euler characteristic.

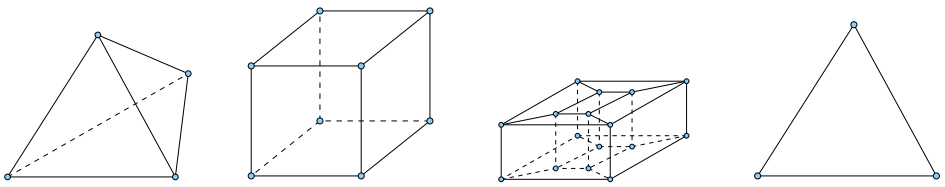
Formula (5.1) is valid for closed surfaces. Assume that surface $\mathscr{S}$ has b boundaries, then the Euler-Poincare characteristic changes to

$$\chi = 2 - 2g - b. \tag{1.3}$$

This corresponds to the topology of a sphere with g handles and b holes. For example, a cylindrical topology can be constructed by opening two separated holes in a sphere. Then $\chi = 2 - 2 \times 0 - 2 = 0$.

Euler's formula can be generalized to general 2D manifolds as

$$n - n_e + n_f = \chi. \tag{1.4}$$



Sphere ($\chi = 2$) $n = 4$, $n_e = 6$, $n_f = 4$ | Sphere ($\chi = 2$) $n = 8$, $n_e = 12$, $n_f = 6$ | Torus ($\chi = 0$) $n = 16$, $n_e = 32$, $n_f = 16$ | Disk ($\chi = 1$) $n = 3$, $n_e = 3$, $n_f = 1$

Figure 1.4: Computation of Euler's characteristic χ for different manifold meshes.

1.3 Poincaré-Hopf Theorem

Assume a surface $\mathscr{S}$ and its Euler-Poincaré characteristic χ. Poincaré-Hopf Theorem is stated as follows: let be a vector field on $\mathscr{S}$ with K isolated zeroes z_i (a zero is an isolated singularity of the field). If $\mathscr{S}$ has a boundary, the is on its normal direction, pointing outside $\mathscr{S}$. Then we have the formula

$$\sum_{i=1}^{K} \text{index}(z_i) = \chi.$$

The index of the singularity is $+1$ for a source singularity and -1 for a saddle singularity. It is possible to develop discrete versions of this theorem.

1.4 Topology of triangular meshes

Now let's specialize Euler's formula to triangulations. Assume a surface $\mathscr{S}$ with genus g and that has b boundaries. Assume a triangulation of that surface that has n vertices, n_e edges and n_f triangles. We assume finally that n_h vertices and n_h edges of the triangulation are situated on the boundaries.

If $n_h = 0$, then every edge of the triangulation is is connected to exactly 2 triangles and each triangle has (obviously) 3 incident edges. We have then

$$2n_e = 3n_f.$$

This last result combined with Eulers's formula gives, for a closed surface (no holes)

$$n_f = 2(n - \chi) \text{ and } n_e = 3(n - \chi).$$

It is easy to take into account boundaries: if n_h edges are on the boundary of the triangulation, then $n_e - n_h$ edges are internal with two adjacent triangles and n_h edges and have only one adjacent triangle. Every triangle being always incident to 3 edges, we get the following result

$$3n_f = 2(n_e - n_h) + n_h. \tag{1.5}$$

Combining Euler's formula (5.7) with (1.5) gives the result $n_f = 2(n-1) - n_h$ and $n_e = 3(n-1) - n_h$. ∎

A triangulation is composed of a collection of "entities" (triangles, edges and points) together with their adjacencies. Any entity bounds and/or is bounded by other ones of higher and/or lower dimension. This *adjacency information* represents the graph of a mesh. In many cases, algorithms that deal with triangulation have to keep track of "upward adjacencies" i.e. the set of triangles or of edges that are adjacent to a given vertex or the triangles that are adjacent to an edge. The number of triangles and edges adjacent to a vertex are called respectively n_{vf} and n_{ve}.

Proposition 1.4.1 *Consider a triangulation T with n points and n_h points on its convex hull $\Omega(T)$. We claim here that a point of T has in average $n_{vf} = 6 - \frac{3n_h+6}{n}$ adjacent triangles and $n_{ve} = 6 - \frac{2n_h+6}{n}$ adjacent edges.*

Proof A triangle is adjacent to three vertices and a vertex is adjacent to n_{vf} triangles. This leads to

$$n_{vf} n = 3n_f = 3\,(2(n-1) - n_h)$$

and we have the result

$$n_{vf} = 6 - \frac{3n_h + 6}{n}. \tag{1.6}$$

Similarly, n_{ve} that is the number of edges adjacent to a vertex can be computed as

$$n_{ve} n = 2n_e = 2\,(3(n-1) - n_h)$$

which gives

$$n_{ve} = 6 - \frac{2n_h + 6}{n}. \tag{1.7}$$

∎

Figure 1.5 illustrate equation (1.6). The average number of adjacencies per entity in the triangulation is know in advance. Yet, as it is seen on Figure (1.5), this number varies from one vertex to another.

1.4.1 Regular triangulations

An internal vertex in a triangular mesh is regular if it has exactly 6 adjacent triangles. Its is said to have a valence of 6. A regular triangulation is a triangulation with regular vertices only. Regular triangulations are usually desirable because they allow regular sampling of the points and good geometrical properties of the triangulation (equilateral triangles in planar triangulations). At that point, let us see wether a triangulation made of regular vertices only is possible. In the triangulation of a closed surface, each triangle is made of 3 edges while each edge has two adjactent triangles. This means that $2n_e = 3n_f$ which leads to

$$n_f = 2(n - \chi).$$

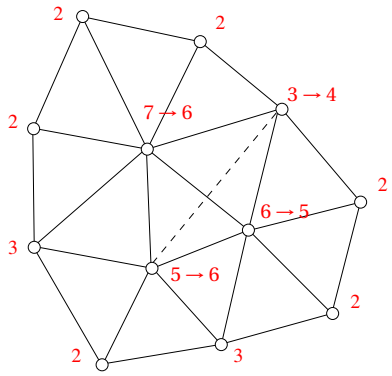


Figure 1.5: A triangulation T with $n = 12$ and $n_h = 9$. The average number of triangles adjacent to a vertex is (see (1.6)) $n_{vf} = 6 - \frac{3\times 9+6}{12} = 3,25$. This average can also be computed explicitely: $n_{vf} = \frac{39}{12} = 3,25$.

If we assume a mesh with regular vertices only, then $3n_f = 6n$ and the conclusion is that only torus-like surfaces with $\chi = 0$ can be covered with a perfecly regular triangular mesh. Introducing n_k irregular vertices of valence $6-k$ leads to

$$3n_f = 6(n - n_k) + (6 - k)n_k.$$

and the discrete version of Poincaré-Hopf Theorem for triangular meshes is:

$$\sum_k \frac{kn_k}{6} = \chi. \tag{1.8}$$

Twelve irregular vertices of valence 5 are require to triangulate a sphere. The simplest version of this mesh is the icosaedron with $n_f = 20$, $n_e = 30$ and $n = 12$, each of the vertices being of valence 5.

1.5 Topology of quadrilateral meshes

When surface $\mathscr{S}$ of genus g has b boundaries, $\chi = 2 - 2g - b$. Some vertices are situated on the boundaries of $\mathscr{S}$: assume that their number is n_h. Then, we can use the usual trick:

$$4n_f = 2(n_e - n_h) + n_h$$

combined with

$$n - n_e + n_f = \chi$$

leads to

$$n - n_f = \chi + \frac{n_h}{2}.$$

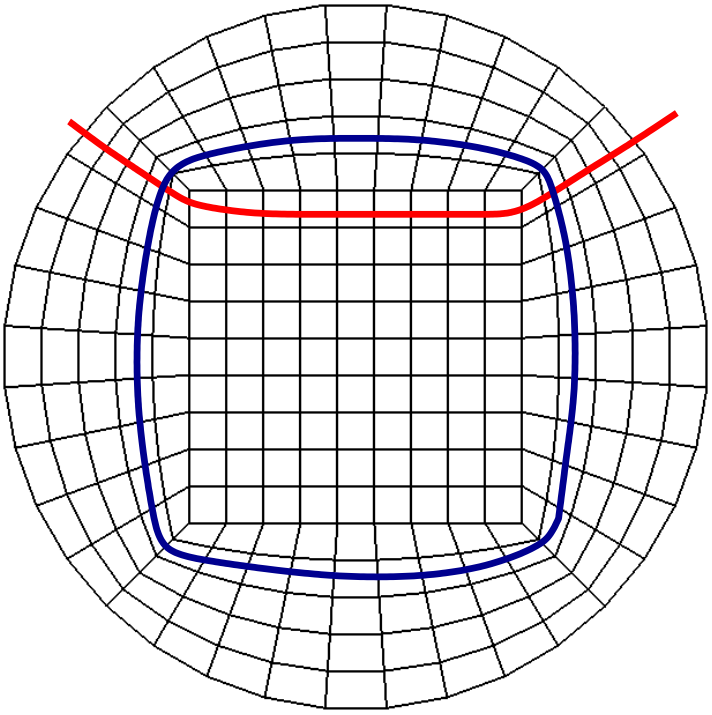


Figure 1.6: A quadrilateral mesh of a circle. Two dual curves have been drawn, one being closed (in blue) and one ending at the boundary.

If boundaries are considered, n_h has to be an even number for allowing a fully quadrilateral mesh regardless of the topology of $\mathscr{S}$.

Another way to view this problem is to dualize, that is, take the dual subdivision. The dual of a quadrilateral mesh is characterized by the property that edges meet in 4's (each quad has exactly 4 neighbors). The dual of a quadrilateral mesh can be drawn as a collection of curves that are either closed or that meet the boundary of the domain at two edges (see Figure 1.6). So a necessary condition for a quadrilateral mesh to exist is that there be an even number of edges on the bounary.

1.5.1 Regular quadrangulations

Assume a quadrangulations of $\mathscr{S}$ made of n vertices, n_e edges and n_f quadrilaterals. An internal vertex in a quadrilateral mesh is regular if it has exactly 4 adjacent quads. Its it said to have a valence of 4. A regular quadrangulation is a quadrangulation with regular vertices only.

At that point, let us see wether a quadrangulation made of regular vertices only is possible. In the quadrangulation of a closed surface, each quadrilateral is made of

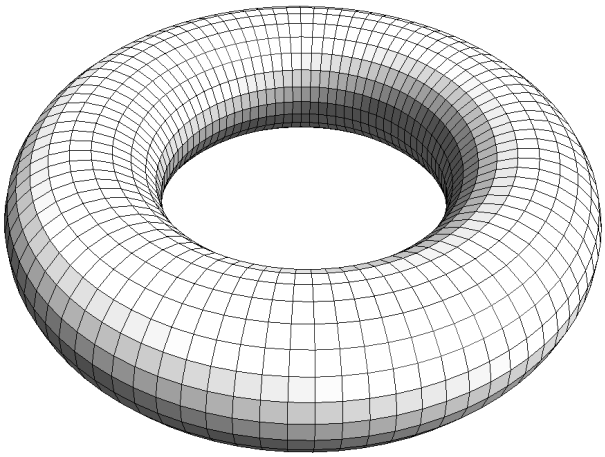


Figure 1.7: A fully regular quadrilateral mesh of a torus

4 edges while each edge has two adjactent quadrilaterals. This means that $n_e = 2n_f$ which leads to

$$n - n_f = \chi.$$

If every vertex is regular, then each vertex has 4 quadrilateral neighbors and each quadrilateral has 4 vertices. Then, $n_f = n$ which means that only torus-like closed surfaces with $\chi = 0$ can be covered with a regular quadrangular mesh (see Figure 5.1).

If $\chi \neq 0$, irregular vertices have to be present in the mesh, i.e. vertices of valence different than 4. Let us now how irregular vertices affect the Euler-Poincaré characteristic. Assume n_k vertices of valence $4-k$ and $n-n_k$ vertices of valence 4. We have

$$4n_f = 4(n-n_k) + (4-k)n_k$$

which means that

$$4n - 4(n-n_k) - (4-k)n_k = 4\chi$$

or

$$\chi = \frac{kn_k}{4}.$$

Each irregular vertex of valence $4-k$ counts for $k/4$ in the Euler characteristic. Each irregular vertex of valence 3 adds $1/4$ to the Euler-Poincare characteristic of the surface: its index is $1/4$. Each irregular vertex of valence 5 adds $-1/4$ to the Euler-Poincare characteristic of the surface and its index is $-1/4$. More generally, we obtain the following discrete version of Poincaré-Hopf Theorem:

$$\sum_k \frac{kn_k}{4} = \chi. \tag{1.9}$$

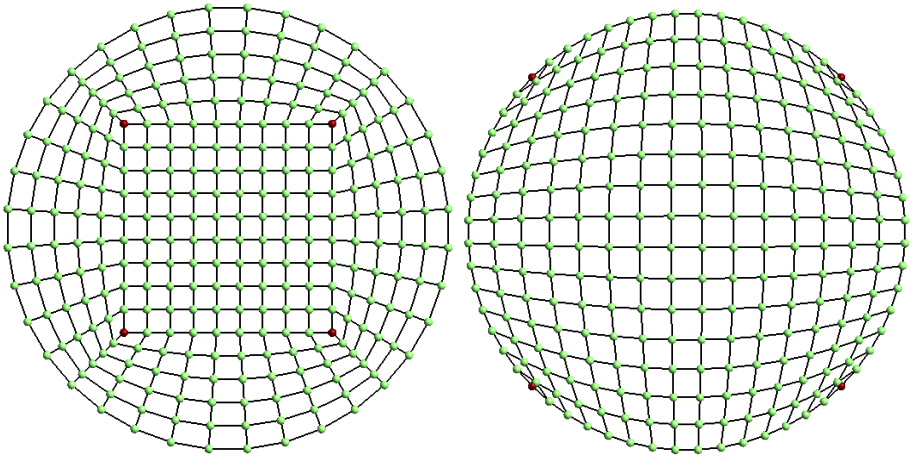


Figure 1.8: A quadrilateral mesh of a circle. Four singularities of index 1/4 (in red) are required to obtain such a mesh. The singularities may be inside the disk (left) or on its boundary (right)

A sphere can then be quadrilateralized using $n_1 = 8$: the simplest version of this mesh is the cube with its 8 corners of valence 3.

In order to extend Poincaré-Hopf to meshes of a domain with boundaries, the regular valence of a vertex on a boundary has to be defined. For quadrilateral meshes, the regular valence of a vertex on the boundary is 2. This is easily justified by the fact that the boundary of the mesh is along one of the two vector fields so the vertex is not a singularity. Assume that $\mathscr{S}$ is a disk. A disk has the topology of a sphere $g = 0$ with one hole in it $b = 1$ so $\chi = 2 - 2 \times 0 - 1 = 1$. Four singularities of index 1/4 are needed to build a quad mesh on the disk. Those 4 singularities may be located anywhere in the disk, eventually on its boundary (see Figure 5.2). A vertex of the boundary has an index 1/4 if it has only one adjacent quadrangle and $-1/4$ if it has 3 adjactent quadrangles. Figure 5.2 (right) shows a quadrilateral mesh of a circle with all 4 singularities on the boundary. There, the unique quadrilateral adjactent to each singularity is ill shaped so it's not a good idea to have irregular vertices on smooth boundaries.

If the boundary is non smooth, it may be a good idea to locate some irregular vertices on the boundary. More precisely, we distinguish external (or reentrant) corners which external angles are about 90 degrees and internal corners which internal angles are about 90 degrees. Irregular vertices of degree $-1/4$ are suitable for reentrant corners while irregular vertices of index 1/4 are suitable for internal corners. Figure 5.3 shows two quadrilateral meshes of a domain with 4 internal corners and 1 reentrant corner. The mesh on the left is the one that has 6 non regular vertices on all corners of the boundary. The right mesh has the minimum amount of non regular vertices. Here, one internal corner and the reentrant corner have been regularized. The two meshes are both valid: choosing either one or the other depends on

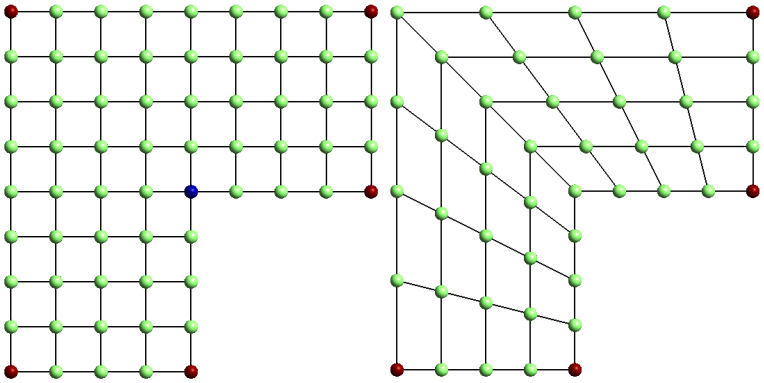


Figure 1.9: Quadrilateral meshes of a non smooth domain. Five singularities of index $1/4$ (in red) and one singularity of index $-1/4$ (in blue) are required to have the sum of the indices to be one (left). It is also possible to use 4 irregular nodes only (right), leading to a different result.

the underlying application. More specifically, those two configurations are typical of boundary layer meshes.

Up to now, we have assumed that it is possible to build a quadrilateral mesh with a minimum amount of irregular vertices. Even though this is the ideal situation, building such a “perfect” mesh is usually difficult. Figure 5.4 shows a quadrilateral mesh that has been generated using a standard technique. It has 8 vertices of valence 5 and 12 vertices ov valence 3.

1.6 Generalization to higher dimensions

1.6.1 Simplical complexes

A d-simplex σ is the convex hull of $d+1$ affinely independent vertices $p_0,\dots,p_d$. Triangles are $2D$ simplices and tetrahedra are $3D$ simplices. The orientation of σ $|\sigma|$ is induced by the ordering of its vertices. For any permutation π of points, the orientation of σ is given as $(-1)^{\text{sign}(\pi)}|\sigma|$. The boundary of σ is composed of $d-1$ faces: the k-th face of σ is constructed by ommiting vertex p_k. It is a simplex of dimension $d-1$ and its orientation is induced by the orientation of σ.

A simplical complex S is a finite set of simplices in R^d. The simplices of S are topologically glued together in a precise fashion.

as in a mesh i.e intersection of celles are themselves lower dimensional topological complexes:

A cellular complex C (or CW complex, C standing for "closure-finite", and W for "weak topology") is made of basic building blocks called cells. assume c_0 points (0-cells) in a space of dimlension d. The c_1 edges (or 1-cells) are made with pairs of

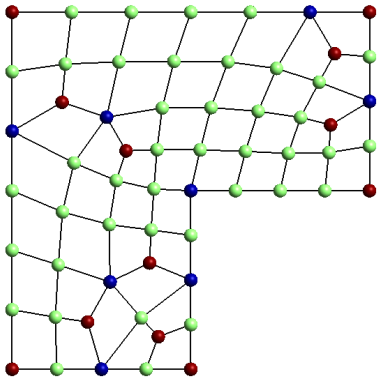


Figure 1.10: Quadrilateral mesh with 8 vertices of index $-1/4$, and 12 of index $1/4$, leading to $\chi = 12/4 - 8/4 = 1$.

points. We assume then that there exist c_j j-cells, $j = 1 \dots d$ that are topologically bounded by c_{j-1} cells. The Euler-Poincaré characteristic of C is computed as the alternate sum:

$$\chi = \sum_{k=0}^{d} (-1)^k c_k.$$

For surfaces ($d = 2$), we have seen that $\chi = 2 - 2g - b$ i.e. there exist a relation between the combinatorics of the cellular complex and the topology of the surface that is considered. Betti numbers are topological objects which were proved to be invariants by Poincaré, and used by him to extend Euler's formula to higher dimensional spaces.

Consider any polyhedron and let n be the number of its vertices, n_f the number of its faces, and n_e be the number of its edges.

CHAPTER 2

Delaunay triangulations in the plane

2.1 The Voronoi Diagram

Definition: Consider a finite set $S = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^2$ of n distinct points in the plane. The *Voronoi cell* V_i of $p_i \in S$ is the set of points x that are closer to p_i than to any other points of the set:

$$V_i = \left\{x \in \mathbb{R}^2 \mid \left\|x - p_i\right\| < \left\|x - p_j\right\|, \ \forall 1 \leq i \leq n, i \neq j\right\}$$

where $\left\|x - y\right\|$ is the euclidian distance between x and y.

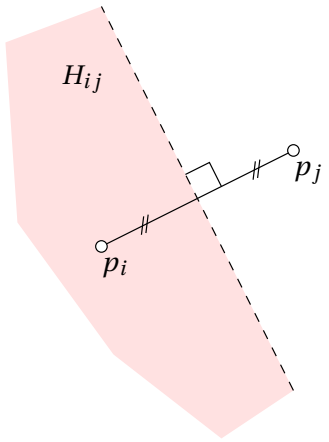


Figure 2.1: Points p_i and p_j, their perpendicular bissector (in dashed lines) and half-plane H_{ij}.

Consider first the case where $S = \{p_i, p_j\}$. The *p*erpendicular bisector of the line segment $p_i p_j$ is a line perpendicular to $p_i p_j$ and passing through its midpoint. The

perpendicular bissector of $p_i p_j$ divides $\mathbb{R}^2$ into two halfplanes H_{ij} and H_{ji}:

$$H_{ij} = \{x \in \mathbb{R}^2 \mid \|x - p_i\| < \|x - p_j\|\}.$$

Here, we have clearly $V_i = H_{ij}$. The perpendicular bissector of line segment $p_i p_j$ is the intersection of the closures of the two half planes: $\overline{H_{ij}} \cap \overline{H_{ji}}$.

Let's make the problem a little more complicated and consider a set $S = \{p_i, p_j, p_k\}$ of 3 points. The Voronoi cell associated to p_i is the intersection of half planes H_{ij} and H_{ik}: $V_i = H_{ij} \cap H_{ik}$ (see Figure 2.2).

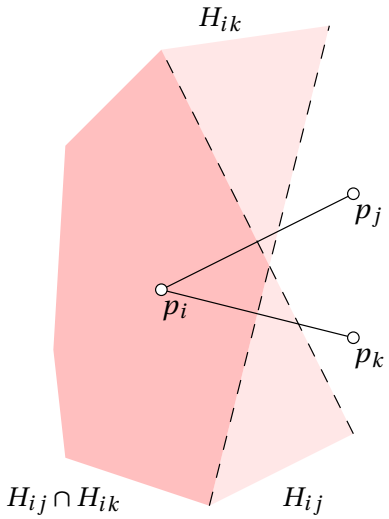


Figure 2.2: Points p_i, p_j and p_k and their perpendicular bissectors.

In the general case, the Voronoi cell relative to p_i is the intersection of all half planes:

$$V_i = \bigcap_{1 \le j \le n, j \ne i} H_{ij}. \tag{2.1}$$

By definition (2.1), each Voronoi cell V_i is the intersection of open half planes containing vertex p_i. The intersection of two convex polygon being itself a convex polygon, V_i is therefore *a convex polygon.*

Definition: *The Voronoi diagram* $V(S)$ is the unique subdivision of the plane into n cells is the union of all Voronoi cells V_p:

$$V = \bigcup_{1 \le i \le n} V_i. \tag{2.2}$$

Each point $x \in \mathbb{R}^2$ having at least one closest point in S, the Voronoi diagram covers the entire plane. Different Voronoi regions are disjoint. Therefore, the Voronoi diagram is unique.

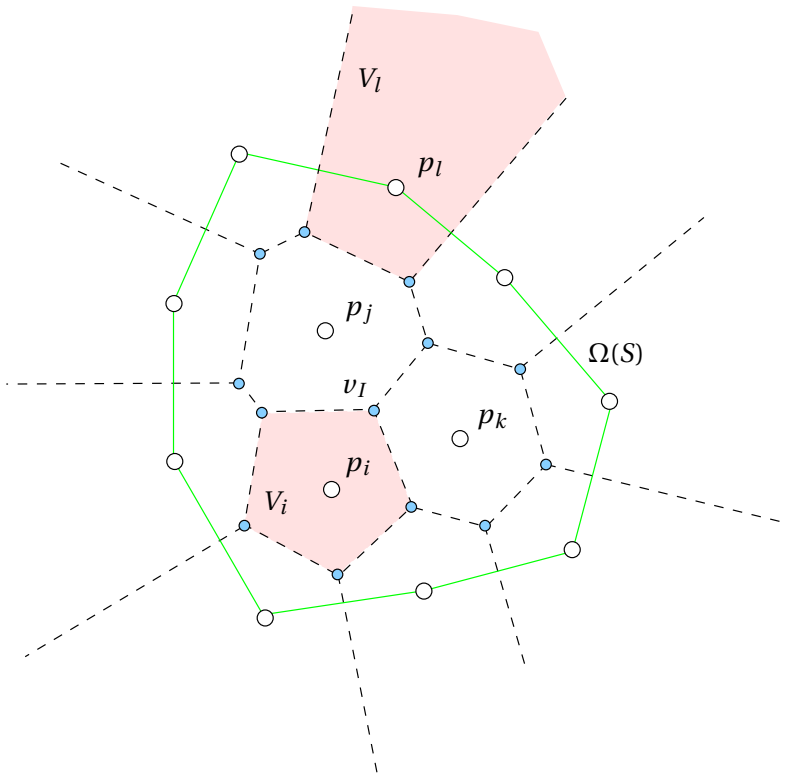


Figure 2.3: Voronoi Diagram. Voronoi cell V_i is closed because it correspond to point p_i that is not on $\Omega(S)$. Voronoi cell V_l is open because $p_l \in \Omega(S)$.

Definition: The *convex hull* $\Omega(S)$ of a finite point set S is the smallest convex polygon that contains S.

Voronoi cells are either closed or open. They can only be open for points (like p_l in see Figure 2.3) that are located on the convex hull $\Omega(S)$ of the point set.

2.2 The Delaunay triangulation

The Delaunay triangulation DT(S) is the geometric dual of the Voroinoi diagram (see Figure 2.4). The Voronoi diagram V is made of n Voronoi cells V_i that correspond to the points p_i, $1 \le i \le n$ of S. The line segments that form the boundaries of Voronoi cells and are the Voronoi edges. Voronoi edges are orthogonal bissectors of neighboring points in the diagram. The endpoints of the Voronoi edges are called Voronoi vertices $v_I, 1 \le I \le N$, N being the number of Voronoi vertices. Voronoi vertices v_I

are those points that are equidistant to three or more vertices.

Definition: Points of S are said to be *in general position* if there exist no quadruplet of points of S that are co-circular.

When the points of S are in general position, Voronoi vertices are *triple points* i.e. they are equidistant of three points of S. Consider a Voronoi Vertex v_I that is equidistant to points p_i, p_j and $p_k \in S$ (see Figure 2.3). Voronoi point $v_I = H_{ij} \cap H_{jk} \cap H_{ki}$ is the circumcenter of a triangle $\Delta_I = p_i p_j p_k$.

Definition: The Delaunay triangulation DT(S) is the triangulation of S that consist in the union of the N triangles $\Delta_I, 1 \leq I \leq N$ that correspond to the triple points of the Voronoi diagram (see Figure 2.4).

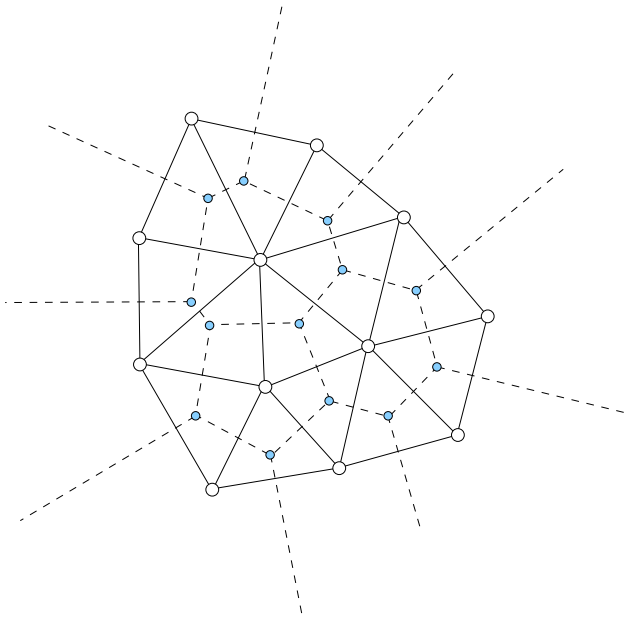


Figure 2.4: Voronoi Diagram (in dashed lines) and Delaunay triangulation. White points are points of S and blue points are Voronoi vertices that are the circumcenters of the triangles.

We should now show that the set of triangles in question is a triangulation in the sense of Definition 1.1. If this is the case, then Property 1.1.1 applies and $N = 2(n - 1) - n_h$. The fact that DT is a triangulation will be the consequense of the following properties of the Delaunay triangles. The fact that DT is a triangulation will be the consequence of the two following properties.

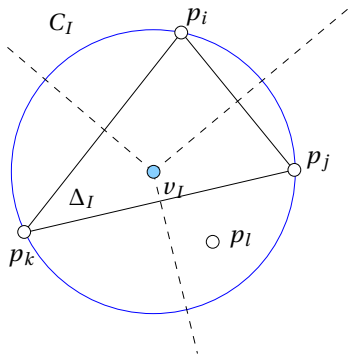


Figure 2.5: Illustration of why the empty circle property is true.

2.2.1 The empty circumcircle property

We will first demonstrate the following remarkable result that is called the empty circumcircle property.

Property 2.2.1 *The empty circumcircle of any triangle in the Delaunay triangulation is empty i.e. it contains no point of S.*

Proof Consider the Delaunay triangle $\Delta_I = p_i p_j p_k$ (see Figure 2.5). Assume now that point $p_l \in C_I$ where C_I is the circumcircle of Δ_I. By definition, the triple point v_I is at equal distance to p_i, p_j and p_k and no other points of S are closer to v_I than those three points. Then, if a point like p_l exist in S, v_I is not a triple point and triangle Δ_I cannot be a Delaunay triangle.

2.2.2 Delaunay Edges

It is useful at that point to look at some geometrical properties of circle bundles that share two points p_i and p_j. The centers of such circles lie on the perpendicular bissectors of line segment $p_i p_j$ (see Figure 2.6). Edge $p_i p_j$ divides disk C_1 into two disk sectors and one of the two sectors completely lies inside C_2. On the Figure, the pink sector of C_1 is inside C_2 and the yellow sector of C_2 lies inside C_1.

Definition: An edge $p_i p_j$ of a triangulation is a *Delaunay edge* if there exist a circle that contains p_i and p_j and that is empty i.e. that contain no point of S.

Property 2.2.2 *A mesh is a Delaunay Triangulation if and only if all its edges are Delaunay edges.*

Proof Let us first show that a Delaunay triangulation has only Delaunay edges. Assume a Delaunay triangulation $T(S)$ and an edge $p_i p_j$ that is not Delaunay. This means that there exist no circle passing through p_i and p_j that is empty. Consider

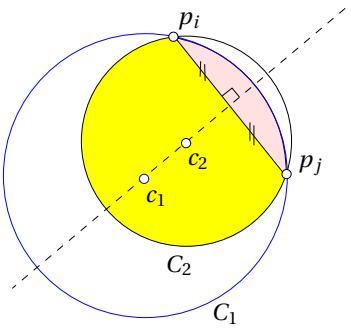


Figure 2.6: Two circles C_1 and C_2 sharing an edge $p_i p_j$. The centers of the circles c_1 and c_2 lie on the perpendicular bissector of segment $p_i p_j$ (in dashed lines).

Delaunay triangle $\Delta_I = p_i p_j p_k$ that contains edge $p_i p_j$. Its circumcircle is empty by definition because T is a Delaunay triangulation. This is in contradiction with the hypothesis that there exist no circle passing through p_i and p_j and that is not empty.

Now let's proof that if every edge of a triangulation is Delaunay, then every triangle is Delaunay as well. Assume that triangle $\Delta_I = p_i p_j p_k$ is not Delaunay, but all its 3 edges $p_i p_j$, $p_i p_k$ and $p_j p_k$ are Delaunay. Figure 2.7 shows a configuration whith a non Delaunay triangle $\Delta_I = p_i p_j p_k$ which circumcircle contains p_l. Because we deal with triangulations as defined in Definition 1.1, p_l cannot be inside triangle Δ_I. It is then situated inside one of the three circular sectors delimited by p_i, p_j and p_k. Assume that p_l and p_j are on opposite sides of $p_i p_k$ like in Figure 2.7. By hypothesis, there exist a circle passing through p_i and p_k and that is empty. The center of such a circle lies on the orthogonal bissector of $p_i p_k$. Any circle like C_1 with its center c_1 that is below c_I contains p_j any circle C_2 that is above c_I contains p_l, which is in contradiction with the hypothesis that there exist a circle passing through $p_i p_k$ and that is empty.

2.2.3 Local Delaunayhood

Definition: Given a triangulation $T(S)$ and an edge $p_i p_j$ in the triangulation that is adjacent to two triangles $\Delta_I = p_i p_j p_k$ and $\Delta_J = p_i p_l p_j$. We call edge $p_i p_j$ *locally Delaunay* if p_l lies on or outside the circumcircle of Δ_I.

Figure 2.8 gives an illustration of an edge $p_i p_j$ that is not locally Delaunay: point p_l lies inside circle C_I. It is easy to see that this condition is symmetric: if point p_l lies inside circle C_I, then point p_k lies inside circle C_J. We'll prove that below.

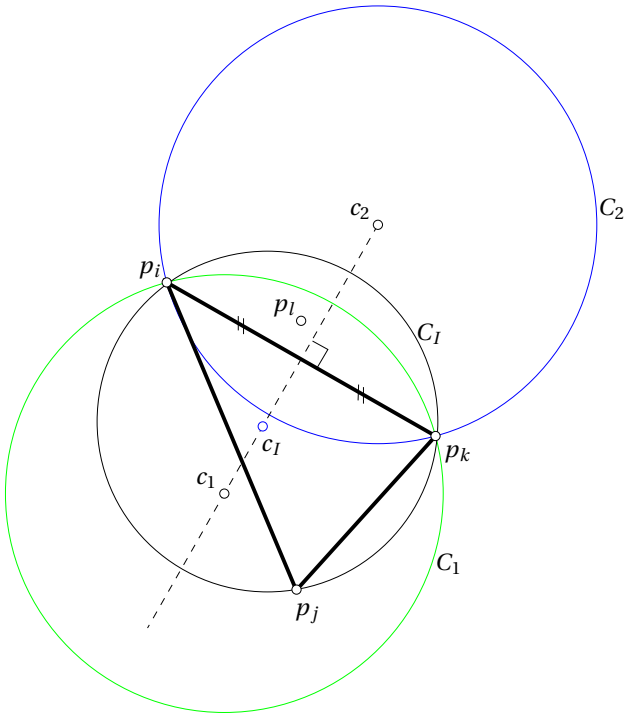


Figure 2.7: Two circles C_1 and C_2 sharing an edge $p_i p_j$. The centers of the circles c_1 and c_2 lie on the perpendicular bissector of segment $p_i p_j$.

2.2.4 Edge Flip

Consider again the situation of two triangles adjacent to edge $p_i p_j$ as depicted in Figure 2.8. Flipping edge $p_i p_j$ consist in replacing triangles $p_i p_j p_k$ and $p_j p_i p_l$ by triangles $p_l p_k p_i$ and $p_k p_l p_j$. Edge $p_i p_j$ has been flipped and replaced by edge $p_k p_l$.

The edge flip operator can only be applied to a pair of triangles that form a convex quadrilateral. If it is concave, then flipping the edge leads to an invalid configuration with two overlapping triangles (see Figure 2.9).

Property 2.2.3 *An edge that is not locally Delaunay is flippable and the new edge resulting of the flip operation is locally Delaunay.*

Proof Let us first show that any edge that is not locally Delaunay is flippable. Consider Figure 2.8. Edge $p_i p_j$ is not locally Delaunay because $p_k \in C_J$ and $p_l \in C_I$. A simple way of checking wether edges $p_i p_j$ and $p_k p_l$ can be flipped is to verify that they actually intersect. Consider triangle $p_j p_k p_i$ on Figure 2.8. The fact that p_l is on the opposite side of $p_i p_j$ than p_k and that it lies inside C_I ensures that $p_k p_l$ inter-

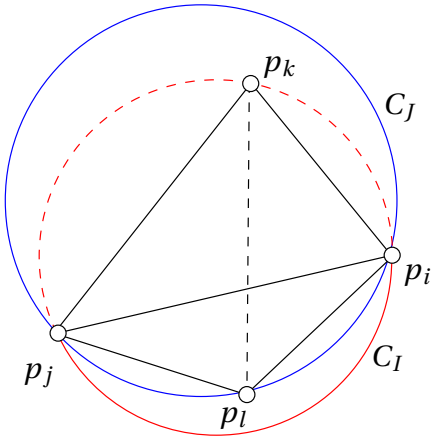


Figure 2.8: An edge $p_i p_j$ that is not locally Delaunay.

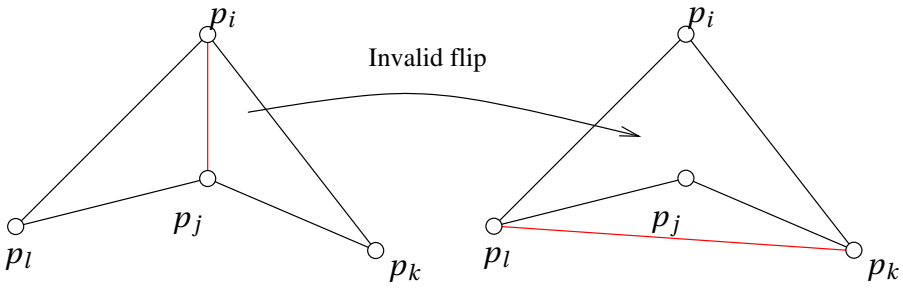


Figure 2.9: Invalid edge flip configurations.

sects $p_i p_j$ which proves that edge $p_i p_j$ is flippable if $p_i p_j$ is not locally Delaunay. Move now to Figure 2.10. and prove that, if $p_i p_j$ is not locally Delaunay, then $p_k p_l$ is locally Delaunay. In other words, we'd like to prove that, provided that p_l is inside C, then p_i is outside C'.

Circles C and C' share edge $p_k p_j$ and points p_i and p_l are on the same sides of edge $p_k p_j$. Edge $p_i p_j$ is not Delaunay by hypothesis. Then point p_l is inside C, as well as the whole arc $\overset{\frown}{p_k p_l p_j}$ (in dashed line on Figure 2.10) of C'. Point p_i belongs to C and is on the same side of $p_k p_j$ as p_l, it is then outside C' and edge $p_k p_l$ is locally delaunay. ∎

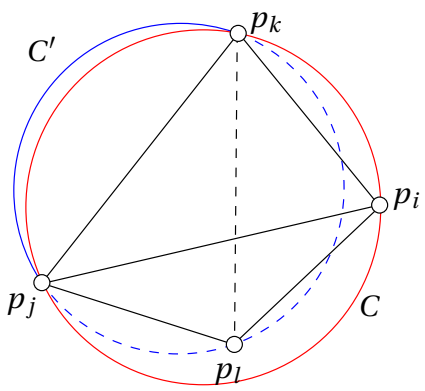


Figure 2.10: If $p_i p_j$ is not locally Delaunay, then $p_k p_l$ is locally Delaunay.

2.2.5 Locally Delaunay vs. Globally Delaunay

Property 2.2.4 *If all edges of triangulation $T(S)$ are locally Delaunay, then T is the Delaunay triangulation $DT(S)$.*

The fact that a specific edge is locally Delaunay does not imply that both its two adjacent triangles are Delaunay triangles. Yet, if all edges are locally Delaunay, then the resulting triangulation is Delaunay.

Proof We prove property 2.2.4 by contradiction. Assume all edges of a triangulation to be locally Delaunay. Assume that triangle $\Delta_I = p_i p_j p_k$ has its circumcircle C_I that contains point $p_l \in S$. The situation is summarized on Figure 2.11. Assume that point p_l and p_i are on opposite sides of $p_j p_k$. Edge $p_j p_k$ is locally Delaunay but triangle $p_i p_j p_k$ is not Delaunay because its circumcircle is not empty (it actually contains point p_l). Consider triangle $p_k p_j p_m$. Points p_i and p_m are on opposite sides of $p_j p_k$ and and p_m is outside C_I. This implies that C_J contains p_l as well. We can continue that and show that C_K and C_L both contain p_l as well. Yet, edge $p_0 p_n$ is supposed to be locally Delaunay which means that p_l should be outside C_L. This is indeed a contradiction.

2.2.6 The Flip Algorithm

Result 2.2.4 is of high importance. Combined with the flip algorithm, we can forsee a simple algorithm that would start with any triangulation $T(S)$ and would produce the Delaunay triangulation DT(S) using edge successive flips. The algorithm could be summarized as follows

- Insert all the internal edges of $T(S)$ in a `stack`.
- Do while the `stack` is not empty

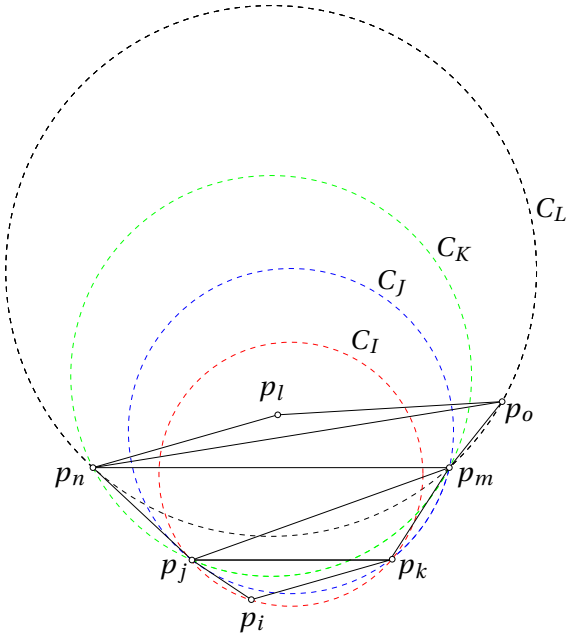


Figure 2.11: An edge $p_i p_j$ that is locally Delaunay (point p_m is outside C_I) but with triangle $p_i p_j p_k$ that is not Delaunay.

- Take edge $p_i p_j$ at the top of the `stack`. This edge is adjacent to triangles $p_i p_j p_k$ and $p_j p_i p_l$. If $p_i p_j$ is not locally Delaunay, then flip it and add edges $p_i p_k$,$p_k p_j$, $p_j p_l$ and $p_l p_i$ in the stack. If one of those edges was already present in the stack, update its neighbors.
- Remove $p_i p_j$ from the `stack`.

Two questions should be asked at that point: (i) does this algorithm produce the Delaunay triangulation of S and (ii) if it achieves to create DT(S), what is its complexity?

Proposition 2.2.1 *The edge flip algorithm converges to DT(S) in at most $\mathcal{O}(n^2)$ flips.*

Proof Consider an edge $p_i p_j$ that is not Delaunay (Figure 2.12) with its two adjacent triangles $p_i p_j p_k$ and $p_j p_i p_l$ and their respective circumcircles C_I and C_J, with $p_l \in C'_J$ and $p_k \in C'_I$. Edge flip will produce triangles $p_j p_k p_l$ and $p_i p_l p_k$ and their respective circumcircles C'_I and C'_J. Edge $p_k p_l$ is locally Delaunay i.e. $p_i \notin C'_I$ and $p_j \notin C'_J$.

Consider now the set of all possible point-triangle relations in a mesh T and a function $F(T)$ that counts how many of those relations violate the Delaunay empty circle property. There is at most $\mathcal{O}(n^2)$ point-triangle pairs in a mesh (see property 1.1.1). So, F's magnitude is not bigger than $\mathcal{O}(n^2)$. Assume now that edge $p_i p_j$ is

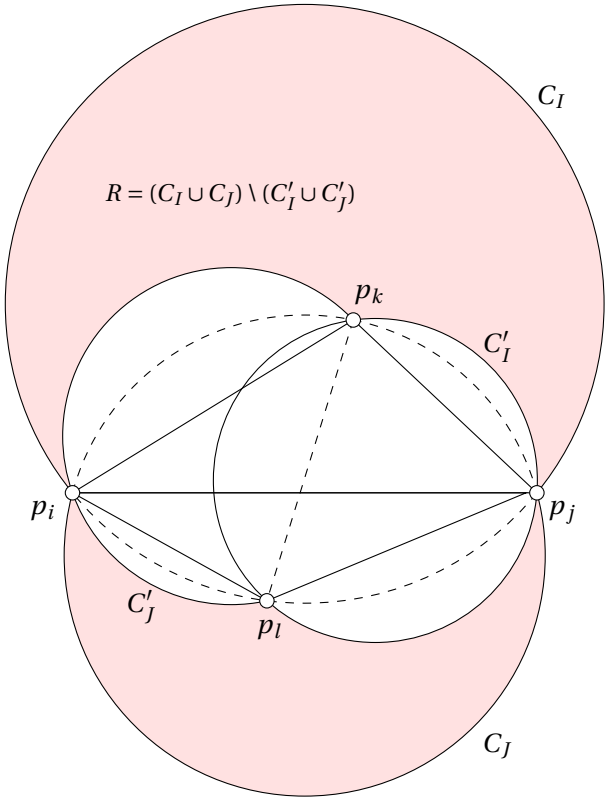


Figure 2.12: Edge flip: $C_I \cup C_J \subset C'_I \cup C'_J$.

flipped, leading to a new triangulation T'. Flipping an edge always leads to $F(T') < F(T)$. Figure 2.12 shows visually that

$$C_I \cup C_J \subset C'_I \cup C'_J,$$

the colored zone in the Figure representing

$$R = (C_I \cup C_J) \setminus (C'_I \cup C'_J).$$

If some points of S were inside circumcenters of triangles $p_i p_j p_k$ and $p_j p_i p_l$ in T, then edge flip will not increase that number because those points will not be anymore invalid. If R contains no points of S, then $F(T') = F(T) - 2$ because the two point-triangle relations associated to points p_l and p_k and triangles $p_i p_j p_k$ and $p_j p_i p_l$ disappear from F. In conclusion, we have

$$F(T') \leq F(T) + 2$$

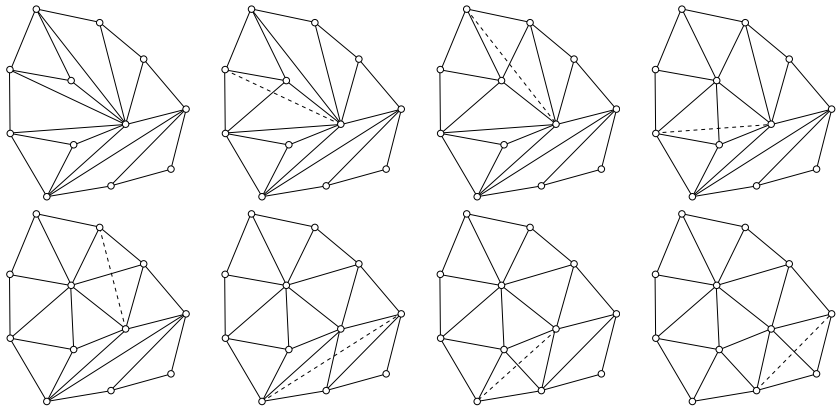


Figure 2.13: Building an angle-optimal triangulation using swaps.

which means that F decreases at each edge flip.

F is bounded by above by $\mathcal{O}(n^2)$. It is also bounded by below: only the Delaunay triangulation has empty circumcircles, $F(\text{DT}) = 0$. The edge flip algorithm converges to the Delaunay triangulation and its complexity is $\mathcal{O}(n^2)$ in the worst case. ∎

This result is outmost importance. It means that every triangulation $T(S)$ is connected to the Delaunay triangulation DT(S) by at most $\mathcal{O}(n^2)$ flips. It also means that any two triangulations T and T' are flip connected. Both T and T' being connected to DT, it is therefore possible to go from T to DT using flips and then from DT to T' using "back flipping". The flip-connectness of 2D triangulations allows to generate meshes of arbitrary domains with low complexity. This will be developped in further chapters. Figure 2.13 illustrate the edge flip procedure.

2.2.7 The MaxMin property

Let us first recall a very old geometry theorem from Thales.

Proposition 2.2.2 *Let C_A and C_B be two circumcircles of edge $p_i p_j$ (see Figure 2.14). Let b_1 and b_2 be two points on C_B on the same side of $p_i p_j$. Then, b_1 and b_2 see the edge $p_i p_j$ with the same angle β. Consider now point a on the same side of $p_i p_j$ as b_1 and b_2 but on circle C_A. Assume that b_1, b_2 are inside C_A. Then, $\alpha < \beta$.*

Consider a triangulation $T(S)$ with n_f triangles. This triangulation has $3n_f$ internal angles (3 angles per triangle). Consider the vector of angles $A(T) = (\alpha_1, \ldots, \alpha_{3n_f})$ sorted by increasing values. We can define such a vector for any triangulation. Each triangulation $T(S)$ has the same number of triangles so each vector $A(T)$ has the same length and it is therefore possible to compare them, e.g. lexicographically. We say that one given triangulation T is angle-optimal if $A(T) \leq A(T'), \forall T'$.

Property 2.2.5 *The Delaunay triangulation DT(S) is angle-optimal: it maximizes the minimum angle among all possible triangulations.*

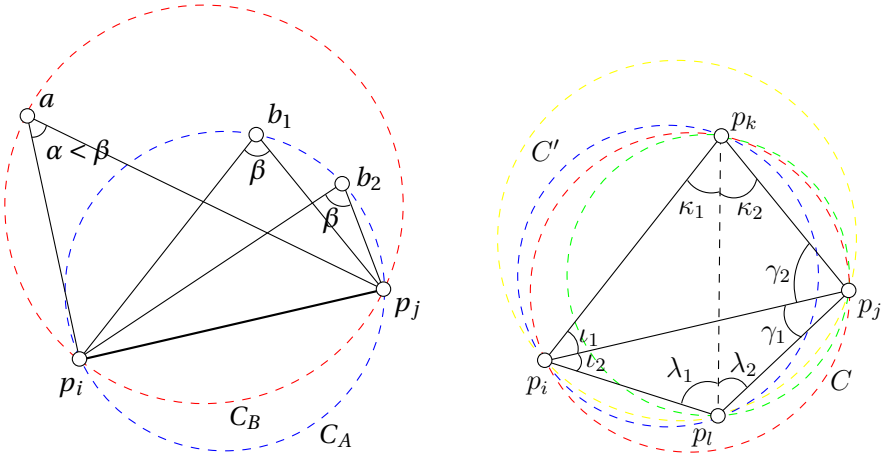


Figure 2.14: Thales theorem (left) and MaxMin property illustrated (right)

Proof Consider two triangulations T and T', where T' differs from T by one edge flip. Let us proove that $A(T') \leq A(T)$. The edge flip procedure consist in replacing triangles $p_i p_j p_k$ and $p_j p_l p_k$ by triangles $p_k p_l p_i$ (see Figure 2.14). The angles of the old configuration are respectively

$$\kappa_1 + \kappa_2, \gamma_2, \iota_1, \iota_2, \gamma_1 \text{ and } \lambda_1 + \lambda_2.$$

The angles of the new configuration are respectively

$$\iota_1 + \iota_2, \kappa_1, \lambda_1, \kappa_2, \lambda_2 \text{ and } \gamma_1 + \gamma_2.$$

Our aim is to bound by above all angles of the old configuration. Two of the 6 relations are obvious: $\gamma_1, \gamma_2 < \gamma_1 + \gamma_2$ and $\iota_1, \iota_1 < \iota_1 + \iota_2$. We use Thales Theorem 2.2.2 for the last four ones. Thales Theorem applied respectively to segments $p_i p_l$ (blue and yellow circles), $p_j p_k$ (red and green circles), $p_i p_k$ (blue and red circles) and $p_l p_j$ (yellow and green circles) gives

$$\gamma_1 < \kappa_1, \ \iota_1 < \lambda_2, \ \gamma_2 < \lambda_1 \text{ and } \iota_2 < \kappa_2$$

which are the four relations that were needed. Successive edge flips lead to the Delaunay triangulation and each flip does not increase the minimum angle. The Delaunay triangulation is therefore angle-optimal. ∎

CHAPTER 3

Construction of 2D Delaunay Triangulations

3.1 The Delaunay Kernel

Let DT_n be the Delaunay triangulation of a point set $S_n = \{p_1, \ldots, p_n\} \subset \mathbb{R}^2$ that are in general position. We describe an incremental process allowing the insertion of a given point $p_{n+1} \in \Omega(S_n)$ into DT_n and to build the Delaunay triangulation DT_{n+1} of $S_{n+1} = \{p_1, \ldots, p_n, p_{n+1}\}$.

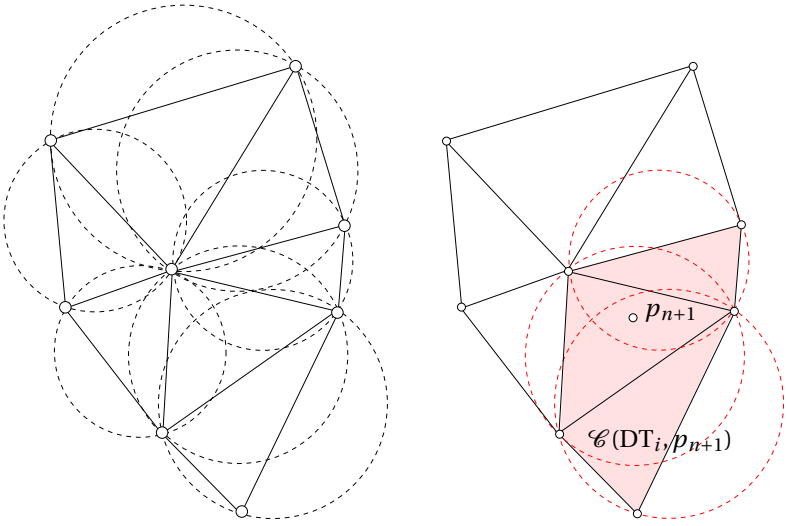


Figure 3.1: Delaunay triangulation T_n (left) and the Delaunay cavity $\mathscr{C}_p(\mathrm{DT}_n, p_{n+1})$ (right).

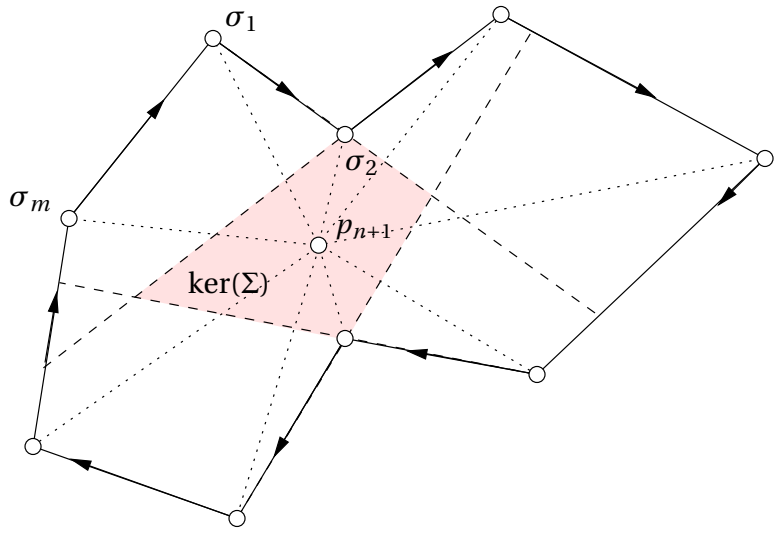


Figure 3.2: A star shaped polygon Σ and its kernel ker(Σ). All the corners σ_j, $1 \leq j \leq m$ of Σ are visible from any $x \in \ker(\Sigma)$.

Definition: The *Delaunay kernel* is the following procedure

$$\mathrm{DT}_{n+1} = \mathrm{DT}_n - \mathscr{C}(\mathrm{DT}_n, p_{n+1}) + \mathscr{B}(\mathrm{DT}_n, p_{n+1}). \tag{3.1}$$

The Delaunay cavity $\mathscr{C}(\mathrm{DT}_n, p_{n+1})$ is the set of all triangles whose circumcircles contain the new point p_{n+1} (see Figure 3.1) in consequense of what they are cannot belong to DT_{n+1}. The Delaunay ball $\mathscr{B}(\mathrm{DT}_n, p_{n+1})$ is a set of triangles that fill the polygonal hole that has been left empty while removing the Delaunay cavity $\mathscr{C}(\mathrm{DT}_n, p_{n+1})$ from DT_n.

In what follows, we will show that the Delaunay cavity $\mathscr{C}(\mathrm{DT}_n, p_{n+1})$ is star-shaped and that p_{n+1} belongs to its kernel. Then, we will explain how to build $\mathscr{B}(\mathrm{DT}_n, p_{n+1})$ in such a way that DT_{n+1} is a Delaunay triangulation.

3.1.1 Star shapeness

Consider a polygon Σ with m corners $\sigma_1, \ldots, \sigma_m$ that is bounded by m edges $\sigma_i, \sigma_{(i+1)\%m}$, $1 \leq i \leq m$.

Definition: The kernel ker(Σ) is the set of point $x \in \mathbb{R}^2$ that are visible to every σ_j i.e. the line segment $x\sigma_j$ them do not intersect any edges of the polygon.

The kernel ker(Σ) can be computed by intersection of the halfplanes that correspond to all oriented edges of the polygon (see Figure 3.2).

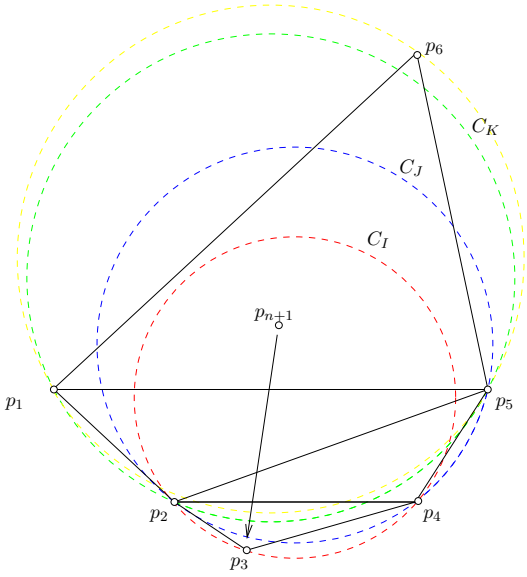


Figure 3.3: The delaunay cavity is star shaped.

3.1.2 The Delaunay Cavity

Definition: The Delaunay cavity $\mathscr{C}(T_n, p_{n+1})$ is the set of m triangles $\Delta_1, \ldots, \Delta_m \in \mathrm{DT}_n$ for which their circumcircle contains p_{n+1} (see Figure 3.1).

The Delaunay cavity contains the set of triangles that cannot belong to T_{n+1}. The region covered by those invalid triangles should be emptied and re-triangulated in a Delaunay fashion. The Delaunay cavity has some interresting properties.

Proposition 3.1.1 *The Delaunay cavity $\mathscr{C}(T_n, p_{n+1})$ is a non empty connected set of triangles which the union form a star shaped polygon with p_{n+1} in its kernel.*

Proof The proof is very similar to the one of proposition 2.2.4. Consider point p_{n+1} of Figure 3.3. Assume that p_{n+1} belongs to the circumcircle C_I of triangle $p_2p_3p_4$. Let's draw a line between p_{n+1} and p_3 which is the triangle that is the furthest away from p_{n+1}. If p_3 is our point of view, p_{n+1} is on the other side of p_2p_4. Point p_5 is outside C_I because triangle $p_2p_4p_5$ is a Delaunay triangle. Then The part of C_J which is on the orther side of p_2p_4 contains the part of C_J which is on the same side. This implies that triangle $p_2p_4p_5$ is invalid and is itself on the Delaunay cavity. We can continue that kind of argument starting with p_4, then p_2. Finally, triangle $p_1p_5p_6$ contains p_{n+1} and is obviously on the Delaunay cavity. So, any vertex of the boundary of the cavity can be seen by p_{n+1}, which proves the proposition. ∎

Property 3.1.1 *The Delaunay cavity $\mathscr{C}(T_n, p_{n+1})$ does not contain any point of S_n.*

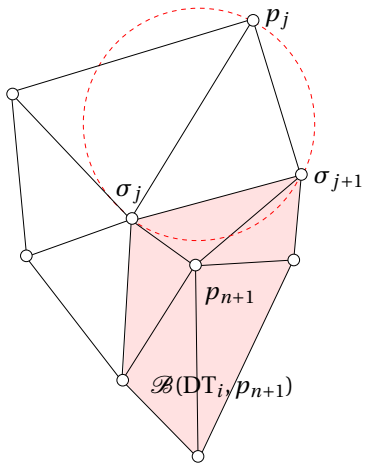


Figure 3.4: The Delaunay Ball.

Proof To do.

3.1.3 The Delaunay Ball $\mathscr{B}(\mathbf{DT}_p, p_{n+1})$

The Delaunay cavity $\mathscr{C}(\mathrm{DT}_n, p_{n+1})$ is star shaped and p_{n+1} belongs to its kernel. So, one possible solution for the Delaunay ball is to create m triangles $\sigma_i\sigma_{(i+1)\%m}p_{n+1}$, $1 \le i \le m$ that all contain the new point p_{n+1}. This procedure indeed produces the desired Delaunay triangulation. DT_{n+1}.

All triangles that are not in $\mathscr{C}(T_n, p_{n+1})$ remain in DT_{n+1}. Those triangles (e.g. $\sigma_i\sigma_{i+1}, p_j$ on Figure 3.4) are Delaunay triangles in DT_{n+1} because their circumcircles neither contain any point of S_n (DT_n is a Delaunay triangulation) nor contain p_{n+1} because they do not belong to $\mathscr{C}(T_n, p_{n+1})$. This implies that edges $\sigma_i\sigma_{i+1}$ are locally Delaunay because the circumcircle of $\sigma_i\sigma_{i+1}, p_j$ do not contain p_{n+1}. The local Delaunaynesss being symetric, it implies that circumcircle of triangle $\sigma_i\sigma_{i+1}, p_{n+1}$ do not contain p_j which proves that every edge of DT_{n+1} is locally Delaunay. Then, DT_{n+1} is the Delaunay triangulation.

3.2 The Bowyer-Watson algorithm

The Bowyer-Watson algorithm is a method for computing the Delaunay triangulation of a finite set of points S in any number of dimensions. It uses the Delaunay kernel in an incremental fashion: starting with an initial triangulation DT_0, points of S are inserted one by one in the triangulation

$$\mathrm{DT}_i = \mathrm{DT}_{i-1} - \mathscr{C}(\mathrm{DT}_{i-1}, p_i) + \mathscr{B}(\mathrm{DT}_{i-1}, p_i), \quad i = 1, \ldots, n.$$

The choice of an initial triangulation DT_0 has to be made.

3.2.1 Super-triangles

The initial Delaunay triangularion DT_0 is composed of 1 or 2 or more "super-triangles". The super-triangles cover the entire convex hull $\Omega(S)$. Super triangles contain points $S_0 = \{p_{-1}, p_{-2}, \ldots, p_{-m}\}$ that do not belong to S (see Figure 3.5).

Points p_j, $1 \leq j \leq n$ are inserted one after the other in the triangulation using the Delaunay kernel (3.1). The final result is a Delaunay triangulation $\mathrm{DT}(S \cup S_0)$ of

$$S \cup S_0 = \{p_{-1}, p_{-2}, \ldots, p_{-m}, p_{-1}, p_1, p_2, \ldots, p_n\}.$$

A naive way to recover $\mathrm{DT}(S)$ would be to remove from $\mathrm{DT}(S \cup S_0)$ every triangle that contains oints of S_0. In reality, the remaining triangles do dot always form the $\mathrm{DT}(S)$. On Figure 3.6, triangle $p_k p_j p_l$ should be present in $\mathrm{DT}(S)$. Yet, its circumcircle contains point p_{-1} which does not belong to S.

The easiest way of addressing that problem is simply not to fix it. In many situations, $\mathrm{DT}(S \cup S_0)$ is a valid input for further use. The is the case for mesh generation.

Yet, one may be interrested in building $\mathrm{DT}(S)$. In this case, some modifications to the algorithm have to be made. On Figure 3.6, triangle $p_k p_j p_l$ has its circumcircle that contains p_{-1} and so edge $p_j p_{-1}$ belongs to the Delaunay triangulation. Disappointingly, triangle $p_k p_j p_l$ belongs to $\mathrm{DT}(S)$. Triangle $p_k p_j p_l$ would be a Delaunay triangle if p_{-j} was sufficiently far i.e. out of the circumcircle of $p_k p_j p_l$. In this specific case, increasing slightly the size of the super-triangles would do the job but it is not clear how to chose *a priori* the size of the super-triangles that would ensure that any triangle that has an edge on the convex hull has its circumcircle that do not contain any of the p_{-j}'s. Some triangles may be arbitrary flat and their circumcircle arbitrary large. it is indeed impossible to decide *a priory* the right size of the super-triangles.

The easiest solution to recover $\mathrm{DT}(S)$ is to start from $\mathrm{DT}(S \cup S_0)$ and to apply edge flips in a specific fashion. Assume here that every point p_{-j} is far enough so that it does not fall into any circumcircle. Consider every edge $p_{-i} p_j$ that connectes a point of negative index to a point of positive index. Edge $p_{-i} p_j$ is flippable if it intersects $p_k p_l$. If $p_{-i} p_j$ is flippable, then it should be flipped because triangle $p_k p_j p_l$'s circumcircle does not contain p_{-1}. The principle is to replace an edge of inifinite length with points of finite lenhgth. Note that an edge like $p_i p_{-2}$ should not be flipped because it would create another edge of infinite length. Applying flips successively in that fashion, allows to recover $\mathrm{DT}(S)$.

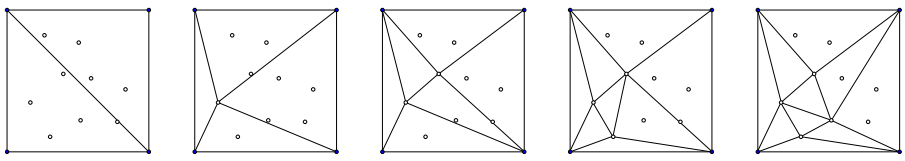


Figure 3.5: A set of 9 points and the two "super-triangles" that contains them all (left). Next Figures show the state of the triangulation after the insertion of 1, 2, 3 and 4 points.

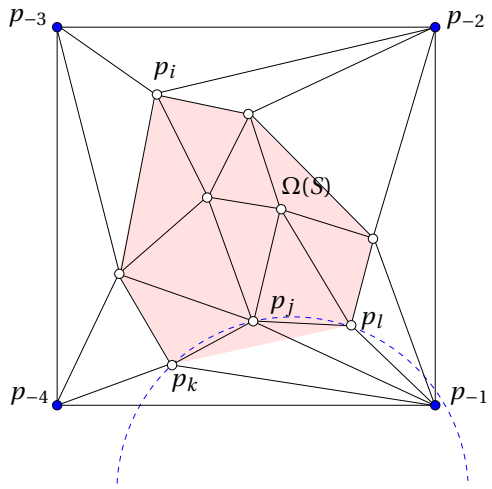


Figure 3.6: Left Figure shows the final triangulation $\mathrm{DT}(S_0 \cup S)$. The convex hull $\Omega(S)$ is shaded and triangles $\mathrm{DT}(S_0 \cup S)$ do not cover it: $\mathrm{DT}(S) \notin \mathrm{DT}(S_0 \cup S)$.

3.2.2 What if $p_{n+1} \notin \Omega(S_n)$?

TODO: explain gift wrapping stuff.

3.3 A robust implementation in $\mathcal{O}(n \log n)$ complexity

Algorithm 1 describes a basic implementation of the Bowyer-Watson algorithm. It has actually two major flaws.

Algorithm 1 is slow: it has a $\mathcal{O}(n^2)$ complexity: at each iteration i, every triangles of DT_{i-1} j is asked if p_i is inside its circumcircle. There is about $2i$ triangles at iteration i which leads to a $\mathcal{O}(n^2)$ complexity. Centers of circumcircles could be computed in advance and stored in the datastructure in order to accelerate the process. Nevertheless, this approach remains quadratic in complexity.

Algorithm 1 suffers from another more subtle flaw that is essentially due to round-off errors. We have assumed that points were in general positions so that no quadruplets of points are cocircular. This hypothesis is indeed not verified in practice: there are numerous applications where circles are involved and where way more than 4 points sit on the same circle. Algorithm 1 could be in trouble because some point may neither be inside nor outside a circulcircle. One solution is to randomly perturbate the position of the points in order to enforce them to be in general position. Here, the question is what is the smallest perturbation that ensures the triangulation process terminates with success.

The first issue can be solved choosing some adequate datastructures and algorithms. The second issue can be addressed by designing essentially two robust pred-

Algorithm 1: Bowyer and Watson's algorithm that creates DT(S)

input : A set of $n+4$ points $S = \{p_{-4}, p_{-3}, p_{-2}, p_{-1}, p_1, \ldots, p_n\} \subset \mathbb{R}^2$
output: The Delaunay triangulation DT(S)

initialize a triangulation data structure DT_0 with 2 super-triangles p_{-1}, p_{-2}, p_{-3} and p_{-2}, p_{-1}, p_{-4}.;

for $i = 1$ **to** n **do**
  for $j = 1$ **to** *size*(DT_{i-1}) **do**
    τ_j is the j^{th} triangle of DT_{i-1};
    if τ_j *'s circumcircle contains* p_i **then**
      Add τ_j to Delaunay cavity $\mathscr{C}(\mathrm{DT}_{i-1}, p_i)$;
      Remove τ_j from DT_{i-1};
    for $j = 1$ **to** *size*($\mathscr{C}(DT_{i-1}, p_i)$) **do**
      τ_j is the j^{th} triangle of $\mathscr{C}(\mathrm{DT}_{i-1}, p_i)$.;
      for $k = 1$ **to** 3 **do**
        e_{jk} is the k^{th} edge of the τ_j. ;
        if e_{jk} *is not shared by any other triangles of* $\mathscr{C}(DT_{i-1}, p_i)$ **then**
          Add a new triangle e_{jk}, p_i into DT_{i-1};

icates.

3.3.1 Robust predicates

Consider three points $a(x_a, y_a)$, $b(x_b, y_b)$ and $c(x_c, y_c)$. The *orientation test*

$$\mathcal{O}_?(a, b, c)$$

determines whether a lies to the left of, to the right of, or on the line L_{bc} defined by points b and c. The orientation test $\mathcal{O}_?$ consist in computing the orientation of triangle abc i.e. to compute:

$$\begin{aligned} \mathcal{O}_?(a, b, c) &= \operatorname{sign} \mathcal{O}(a, b, c) \\ &= \operatorname{sign} \begin{vmatrix} 1 & 1 & 1 \\ x_a & x_b & x_c \\ y_a & y_b & y_c \end{vmatrix} \end{aligned} \tag{3.2}$$

The orientation test is useful in many situations. First, it allows to compute the orientation of a triangle, which is useful by itself. It also allows to verify if two edges ab and cd intersect, which is the case if

$$\mathcal{O}(a, b, c) \times \mathcal{O}(a, b, d) < 0 \text{ and } \mathcal{O}(c, d, a) \times \mathcal{O}(c, d, b) < 0.$$

The computation of the orientation test $\mathcal{O}_?(a, b, c)$ looks very simple: it consist in computing the determinant of a 3×3 matrix. Some interresting issues appear yet

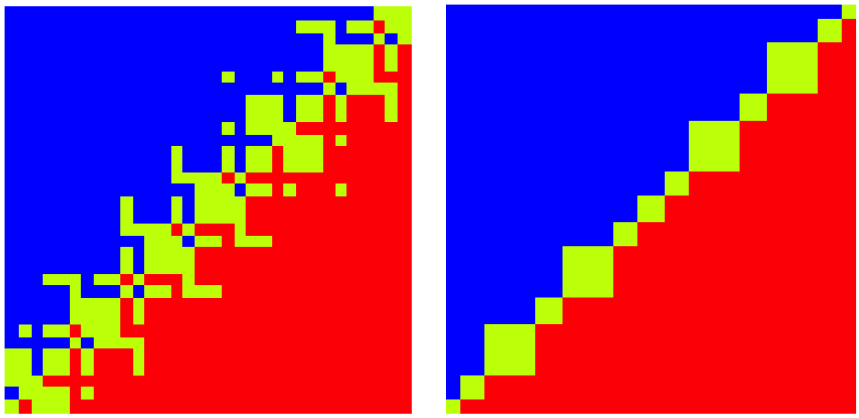


Figure 3.7: Strange behavior of the orientation test. Left figure shows $\mathcal{O}_?(a,b,c)$ for $b(12,12)$, $c(24,24)$, and $a(1/2+i\varepsilon),1/2+j\varepsilon)$, $\varepsilon = 2^{-53}$ and $0 \leq i,j \leq 2^8$. Right figure shows $\mathcal{O}_?(c,b,a)$. .

when a is sufficiently close to line bc. As an example [**?**], consider

$$b(12,12),\ \ c(24,24),\ \text{ and }\ a(1/2+i\varepsilon),1/2+j\varepsilon),$$

$\varepsilon = 2^{-53}$ and $0 \leq i,j \leq 2^8$. Note that 2^{-53} is the significand precision of a double-precision.

In Figure 3.7 the 256^2 results of the $\mathcal{O}_?$ were reported on a $2D$ graph. Green dots are for $\mathcal{O}_?(a,b,c) = -1$, red dots are for $\mathcal{O}_?(a,b,c) = 1$ and yellow dots are for $\mathcal{O}_?(a,b,c) = 0$. We should only see yellow dots only on the diagonal of the square. This is obviously not the case: the orientation test behaves randomly when points are close to be aligned. The result of $\mathcal{O}_?$ is wrong even for points that are at 20 times the significand precision away from the diagonal. Even worse: results obtained with $\mathcal{O}_?(c,b,a)$ should be the same as the ones for $\mathcal{O}_?(a,b,c)$. The second graph proves that this is far from being true. This strange behavior is due to roundoffs. A robust way of computing the orientation test requires more precise (and more expensive) floating-point arithmetics. It is of course too expensive to compute every predicate in an exact fashion. Static filtering consist in assuming that $\mathcal{O}_?$ gives the right answer if

$$|\mathcal{O}(a,b,c,d)| > \epsilon \times (\max(x_a,y_a,x_b,y_b,x_c,y_c))^2.$$

In [**?**], authors show that $\epsilon = 10^{-15}$ is considered as secure for the 2D orientation test. This value is verified experimentally on Figure 3.7. If $|\mathcal{O}_?|$ is too small, arbitrary precision arithmetic on floating-point numbers is applied (we use here the GNU Multiple Precision Floating-Point Reliable Library [**?**] that allows to to choose the precision of the computations). Double-precision floating point numbers have a precision of 53 bits (16 significant digits). We have implemented $\mathcal{O}_?$ using 200 bits i.e. with about 60 significand digits! High precision floating point arithmetics coupled with

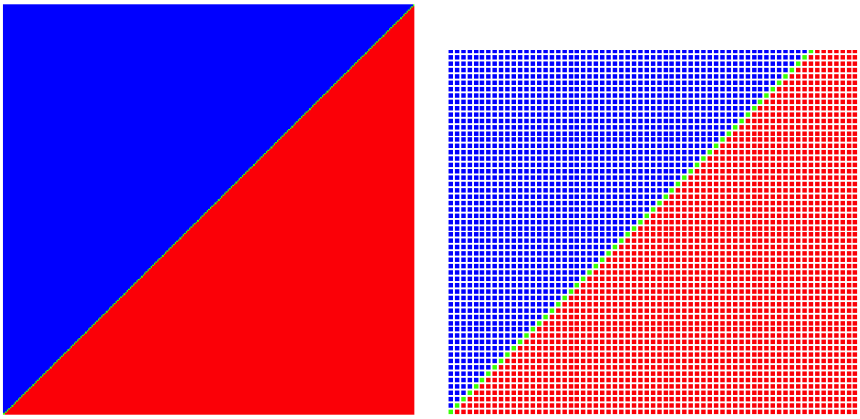


Figure 3.8: $\mathcal{O}_?(a,b,c)$ using a robust predicate. Right Figure is a zoom.

a static filter ($\epsilon = 10^{-15}$) allow to produce the excpected results (see Figure 3.8). The strange behavior of the orientation test of Figure 3.7 has completely disappeared. Only points on the diagonal of the square are considered to be on line bc.

The *incircle test* is a critical piece in the implementation of Delaunay triangulations. Consider three points $a(x_a, y_a)$, $b(x_b, y_b)$, $c(x_c, y_c)$ and $d(x_d, y_d)$. The incircle test computes

$$\begin{aligned} \mathcal{C}_?(a,b,c,d) &= \operatorname{sign} \mathcal{C}(a,b,c,d) \\ &= \operatorname{sign} \begin{vmatrix} 1 & 1 & 1 & 1 \\ x_a & x_b & x_c & x_d \\ y_a & y_b & y_c & y_d \\ (x_a^2+y_a^2) & (x_b^2+y_b^2) & (x_c^2+y_c^2) & (x_d^2+y_d^2) \end{vmatrix}. \end{aligned} \tag{3.3}$$

Equation (3.3) determines whether d lies inside the circle defined by points a, b, and c. The incircle test has the same robustness issues as the orientation test. Bowyer-Watson algorithm 1 that uses a non robust circle test can possibly produce invalid meshes.

A strategy that couples static filtering ($\epsilon = 10^{-11}$) and high precision floating point arithmetics result in a robust incircle test.

In the Delaunay kernel, predicate $\mathcal{C}_?$ should predict wether a point d is inside or inside the circle defined by points a, b, and c. If d lies exactly on the circle, the points are not in general position and the Delaunay triangulation is not unique. This situation should never happen. The most common way to avoid that situation is to perturbate the set of points. A naive approach consist in slightly modifying the position of the initial point set, doing the triangulation and then moving back the points to their original positions. This perturbation method is not robust: there is no guarantee that triangles remain valid after repositioning the nodes.

In [**?**, **?**], authors propose to perturbate the points in a symbolic fashion. We first acknowledge that predicate (3.3) in $\mathbb{R}^2$ can indeed be seen as an orientation predi-

cate $\mathcal{O}_?$ in $\mathbb{R}^3$ where the third coordinate of a is $r_a^2 = x_a^2 + y_a^2$. The idea of a symbolic perturbation is to perturbate this third coordinate by a factor ϵ_a. The perturbate predicates writes:

$$\mathcal{C}_?^\epsilon(a,b,c,d) = \text{sign} \begin{vmatrix} 1 & 1 & 1 & 1 \\ x_a & x_b & x_c & x_d \\ y_a & y_b & y_c & y_d \\ r_a^2+\epsilon_a & r_b^2+\epsilon_b & r_c^2+\epsilon_c & r_d^2+\epsilon_d \end{vmatrix}. \tag{3.4}$$

Expending that determinant with respect to its last row, we get

$$\mathcal{C}^\epsilon(a,b,c,d) = \mathcal{C}(a,b,c,d) + \epsilon_a\mathcal{O}(b,c,d) - \epsilon_b\mathcal{O}(a,c,d) + \epsilon_c\mathcal{O}(a,b,d) - \epsilon_d\mathcal{O}(a,b,c).$$

This perturbation test is only used when $\mathcal{C}(a,b,c,d) = 0$. This leads to

$$\mathcal{C}^\epsilon(a,b,c,d) = \epsilon_a\mathcal{O}(b,c,d) - \epsilon_b\mathcal{O}(a,c,d) + \epsilon_c\mathcal{O}(a,b,d) - \epsilon_d\mathcal{O}(a,b,c). \tag{3.5}$$

The four terms of (3.5) are examined in order of increasing ϵ_i. When a non zero value is found, it determines the sign of $\mathcal{C}_?^\epsilon$. A simple choice for the perturbation is to choose $\epsilon_i = i$ i.e. the index of the point is chosen as pertubation. One can also use the lexicographic ordering. Note finally that the last term of $\mathcal{C}^\epsilon$ is non zero because $\mathcal{O}(a,b,c)$ is the area of triangle abc which is positive. Using $\epsilon_i = n - i$ assumes that the last point that is inserted in the triangulation is the most perturbed. In this case, $\mathcal{C}(a,b,c,d) = 0$ implies $\mathcal{C}_?^\epsilon(a,b,c,d) = -1$ and the point d is considered exterior.

3.3.2 Choice of a datastructure

Figure 1.5 illustrate equation (1.6). The average number of adjacencies per entity in the triangulation is know in advance. Yet, as it is seen on Figure (1.5), this number varies from one vertex to another. This number may also change locally: an edge flip removes one triangle of the adjacency of the two vertices of the edge that is flipped and adds one triangle tho the adjacency of the two vertices of the new edge (see Figure (1.5)). The number of upward adjacencies of a given vertex may change which implies that datastructures that would keep track of such adjacencies should be of variable size.

When the size of data may vary, memory allocation has to be used, which implies indirect memory access and extra data storage. Datastructures of fixed size are always preferred. Yet, some kind of upward adjacencies should exist in the datastructure in order to accesss neighborhood of a mesh entity withoud traversing the whole triangulation.

There exist one type of upward adjacency that is of fixed size: there is either 1 or 2 adjacent triangle to an edge. Note that this hypothesis implies that the triangulation is *manifold* i.e. each edge is shared by no more than 2 faces. This is a common assumption for many algorithms and we will assume the triangulation to be manifold for now.

At this point, we'd like to choose how we will represent our triangulation on a computer. The problem of choosing a datastructure is crucial. A good datastructure

is has a low memory footprint but allows to compute local adjacencies in constant time.

Technically, an adjacency is implemented as a pointer (the address of the adjacent entity). Moreover, if a given entity (point, edge or face) is explicitely represented in a datastructure, it also requires one pointer (the address of the entity).

It is interresting to cout the total amount of pointers N_p that are required in a given mesh representation (i.e. for a given datastructure). In what follows, we assume that $n \gg n_h$ and $n \gg 1$ which implies that $n_e \simeq 3n$, $n_f \simeq 2n$, $n_{ve} \simeq 6$, $n_{vf} \simeq 6$ and $n_{ef} \simeq 2$.

A naive choice could be to store all entities and all their adjacencies. This datastructure is said to be full for obvious reasons. The number of pointers that is required is

$$N_p = n(1 + n_{ve} + n_{vf}) + n_e(2 + 1 + n_{ef}) + n_f(3 + 3 + 1) \simeq 42n.$$

The full datastructure is clearly overkill in term of memory. Moreover, using such a datatrsucture in algorithms requires complicated updates which makes that approach totally ininterresting.

Another choice is the bidirectional datastructure [**?**]. In this datastructure, vertices keep track of their adjacent edges, edges know about their adjacent vertices and faces and face know about their edges. This datastructure is complete in the sense that all entities are represented explicitely and that any adjacency information can be recovered using local searches. The number of pointers that is required is

$$N_p = n(1 + n_{ve}) + n_e(2 + 1 + n_{ef}) + n_f(3 + 1) \simeq 30n.$$

This is again a very heavy datastructure that requires complex updates while used in algorithms.

In many cases, the only information that is required in a representation is the list of vertices of a triangle. This is the case in most of the finite element formulations or to draw the mesh. Here, the number of pointers that is required is

$$N_p = n + n_f(3 + 1) \simeq 9n.$$

This is clearly the minimum amount of information possible. No upward adjacenci is available here so that it is impossible to devise efficient meshing algorithms with such a datastructure.

Most popular data structures for storing adjacency information of polygonal meshes are edge-based. Winged-edge [**?**] and half-edge [**?**] datastructures apply to manifold meshes while Winged-edge and half-edge data structure uses edges to keep track almost everything. In a winged-edge datastructure, each edge stores 8 pointers to neighboring edges, faces and points. Faces and points store one pointer, so that the number of pointers that is required is

$$N_p = n(1 + 1) + n_e(1 + 8) + n_f(1 + 1) = 33n.$$

The advantage of such a datatrsucture is that it is easy to update when local operations are performed. Yet, it is quite heavy.

```
struct Vertex {
  double x,y,z;
  Vertex (double X, double Y, double Z) :
    x(X), y(Y), z(Z) {}
};
```

Listing 3.1: Vertex Datastructure

Those datastructures are suboptimal for algorithms like the Delaunay triangulation. Representing edges explicitely is not mandatory here and edges are the entities that are the most numerous in a triangulation. In this text, we use a datastructure that is face-based: each triangle knowns about its 3 vertices and its 3 neighboring triangles. Each vertex knows about its coordinates. That's pretty much all. The number of pointers that is required is

$$N_p = n + n_f(1+6) = 15n.$$

This datastructure is way lighter than edge-based ones. With 8 bytes pointers, the memory footprint of a mesh with $n = 10^6$ is 120 Mb. Another advantage is that it can be extended in 3D, which is not the case for edge-based datastructures.

Vertex datastructure

The vertex datastructure is quite simple: a vertex knows about its coordinates (see Listings 3.1.

Edge datastructure

Even though we do not maintain edges of the triangulation in our algorithms, it is sometimes necessary to build edges for a subset of triangles of the triangulation. The edges that we consider are nort oriented: they are equal if they connect the same two vertices. The datastructure shown in Listings 3.2 allows to construct an edge with two vertices and to compare two edges (edges are compared comaring their vertex pointers in a lexicographic manner).

Face datastructure

The triangles are maintained in the triangulation. Each triangle maintains its three vertices and its three neighbors. We assume that neighbor `F[k]` is the triangle that is on the other side of edge with vertices `V[k]` and `V[(k+1)%3]`. We also assume that we have a function `inCircle` that predicts if vertex `V` is inside the circumcircle of the `Face` and a function `centroid` that computes the centroid of the face. The Face datastructure is shown in Listings 3.3.

```
struct Edge {
   Vertex *vmin,*vmax;
   Edge (Vertex *v1, Vertex *v2)
      vmin = std::min (v1,v2);
      vmax = std::max (v1,v2);
   }
   bool operator < (const Edge &other) const {
      if (vmin < other.vmin) return true;
      if (vmin > other.vmin) return false;
      if (vmax < other.vmax) return true;
      return false;
   }
};
```

Listing 3.2: Edge Datastructure

```
struct Face {
    Face    *F[3];
    Vertex  *V[3];
    bool deleted;
    Face (Vertex *v0, Vertex *v1, Vertex *v2){
       V[0] = v0; V[1] = v1; V[2] = v2;
       F[0] = F[1] = F[2] = NULL;
       deleted = false;
    }
    Edge getEdge (int k) {
       return Edge (V[k],V[(k+1)%3]);
    }
    bool inCircle (Vertex *c);
    Vertex centroid ();
};
```

Listing 3.3: Face Datastructure

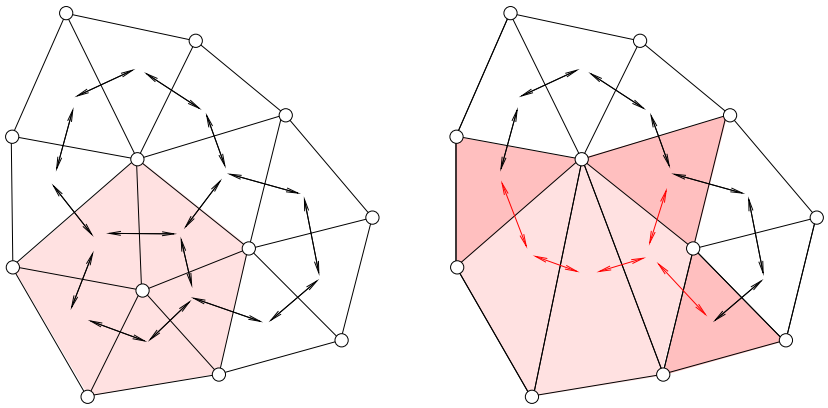


Figure 3.9: A cavity (left figure in light pink) is removed from the meh. It is remeshed (left figure in light pink). Adjacencies (double arrows) are updated (red double arrows) for all new triangles (light pink) as well as for all neighboring triangles of the cavity (dark pink).

3.3.3 Algorithms

A local mesh modification works as follows. A cavity of triangles is removed from the mesh (see Figure 3.9). The cavity is remeshed and mesh datastructures are updated in order to take into account the modification. More specifically, each new `Face` of the remeshed cavity has to be connected to its neighboring faces. Those neighboring faces may be new as well or may be neighboring triangles of the cavity.

Algorithm depicted in Listings 3.4 is the building block of all other algorithms that are performing local mesh modifications: `computeAdjacencies` computes adjacencies of a list of N triangles. It has a $\mathcal{O}(N \log N)$ complexity (one search/insert on a `std::map` per triangle). We use here some associative containers from the standard template library.

Another important building block in our implementation is the computation of the Delaunay cavity. We assume here that one initial triangle t has been found that has its circumcircle containing a given vertex. Algorithm in Listings 3.5 allows to compute the Delaunay cavity using a depth-first search technique. The theory ensures that the Delaunay cavity is simply connected: triangles that form the Delaunay cavity are neighbors of t, neighbors of the neighbors of t and so on. The neighborhood of t is searched recursively until a triangle is found that is valid i.e. that does not violate the empty circumcircle property. Triangles that have been checked are marked as deleted to avoid infinite loops. Two other outputs are computed that will serve us in constructing the Delaunay ball and in computing adjacencies. The set of edges that form the boundary of the cavity is also computed. The corresponding valid triangles that are on the other side of the boundary of the Delaunay cavity are also computed.

```
void computeAdjacencies (std::vector<Face*> &cavity) {
  std::map < Edge , std::pair < int , Face* > >edgeToFace;
  for (int iFace=0 ; iFace < cavity.size() ; iFace++ ){
     for (int iEdge=0 ; iEdge < 3 ; iEdge++){
        Edge edge = cavity[iFace]->getEdge(iEdge);
        std::map < Edge , std::pair < int , Face* > >::iterator it =
            edgeToFace.find(edge);
        if (it == edgeToFace.end()){
           // edge has not yet been touched, so create an entry
           edgeToFace.insert(std::make_pair (edge,
              std::make_pair(iEdge, cavity[iFace])));
         }
         else{
          // Connect the two neighboring triangles
           cavity[iFace]->F[iEdge] = it->second.second;
           it->second.second->F[it->second.first] =  cavity[iFace];
          // Erase edge from the map
           edgeToFace.erase(it);
         }
      }
   }
}
```

Listing 3.4: An algorithm for connecting triangles in a cavity

```
void delaunayCavity (Face *f, Vertex *v, std::vector<Face*> &cavity,
                        std::vector<Edge> &bnd, std::vector<Face*>
                            &otherSide){
  if (f->deleted)return;
  f->deleted = true; // Mark the triangle
  cavity.push_back(f);
  for (int iNeigh=0; iNeigh<3 ; iNeigh++){
    if (f->F[iNeigh] == NULL){
      bnd.push_back(f->getEdge(iNeigh));
    }
    else if (!f->F[iNeigh]->inCircle(v)){
      bnd.push_back(f->getEdge(iNeigh));
      if (!f->F[iNeigh]->deleted){
          otherSide.push_back(f->F[iNeigh]);
          f->F[iNeigh]->deleted = true;
      }
    }
    else delaunayCavity (f->F[iNeigh], v, cavity,bnd,otherSide);
  }
}
```

Listing 3.5: An algorithm for computing the Delaunay cavity

```
Face* lineSearch (Face *f, Vertex *v){
    while(1){
      if (f == NULL) return NULL; // we should NEVER return here
      if (f->inCircle(v)) return f;
      Vertex c = f->centroid();
      for (int iNeigh=0; iNeigh<3 ; iNeigh++){
        Edge e = f->getEdge (iNeigh);
        if (orientationTest (&c,v,e.vmin) *
            orientationTest (&c,v,e.vmax) < 0 &&
            orientationTest (e.vmin, e.vmax, &c) *
            orientationTest (e.vmin, e.vmax, v) < 0) {
          f = f->F[iNeigh];
          break;
       }
    }
 }
```

Listing 3.6: An algorithm that finds a invalid triangle

Computing the Delaunay cavity requires a seed triangle i.e. a triangle t of the triangulation that is invalid. The last bit algorithm that is provided here allows to perform a search in a mesh along a given direction and find the desired triangle. Triangulations we are dealing with cover the convex hull $\Omega(S)$ of the set of points S. So, if c is the centroid of a given triangle t and if $p \in S$ is a target point, line cp is entirely inside the triangulation and it is possible to find a path of triangles that connect t to the triangle t' that contains p. Algorithm in Listings 3.6 strats from a given triangle and traverses the mesh until an invalid triangle is found. It assumes that a robust orientation test $\mathcal{O}_?$ function is available. Assume a triangulation with n_f triangles, the complexity of algorithm `lineSearch` is at most linear. Asymptotically, it is not absurd to guess that only $\mathcal{O}(\sqrt{n_h})$ triangles will be touched by `lineSearch` which reduce its complexity in practice.

Algorithm in Listings 3.7 is a C++ version of 1. It has clearly a worst complexity of $\mathcal{O}(n^2)$ but could possibly behave better i.e. like $\mathcal{O}(n^{3/2})$. It starts with an initial triangulation made of some super triangles that cover the convex hull of S and inserts the points incrementally.

The code that is provided here is actually working as is. We have used it to compue Delaunay triangulations of random points. The following table presents results of the algorithm for a set of n random points in the plane that have been inserted in a random order.

In Table 3.1, N_{search} is the average number of serarches that have been performed in `lineSearch` and N_{cavity} is the average size of the Delaunay cavity. Even though this implementation may not be optimal, it shows the basic features of the algorithm. First, the average cavity size is asymptotically optimal: a cavity of size 4 produce 6 new triangles adjacent to a vertex which is what the theory predicts. Then the number of walks that the `lineSearch` algorithm increases approximatively like

```
void delaunayTrgl (std::vector<Vertex*> &S, std::vector<Face*> &T){
  for (int iP=0 ; iP < S.size() ; iP++ ){
    Face * f = lineSearch ( T[0] , S[iP]);
    std::vector<Face*> cavity;
    std::vector<Edge> bnd;
    std::vector<Face*> otherSide;
    delaunayCavity (f, S[iP], cavity, bnd, otherSide);
    if(bnd.size() != cavity.size() + 2) throw;
    for (int i=0; i<cavity.size(); i++) {
      // reuse memory slots of invalid elements
      cavity[i]->deleted = false;
      cavity[i]->F[0] = cavity[i]->F[1] = cavity[i]->F[2]  = NULL;
      cavity[i]->V[0] = bnd[i].V[0];
      cavity[i]->V[1] = bnd[i].V[1];
      cavity[i]->V[2] = S[iP];
    }
    unsigned int cSize = cavity.size();
    for (int i=cSize; i<cSize+2; i++) {
      Face *newf = new Face (bnd[i].V[0],bnd[i].V[1],S[iP]);
      T.push_back[newf];
      cavity.push_back(newf);
    }
    for (int i=0;i<otherSide.size();i++)
      if (otherSide[i])cavity.push_back(otherSide[i]);
    computeAdjacencies (cavity);
  }
}
```

Listing 3.7: An algorithm for computing the Delaunay triangulation

n	10^3	10^4	10^5	10^6
N_{search}	19.8	56.8	161	503
N_{cavity}	3.85	3.97	3.99	3.99
t(sec)	0.012	0.198	4.85	172

Table 3.1: Results of the `delaunayTrgl` algorithm applied to random points.

the square root of n: $56.8 \times \sqrt{10} = 179.6$ which is close to 161. The complexity of the algorithm written as is is close to $\mathcal{O}(n^{3/2})$. For large meshes, the $\mathcal{O}_?$ predicate takes about 50% of the CPU time so that the most significant part if the time is spend in searching for an initial triangle.

The bottleneck of the Delaunay triangulation as it is written in `delaunayTrgl` is the increasing effort that has to be done at each point insertion to find a triangle seed for building the Delaunay cavity.

Assume now that we are able to sort the set of points S in such a way that two successive points in the list would be close to each other. In Algorithm `delaunayTrgl`, we take as initial guess the first triangle of the list and search into the domain. We could chage that by choosing one of the triangles of the cavity that is associated to the vertex that was inserted previously in the list.

3.3.4 Hilbert Curves

A curve $x(t)$ is defined as the mapping

$$x(t)\ ,\ t \in [0,1] \rightarrow x \in \mathbb{R}^3.$$

Curves are perceived as one dimensional objects. Yet, it can be shown that a continuous curve can pass through every point of a unit square. The Hilbert space filling $\mathscr{H}(t)$ curve is a one dimensional curve which visits every point within a two dimensional space. It may be thought of as the limit

$$\mathscr{H}(t) = \lim_{k \rightarrow \infty} \mathscr{H}_k(t)$$

of a sequence of curves $\mathscr{H}_k$ (see Figure 3.10).

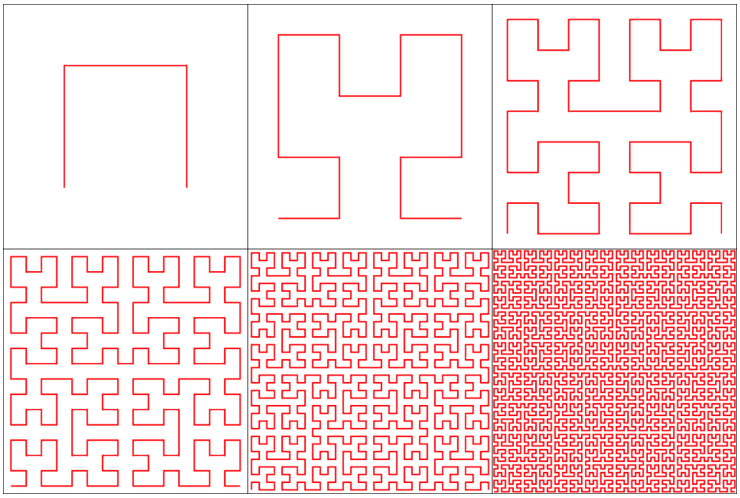


Figure 3.10: Sequense of Hilbert curves $\mathscr{H}_k$.

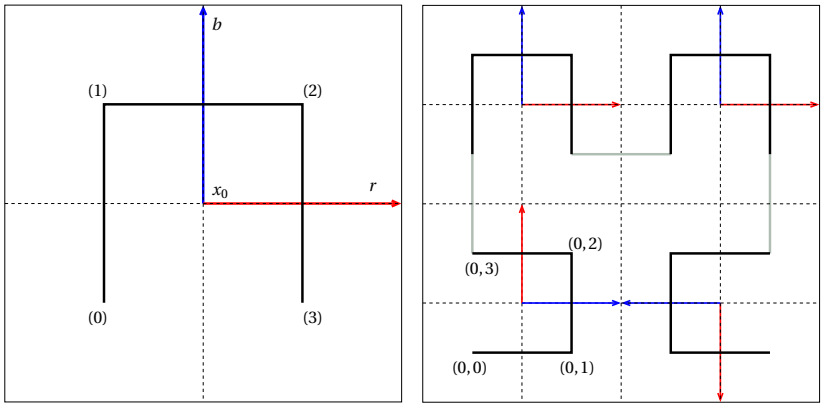


Figure 3.11: Curves $\mathscr{H}_1$ and $\mathscr{H}_2$.

Curves $\mathscr{H}_1$ and $\mathscr{H}_2$ are depicted on Figure 3.11. There are lots of references that show how to actually draw Hilbert curves: this is a distraction from the essential property of the curve, and its importance to mesh generation.

Hilbert curves provide an ordering for points on a plane. Forget about how to connect adjacent sub-curves, and instead focus on how we can recursively enumerate the quadrants.

A local frame is associated to each quadrant: it consist in its center x_0 two orthogonal vectors b and r (see Figure 3.11). At the root level, enumerating the points is simple: proceed around the four quadrants, numbering them

$$(0) = x_0 - \frac{b+r}{2} \quad (1) = x_0 + \frac{b-r}{2} \quad (2) = x_0 + \frac{b+r}{2} \quad (3) = x_0 - \frac{b-r}{2}.$$

We want to determine the order we visit the sub-quadrants while maintaining the overall adjacency property. Examination reveals that each of the sub-quadrants curves is a simple transformation of the original pattern. Figure 3.11 illustrate the first level of that recursion.

Quadrant (0) is itself divided into four quadrants $(0,0)$, $(0,1)$, $(0,2)$ and $(0,3)$. Its center is simply set to (0) and two vectors b and r are changed as

$$b \leftarrow r/2 \text{ and } r \leftarrow b/2.$$

For quadrant $(0,1)$ and $(0,2)$ we have

$$b \leftarrow b/2 \text{ and } r \leftarrow r/2.$$

and finally for quadrant $(0,3)$:

$$b \leftarrow -r/2 \text{ and } r \leftarrow -b/2.$$

```
void HilbertCoord( double x,    double y,    double x0,    double y0,
                   double xRed, double yRed, double xBlue, double yBlue,
                   int d,       int bits[] ){

  for (int i = 0; i <d; i++) {
    double coordRed  =  (x-x0) * xRed  + (y-y0) * yRed;
    double coordBlue =  (x-x0) * xBlue +  (y-y0) * yBlue;
    xRed/=2; yRed/=2;       xBlue/=2; yBlue/=2;
    if (coordRed <= 0 && coordBlue <= 0) { // quadrant 0
      x0 -= (xBlue+xRed);       y0 -= (yBlue+yRed);
      swap (xRed,xBlue);        swap (yRed,yBlue);
      bits[i] = 0;
   }
    else if (coordRed <= 0 && coordBlue >= 0) { // quadrant 1
      x0 += (xBlue-xRed);       y0 += (yBlue-yRed);
      bits[i] = 1;
    }
    else if (coordRed >= 0 && coordBlue >= 0) { // quadrant 2
     x0 += (xBlue+xRed);       y0 += (yBlue+yRed);
     bits[i] = 2;
    }
    else if (coordRed >= 0 && coordBlue <= 0) { // quadrant 3
      x0 += (-xBlue+xRed);     y0 += (-yBlue+yRed);
      swap (xRed,xBlue);       swap (yRed,yBlue);
      xBlue = -xBlue;          yBlue = -yBlue;
      xRed = -xRed;            yRed = -yRed;
      bits[i] = 3;
    }
  }
}
```

Listing 3.8: An algorithm for computing Hilbert coordinates

creates 4 sub quadrants. If we consider a maximal recursion depth of d, each of the final subquadrants will be assigned to a set of d "coordinates" i.e. $(k_0, k_1, \ldots, k_d)$, k_j being 0,1,2 or 3.

Algorithm in Listings 3.8 compute the Hilbert coordinates of a given point x, y, starting from an initial quadrant define by its center x_0, y_0 and two orthogonal directions.

Each point x of $\mathbb{R}^2$ has its coordinates on the Hilbert curve. Sorting a point set with respect to Hilbert coordinates allow to ensure that two successive points of the set are close to each other. In the context of the Bowyer-Watson algorithm, this kind of data locality could potentially decrease the number of local searches N_{search} that were required to find the next invalid triangle.

Algorithm 3.8 was used to sort sets of 1000 and 10000 points. The results are presented on Figure 3.12. On the Figure, two successive points in the sorted list are linked with a line.

n	10^3	10^4	10^5	10^6
N_{search}	2.34	2.46	2.50	2.50
N_{cavity}	4.06	4.13	4.16	4.17
t(sec)	0.0097	0.090	0.92	9.2

Table 3.2: Results of the `delaunayTrgl` algorithm applied to random points. Points were initially sorted through using a Hilbert sort.

The main cost of sorting points is on the sorting algorithm itself and not on the computation of the Hilbert curve coordinates: sorting over a million points takes less than a second on a standard laptop. Table 3.2 present timings and statistics for the same point sets as in table 3.1, but while having sorted the points S using the Hilbert curve. The number of serarches is not increasing anymore with the size of the set. This is important: the complexity of the Delaunay triangulation algorithm now is linear in time. Of course, sorting points has a $n \log n$ complexity so that the overall process is in $n \log n$ as well. Yet, the relative cost of sorting the points is negligible with respect to the cost of the triangulation itself.

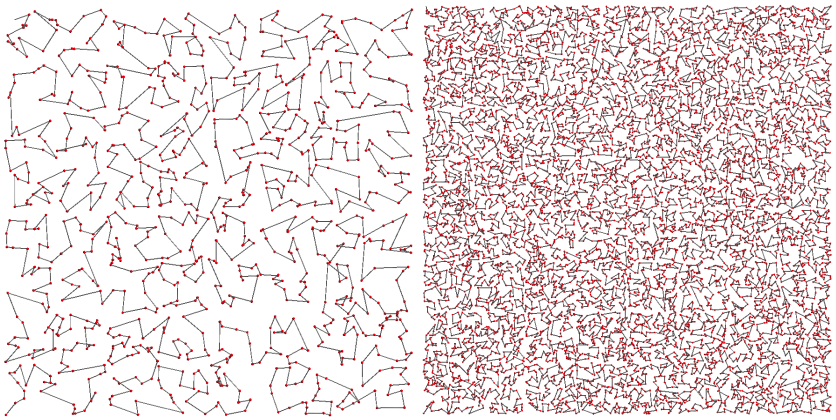


Figure 3.12: Hilbert sort of sets of 1000 and 10000 random points.

Explain brio : The trick is to organize the point set in random buckets of increasing sizes, Hilbert sort being used only inside a bucket. I observe that this is useless in my implementation that do not really care a lot of memory allocation optimization strategies.

3.3.5 Edge flip

TODO: rite the edge flip algorithm and write the algorithm that recovers the Delaunay triangulation DT(S) starting from DT($S_0 \cup S$).

CHAPTER 4

Finite Element Mesh generation in the plane

Up to now, we have considered the problem of triangulating a given point set S which output is a set of non-overlapping triangles that cover the convex hull $\Omega(S)$.

The problem of planar mesh generation is different. Meshing takes as input a domain $G \subset \mathbb{R}^2$ that has to be triangulated. 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 entities together with adjacencies. Model vertices and model edges of G from a boundary representation of G. Each model edge is topologically oriented: it has a starting and a ending model vertex. Model faces of G are bounded by oriented model edges. As an example, Figure 4.4 presents a model that is composed of two model faces, 20 model edges and 25 model vertices. Mode edge E_{13} is bounded by model vertices V_3 and V_2. The problem of planar mesh generation consist in filling the different model faces of G with triangles. This problem is significantly different from the problem of triangulating a set of points:

- the input data is the geometrical model,
- the point set is to be determined.

Model faces are bounded by model edges. In the mesh, triangles are bounded by mesh edges. The input data of a planar mesher is therefore a set of mesh edges. The problem of discretizing the model edges is the problem of one dimensional meshing.

The mesh generation procedure aims at building triangles that have controlled size and shape. For addressing those aims, we have to clarify what is right for an element, both in terms of size and shape.

4.1 Triangle shape or quality measures

It is not simple to talk about mesh quality because there is no clear definition of what is a “good triangle”. Here, we will take the point of view of finite elements: we are going to define triangle quality using arguments of the finite element theory.

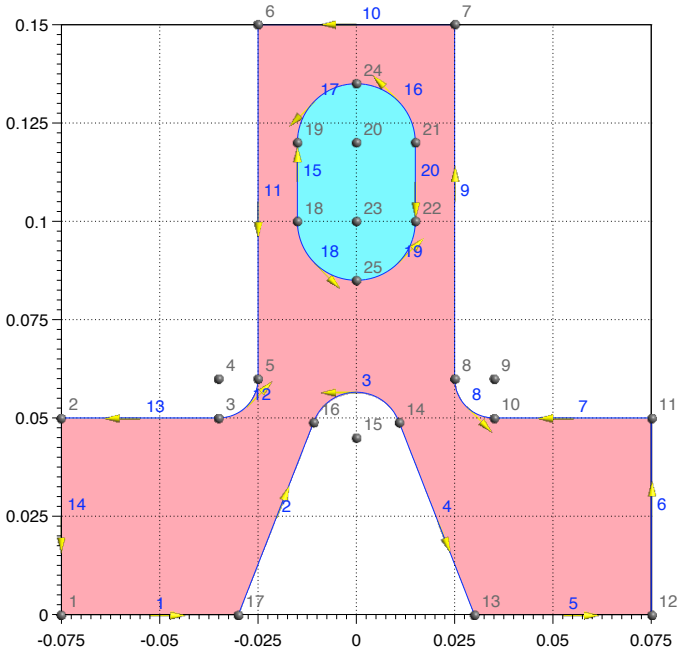


Figure 4.1: A simple 2D model (`conge.geo`).

4.1.1 The famous angle condition

Angle conditions play an important role in the analysis of the finite element method. In this section, we will show the influence of triangle shapes on the quality of a finite element interpolation using (mostly) geometrical arguments.

A finite element is a set with three components: (i) a reference element with a simple shape (the canonical triangle $\hat{T}$ in this case, with vertices $(\xi,\eta) = (0,0)$, $(1,0)$ and $(0,1)$), (ii) a finite-dimensional space of polynomial functions P defined on $\hat{T}$ (the space of shape functions) and (iii) a basis N for P', the dual of P. We note p the polynomial order, $k = (p+1)(p+2)/3$ the dimension of P and $N = \{N_1, \ldots, N_k\}$ is the set of nodal variables that are the actual basis of P'. The set of shape functions $\psi_1, \ldots, \psi_k$ is chosen in such a way that

$$N_i(\psi_j) = \delta_{ij}.$$

It is easy to see that, with such a definition, the ψ_j's form a basis of P.

Shape functions are defined in the reference or canonical triangle Finite elements are meant to interpolate functions. Consider a function $\hat{u}(\xi,\eta) \in H^s(\hat{T})$ where $H^s(\hat{T})$ is the (Sobolev) space of functions defined on $\hat{T}$ that have their sth derivatives

in $L^2(\hat{T})$. Its finite element interpolation on $\hat{T}$ is

$$\hat{U}(\xi,\eta) = \sum_{i=1}^{k} N_i(\hat{u})\psi_i(\xi,\eta).$$

This interpolant $\hat{U}$ differes from $\hat{u}$ and it is interresting to measure the interpolation error by computing some (Sobolev) norm of the difference between $\hat{U}$ and $\hat{u}$:

$$\left\|\hat{u}-\hat{U}\right\|_{s,\hat{T}} = \left(\sum_{|\alpha|\le s} \left\|D^\alpha \hat{u}\right\|^2_{L^2(\hat{T})}\right)^{\frac{1}{2}} \tag{4.1}$$

where α is a multi-index of order $|\alpha| = s$ and

$$D^\alpha \hat{u} = \frac{\partial^{|\alpha|}\hat{u}}{\partial\xi^{\alpha_1}\partial\eta^{\alpha_2}}$$

For example, the L^2 norm is defined as

$$\|\hat{u}\|^2_{L^2(\hat{T})} = \|\hat{u}\|^2_{0,\hat{T}} = \int_{\hat{T}} \hat{u}^2 d\xi d\eta.$$

and the H^1 semi-norm and the H^1 norm are defined as

$$|\hat{u}|^2_{1,\hat{T}} = \int_{\hat{T}} \left(\frac{\partial\hat{u}}{\partial\xi}\right)^2 + \left(\frac{\partial\hat{u}}{\partial\eta}\right)^2 d\xi d\eta \quad , \quad \|\hat{u}\|^2_{1,\hat{T}} = \|\hat{u}\|^2_{0,\hat{T}} + |\hat{u}|^2_{1,\hat{T}}.$$

The Bramble-Hilbert lemma is a classical result of the interpolation theory []. It provides bounds to the interpolation error:

$$\left\|\hat{u}-\hat{U}\right\|_{s,\hat{T}} \le C(s,p)\,|\hat{u}|_{p+1,\hat{T}}, \quad s = 0,1,\ldots,p. \tag{4.2}$$

Lemma (4.2) is quite intuitive: the interpolation error at order p of a smooth function (s times derivable in a weak sense) is dominated by some norm of the $(p+1)$st derivative of the function. Expression (4.2) assumes that $\hat{T}$ has a unit diameter (or size), which is the case for reference triangle $\hat{T}$.

The finite element interpolation can be used on general triangles. For that, we consider a linear mapping of a reference unit right triangle $\hat{T}$ in the $(\xi-\eta)$-plane to an element T in the (x,y)- plane (See Figure **??**). More complex higher order mappings will be studied in further section §**??**. This linear mapping can be explicitely written as

$$x(\xi,\eta) = (1-\xi-\eta)x_1 + \xi x_2 + \eta x_3 \quad , \quad y(\xi,\eta) = (1-\xi-\eta)y_1 + \xi y_2 + \eta y_3. \tag{4.3}$$

The question that is asked here is how does result (4.2) extend when the affine transformation (4.3) is applied to the triangle. Here, we now assume that a function $u(x,y)$ should be interpolated on the mesh. Consider a triangle $T \subset \mathbb{R}^2$ with its three coordinates (x_1,y_1), (x_2,y_2) and (x_3,y_3). We'd like to compute an estimate of

$$|u-U|^2_{1,T} = |w|^2_{1,T} = \int_T \left(\frac{\partial w}{\partial x}\right)^2 + \left(\frac{\partial w}{\partial y}\right)^2 dxdy.$$

We have $\hat{w}(\xi,\eta) = w(x(\xi), y(\eta))$ and

$$\|w\|_{0,T}^2 = \int_T w^2 dxdy = \int_{\hat{T}} \hat{w}^2 \, |\det J| \, d\xi d\eta = |\det J| \int_{\hat{T}} \hat{w}^2 d\xi d\eta = |\det J| \, \|\hat{w}\|_{0,\hat{T}}^2 \tag{4.4}$$

where

$$J = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{bmatrix} = \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix}$$

and $|\det J| = |(x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)| = 2|T|$ is twice the area $|T|$ of T Let us now find a bound on $|\det J|$. Consider that triangle T has with its angles $\alpha \le \beta \le \gamma$ and its three sides [1]

$$a \le b \le c. \tag{4.5}$$

The area of T is equal to $|T| = \frac{1}{2} bc \sin\alpha$ with α the smallest angle of T as defined. Triangle inequality $a + b \ge c$ combined with (4.5) gives $b \ge c/2$. On the other hand, $b \le c$ which gives the following useful upper and lower bounds to $|\det J|$ as:

$$\frac{c^2 \sin\alpha}{2} \le |\det J| \le c^2 \sin\alpha.$$

This, combined with (4.4) leads to the bounds

$$\frac{\|w\|_{0,T}^2}{c^2 \sin\alpha} \le \|\hat{w}\|_{0,\hat{T}}^2 \le \frac{2\|w\|_{0,T}^2}{c^2 \sin\alpha}. \tag{4.6}$$

Transformation of derivatives requires the inverse J^{-1} of J

$$J^{-1} = \begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \xi}{\partial y} \\ \frac{\partial \eta}{\partial x} & \frac{\partial \eta}{\partial y} \end{bmatrix} = \frac{\begin{bmatrix} \frac{\partial y}{\partial \eta} & -\frac{\partial x}{\partial \eta} \\ -\frac{\partial y}{\partial \xi} & \frac{\partial x}{\partial \xi} \end{bmatrix}}{|\det J|} = \frac{\begin{bmatrix} (y_3 - y_1) & -(x_3 - x_1) \\ -(y_2 - y_1) & (x_2 - x_1) \end{bmatrix}}{(x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)}.$$

Let us now define the symmetric metric tensor

$$G = J^{-T} J^{-1} = \begin{bmatrix} \left(\frac{\partial \xi}{\partial x}\right)^2 + \left(\frac{\partial \xi}{\partial y}\right)^2 & \frac{\partial \xi}{\partial x}\frac{\partial \eta}{\partial x} + \frac{\partial \xi}{\partial y}\frac{\partial \eta}{\partial y} \\ \frac{\partial \xi}{\partial x}\frac{\partial \eta}{\partial x} + \frac{\partial \xi}{\partial y}\frac{\partial \eta}{\partial y} & \left(\frac{\partial \eta}{\partial x}\right)^2 + \left(\frac{\partial \eta}{\partial y}\right)^2 \end{bmatrix} = \begin{bmatrix} g_1 & g_2 \\ g_2 & g_3 \end{bmatrix}.$$

Applying Young's inequality $2ab \le a^2 + b^2$ allows to write

$$\begin{aligned} \left(\frac{\partial w}{\partial x}\right)^2 + \left(\frac{\partial w}{\partial y}\right)^2 &= g_1 \left(\frac{\partial \hat{w}}{\partial \xi}\right)^2 + 2g_2 \left(\frac{\partial \hat{w}}{\partial \xi}\frac{\partial \hat{w}}{\partial \eta}\right) + g_3 \left(\frac{\partial \hat{w}}{\partial \eta}\right)^2 \\ &\le g_1 \left(\frac{\partial \hat{w}}{\partial \xi}\right)^2 + g_2 \left(\left(\frac{\partial \hat{w}}{\partial \xi}\right)^2 + \left(\frac{\partial \hat{w}}{\partial \eta}\right)^2\right) + g_3 \left(\frac{\partial \hat{w}}{\partial \eta}\right)^2 \\ &\le \max(|g_1 + g_2|, |g_2 + g_3|) \left(\left(\frac{\partial \hat{w}}{\partial \xi}\right)^2 + \left(\frac{\partial \hat{w}}{\partial \eta}\right)^2\right). \end{aligned}$$

[1] In a triangle,

$$\frac{\sin\alpha}{a} = \frac{\sin\beta}{b} = \frac{\sin\gamma}{c}.$$

The elements of J^{-1} can all be bounded as

$$\left|\frac{\partial \xi}{\partial x}\right|, \left|\frac{\partial \xi}{\partial y}\right|, \left|\frac{\partial \eta}{\partial x}\right|, \left|\frac{\partial \eta}{\partial y}\right| \leq \frac{c}{|\det J|}.$$

This last result together with (4.6) allow to find bounds to the elements of the metric tensor

$$g_1 , g_2 , g_3 \leq \frac{2c^2}{|\det J|^2} \leq \frac{8}{c^2 \sin^2 \alpha}.$$

Then,

$$|w|^2_{1,T} \quad \leq \quad \frac{16}{\sin \alpha} |\hat{w}|^2_{1,\hat{T}}. \tag{4.7}$$

Another inequality can be established using the other side of the bounds (4.6). Here, we invert the transformation i.e. we compute

$$\begin{aligned} \left(\frac{\partial \hat{w}}{\partial \xi}\right)^2 + \left(\frac{\partial \hat{w}}{\partial \eta}\right)^2 &= \hat{g}_1 \left(\frac{\partial w}{\partial x}\right)^2 + 2\hat{g}_2 \left(\frac{\partial w}{\partial x}\frac{\partial w}{\partial y}\right) + \hat{g}_3 \left(\frac{\partial w}{\partial y}\right)^2 \\ &\leq \max(|\hat{g}_1 + \hat{g}_2|, |\hat{g}_2 + \hat{g}_3|) \left(\left(\frac{\partial \hat{w}}{\partial \xi}\right)^2 + \left(\frac{\partial \hat{w}}{\partial \eta}\right)^2\right). \end{aligned}$$

with

$$\hat{G} = J^T J = \begin{bmatrix} \left(\frac{\partial x}{\partial \xi}\right)^2 + \left(\frac{\partial x}{\partial \eta}\right)^2 & \frac{\partial x}{\partial \xi}\frac{\partial y}{\partial \xi} + \frac{\partial x}{\partial \eta}\frac{\partial y}{\partial \eta} \\ \frac{\partial x}{\partial \xi}\frac{\partial y}{\partial \xi} + \frac{\partial x}{\partial \eta}\frac{\partial y}{\partial \eta} & \left(\frac{\partial y}{\partial \xi}\right)^2 + \left(\frac{\partial y}{\partial \eta}\right)^2 \end{bmatrix} = \begin{bmatrix} \hat{g}_1 & \hat{g}_2 \\ \hat{g}_2 & \hat{g}_3 \end{bmatrix}.$$

Elements of $\hat{G}$ can be bounded as

$$|\hat{g}_1| , |\hat{g}_2| , |\hat{g}_3| \leq 2c^2$$

which leads to

$$|\hat{w}|^2_{1,\hat{T}} \leq \frac{8}{\sin \alpha} |w|^2_{1,T}. \tag{4.8}$$

Finally,

$$\frac{\sin^{1/2} \alpha}{4} |w|_{1,T} \leq |\hat{w}|_{1,\hat{T}} \leq 2\sqrt{2} \sin^{-1/2} \alpha |w|_{1,T}. \tag{4.9}$$

Expressions (4.9) and (4.6) can be easily generalized to

$$\gamma(s) \sin^{s-1/2} \alpha c^{s-1} |w|_{s,T} \leq |\hat{w}|_{s,\hat{T}} \leq \Gamma(s) \sin^{-1/2} \alpha c^{s-1} |w|_{s,T}. \tag{4.10}$$

In (4.6), we have $\Gamma(0) = \sqrt{2}$ and $\gamma(0) = 1$. In (4.9), we have $\Gamma(1) = 2\sqrt{2}$ and $\gamma(s) = 1/4$. It is now possible to demonstrate a very important result of the finite element theory. We know by (4.10) that

$$|u - U|^2_{s,T} \leq \gamma(s)^{-2} \sin^{-2s+1} \alpha c^{-2s+2} |\hat{u} - \hat{U}|^2_{s,\hat{T}}.$$

Next, use (4.2) to have

$$|u-U|_{s,T}^2 \leq \gamma(s)^{-2}\sin^{-2s+1}\alpha\, c^{-2s+2}C|\hat{u}|_{p+1,\hat{T}}^2.$$

Finally, use the right inequality of (4.10) to find

$$|u-U|_{s,T}^2 \leq \gamma(s)^{-2}\sin^{-2s+1}\alpha\, c^{-2s+2}C\Gamma^2(p+1)\sin^{-1}\alpha\, c^{2p}|u|_{p+1,T}^2.$$

Combining the constants into $\mathscr{C}$, we find the desired result

$$|u-U|_{s,T} \leq \frac{\mathscr{C}(s,p)}{\sin^s\alpha}\, c^{p+1-s}|u|_{p+1,T}. \tag{4.11}$$

There are other ways to get result (4.11). Here is one. We call δ (resp. $\hat{\delta}$) the diameter of the inner circle of T (resp. $\hat{T}$) and c (resp. $\hat{c}$) its largest edge measure. The affine mapping (4.3) maps a vector $\hat{x} \mapsto x = x_0 + J\hat{x}$. The norm of J defined as

$$\|J\| = \sup_{\hat{v}\in\mathbb{R}^2} \frac{\|J\hat{v}\|}{\|\hat{v}\|}.$$

tells us how much a vector $\hat{x}$ can be magnified by an affine mapping. This definition is equivalent to

$$\|J\| = \sup_{\hat{v}\in\mathbb{R}^2,\ \|\hat{v}\|=\hat{\delta}} \frac{\|J\hat{v}\|}{\delta_{\hat{T}}}.$$

Indeed, we like to compute this norm using vectors $\|\hat{v}\| = \hat{\delta}$ that have a known norm equal to the innercircl radius of $\hat{T}$ because it allows a very simple geometrical interpretation of $\|J\|$ (see Figure **??**). Every vector of norm $\hat{\delta}$ correspond to one diameter of the inner-circle of $\hat{T}$. Each diameter of the inner-circle of $\hat{T}$ is transformed through J on a vector that is indeed completely inscribed into T. There is no vector that is totally inside T and that has a length larger than c, which leads to the bound

$$\|J\| \leq \frac{c}{\hat{\delta}}.$$

Inverting the role of T and $\hat{T}$, we get the following result

$$\|J^{-1}\| \leq \frac{\hat{c}}{\delta}.$$

Combining both bounds on $\|J\|$ and $\|J^{-1}\|$ gives

$$\|J\|\|J^{-1}\| \leq \frac{c\hat{c}}{\delta\hat{\delta}}.$$

The diameter of the inner-circle of a triangle T is given by the formula

$$\delta = \frac{4|T|}{a+b+c}.$$

In the specific case of the canonical triangle $\hat{T}$, we have

$$\hat{\delta} = \frac{2}{2+\sqrt{2}} \quad \text{and} \quad \hat{c} = \sqrt{2} \quad \rightarrow \quad \|J\|\|J^{-1}\| \leq \left(\frac{2}{\sqrt{2}+1}\right)\frac{c}{\delta} = \mathscr{C}\frac{c}{\delta}.$$

Now, it is easy to see [**?**] that

$$\|\hat{w}\|_{s,\hat{T}} \leq c\|J\|^s |\det J|^{-1/2} \|w\|_{s,T}.$$

Inverting again the role of T and $\hat{T}$, we get the following result

$$\|w\|_{m,T} \leq c\|J^{-1}\|^s |\det J|^{1/2} \|\hat{w}\|_{m,\hat{T}}.$$

We can now re-compute bounds on discretization errors as

$$\|u-U\|_{s,T} \leq c\|J^{-1}\|^s |\det J|^{1/2} \|\hat{u}-\hat{U}\|_{s,\hat{T}}.$$

Next, use (4.2) to have

$$\|u-U\|_{s,T} \leq c\|J\|^{-s} |\det J|^{1/2} C\|\hat{u}\|_{p+1,\hat{T}}.$$

Then we finally get

$$\begin{aligned} \|u-U\|_{s,T} &\leq \mathscr{C}(\|J\|\|J^{-1}\|)^s \|J\|^{p+1-s} \|u\|_{p+1,T} \\ &\leq \mathscr{C}(s,p)\left(\frac{c}{\delta}\right)^s c^{p+1-s} \|u\|_{p+1,T}. \end{aligned}$$

Equations (4.11) and (4.12) tell very much the same thing. More precisely,

$$\begin{aligned} \frac{c}{\delta} = \frac{a+b+c}{2b\sin\alpha} &\leq \frac{a+b+a+b}{(a+b)\sin\alpha} = \frac{2}{\sin\alpha} \\ &\geq \frac{c+c}{2c\sin\alpha} = \frac{1}{\sin\alpha} \end{aligned}$$

which means that $\frac{c}{\delta}$ and $\frac{1}{\sin\alpha}$ are two equivalent measures (one cannot blow up while the other remains finite).

Result (4.11) was proved in 1968 by Milos Zlamal [**?**]. The consequense is the following one: when a mesh is refined, i.e. when $c \to 0$, then ensuring that the sine of the smallest angle of the triangulation does not go to zero ensures the optimal convergence of the interpolation error. In a triangle, $\sin\alpha \to 0$ for $\alpha \to 0$ or for $\alpha \to \pi$. In the case $\alpha \to \pi$, the two other angles β and γ will inevitably go to zero. It all cases, results (4.11) and (4.12) indicate the importance of not generating triangles with small angles. This is the famous angle condition.

Seven years after Zlamal's paper, a famous paper of Babuska and Aziz [**?**] showed that the minimum angle condition was not the right condition: even though Zlamal's condition $\sin\alpha$ bounded away from zero is a sufficient condition for convergence, it is actually not necessary. What Babuska and Aziz point out in their famous paper is that bounding the maximum angle γ away from π while refining the mesh is the right condition in order to ensure optimal convergence properties of the finite element approximation.

4.1.2 Discrete maximum principle

Consider a domain $\Omega \subset \mathbb{R}^2$ with its (smooth) boundary $\partial\Omega$. The Dirichlet-Laplace problem consist in finding $u(x,y) \in H^2(\Omega)$ solution of

$$\begin{aligned} \nabla^2 u = 0 \quad & \text{on} \quad \Omega, \\ u = \bar{u}(x,y) \quad & \text{on} \quad \partial\Omega. \end{aligned}$$

It is possible to show that, for any disk D of center (x_0, y_0) and of radius δ that is totally inside Ω, we have

$$u(x_0,y_0) = \frac{1}{\pi\delta^2}\int_D u(x,y)\,dx\,dy.$$

The solution of the Dirichlet-Laplace problem at any interior point (x_0, y_0) of the domain Ω is equal to the average of u on any disk centered at (x_0, y_0) that is totally inside Ω. This implies that the solution u cannot have maximas or minimas inside Ω. The maximum value of u is therefore on Ω. For example, if u is a temperature, $\bar{u}$ is a fixed temperature at the boundary, then our Dirichlet-Laplace describes the steady-state thermal equilibrium inside Ω and the temperature cannot be higher inside the domain than on $\partial\Omega$. This is the maximum principle.

FInite elements are certainly the most common tool to solve such a Dirichlet-Laplace problem numerically. It is often important that the numerical scheme employed is endowed with a discrete maximum principle. Consider a triangular mesh and a standard finite element discretization of the Laplace operator using $P1$ conforming finite elements.

Another interresting result is the following one: it can be shown [**?**] that discrete maximum principles hold for P1-conforming finite element approximations of the Laplace problem when a triangulation has no obtuse angles. Continuous solutions u of the Laplace problem do not possess local extrema [**?**] which means that the maximum and the minimum of u are always on the boundary of the domain. This strong maximum principle is often very important to be fullfilled by the numerical scheme (see the reparametrization Chapter §**??** of this book). The P1-conformal finite element formulation of the Laplace proble consist in writing one equation per vertex q of the mesh (see Figure 4.2) that is

$$c_q u_q + \sum_{i=1}^{m} c_{iq} u_{q_i} = 0.$$

This value u_q should be bounded by above and by below by the q_i's in order to fullfill discrete maximum principle:

$$\min_i u_{q_i} \leq u_q \leq \max_i u_{q_i}.$$

Any consistent method that discretizes the Laplace operator is such that

$$c_q + \sum_{i=1}^{m} c_{iq} = 0.$$

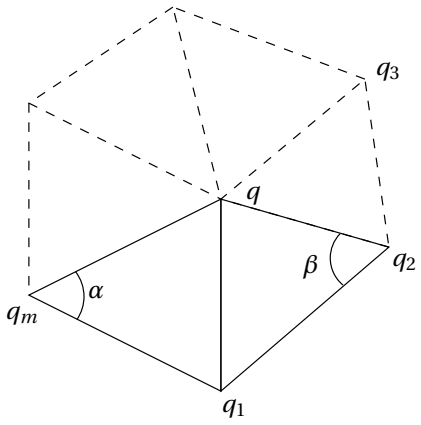


Figure 4.2: A simple configuration.

This is simply due to the fact that the Laplace operator is a derivative. Assume that every c_{iq} is negative. Then, c_q is obviously positive and

$$|c_q| > |c_{iq}| \quad \forall i$$

which also means that $1 < \frac{c_{iq}}{c_q} < -1$. Then we re-write,

$$u_q = -\sum_{i=1}^{m} \frac{c_{iq}}{c_q} q_i.$$

The vaue of u_q is bounded by its neighbors u_{q_i} if $0 < \frac{c_{iq}}{c_q} < -1$. which is true if $c_{iq} < 0$. It is possible to prove [**?**] that

$$c_{1q} = -\frac{\sin(\alpha+\beta)}{2\sin\alpha\sin\beta}.$$

Then, a mesh for which $\alpha + \beta < \pi$ for all pairs of elements is such that it verifies the discrete maximum principle. It is of course true if no obtuse angle is present in the mesh. For measuring the quality of elements, various element shape measures are available in the literature [**?**, **?**]. In a finite element point of view, one should choose a mesh quality that penalize elements with obtuse angles.

Finally, one can argue that equilateral triangles are optimal triangles they are the only ones that are able to fill the entire space with edges of constant size.

On good candidate for mesh quality is certainly the ratio between the inscribed radius and the circumradius of the triangle. We scale that quantity in order that an equilateral triangle has a quality of 1. Consider a triangle t with its three inner angles $\hat{a}$, $\hat{b}$ and $\hat{c}$. We define the triangle quality as

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

With this definition, the equilateral triangle has a $\gamma_t = 1$ and degenerated (zero surface) triangles have a $\gamma_t = 0$.

4.1.3 Triangle quality measures

Results (4.11) and (4.12) clearly tell that angles should be bounded away from 0 and π. A triangle quality measure should then detect a triangle that has one angle close to π (we call it a cap) and a triangle with one angle close to zero (we call it a needle). The best possible triangle is the equilateral one with all its angles being equal to $\alpha = \beta = \gamma = \pi/3$. Let us then define the shape measure of triangle T as a number $\kappa_T \in [-1,1]$ such that

1. $\kappa_T = 1$ if and only if the triangle is equilateral.
2. $\kappa_T = 0$ if and only if the triangle has a zero area.
3. $\kappa_T < 0$ if and only if the triangle is inverted.
4. κ_T is smooth i.e. it is derivable with respect to the position of the vertices.

The following choice

$$\boxed{\kappa_{T,1} = \frac{2}{\sqrt{3}} \sin\alpha}$$

is directly related to interpolation result (4.11) . Yet, this quality measure is not smooth due to the fact that α is the minimum angle and the derivative of α is not smooth. Quality measure $\kappa_{T,1}$ is therefore not usable if its gradient is used for optimizing a mesh. Note that this measure is naturally improved by Delaunay refinement algorithms for mesh generation (see §**??**). This other choice

$$\boxed{\kappa_{T,2} = \sqrt{3}\frac{\delta}{c}}$$

is related to and (4.12). It is also not smooth.

Some common shape measure for triangles involve D, the outer diameter T i.e. twice its circum-circle radius. The circumdiameter of a triangle T can be computed using the simple relation

$$D = \frac{abc}{2|T|}.$$

The following measure

$$\boxed{\kappa_{T,3} = 2\frac{\delta}{D} = \frac{16|T|^2}{abc(a+b+c)}} \tag{4.12}$$

that correspond to twice the ratio of inner and outer diameters (factor 2 is there in order to have $\kappa_{T,3} = 1$ for the equilateral triangle as well) is the most common triangle measure that is used in practice. It is smooth, easy to compute (no min or max) but it does not detect inverted elements. Its square root

$$\boxed{\kappa_{T,4} = \sqrt{\kappa_{T,2}} = \frac{4|T|}{\sqrt{abc(a+b+c)}}} \tag{4.13}$$

has all the good properties and is very useful in practice.

We have seen in previous sections that caps i.e. triangles with one very obtuse angles should be avoided. Previous quality measures do not distinguish between needles and caps. The following measure

$$\boxed{\kappa_{T,5} = \frac{4}{\sqrt{3}} \frac{|T|}{(abc)^{2/3}}} \tag{4.14}$$

4.2 Mesh size

A mesh size field is a tool that aims at controlling the mesh size. We define the mesh size function $h(x, y)$ as a function the defines at every point of the planar domain a target size h for the mesh edges at the point. Consider an edge e of the mesh. We define the adimensional length of the edge with respect to the size field h as

$$l_e = \int_e \frac{1}{h(x,y)} dl. \tag{4.15}$$

The mesh generation being a discrete process, it is in general impossible to have exactly $l_e = 1$ for all the edges of the mesh. We say that edge e the size criterion if its adimensional size verifies

$$1 - \xi \leq l_e \leq 1 + \xi.$$

A long edge is such that $l_e > 1 + \xi$ and a short edge is such that $l_e < 1 - \xi$. Here, we choose ξ in such a way that splitting a long edge in two equal parts does not create a short edge:

$$\frac{1+\xi}{2} = 1 - \xi \;\rightarrow\; \xi = \frac{1}{3}.$$

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

$$\tau = \exp\left(\frac{1}{ne} \sum_{e=1}^{ne} \tau_e\right) \tag{4.16}$$

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 $\tau = 1$.

4.3 One dimensional meshing

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

$$\int_{t_1}^{t_2} \frac{\|p'(t)\|}{h(x,y)} dt = N. \tag{4.17}$$

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:

$$\int_{t_1}^{T_i} \frac{\|p'(t)\|}{h(x,y)} dt = i. \tag{4.18}$$

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, (4.18) is evaluated with a recursive numerical integration rule.

It is sometimes useful to limit mesh size variations in order to have a mesh that is smoothly varying in size. Two edges of the mesh which are connected by a vertex should not have a size that varies more than α, with a typical value $\alpha = 1.3$.

Let us call l the curvilinear abscissa

$$l(t) = \int_{t_1}^{t} \|p'(t)\|.$$

Let us call $h(l)$ the size field at point $p(l(t))$. on the curve and $h(l)$ the size field. Now, assume that we want to control the variation of the size of the 1D mesh, i.e. we want that

$$h(l + \Delta l) - h(l) < (\alpha - 1)\Delta l$$

or

$$\frac{\partial h}{\partial l} = \frac{\partial h}{\partial t} \frac{1}{\|p'(t)\|} < \alpha - 1.$$

In Gmsh we use a Gauss-Seidel iteration (iterate forward and then backward). The convergence is usually very fast.

An issue that may occur during during the 1D mesh procedure concerns possible self-intersections of 1D mesh edges. This may happen even if in the case when model edges of the geometry do not self intersect: Figure 4.3 shows two model edges that are very close to each other. Yet, even the geometry is itself not self-intersecting, it is indeed possible that a 1D discretization intersects itself. This kind of issue will definitively happend at some point in the life of a mesh generator. It is therefore mandatory to define a systematic procedure that fixes the issue. In Gmsh, the mesh flow is modified in the following fashion. Mesh edges that intersected with each other are found using the Bentley-Ottmann Algorithm. Every intersecting edges e_k is split into 2 segments and the new point is snapped onto the geometry. Then, we go back to the first step until the list of intersecting edges is empty. If an intersecting edge is smaller than the geometrical tolerance, then an error message is thrown claiming that the geometry is itself self intersecting.

4.4 The general 2D Meshing procedure

Let us start by defining a simple planar surface in Gmsh (Figure 4.4). This model is contained in the Gmsh distribution and is called `conge.geo`. This model is composed of two planar surfaces, 20 model edges and 25 model vertices. We consider a uniform size field $\delta = 0.005$. The first step for doing the mesh is to discretize (or

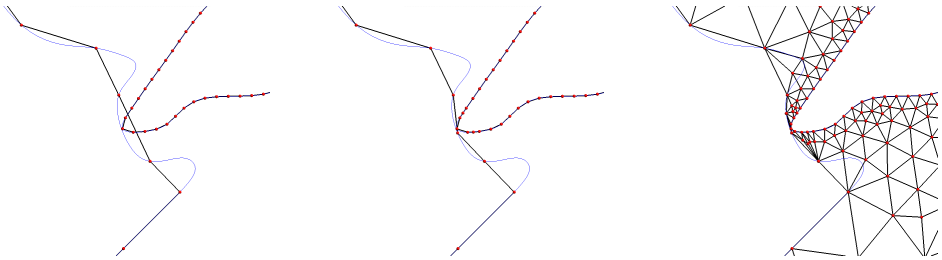


Figure 4.3: A geometry with two islands that are very close to each other. The first image shows the initial 1D mesh that respects mesh size field. The second image shows the first iteration of the recovery algorithm. The third image shows the final mesh that was possible to realize after 2 recovery iterations

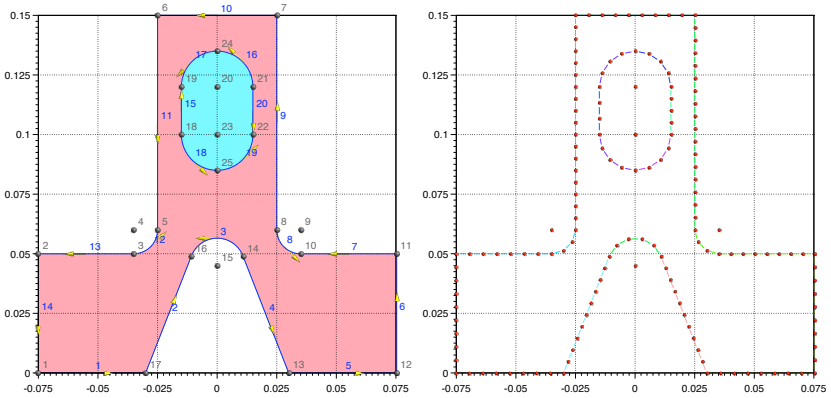


Figure 4.4: A simple 2D the model `conge.geo` (left) and the 1D mesh (right)

mesh) the edges of the model. This operation is done computing the primitive (4.18) for every edge of the model. The result is presented in Figure 4.4. Then, each of the two model faces is triangulated.

At first, all the mesh vertices $S = \{p_1, \ldots, p_n\}$ that bounds the model face are considered. Four extra mesh vertices $S_0 = \{p_{-4}, p_{-3}, p_{-2}, p_{-1}\}$ are added to the list. These four vertices are the corners of a rectangle that encompasses the model surfaces (Figure 4.8).

The set of points $S_0 \cup S$ is then triangulated using the Delaunay algorithm (see §3.3). Two possible issues have to be addressed at that point.

4.4.1 Recovering the boundary edges

Building a Delaunay triangulation of the set of points that consist in all vertices of the 1D mesh $S = \{p_1, \ldots, p_n\}$ plus the 4 additional infinite points $S_0 = \{p_{-4}, p_{-3}, p_{-2}, p_{-1}\}$ do not give any guarantee that all edges of the 1D mesh are present in $\mathrm{DT}(S_0 \cup S)$. *Delaunay triangulations deal with points and triangles, not with edges.*

In a mesh generation context, it is mandatory that every mesh edge $p_i p_j$ of the 1D mesh belong to the triangulation. There are two ways to recover the boundary edges: (i) the first one ensures that the 1D mesh enforces the conformity of the Delaunay triangulation *a priori* and (ii) the second one modifies topologically the mesh using edge flips, leading to a constrained Delaunay mesh.

A priori Delaunay conformity

Let us recall the definition of a locally Delaunay edge: given a triangulation $T(S)$ and an edge $p_i p_j$ in the triangulation that is adjacent to two triangles $\Delta_I = p_i p_j p_k$ and $\Delta_J = p_i p_l p_j$. We call edge $p_i p_j$ *locally Delaunay* if p_l lies on or outside the circumcircle of Δ_I.

We have shown in §2.2 that a triangulation $T(S)$ is a Delaunay triangulation if and only if all its edges are locally Delaunay. There is indeed a very simple way to enforce that every edge $p_i p_j$ of the 1D mesh belongs to $\mathrm{DT}(S)$

Consider an edge $p_i p_j$ and the circle centered at $\frac{1}{2}(p_i + p_j)$ of diameter $D = \|p_2 - p_1\|$. If this circle is empty, then this edge is a Delaunay edge. If it is not empty, then a new point $p_{ij} = \frac{1}{2}(p_i + p_j)$ is inserted at the center of edge $p_i p_j$. The two new edges $p_i p_{ij}$ and $p_{ij} p_j$ are twice shorter so that the area $2 \times \pi(D/4)^2$ covered by the union of ther respective circles is twice smaller than the area $\pi(D/2)^2$ coverd by the initial circle. This algorithm applied sequentially to every edge that is not locally Delaunay will converge to a 1D mesh that have all its edges in the Delaunay triangulation. Figure 4.5 shows an example where only one edge has to be refined. Edge $p_i p_j$ does not belong to the Delaunay triangulation. There exis indeed no circle passing through p_i and p_j that is empty. After a first refinement, the circle centered at $\frac{1}{2}(p_{ij} + p_j)$ is not empty even though the Delaunay triangulation is conforming to the 1D mesh. In that case, there exist a circle passing through p_{ij} and p_j that is empty, but it is not the one that is centered at $\frac{1}{2}(p_{ij} + p_j)$. This empty circle is represented in dashed red lines on Figure 4.5 . Obviously, this *a priori* Delaunay

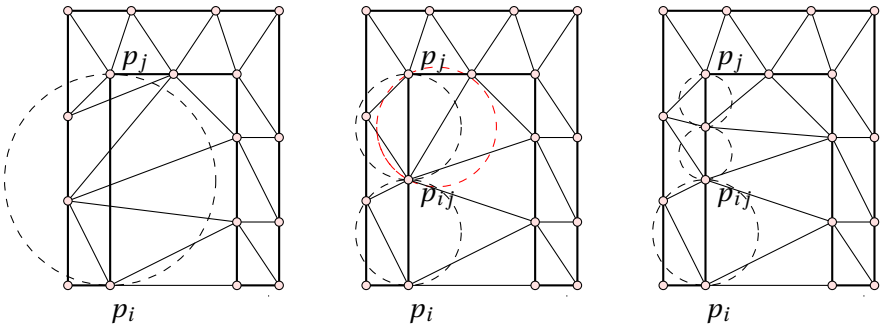


Figure 4.5: A priori Delaunay conformity.

conformity algorithm may overrefine the 1D mesh. Finally, step 3 allows to have every circle empty. Such an approach has the advantage of maintaining a Delaunay mesh in the whole meshing process. The main issue here is that there exist no control on the number of points that should be added to the 1D mesh to enforce this *a priori* conformity. For example, when two boundary curves are very close to each others, the 1D mesh could be refined in such a way that the 1D spacing is equal to the distance between the two curves, which could be arbitrary small (see Figure 4.6).

Boundary edges recovery using local edge flips

The *a priori* approach is not the one that is used in enginnering applications. The main reason has of not using the *a priori* approach is that mesh size cannot be controlled. It is indeed possible to recover the missing 1D edges using edge swaps. In general, any edge $p_i p_j$ of any triangulation $T(S)$ can be recovered using edge flips. The algorithm work as follows.

- Create a list of edges $E = \{e_1, \ldots, e_m\}$ that intersect $p_i p_j$ (see Figure 4.7).
- For $i = 1, \ldots, m$
  - If flip edge e_i is flippable then flip edge e_i. If the new edge issued from the flip intersects $p_i p_j$, then put this new edge at the end of the list: $m = m + 1$.
  - Else put e_i at the end of the list: $m = m + 1$, $e_m = e_i$.

This algorithm always terminates. Yet, if the initial mesh was a Delaunay mesh, then the constrained mesh may not be Delaunay anymore. In what follows, we will exted the Delaunay cavity procedure to constrained meshes.

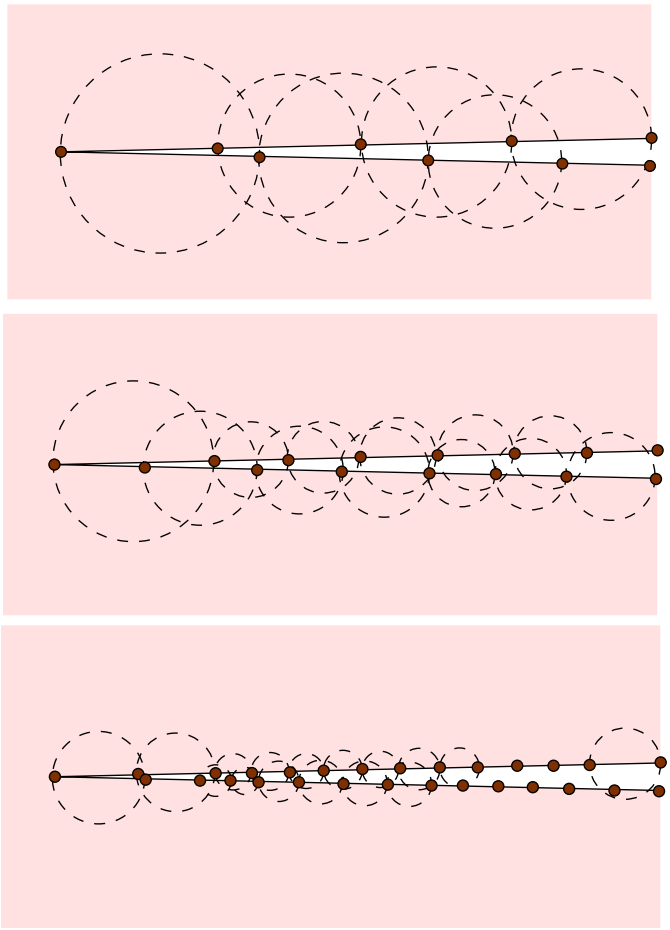


Figure 4.6: Three steps of the *a priori* Delaunay conformity in the case of two curves that form a crack. At the third stage, the edges close to the crack tip (left) are still refined because the distance between the upper and the lower lip of the crack is getting closer.

4.4.2 The empty mesh

At that point, a mesh $T(S_0 \cup S)$ is available that is conforming to the boundary which means that it contains all the edges of the 1D mesh (see Figure 4.8). At that point, a mesh that Triangles that are outside the domain have to be removed to form what is called the *empty mesh*. This can be done simply by choosing one triangle that contains one of the four infinite and to walk through its edge-neighbors, recursively, stopping the process when crossing an edge that belongs to the boundary of the domain. The element that is on the other side of this edge is inside the domain. Then,

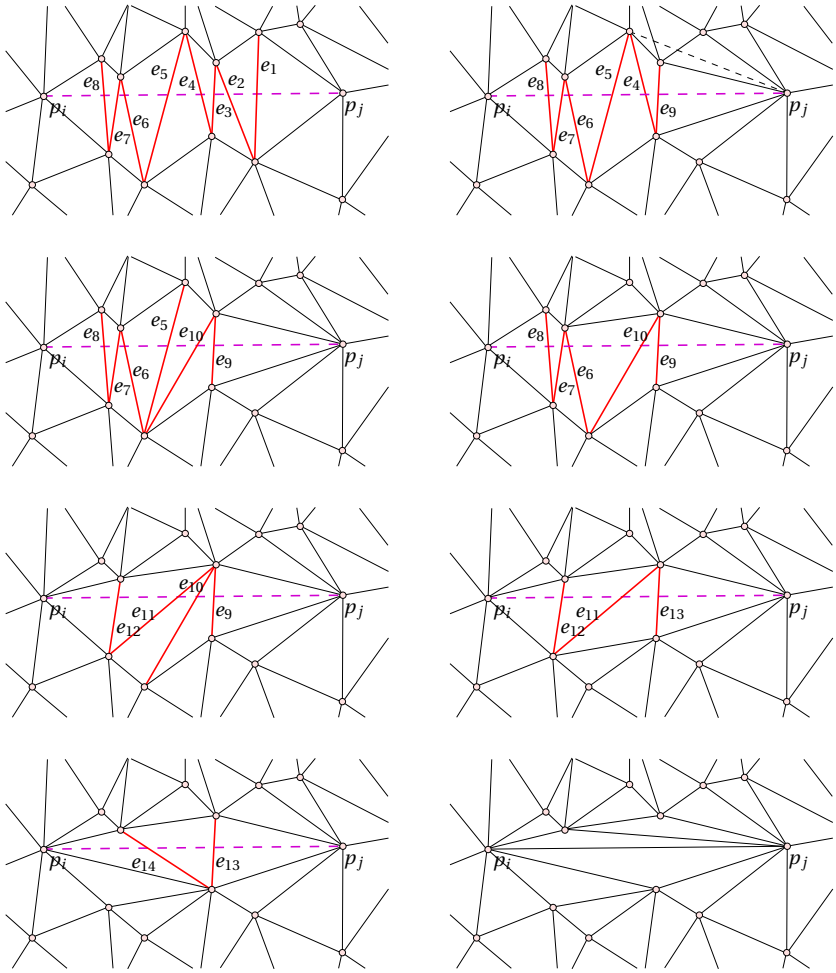


Figure 4.7: Recovering edge $p_i p_j$. Edges e_1 and e_2 are flipped. Edge e_3 that is not flippable is changed to e_9. Then, edge e_4 is flipped but this flip does not reduce the number of intersections. Edge e_{10} is then added to the list. Edge e_5 is then flipped and removed from the list. Edge e_6 is then flipped and edge e_{11} is added. Edge e_7 is not flippable and e_{12} is added. Edge e_8 is flipped and removed. Edge e_{11} is flipped and e_{14} is added. Edge e_{12} is then flipped and removed. Edge e_{13} is flipped and removed. Finally, e_{14} is flipped and $p_i p_j$ is recovered.

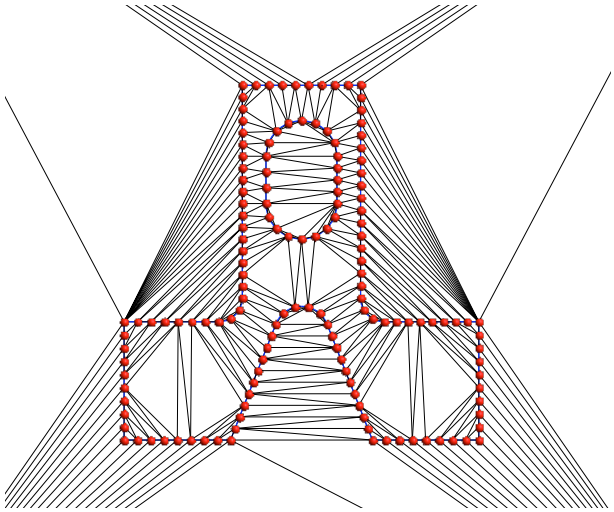


Figure 4.8: Mesh $T(S_0 \cup S)$ of `conge.geo` that is conforming to the boundary.

we walk again through the neighbors allow to get all element that are inside the domain: this is the empty mesh (Figure 4.9, left) in the sense that it has no interior points.

4.4.3 Mesh refinement

Points are inserted in the empty mesh in order to obtain elements of desired shape and size (Figure 4.9, right). Different strategies are possible to refine the empty mesh. Here, we describe the three approaches that are available in Gmsh's native 2D meshers.

Local mesh modifications

One first approach for refining the empty mesh is the use of local mesh modifications. In 2D, there exist 3 operators that can locally modify a the topology of a triangular mesh (see Figure 4.10):

Edge splits are applied to long edges i.e. edges with adimensional lengths (4.15) $l_e > l_{up}$ where l_{up} is a threshold that will be defined shortafter. The new point is usually located at the adimensional middle of e with the aim of creating two new edges of adimensional size equal to $l_e/2$.

Edge collapses are applied to short edges i.e. edges with adimensional lengths $l_e < l_{low}$. An edge cannot be collapsed if one of the remaining triangles after the collapse is inverted.

An edge flip is applied if the flipped configuration is composed of triangles of better quality than the initial configuration.

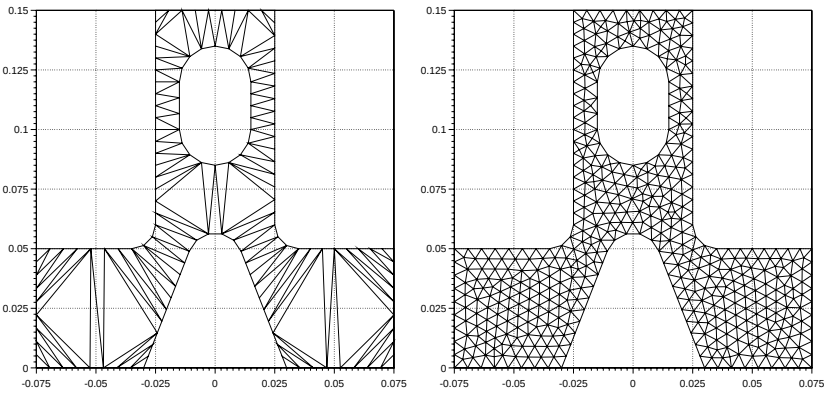


Figure 4.9: The empty mesh of `conge.geo` (left) and. final mesh (right).

A long edge that is split should not become a short edge: $l_{up}/2 \geq l_{low}$. A reasonable choice for the admissible edges size interval $[l_{low}, l_{up}]$ is to center it on 1. The latter two condition lead to the choice $[l_{low}, l_{up}] = [2/3, 4/3]$. This is of course not the only possible choice. For example, choosing $[l_{low}, l_{up}] = [\sqrt{2}/2, \sqrt{2}]$ is a choice that enables longer edges and that essentially leads to meshes with less nodes and triangles.

Another local mesh modification is the Vertex Re-positioning: a vertex can be moved optimally inside the cavity made of all its surrounding triangles. The optimal position is chosen in order to maximize the worst element quality.

1. Each edge that is too long is split;
2. Each edge that is too short is removed using an edge collapse operator;
3. Edges for which a better configuration is obtained by swapping are swapped;
4. Vertices are re-located optimally.

One first idea would be for example to insert a new vertex in the middle of the longest edge $p_i p_j$ (using, of course, its adimensional length). This is the default surface meshing algorithm of Gmsh. We are aware that this point insertion algorithm has a fundamental flaw. There is indeed no guarantee that there exist no point in the mesh that is closer to the middle point than p_1 and p_2. This point insertion algorithm may then generate a new edge that is indeed too short and that will have to be removed afterwards through edge collapsing.

A variant of the previous technique is used in BAMG [**?**]. It consist in saturating the edges of the mesh with points, using e.g. the 1D technique described in §**??**. A bucket tree datastructure is used to remove points that are too close. Thanks to that filtering operation, no point removal has to be performed afterwards.

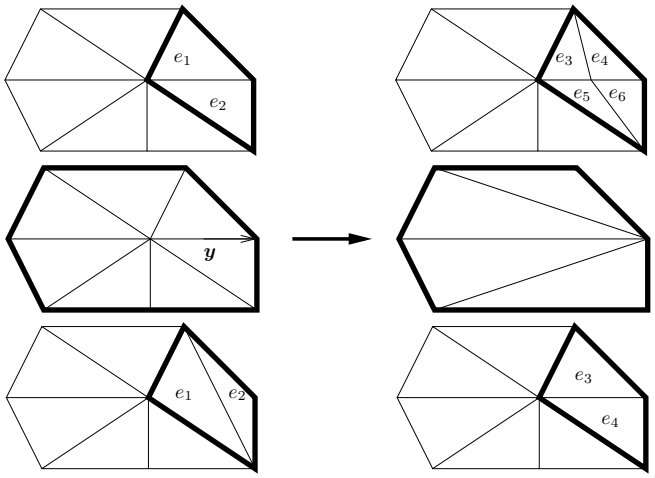


Figure 4.10: Illustration of local mesh modifications.

An optimal point insertion procedure should be able to insert points that will not have to be removed afterwards. In other words, one should not insert points that are too close to other points. An alternative to edge splitting would be insert the new point at the circumcenter of the largest triangle [**?**]. We could define the adimensional size of a triangle $t(a,b,c)$ as the ratio between its circumradius and the mesh size at its circumcenter. The circumcenter C of a planar triangle t can be computed as follows:

$$(a_x - C_x)^2 + (a_y - C_y)^2 = (b_x - C_x)^2 + (b_y - C_y)^2$$
$$(a_x - C_x)^2 + (a_y - C_y)^2 = (c_x - C_x)^2 + (c_y - C_y)^2.$$

This leads to the system:

$$\begin{bmatrix} (b_x - a_x) & (b_y - a_y) \\ (c_x - a_x) & (c_y - a_y) \end{bmatrix} \begin{bmatrix} C_x \\ C_y \end{bmatrix} = \frac{1}{2} \begin{bmatrix} b_x^2 - a_x^2 + b_y^2 - a_y^2 \\ c_x^2 - a_x^2 + c_y^2 - a_y^2 \end{bmatrix}.$$

Circumradius R_t and inner radius r_t of a triangle t are nicely related as

$$R_t = \|C - a\| = \frac{1}{2r_t} \frac{\|b-a\|\|c-a\|\|a-b\|}{\|b-a\| + \|c-a\| + \|a-b\|}.$$

The adimensional size l_t of triangle t is computed as

$$l_t = \sqrt{3} \frac{R_t}{\delta(C)}. \tag{4.19}$$

The $\sqrt{3}$ factor in (4.19) is explained by the fact that the circumradius of an equilateral triangle with edges of size 1 being $R_t = 1/\sqrt{3}$.

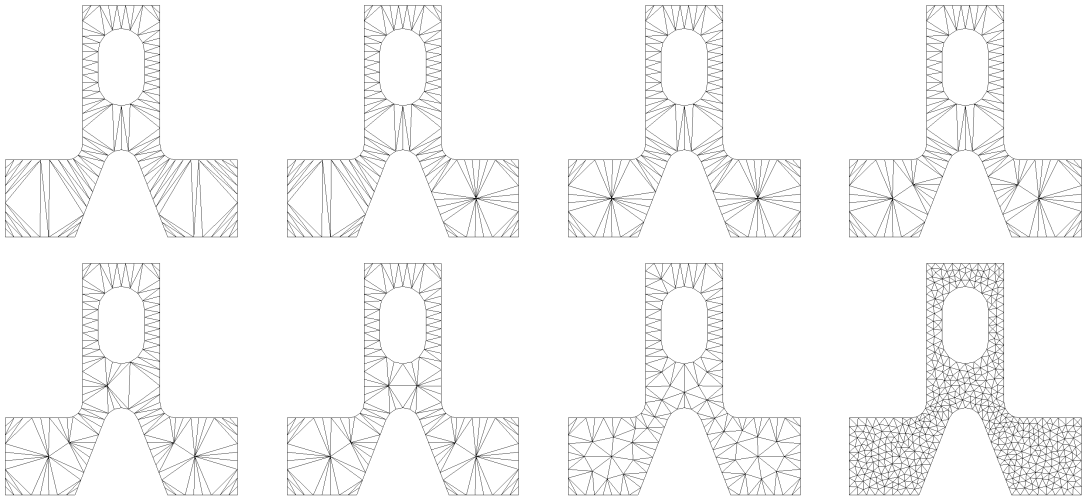


Figure 4.11: Illustration of the Bowyer-Watson algorithm with a point insertion scheme based on the circumcenter of the largest triangle

Triangles are sorted with respect to this adimensional length. A new point is inserted at C and the Bowyer-Watson algorithm is used to locally recreate a new Delaunay triangulation with the additional point. The algorithm stops when $l_t < l_{max}$ with $l_{max} \geq 1$. It is indeed NOT a good idea to take $l_{max} > 1$. Imagine that triangle t has an adimensional length of $l_t = 1.1$. Inserting a new point at its circumcenter would generate edges that are about two times smaller than the existing ones that are of size 1.1. We define a range of acceptable sizes $[l_{min}, l_{max}]$ in a way that splitting a large triangle edge $l_t > l_{max}$ would not generate a triangle that is too small i.e. for which $l_t < l_{min}$. This can be translated in $l_{max} = 2l_{min}$. An usual choice for the acceptable range is to center it around 1 i.e. $l_{max} + l_{min} = 2$. This gives $[l_{min}, l_{max}] = [2/3, 4/3]$. Figure 4.11 gives an example of the Bowyer and Watson algorithm with a point insertion based on the circumcenter of the largest triangle. Points on the final mesh are nicely distributed so that only a very light optimization (laplace smoothing) is necessary for obtaining the final mesh.

All point insertion algorithms that we have described before do not converge to a mesh that is composed of a majority of equilateral triangles. Frontal methods have the advantage of generating nicely shaped triangles. Yet, those latter methods are less robust and slower than Delaunay meshers. It is possible to build a hybrid method that uses the Bowyer and Watson algorithm that guarantees always to play with a valid mesh but where points are inserted in a “frontal fashion”. Such a frontal-delaunay method should remain fast, which means that a point that is inserted should never be removed afterwards.

CHAPTER 5

Quadrangulations

5.1 Topology of quadrilateral meshes

5.1.1 Euler Characteristic

Assume a closed orientable surface $\mathscr{S}$. The genus g of $\mathscr{S}$ is an integer representing the maximum number of cuttings along non-intersecting closed simple curves without rendering the resultant manifold disconnected. Consider a sphere. There exist no closed curve on the sphere that does not divide it into two disconnected parts: its genus is $g = 0$. A simple torus has a genus of one. The genus of a surface $\mathscr{S}$ can be defined in terms of its Euler characteristic χ (see §1.1). Both g and χ carry the same topological information and their relationship (valid for orientable closed surfaces) is

$$\chi = 2 - 2g. \tag{5.1}$$

The Euler characteristic of a sphere is therefore $\chi = 2$ and the one of a torus is $\chi = 0$.

Formula (5.1) is valid for closed surfaces. Assume that surface $\mathscr{S}$ has b boundaries, then the Euler-Poincare characteristic changes to

$$\chi = 2 - 2g - b \tag{5.2}$$

Which correspond to the topology of a sphere with g handles and b holes. For example, a cylindrical topology can be constructed by opening two separated holes in a sphere. Then $\chi = 2 - 2 \times 0 - 2 = 0$.

5.1.2 Poincaré-Hopf Theorem

In a mesh with n nodes, n_e edges and n_f faces, Euler characteristic can be computed as

$$\chi = n - n_e + n_f.$$

Assume a quadrangulations of $\mathscr{S}$ made of n vertices, n_e edges and n_f quadrilaterals. An internal vertex in a quadrilateral mesh is regular if it has exactly 4 adjacent quads.

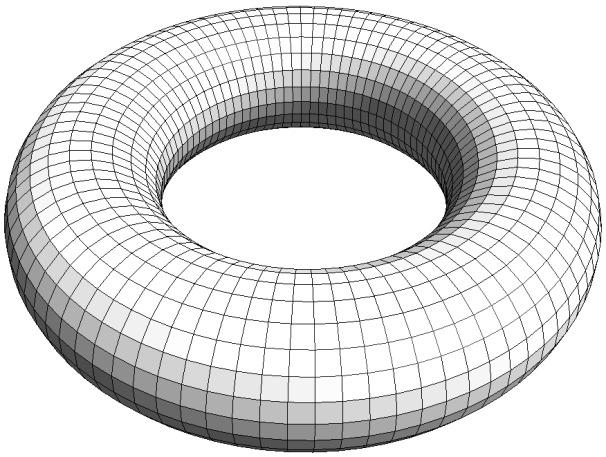


Figure 5.1: A fully regular quadrilateral mesh of a torus

Its it said to have a valence of 4. A regular quadrangulation is a quadrangulation with regular vertices only.

At that point, let us see wether a quadrangulation made of regular vertices only is possible. In the quadrangulation of a closed surface, each quadrilateral is made of 4 edges while each edge has two adjactent quadrilaterals. This means that $n_e = 2n_f$ which leads to

$$n - n_f = \chi.$$

If every vertex is regular, then each vertex has 4 quadrilateral neighbors and each quadrilateral has 4 vertices. Then, $n_f = n$ which means that only torus-like closed surfaces with $\chi = 0$ can be covered with a regular quadrangular mesh (see Figure 5.1).

Poincaré-Hopf Theorem is stated as follows: let be a vector field on $\mathscr{S}$ with K isolated zeroes z_i (a zero is an isolated singularity of the field). Then we have the formula

$$\sum_{i=1}^{K} \text{index}(z_i) = \chi.$$

The index of the singularity is $+1$ for a source singularity and -1 for a saddle singularity. It is possible to develop a discrete version of this theorem.

If $\chi \neq 0$, irregular vertices have to be present in the mesh, i.e. vertices of valence different than 4. Let us now how irregular vertices affect the Euler-Poincaré characteristic. Assume n_k vertices of valence $4-k$ and $n-n_k$ vertices of valence 4. We have

$$4n_f = 4(n-n_k) + (4-k)n_k$$

which means that

$$4n - 4(n-n_k) - (4-k)n_k = 4\chi$$

or

$$\chi = \frac{kn_k}{4}.$$

Each irregular vertex of valence $4-k$ counts for $k/4$ in the Euler characteristic. Each irregular vertex of valence 3 adds $1/4$ to the Euler-Poincare characteristic of the surface: its index is $1/4$. Each irregular vertex of valence 5 adds $-1/4$ to the Euler-Poincare characteristic of the surface and its index is $-1/4$. More generally, we obtain the following discrete version of Poincaré-Hopf Theorem:

$$\sum_k \frac{kn_k}{4} = \chi. \tag{5.3}$$

A sphere can then be quadrilateralized using $n_1 = 8$: the simplest version of this mesh is the cube with its 8 corners of valence 3.

5.1.3 Poincaré-Hopf theorem for triangular meshes

For a triangular mesh, the regular valence is 6. Taking into account that $3n_f = 2n_e$, we have

$$n_f = 2(n-\chi).$$

If we assume a mesh with regular vertices only, then $3n_f = 6n$ and the conclusion is similar: only torus-like surfaces with $\chi = 0$ can be covered with a perfecly regular triangular mesh. Introducing n_k irregular vertices of valence $6-k$ leads to

$$3n_f = 6(n-n_k) + (6-k)n_k.$$

and the discrete version of Poincaré-Hopf Theorem for triangular meshes is:

$$\sum_k \frac{kn_k}{6} = \chi. \tag{5.4}$$

Twelve irregular vertices of valence 5 are require to triangulate a sphere. The simplest version of this mesh is the icosaedron with $n_f = 20$, $n_e = 30$ and $n = 12$, each of the vertices being of valence 5.

5.1.4 Poincaré-Hopf theorem with boundaries

When surface $\mathscr{S}$ of genus g has b boundaries, $\chi = 2-2g-b$. Some vertices are situated on the boundaries of $\mathscr{S}$: assume that their number is n_h. Then, we can use the usual trick:

$$4n_f = 2(n_e - n_h) + n_h$$

combined with

$$n - n_e + n_f = \chi$$

leads to

$$n - n_f = \chi + \frac{n_h}{2}.$$

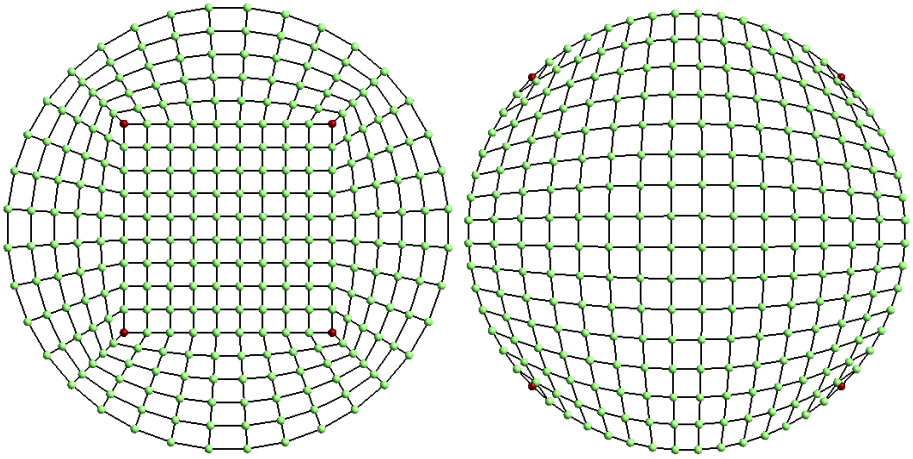


Figure 5.2: A quadrilateral mesh of a circle. Four singularities of index 1/4 (in red) are required to obtain such a mesh. The singularities may be inside the disk (left) or on its boundary (right)

If boundaries are considered, n_h has to be an even number for allowing a fully quadrilateral mesh regardless of the topology of $\mathscr{S}$.

In order to extend Poincaré-Hopf to meshes of a domain with boundaries, the regular valence of a vertex on a boundary has to be defined. For quadrilateral meshes, the regular valence of a vertex on the boundary is 2. This is easily justified by the fact that the boundary of the mesh is along one of the two vector fields so the vertex is not a singularity. Assume that $\mathscr{S}$ is a disk. A disk has the topology of a sphere $g = 0$ with one hole in it $b = 1$ so $\chi = 2 - 2 \times 0 - 1 = 1$. Four singularities of index 1/4 are needed to build a quad mesh on the disk. Those 4 singularities may be located anywhere in the disk, eventually on its boundary (see Figure 5.2). A vertex of the boundary has an index 1/4 if it has only one adjacent quadrangle and $-1/4$ if it has 3 adjactent quadrangles. Figure 5.2 (right) shows a quadrilateral mesh of a circle with all 4 singularities on the boundary. There, the unique quadrilateral adjactent to each singularity is ill shaped so it's not a good idea to have irregular vertices on smooth boundaries.

If the boundary is non smooth, it may be a good idea to locate some irregular vertices on the boundary. More precisely, we distinguish external (or reentrant) corners which external angles are about 90 degrees and internal corners which internal angles are about 90 degrees. Irregular vertices of degree $-1/4$ are suitable for reentrant corners while irregular vertices of index 1/4 are suitable for internal corners. Figure 5.3 shows two quadrilateral meshes of a domain with 4 internal corners and 1 reentrant corner. The mesh on the left is the one that has 6 non regular vertices on all corners of the boundary. The right mesh has the minimum amount of non regular vertices. Here, one internal corner and the reentrant corner have been regularized. The two meshes are both valid: choosing either one or the other depends on

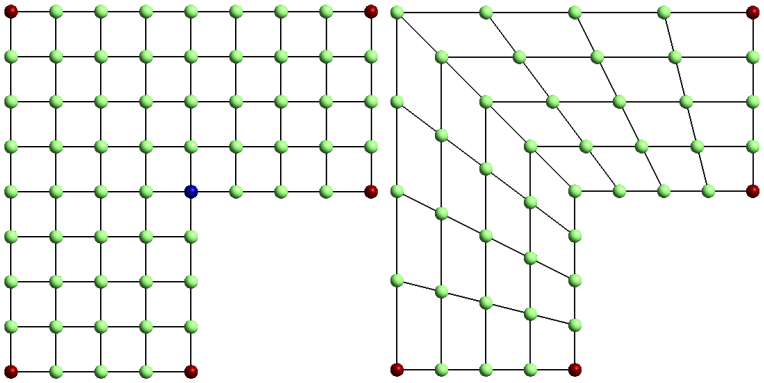


Figure 5.3: Quadrilateral meshes of a non smooth domain. Five singularities of index $1/4$ (in red) and one singularity of index $-1/4$ (in blue) are required to have the sum of the indices to be one (left). It is also possible to use 4 irregular nodes only (right), leading to a different result.

the underlying application. More specifically, those two configurations are typical of boundary layer meshes.

Up to now, we have assumed that it is possible to build a quadrilateral mesh with a minimum amount of irregular vertices. Even though this is the ideal situation, building such a "perfect" mesh is usually difficult. Figure 5.4 shows a quadrilateral mesh that has been generated using a standard technique. It has 8 vertices of valence 5 and 12 vertices ov valence 3.

5.2 Indirect generation of quadrilateral meshes

Let us first briefly recall which kinds of methods can be used to build quadrilateral meshes in an automatic manner. There are essentially two categories of methods.

In *direct methods*, the quadrilaterals are constructed at once, either using some kind of advancing front technique [**?**] or using regular grid-based methods (quadtrees). Advancing front methods for quads are considered to be non robust and quadtree methods usually produce low quality elements close to the boundaries of the domain and are unable to fulfill general size constraints (anisotropy, strong variations).

In *indirect methods*, a triangular mesh is built first. Triangle-merge methods then use the triangles of the initial mesh and recombine them to form quadrangles [**?**, **?**]. Other more sophisticated indirect methods use a mix of advancing front and triangle merge [**?**].

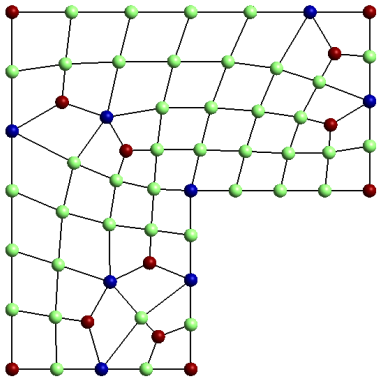


Figure 5.4: Quadrilateral mesh with 8 vertices of index $-1/4$, and 12 of index $1/4$, leading to $\chi = 12/4 - 8/4 = 1$.

5.2.1 A greedy algorithm for quad-meshing

Consider a quadrilateral element q and its the four internal angles α_k, $k = 1,2,3,4$. We define the quality (q) of q as:

$$(q) = \max\left(1 - \frac{2}{\pi}\max_k\left(\left|\frac{\pi}{2} - \alpha_k\right|\right), 0\right). \tag{5.5}$$

This quality measure is 1 if the element is a perfect quadrilateral and is 0 if one of those angles is either ≤ 0 or $\geq \pi$.

Consider a triangular mesh made of n_t triangles t_i, $i = 1,\dots n_t$. In what follows, we consider internal edges e_{ij} of the mesh that are common to triangles t_i and t_j. We define a cost function $c(e_{ij}) = 1 - (q_{ij})$ that is associated to each graph edge e_{ij} of the mesh and that is defined as the mesh quality of the quadrilateral q_{ij} that is formed by merging the two adjacent triangles t_i and t_j. Usual indirect quadrilateralization procedures work as follows [**?**]. Edges e_{ij} of the graph are sorted with respect to their individual cost functions. Then, the two triangles that are adjacent to the best edge e_{ij} of the list are recombined into a quadrilateral. Triangles t_i and t_j are tagged in order to prevent other edges that are adjacent either to t_i or to t_j to be used for another quadrilateral forming. Then, the algorithm processes the ordered list of edges, forming quadrilaterals with triangles adjacent to an edge as long as none of those adjacent triangles are tagged. Fig. 5.5 shows an illustration of this procedure for a rectangular domain of size 1×3 and a mesh size field defined by

$$h(x,y) = 0.1 + 0.08\sin(3x)\cos(6y).$$

Isolated triangles inevitably remain in the mesh and the resulting mesh is not made of quadrilaterals only. The mesh is then said to be *quad-dominant*. In the example of Fig. 5.5, the resulting mesh is made of 836 quads and 240 triangles.

A mesh composed of quadrilaterals can be build subsequently using a uniform mesh refinement procedure [**?**]. Every quadrilateral of the quad-dominant mesh is

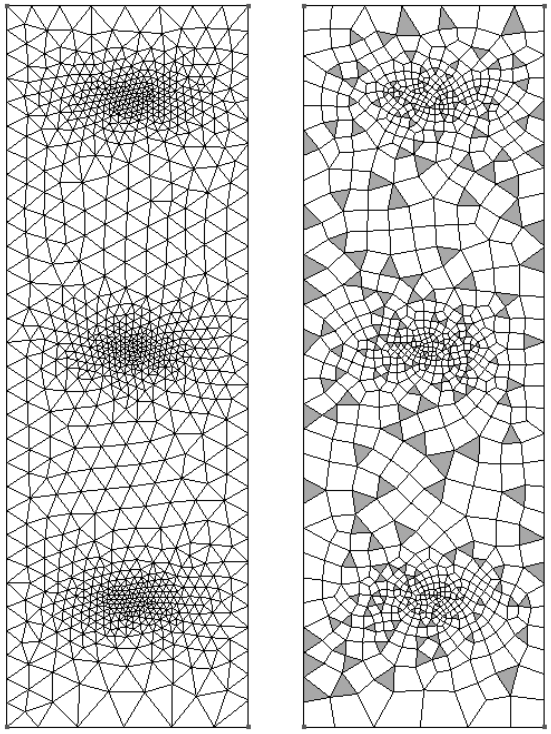


Figure 5.5: Illustration of the quad-dominant algorithm. Left mesh is the initial triangular mesh and right mesh is the quad dominant mesh, after smoothing (triangles are in grey).

split into four sub-quadrilaterals and every triangle is split into three sub-quadrilaterals (see Fig. 5.6). In order to fulfill the size criterion $h(\vec{x})$, the initial triangular mesh should thus be built using a size field with twice the value (i.e. $2h$) that is expected in the final mesh.

The recombination process just described is sub-optimal. It does not provide the best set of edges to be recombined with respect to some general cost function. Indeed, the only optimality property of this algorithm is that it ensures that the best triangle pair will be recombined.

The second part of the algorithm, namely the mesh refinement step, also has some drawbacks. Splitting every element of the mesh produces a mesh that has half the size of the initial mesh. It is of course possible to generate an initial mesh with double the required size. Yet, with real geometries, the new vertices will have to be added on the geometry, which is not trivial. On the other hand, the refinement step does not allow a sharp control of the mesh size. On Fig. 5.6, the procedure ends with an efficiency index of 79%, which cannot be considered as good.

In [**?**] the authors propose a scheme for recombining triangular meshes that does

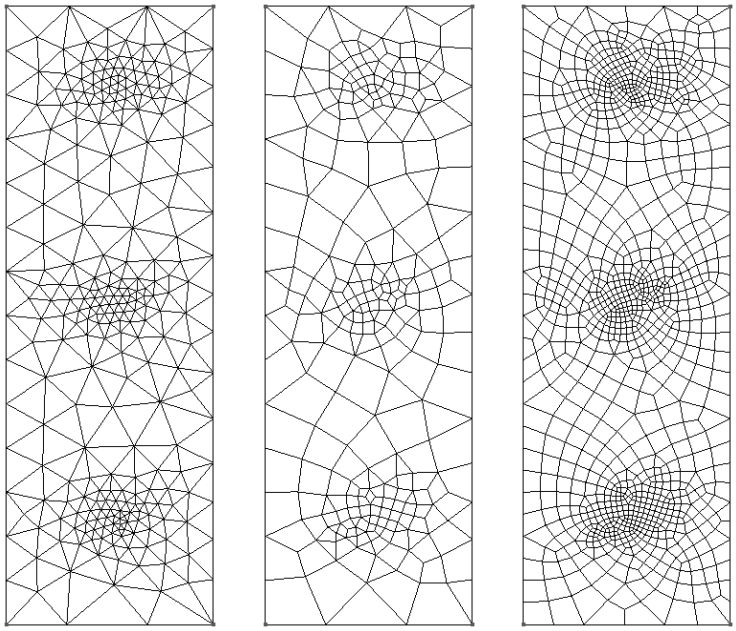


Figure 5.6: A quad dominant algorithm using $h^{\text{algo}} = 2h$ followed by a one mesh refinement procedure.

not always require the refinement step, using a kind of advancing front technique. The merging of triangles starts at the boundary; when a front closes, the algorithm attempts to maintain an even number of triangles on any sub-front. Again, this approach is sub-optimal because the result depends on the ordering of elements and on the choice of the initial front.

5.2.2 The Blossom-Quad algorithm

Here, our aim is to build a mesh generation scheme that starts with a triangular mesh and attempts to find the set of pairs of triangles that form the best possible quadrilaterals with the constraint of not leaving any remaining triangle in the mesh.

5.2.3 Blossom: a minimum cost perfect matching algorithm

Let us consider $G(V,E,c)$ an undirected weighted graph. Here, V is the set of n_V vertices, E is the set of n_E undirected edges and $c(E) = \sum c(e_{ij})$ is an edge-based cost function, i.e., the sum of all weights associated to every edge $e_{ij} \in E$ of the graph. A *matching* is a subset $E' \subseteq E$ such that each node of V has at most one incident edge in E'. A matching is said to be perfect if each node of V has exactly one incident edge in E'. As a consequence, a perfect matching contains exactly $n_{E'} = n_V/2$ edges.

A perfect matching can therefore only be found for graphs with an even number of vertices. A matching is optimum if $c(E')$ is minimum among all possible perfect matchings.

In 1965, Edmonds [**?**, **?**] invented the *Blossom algorithm* that solves the problem of optimum perfect matching in polynomial time. A straightforward implementation of Edmonds's algorithm requires $\mathcal{O}(n_V^2 n_E)$ operations.

Since then, the worst-case complexity of the Blossom algorithm has been steadily improving. Both Lawler [**?**] and Gabow [**?**] achieved a running time of $\mathcal{O}(n_V^3)$. Galil, Micali and Gabow [**?**] improved it to $\mathcal{O}(n_V n_E \log(n_V))$. The current best known result in terms of n_V and n_E is $\mathcal{O}(n_V(n_E + \log n_V))$ [**?**].

There is also a long history of computer implementations of the Blossom algorithm, starting with the Blossom I code of Edmonds, Johnson and Lockhart [**?**]. In this paper, our implementation makes use of the Blossom IV code of Cook and Rohe [**?**][1], which has been considered for several years as the fastest available implementation of the Blossom algorithm.

5.2.4 Optimal triangle merging

Consider now a mesh made of n_t triangles and n_v vertices. Consider a specific weighted graph $G(V, E, c)$ that is build using triangle adjacencies in the mesh. Here, every vertex of the graph is a triangle t_i of the mesh and every edge of the graph is an internal edge e_{ij} of the mesh that connects two neighboring triangles t_i and t_j. Fig. 5.7 shows a simple triangular mesh with its graph and one perfect matching.

Let us come back first to the non-optimal triangle merging algorithms of §5.2.1. In term of what has just been defined, the subset E' of edges that have been used for triangle merging in the approach of [**?**] is a matching that is very rarely a perfect matching. The one of [**?**] is usually a perfect matching, but not necessarily the optimal one.

Here, we propose a new indirect approach to quadrilateral meshing that takes advantage of the Blossom algorithm of Edmonds. To this end we apply the Blossom IV algorithm to the graph of the mesh. We intend to find the optimum perfect matching with respect to the following total cost function

$$c = \sum_{e \in E'} (1 - (q_{ij})), \tag{5.6}$$

that is, the sum of all elementary cost functions (or "badnesses") of the quadrilaterals that result in the merging of the edges of the perfect matching E'.

An obvious requirement for the final mesh to be quadrilateral only is that the initial triangular mesh contains an even number of triangles (i.e., an even number of graph vertices). Euler's formula for planar triangulations states that the number of triangles in the mesh is

$$n_t = 2(n_v - 1) - n_v^b, \tag{5.7}$$

where n_v^b is the number of mesh nodes on its boundary. So, the number of mesh points on the boundary n_v^b should be even. Here our algorithms are applied to general solid models that have a boundary representation (BRep) [**?**]. This means that

[1] Computer code available at http://www2.isye.gatech.edu/~wcook/blossom4/.

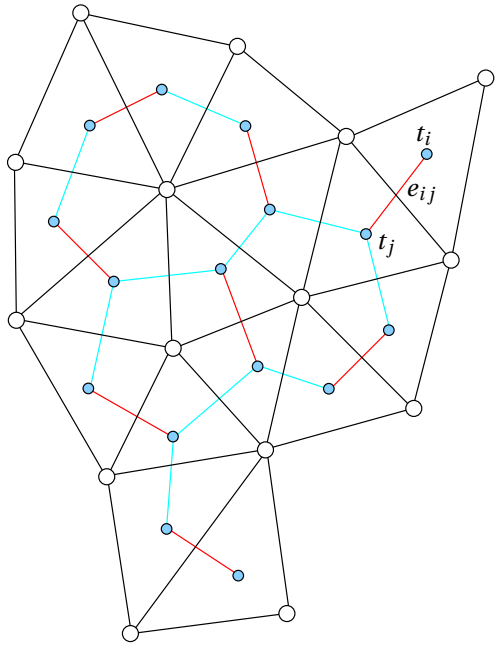


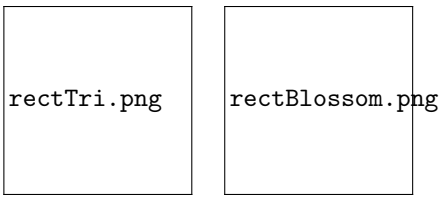
Figure 5.7: A mesh (in black) and its graph (in cyan and red). The set of graph edges colored in red forms a perfect matching.

model surfaces are bounded by connected model edges that form edge loops and that the model edges are bounded by model vertices. The mesh vertices of a model edge $n_v^{b_i}$ are defined as the mesh vertices on that edge minus the model vertices. The total number of mesh points on the boundary n_v^b can thus be written as:

$$n_v^b = \sum_{i=0}^{N_E} (1 + n_v^{b_i}). \tag{5.8}$$

It is then easy to see that for n_v^b to be even, it is sufficient for $n_v^{b_i}$ to be odd. This means that a sufficient condition for having an even number of triangles in the mesh is to have every model edge b_i discretized with an odd number of mesh vertices.

Fig. 5.8 shows the same illustrative example as Figures 5.5 and 5.6 using the Blossom algorithm for recombining the triangles together with the optimization procedure that will be described in §**??**. The final result is not only much better with respect to the efficiency index (with $\tau = 83\%$), but also with respect to worst element quality ($_w = 0.405$ instead of $_w = 0.310$ for the mesh of Fig. 5.6). The average quality is better as well.



$\tau = 88.9\%$ $\tau = 83.9\%$

Figure 5.8: Illustration of the Blossom-Quad algorithm.