Optimisation de maillages

Jane Tournois

UNIVERSITÉ DE NICE - SOPHIA ANTIPOLIS

ÉCOLE DOCTORALE STIC
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE

pour obtenir le titre de

Docteur en Sciences

de l'Université de Nice - Sophia Antipolis
Mention : Informatique

présentée et soutenue par

Jane Tournois

Optimisation de Maillages

Mesh Optimization

Thèse dirigée par Pierre Alliez

soutenue le 4 novembre 2009

Jury :

Mr Pierre Alliez	Chargé de Recherche, INRIA	*Directeur*
Mme Dominique Attali	Chargée de Recherche, CNRS	*Examinatrice*
Mr Stéphane Lanteri	Directeur de Recherche, INRIA	*Président*
Mr Bruno Lévy	Directeur de Recherche, INRIA	*Rapporteur*
Mr Niloy Mitra	Associate Professor, KAUST	*Examinateur*
Mr Alain Rassineux	Professeur des Universités, UTC	*Rapporteur*
Mme Mariette Yvinec	Chargée de Recherche, INRIA	*Invitée*

Remerciements

Je tiens à remercier tout particulièrement Pierre Alliez pour avoir encadré mon travail depuis le Master et pendant ces années de thèse. Sa disponibilité, son expérience, ses connaissances scientifiques dont il a su me faire profiter, son soutien et ses conseils ont grandement contribué au bon déroulement de ma thèse.

Je remercie toute l'équipe Geometrica pour l'accueil qui m'a été offert dès mon arrivée dans l'équipe. L'ambiance sympathique, amicale et propice au travail, ainsi que pour la bonne humeur quotidienne qui y règnent ont été très importantes pour moi. L'organisation des "thés et après-thés" hebdomadaires en sont le meilleur exemple. Merci aussi à l'ensemble des membres de Geometrica pour l'aide qu'ils ont pu m'apporter au travers des différentes discussions que nous avons eues ensemble. Je remercie également Agnès Bessière et Caroline French, assistantes de l'équipe, pour leur aide précieuse.

J'exprime ma gratitude envers toutes les personnes avec qui j'ai eu la chance de travailler sur des publications : Pierre Alliez, Olivier Devillers, Camille Wormser, Mathieu Desbrun, Pedro Machado et Rahul Srinivasan.

Je remercie également Mathieu Desbrun et toute son équipe pour leur accueil lors de ma visite à Caltech, ainsi que pour leurs encouragements pendant la conférence SIGGRAPH'2009, à laquelle nous avons participé ensemble.

Je remercie mes collègues moniteurs et enseignants de l'École Polytechnique Universitaire de Nice Sophia Antipolis pour leur accueil, la confiance qu'ils m'ont accordée et les échanges que nous avons eus. Merci pour tout, en particulier à Claudine Peyrat, Pierre Bernhard, Vincent Granet et Sébastien Mosser, avec qui j'ai eu l'opportunité de collaborer pour mes enseignements.

Je suis reconnaissante à Bruno Lévy et Alain Rassineux d'avoir accepté d'être rapporteurs de ma thèse. Leurs rapports et les échanges que nous avons eus m'ont permis d'envisager de nouvelles pistes de réflexion.

Je remercie tous les membres de mon jury de thèse, Pierre Alliez, Dominique Attali, Stéphane Lanteri, Bruno Lévy, Niloy Mitra, Alain Rassineux et Mariette Yvinec, de m'avoir fait l'honneur d'assister à ma soutenance, pour leurs questions au cours de celle-ci, et les discussions que nous avons eues.

Je tiens également à remercier Marie Samozino, François Grimbert, Thierry Viéville et Mireille Bossy pour m'avoir donné la chance de représenter les doctorants, au cours de mes trois années de thèse, au Comité de Suivi Doctoral (CSD) de l'INRIA Sophia Antipolis Méditerranée. Cette mission, que j'ai partagé avec Christelle Molle, est importante et a été très enrichissante pour moi.

J'adresse un amical "merci" à Pooran Memari avec qui j'ai partagé mon bureau, pour tous les échanges tant scientifiques que culturels, linguistiques et amicaux que nous avons eus.

Mes derniers remerciements vont à ma famille et mes amis, qui croient en moi et sont toujours là pour m'encourager. Merci à Sébastien de m'avoir supportée et aidée depuis le début.

Contents

Chapter 1

Introduction

Contents

1.1 Motivations

Meshing a domain consists in defining a concise set of simple elements whose non-overlapping union best describes the domain and its boundaries, while satisfying a series of criteria on element shapes and sizes. These simple elements are typically triangles or quadrilaterals in 2D, and tetrahedra, hexahedra or other polyhedra in 3D.

Meshes are a key component in many domains of application, as various as visualization [LM98, LP01], simulation [KTY09] (for robotical surgery for example [CDA96, DPS+06]), animation [MCP+09], architecture [PAH+07], and CAD [Fle99]. They also are some applications such as surface reconstruction [HDD+92] where meshes are used as core data structures. See Figure 1.1 for examples.

Most ubiquitous in computer animation and computational sciences is the need for unstructured isotropic tetrahedral meshes. These versatile geometric representations are used in finite element and finite volume simulations of physical phenomena as varied as material deformation [KMOD09], heat transfer [ATP84],

and electromagnetic effects [PSB+07].

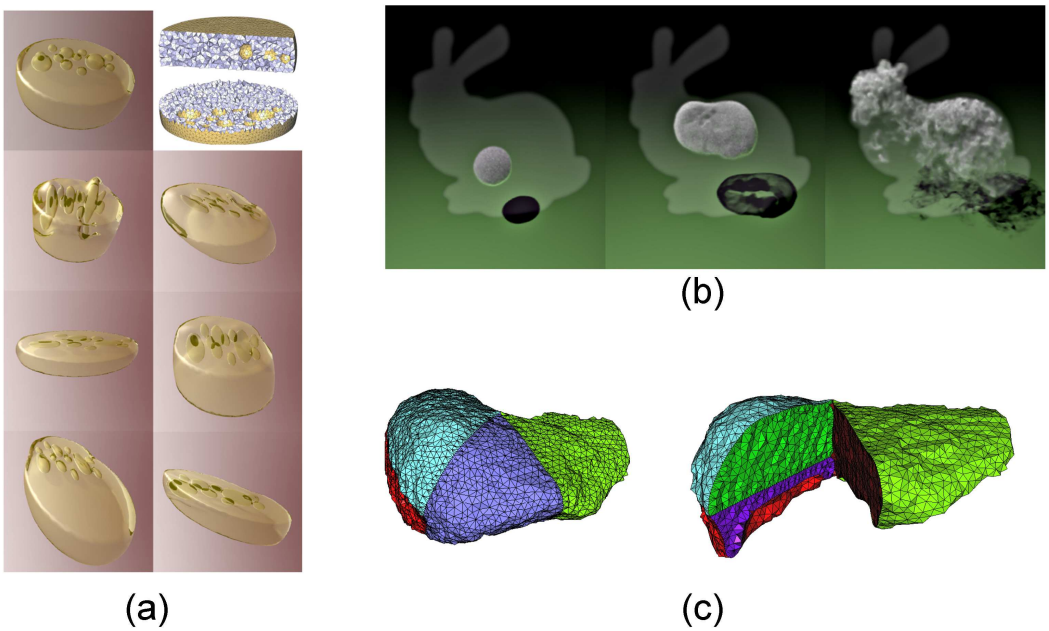


Figure 1.1: Applications. (a) Simulation of physical deformations. A falling cheese mesh [KMOD09]. (b) Computational Fluid Dynamics for animation. Smoke behavior in a bunny-shaped domain [MCP+09]. (c) Biomedical applications. Mesh generated from a segmented liver image representing 4 anatomical regions [BYB09].

As the accuracy and stability of such computational endeavors heavily depend on the shape of the worst element [She02d], mesh element quality is a priority when conceiving a mesh generation algorithm.

Automatic mesh generation is still a challenging problem, both from a theoretical and a practical point of view. Though the literature dealing with meshing is very rich, a lot of work remains to be done in order to reach the "perfect mesh generator". The wide range of input types and application domains makes the problem difficult to solve in its whole. The scientific motivation is to get an algorithm (or several algorithms) that provably terminates, whatever type of input is given, and that generates optimal output meshes (*e.g.* with the optimal number of elements). Turning towards application fields, the challenge is to get a practical algorithm that is fast, robust, and generates meshes of high quality with regard to the application requirements.

Owen [Owe98] gives a survey of many fundamental unstructured mesh generation algorithms, including triangle and quadrilateral meshing in 2D and in 3D. Frey and George [FG07] complete this work with more recent algorithms, and theoretical

aspects of mesh generation.

In this thesis, we focus on the problem of generating quality unstructured isotropic meshes. Our main concern stands in generating meshes of high quality in terms of angles (dihedral angles in 3D), such that all simplices are close to the regular simplex, with a reasonable number of vertices.

1.2 Fundamentals

Let $\mathcal{P} = \{p_i\}_{i=1..n}$ be a set of vertices in $\mathbb{R}^d$.

Voronoi diagram. The *Voronoi diagram* associated to $\mathcal{P}$, denoted $\mathcal{V}(\mathcal{P})$, is a simplicial decomposition of $\mathbb{R}^d$ into d-dimensional subspaces called *Voronoi cells* and denoted $\mathcal{V}(p_i) = \mathcal{V}_i$. Each *Voronoi cell* $\mathcal{V}_i$ is composed of the set of points of $\mathbb{R}^d$ which are closer from p_i than from any other point in $\mathcal{P}$:

$$\mathcal{V}_i = \{p \in \mathbb{R}^d : \forall j \neq i, \quad d(x, p_i) \leq d(x, p_j)\}.$$

$\mathcal{V}_i$ can also be considered as the intersection of the $n-1$ half-spaces bounded by the bisector planes of segments $[p_i p_j]$, $j \neq i$. $\mathcal{V}_i$ is therefore a convex polytope, possibly unbounded. The points of $\mathcal{P}$ are called *generators*.

In two dimensions, *Voronoi edges* are the edges shared by two Voronoi cells, and *Voronoi vertices* are the points shared by three Voronoi cells. In three dimensions, *Voronoi facets, edges and vertices* are the geometric objects shared by one, two and three Voronoi cells, respectively.

Delaunay triangulation. The *Delaunay triangulation* $\mathcal{DT}(\mathcal{P})$ of $\mathcal{P}$ is the geometric dual of $\mathcal{V}(\mathcal{P})$. $\mathcal{DT}(\mathcal{P})$ contains the edge $[p_i p_j]$ if and only if the intersection of $\mathcal{V}(p_i)$ with $\mathcal{V}(p_j)$ is not empty. A triangulation $\mathcal{T}$ of $\mathcal{P}$ can be described this way. $\mathcal{T}$ describes a partition of the convex hull of $\mathcal{P}$ into d-dimensional simplices (*i.e.* triangles in 2D, tetrahedra in 3D, etc).

The *Delaunay triangulation* can also be defined through the empty circle (resp. empty sphere in 3D) property [Del34]. A triangle (resp. a tetrahedron) is in the Delaunay triangulation if and only if its circumcircle (resp. circumsphere) does not contain any other points of $\mathcal{P}$ in its interior (see Figure 1.2).

Using this type of triangulation is very convenient since it is canonically defined through the locations of its vertices. We only have to care about the positions of

the vertices, and not about the connectivity. Moreover, the Delaunay framework is powerful for guaranteed-quality mesh generation.

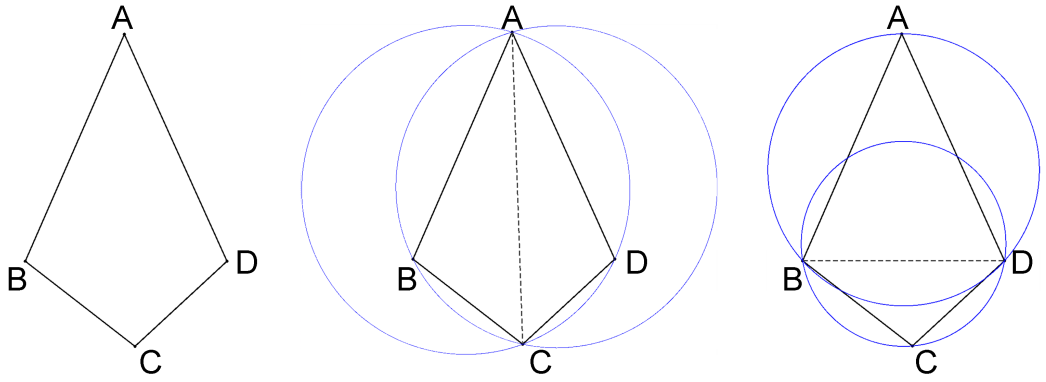


Figure 1.2: Delaunay triangulation empty sphere property. (Left) Four points and their convex hull. (Middle) This triangulation does not meet the empty-sphere Delaunay condition (circumcircles contain more than 3 vertices). (Right) *Flipping* an edge turns it into a Delaunay triangulation for the four points.

Each simplex in the Delaunay triangulation is dual to a Voronoi simplex. In 2D, the dual of a Delaunay triangle is the Voronoi vertex which also is the circumcenter of the triangle. In 3D, each Delaunay tetrahedron is the dual of a Voronoi vertex which also is the tetrahedron's circumcenter. The dual of a Delaunay facet is a Voronoi edge, the dual of a Delaunay edge is a Voronoi facet, and the dual of a Delaunay vertex p_i is the Voronoi cell $\mathcal{V}_i$. Figure 1.3 illustrates this dual relationship in 2D.

Restricted Delaunay triangulation. The Delaunay triangulation of $\mathcal{P}$ forms a triangulation of the convex hull of $\mathcal{P}$. In most applications, the only tetrahedra in which the user is interested are the ones lying inside the domain Ω, or on its boundary $\partial_2\Omega$ for surface meshes (see Figure 1.4).

Let S be a subset of $\mathbb{R}^d$. In general, S is a manifold of dimension $k \leq d$ (in 3D, usually a volume or a surface in $\mathbb{R}^3$). The *restricted Delaunay triangulation* of $\mathcal{P}$ to S, denoted by $\mathcal{DT}|_S(\mathcal{P})$, is a sub-complex of $\mathcal{DT}(\mathcal{P})$ composed of the so-called *restricted* simplices. A simplex is said to be *restricted* to S if its dual Voronoi simplex intersects S. For example, in 2D, the Delaunay triangulation restricted to a curve ∂Q is composed of the Delaunay edges whose dual Voronoi edges intersect Q. Similarly, the Delaunay triangulation restricted to the polygonal region Q defined by ∂Q is composed of the Delaunay triangles whose circumcenters are inside Q.

It can be shown that the restricted Delaunay triangulation forms a good approximation of the object. Under some assumptions, and in particular if $\mathcal{P}$ is

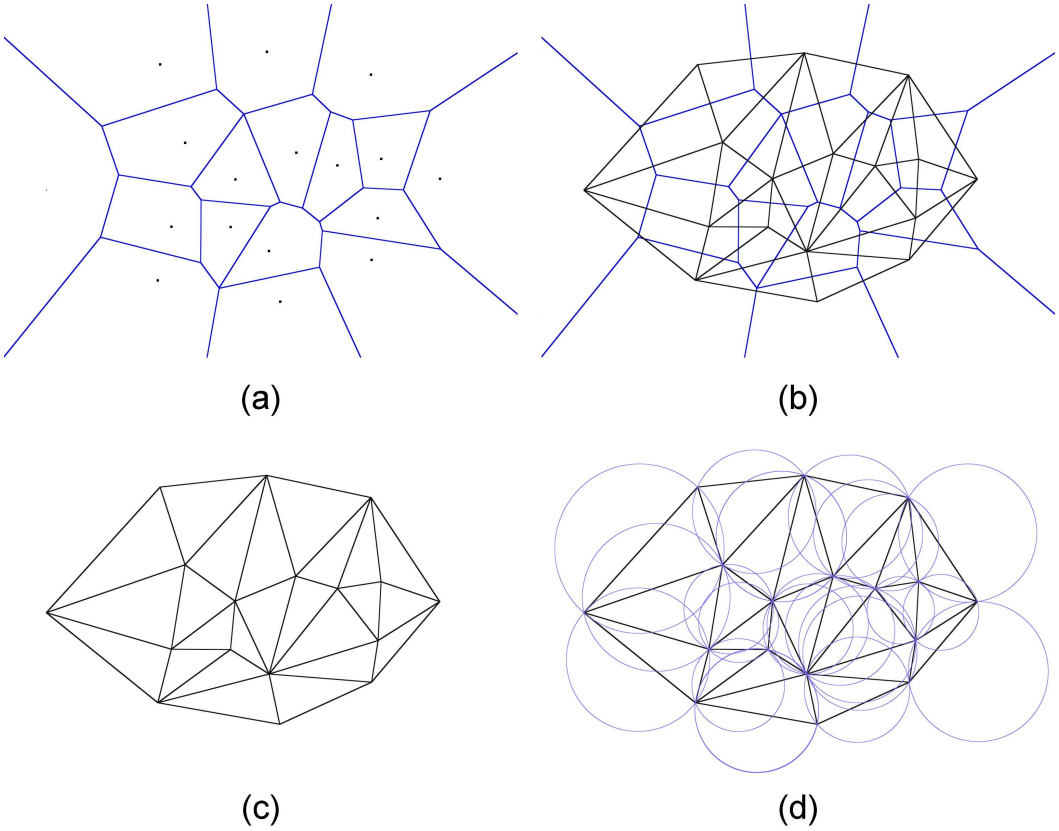


Figure 1.3: The Voronoi diagram of a set of points (a) is dual (b) to the Delaunay triangulation (c), which satisfies the empty sphere property (d).

a sufficiently dense sample of a domain Ω [AB99], $\mathcal{DT}|_\Omega(\mathcal{P})$ is a good approximation of Ω. From a topological point of view, $\mathcal{DT}|_\Omega(\mathcal{P})$ is homeomorphic to Ω. From a geometrical point of view, the Hausdorff distance between $\mathcal{DT}|_\Omega(\mathcal{P})$ and Ω can be made arbitrarily small, and normals and curvatures can be consistently approximated from the restricted triangulation. Thanks to these approximation properties, some provably correct algorithms have been developed in the last decade for mesh generation and mesh reconstruction from point sets [BO05].

From an implementation point of view, (in particular in 3D) computing all the Boolean tests needed to select the restricted simplices and the intersection points already is a difficult task. The predicates and point constructions need to be fast and robust. They are responsible for robustness and efficiency of the whole refinement process.

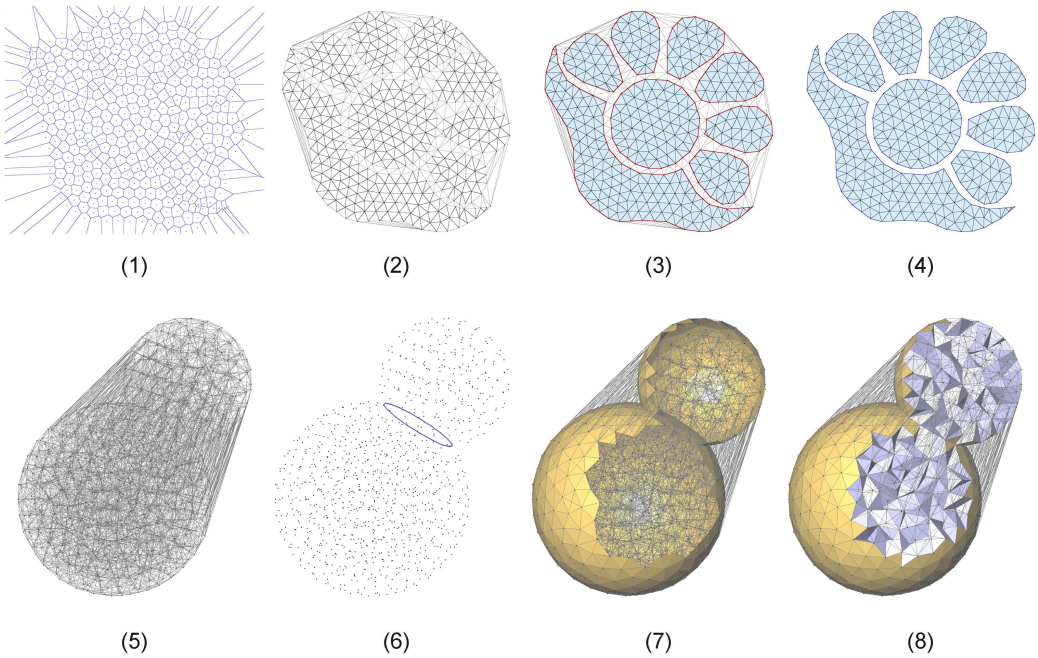


Figure 1.4: Restricted triangulations. (1) Voronoi diagram of a set of points in the plane. (2) Dual Delaunay triangulation. (3) Delaunay triangulation restricted to the boundary of the blue domain (depicted in red). (4) Delaunay triangulation restricted to the blue domain (depicted with filled blue triangles). (5) Delaunay triangulation of a set of points in $\mathbb{R}^3$. (6) Delaunay triangulation restricted to a curve (depicted with a blue line). (7) Delaunay triangulation restricted to the boundary of two intersecting spheres. (8) Delaunay triangulation restricted to two intersecting spheres.

1.3 Problem statement

This thesis addresses the problem of generating high quality Delaunay triangulations, in two and three dimensions. Triangle mesh generation can be considered as a succession of point insertions inside the domain to be discretized, along with connectivity updates defining the simplices of the mesh. We choose to stay within the Delaunay framework, which defines the connectivity in a canonical manner. In this way, we can focus on vertex locations and number.

This discretization - or meshing - problem takes as input the definition of the domain to be discretized. User-defined criteria are also taken as input. For example, the user can provide a sizing function bounding the maximal size of simplices everywhere in the domain. It can be uniform or not, and gradation can be tuned too. Shape of simplices, topology of the output boundary and approximation error can also be controlled through user-defined criteria. The output mesh has to

satisfy all these criteria, when they are required by the user. Vertex insertions and relocations are used to achieve this goal.

We need to define what is a quality mesh, and here within our framework a good Delaunay triangulation. Simplices quality and number are the main characteristics defining a good mesh. Many criteria have been used to define a quality mesh.

1.3.1 Mesh quality

Many finite element methods require discretizing a domain into a set of tetrahedra. These applications require more than just a triangulation of the domain for simulation and rendering. The accuracy and the convergence of these methods depend on the size and shape of the elements apart from the fact that the mesh should conform to the domain boundary [She02d]. Both the bad quality and the large number of the mesh elements can negatively affect the execution of a simulation. It is required that all elements of the mesh are well-shaped as the accuracy of the simulations and computations can be compromised by the presence of a single badly shaped element. In general it is desirable to bound the smallest dihedral angle in the mesh. Large dihedral angles can be responsible for large discretization and interpolation errors [LS07], and compromise the accuracy of numerical simulations [Jam76, Kri92, She02d, BA76]. Small dihedral angles are responsible for the ill-conditioning of matrices associated with the finite element methods [BS89, She02d].

The Delaunay refinement technique guarantees a bound on the radius-edge ratio of all mesh elements, which is the ratio of the circumradius to the shortest edge length of a triangle or a tetrahedron. Although in 2D this translates into a lower bound on the minimum angle in the mesh, in 3D it does not: A bound on the radius-edge ratio is not equivalent to a bound on the smallest dihedral angle.

1.3.1.1 Triangle quality (2D)

In 2D, the **radius-edge ratio** ρ of a triangle is linked to its minimum angle θ_{min} by the formula $\rho = \frac{1}{\sin \theta_{min}}$. Moreover, Delaunay refinement is providing guarantees on the radius-edge ratios of output simplices, by satisfying the user-defined criteria (typically a radius-edge ratio for triangle shape). When refinement terminates, an upper-bound is available on the radius-edge ratios of simplices. This way, bounding the triangulation angles from below becomes easy.

The **radius ratio**, computed as the ratio of inradius over circumradius of a triangle, can also be used as a quality criterion. However, this criterion is not used for defining which triangle should be refined, and no guarantee on this value is given on the output mesh. It can be considered a post-meshing quality criterion.

1.3.1.2 Tetrahedron quality (3D)

Several tetrahedron quality criteria have been defined and used in the literature depending on the application.

In Delaunay-based isotropic mesh generation, one wants to avoid the creation of bad tetrahedra (see Figure 1.5). Among them, a *sliver* is a tetrahedron that has its four vertices close to the equator of a sphere, and can have dihedral angles arbitrarily close to 0 and to π, and is usually considered as the worse kind of tetrahedron. As we have seen, a single bad tetrahedron in the final mesh can compromise a simulation method convergence. Let us list some tetrahedron quality criteria.

- The **radius edge ratio** ρ of a simplex. This measure, which is minimal for the regular tetrahedron, unfortunately cannot detect slivers, though it is used in Delaunay refinement algorithms to define bad simplices.

- The **radius ratio**, defined as the ratio of the inradius (insphere radius) to the circumradius (circumsphere radius), is another popular measure of tetrahedron quality. It is desired to ensure that radius ratio of all tetrahedra are bounded from below by a constant.

- Denote the volume of tetrahedron $pqrs$ by V and its shortest edge length by l. The **volume per cube of shortest edge length** ($\sigma = \frac{V}{l^3}$) is used as a measure of the shape quality along with the radius-edge ratio ρ, or on its own [CDE+00]. According to Li [Li00a], a tetrahedron $pqrs$ is called *sliver* if $\rho(pqrs) \leq \rho_0$ and $\sigma(pqrs) \leq \sigma_0$, where ρ_0 and σ_0 are constant.

- The **minimum dihedral angle** θ_{min} is also used as a criterion for quality mesh generation. In the sequel, we choose this measure to evaluate the mesh quality as it is more intuitive and geometrically meaningful than, e.g., the radius ratio, which combines the six dihedral angles of a tetrahedron.

It can be shown that a lower bound on the radius ratio is equivalent to a lower bound on the minimum dihedral angle.

Consider an arbitrary tetrahedron τ with triangular faces T_1, T_2, T_3, T_4. Let the areas of these triangles be denoted by S_1, S_2, S_3, S_4 respectively, the dihedral angle between T_i and T_j by θ_{ij} and the length of the edge shared by T_i and T_j by l_{ij}. The volume V of τ is given by

$$V = \frac{2}{3l_{ij}} S_i S_j \sin\theta_{ij} \quad \text{for } i \neq j \text{ in } \{1,2,3,4\}. \tag{1.1}$$

Let r_C, r_I be the circumradius and inradius of τ and r_i be the circumradius of T_i for i in $\{1,2,3,4\}$. We know that for any tetrahedron, $r_i \leq r_C$. This gives $S_i \leq \pi\, r_i^2 \leq \pi\, r_c^2$, and we also have a bound on the volume $V \geq \frac{4}{3}\, \pi r_I^3$.

Using Equation 1.1, for $i \neq j$ we have

$$\frac{4}{3}\pi r_I^3 \leq \frac{2}{3l_{ij}} S_i S_j \sin\theta_{ij} \leq \frac{2}{3l_{ij}} \pi^2 r_c^4 \sin\theta_{ij},$$

and we get

$$sin\theta_{ij} \geq \frac{2}{\pi} \cdot \frac{r_I^3}{r_c^3} \frac{l_{ij}}{r_C} \geq \frac{2}{\pi} \cdot \frac{a_0^3}{\rho_0},$$

where a_0 is the radius-radius ratio and ρ_0 is the radius-edge ratio. Finally,

$$\theta_{ij} \geq sin^{-1}\left(\frac{2}{\pi} \cdot \frac{a_0^3}{\rho_0}\right).$$

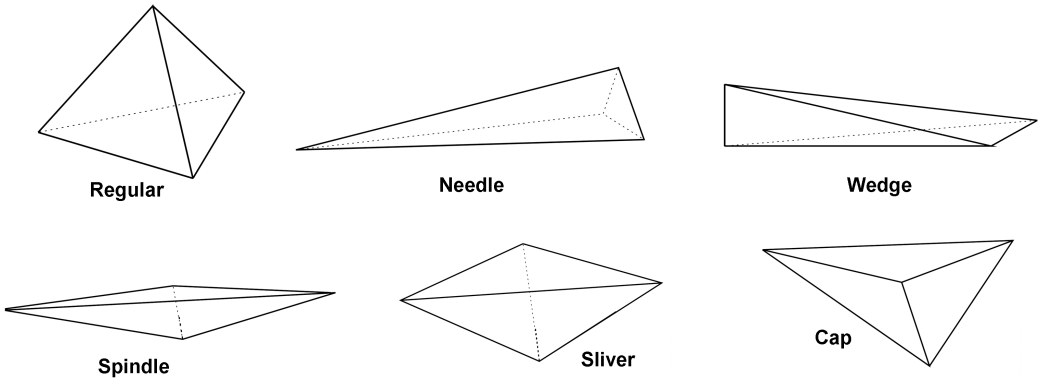


Figure 1.5: Tetrahedra shapes [BCER95, Epp01]. A regular tetrahedron is well shaped and has its dihedral angles close to 70.5°. A sliver has its four vertices close to a circle, four very small dihedral angles (close to 0°), and two very large (close to 180°). Each of the other tetrahedra presents a different type of degeneracy.

Parthasarathy et al. describe more tetrahedron quality criteria [PGH94]. They provide some experimental results on the stability of these quality measures under different types of vertex perturbations, and a comparison of their computational cost.

1.3.2 Mesh complexity

It is clear that the number of simplices is important in all applications, as the computational cost is directly linked to the number of simplices on which a value needs to be computed.

We make the observation that filling in a domain with elements that are both well-shaped and well-sized (*i.e.* as big as possible) requires fewer elements, by a simple packing argument. Similarly, dealing with a well-spaced point set (*i.e.* no

two simplices are too close together, and the domain is covered by vertices) has nice properties [Epp01]. For example, in 2D, the constrained Delaunay triangulation of a well-spaced point set has a bounded aspect ratio (longest edge length divided by shortest altitude length) [Rup93]. Mitchell shows that a bound on angles can be turned into lower and upper bounds on the cardinality of the triangulation [Mit94].

We aim to make the most of these observations in order to obtain meshes with few elements, both well-shaped and well-sized.

1.4 State-of-the-art

We are giving here a short survey of the mesh refinement and mesh optimization methods known in the literature. A more extensive study will be presented, for 2D in Chapter 2 and for 3D in Chapter 3.

1.4.1 Refinement

Pioneered by Lawson [Law77] and Watson [Wat81], Delaunay triangulation algorithms are now of common use in the mesh generation field. The Delaunay criterion itself only defines the connectivity of the triangulation, from a given point set. Next to connectivity, the main problem to consider is how to insert vertices in the triangulation to improve simplices quality, and reach some user-defined criteria. This point insertion problem is called *refinement*.

Refinement algorithms have been extensively studied in the literature. They are amenable to analysis, and hence are reliable algorithms. In addition, the robust implementations of Delaunay triangulations which are now available greatly facilitate the implementation of Delaunay-based mesh refinement algorithms.

Using a simple probing device, refinement can also be considered as a good technique for learning smooth objects by probing, as done by Boissonnat et al. [BGO05]. In 2D as in 3D, the output mesh is shown to be a good approximation of the smooth object under some conditions. Moreover, Shewchuk showed that the output mesh can conform to some input geometric constraints [She02a] by a bounded number of vertex insertions.

Delaunay refinement Delaunay refinement algorithms, first introduced in 2D by Chew [Che89b], Ruppert [Rup93, Rup95], and Shewchuk [SMD97] have been extensively studied in the last decades. Chew [Che97], Li and Teng [LT01], Rineau and Yvinec [RY07] propose Delaunay refinement algorithms. They provide

guarantees on the output meshes, satisfying some user-defined criteria.

These greedy algorithms are know to terminate when there is no acute dihedral angles in the input. For example, Rineau and Yvinec [RY07] provide a proof of termination of their Delaunay tetrahedral mesh generation algorithm. The main idea is to prove that when a new element is created, its size is bounded from below. By a packing argument, the number of inserted vertices is bounded too.

To iteratively improve the quality of the current mesh and reach the user-defined criteria, these greedy Delaunay refinement algorithms insert vertices inside the domain and on the boundary of the domain to be meshed. Usually, the inserted points are bad edge midpoints, and bad facet and tetrahedron circumcenters. Some variants have been developed to obtain output meshes of higher quality.

Algorithm 1 Classical Delaunay refinement

Input: A domain $\Omega \subset \mathbb{R}^3$.
  Insert 3 points on each surface patch to initialize refinement.
  while There is a bad edge e restricted to $\partial_1\Omega$, **do**
    refine(e).
  while There is a bad facet f restricted to $\partial_2\Omega$, **do**
    refine(f).
  while There is a bad cell c restricted to Ω, **do**
    refine(c).
Output: The final triangle mesh.

Algorithm 1 gives an overview of a classical greedy Delaunay refinement algorithm, in 3D. Each **refine**(.) operation consists in computing the point to be inserted, and insert it into the triangulation. Refinement of facets and cells are conditioned by encroachment. First defined in 2D [Rup95] and extended to 3D [She98], encroachment handling is necessary for the output mesh to conform the input boundary. To describe the refinement process of each type of boundary simplex, we need to define the *surface Delaunay ball*.

Definition 1.4.1. *The* surface Delaunay ball *of a facet f (resp. an edge) restricted to a subset $\Omega \in \mathbb{R}^3$ is the ball circumscribing f and centered at the intersection point of f's dual Voronoi object with Ω.*

Let p_c be a candidate point to insertion. For an edge e restricted to $\partial_1\Omega$, **refine**(e) computes p_c as e's midpoint (or surface Delaunay ball on the boundary), and inserts it. For a facet f restricted to $\partial_2\Omega$, **refine**(f) computes p_c as the intersection point between f's dual Voronoi edge and $\partial_2\Omega$. If this point is in the surface Delaunay ball $\mathcal{B}$ of an edge restricted to $\partial_1\Omega$, p_c is discarded and $\mathcal{B}$'s center

is inserted. Similarly, for a cell c restricted to Ω, **refine**(c) computes p_c as c's circumcenter. If this point is in the surface Delaunay ball $\mathcal{B}'$ of a facet restricted to $\partial_2\Omega$, then p_c is set to be $\mathcal{B}'$'s center, and this facet is refined, as previously.

However, most Delaunay refinement algorithms fail at removing all badly-shaped tetrahedra, and a special class of almost-flat tetrahedra (so-called slivers) may remain in the triangulation. The slivers, with dihedral angles close to 0 and to π, are problematic for many numerical simulations.

Some variants of the standard Delaunay refinement algorithm have been developed. Some of them are based on the same algorithm, but define other Steiner points for every type of simplex. For example, Üngör defines the *off-centers* as locally optimal Steiner points [Üng04], to get output meshes of higher quality (see Section 2.1.1 for more details). Chernikov and Chrisochoides improve this idea [CC06] by defining a complete area, called *selection disk*, where the Steiner point can be chosen. For a bad simplex, inserting as Steiner point a point chosen in this area will for sure improve the quality of the mesh.

Weighted Delaunay refinement Initially based on Cheng et al.'s [CDRR04] idea of "protecting" the sharp angle edges from over-refinement, one of the greedy Delaunay refinement variants is based upon the idea of using a regular triangulation, *i.e.* a weighted Delaunay triangulation [CDR07, CDL07]. This approach makes possible to handle sharp input edges, with a proof of termination.

In presence of edges supporting dihedral angles smaller than 90°, weights are inserted on Steiner points of the input sharp edges, to "protect" sharp edges from being responsible for the infinite looping of classical greedy Delaunay refinement algorithms. The basic algorithm consists in first, protecting the sharp edges with weighted vertices, and then run a classical Delaunay refinement algorithm while replacing circumcenters by orthocenters.

Using weighted Delaunay triangulations is difficult. Indeed, one has to deal with hidden points that can appear and disappear depending on their neighbourhood. This becomes even more difficult when optimization comes into play. In Section 4.2.3, we will present some ideas for optimizing vertex locations along with their weights.

Grid-based mesh generation Grid-based mesh generation has also been the subject of numerous studies. Octree algorithms were introduced by Yerry and Shepard [YS84], followed by Baker et al. [BGR88], Bern et al. [BEG90], Mitchell and Vavasis [MV96], Fuchs [Fuc98] and Naylor [Nay99]. Naylor developed the idea of body centered cubic (BCC) lattice. Since then, the BCC lattice and octrees have

been used in grid-based tetrahedron mesh generation to improve the quality of output tetrahedra.

The more recent provably good algorithm was proposed by Labelle and Shewchuk [LS07] to mesh domains bounded by smooth boundaries, while providing theoretical bounds on their output minimal and maximal dihedral angles.

The methods generating meshes based on a regular grid have several drawbacks. First, they can only handle smooth surfaces. Second, the grading of elements sizes inside the mesh is not good enough for many simulation purposes. Finally, since three directions (corresponding to the initial grid axes) are clearly favored, the generated meshes can not be considered as isotropic. In simulation, the use of structured grids can lead to undesired severe stair-casing or aliasing effects [WBOL07].

Unstructured grids conforming to input structures (anatomical data, mechanical parts, etc) are very suitable to finite element simulations [PSB+07, SCL+06].

Other types Some other types of triangulations, with non-Delaunay connectivity, have also been developed. Among them, the max-min solid angle triangulation [Joe91] is computed from a Delaunay triangulation by modifying its connectivity. When the points are fixed, the solid angles are as good as they can be. However, there is no direct link between a solid angle value and the quality of the corresponding tetrahedron dihedral angles values.

According to Frey and George [FG07] a mesh is said to be *optimal* when each edge in the mesh is of length 1 (according to the given metric), and each simplex volume is $\frac{\sqrt{d+1}}{d!\sqrt{2^d}}$. This optimality criterion has to be balanced with a minimal number of vertices inside the mesh.

1.4.2 Optimization

To improve the quality of a mesh, one can decide to insert vertices (see Section 1.4.1). However, when the required point density is already met, inserting more points would lead to over-refining the mesh. Another approach consists in modifying the vertex characteristics to improve, at each vertex locally, the quality of the mesh. This can be done by moving vertices, or keeping vertices locations and changing the connectivity.

1.4.2.1 Vertex relocation

First, some methods, often called *mesh smoothing* algorithms, perform vertex relocations to improve the quality of the mesh. Most mesh smoothing algorithms iterate over all the internal vertices in the mesh several times until every vertex has not moved more than a specified tolerance, which can be local or global.

Mesh smoothing methods can be enriched by alternating smoothing steps with connectivity updates. These steps ensure that each triangle remains well-oriented, and that there is no inconsistency in the mesh. In our framework, connectivity updates are canonical since the output mesh should keep the Delaunay property.

Amenta, Bern and Eppstein reformulate mesh smoothing problems as quasi-convex programs, a special case of generalized linear programs [ABE97]. Triangular mesh smoothing methods can be designed to optimize a single criterion, or a mixture of several criteria. For some criteria, quasi-convex programming allows to compute the optimal location of each vertex, iteratively. They show that optimizing the area, altitude, angles, edge lengths, aspect ratio perimeter, circumradius, inradius, area over squared edge length or a mixture of these can be done in linear time by quasi-convex programming.

Laplacian smoothing The most direct mesh smoothing method in computer graphics, due to its simplicity, is Laplacian smoothing. First introduced by Hermann [Her76], the Laplacian mesh smoother is a finite difference approximation of the Laplace operator. The optimization process is iterative. At each step, every vertex $\mathbf{x}$ is relocated to the centroid $\mathbf{x}^\star$ of its neighbor vertices (the ones with which it is sharing an edge), according to the following formula

$$\mathbf{x}^\star = \frac{1}{n}\sum_{i=1}^{n} w_i p_i,$$

where p_i is one of $\mathbf{x}$'s neighbors, and w_i the associated weight. Vertices weights can be 1, or any other value as long as $\sum_{i=1}^{n} w_i = n$. It can for example be proportional to the volume of the shared simplices.

Laplacian smoothing has been extensively used, studied and extended in the literature of triangle mesh generation [Fie88, Geo91, dlG95, CTS98]. It results in homogeneous meshes, but the associated energy is only linked to edge length. Spatial distribution of vertices is not taken into account. Therefore, it often fails to improve the shape of mesh simplices. Some other mesh optimization methods have been developed.

Centroidal Voronoi Tesselation Du and Wang [DW02] propose to generate quality meshes considering them as dual to optimal Voronoi diagrams. Their optimal Voronoi tesselation minimized the following quadratic energy

$$E_{\text{CVT}}(\mathcal{P}) = \sum_{i=1}^{N} \int_{y \in \mathcal{V}_i} \rho(y)||p_i - y||^2 dy,$$

where $\mathcal{P}$ is the set of vertices of the mesh and $\{\mathcal{V}_i\}_{i=1..N}$ the corresponding Voronoi cells.

This optimization process is iterative and alternates updates of the Voronoi diagram and vertex relocations. First, compute the Voronoi diagram of $\mathcal{P}$, since it is the partition that minimizes E_{CVT} for a given fixed set of vertices [Che05]. Second, perform one Lloyd iteration [Llo82] by relocating each vertex p_i to the centroid of its Voronoi cell $\mathcal{V}_i$. This operation also minimizes E_{CVT}, when the partition is fixed [Che05]. See Section 2.2.2 for more details on Lloyd iteration.

Du et al. [DFG99, DW02] describe how a mesh that minimizes the energy E_{CVT} has each vertex at the centroid of its Voronoi cell. The Voronoi diagram of $\mathcal{P}$ is hence said to be a *Centroidal Voronoi Tesselation*, CVT for short. Du and Wang [DW02] also show that CVT is generating meshes of higher quality than the widely used Laplacian smoothing.

The minimization of E_{CVT} corresponds to a function approximation problem. It can be seen as the minimization of the volume between the paraboloid of dimension $d+1$, defined by $f(x) = ||x||^2$, and an *underlaid, circumscribing piecewise linear approximant* f^{dual}_{PWL}, formed by planar patches tangent to the paraboloid (see Figure 1.8). Tangency points are obtained by lifting Voronoi vertices onto the paraboloid. Hence, E_{CVT} can be expressed as

$$E_{\text{CVT}} = ||f^{dual}_{\text{PWL}} - f||_{\mathcal{L}^1}.$$

In 2D, CVT tends to produce hexagonal Voronoi diagrams, dual to a triangulation composed of equilateral triangles. This is due to the fact that any $\mathcal{L}^p$ optimal approximation of a smooth function f tends to align and shape its elements with respect to the eigenvectors and eigenvalues of its Hessian [She02b]. The Hessian of $f = ||x||^2$ is isotropic, hence the resulting meshes must have nearly hexagonal cells. On a surface in 3D, CVT has the same effect and tends to produce nice surface meshes. However, in 3D volume meshes, getting a isotropic distribution of the vertices can have a negative effect on the tetrahedra quality [Epp01]. For example, a *sliver* tetrahedron has its four vertices located close to the equator of a sphere, with its vertices locations evenly distributed. Since Lloyd iteration tends to optimize the compactness of Voronoi cells, it makes the distribution of vertices more and more isotropic. It thus is blind to the creation and persistence of slivers in the mesh. Figure 1.6 illustrates the comparison between the 2D and the 3D case.

Recently, some approaches have been developed to accelerate the convergence of Lloyd algorithm to obtain a CVT. Du and Emelianenko based their work on using the

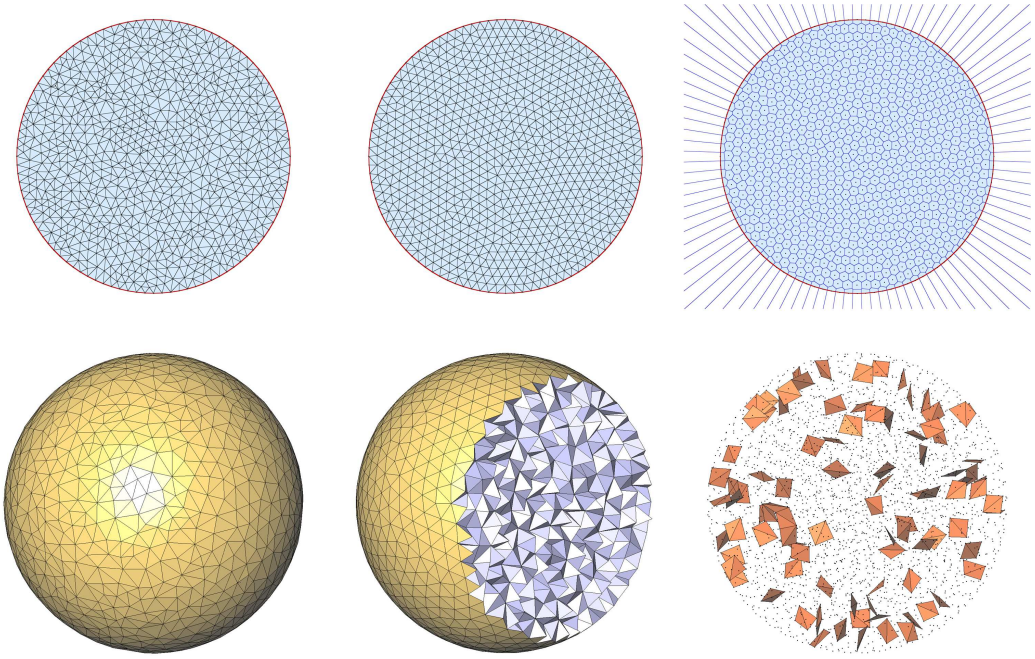


Figure 1.6: CVT mesh optimization. In 2D (top), (left) a 2D Delaunay mesh M_2 generated by Delaunay refinement, (center) M_2 optimized with CVT, and (right) M_2's Voronoi diagram. In 3D (bottom), (left) a 3D Delaunay mesh M_3 generated by Delaunay refinement, (center) M_3 optimized with CVT, and (right) M_3's slivers (tetrahedra with dihedral angles smaller than 5°).

Newton method [DE06a] when Lloyd's algorithm was getting close to convergence. Liu et al. provide an improvement by using a quasi-Newton approach, which can be used since the beginning of the optimization process on 3D surface [YLL+] and volume [LWL+09] meshes. See Section 4.2.2 for more details.

Optimal Delaunay Triangulation For 3D volume meshes, Chen [Che04] proposes a "dual" approach to CVT optimization. The energy that he aims at minimizing is the following

$$E_{\text{ODT}} = ||f^{primal}_{\text{PWL}} - f||_{\mathcal{L}^1} = \sum_{T_j \in \mathcal{DT}} \int_{T_j} |f^{primal}_{\text{PWL}} - f|,$$

where T_j is a mesh tetrahedron. This energy describes the volume between the paraboloid and an *overlaid, inscribed piecewise linear approximant* f^{primal}_{PWL} formed by a linear interpolation of points on the paraboloid (see Figure 1.8). Points are obtained by lifting the mesh vertices onto the paraboloid. As described in Section 3.2.1.5, minimizing this energy by relocating vertices turns the mesh into an *Optimal Delaunay Triangulation*, ODT for short.

Chen [Che04] observes that E_{ODT} can be expressed in a similar way to E_{CVT}, by the

formula

$$E_{\text{ODT}} = \frac{1}{n+1} \sum_{i=1..N} \int_{y \in \Omega_i} \rho(y) ||y - p_i||^2.$$

Each integral is here taken over Ω_i, the 1-ring neighborhood of p_i, composed of incident simplices (triangles in 2D, tetrahedra in 3D).

Chen's E_{ODT} energy is computed over primal simplices. It is thus more likely to improve the shape of primal simplices than E_{CVT}, which is focusing on the compactness of Voronoi cells. As shown on Figure 1.7, ODT iterations perform much better than CVT at improving the histogram of dihedral angles inside the mesh, and fewer slivers are left after optimization.

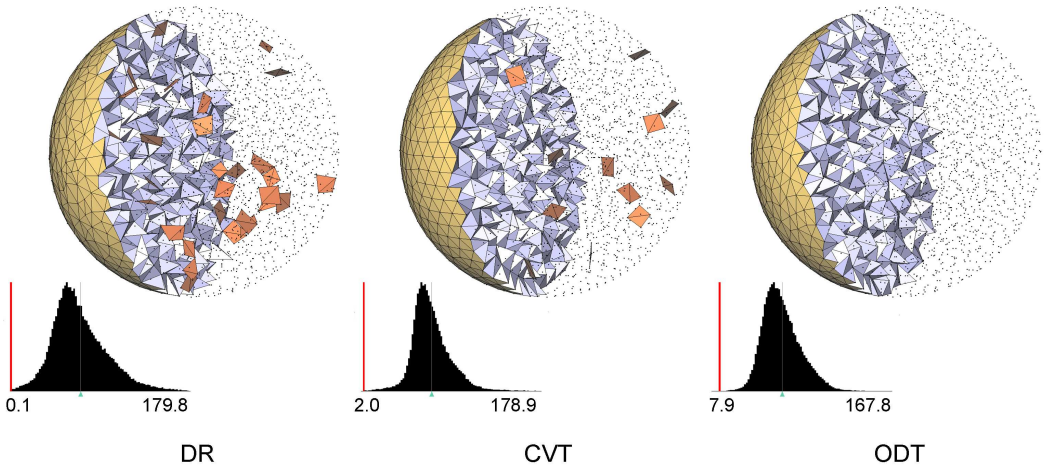


Figure 1.7: Comparison of the dihedral angles histograms after DR, CVT and ODT. Slivers (tetrahedra with a dihedral angle smaller than 5°) are plotted in red. (Left) Mesh M of a sphere, generated by greedy Delaunay refinement. (Middle) M optimized with CVT. (Right) M optimized with ODT.

Explicit vertex perturbation Li [Li00a] introduces the idea of a post-processing step to any mesh generation algorithm. This algorithm consists in moving vertices in a random manner inside a small sphere around each vertex. A small perturbation of each vertex incident to a sliver happens to be sufficient to trigger the edge flips necessary to remove most slivers. He shows that if such a perturbation exists, then it can be found in a finite number of random trials.

In Section 3.2.2, we describe an extended version of Li's algorithm. Our extension favors deterministic directions for vertex perturbation. Being more deterministic makes our algorithm faster, and allows it to reach higher minimal dihedral angles.

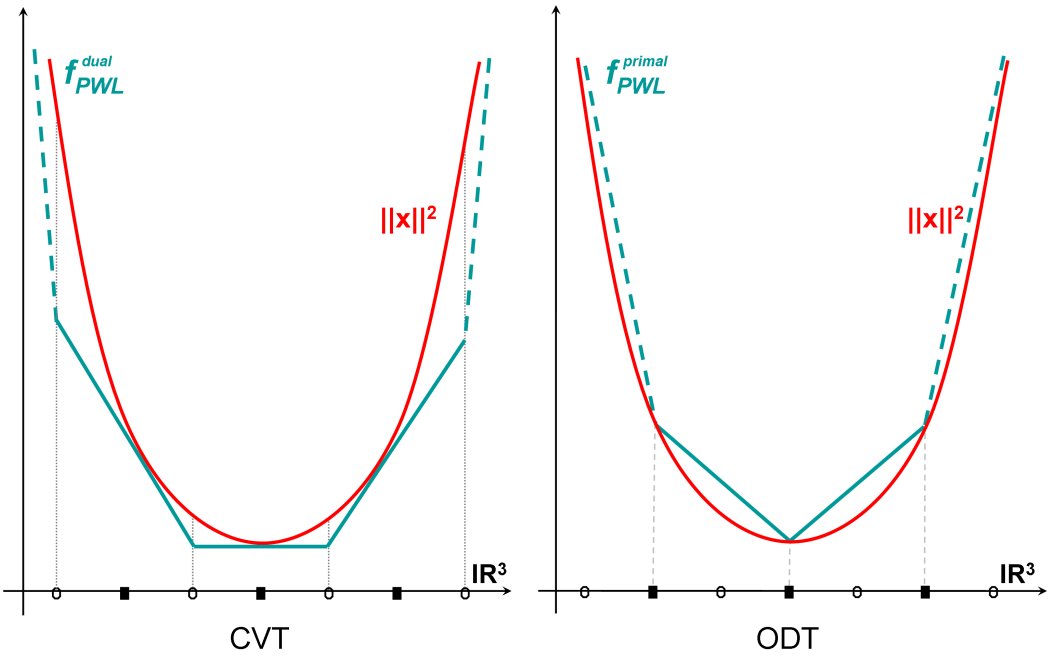


Figure 1.8: PWL approximations of the paraboloid. A paraboloid can be approximated by an underlaid circumscribed PWL function (left, CVT), or by an overlaid inscribed PWL function (right, ODT).

1.4.2.2 Connectivity modification

Sliver exudation First introduced by Cheng et al. [CDE+00], sliver exudation consists in changing the vertices weights to trigger flips and improve the overall quality of the mesh. Edelsbrunner and Guoy's experimental study [EG02] gives more practical details, and shows nice results on the final mesh dihedral angles.

Associated to classical Delaunay refinement, sliver exudation can be used for complete mesh generation [CD02, CDR05]. It can also be considered as a post-processing step [RY07, PSB+07] to improve the dihedral angles inside the mesh. Indeed, changing the vertices weights triggers edge flips inside the mesh, and is sometimes sufficient to improve the overall quality of the mesh.

Explicit modification Some other algorithms perform explicit modifications of the mesh connectivity. Their strategy is to leave the Delaunay framework and its canonical connectivity, and operate edge and facet flips to locally and globally improve the quality of the mesh.

Among them, Joe proposes an algorithm to turn a Delaunay triangulation into a max-min solid angle triangulation by edge flips [Joe91]. This algorithm ensures that

the minimum solid angle at tetrahedra vertices in the mesh is maximal, among all possible connectivity changes.

Klinger and Shewchuk [KS07] improve the quality of a tetrahedral mesh through explicit edge and facet flips, associated to vertex insertion and removal, and mesh smoothing steps. This complex mesh improvement strategy is definitely able to get high quality output dihedral angles in the mesh.

These methods have some drawbacks. First, the connectivity needs to be handled "by hand", and the propagation of modifications into the mesh is not trivial. Second, they are not able to take into account a sizing function, defining where in the domain the simplices should be large or small. Gradation of the initial mesh can thus the damaged.

1.5 Contributions

We present mesh generation algorithms, in 2D and in 3D, able to produce high quality isotropic and Delaunay meshes. Our first goal is to reach good dihedral angles, and also to insert as few Steiner points as possible, compared to existing mesh generation algorithms.

Since mesh optimization is a costly process, mesh quality has to be traded off against computation time. Assume that the minimum amount of time required for meshing a domain is the time spent by Delaunay refinement. Ideally, the mesh quality should then increase monotonously with the additional computation time.

Our algorithms take as input any closed polygonal domain in 2D, and any watertight manifold polyhedral domain in 3D. The Delaunay triangulation triangulates the whole convex hull of the mesh vertices. In 2D, our meshing algorithm exactly conforms the input boundary edges. In 3D, our meshing algorithm computes an interpolant approximation of the input boundary. It is a surface Delaunay triangulation restricted to the input boundary.

Interleaving refinement and optimization. The main idea that we develop in this thesis is that interleaving Delaunay refinement steps and mesh optimization steps is a valid approach for inserting a small number of Steiner points while obtaining well-shaped simplices. The idea comes from the following 1D observation. A greedy Delaunay refinement approach could, by recursively bisecting the edges, insert almost twice the number of Steiner points inserted by the interleaving approach. Figure 1.9 shows an example in 1D, where this observation is highlighted.

In the next sections, Algorithm 2, in 2D, and Algorithm 5, in 3D, describe the whole interleaved algorithms.

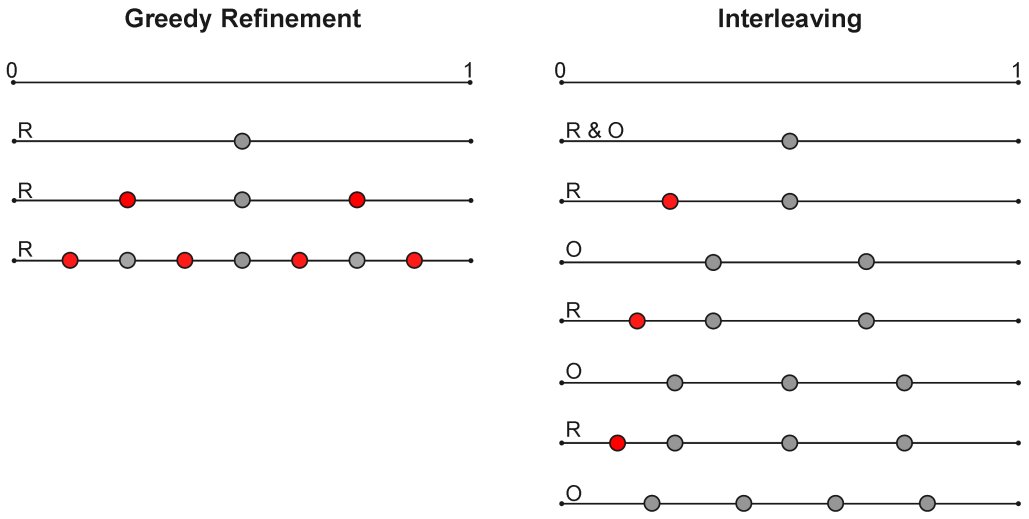


Figure 1.9: One-dimensional comparison between greedy refinement and interleaved refinement and optimization. In both cases in this particular example, one wants to discretize an edge of length 1 into sub-edges of length at most 0.21. On the left, refinement steps by bisection, 'R', are recursively processed on the edge. Seven (red) points are inserted recursively, and the process requires 3 refinement steps. On the right, refinement by bisection, 'R', and smoothing points locations along the edge, 'O', are iteratively processed. Four (red) points are inserted, and the process requires 7 steps.

Optimization. Most of the state-of-the-art mesh smoothing algorithms take into account interior vertices only, or boundary vertices only for surface meshing. In our algorithms, we provide a consistent mesh optimization framework for inside vertices and boundary vertices. To achieve this goal, we use different mesh optimization schemes in 2D and in 3D, since they had not the same efficiency.

In 2D, we use Lloyd iteration to reach a CVT and apply it consistently to the vertices inside the domain, and in 1D to the vertices lying on input constrained edges (see Section 2.2.2).

In 3D, we extend the *ODT* volume optimization process to surface boundary vertices (see Section 3.2.1.5).

Refinement. Interleaving refinement steps and optimization steps requires defining what is a *refinement step*. Since we don't want the algorithm to end up as a greedy Delaunay refinement algorithm would, each refinement step should insert a

set of well chosen Steiner points. In any dimension, our main goal is to avoid the creation of points clusters.

In 2D, we refine worst simplices first, and we propose a solution to avoid refining other simplices that would be too close (see Section 2.2.1).

In 3D, we compute an independent set of Steiner vertices. Starting with worst quality simplices, two vertex insertions should not impact the same tetrahedra (see Section 3.2.1.4).

Sliver removal (3D). Though our extension of *ODT* mesh optimization is able to achieve good dihedral angles bounds and histograms, we propose an additional post-processing sliver removal algorithm (see Section 3.2.2). Our sliver removal step is hill-climbing by construction in terms of dihedral angles. Extending Li's random vertex perturbation [Li00a], we experimentally show that our algorithm is able to reach higher dihedral angles within shorter computation times. On some examples, it is able to reach a minimum dihedral angle as high as 30°.

Chapter 2

2D Triangle Mesh Generation

Contents

We consider the problem of generating 2D triangle meshes from a bounded domain and a set of geometric and sizing constraints provided respectively as a planar straight line graph (PSLG) and a sizing function [TAD07]. This problem is mainly motivated by several practical applications, such as numerical simulation of physical phenomena. The latter applications require *quality* meshes, ideally of smallest size. Quality herein refers to the shape and size of the elements: A triangle is said to be good if its angles are bounded from below, and if the length of its longest edge does not exceed the sizing function. The smallest size triangulation, conforming the sizing function, is sought after for efficiency reasons as the number of elements governs the computation time.

Quality triangulation is a well-studied problem, and many meshing strategies have been proposed and studied. Existing algorithms could be roughly classified as being *greedy*, *variational*, or *pliant*. Greedy algorithms commonly perform one local change at a time, such as vertex insertion, until the initial stated goal is satisfied. Variational techniques cast the initial problem into the one of minimizing an energy functional such that low levels of this energy correspond to good solutions for this problem (reaching a global optimum is in general elusive). A minimizer for this energy may perform global relaxation, i.e., vertex relocations and re-triangulation until convergence. Finally, an algorithm is pliant when it combines both refinement and decimation, possibly interleaved with a relaxation procedure [BH96, CGS06].

Although satisfactory bounds have been obtained for the triangle angles using Delaunay refinement algorithms [MPW05], one recurrent question is to try lowering the number of points (so-called *Steiner points*) added by the meshing algorithm. Recall that the Steiner points are inserted either on the constrained edges or inside the domain, to satisfy sizing and quality constraints, as well as to preserve the input constrained edges. At the intuitive level, inserting "just enough" Steiner points requires generating triangles which are everywhere as big as possible, while preserving the sizing constraints. Luckily enough, the triangle which covers the biggest area for a given maximum edge length is the equilateral triangle. In other words, and by using a simple domain covering argument, trying to generate *large and well-shaped triangles* simultaneously serves the goal of lowering the number of Steiner points. Our approach builds upon this observation.

Contributions This chapter proposes to mesh a 2D domain into large and well-shaped triangles by alternating Delaunay refinement and optimization. We pursue the goal of generating large triangles by performing refinement in a multilevel manner, i.e., by inserting a subset of the Voronoi vertices batch-wise at each refinement stage. Each refinement stage is parameterized with a decreasing size, which is computed from the current mesh. Each mesh optimization stage is performed by applying the Lloyd iteration both in 1D along constrained edges, and in 2D inside the domain. To ensure that the constraints are preserved during mesh optimization, the bounded Voronoi diagram (Voronoi diagram with constraints) is computed using a novel robust and effective algorithm. For the sake of efficiency, the optimization stages are applied with increasing accuracy. Our experiments show that, for the required sizing, our algorithm generates in general meshes with fewer Steiner points and better quality than Delaunay refinement alone. We now briefly review Delaunay refinement and mesh optimization techniques.

2.1 Related work

2.1.1 Delaunay Refinement

A mesh refinement algorithm iteratively inserts well chosen points, so-called *Steiner points* in a given coarse mesh until all constraints are satisfied. One trend in refinement algorithms is to insert as few Steiner points as possible. One popular approach is to take as initial coarse mesh the constrained Delaunay triangulation of the input PSLG, and to refine it. Refinement algorithms of this kind are called *Delaunay Refinement algorithms*. They were pioneered by Chew [Che89a], and later extended by many authors [Rup95, She02b]. Delaunay refinement algorithms proceed as follows: During refinement, an edge is said to be encroached if a point of the triangulation (not its endpoints) is on or inside its diametral circle. As long as the current mesh contains encroached edges, the algorithm inserts their

midpoints. It then iteratively inserts the circumcenter of the "worst" triangle of the triangulation (according to size and shape criteria), unless it encroaches an edge. These steps are performed until all triangles are good, i.e., until each triangle has the size and shape criteria satisfied. Shewchuk [She02b] shows that this algorithm terminates with a finite number of Steiner points and with bounds on the triangles angles (provided the input PSLG does not contain any small angle). Several studies have been made on the insertion order of the Steiner points, and a satisfactory choice is to insert the circumcenter of the worst triangle first.

Another way to grasp the problem of minimizing the number of Steiner points is to define another type of Steiner points than circumcircle centers. The main idea is that if a triangle contains a large angle, Ruppert's algorithm inserts a Steiner point far away from it, in a place which may be irrelevant. Üngör [Üng04] defines a so-called *off-center* as follows. The off-center off_c of a triangle $\triangle(p, q, r)$ of shortest edge pq is defined as the circumcenter c of $\triangle(p, q, r)$ if the radius-edge ratio of $\triangle(p, q, c)$ is lower or equal to β, the angle quality bound. Otherwise, off_c is the point v on the bisector of pq, and inside its circumcircle, such that the radius-edge ratio of $\triangle(p, q, off_c)$ equals β. This technique is shown to insert fewer Steiner points than Ruppert's algorithm.

Chernikov and Chrisochoides generalize and improve over Üngör's off-centers solution [CC06]. They show that, more generally, any point in the *selection disk* of a bad triangle $\triangle$ can be chosen as a Steiner point. It is shown that, choosing any point inside the *selection disk* as Steiner point eliminates $\triangle$, and that the algorithm terminates. They propose an example of new Steiner point inside the *selection disk* and show that the corresponding refinement algorithm in general inserts fewer points than when using off-centers.

Some other methods combine both insertion and removal of mesh elements (including Steiner points) to obtain higher quality meshes. Such combination was used by Bossen and Heckbert [BH96] as well as by Coll et al. [CGS06], to cite a few. Finally, Erten and Üngör [EÜ07] have recently introduced a new method which first tries relocating the vertices of bad triangles. If the relocations would not improve the mesh, they are canceled and a Steiner point is inserted inside $\triangle$.

2.1.2 Optimization

Many different triangulations covering a given domain and satisfying a set of constraints may exist, each of them of different quality. When high quality meshes are sought after, it is therefore desirable to resort to an optimization procedure so as to optimize a specific quality measure (see [She02d] for a comprehensive study of quality measures). Two questions now arise: Which criterion should be optimized?

By exploiting which degrees of freedom? The optimized criterion can be directly related to the shape and size of the triangles [ABE97], but other criteria have been proposed as well. We refer the reader to [Epp01] for a comprehensive survey of mesh optimization techniques. As the number of degrees of freedom are both continuous and discrete (vertex positions and mesh connectivity), there is often a need for narrowing the space of possible triangulations. For example, Chen proposes to cast the isotropic meshing problem as an (isotropic) function approximation problem, optimizing within the space of Delaunay triangulations [Che04].

Eppstein [Epp01] highlights the fact that, in 2D, evenly distributed points lead to well-shaped triangles, assuming an isotropic triangulation such as the Delaunay triangulation. Isotropic meshing can therefore be casted into the problem of isotropic point sampling, which amounts to distribute a set of points on the input domain in as even a manner as possible. One way to distribute a set of points isotropically and in accordance with a sizing function is to apply the Lloyd iteration procedure (described in Section 2.2.2) over an initial Voronoi diagram. Du et al. [DFG99] have shown how the Lloyd iteration transforms an initial ordinary Voronoi diagram into a centroidal Voronoi diagram, where each generator happens to coincide with the center of mass of its Voronoi cell. Another interesting feature of the Lloyd iteration in our context is that it can be applied to any dimension, i.e., in 1D for the Steiner points inserted on the constrained edges, and in 2D for the Steiner points inserted inside the domain. One drawback of the Lloyd iteration is its slow convergence. Moreover, it only converges to a local minimum of a certain energy functional [DEJ06]. Convergence accelerations are possible either by using Newton-Lloyd iterations or by using specific types of multilevel refinements [DE06a, DE06b].

2.2 Algorithm

We introduce an algorithm which interleaves Delaunay refinement and optimization, in order to generate a triangle mesh satisfying both shape and size properties for each triangle. We sketch our algorithm in pseudo-code as described in Algorithm 2.

The complete algorithm sequence is illustrated by Figure 2.1. Figures 2.6, 2.7, 2.8, and 2.9 compare the output of our algorithm with a mesh obtained by Delaunay refinement alone. Our method improves over the number of Steiner points as well as the shape of the elements.

2.2.1 Refinement

The refinement steps are specifically designed to insert “just enough” Steiner points to respect the sizing function, and to provide the optimization steps with good initial solutions. We achieve this goal by inserting batches of Steiner points chosen

Algorithm 2 Meshing algorithm

Input: A PSLG domain $\Omega \subset \mathbb{R}^2$.
  Let T be the constrained Delaunay triangulation of Ω.
  repeat
    Batch refinement of the triangulation T (Section 2.2.1).
    repeat
      Optimization of T by the Lloyd algorithm (Section 2.2.2),
    until Stopping criterion S.
  until Refinement does not insert any new Steiner point in T.
  repeat
    Optimization of T by the Lloyd algorithm,
  until Stopping criterion S' stronger than S.
Output: The final triangle mesh.

as Voronoi vertices. The roles are separated: Each refinement step acts on the size of the elements, while each optimization step acts on the shape of the elements (see Section 2.1.2).

Let us call $\mu(p)$ the desired mesh sizing specified at point p. We assume that this sizing function is provided by the user. During meshing we call *sizing ratio* the ratio between the current size of an element and the local desired sizing. An element can be an edge or a triangle, i.e., the sizing ratio is computed as $sr(f) = \frac{\text{longest edge(f)}}{\mu(centroid(f))}$ for a triangle and $sr(e) = \frac{\text{length(e)}}{\mu(centroid(e))}$ for an edge. After meshing, the sizing ratios of all elements must be lower or equal to 1 in order to satisfy the sizing constraints, and a low number of Steiner points is obtained by keeping the sizing ratios as close to one as possible under these constraints.

Before each refinement step, the algorithm iterates over each triangle and constrained edge of the mesh in order to compute the maximum sizing ratio sr_{max}. From this value we compute the current *target sizing ratio* as $sr_{target} = \max(\frac{sr_{max}}{\sqrt{3}}, 1)$, which is then used as a slowly decreasing sizing criterion for refinement. The refinement factor $1/\sqrt{3}$ is justified by the fact that inserting all circumcenters in a triangulation made of equilateral triangles operates a $\sqrt{3} - section$ of the edges (compared to a bisection on the constrained edges). In 2D, the refinement corresponding to the target sizing ratio can therefore not be faster than this value. As we aim at refining the mesh with sizing ratios as uniform as possible, we parameterize the 1D refinement with the same target sizing ratio. The target sizing ratio is thresholded to 1 in order not to insert points that would introduce new triangles or edges with a sizing ratio lower than 1, i.e. smaller than the desired sizing. Hence, at each refinement iteration, the target sizing ratio slowly decreases, until it reaches the value of 1, which corresponds to satisfying the input sizing requirement, via the sizing function μ, for each simplex of the mesh.

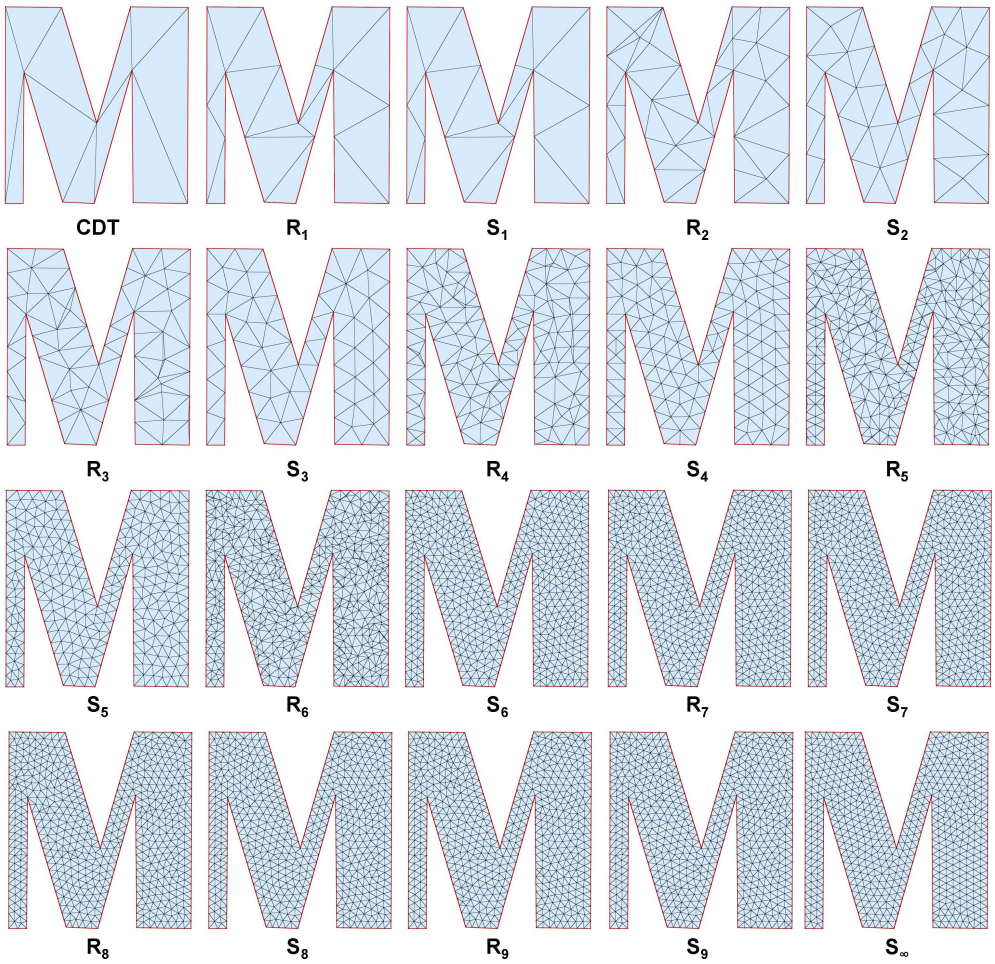


Figure 2.1: Interleaved Delaunay refinement and optimization with a uniform sizing parameter. In the reading order: The constrained Delaunay triangulation (CDT) of the initial PSLG (13 vertices), then refinement (R_i) and smoothing (S_i) in alternation. The last step (S_∞) is the final optimization until the stronger stopping criterion. The final mesh contains 645 vertices.

Refinement is first performed in 1D along the constraint edges, then in 2D. Call $\mathcal{L}$ the list of Steiner points to be inserted. The algorithm iterates over all triangles and constrained edges and measure their sizing ratio. If it is larger than the current target sizing ratio sr_{target}, we add its midpoint or circumcenter to $\mathcal{L}$, and alter the sizing ratio of its (unprocessed) incident elements in order to avoid over-refinement.

More specifically, and relying on the fact that a local point insertion involves neighboring triangles, we inform the neighboring triangles of a split triangle that they should take upon themselves a part of this refinement. To do so, when a triangle f

is processed and eligible for splitting, the quantity $1/3 * (sr_f/\sqrt{3} - sr_{target})$ is added to the sizing ratios of each of its unprocessed neighboring triangles. In this way, we divide between f's neighboring triangles the difference between what we expect to be the sizing ratio of f after splitting ($sr_f/\sqrt{3}$) and the target sizing ratio. Then, these triangle sizing ratios may be increased or decreased, and respectively get more or less likely to be split during the current refinement step. A similar method is used in 1D over the edges, with 1/2 as refinement factor. When all elements have been visited, all points in $\mathcal{L}$ are inserted to the constrained Delaunay triangulation. The next step is mesh optimization.

2.2.2 Optimization

The mesh optimization step involves the Lloyd iteration applied to the cells of a bounded Voronoi diagram (the pseudo-dual of the constrained Delaunay triangulation).

Lloyd iteration. The Lloyd iteration [Llo82] is a minimizer for the following energy functional:

$$E_{\text{CVT}} = \sum_{i=1}^{N} \int_{y \in \mathcal{V}_i} \rho(y) ||y - \mathbf{x}_i||^2 dy,$$

where $\{\mathbf{x}_i\}_{i=1}^N$ are a set of generators and $\{\mathcal{V}_i\}_{i=1}^N$ the corresponding Voronoi cells. The Lloyd iteration minimizes this energy by alternately moving the generators to the centroid of their Voronoi cells, and recomputing the Voronoi diagram. The centroid $\mathbf{x}^\star$ of the cell $\mathcal{V}$ is defined as:

$$\mathbf{x}^\star = \frac{\int_{\mathcal{V}} y\rho(y)dy}{\int_{\mathcal{V}} \rho(y)dy},$$

where ρ is a density function defined on the domain Ω. After convergence, the space subdivision obtained is a *centroidal Voronoi tessellation* (CVT) [DFG99, DEJ06]. As it corresponds to a critical point of the energy E_{CVT}, it is a necessary condition for optimality, in the sense of minimizing E_{CVT}.

We choose to stop the Lloyd iteration when all generators move less than a user-defined distance threshold. We first define the notion of move ratio of a generator $\mathbf{x}$ in its Voronoi cell as $m_r(\mathbf{x}) = \frac{||\mathbf{x}-\mathbf{x}^\star||}{size(cell(\mathbf{x}))}$, where the size of the cell is defined as the longest distance between any pair of points of this cell. The Lloyd iteration is stopped when $\max_{\mathbf{x} \in \{\mathbf{x}_i\}_{i=1}^N} m_r(\mathbf{x}) < p$ (p is typically set to 3% during refinement, and to 1% for the final relaxation step). In practice, the 1D Lloyd iteration, which allows moving the Steiner points to the centroids of their 1D cells (along the constraints), plays an important role at minimizing the number of Steiner points inserted. We could exhibit an extreme example

where combining refinement with 1D Lloyd iteration inserts half of the number of 1D Steiner points inserted by the common recursive edge bisection of the constraints.

Our algorithm applies the Lloyd iteration both in 1D by moving Steiner points along the input constrained edges (Voronoi cells are line segments), and in 2D by moving Steiner points at the centroid of their bounded Voronoi cell. Figure 2.2 illustrates both 1D and 2D iterations.

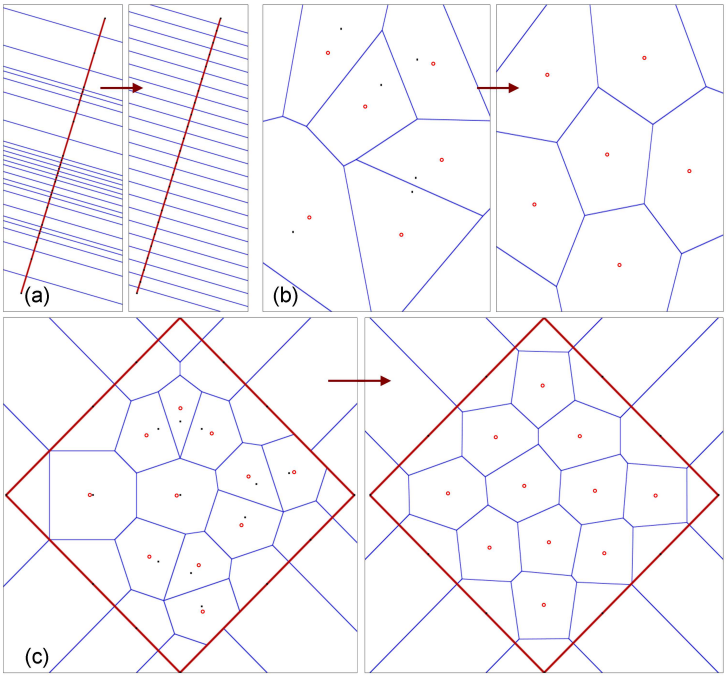


Figure 2.2: Lloyd iteration with a uniform density. (a) Steiner points randomly inserted on a constrained edge, and after convergence of the Lloyd iteration. (b) Steiner points randomly inserted on a 2D domain, and after convergence of the Lloyd iteration. Sites are filled dots, and centroids are outlined dots (they coincide after convergence). (c) Lloyd iterations applied both in 1D and 2D.

Bounded Voronoi Diagram. While the ordinary Voronoi diagram and Delaunay triangulation do not take constraints into account, we wish here to prevent the Voronoi regions to cross over the constraints. To this aim we use a *constrained Delaunay triangulation* [Che89a, She96, DGJ03] and a variant of its dual, the *bounded Voronoi diagram* (BVD), defined by Seidel [Sei88]. The common duality between Delaunay triangulation and Voronoi diagram links each triangle to its circumcenter. In our context, each triangle $\triangle$ may have its circumcenter c on the other side of a constrained edge. Hence, c is the dual of $\triangle$ if it is on the same side of the constraint. Otherwise, it is a pseudo-dual, and some Voronoi edges of $\triangle$ must be clipped by the constraint. The BVD is defined as follows: each cell V_i of a generator x_i is composed

by the points of the domain Ω which are closer to x_i than to any other generator. As for the constrained Delaunay triangulation, the distance incorporates visibility constraints. The distance $d_S(x, y)$ between two points x and y of $\mathbb{R}^2$ is defined as:

$$d_S(x,y) = \begin{cases} ||x-y||_{\mathbb{R}^2} & \text{if } x \text{ “sees” } y, \\ +\infty & \text{otherwise.} \end{cases}$$

In this definition, x “sees” y when no constrained edge intersects the segment $[x, y]$. This visibility notion can be extended to triangles. We will see later how the notion of triangle sight, or symmetrically triangle “blindness”, is important to construct the bounded Voronoi diagram. Figure 2.3 illustrates a constrained Delaunay triangulation and its pseudo-dual bounded Voronoi diagram. Notice that trying to construct the naïve Voronoi diagram by joining the circumcenters of all pairs of incident triangles would not even form a partition. The notion of triangle blindness is pivotal for constructing the bounded Voronoi diagram.

Definition 2.2.1 (Blind triangle)**.** *A triangle $\triangle$ is said to be* blind *if the triangle and its circumcenter c lie on the two different sides of a constrained edge E. Formally, $\triangle$ is* blind *if and only if there exists a constrained edge E such that one can find a point p in $\triangle$ (not an endpoint of E), such that the intersection $[p, c] \cap E$ is non empty.*

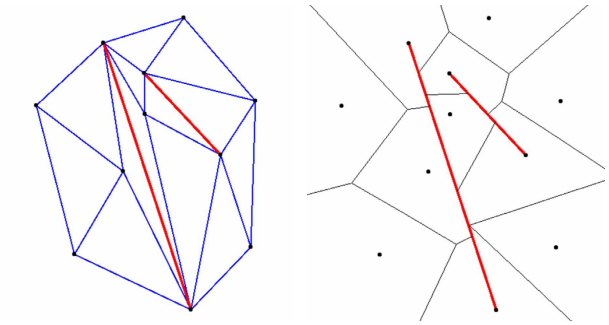


Figure 2.3: Constrained Delaunay triangulation of a set of points (left) and its pseudo-dual bounded Voronoi diagram (right).

The BVD construction algorithm initially tags all triangles of the triangulation as being blind or not blind (Algorithm 3). It then constructs each cell of the diagram independently using these tags (Algorithm 4). Finally, all cells are assembled to build the complete bounded Voronoi diagram of a given set of points and constrained edges.

Algorithm 3 tags all triangles of the triangulation as being either *blind* or *non-blind*. In addition, each *blind* triangle stores which constrained edge in the triangulation acts as a visibility obstacle, i.e., which edge prevents it to *see* its circumcenter (it is the first constraint intersected by the oriented line joining any point of the triangle to any point of the constraint). Notice how the algorithm only needs to iterate

Algorithm 3 Tag blind triangles

Input: Constrained Delaunay triangulation *cdt*.

```
Tag all triangles non-blind by default.
for each Constrained edge e of cdt do
  Create a stack: triangles
  for each Adjacent triangle f_e to e tagged non-blind do
    Push f_e into triangles
    while triangles is non-empty do
      Pop f from stack triangles
      if f is blinded by e (use P) then
        Tag f as blinded by e
        for each Adjacent triangle f' to f do
          if f' is finite and tagged non-blind
          and the common edge between f and f' is unconstrained then
            Push f' into triangles.
```

over the constrained edges of the triangulation, as all sets of blinded triangles form connected components incident to constrained edges. Indeed, the Voronoi diagram of the vertices of the blind triangles on one side of any constraint is a tree rooted from the dual ray of the constraint.

We define a robust predicate, called $\mathcal{P}$ in the sequel, to test if a triangle is blinded by a constrained edge. More specifically, $\mathcal{P}$ takes as input a triangle and a segment, and returns a Boolean indicating whether or not the circumcenter of the triangle lies on the same side of the segment than the triangle. The circumcenter is never constructed explicitly in order to obtain a robust tagging of the blind triangles. Each cell of the bounded Voronoi diagram can be constructed by circulating around vertices of the triangulation, and by choosing as cell vertex either circumcenters or intersections of the standard Voronoi edges with the constrained edges. Note that we do not need to construct bounded Voronoi cells incident to input constrained vertices as the latter are constrained and therefore not relocated by the Lloyd iteration. Moreover, these cells are the only ones which would not necessarily be convex. This is not an issue since the constrained vertices are not meant to be relocated by the Lloyd iteration. Algorithm 4 describes this construction, and Figure 2.4 illustrates the construction of a single bounded Voronoi cell. Figure 2.5 illustrates a bounded Voronoi diagram.

Quadratures The Lloyd iteration requires computing centroids of line segments in 1D and of (possibly bounded) Voronoi cells in 2D. Such computations require quadrature formulas when a variable density function is specified, i.e., when the input sizing function is not uniform. The density function ρ and the sizing function μ are linked by the following formula [DW05]: $\mu(x) = \frac{1}{\rho(x)^{d+2}}$, where d is the

Algorithm 4 Construct a BVD cell

Input: Unconstrained vertex z of the constrained Delaunay triangulation *cdt*.
  Call P the polygon (cell) in construction,
  Call f the current triangle and f_{next} the next triangle in the circulation,
  Call $\mathcal{L}_{f,f_{next}}$ the line going through the circumcenters of f and f_{next}.
  for each Incident triangle f to z in *cdt* **do**
    if f is tagged non-blind **then**
      Insert the circumcenter of f into P.
      if f_{next} is blind **then**
        Call $S_{f_{next}}$ the constrained edge blinding f_{next},
        Insert point $\mathcal{L}_{f,f_{next}} \cap S_{f_{next}}$ into P.
    else
      Call S_f the constrained edge blinding f.
      if f_{next} is tagged non-blind **then**
        Insert $\mathcal{L}_{f,f_{next}} \cap S_f$ into P.
      else
        Call $S_{f_{next}}$ the constrained edge blinding f_{next},
        if $S_f \neq S_{f_{next}}$ **then**
          Insert $\mathcal{L}_{f,f_{next}} \cap S_f$ and $\mathcal{L}_{f,f_{next}} \cap S_{f_{next}}$ into P.
Output: Bounded Voronoi cell of z.

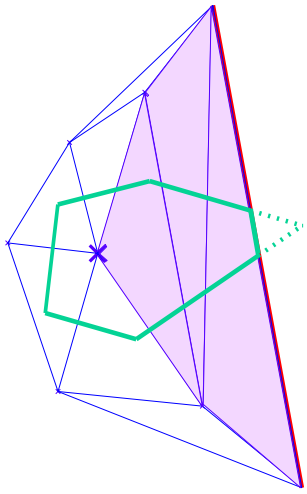


Figure 2.4: Construction of a cell of the bounded Voronoi diagram. The standard Voronoi diagram is truncated on the constrained edge (colored triangles are blind).

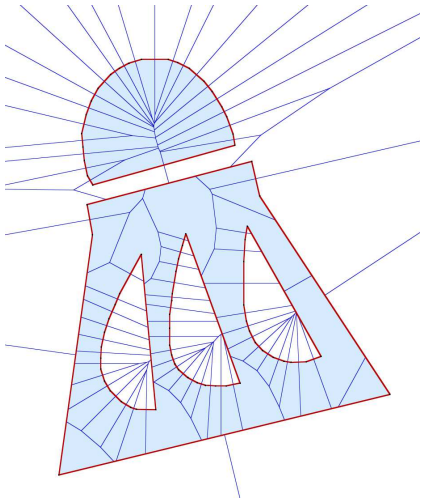


Figure 2.5: Bounded Voronoi diagram of a PSLG.

dimension of the domain. In 2D, we have

$$\mu(x) = \frac{1}{\rho(x)^4}.$$

The key idea behind a quadrature is to decompose a simple domain (be it an edge or a triangle in our case) into smaller sub-domains (so-called quadrature primitives) where simple interpolation schemes are devised. The number N of quadrature primitives used for each element allows the user to tune the computation accuracy of the centroids. In practice this number is increased during refinement so that a low precision is used at coarse levels (typically $N = 10$), and a high precision is used at fine levels ($N = 100$ for the final level).

We use the trapezium rule in 1D, with N sub-segments, and the midpoint approximation rule in 2D, with a decomposition of each bounded Voronoi cell into N sub-triangles. More precisely, an initial step triangulates the cell by joining each of its vertices to its generator. The next step recursively bisects the longest edge of these triangles until the number of quadrature triangles reaches N. On each quadrature triangle, the midpoint approximation formula is applied:

$$\int_{\triangle} f(x)dx \approx \frac{|\triangle|}{3}(f(x_{12}) + f(x_{23}) + f(x_{13})),$$

where x_{12}, x_{23} and x_{13} are the midpoints of a quadrature triangle edges. Finally, we sum the integrals on each quadrature triangle in order to obtain an approximate centroid of the whole bounded Voronoi cell.

2.3 Implementation

Our algorithm is implemented in C++, using the *Computational Geometry Algorithms Library* CGAL [cga, FGK+00]. The Delaunay refinement algorithm that we use to compare our meshes to is the Terminator algorithm defined by Shewchuk [She00a], implemented as a CGAL package [Rin07]. The chosen angle bound is set to 20.7°.

2.4 Results

To estimate the added value of our interleaving approach, we run our meshing algorithm on different types of PSLG.

Figures 2.7 and 2.8 show uniform meshes, and compare our algorithm with the standard Delaunay refinement algorithm. The number of Steiner points inserted are

respectively 22% and 23% fewer with our method, and the angle distributions are better centered around 60 degrees, as expected. The intervals in which the angles lie are also tighter: $[33^\circ, 99^\circ]$ vs $[20^\circ, 124^\circ]$ for Figure 2.7, and $[29^\circ, 103^\circ]$ vs $[22^\circ, 127^\circ]$ for Figure 2.8. The distributions of aspect ratios (circumradii to shortest edge ratios) show that our algorithm produces many more triangles with aspect ratios close to the optimal value, which is $1/\sqrt{3} \approx 0.57$ for an equilateral triangle. This shows that, although our algorithm does not provide theoretical bounds on the triangle angles or aspect ratios, the practical bounds are improved.

Obviously, any mesh generation algorithm which incorporates smoothing or optimization is expected to obtain meshes of higher quality than a greedy algorithm such as Delaunay refinement, at the price of higher computation times (10 times slower for our algorithm in the uniform cases, and close to 100 times slower in the non-uniform cases which involve costly quadratures). In our experiments, interleaving refinement and optimization, similar in spirit to multilevel algorithms, consistently leads to higher quality meshes than refining, then optimizing at the finest level. In addition to the mesh quality obtained, the choice of the Lloyd iteration instead of the many other mesh optimization techniques is justified by the possibilities to apply it in 1D and 2D, and to take a sizing function as input with variable precision quadratures.

One of the main problems in mesh generation is to limit the number of small angles, even when small angles are part of the input [She02d]. The input PSLG of Figure 2.9 contains 17 "spins", with inside angles ranging from 5 to 37 degrees. As depicted by the angles distributions, small angles do not hurt our algorithm, which does not produce more small angles than Delaunay refinement does. In addition, our algorithm inserts 26% fewer Steiner points than the Terminator algorithm does. The practical upper bound on angles is lower, albeit the improvement is small (128° vs 131°).

As already discussed in Section 2.2.2, optimizing the mesh simultaneously in 1D and 2D has an important impact on the number of Steiner points inserted. For example, when the input PSLG contains two long constraints parallel and close to each other, the local improvement in terms of number of 1D Steiner points can be as high as 50%. Intuitively, Delaunay refinement is restricted to recursive constrained edge bisection, which may lead to bisect all constrained edges at the finest level, even if they are slightly longer than the local admissible sizing. In our algorithm, each refinement step is followed by an optimization step, which prevents the next refinement step to over-refine. This behavior is illustrated by Figure 2.10, where our method inserts 25% fewer Steiner points than Delaunay refinement alone.

For an input domain Ω and a sizing function μ, one can compute the minimum number of triangles needed to cover Ω. After computing the mean edge length over the domain: $\bar{\mu} = \frac{\int_\Omega \mu(x)dx}{area(\Omega)}$, the minimum number of triangles needed is given by $m_\triangle = \frac{area(\Omega)}{\sqrt{3}/4.\bar{\mu}^2}$, where the denominator is the mean area of an equilateral triangle

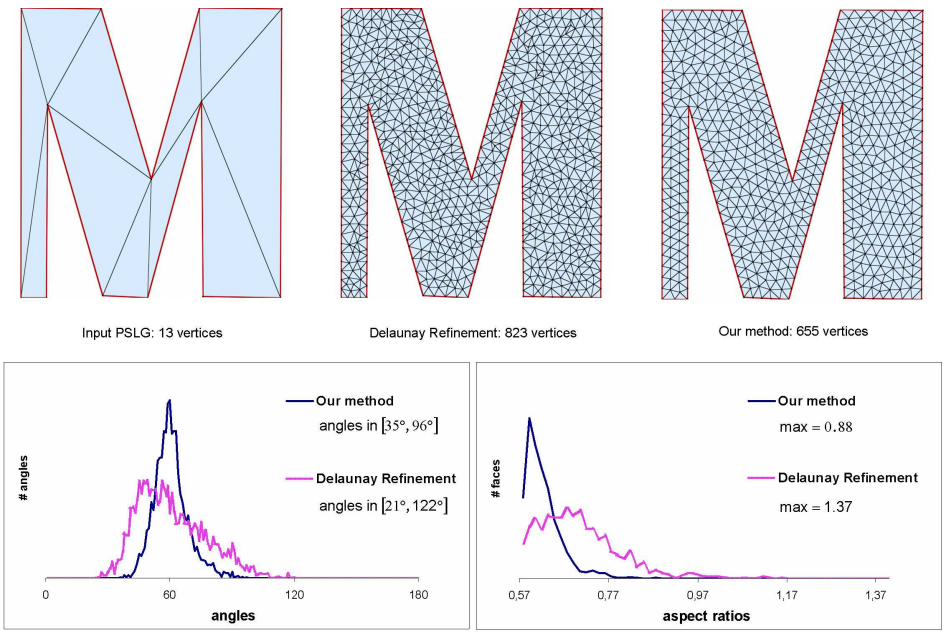


Figure 2.6: Comparison between Delaunay refinement alone and our algorithm. The sizing function is uniform and equals 0.05. Our method inserts 20% fewer Steiner points than Delaunay refinement does. The distribution of angles is narrower at the end of our algorithm. The angles are within $[35^\circ, 96^\circ]$ with our method, vs $[21^\circ, 122^\circ]$ with Delaunay refinement. The practical upper bound on aspect ratios is lower with our method: 0.88 vs 1.37.

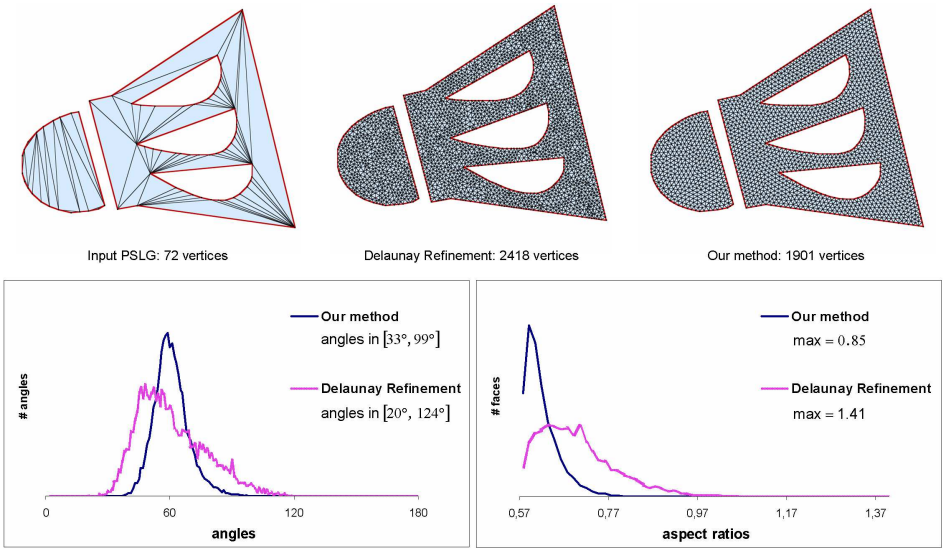


Figure 2.7: The sizing function is uniform and equals 0.02. Our method inserts 22% fewer Steiner points than Delaunay refinement does. The angles are in $[33^\circ, 99^\circ]$ with our method, vs $[20^\circ, 124^\circ]$ with Delaunay refinement. The maximum aspect ratio is lower: 0.85 vs 1.41.

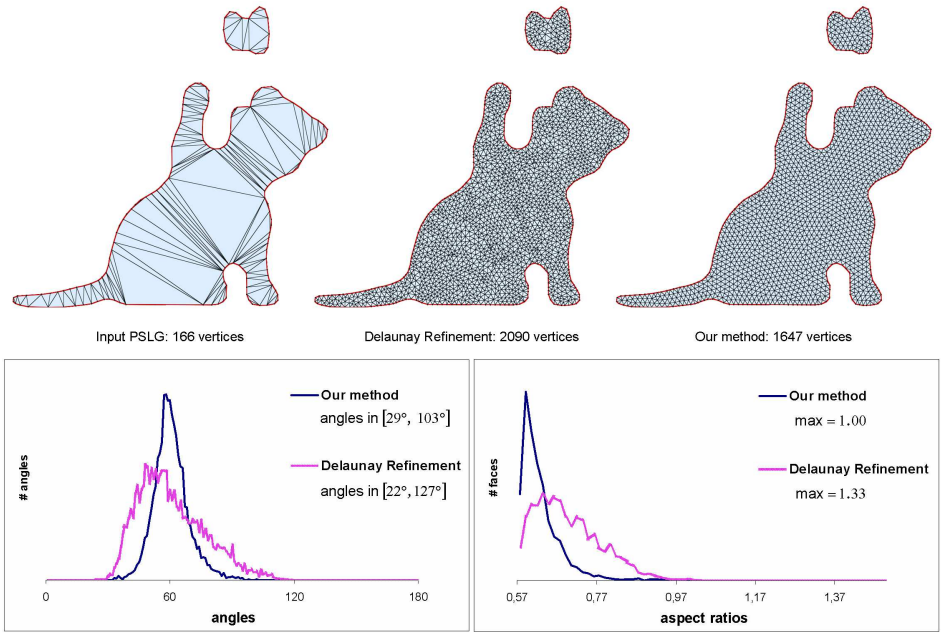


Figure 2.8: The sizing function is uniform and equals 0.02. Our method inserts 23% fewer Steiner points than Delaunay refinement does. The angles are in $[29^\circ, 103^\circ]$ with our method, vs $[22^\circ, 127^\circ]$ with Delaunay refinement. The maximum aspect ratio is lower: 1.00 vs 1.33.

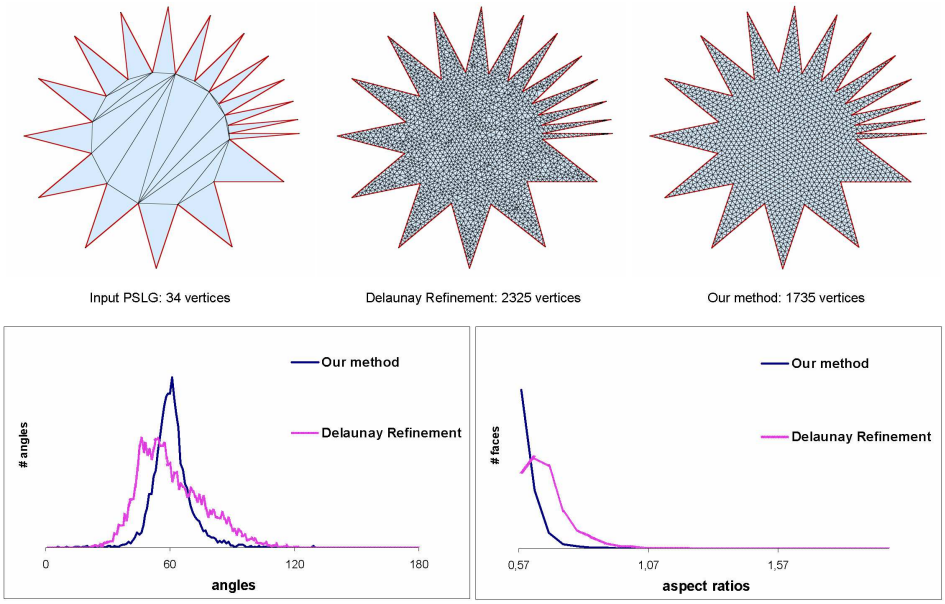


Figure 2.9: The input PSLG constraints form acute angles from 5° to 37°. The sizing function is uniform and equals 0.02. Our method inserts 26% fewer Steiner points than Delaunay refinement does. The angle distribution shows small angles due to the input. The angles range in $[5^\circ, 128^\circ]$ with our method, vs $[5^\circ, 131^\circ]$ with Delaunay refinement. As expected the upper bound on aspect ratios is similar with both methods: 5.73.

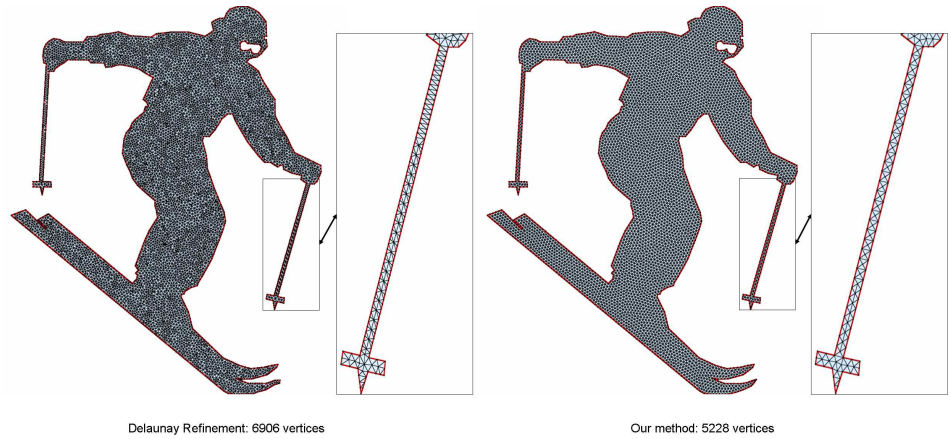


Figure 2.10: The sizing function is uniform and equals 0.01. The input PSLG contained 248 vertices. Our method inserts 25% fewer Steiner points than Delaunay refinement does. The closeup depicts two constrained edges, parallel and close to each other. Combining 1D and 2D optimization allows inserting fewer Steiner points in this area.

inside the domain. In the uniform case, we observe that our method produces meshes with $1.60 m_{\triangle}$ triangles, whereas Delaunay refinement produces meshes with $2.20 m_{\triangle}$ triangles. These values are very consistent over all the examples we have tested. In the non-uniform case, $\bar{\mu}$ depends too much on the gradation of μ and on the domain geometry to be able to exhibit a representative average.

Figures 2.11, 2.12, 2.13 and 2.14 illustrate results from larger inputs and with non-uniform sizing functions, such as the one described by Alliez et al. [ACSYD05] as:

$$\mu(x) = \inf_{s \in \partial\Omega} [kd(s,x) + lfs(s)],$$

where $\partial\Omega$ is the domain boundary, d the Euclidian distance, lfs the local feature size, and k a constant. This sizing function is shown [ACSYD05] to be the maximum k-Lipschitz function that is smaller or equal to lfs on $\partial\Omega$. In the non-uniform case, our algorithm inserts on average 28% fewer Steiner points than the standard Delaunay refinement.

2.5 Summary

In this chapter, we have presented a new 2D triangle mesh generation algorithm, interleaving Delaunay refinement and optimization using the Lloyd iteration. The meshes generated are bound to cover the input constraints provided as a PSLG, and to satisfy sizing criteria provided as a sizing function. Although our algorithm comes with no theoretical bound on the triangle angles, we show experimental evidence that it produces meshes with fewer Steiner points (25% on average) than

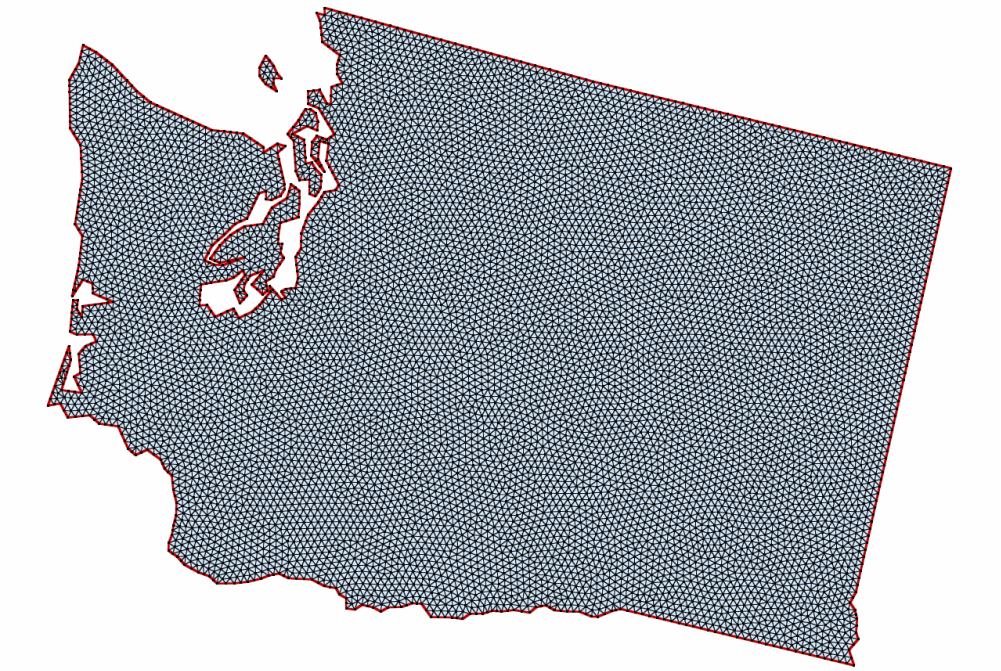


Figure 2.11: The sizing function is uniform and equals 0.01. The initial PSLG contained 256 vertices. The final mesh contains 9761 vertices.

the standard Delaunay refinement algorithm, and of better quality. An efficient and robust algorithm for computing bounded Voronoi diagrams is also provided. We plan to integrate the latter into the CGAL library.

The main added value provided by the interleaved refinement and optimization steps in a multilevel manner is to provide the Lloyd optimization steps with good initial solutions, and therefore to generate meshes of higher quality than the ones optimized once after refinement. More specifically, such interleaving not only contributes to obtain lower minima for the functional energy described in Section 2.2.2, but also prevents the Lloyd iteration to perform long range vertex relocations, and therefore to converge slowly. The final number of Steiner points is (non-trivially) related to the decreasing speed of the "target sizing" parameter used by the refinement step. The ultimate goal being to insert just enough Steiner points (and therefore to generate large well-shaped elements), it is desirable to slow down the decreasing of this parameter, especially when approaching the objective sizing function. Although our experiments in this direction were satisfactory in terms of Steiner points added, the computation times substantially increase.

As future work we plan to derive a "hill-climbing" version of our algorithm, where the Lloyd iteration would be modified so as to move each generator *toward* their centroid while staying within some fixed angle bounds. A challenging goal would be to

provide theoretical bounds greater than the ones provided by Delaunay refinement.

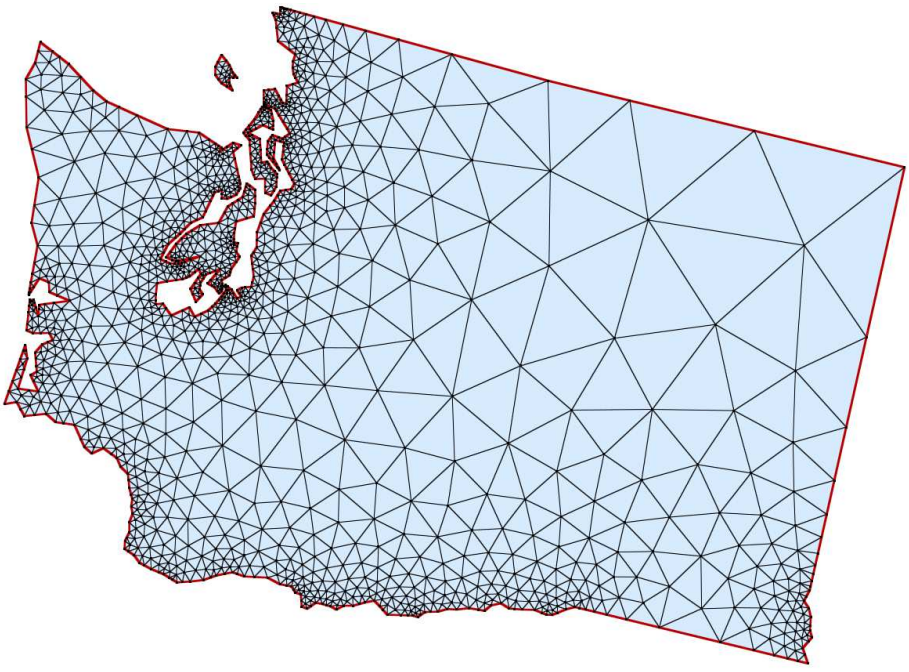


Figure 2.12: Mesh generated with respect to the sizing function $\mu(x) = \inf_{s \in \partial\Omega}[0.45 * d(s,x) + lfs(s)]$. The initial PSLG contained 256 vertices. The final mesh contains 1440 vertices.

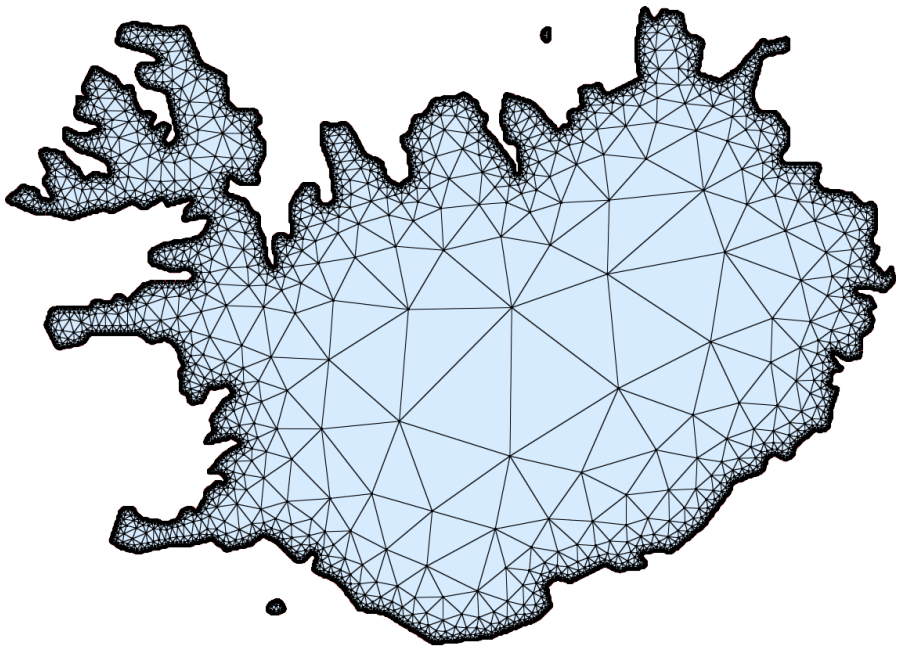


Figure 2.13: Mesh generated with respect to the sizing function $\mu(x) = \inf_{s \in \partial\Omega}[d(s,x) + lfs(s)]$. The initial PSLG contained 3632 vertices. The final mesh contains 9691 vertices.

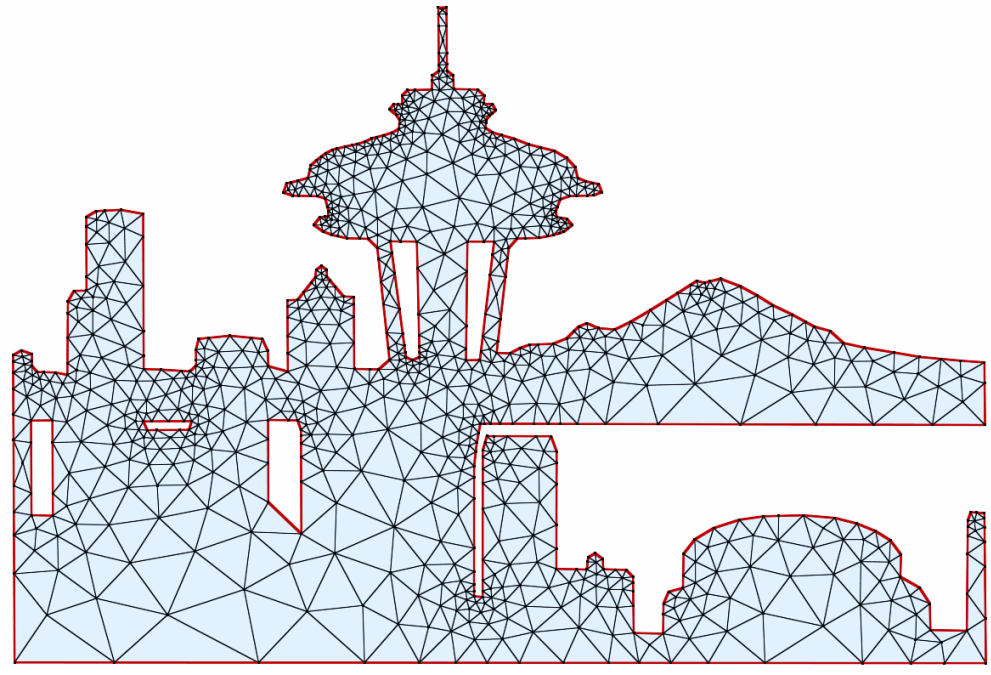


Figure 2.14: Mesh generated with respect to the sizing function $\mu(x)\inf_{s \in \partial\Omega}[0.6 * d(s,x) + lfs(s)]$. The initial PSLG contained 205 vertices. The final mesh contains 981 vertices.

Chapter 3

3D Tetrahedral Mesh Generation

Contents

In this chapter, we introduce a robust, hybrid meshing algorithm to generate high-quality isotropic tetrahedral meshes. Delaunay refinements and variational optimizations are interleaved in order to produce a discretization of the domain that meets a series of desirable geometric and topological criteria, while offering smooth gradation of the resulting well-shaped tetrahedra.

3.1 Related work

Most previous work aimed at generating isotropic tetrahedral meshes were designed around four basic concepts: packing, regular lattices, refinement, and optimization. While packing methods (including advancing front approaches) were initially favored, their relatively high computational complexity and lack of theoretical guarantees have spawned the investigation of alternative methods. Regular lattices have been at the core of some of the fastest meshing techniques, as they provide a blazingly fast approach to meshing most of the domain. While smooth surface boundaries can be efficiently handled with guaranteed minimum dihedral angles [LS07], the regularity of the mesh resulting from these methods (*i.e.*, the presence of preferred edge directions) can induce severe aliasing effects in simulation [WBOL07].

Techniques combining Delaunay triangulation and refinement have received special attention due to their versatility and theoretical foundations. They have

been used initially in 2D [Che89b], then in 3D for polyhedral domains [NCC02, AHMP07], for smooth surfaces [Che93], for 3D domains bounded by smooth surfaces [ORY05, BOG02] and for 3D domains bounded by piecewise smooth surfaces [RY07, CDL07, CDR07]. They proceed by refining and filtering a 3D triangulation until a set of user-specified criteria is satisfied. Refinement is achieved

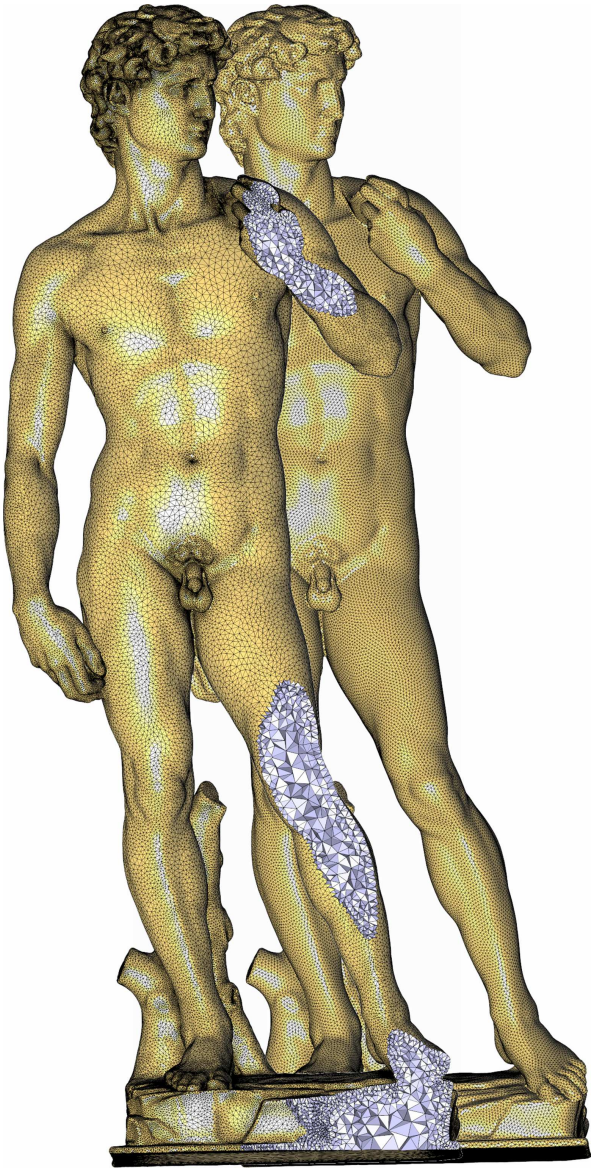


Figure 3.1: Michelangelo's David. Our mesh generation algorithm produces high quality meshes through Delaunay refinements interleaved with optimization. The input PSC has 800K triangles; (right) a uniform sizing criterion generates a 1M vertices mesh; (left) approximation error and shape criteria alone generate a smaller graded mesh (250K vertices), while guaranteeing the same mesh quality and a better local approximation error.

through iterative insertion of Steiner points, either inside the domain or on the domain boundary, to meet the desired criteria. A filtering process is applied to cull simplices such that the triangulation restricted to the input domain tessellates the domain, and such that the boundary of this restricted triangulation approximates the domain boundary. This procedure becomes more delicate for piecewise-smooth inputs (a more general class of domains where the boundary is a collection of smooth patches meeting at potentially sharp creases), as sharp creases require additional care. Non-smooth regions subtending small angles add another level of difficulty for Delaunay refinement. Refinement techniques are usually judged on the quality of the resulting mesh elements and on the sparsity of Steiner point insertion.

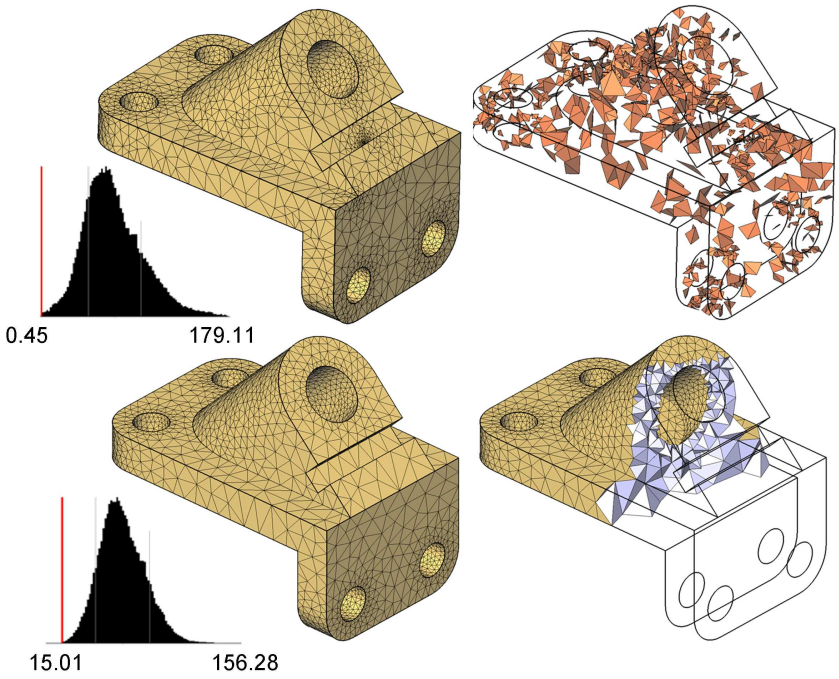


Figure 3.2: Top: Mesh (5,499 vertices) generated by Delaunay refinement (shape and boundary approximation criteria activated). Notice the cluster in the middle of the armhole. Right image shows tetrahedra with dihedral angles smaller than 15 degrees. Bottom: Mesh (3,701 vertices) generated by interleaving Delaunay refinement and optimization so as to satisfy the same criteria. Distributions of dihedral angles are shown on the left.

The quest for ever better quality meshes has sparked advances in mesh optimization through local vertex relocation to optimize a specific notion of mesh quality [ABE97, PS04], topological operations [CDE+00], or both [FOG97]. Further improvement of the mesh quality can be achieved by inserting additional vertices and/or incorporating a rollback mechanism to undo previous optimizations in order to guarantee a monotonic increase in mesh quality [KS07]. Among the large body of work in mesh optimization, the *Optimal Delaunay Triangulation* approach (ODT

for short) stands out, as it casts both geometric and topological mesh improvement as a single, unified functional optimization [CX04b, Che04] that tries to minimize in $\mathbb{R}^4$ the volume between a paraboloid and the linear interpolation of the mesh vertices lifted onto the paraboloid. This approximation-theoretical method to obtain isotropic meshes was adapted for tetrahedral meshing of 3D domains [ACSYD05], mixed with a constrained Lloyd relaxation on the domain boundary. While this technique was shown to only produce nicely-shaped tetrahedra throughout the domain, slivers (i.e., nearly degenerate elements) could appear near the domain boundary, as the boundary vertices were guided by Lloyd relaxation and were thus unaffected by the 3D optimization. Furthermore, this method lacks a number of useful features. First, the algorithm is not designed to satisfy the type of user-defined criteria commonly handled by Delaunay-based mesh generation techniques [RY07]. Also, an estimate of the boundary local feature size (lfs) is required to derive a sizing function; however, there is currently no consensus on how to extend the notion of lfs to polyhedral domains. Finally, this method cannot handle arbitrary boundary meshes, requiring a *restricted* Delaunay triangulation instead.

Contributions We combine the efficacy of Delaunay refinement methods with the isotropic quality induced by optimal Delaunay optimization techniques (extending the 2D approach of Chapter 2) to provide a practical, high-quality meshing algorithm for 3-Dimensional domains bounded by piecewise smooth boundaries. This combination of techniques is motivated by the desire to maximize mesh quality while reducing mesh size.

Delaunay refinement alone tends to generate overly complex meshes, with, *e.g.*, spurious clusters of vertices due both to the greedy nature of the algorithm and to encroachment mechanisms; interleaving parsimonious refinement and mesh optimization instead turns out both to reduce the number of Steiner points and to improve the overall mesh quality (see Figure 3.2). Unlike previous mesh optimization methods which either consider the boundary fixed or use boundary conditions incompatible with global mesh improvement, we introduce a consistent variational treatment applied to both interior and boundary nodes, improving the overall quality of the mesh. To speed up Delaunay refinement and make it parsimonious, we select subsets of isolated Steiner points using the probabilistic multiple choice approach [WK02] to reduce the treatment of short-lived primitives and provide independent refinements before each round of optimization. The practicality of our approach further stems from additional, distinctive features. First, we only rely on simple intersection tests to probe the domain boundary to make the approach as generic as possible with respect to the boundary surface representation. Second, we do not require a mesh sizing function as input and provide instead a dynamic sizing function which evolves throughout refinement until all user-specified criteria are satisfied.

Finally, the method is versatile enough to serve as a general framework for isotropic tetrahedral meshing, as each step involved in the process can be adapted to special requirements.

3.2 Algorithm

3.2.1 Interleaving Refinement and Optimization

The algorithm we now detail interleaves refinement and optimization of an initial 3D Delaunay triangulation [TWAD09]. Mesh simplices are gradually improved to meet user-defined criteria on boundary approximation and on the shapes and sizes of elements through refinements, while passes of optimization further improve the shape of the elements. The high-level pseudo-code is as follows:

Algorithm 5 Mesh generation at a glance

Input: *Domain* $\Omega \in \mathbb{R}^3$ (Section 3.2.1.1)
and a set $\{k_1, k_2, ..., k_n\}$ *of user-defined criteria* (Section 3.2.1.2).
Initialize coarse mesh $\mathcal{M}$ (Section 3.2.1.3)
while Criteria $\{k_1, k_2, ..., k_n\}$ not all met **do**
Refine through sparse vertex insertions (Section 3.2.1.4)
Optimize mesh (Section 3.2.1.5)
Perturb remaining slivers (Section 3.2.2)

3.2.1.1 Input

The input is a 3-Dimensional domain Ω whose boundary $\partial\Omega$ is defined as a piecewise smooth complex (PSC). More specifically, our current implementation takes as input a piecewise linear approximation of a PSC. This approximation is provided as a triangle surface mesh, watertight, and forming a 2-manifold with no self-intersection. In addition, we assume that sharp edges as well as feature vertices of this mesh are tagged. Dart (resp., corner) vertices are deduced from tagged sharp edges as they are incident to one (resp., three or more) sharp edges. Tip and cusp vertices, which are incident respectively to zero and two sharp edges, cannot be derived solely using the sharp edge tags and hence must be specified by the user. By chaining sharp edges together, we obtain a set of polylines that we will refer to as *creases*. A crease may either connect two feature vertices or form a cycle. All creases are enumerated, and each sharp edge of the input surface mesh is marked with the index of its associated crease. Finally, we identify and enumerate surface *patches* as connected components of the boundary, bounded (or not) by sharp creases. Each face of the input surface mesh is marked with the index of its associated patch as depicted in Figure 3.3. The sharp input creases subtending angles should not be too small (the

theoretical bound is 90 degrees) to have a guarantee that Delaunay refinement steps will terminate (see [RY07]).

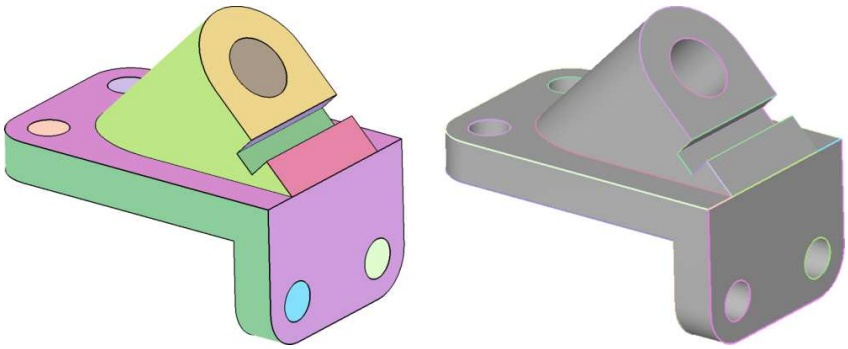


Figure 3.3: Input piecewise smooth complex (PSC) with all surface patches (left) and sharp creases (right) enumerated.

3.2.1.2 Parameters

The user selects a set of criteria that the final mesh must satisfy. These criteria, which accommodate the typical user requirements for mesh generation, are used to guide the refinement process as explained in Section 3.2.1.4. All of them but the first criterion are optional in our implementation:

- *Sizing:* a spatially-varying sizing function (or possibly a single value if constant) indicates the maximum mesh edge length allowed within the domain.

- *Approximation:* an approximation control function defines a local upper bound $\epsilon_{\max}$ for the surface or crease approximation error. Similar to the mesh sizing function, it is defined either as a single value if the function is constant over the boundary, or as a spatially-varying scalar function.

- *Shape:* two global element shape quality bounds are defined as the maximum circumradius to shortest edge ratio allowed in the final mesh. We denote by ${\sigma^f}_{\max}$ and ${\sigma^t}_{\max}$ these bounds for facets and tetrahedra, respectively.

- *Topology:* a Boolean flag determines whether the topology of the input PSC should be preserved, *i.e.*, if the vertices of each restricted facet must belong to the same patch, and if the vertices of each restricted edge must belong to the same crease.

- *Manifold:* a Boolean flag determines whether the final mesh boundary should be a two-manifold surface.

3.2.1.3 Initialization

A first mesh $\mathcal{M}$ of the domain is obtained by inserting in $\mathcal{M}$ all *feature vertices* (corners and such) of the input surface mesh. These vertices remain untouched throughout the mesh generation procedure. We also add the eight corners of a large bounding box of the input domain, in order not to have to deal with infinite Voronoi faces in later stages. Finally, we ensure that each surface patch and each crease have received the minimal number of sample points to seed the refinement process by adding more vertices if necessary, as in [RY07]. The mesh $\mathcal{M}$ is defined to be the Delaunay mesh of all these vertices. Finally, we refine this initial mesh with respect to looser criteria than those defined by the user (typically, we relax the various input criteria parameters by a factor two), using our refinement procedure that we detail next.

3.2.1.4 Refinement

The refinement process is entirely driven by the user-defined criteria listed in Section 3.2.1.2. Each refinement step is designed to remove a set of *bad elements* (simplices not satisfying at least one of the given criteria) by inserting so-called *Steiner vertices* to $\mathcal{M}$. Unlike typical Delaunay refinement techniques that insert one Steiner point at a time, we proceed in batches of refinement, inserting a sparse subset of all the candidate Steiner points per batch (see Fig. 3.5).

Bad Elements The simplices considered for refinement are the so-called *restricted simplices*, that is, the ones considered as inside the domain Ω or on $\partial\Omega$—namely, tetrahedra whose dual Voronoi vertex is located inside Ω, facets whose dual Voronoi edge intersects $\partial\Omega$ and edges whose dual Voronoi facet intersects an input crease. We consider one of these restricted elements bad if it violates one of the following criteria:

- *Size:* A restricted edge is considered bad if it is longer than the sizing function evaluated at its midpoint. A restricted facet or a tet is considered bad if at least one of its edges is badly sized.

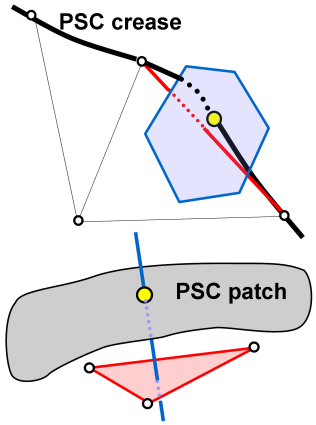
- *Approximation error:* A restricted edge e is considered bad if the distance from its midpoint to the farthest intersection point between its dual Voronoi face and an input crease is larger than the local approximation bound. Similarly, a restricted facet f is considered bad if the distance from f's circumcenter to the farthest intersection point between its dual Voronoi edge and $\partial\Omega$ is larger than the approximation bound.

- *Shape.* A restricted facet (resp., tetrahedron) is considered bad if the ratio of its circumradius to shortest edge is higher than the user-specified bound ${\sigma^f}_{\max}$ (resp., ${\sigma^t}_{\max}$).

- *Topology.* A restricted edge (resp., facet) is considered as not capturing the proper topology if its two (resp., three) vertices do not belong to the same input crease (resp., surface patch). If the topology criterion is activated, we store for each vertex v of the mesh its location with respect to the input PSC. That is, each vertex is tagged either as an *interior*, a *feature* (e.g. corner), a *crease*, or a *boundary* (i.e. surface) vertex. In the last two cases, the index of the feature (crease or surface patch) is stored too.

In addition to these types of bad elements, we add an extra one to enforce the *topological disk condition* [RY07] as it is an important indicator of topological conformity of the mesh to the input domain. For a vertex v tagged as boundary (*i.e.*, on an input surface patch), the topological disk condition is satisfied iff the boundary facets incident to v form a topological 2-disk. If v belongs to an input crease, its incident restricted edges (edges whose dual Voronoi facet intersects input creases) have to form a topological 1-disk. We thus mark every boundary vertex of the mesh whose topological disk condition is not satisfied as bad as well.

Steiner Vertices For each bad simplex, we define its associated Steiner point location. The associated Steiner point to a restricted*edge* is the farthest intersection point between its dual Voronoi face and the input creases. The associated Steiner point to a restricted *facet* is the farthest intersection point between its dual Voronoi edge and $\partial\Omega$. The associated Steiner point to a restricted tetrahedron is its circumcenter. Finally, for each boundary vertex of the mesh whose topological disk condition is not satisfied, we define its associated Steiner point to be the Steiner point of the facet (resp., crease edge) incident to v that realizes the largest approximation error: its insertion will help enforce the topological disk condition.



To ensure termination of the refinement process, we further check for *encroachment* [She02b, CDL07, RY07]. The candidate Steiner point p of a tetrahedron is said to *encroach* a boundary facet f if it is inside its restricted Delaunay ball (centered at f's Steiner point and passing through the vertices of f). Similarly, the Steiner point of a facet is said to *encroach* on a crease edge if it is inside its restricted Delaunay ball (centered at its Steiner point and passing through its endpoints). In these two cases of encroachment, we alter the position of the associated Steiner point, replacing it by the Steiner point of the encroached primitive (and recursing the encroachment check).

Independent Set Refinement To help define a good subset of Steiner points to add in batch, we introduce the notion of *conflict regions* and *independent sets of*

conflict regions. For each Steiner point p, we call the "conflict region" the tetrahedra that would be affected by its insertion as well as their adjacent tetrahedra: these elements are likely to be destroyed by the insertion of p. We call an "independent set" of conflict regions a set that does not contain overlapping conflict regions, so that none of the insertions of these selected Steiner points would influence each other. We construct such an independent set of conflict regions as described in Algorithm 6: we iteratively select Steiner points *in order of increasing dimension* of their associated simplices. That is, first crease edges are collected and sorted from worst to best. As many crease-edge Steiner points as possible are inserted into the set, along with their conflict regions, while making sure there is no overlap of conflict regions. Second, we similarly treat boundary facets. Finally, tetrahedra are handled; however, as there can be a large number of bad tetrahedra during the meshing process, the same process of sorting elements before choosing them would be too costly. We therefore process bad tetrahedra through a more efficient multiple-choice approach as explained next, and this is done iteratively until no Steiner point can be inserted to the independent set without overlapping the regions already inserted. Figure 3.4 shows an independent set on the mesh of a cylinder for which only the approximation criterion is not yet satisfied.

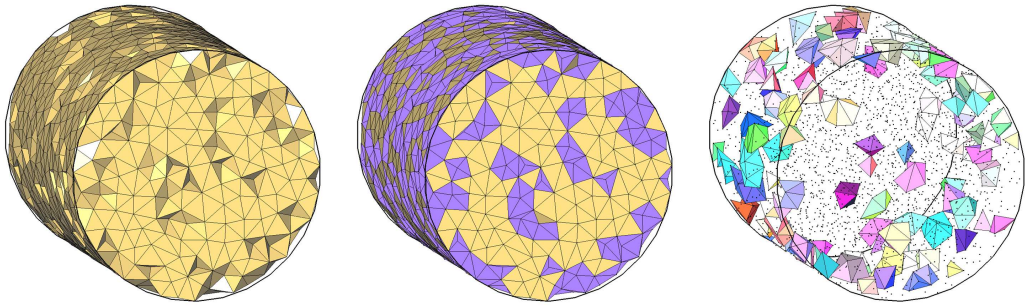


Figure 3.4: Independent set of conflict regions. Left: Mesh of a cylinder obtained by Delaunay refinement based only on an edge length criterion. Middle: Facets that do not satisfy the approximation error criterion. Right: The computed independent set of conflict regions.

Multiple-Choice Selection of Tetrahedra Although many Delaunay refinement algorithms use modifiable priority queues to store all bad simplices of the mesh $\mathcal{M}$, most queue elements are short-lived as each Steiner point insertion affects its surrounding. In fact, our experiments consistently showed that the computational burden spent maintaining the global priority queue of all bad simplices is overly high compared to the number of primitives actually refined. We thus depart from the usual refinement strategy by using a multiple choice approach (proposed for mesh decimation by Wu and Kobbelt [WK02]) as follows. At each step, a small container of N_{mc}"bad" tetrahedra (N_{mc}= 20 in our implementation) is filled with randomly selected non-conflicted tetrahedra. The worst tetrahedron in this container is then

selected, and the container is updated with another random non-conflicted tetrahedron. As our goal is to only sparingly refine the mesh before further optimization, this multiple-choice approach significantly speeds up our refinement process while preserving its overall performance.

Algorithm 6 Construction of Independent Set of Conflict Regions

Input: *A PSC as input domain,*
a coarse initialization of the mesh, and
a set $K = \{k_i\}_i$ *of criteria to be met.*

Set Independent Set IS to *nil*.
Collect all restricted tets in T_{bad}.
Collect all bad crease edges in E_{bad}, bad boundary facets in F_{bad}.
for each bad simplex s in E_{bad} and F_{bad} (from worst to best), **do**
  Let p be the Steiner point of s.
  Let U_c be the set of all tets in direct conflict with p's insertion.
  Let U_n be the set of all tets sharing a facet with a tet in U_c.
  if No tetrahedron of $U_c \cup U_n$ is in the Independent Set IS, **then**
    Insert conflict region $U_c \cup U_n$ in IS along with p.
Let C_{mc} be a multiple-choice container of N_{mc} tets.
while There are non-conflicted tets **do**
  Fill up $C_{mc}$with random tets from T_{bad} which Steiner points'
  conflict regions $U_c \cup U_n$ do not intersect regions already in IS.
  Add Steiner point of C_{mc}'s worst tet & its conflict region to IS.
Batch-insert all Steiner points stored in IS to mesh.
Update restricted Delaunay triangulation.

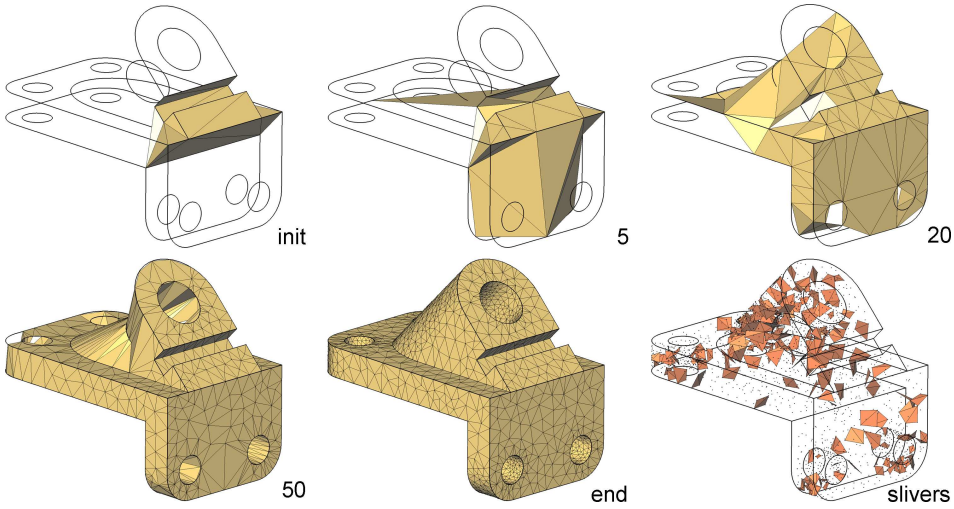


Figure 3.5: Refinement steps without optimization. The mesh initialized with feature vertices; after a few batch refinement steps (from 5 to 50); the final refined mesh with shape and approximation criteria satisfied; and its 244 slivers (tetrahedra with dihedral angle smaller than 10 degrees).

3.2.1.5 Optimization

Chen [Che04] defines an *Optimal Delaunay Triangulation (ODT)* as the minimizer of the energy

$$E_{\text{ODT}} = ||f_{\text{PWL}}^{primal} - f||_{\mathcal{L}^1} = \sum_{T_j \in \mathcal{DT}} \int_{T_j} |f_{\text{PWL}} - f|,$$

where $f(\mathbf{x}) = ||\mathbf{x}||^2$ and f_{PWL} is the linear function interpolating the values of f at the vertices of each tetrahedron T_j. This energy has a simple geometric interpretation: it is the volume between the 4D paraboloid defined by f and its inscribed piecewise linear approximation f_{PWL} through lifting the triangulation onto the paraboloid [BWY07]. Because of a result of function approximation theory [She02d] stating that the best interpolating approximation of a function is achieved when the elements' size and orientation match the Hessian of the function, an ODT is thus isotropic.

The energy E_{ODT} can be reformulated [ACSYD05] as

$$E_{\text{ODT}} = \frac{1}{4} \sum_{\mathbf{x}_i \in T_j} {\mathbf{x}_i}^2 |\Omega_i| - \int_{\mathcal{M}} \mathbf{x}^2 d\mathbf{x}, \tag{3.1}$$

where $|\Omega_i|$ is the volume of the 1-ring neighborhood of vertex $\mathbf{x}_i$. Noting that the last term is constant given a fixed boundary $\partial\mathcal{M}$, and that $|\Omega_i| = \sum_{T_j \in \Omega_i} |T_j|$, a derivation of this quadratic energy in $\mathbf{x}_i$ leads to the following *optimal position* $\mathbf{x}_i^\star$ of interior vertex $\mathbf{x}_i$ in its 1-ring [Che04]:

$$\mathbf{x}_i^\star = -\frac{1}{2\,|\Omega_i|} \sum_{T_j \in \Omega_i} \left(\nabla_{\mathbf{x}_i} |T_j| \left[\sum_{\substack{\mathbf{x}_k \in T_j \\ \mathbf{x}_k \neq \mathbf{x}_i}} ||\mathbf{x}_k||^2 \right] \right). \tag{3.2}$$

The term $\nabla_{\mathbf{x}_i} |T_j|$ is the gradient of the volume of the tetrahedron T_j with respect to $\mathbf{x}_i$. Replacing the paraboloid function $f(\mathbf{x}) = ||\mathbf{x}||^2$ by the translated function $f(\mathbf{x}) = ||\mathbf{x} - \mathbf{x}_i||^2$, does not change the interpolation error, leading to the same optimal position. We thus get the following equivalent expression used to update a vertex position :

$$\mathbf{x}_i^\star = \mathbf{x}_i - \frac{1}{2\,|\Omega_i|} \sum_{T_j \in \Omega_i} \left(\nabla_{\mathbf{x}_i} |T_j| \left[\sum_{\mathbf{x}_k \in T_j} ||\mathbf{x}_i - \mathbf{x}_k||^2 \right] \right). \tag{3.3}$$

We also know that, given $\mathbf{x}_k$ a vertex of T, and f_k its opposite facet in T, the volume of T is $|T| = \frac{1}{3}\, d(\mathbf{x}_k,\ f_k)\ |f_k|$, where d is the Euclidian distance. By taking

its gradient, we obtain

$$\nabla_{\mathbf{x}_k}|T| = \frac{1}{3}|f_k|\nabla_{\mathbf{x}_k} d(\mathbf{x}_k, f_k) = \frac{1}{3}|f_k|n_k,$$

were n_k is the unit normal to f_k pointing towards $\mathbf{x}_k$. Most importantly, notice that $\nabla_{\mathbf{x}_k}|T|$ is independent on the location of $\mathbf{x}_k$.

From what we have seen, we get $\sum_{T_j \in \Omega_i} \nabla_{\mathbf{x}_i}|T_j| = 0$. Thus, it follows that when all $||\mathbf{x}_i - \mathbf{x}_k||^2$ are equal, $\mathbf{x}_i^\star = \mathbf{x}_i$. In other words, when the neighbors of $\mathbf{x}_i$ lie on a sphere with center $\mathbf{c}$, $\mathbf{x}_i^\star = \mathbf{c}$; we call this property the *ODT circumsphere property*.

As a special case of this property, the optimal position of a vertex that has only four neighbors is exactly *at* T's circumcenter, denoted $\mathbf{c}_T$. Using Eq. (3.3) in this special case of a 1-ring in the shape of a tetrahedron $T = (\mathbf{x}_p, \mathbf{x}_q, \mathbf{x}_r, \mathbf{x}_s)$, and taking the point $\mathbf{x}_i$ to be located at $\mathbf{x}_p$, we get:

$$\begin{aligned}\mathbf{c}_T = \mathbf{x}_p - \frac{1}{2\,|T|}\Bigg[\nabla_{\mathbf{x}_p}|T|\,\Big[\sum_{\mathbf{x}_k \in T} ||\mathbf{x}_p - \mathbf{x}_k||^2\Big] \\ +\ \mathrm{F}(\mathbf{x}_p, \mathbf{x}_q, \mathbf{x}_r) + \mathrm{F}(\mathbf{x}_p, \mathbf{x}_q, \mathbf{x}_s)\ +\ \mathrm{F}(\mathbf{x}_p, \mathbf{x}_r, \mathbf{x}_s)\ \Bigg]\end{aligned} \tag{3.4}$$

where the extra terms on the right-hand side only depend on each face of the tetrahedron (because, as we took $\mathbf{x}_i$ to be at $\mathbf{x}_p$, all but one of the tetrahedra inside T are degenerate and become *faces* of T). More precisely, these terms are explicitly given as:

$$\mathrm{F}(\mathbf{x}_p, \mathbf{x}_q, \mathbf{x}_r) = +\frac{1}{3}\Big[||\mathbf{x}_p - \mathbf{x}_q||^2 + ||\mathbf{x}_p - \mathbf{x}_r||^2\Big]\mathbf{N}_{p,q,r}$$

where $\mathbf{N}_{p,q,r}$ is the area-weighted normal of the face (p, q, r) pointing towards the inside of the tetrahedron, *i.e.*, $\mathbf{N}_{p,q,r} = |(p, q, r)|\ \mathbf{n}_{p,q,r}$. Now, go back to Eq. (3.3) for an arbitrary 1-ring centered on $\mathbf{x}_p$, and note that the term in parenthesis appears as is (for $p \equiv i$) in Eq. 3.4. Substitute this term by the circumcenter and all the other terms that Eq. 3.4 contains. All the face terms F cancel each other out, thus simplifying the expression to:

$$\mathbf{x}_i^\star = \frac{1}{|\Omega_i|}\sum_{T_j \in \Omega_i} |T_j|\mathbf{c}_j. \tag{3.5}$$

Natural ODT for Boundary Vertices While [Che04, ACSYD05] do not involve the boundary vertices in the minimization of the ODT energy, we propose an extension that changes the update of boundary vertices during optimization so as to further reduce the total energy, thus providing a boundary extension to the

original ODT mesh smoothing procedure. Denote by $\mathbf{x}_i$ a vertex on the *boundary* of a 3D mesh (i.e., it does not have a full 1-ring $\mathcal{N}(\mathbf{x}_i)$ of restricted tetrahedra). For a given connectivity, the new position $\mathbf{x}_i^\star$ of $\mathbf{x}_i$ that extremizes the ODT energy is a bit more complicated, as some of the face terms F do not disappear:

$$\mathbf{x}_i^\star = \left[\left(\sum_{T \in \mathcal{N}(\mathbf{x}_i)} |T| \mathbf{c}_T \right) + B \right] / \sum_{T \in \mathcal{N}(\mathbf{x}_i)} |T|,$$

where the boundary terms B are

$$B = \frac{1}{6} \left(\sum_{(i,p,q) \in \partial\mathcal{M}} \mathbf{N}_{i,p,q} \left[||\mathbf{x}_i - \mathbf{x}_q||^2 + ||\mathbf{x}_i - \mathbf{x}_q||^2 \right] \right).$$

The first part ($|T|\mathbf{c}_T$) is the weighted barycenter of the circumcenters divided by the total 1-ring volume just as before. At the boundary appears an extra term, a sum over boundary triangles (i, p, q) involving the squared length of the "spokes" of the triangle 1-ring. This formula, applied as is, shrinks the domain as it obviously decreases the total energy. However, as seen previously, we can assign a multiplicative weight λ to the supplementary term B without affecting the update rule for internal vertices, because the boundary terms cancel each other out for a full 1-ring:

$$\mathbf{x}_i^\star = \left[\left(\sum_{T \in \mathcal{N}(\mathbf{x}_i)} |T| \mathbf{c}_T \right) + \lambda B \right] / \sum_{T \in \mathcal{N}(\mathbf{x}_i)} |T|.$$

We now use set this extra degree of freemdom λ so as to retain the *ODT circumsphere property* mentioned earlier, but now in the case of an incomplete 1-ring: if all neighbors of $\mathbf{x}_i$ are at the same distance from $\mathbf{x}_i$, we want $\mathbf{x}_i^\star = \mathbf{x}_i$: we will thus obtain a formula valid for both the complete 1-ring and incomplete 1-ring cases, while preserving the ODT circumsphere property. We have

$$\mathbf{x}_i^\star = \mathbf{x}_i - \frac{1}{2|\Omega_i|} \sum_{T_j \in \Omega_i} \left(\nabla_{\mathbf{x}_i} |T_j| \Big[\sum_{\mathbf{x}_k \in T_j} ||\mathbf{x}_i - \mathbf{x}_k||^2 \Big] \quad + (1-\lambda) \sum_{p,q \neq i} \mathrm{F}(\mathbf{x}_i, \mathbf{x}_p, \mathbf{x}_q) \right). \tag{3.6}$$

Consider the case where all $||\mathbf{x}_i - \mathbf{x}_k||^2$ are equal to some constant R. We want $\mathbf{x}_i^\star = \mathbf{x}_i$ and we know, from the divergence theorem applied on the 1-ring of the boundary vertex, that

$$\nabla_{\mathbf{x}_i} |T_j| = - \sum_{p,q \neq i} \frac{1}{3} \mathbf{N}_{i,p,q}.$$

On the one hand, $\nabla_{\mathbf{x}_i} |T_j|$ is weighted by $3R$ in (3.6). On the other hand, each $\frac{1}{3}\mathbf{N}_{i,p,q}$ is weighted by $2R$ in (3.6). Enforcing the circumsphere property on partial

1-rings at the boundary thus requires $(1-\lambda) = 3/2$, i.e., $\lambda = -1/2$. The optimal position for this variant (denoted NODT for *Natural ODT*) is now computed as

$$\mathbf{x}_i^\star = \left[\left(\sum_{T \in \mathcal{N}(\mathbf{x}_i)} |T| \mathbf{c}_T \right) - \frac{1}{2} B \right] / \sum_{T \in \mathcal{N}(\mathbf{x}_i)} |T|,$$

where the boundary terms B (if any) are

$$B = \frac{1}{6} \left(\sum_{(i,p,q) \in \partial\mathcal{M}} \mathbf{N}_{i,p,q} \left[||\mathbf{x}_i - \mathbf{x}_p||^2 + ||\mathbf{x}_i - \mathbf{x}_q||^2 \right] \right).$$

Variable Sizing The optimization formula above is only valid for generating uniform isotropic meshes. To account for a variable mesh sizing, we update a dynamic mesh sizing function after each batch of refinement, and replace all measures in above formulas (lengths, areas, volumes) by measures in the metric of the sizing function. Such measures are obtained by quadratures over the mesh elements. This sizing function [ADA07] is guaranteed to be K-Lipschitz and is initialized with values computed by averaging the lengths of the mesh edges incident to all mesh vertices. Intuitively, the refinement is in charge of discovering the local feature size of the domain boundary. One of the user-defined criteria triggers a local refinement of the mesh, which induces an update of the sizing function, which in turn imposes further refinements to maintain the K-grading of the mesh. The optimization part of the algorithm then takes the current sizing function as input to avoid the undoing of local refinements that a uniform sizing would produce.

Restriction and Projection In practice, as we want the mesh to interpolate the domain, each boundary vertex of the mesh should be on $\partial\Omega$. To enforce this property, the new location $\mathbf{x}_p^\star$ of $\mathbf{x}_p$ is projected onto $\partial\Omega$. Two cases are distinguished: $\mathbf{x}_p^\star$ can belong to a surface patch, or to a sharp feature of the mesh. If at least one of the incident edges to $\mathbf{x}_p$ is a *crease edge* (i.e., its dual Voronoi facet intersects a PSC crease), then we project $\mathbf{x}_p^\star$ onto the closest crease. Similarly, if at least one of the incident facets to $\mathbf{x}_p$ is a boundary facet (i.e., its dual Voronoi edge intersects the PSC), we project $\mathbf{x}_p^\star$ onto the closest facet of the input PSC.

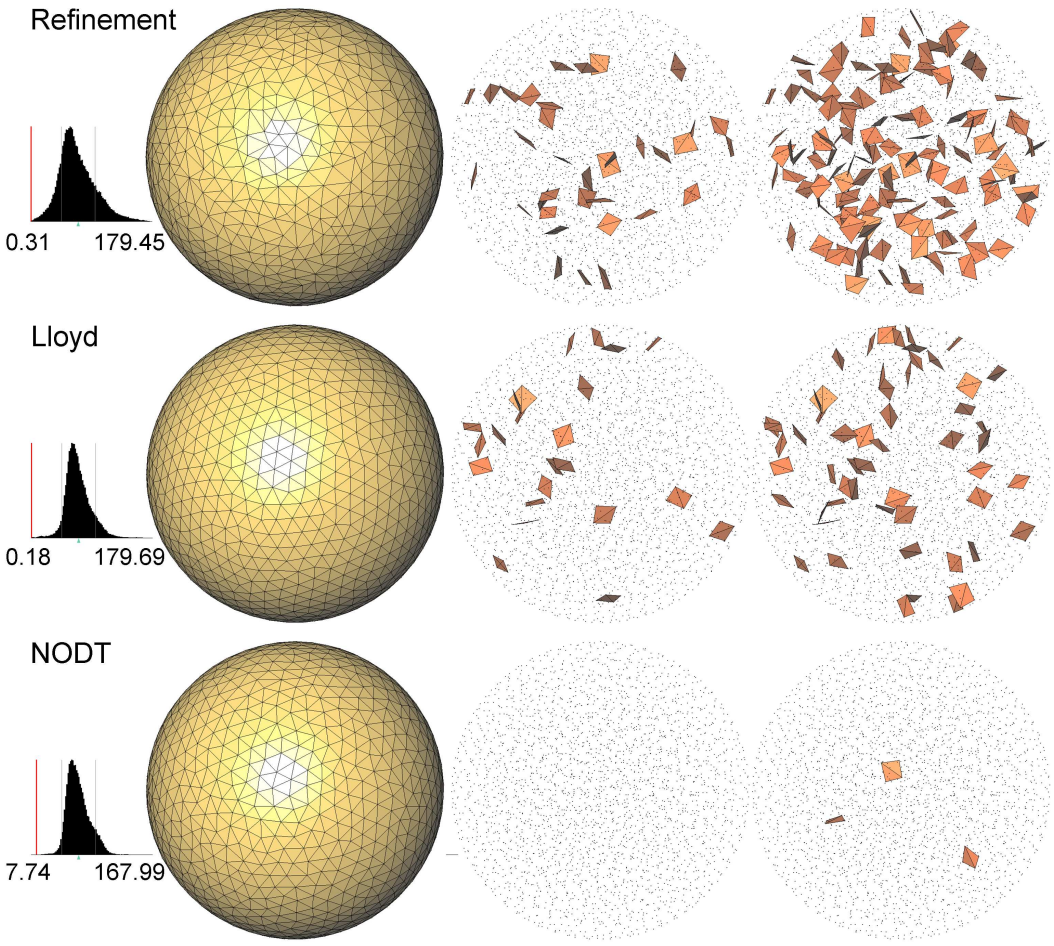


Figure 3.6: Comparing Delaunay refinement and mesh optimization. Distributions of dihedral angles are shown to the left. Slivers are shown for a dihedral angle bound of respectively 5 (middle) and 10 degrees (right). Top: Delaunay Refinement alone (resp. 35 and 136 slivers). Middle: Optimized mesh with 100 Lloyd iterations (resp. 23 and 55 slivers). Bottom: Optimized mesh with 100 NODT iterations (resp. 0 and 3 slivers).

3.2.2 Sliver removal

The only bad elements that remain after Delaunay refinement are slivers. A sliver tetrahedron is formed by evenly placing its 4 vertices near the equator of its circumsphere (see Figure 1.5). In such a sliver the smallest dihedral angle can be very close to 0°, and a numerical simulation may be far from accurate in the presence of slivers.

While our NODT boundary treatment significantly reduces the number of slivers compared to the results reported in [ACSYD05], we cannot guarantee a total absence of slivers (see Figure 3.8). We thus perform a final phase of sliver removal. We implemented an explicit perturbation inspired by [Li00a] which performed better than sliver exudation [CDE+00] in all our experiments. This phase applies small perturbations to each vertex incident to slivers. We first try to move along the gradient of the squared circumradius of the sliver; if unsuccessful, the volume gradient is tried. If neither of these perturbations remove the sliver without adding new ones, random perturbations are applied repeatedly.

3.2.2.1 Previous Work

The problem of removing slivers from a 3D Delaunay mesh has received some attention over the last decade. Delaunay refinement gets so close to providing a perfect output that removing the leftover slivers is generally performed as a post-processing step that is worth it. Previous work on removing and avoiding the creation of slivers can be classified into three parts: The Delaunay-based methods, the weighted Delaunay-based methods, and the non-Delaunay methods. For each part, post-processing steps and complete mesh generation algorithms can be studied. This section focuses on a post-processing step, devised to take as input a Delaunay mesh and to improve its quality in terms of dihedral angles.

Delaunay-based

Vertex Perturbation Li [Li00a, ELM+00] proposes to explicitly perturb the vertices incident to a sliver in an almost-good mesh, by locally relocating them so as to remove the incident slivers. The idea is based on the fact that, for any triangle qrs, the region of locations of the vertex p such that the tetrahedron $pqrs$ is a sliver, is very small. Moving the point p out of this region ensures that the tetrahedron is not a sliver anymore, or has disappeared once the Delaunay connectivity is updated. This is achieved by moving the point p to a new location inside a small ball centered at p, whose radius is proportional to the distance from p to its the nearest neighbor. The authors show that for certain values of the involved parameters, there always exists some points in this ball which are outside all regions that form slivers with nearby triangles. Li uses the union graph concept to avoid circular dependencies on

vertex perturbations. The following theorem [Li00a] proves the existence of such a point that makes the mesh locally sliver-free.

Theorem 3.2.1 (Sliver theorem [Li00a]). *If every simplex in a Delaunay triangulation has radius-edge ratio of at least ρ_0, then there is a constant $\sigma_0 > 0$ and a very mild perturbation S' such as the volume per cube of shortest edge length $\sigma(\tau) \geq \sigma_0$ for each tetrahedron τ in the perturbed triangulation.*

Based on this theorem, Li proposes an algorithm that applies mild random perturbations to the mesh until one which removes slivers is found. One drawback of the above result is the pessimistic theoretical estimate of the bounds on the involved parameters. These bounds are either too small or too large to have any significance. In practice, though this technique is very effective, when targeting a large bound on the minimum dihedral angle (e.g. 15°), the average number of trials of random perturbations required is very large. In our experiments, it is not rare to apply hundreds of random perturbation trials on a single vertex before succeeding in removing a sliver. This number is not surprising when seeking a high minimum dihedral angle such as 15° since the corresponding tetrahedron is not a real sliver anymore. However the fact that the perturbation succeeds even for a high minimum dihedral angle is at the core of our motivation. Finally, the fact that this method always maintains the mesh as a true Delaunay triangulation makes it both robust and practical.

Vertex insertion One strategy for sliver removal is to insert vertices, in order to cause connectivity changes inside the Delaunay triangulation. Usual Delaunay refinement shape criterion is the radius-edge ratio, which does not detect slivers. Labelle proposes a vertex insertion strategy [Lab06] based on a dihedral criterion to characterize bad tetrahedra, and in particular slivers. In presence of slivers, points located on two different regular grids are inserted to the mesh. The algorithm is complex, but has good guarantees on output dihedral angles, which provably are in the interval $[30°; 135°]$.

Sliver-free mesh generation. Some mesh generation algorithms are designed to avoid creating slivers. For example, Delaunay refinement can be modified by choosing a new type of Steiner point which does not create any sliver [Li00b, LT01, LH01]. As an example, Chew's algorithm [Che97] inserts Steiner points in a randomized manner, to avoid the creation of slivers. This method has a theoretical lower bound of $\arcsin 1/4 \approx 14.5°$ on the angles of the triangular faces of the mesh. Labelle's algorithm [Lab06] can also be used as a complete mesh generation algorithm.

Weighted Delaunay-based

Sliver exudation First described by Cheng et al. [CDE+00], sliver exudation is a technique based on turning a Delaunay triangulation into a weighted Delaunay triangulation [BWY07], devised to trigger flips so as to increase the minimal angle. Edelsbrunner and Guoy [EG02] provide an experimental study of sliver exudation, and show that it works pretty well in practice as a post-treatment applied to a triangulation obtained by Delaunay refinement [RY07]. The main strategy of the algorithm consists of assigning a weight to each vertex so that the weighted Delaunay triangulation is free of any slivers after connectivity updates, without any changes over the vertex locations. This method successfully increases all dihedral angles above 5° in the best configuration (see Section 3.2.2.3), but as admitted in [EG02], the theoretical bound on the dihedral angle is too small to be of any practical significance.

Beside being not strictly Delaunay anymore, the main disadvantage of sliver exudation is that the process often ends with leftover slivers near the boundary [EG02]. This is mainly due to the fact that sliver exudation is not allowed to modify the topology of the boundary of the mesh. Hence, weight assignments close to the boundary are constrained and do not always manage to remove the slivers.

Sliver-free mesh generation. Cheng and Dey [CD02] propose a complete Delaunay refinement algorithm, combined with the sliver exudation technique. This type of weighted-Delaunay algorithm is also used to handle input domains containing sharp creases subtending small angles [CDR05].

Non-Delaunay

Local combinatorial operations. Though a Delaunay-refined triangulation is known to have nice properties on its angles in 2D [Ede87], there is no theoretical guarantee on the dihedral angles in 3D. One valid choice consists of leaving the Delaunay framework by flipping some well-chosen simplices [She02c, KS07], either as a post-processing step to the meshing process [KO01], or during the whole process [CDM04]. As long as the triangulation remains valid, flips can be performed on its edges and facets. Joe gives a description of all possible flips [Joe95] that can be made in a triangulation, and a triangulation improvement algorithm through these flips. Although each improvement in this algorithm is local, the complete algorithm succeeds in improving the overall quality of the mesh.

Dealing with non-Delaunay meshes can also be combined with optimization steps, such as Laplacian smoothing [FK94], which relocates each vertex to a new location computed as an average of the incident vertex positions. Laplacian smoothing can be applied to any valid triangulation.

Sliver-free mesh generation. Some other types of triangulations, such as for example max-min solid angle triangulations [Joe91] can be computed to improve the solid angles as compared to that in a Delaunay triangulation. This method generates a set of well-distributed points in the input polyhedral domain and first computes a Delaunay triangulation of these vertices. Then, local combinatorial transformations are applied to satisfy the local max-min angle criterion. These local transformations can in fact be applied to any triangulation as a post-processing step.

Instead of performing local improvements through flips in a Delaunay mesh, Labelle and Shewchuck [LS07] propose a fast lattice refinement technique which constructs a triangulation based on two nested regular or adapted grids. In its graded version this algorithm provides a theoretical bound on the dihedral angles which is much more practical than provided by other algorithms.

Contribution We present a sliver removal algorithm inspired by Li's random perturbation algorithm [Li00a]. Our algorithm is made more deterministic by choosing a favored perturbation direction for each vertex incident to one or more slivers, *before* resorting to Li's random perturbation if the favored perturbation fails at removing the incident slivers. Our experiments show that the chosen deterministic directions are sufficient to remove more than 80% of the slivers of a mesh, leading to shorter computational times. In addition, our approach is able to deal with uniform and graded meshes (see Figure 3.7), and reaches higher minimum dihedral angles than random perturbation in practice [TSA09].

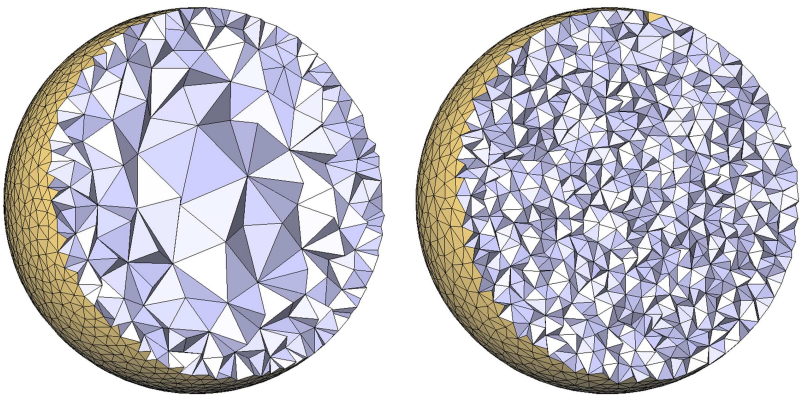


Figure 3.7: Sphere. (Left) Graded mesh with 3195 vertices, output angles are in $[23.5; 142.5]$. (Right) Uniform mesh with 7041 vertices, output angles are in $[30.02; 138.03]$.

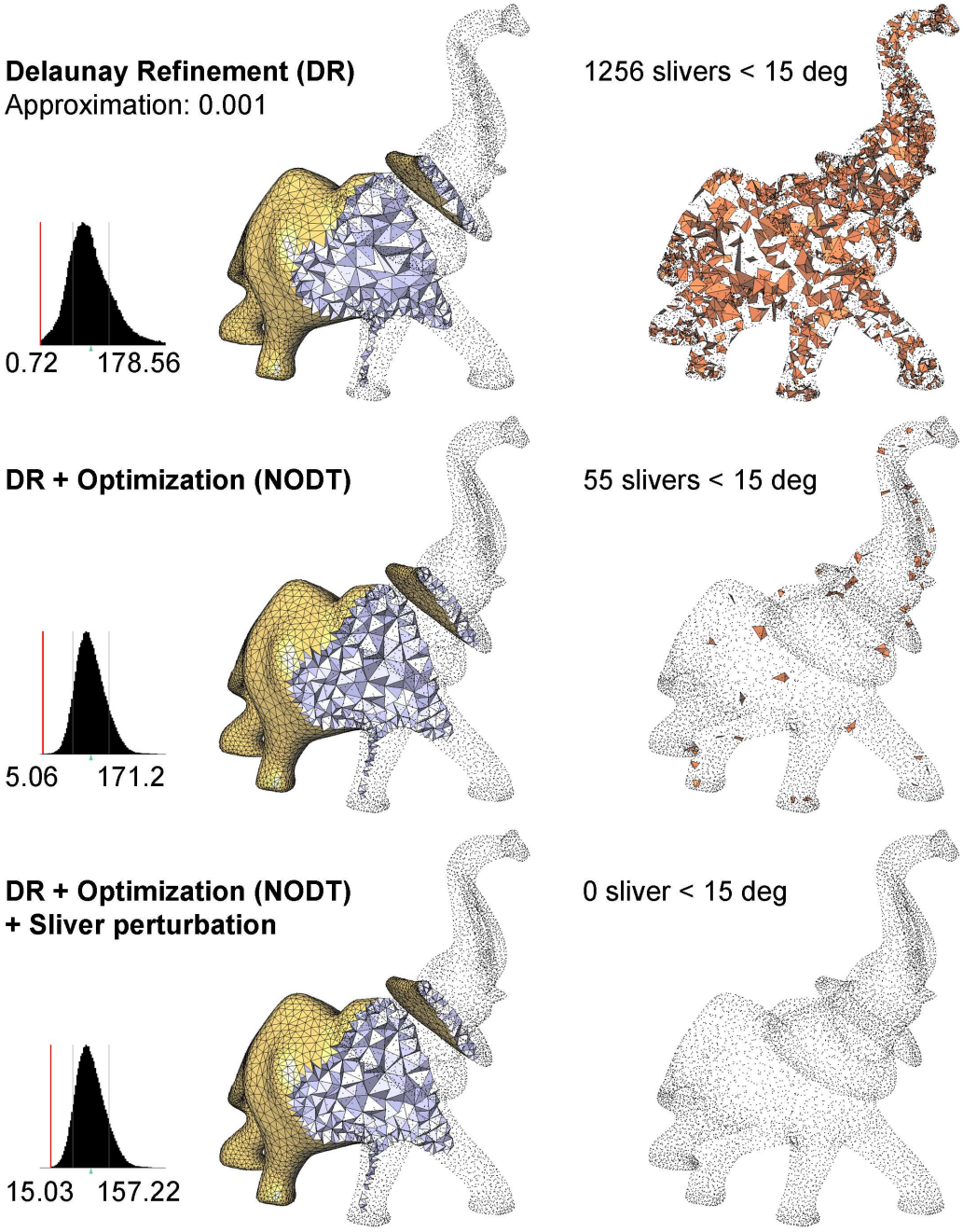


Figure 3.8: Elephant. Top: mesh generated by Delaunay refinement with $l_{max} = 0.1$, $\epsilon_{max} = 0.001$. Distribution of dihedral angles, and sliver tetrahedra with dihedral angles lower than 15 degrees are shown. Middle: output of Delaunay refinement (i.e., top row) optimized with our technique. Bottom: optimized mesh (middle row) after sliver perturbation.

3.2.2.2 Algorithm

We describe a sliver perturbation algorithm which improves in a hill-climbing manner the dihedral angles of an input isotropic Delaunay mesh. This algorithm can be used as a post-processing step after refinement or optimization.

To improve the dihedral angles of the mesh tetrahedra, the rationale behind our approach is as follows: each vertex v incident to at least one sliver is repeatedly relocated through a perturbation vector $\vec{p_v}$ such that when v moves to $v + \vec{p_v}$, the incident slivers get flipped. More specifically, the chosen direction for $\vec{p_v}$ is not devised to improve the shape of the slivers, but rather to worsen them instead, so that they get flipped. Two directions are favored by the algorithm: the incident squared circumradius gradient ascent (see Section 3.2.2.2) and the sliver volume gradient descent (see Section 3.2.2.2). The length of the perturbation vector is heuristically chosen as a fraction (usually between 0.05 and 0.2) of the minimum incident edge length. If neither of these two perturbation vectors succeed in flipping a sliver we resort to random perturbations (see Section 3.2.2.2). If the whole sequence does not improve the local minimum dihedral angle then we restore the vertex to its original location before perturbation.

Algorithm 7 Sliver perturbation

Input: *$\mathcal{T}$: a Delaunay triangulation,*
α: the angle bound defining slivers, and
N_{max}: the maximum number of random trials, or gradient steps.

Let $\mathcal{P}$ be a priority queue of Delaunay vertices.
Fill $\mathcal{P}$ with vertices incident to slivers,
Compute perturbation vector $\vec{p_v}$ for each vertex v in $\mathcal{P}$,

while $\mathcal{P}$ non-empty **do**
  Pop v from $\mathcal{P}$,
  $v' \leftarrow v$,
  while relocating v to v' would not trigger a combinatorial change,
  and $\#loops < N_{max}$ **do**
    $v' \leftarrow v' + \vec{p_v}$,
    if $\vec{p_v}$ is random, **then**
      compute a new $\vec{p_v}$,
      $v' \leftarrow v$.
  Conditionally relocate v to v'.
  if v is still incident to slivers and $\vec{p_v}$ is not random, **then**
    Compute a new perturbation vector (another type, if possible),
    and re-insert v into $\mathcal{P}$.
  Insert all vertices affected by relocation into $\mathcal{P}$,
  with their new perturbation vector.

When more than one sliver is incident to a vertex v, all perturbation vectors must be compatible (i.e., pushing in a similar direction) to be effective. In our current algorithm, a set of perturbation vectors are said to be compatible if all their pairwise dot products are positive. The perturbation vector $\vec{p_v}$ is then set to be the average of these vectors. When not compatible, v is perturbed only using random perturbations. The algorithm relies on a modifiable priority queue, built in a way such that vertices incident to fewer slivers are processed first. Hence, any "chain" of slivers (set of slivers sharing at least one vertex) is treated starting from its endpoints thereby minimizing the need to process vertices incident to more than one sliver.

Note that each vertex relocation is conditional, as we want our algorithm to be hill-climbing in terms of dihedral angles. We need to check that the minimum dihedral angle of the triangulation does not decrease, and that the topology of the boundary is not affected. Otherwise, the relocation is canceled.

Each time a vertex is effectively relocated, the priority queue is updated. Moving v to v' in a Delaunay mesh makes combinatorial changes (and, hence, changes on incident dihedral angles) on the vertices incident to v before its removal, and the ones incident to v' after its insertion. We first compute the perturbations associated with all these vertices, and insert them into the priority queue.

The order in which the vertices are processed in the priority queue is related to the vertex type. Interior vertices are processed first, since they are more likely to be easily perturbable than boundary vertices. The boundary vertices are constrained to be located on the boundary, and their move must not break the topology of the mesh. These constraints make them more difficult to perturb. The other ordering criteria are discussed in Section 3.2.2.3.

By construction, our combined perturbation algorithm is hill-climbing in the sense that the dihedral angles in the output mesh must be higher than the ones in the input mesh. Li's random perturbation [Li00a] has theoretical guarantees. In particular, it can be shown that, given an input mesh, a small perturbation can be found that improves the radius-edge ratios of simplices (Theorem 3.2.1). This combined method makes some deterministic perturbation trials, which are aborted when they fail to improve the dihedral angles in the mesh. At this point, the combined method has not damaged the quality of the mesh. Since we resort to random perturbation in case deterministic perturbation has failed, combined perturbation appropriates random perturbation guarantees [Li00a].

Circumsphere Radius In an almost-good isotropic tetrahedron mesh, the distribution of the mesh vertices is locally uniform. Hence, perturbing the vertex locations so as to make the radius of the sliver's circumsphere explode triggers many flips as the empty circumsphere property must hold after Delaunay connectivity update.

Let τ be the sliver, and $\{p_i\}_{i=0,1,2,3}$ its vertices. Without loss of generality, and since the sequel remains true by translation, we can assume that $p_0 = 0_{\mathbb{R}^3}$. We also assume that this vertex is fixed. Let c be τ's circumcenter. We have $||c|| = R$ the radius of τ's circumsphere. Then, $\nabla R^2 = \nabla ||c||^2$. We aim at computing ∇R^2.

Let $p_i = (x_i, y_i, z_i)$ for i in $\{0,1,2,3\}$ be τ's vertices, with $p_0 = 0_{\mathbb{R}^3}$. Also, let p_i^2 be $(x_i^2 + y_i^2 + z_i^2)$. The center c of the circumsphere of τ is given by $c = \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} = \begin{pmatrix} \frac{D_x}{2a} \\ \frac{D_y}{2a} \\ \frac{D_z}{2a} \end{pmatrix}$, where $a = \begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}$,

$$D_x = -\begin{vmatrix} p_1^2 & y_1 & z_1 \\ p_2^2 & y_2 & z_2 \\ p_3^2 & y_3 & z_3 \end{vmatrix},\ D_y = +\begin{vmatrix} p_1^2 & x_1 & z_1 \\ p_2^2 & x_2 & z_2 \\ p_3^2 & x_3 & z_3 \end{vmatrix}, \text{ and } D_z = -\begin{vmatrix} p_1^2 & x_1 & y_1 \\ p_2^2 & x_2 & y_2 \\ p_3^2 & x_3 & y_3 \end{vmatrix}.$$

Thus we have, $\nabla_{p_1} ||c||^2 = \begin{pmatrix} \frac{\partial ||c||^2}{\partial x_1} \\ \frac{\partial ||c||^2}{\partial y_1} \\ \frac{\partial ||c||^2}{\partial z_1} \end{pmatrix}$ with

$$\begin{aligned}
\frac{\partial ||c||^2}{\partial x_1} &= \frac{\partial}{\partial x_1}\left(\frac{D_x^2 + D_y^2 + D_z^2}{4a^2}\right) \\
&= \frac{1}{2a^3} \cdot \left(a \cdot (D_x \cdot \frac{\partial D_x}{\partial x_1} + D_y \cdot \frac{\partial D_y}{\partial x_1} + D_z \cdot \frac{\partial D_z}{\partial x_1}) - \frac{\partial a}{\partial x_1} \cdot (D_x^2 + D_y^2 + D_z^2) \right), \\
\frac{\partial ||c||^2}{\partial y_1} &= \frac{1}{2a^3} \cdot \left(a \cdot (D_x \cdot \frac{\partial D_x}{\partial y_1} + D_y \cdot \frac{\partial D_y}{\partial y_1} + D_z \cdot \frac{\partial D_z}{\partial y_1}) - \frac{\partial a}{\partial y_1} \cdot (D_x^2 + D_y^2 + D_z^2) \right), \\
\frac{\partial ||c||^2}{\partial z_1} &= \frac{1}{2a^3} \cdot \left(a \cdot (D_x \cdot \frac{\partial D_x}{\partial z_1} + D_y \cdot \frac{\partial D_y}{\partial z_1} + D_z \cdot \frac{\partial D_z}{\partial z_1}) - \frac{\partial a}{\partial z_1} \cdot (D_x^2 + D_y^2 + D_z^2) \right).
\end{aligned}$$

and

$$\nabla_{p_1} a = \begin{pmatrix} \frac{\partial a}{\partial x_1} \\ \frac{\partial a}{\partial y_1} \\ \frac{\partial a}{\partial z_1} \end{pmatrix} = \begin{pmatrix} y_2 z_3 - y_3 z_2 \\ -(x_2 z_3 - x_3 z_2) \\ x_2 y_3 - x_3 y_2 \end{pmatrix},$$

$$\nabla_{p_1} D_x = \begin{pmatrix} \frac{\partial D_x}{\partial x_1} \\ \frac{\partial D_x}{\partial y_1} \\ \frac{\partial D_x}{\partial z_1} \end{pmatrix} = \begin{pmatrix} -2x_1 \frac{\partial a}{\partial x_1} \\ -2y_1 \frac{\partial a}{\partial x_1} + p_2^2 z_3 - p_3^2 z_2 \\ -2z_1 \frac{\partial a}{\partial x_1} - p_2^2 y_3 + p_3^2 y_2 \end{pmatrix},$$

$$\nabla_{p_1} D_y = \begin{pmatrix} \frac{\partial D_y}{\partial x_1} \\ \frac{\partial D_y}{\partial y_1} \\ \frac{\partial D_y}{\partial z_1} \end{pmatrix} = \begin{pmatrix} -2x_1 \frac{\partial a}{\partial y_1} - p_2^2 z_3 + p_3^2 z_2 \\ -2y_1 \frac{\partial a}{\partial y_1} \\ -2z_1 \frac{\partial a}{\partial y_1} + p_2^2 x_3 - p_3^2 x_2 \end{pmatrix},$$

$$\nabla_{p_1} D_z = \begin{pmatrix} \frac{\partial D_z}{\partial x_1} \\ \frac{\partial D_z}{\partial y_1} \\ \frac{\partial D_z}{\partial z_1} \end{pmatrix} = \begin{pmatrix} -2x_1 \frac{\partial a}{\partial z_1} + p_2^2 y_3 - p_3^2 y_2 \\ -2y_1 \frac{\partial a}{\partial z_1} - p_2^2 x_3 + p_3^2 x_2 \\ -2z_1 \frac{\partial a}{\partial z_1} \end{pmatrix}.$$

Following a gradient ascent scheme, the vertex position p_i evolves this way:

$$p_i^{next} \;=\; p_i \;+\; \epsilon \frac{\nabla p_i R_\tau^2}{||\nabla p_i R_\tau^2||},$$

where the step length ϵ is taken as a fraction of the minimum incident edge length to p_i. A relocation is performed only if the new minimal dihedral angle in the tetrahedra impacted by the relocation is not smaller than it was before relocation. As shown by Figure 3.9, the squared radius of τ's circumsphere increases very fast for a small perturbation of one of its vertices' positions. The circumsphere, now huge, most probably includes other mesh vertices, which triggers a flip to maintain the empty sphere Delaunay property.

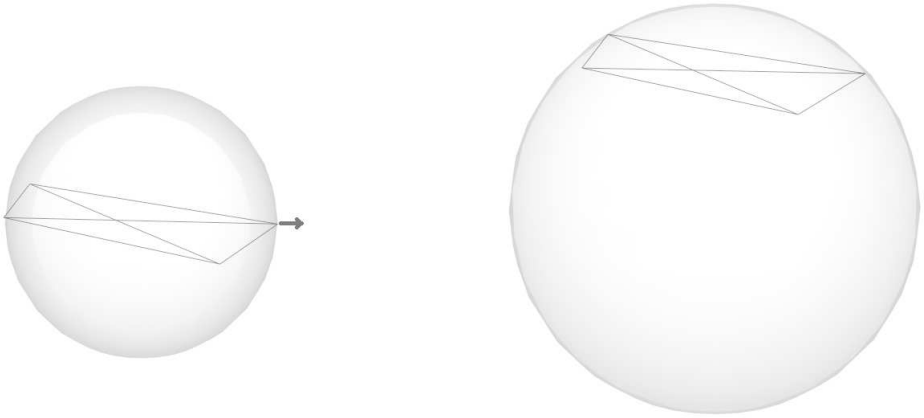


Figure 3.9: Circumsphere of a sliver. Before perturbation (left), the sliver is close to the equatorial plane of its circumsphere. A very mild perturbation of one of the sliver vertices (right) makes its circumradius increase considerably.

Volume One of the main characteristics of a sliver is that its volume is strictly positive albeit small with respect to its smallest edge length, and possibly arbitrarily small. This property can be exploited in order to apply a perturbation devised to generate a sliver with negative volume and hence to trigger a combinatorial change.

Let $\{p_i\}_{i=1,2,3}$ be the three fixed points of τ, and p_0 the vertex to be perturbed.

The volume of τ is

$$V_\tau = \frac{1}{6}\begin{vmatrix} x_0 & y_0 & z_0 & 1 \\ x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \end{vmatrix}.$$

Then, we get the volume gradient:

$$\nabla p_0 V_\tau = \frac{1}{6}\begin{pmatrix} y_2 z_3 + y_1(z_2 - z_3) - y_3 z_2 - z_1(y_2 - y_3) \\ -x_2 z_3 - x_1(z_2 - z_3) + x_3 z_2 + z_1(x_2 - x_3) \\ x_2 y_3 + x_1(y_2 - y_3) - x_3 y_2 - y_1(x_2 - x_3) \end{pmatrix}.$$

Following a gradient descent scheme, the vertex position p_i evolves this way:

$$p_i^{next} \;=\; p_i \;-\; \epsilon \frac{\nabla p_i V_\tau}{||\nabla p_i V_\tau||},$$

where the step length ϵ is taken as a fraction of the minimum incident edge length to p_i. A relocation is performed only if the new minimal dihedral angle of the tetrahedra impacted by the relocation is not smaller than it was before relocation. A negative tetrahedron volume triggers a flip to maintain a valid Delaunay triangulation.

Random perturbation When both ∇V and ∇R^2 fail at flipping the considered slivers by vertex perturbation, we use a random perturbation based on Li's approach [Li00a]. A perturbation satisfying three conditions (flip sliver, improve minimum dihedral angles, preserve restricted Delaunay triangulation) is searched for randomly inside a sphere centered at v. In accordance with Li's algorithm, the magnitude of the perturbation vector is set to fraction of the minimum incident edge length.

3.2.2.3 Experiments and Results

The algorithm presented has been implemented with the 3D Delaunay triangulation of the *Computational Geometry Algorithms Library* [cga]. Our implementation of Li's random perturbation algorithm is based upon Algorithm 7, with one single perturbation type: the random one, described in Section 3.2.2.2. For each of the following experiments we set 100 trials of random perturbations (in our combined version as well as in the purely random algorithm).

The order in which the vertices are processed in the priority queue has been chosen empirically as a result of many experiments. Interior vertices are processed first, with priority over boundary vertices. Boundary vertices are constrained so as to remain on the domain boundary and their relocation is invalid if they modify the local restricted triangulation. This makes boundary vertices more difficult to perturb than interior vertices. The second order criterion is the number of incident slivers to the processed vertex. The idea behind this choice is that a *chain* of slivers (several incident slivers) is more difficult to perturb than an isolated sliver as the directions of gradients may not be compatible. However, if the endpoints of the chain are successfully perturbed, we ideally would not have to deal with vertices incident to more than one sliver. Thirdly, the vertex incident to a smaller dihedral angle is processed first, as our first goal is to remove the worst tetrahedra.

In our experiments, the ∇R^2 direction turns out to be more effective than ∇V at perturbing a sliver. On average, this perturbation is responsible for about 80% of all sliver flips. The ∇V perturbation accounts for about 15% of the flips while the random perturbation counts for the remaining 5%. The priority given to ∇R^2 over ∇V and random while picking the perturbation vector can be blamed for distorting these statistics, but we have chosen this order because it turns out to be the most effective. Giving priority to ∇V results in an overall slowdown. Random perturbation always remains the last resort in the combined perturbation algorithm as the deterministic directions are favored.

The following experiments show what our combined algorithm can achieve on meshes generated by Delaunay refinement alone and on some meshes which have been optimized after refinement. A mesh optimization algorithm is in general devised to improve the mesh quality [ABE97] while simpler algorithms aim at evenly distributing the vertices in accordance to a given mesh sizing function. Note that a mesh with well-spaced vertices does not mean an absence of slivers inside the mesh [Tal97], and hence sliver removal is still required. The mesh optimization schemes used in our experiments are the centroidal Voronoi tessellation [DFG99] using the Lloyd iteration, and the Optimal Delaunay triangulation (ODT for short) [CX04a]. Both of these optimization methods have been implemented in a

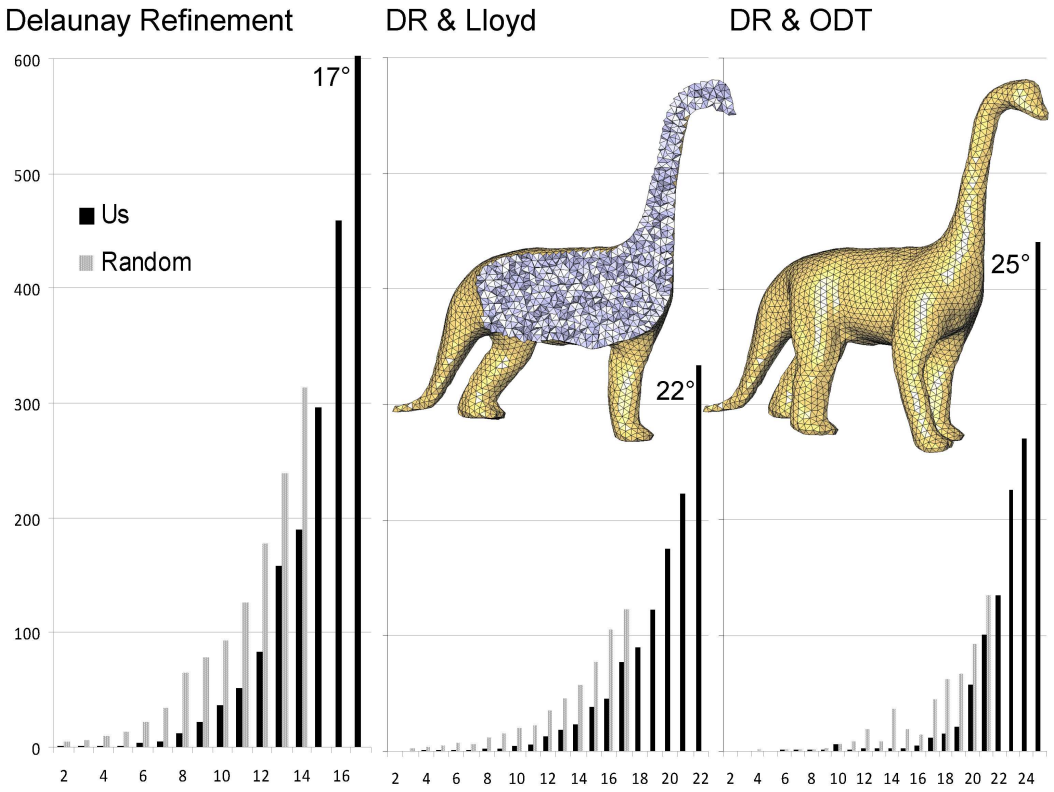


Figure 3.10: Dinosaur. Comparison of the timings for our perturbation and random perturbation (in seconds) w.r.t. the sliver angle bound α on the Dinosaur model meshes obtained by Delaunay refinement (left), followed by Lloyd optimization (middle) and ODT optimization (right).

way that respects the local density of the mesh. It is important to not modify the density of a graded mesh, and to not decrease its quality.

Figures 3.10 and 3.11 provide the computation times and the best minimum dihedral angles obtained in our experiments. The same experiment has been carried out on many other models (not shown), giving similar results. Figures 3.10 and 3.11 emphasize that, for the same definition of a sliver (in terms of smallest dihedral angle), the combined algorithm is faster in removing all slivers by explicit perturbation compared to using Li's random perturbation alone. Moreover the combined algorithm reaches higher minimum dihedral angles.

The algorithm obtains fairly high minimum dihedral angles when the input is a mesh obtained by Delaunay refinement. Figures 3.10 and 3.11 illustrate that when the mesh is optimized prior to perturbation, the time taken for the algorithm to succeed in removing all slivers is shorter and that it can reach a higher minimum

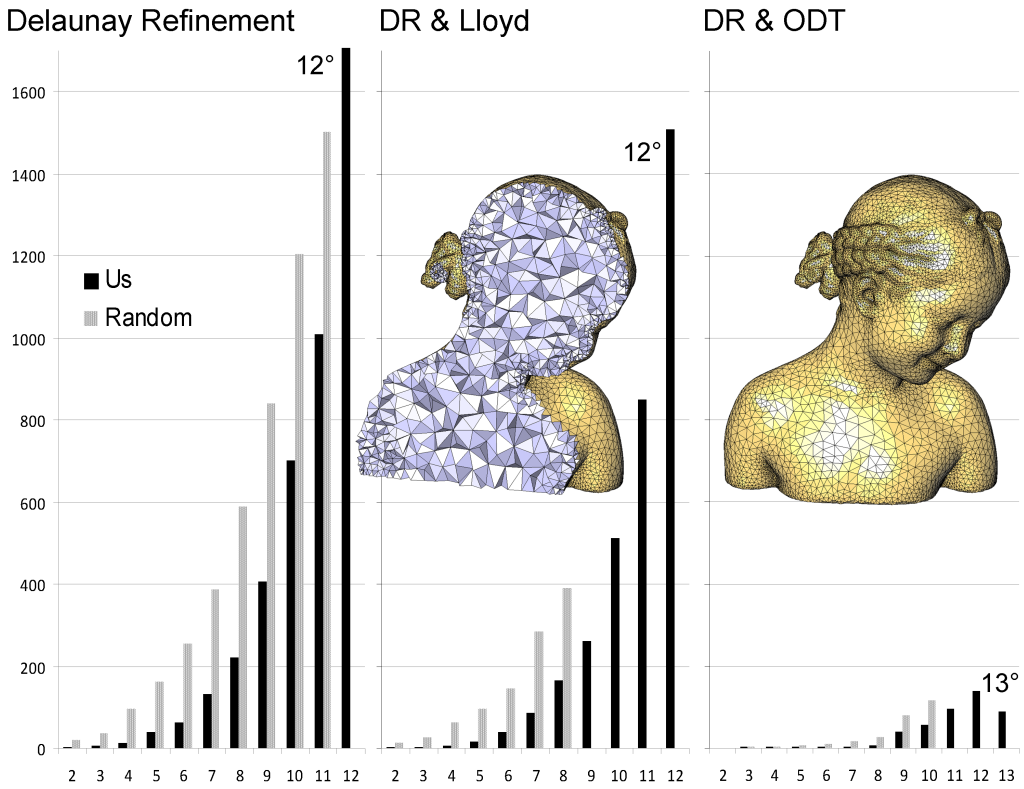


Figure 3.11: Bimba. Comparison of the timings for our perturbation and random perturbation (in seconds) w.r.t. the sliver angle bound α on the Bimba model meshes obtained by Delaunay refinement (left), followed by Lloyd optimization (middle) and ODT optimization (right).

dihedral angle. As shown by histograms of Figure 3.10, the algorithm takes 611 seconds to perturb the mesh obtained after Delaunay refinement so that no dihedral angle is below 17°. If the same mesh is optimized prior to perturbations, the time taken goes down to 76 seconds for Lloyd and even further down to 11 seconds for ODT. Overall the same histograms show that a mesh optimized by ODT is easier to perturb and can reach a higher minimum angle (25°) than a mesh optimized by Lloyd (21°). However, optimization can be costly. The optimizations performed on Figure 3.10 meshes before applying perturbation took about 200 seconds. In spite of this additional cost, the combined perturbation algorithm remains more efficient than the random one. The same comments apply to Figure 3.11. The gradation of the mesh in Figure 3.11, along with the numerous high curvature regions, make it more difficult to perturb in a way that still preserves the gradation, even after optimization. Even in this case, ODT reaches a higher minimum angle.

For comparison we have also performed sliver exudation on meshes generated by Delaunay refinement and on meshes optimized after refinement. As expected sliver

exudation performs better on the optimized meshes.

We performed two other experiments that were abandoned since they rarely succeeded in improving the mesh quality. While computing the perturbation of a vertex incident to more than one sliver, we tried combining ∇V vector of one of the slivers and ∇R^2 of the other by using their average as perturbation direction if they were compatible. In practice such a combination was almost never successful removing the slivers. The other aborted experiment consisted of removing from the mesh the vertices that every explicit perturbation failed to perturb. In practice this never resulted in improving the minimum dihedral angle.

Moreover, our experiments show that successively applying our combined algorithm to the mesh several times while progressively increasing the angle bound that defines a sliver provides higher minimal dihedral angles at the price of higher computation times. This amounts to giving priority to vertices incident to the worst slivers, cluster by cluster of minimum dihedral angles.

Table 3.1 summarizes the best angles obtained in this way using combined perturbation, random perturbation and sliver exudation. In this labor-intensive experiment we only measure how far we can go in terms of dihedral angles and do not consider timing. Finally, Figure 3.12 shows some Delaunay meshes obtained by Delaunay refinement followed by ODT optimization and perturbed with the combined algorithm along with their dihedral angle histograms.

Mesh	input	us	random	exudation
Dinosaur (DR)	0.65	25.0	24.2	1.68
Dinosaur (DR & Lloyd)	0.24	26.15	23.5	4.47
Dinosaur (DR & ODT)	2.26	28.55	22.0	4.55
Bimba (DR)	0.16	15.51	15.64	1.11
Bimba (DR & Lloyd)	0.11	16.02	15.63	3.84
Bimba (DR & ODT)	0.84	19.8	18.85	4.47

Table 3.1: Angles. Minimum dihedral angles obtained by the different perturbation algorithms (combined perturbation, random perturbation, and sliver exudation). To achieve these maxima, combined perturbation takes about twice the exudation time, and random perturbation takes about six times the exudation time.

3.2.2.4 Summary

We have presented a practical vertex perturbation algorithm for improving the dihedral angles of a 3D isotropic Delaunay triangulation. The key idea consists of performing a gradient ascent over the sliver circumsphere radius as well as a gradient descent over the sliver volume. All vertices incident to slivers are processed, in an order devised to improve effectiveness and computation times. We compare our approach with pure random perturbation and sliver exudation.

Our experiments show that we are both faster and able to reach higher minimum dihedral angles. Our scheme is particularly well suited as a post-processing step after mesh optimization [TWAD09]. We also plan to use it in the context of mesh generation from multi-material voxel images [BYB09].

In the cases where all vertices of a sliver are on the domain boundary, the perturbation can fail in removing a sliver as the boundary vertices are too constrained. One way to extend our approach would be to also perturb the vertices of the sliver's adjacent tetrahedra whose relocation can impact the sliver. Future work will focus on obtaining a proof of termination of our combined perturbation algorithm, and some tighter lower bounds on output dihedral angles.

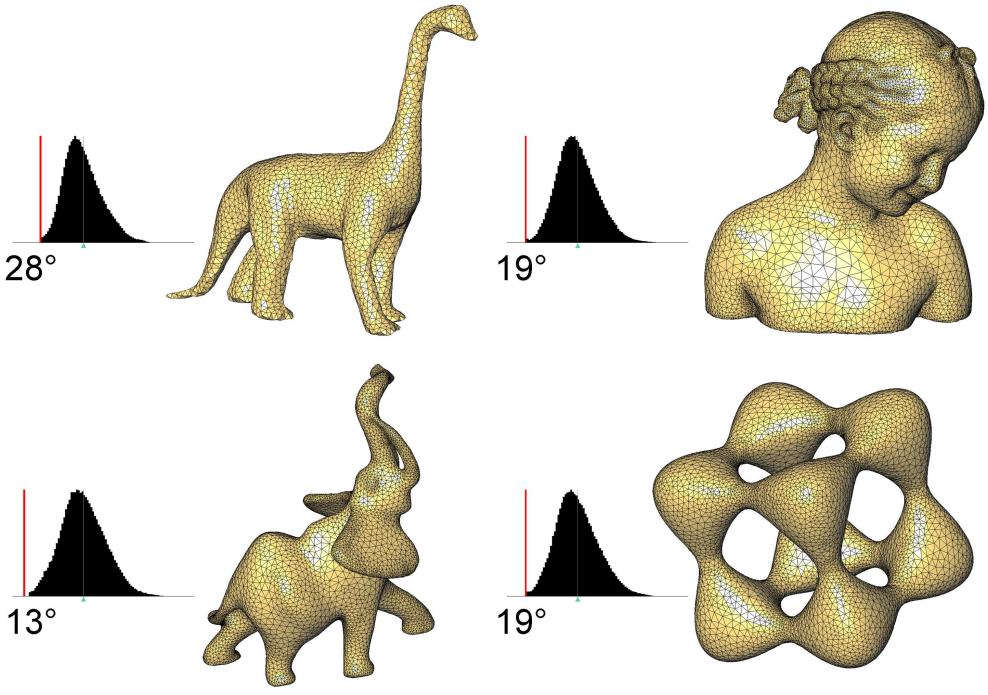


Figure 3.12: Delaunay meshes perturbed with combined perturbation algorithm after ODT optimization.

3.3 Implementation

Our algorithm is implemented in C++ using the CGAL library [cga]. We use its 3D Delaunay triangulation as our core data structure. The input PSC is represented as a surface triangle mesh, itself being represented by a CGAL polyhedral surface.

3.3.1 Intersection and Projection

One crucial component for reaching good timings is the efficient update of the restricted triangulation and computation of Steiner points. This requires many intersection tests between rays and Voronoi edges and the input domain boundary, as well as intersections between Voronoi faces and the input sharp creases. We have implemented a collision detection library based on the principles used in OPCODE [Ter05]. Two hierarchies of axis-aligned-bounding-boxes (AABBs) are created right after loading the input PSC: one for the PSC triangle facets and one for the PSC segment sharp creases. Figure 3.13 shows two input surface triangle meshes and the associated triangle facets AABB trees constructed.

Each intersection query (be it a test or an exhaustive enumeration) then calls intersection with AABBs during traversal, and intersection with PSC primitives (triangle or segments) at the leaves of the tree. In addition, the same AABB trees are used for projecting the optimized boundary vertices onto the domain boundary or creases. The trees are this time queried with 3D balls whose radius decreases during the tree traversal. A projection (or distance) query between a point p and the input primitives is turned into a ball (centered at p) query. Similarly to intersection queries, the ball traverses the AABB tree and recursively queries intersection tests with the AABBs. When the traversal ends, at the leaves of the tree, the closest point p' from p on the input primitives is computed.

Construction The AABB tree construction starts by computing the AABB of the complete set of input primitives. Then, all primitives are sorted along the longest coordinate axis of this AABB. The primitives are separated into two subsets of equal size. This splitting procedure is applied recursively until each bounding box contains a single primitive.

Functionalities The AABB tree data structure is built to provide a number of intersection detection, intersection computation and distance computation functionalities. An intersection query can be made to get the intersection objects (*e.g.* intersection points for ray queries). A distance query can be made to compute a distance or to get, for example, the closest point or primitive from the query point.

An AABB tree can be built on several types of input data, including a set of triangles and a set of segments.

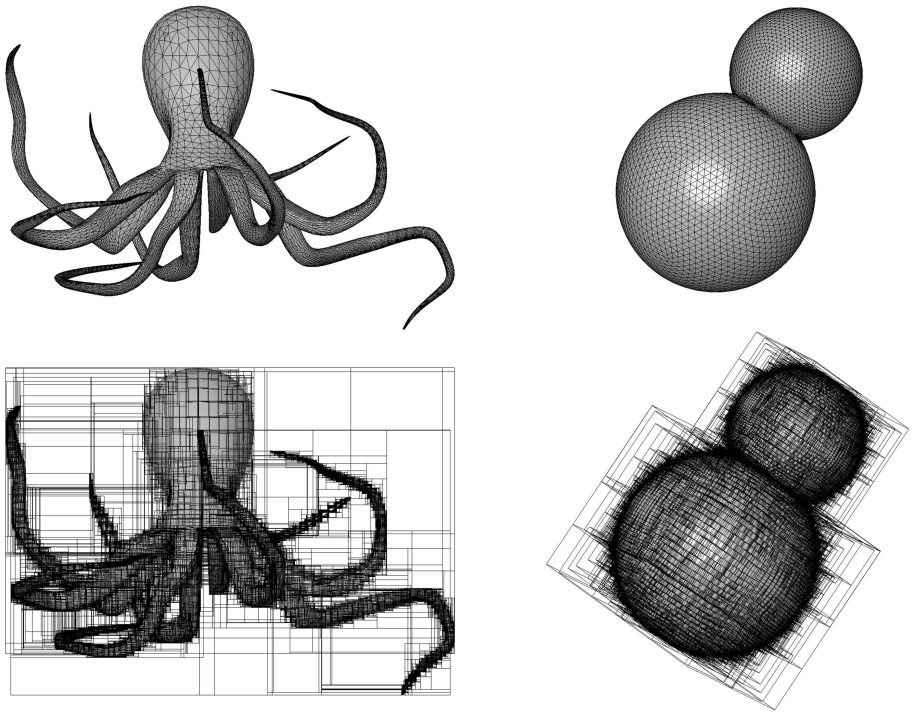


Figure 3.13: AABB tree. Top: input surface triangle meshes. Bottom: AABB trees constructed from triangle facets.

Intersection queries Intersection queries are very important for Delaunay refinement. Our first AABB tree is constructed from a set of triangles: the input PSC triangle facets. This is useful for extracting the restricted Delaunay tetrahedra from the mesh. A tetrahedron is said to be *restricted* if its circumcenter is on the bounded side of the input surface triangle mesh, which should be manifold and watertight. Testing if a tetrahedron τ is restricted or not can thus be done by counting the number of intersections between the input boundary triangles and an arbitrary ray whose source point coincides with τ's circumcenter. The tetrahedron is *restricted* if and only if this number is odd. Similarly, a Delaunay facet is said to be *restricted* if its dual Voronoi edge intersects the input boundary. The AABB tree of triangles is used for computing the intersection between the input surface triangle mesh and Voronoi segments.

The second AABB tree employed is constructed from a set of segments: the input PSC segment sharp creases. This is useful for extracting the restricted Delaunay edges from the mesh. A Delaunay edge is said to be *restricted* if its dual Voronoi facet intersects the input surface sharp edges.

Figure 3.14 shows an input PSC and the restricted edges, facets and tetrahedra of a mesh with respect to this PSC.

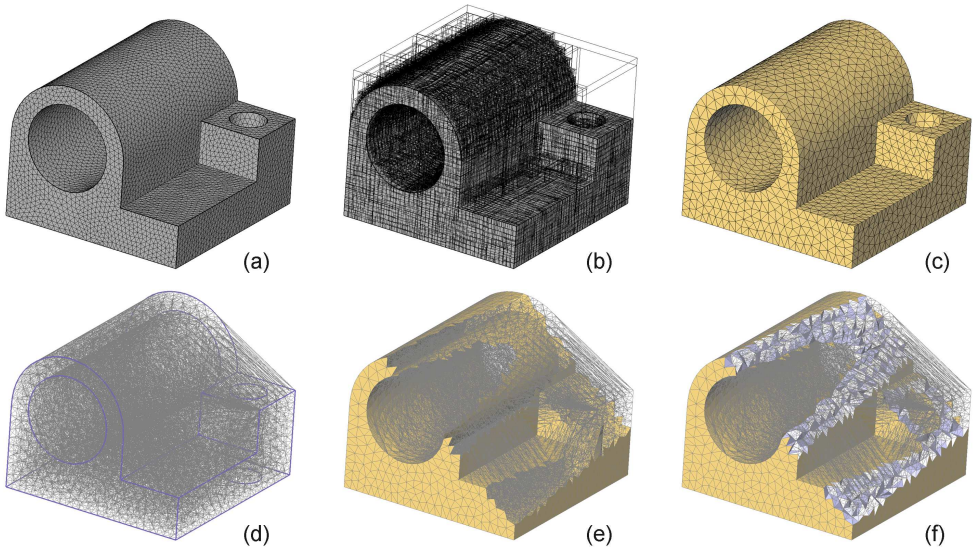


Figure 3.14: AABB tree. (a) Input PSC. (b) AABB tree computed from the PSC's triangle facets. (c) A mesh M computed by interleaving refinement and optimization (inside tetrahedra). (d) Restricted edges in M are plotted in blue. (e) Restricted facets in M. (f) Restricted tetrahedra in M.

Besides the detection of restricted simplices, intersection queries are needed to compute Steiner points of bad simplices. The Steiner point of a bad (restricted) simplex is computed as the intersection point between its dual Voronoi object and its corresponding input boundary simplex (see Section 3.2.1.4 for details on the choice of Steiner vertices).

An intersection query starts a tree traversal. During traversal, intersection tests only with respect to the AABBs are computed. Intersection constructions, with respect to the input primitives, are computed at the end of the traversal, in the leaves of the tree.

Projection queries In our algorithm, projection queries are mostly used in mesh optimization. We constrain every vertex of on the boundary of the mesh, *i.e.* incident to inside and outside tetrahedra, to lie exactly on the input PSC. Since the Natural ODT (see Section 3.2.1.5) formula does not guarantee that the vertex lies exactly on the input PSC, each computed optimal vertex $\mathbf{x}_i^\star$ is projected onto it. Computing the projection is done by finding the closest point on the input PSC. For crease edge vertices, the closest point is searched on input crease edges, using the segment AABB tree. For other boundary vertices, the closest point is searched on input PSC triangles.

This data structure for intersection and distance computation has been implemented as a CGAL package [ATW09].

3.3.2 Filtering relocations

One of the costly parts of our algorithm lies in the fact that, at each optimization step, all the vertices of the mesh are relocated. This implies that, each time, the Delaunay connectivity of the whole mesh has to be re-computed. To handle these relocations, two main approaches can be considered. First, *relocation* consists in moving vertices one by one in the triangulation. Second, *rebuilding* consists in recomputing the complete connectivity of the Delaunay triangulation from scratch. We compare these approaches and study [MTAD09] some filtering techniques for relocation speedup.

Surprisingly, rebuilding the triangulation from scratch often is a good option compared to relocating the vertices. However, when all the vertices move with a small displacement vector, or when only some vertices move, relocation can be faster than rebuilding. We focus on each vertex relocation.

The naive algorithm for relocating a single vertex v to position p is to remove v, and to insert a new vertex at p in the triangulation. This is costly and can be improved. The main idea is that, if the displacement is small enough, the connectivity around v will not change. In this case, removal and insertion should not be necessary. Modifying v's coordinates should be sufficient. For each vertex, a *safety region* in which it can move with no need of updating the local connectivity is computed. The *filtering algorithm* [MTAD09] is able to correctly decide whether or not a vertex relocation requires a connectivity update.

Some experiments are carried out in 2D and in 3D, for clustering and mesh optimization. In 2D, these experiments show that the filtering algorithm can be as much as six times faster than rebuilding. In 3D, it becomes more and more efficient when the optimization schemes get closer to convergence. As the relocation vectors become smaller and smaller, connectivity updates are less frequent. Close to convergence, the filtering algorithm can get three times faster as rebuilding, and thirty times faster than the naive remove-insert relocation algorithm.

3.3.3 Locking

We also significantly speed up the NODT procedure through a locking process. We lock up (*i.e.*, deactivate the optimization of) all mesh vertices which are incident to only excellent restricted tetrahedra. A tetrahedron is defined as excellent when all its dihedral angles are within a user-specified interval (typically $[45 - 95]$). Only the vertices newly inserted during refinement or relocated during optimization are allowed to unlock their incident vertices. Consequently, entire parts of the mesh which do not need to be improved either by refinement or by optimization are skipped throughout the refinement/optimization alternation. Tuning the interval bounds which qualify an excellent tetrahedron is our way to trade efficiency for the final mesh quality. Finally, each time the restricted Delaunay triangulation is

updated, the circumcenters are cached to avoid recomputing them at each refinement and optimization step.

3.4 Results

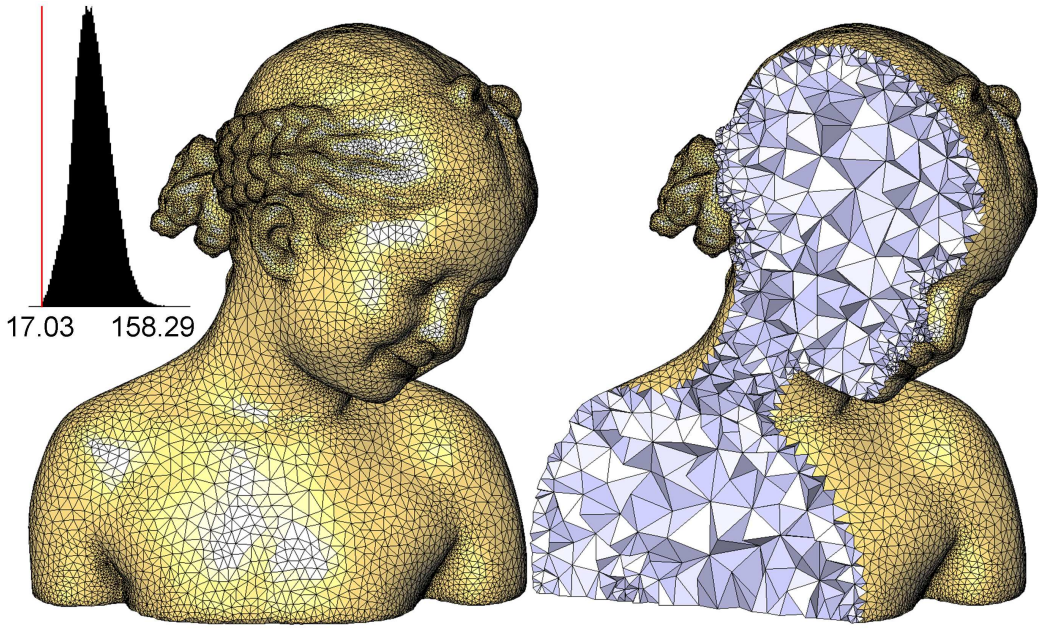


Figure 3.15: Bimba. Mesh generated by interleaved refinement and optimization with $l_{max} = 0.1$, $\epsilon_{max} = 0.0005$.

To evaluate our approach, we test the various steps of our algorithm separately, then together. Figure 3.5 shows our refinement routine when no optimization step is performed. Notice that the resulting mesh lacks gradation, as typical for Delaunay refinement methods. We compare results of Delaunay refinement, Lloyd relaxation [DFG99], and our NODT in terms of number of slivers (*before* sliver removal for fairness) in Figure 3.6. Figure 3.8 shows an elephant mesh obtained by Delaunay refinement (top) as it gets optimized by our NODT routine (middle), then after sliver removal (bottom).

Figure 3.15 shows the mesh of the bimba model obtained by interleaved refinement and optimization with approximation and element quality criteria activated (the sizing criterion $l_{max} = 0.1$ is not significant as the input PSC fits into a unit bounding box). The mesh contains 43K vertices and all dihedral angles are above 17 degrees. The input PSC has 400K vertices. We also test our method on mechanical parts. Figure 3.16 shows the mesh of a turbine generated by our interleaved algorithm. The mesh contains significantly fewer vertices (13%) than Delaunay refinement alone.

We also compare our technique to DelPSC [CDL07], TetGen [Si07] and GHS

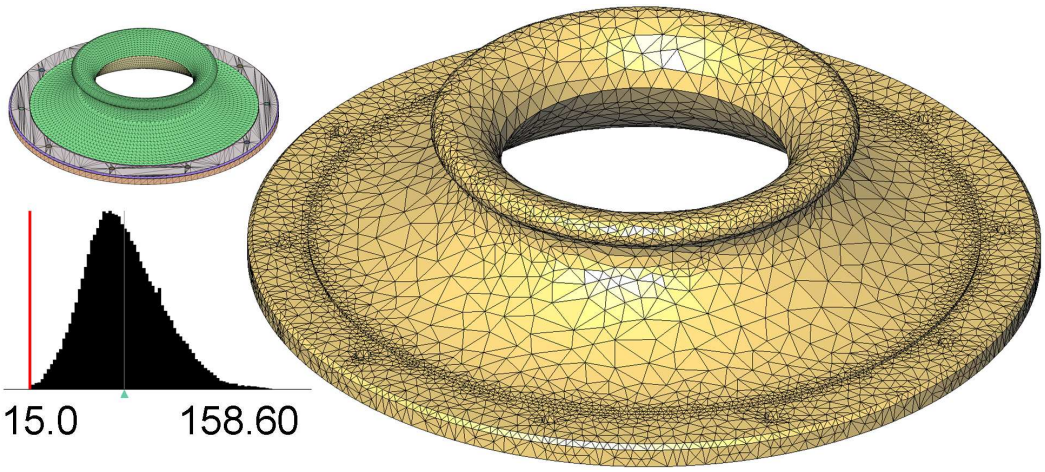


Figure 3.16: Turbine. Mesh generated by interleaved refinement and optimization with $l_{max} = 0.1$, $\epsilon_{max} = 0.001$. The inset shows the input PSC with all patches segmented. The mesh has 14K vertices and 51K tetrahedra, with all dihedral angles greater than 15 degrees.

3D [Geo04] in Figure 3.17 in terms of both mesh quality and vertex count. Note that for TetGen and GHS 3D we provided as input the boundary of our optimized mesh for fair comparison. Figure 3.19 shows the mesh of the Buddha model obtained by interleaved refinement and optimization, showcasing our method on a domain with a large range of local feature sizes.

Activating the topology criterion enforces that each restricted facet has its three vertices on the same PSC patch, and that each restricted edge has its two vertices on the same PSC crease. The mesh can thus be refined beyond the specified approximation criterion until all surface sheets are separated, as illustrated by Figure 3.18.

In our experience (Table 3.2), computational times to obtain a mesh range from seconds for the sphere and nested-spheres models, to minutes for the anchor, turbine and bimba models (resp. 10, 15 and 23), to two hours for Michelangelo's David model.

Model	ϵ_{max}	Nb of tetrahedra	Time
Coverrear	0.001	22,091	3 min
Bimba(1)	0.001	52,866	10 min
Bimba(2)	0.0005	121,644	41 min
Buddha	0.001	161,378	20 min

Table 3.2: Timings. The four meshes mentioned in this table were generated with $l_{max} = 0.1$, $\sigma^f_{\max} = 2$, and $\sigma^t_{\max} = 2$.

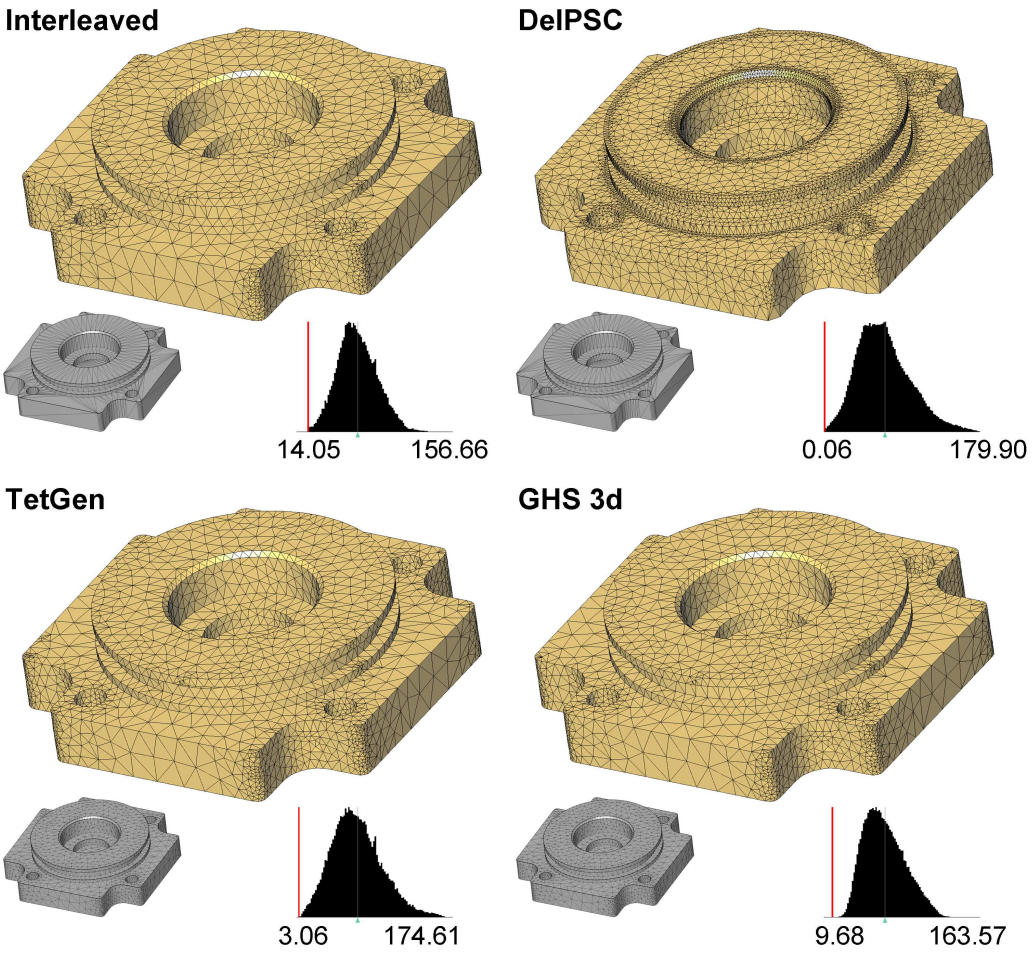


Figure 3.17: Cover rear. Top left: mesh obtained by interleaved refinement and optimization with $l_{max} = 0.1$, $\epsilon_{max} = 0.001$, $\sigma^{f}_{\max} = 1.5$, $\sigma^{t}_{\max} = 1.5$, with topology criterion activated; 4,050 vertices. Top right: mesh generated by DelPSC, same parameters; 15,157 vertices. Bottom: meshes generated by TetGen (left, 4,189 vertices) and GHS 3D (right, 6,641 vertices), same parameters and with the boundary of our optimized mesh taken as input.

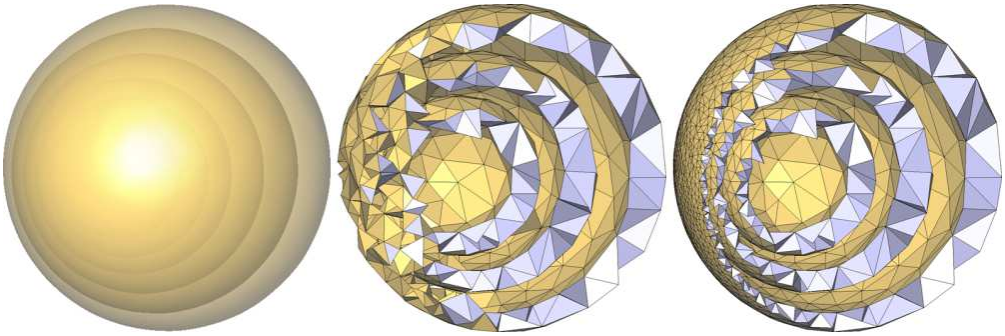


Figure 3.18: Nested spheres. Left: input PSC. Middle: mesh generated by refinement with $l_{max} = 1$, $\epsilon_{max} = 0.03$ and topology criterion not activated. Right: mesh further refined with same criteria but with topology activated.

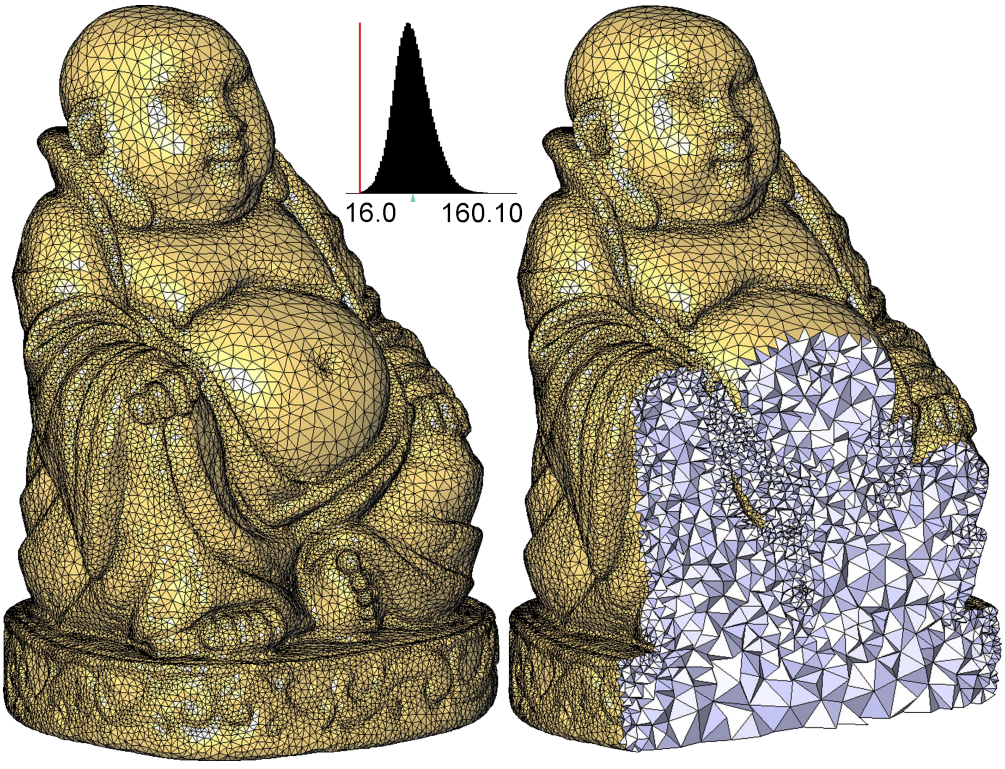


Figure 3.19: Buddha. Isotropic graded mesh obtained by interleaving refinement and optimization. All resulting dihedral angles are greater than 16 degrees.

3.5 Summary

The algorithm presented in this chapter introduces a new mesh generation framework, based on the idea of interleaving steps of Delaunay refinement and optimization. Guided by user-defined criteria such as size, shape, and approximation error of mesh elements, refinement steps are parsimoniously applied batch-wise through the insertion of independent sets of Steiner vertices. Optimization steps are performed through a variant of Chen's ODT that handles boundary as well as spatially-varying mesh sizing. A post-processing step extending Li's random vertex perturbation enables the algorithm to remove the possibly remaining slivers after interleaving.

The meshes generated by this algorithm - interleaving and perturbation - are experimentally shown (see Section 3.4) to have a smaller number of Steiner vertices than state-of-the-art methods it has been compared too. However, the main improvement provided by this algorithm stands in the quality of dihedral angles.

The main limitation of our algorithm is that it does *not* handle sharp input creases subtending small angles (the theoretical bound on input angles is 90°, see [RY07]).

Chapter 4

Conclusion & Future work

Contents

4.1 Conclusion

This thesis describes a new approach to practical isotropic Delaunay mesh generation. In two and three dimensions, the idea of interleaving Delaunay refinement steps and mesh optimization steps is put to work with satisfactory results. The whole mesh generation process is driven by user-defined criteria that the output mesh needs to satisfy (*e.g.*, size or shape of simplices). Each Delaunay refinement inserts a batch of Steiner vertices so as to avoid creating clusters of vertices. Each optimization step relocates vertices to minimize an energy functional consistently inside the mesh and on its boundary, and is followed by a connectivity update. The optimization methods can conform to a user-defined mesh sizing or to the actual gradation of the mesh. The 3D algorithm comprises a sliver removal post-processing step inspired by Li's random vertex perturbation, which has been made more deterministic, and improves the quality of the mesh obtained by interleaving when it is needed.

Our whole processes of mesh generation and improvement are designed within the Delaunay framework. Our implementation thus inherits robustness and efficiency from the CGAL [cga] implementation of data structures such as the Delaunay triangulation.

The meshes generated by both algorithms presented (Algorithm 2 in 2D and Algoritm 5 in 3D), based on interleaving refinement and optimization, are experimentally shown to be smaller in terms of number of simplices. Complexity is also a quality criterion for a mesh (see Section 1.3.2), yet our main contribution stands for the

quality of the output angles. The output meshes show a good balance between simplex quality and vertex parsimony.

Our algorithms are designed to be as generic as possible. For example, it is possible to add some other user-defined criteria to complete the behavior of refinement for a particular application. The oracle for collision detection and projection (implemented as an AABB tree in 3D, see Section 3.3.1), can be extended for other types of inputs, depending on the application (see Section 4.2.1). The oracle framework makes our algorithms generic and valid as long as the associated oracle is properly defined.

The optimization steps are designed to consistently minimize an energy functional (E_{CVT} in 2D and E_{ODT} in 3D). Both vertex relocations and connectivity updates (devised to maintain the triangulation Delaunay) consistently minimize the same energies. The fact that theses energies are quadratic allows us deriving closed form formulas for vertex optimal locations. In both cases the Delaunay connectivity is the global minizer of these energies for a fixed set of vertices.

The main drawback of our 3D mesh generation method is that it does not handle input sharp creases subtending small angles. Refinement is proven to terminate when no input edge is incident to a dihedral angle smaller than 90° [RY07]. Though we experimentally show the ability of our algorithm to generate meshes with good dihedral angles, and though the sliver removal post-processing step is hill-climbing by construction, we can not provide any guarantee that the output mesh is sliver-free. Another theoretical drawback is that it requires a post-processing sliver removal step. The interleaving framework would be more elegant with a single procedure, ending up with no sliver. Finally, the optimization steps are slow. Running fewer iterations or elaborating upon faster iterations would be much desired.

Ideally we would like a mesh generation and optimization algorithm able to trade quality for time in a continuous manner. The first stages of the algorithm should be devised so as to induce a fast increase of quality. In our interleaving framework we are able to get a higher quality in the end, rather than applying refinement followed by optimization. However, the quality increasing slope is smaller at the beginning of the process than refinement alone because the optimization procedure is more time consuming.

Ideally the algorithm should be robust to dirty inputs, so as to handle for example non-manifold polyhedra, self-intersecting domains, and other degeneracies. Extending the optimization method to the anisotropic framework would also be very interesting. The algorithm could finally be enriched with a decimation operator. The introduction of a mesh adaptation loop, able to insert and remove vertices from the

mesh depending on the needs, and on the application, particularly for simulation, would be very nice.

Additional degrees of freedom can be considered for mesh optimization. For example, ODT and CVT optimization are both minimizing an interpolation error between the paraboloid and the lifted primal triangulation or dual tessellation. We could reformulate this minimization problem as an approximation problem instead of an interpolation one. The hope is to get a better minimum for the approximation error, and maybe a triangulation composed of better-shaped elements. The norm used to define this interpolation/approximation error could also be modified.

4.2 Future Work

One limitation of the implementation of our algorithms is that they are limited to a particular type of domain. In 2D, the domain must be a closed PSLG, possibly with holes and several connected components. In 3D, the domain must be described as a polyhedron, watertight and manifold.

4.2.1 Variety of inputs

4.2.1.1 Sharp creases

Some problems arise in triangle mesh generation when small angles exist in the input domain boundary.

In 2D for example, Ruppert's [Rup93] Delaunay refinement algorithm fails to terminate in presence of angles smaller than 45 deg in the input PSLG. Two input segments forming a small angle and a badly shaped facet could cause an endless loop of mutual encroachments and refinements. The segments become shorter and shorter and the algorithm does not terminate.

Some practical solutions have been proposed to handle small angles. One of them is denoted by corner lopping. This idea has been introduced by Bern, Eppstein, Gilbert [BEG90], and Ruppert [Rup95]. Corner lopping consists in modifying the input PSLG so as to remove the small angles that may hamper the termination of standard Delaunay refinement algorithms. It then refines this new PSLG and finally triangulates the cut-off corners and put back the sharp angles with these new triangles in the mesh.

To avoid this type of pre- and post-processing steps, Ruppert [Rup95] introduces the idea of using concentric circular shells around input vertices incident to small angles. Steiner vertices of edges adjacent to small angles are chosen at their intersection with these circular shells. At some point, the two edges adjacent to a small angle have the

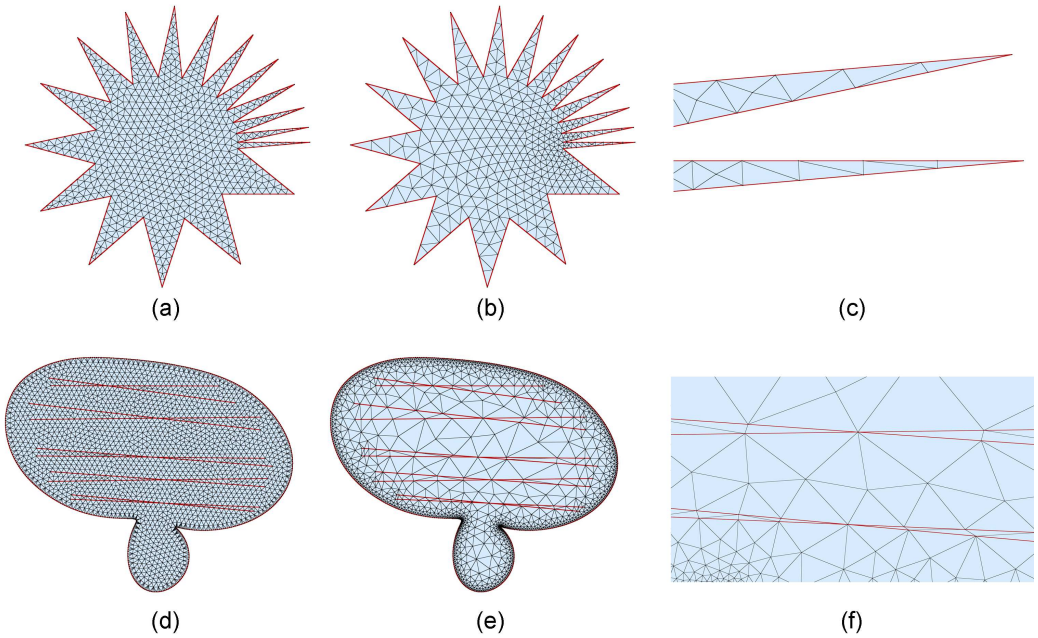


Figure 4.1: Meshes generated by interleaving refinement and optimization in 2D. (Top) Star meshes. The minimum input angle is 5°. (a) 855 vertices, with sizing function $\mu(x) = 0.03$. (b) 472 vertices, $\mu(x) = \inf_{s \in \partial_2 \Omega}[0.1d(x,s) + lfs(s)]$. (c) Close-up to the smallest angle. (Bottom) Mushroom meshes. The minimum input angle is 2.7°. (d) 2392 vertices, $\mu(x) = 0.02$. (e) 1737 vertices, $\mu(x) = \inf_{s \in \partial_2 \Omega}[0.7d(x,s) + lfs(s)]$. (f) Close-up to the smallest angle.

same length, thus they do not encroach upon each other, and refinement is prevented from infinite looping. This technique works for angles as small as 10° [She02b], and sometimes even for smaller angles.

Shewchuk proposes the Terminator algorithm [She00b], based on the concentric shells protection idea, associated to some conditions to avoid unsafe edge splits. This algorithm is shown to terminate with no condition over the input angles.

In 3D most Delaunay refinement algorithms are proven to terminate when no input dihedral angles are smaller than 90° [RY07]. The idea of using concentric shells to prevent refinement from infinite looping can also be applied using concentric spheres. Cheng et al. introduce the idea of dealing with a weighted Delaunay triangulation (*i.e.* a *regular triangulation*) in order to deal with sharp creases subtending small angles [CD02, CDRR04, CDR05, CDR07] and a practical algorithm [CDL07]. The algorithm first refines sharp edges with weighted points, represented by protecting balls. No three balls should intersect and two intersecting balls should not have too different sizes. In this framework circumcenters are replaced by orthocenters during Delaunay refinement. Inserted orthocenters are kept away from the sharp edges by these weights, which prevent infinite refinement.

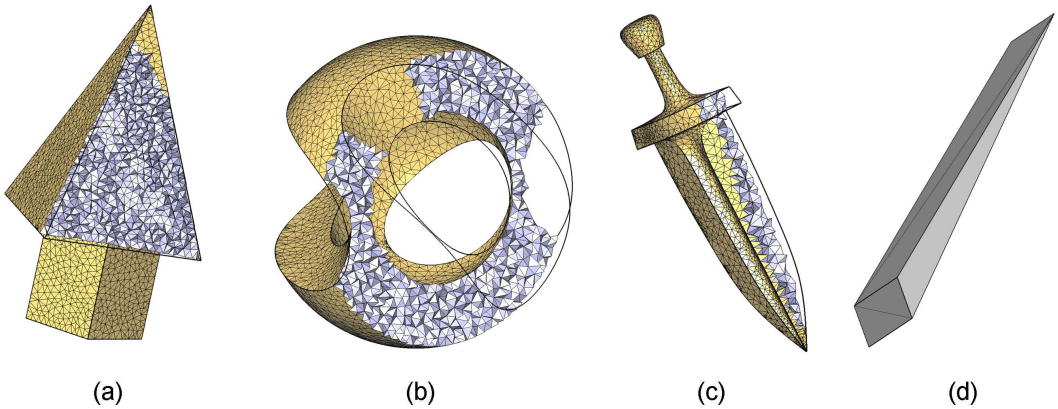


Figure 4.2: Meshes generated by interleaving refinement and optimization in 3D, from inputs with angles lower than 90°. (a) Point (9,374 vertices). The minimum input angle is 70°. (b) Sculpt (11,897 vertices). The minimum input angle is 52°. (c) Sword (3,574 vertices). The minimum input angle is 47°. (d) Blade. The minimum input angle is 5°. Interleaving fails to terminate.

In 2D, our algorithm is able to handle sharp angles: Refinement is based on a size criterion, and no size inconsistency should appear thanks to the Lloyd optimization steps which tend to locally uniformize the size of simplices. Figure 4.1 shows some examples where our interleaving algorithm terminates, though the input PSLG contains angles as small as 2.7°.

In 3D, our algorithm is sensitive to input edges subtending dihedral angles smaller than 90°, as standard Delaunay refinement algorithms [RY07]. In practice, our algorithm is able, in certain cases, to terminate when some input dihedral angles are about 40°. Note that the configuration of these sharp edges can also play a role: Refinement will not have the same trouble to terminate whether the sharp features are isolated, adjacent, close to a high curvature area, etc. Figure 4.2 shows examples where our meshing algorithm terminates though dihedral angles smaller than 90° exist in the input polyhedron.

As future work we want our mesh generator to be practical through handling any types of angles in the input polyhedron. We plan to complete our algorithm with some provably good handling of sharp edges using one of the aforementioned algorithms, or using a new approach specialized to mesh optimization.

4.2.1.2 Multi-domains

Our 3D meshing algorithm is designed to work within the framework of a restricted Delaunay triangulation. The triangulation is currently restricted to a single domain Ω, characterized by some intersection tests. Recall that a tetrahedron is considered for refinement if and only if it is inside, *i.e.*, if its circumcenter is inside Ω. For medical applications, Pons et al. [PSB+07, BPY09] and Boltcheva et al. [BYB09] consider a multi-domain $\Omega = \cup_{i=1..m}\Omega_i$ setting. The domain is divided into m sub-domains, as shown by Figure 4.3. The input data describing Ω is given as a 3D image, where each sub-domain Ω_i is tagged with its number i. Let Ω_0 be the region outside the domain Ω.

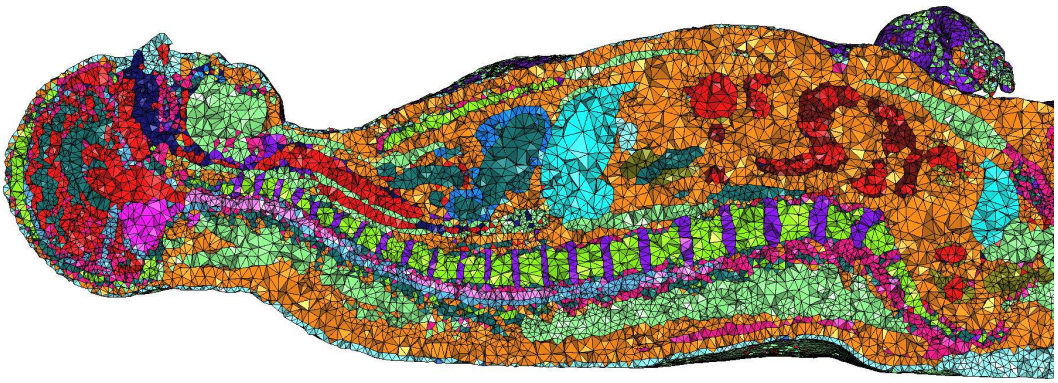


Figure 4.3: Multi-domain mesh of the *visible human*.

For multi-domain mesh generation, the definition of *restricted* must be adapted to each type of simplex. These definitions are not binary anymore, since a simplex can be restricted to different sub-domains. A tetrahedron is *restricted* to the sub-domain Ω_i if its circumcenter is in the sub-domain Ω_i. A facet is *restricted* if its dual Voronoi edge intersects sub-domains of different indices, or if at least one of its vertices is outside the domain. Note that a restricted facet is not necessarily on the boundary of the mesh, but can be at the interface between two sub-domains. Finally, an edge is *restricted* if its dual Voronoi facet intersects sub-domains of different indices. Similarly to facets, a restricted edge is not necessarily on the boundary of the mesh but can describe a junction between two, three or more sub-domains. Boltcheva et al. [BYB09] are using a weighted Delaunay triangulation to preserve in their output mesh the junction edges detected in the input 3D image.

Figure 4.3 shows a multi-domain tetrahedral mesh, generated on the well-known *visible human* data, given as a 3D image. These numerical input data have been generated from real anatomic data.

We would like to extend our algorithm to multi-domain inputs. First, our collision detection oracle, the AABB tree (see Section 3.3.1) must be extended so as to

manage this type of input. This would allow us detecting restricted simplices and their Steiner points, which are sufficient to perform refinement. Extending the optimization part would probably be more difficult. Inside vertices can be optimized in the same way they are in our mono-domain approach, with ODT or NODT iterations. Boundary vertices (incident to inside and outside tetrahedra) can also be optimized by using the NODT optimization scheme. An open question remains on how to optimize the location of vertices lying at the interface of two sud-domains. Each of these vertices should remain on its "inside surface patch", and its optimal location should take into account its incident restricted tetrahedra, and its incident restricted facets (and/or restricted edges depending on where the vertex lies).

4.2.1.3 Other types

In some application the input domain can be given through other definitions. For example, the boundary can be defined as implicit surfaces or as piecewise smooth parametric surfaces represented by NURBS patches or by parametric function. To handle these types of inputs - or others - our algorithm needs an extended oracle. Collision detection, intersection tests and projections query are needed whatever type of input is given.

4.2.2 Convergence speedup

The mesh optimization procedures we have described, CVT, ODT and NODT, are powerful for improving the quality of a mesh. The main drawback of these optimization methods is their computational cost. In particular, a large number of iterations is often needed to reach convergence. Since these methods aim at minimizing an energy functional, let us go back to this more abstract point of view.

Functional minimization is a very well-known problem in numerical analysis. Many ideas have been developed for reaching a good minimum (preferably the global minimum) in a small number of iterations. Among them, the Newton method can be applied to the problem of minimizing CVT and ODT energies. Indeed, CVT and ODT vertex position updates can be formulated as $\mathbf{x}^{n+1} = F(\mathbf{x}^n)$, where $\mathbf{x}^n$ is the vector of all vertex positions $\{\mathbf{x}_k^n\}_{k=1..N}$ at iteration n, and F the function that applies the update rule to every vertex.

Let us remind the basics of Newton's method. For the non-linear system $F(\mathbf{x}) = 0$, with $\mathbf{x}$ a vector, the *Newton iteration* is given by

$$\mathbf{x}^{n+1} = \mathbf{x}^n - \mathbf{d}F|_{\mathbf{x}^n}^{-1} F(\mathbf{x}^n),$$

where $\mathbf{d}F$ is the Jacobian matrix of the application F.

Lloyd-Newton Du and Emelianenko [DE06a] introduce the idea of coupling Newton and Lloyd methods to speedup CVT optimization convergence. In this context, F is taken as the Lloyd map F_{CVT} from generators to centroids. This approach consists in finding a fixed point of the Lloyd map, or computing a minimum of the function $\frac{1}{2}||z - F_{CVT}(z)||^2$. CVT is a solution of a system of non-linear equations, and Newton's method can be re-written as follows:

$$\mathbf{x}^{n+1} = \mathbf{x}^n + (\mathbf{d}F_{CVT}|_{\mathbf{x}^n} - \mathbf{I})^{-1}(\mathbf{x}^n - F_{CVT}(\mathbf{x}^n)).$$

The matrix $(\mathbf{d}F_{CVT}|_{\mathbf{x}^n} - \mathbf{I})$ has dimensions $dN \times dN$, where d is the dimension. Given this matrix, a Lloyd-Newton iteration can be performed.

Algorithm 8 describes the Lloyd-Newton iteration. Lloyd iterations are executed while the vertex displacements from generators z to centroids $z^\star$ are large enough. This displacement is evaluated as a difference of the distortion functional

$$\mathcal{H}(z) = \sum_{i=1}^{N} \int_{\mathcal{V}(z_i)} \rho(y)|y - z_i|^2 dy.$$

Algorithm 8 Lloyd-Newton iteration [DE06a]

Input: A domain $\Omega \in \mathbb{R}^3$,
$z = \{z_i\}_{i=1}^N$ the initial set of generators,
a tolerance value ϵ.
(1) Let $\mathcal{V}(z)$ be the Voronoi tessellation of Ω with generators $z = \{z_i\}_{i=1}^N$.
(2) Compute the centroids $z^\star = \{z_i^\star\}_{i=1}^N$.
if $||\mathcal{H}(z^\star) - \mathcal{H}(z)|| \geq \epsilon$ **then**
$z := z^\star$ and goto (1).
else
set Newton step size $s = 1$.
(3) Compute $\tilde{z} = z + s(\mathbf{d}F_{CVT}|_z - \mathbf{I})^{-1}(z - F_{CVT}(z))$.
(4) Let n_{out} be the number of points in $\tilde{z} = \{\tilde{z}_i^\star\}_{i=1}^N$ that are outside of Ω.
if $n_{out} = 0$ **then**
$z := \tilde{z}$ and goto (3).
else if $n_{out} = 1$ **then**
reduce step size $s := s/2$ and goto (3).
else
goto (1).
Output: A CVT with generators $\{z_i\}_{i=1}^N$ in $\mathbb{R}^3$.

It is shown that such a combination of Lloyd iteration and the Newton method improves the convergence rate of CVT optimization. However, the main drawback of this method is that, when the Lloyd-Newton iteration converges, it may converge to a local minimum of $\frac{1}{2}\ ||z - F_{CVT}(z)||^2$, which can be an unstable critical point of F_{CVT}.

ODT-Newton Similarly to Du and Emelianenko's Lloyd-Newton [DE06a] method, Newton's method can be applied to speedup the ODT optimization convergence. Let F_{ODT} be the functional defined by an ODT iteration. Recall that the formula used during the ODT vertex position update is (Equation 3.5):

$$\mathbf{x}_i^\star = \frac{1}{|\Omega(\mathbf{x}_i)|} \sum_{T_j \in \Omega(\mathbf{x}_i)} |T_j(\mathbf{x}_i)| \mathbf{c}_j(\mathbf{x}_i),$$

where the dependence on $\mathbf{x}_i$ is carefully noted for clarity.

In more explicit notation, we have:

$$\mathbf{x}^{n+1} = F_{ODT} \begin{pmatrix} x_1^n \\ x_2^n \\ \vdots \\ x_N^n \end{pmatrix} = \begin{pmatrix} \frac{1}{|\Omega(\mathbf{x}_1^n)|} \sum_{T_j \in \Omega(\mathbf{x}_1^n)} |T_j(\mathbf{x}_1^n)| \mathbf{c}_j(\mathbf{x}_1^n) \\ \frac{1}{|\Omega(\mathbf{x}_2^n)|} \sum_{T_j \in \Omega(\mathbf{x}_2^n)} |T_j(\mathbf{x}_2^n)| \mathbf{c}_j(\mathbf{x}_2^n) \\ \vdots \\ \frac{1}{|\Omega(\mathbf{x}_N^n)|} \sum_{T_j \in \Omega(\mathbf{x}_N^n)} |T_j(\mathbf{x}_N^n)| \mathbf{c}_j(\mathbf{x}_N^n) \end{pmatrix}.$$

Applied to F_{ODT}, Newton's update rule becomes

$$\mathbf{x}^{n+1} = \mathbf{x}^n + (\mathbf{d}F_{ODT}|_{\mathbf{x}^n} - \mathbf{I})^{-1}(\mathbf{x}^n - F_{ODT}(\mathbf{x}^n)),$$

where $\mathbf{d}F_{ODT}$ is the Jacobian matrix associated to F_{ODT}. Computing $\mathbf{d}F_{ODT}$ is the key issue. It amounts to computing the N^2 matrices $\frac{\partial F_i}{\partial \mathbf{x}_k}$. Each coefficient $\mathbf{d}F_{ODT,i,j}$ can be computed starting from the original formula

$$F_{ODT,i}(\mathbf{x}) = \mathbf{x}_i^\star = \mathbf{x}_i - \frac{1}{2|\Omega_i|} \sum_{T_j \in \Omega_i} \left(\nabla_{\mathbf{x}_i} |T_j| \left[\sum_{\mathbf{x}_k \in T_j} ||\mathbf{x}_i - \mathbf{x}_k||^2 \right] \right).$$

As we have seen in Section 3.2.1.5, $\frac{\partial |\Omega_i|}{\partial \mathbf{x}_i} = 0$. It follows that:

$$\frac{\partial F_{ODT,i}}{\partial \mathbf{x}_i}(x) = x - \frac{1}{|\Omega_i|} \sum_{T_j \in \Omega_i} \left(\nabla_{\mathbf{x}_i} |T_j| \left\langle \sum_{\mathbf{x}_k \in T_j} \mathbf{x}_i - \mathbf{x}_k, x \right\rangle \right).$$

Consider now the case of $\frac{\partial F_i}{\partial \mathbf{x}_k}$, with $k \neq i$. We have, by applying the chain rule:

$$\begin{aligned} \frac{\partial F_{ODT,i}}{\partial \mathbf{x}_k}(x) \quad = \quad & \frac{1}{2\,|\Omega_i|^2} \left\langle \sum_{T_j \in \Omega_i \cap \Omega_k} \nabla_{\mathbf{x}_k} |T_j|, x \right\rangle \sum_{T_j \in \Omega_i} \left(\nabla_{\mathbf{x}_i} |T_j| \left[\sum_{\mathbf{x}_n \in T_j} ||\mathbf{x}_i - \mathbf{x}_n||^2 \right] \right) \\ & - \frac{1}{|\Omega_i|} \left(\sum_{T_j \in \Omega_i \cap \Omega_k} \nabla_{\mathbf{x}_i} |T_j| \right) \langle \mathbf{x}_k - \mathbf{x}_i, x \rangle \\ & - \frac{1}{2\,|\Omega_i|} \sum_{T_j \in \Omega_i \cap \Omega_k} \left(\frac{\partial \nabla_{\mathbf{x}_i} |T_j|}{\partial \mathbf{x}_k}(x) \left[\sum_{\mathbf{x}_n \in T_j} ||\mathbf{x}_i - \mathbf{x}_n||^2 \right] \right). \end{aligned}$$

We still have to compute $\frac{\partial \nabla_{\mathbf{x}_i}|T_j|}{\partial \mathbf{x}_k}$, for $T_j \in \Omega_i \cap \Omega_k$. Denote by $\mathbf{x}_k$, $\mathbf{x}_p$, $\mathbf{x}_q$ and $\mathbf{x}_i$ the vertices of T_j in direct order. Then,

$$\begin{aligned} \nabla_{\mathbf{x}_i}|T_j| &= \frac{1}{6}(\mathbf{x}_k - \mathbf{x}_p) \times (\mathbf{x}_k - \mathbf{x}_q) \\ \frac{\partial \nabla_{\mathbf{x}_i}|T_j|}{\partial \mathbf{x}_k}(x) &= \frac{1}{6}\ (x \times (\mathbf{x}_k - \mathbf{x}_q) + (\mathbf{x}_k - \mathbf{x}_p) \times x) \\ &= \frac{1}{6}\ x \times (\mathbf{x}_p - \mathbf{x}_q). \end{aligned}$$

Though the derivatives of F_{ODT} are intricate our preliminary experiments are encouraging. It seems that Algorithm 8 could be applied to the ODT mesh optimization scheme. Formulas are needed to be simplified or approximated to get some reasonable computation times, in particular for the matrix inversion.

Quasi-Newton-Lloyd Liu et al. [LWL+09] have recently proposed to use quasi-Newton methods for speeding-up the computation of CVT [DFG99]. This approach consists in minimizing the CVT energy function directly, instead of computing a fixed point of the Lloyd map. They prove the $\mathcal{C}^2$ smoothness of the CVT energy function, and perform quasi-Newton optimization steps to compute the minimum. Quasi-Newton optimization is done through the *BFGS* method [Noc80, WN06]. It can be done since the very first iteration, unlike the previous Lloyd-Newton algorithm. It is also used for optimization of surface meshes in the context of surface remeshing [YLL+].

This method is shown to give a nice speedup on optimization convergence. Moreover, it should not converge to an unstable critical point of the energy. As future work, we are thinking of exploring some quasi-Newton-ODT algorithms to minimize directly E_{ODT} and speedup the convergence.

4.2.3 Optimization of a regular triangulation

As shown in 3D, *regular* triangulations (*i.e.*, weighted Delaunay triangulations) are useful for handling small input angles [DL08], and for removing slivers [CDE+00]. Both of these features make tetrahedron meshing practical, and help generating quality meshes.

As we wish to mix these approaches with our optimization methods to generate tetrahedral meshes of higher quality, we are thinking about some optimization scheme in a weighted Delaunay triangulation. Initial weights are inserted in the mesh for handling sharp input creases, or as a post-processing sliver exudation step. Given this weighted Delaunay triangulation, how could ODT, NODT or CVT be extended?

Let us consider the ODT energy

$$E_{\text{ODT}} = ||f_{\text{PWL}} - f||_{\mathcal{L}^1} = \sum_{T_j \in \mathcal{DT}} \int_{T_j} |f_{\text{PWL}} - f|.$$

Recall that $f(\mathbf{x}) = ||\mathbf{x}||^2$ and f_{PWL} is the linear function interpolating the values of f at the vertices of each tetrahedron T_j. E_{ODT} measures the volume between the 4D paraboloid defined by f and its inscribed piecewise linear approximation f_{PWL} through lifting the triangulation onto the 4D paraboloid [BWY07].

A vertex $\mathbf{x} = (\mathbf{x}_x, \mathbf{x}_y, \mathbf{x}_z)$ of $\mathcal{DT}$ is lifted onto the paraboloid $\mathcal{P}_4$ in $\mathbb{R}^4$ by computing $\mathbf{x}^{\mathcal{P}_4} = (\mathbf{x}, ||\mathbf{x}||^2) = (\mathbf{x}_x, \mathbf{x}_y, \mathbf{x}_z, \mathbf{x}_x^2 + \mathbf{x}_y^2 + \mathbf{x}_z^2)$ to be its location. A weighted vertex $\mathbf{x}' = (\mathbf{x}_x, \mathbf{x}_y, \mathbf{x}_z, \mathbf{x}_w)$ of the weighted triangulation $\mathcal{DT}^w$ is lifted onto the paraboloid $\mathcal{P}_4$ by computing its location $\mathbf{x}^{\mathcal{P}_4} = (\mathbf{x}_x, \mathbf{x}_y, \mathbf{x}_z, \mathbf{x}_x^2 + \mathbf{x}_y^2 + \mathbf{x}_z^2 - \mathbf{x}_w)$. Hence, in the weighted setting, lifted vertices do not interpolate the paraboloid in $\mathbb{R}^4$ anymore, but approximate it (see Figure 4.4).

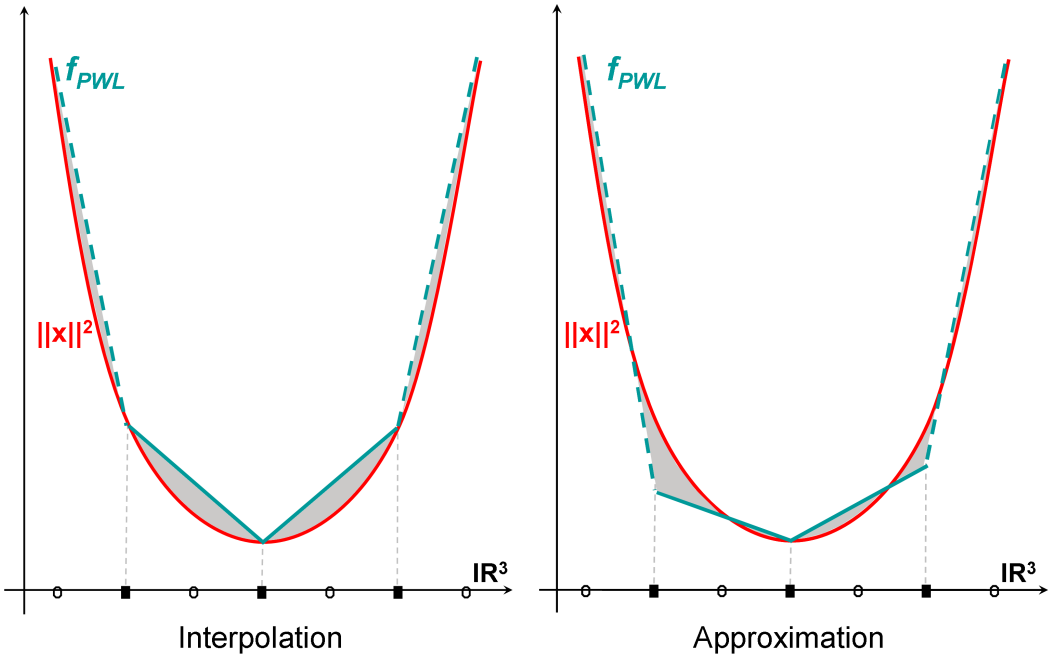


Figure 4.4: Interpolation vs approximation of the paraboloid. $||f_{\text{PWL}} - f||_{\mathcal{L}^1}$ is represented by the grey-shaded areas.

One difficulty with weighted triangulation lies in the existence of *hidden points*. When the weighted triangulation is lifted, it may be different from its convex hull in $\mathbb{R}^4$. Let $v_4^{\mathcal{P}}$ be a lifted vertex that is not on the convex hull. Then v (the vertex in $\mathbb{R}^3 \times \mathbb{R}$) is said to be *hidden*. It does not appear in the (unlifted) triangulation.

Two approaches for optimizing a weighted triangulation can be investigated upon. First, one could consider a weighted vertex as a vertex in $\mathbb{R}^4$, and minimize the approximation error by changing location and weight at the same time. Second, one

could alternate the optimization of vertex locations in $\mathbb{R}^3$, and of vertex weights, in $\mathbb{R}$. There are a lot of degrees of freedom in this approach. First, the approximation error between the lifted weighted triangulation and the paraboloid $\mathcal{P}_4$ can be estimated according to different norms. For example, the $\mathcal{L}^\infty$ norm could be a valid choice in order to remove the worst elements first.

Many open questions remain around the idea of optimizing a weighted triangulation by minimizing the volume (whatever norm is chosen) between the lifted triangulation and $\mathcal{P}_4$. Would this weighted triangulation optimizer this way embed a sliver exuder as part of its optimization procedure? This would be a desired property of this 4D optimization scheme.

Bibliography

[AB99] N. Amenta and M. Bern. Surface reconstruction by Voronoi filtering. *Discrete and Computational Geometry*, 22(4):481–504, 1999. 9

[ABE97] N. Amenta, M. Bern, and D. Eppstein. Optimal point placement for mesh smoothing. In *Proceedings of the 8th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 528–537. Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 1997. 18, 30, 49, 73

[ACSYD05] P. Alliez, D. Cohen-Steiner, M. Yvinec, and M. Desbrun. Variational tetrahedral meshing. *ACM Transactions on Graphics*, 24(3):617–625, july 2005. 42, 50, 58, 59, 63

[ADA07] L. Antani, C. Delage, and P. Alliez. Mesh Sizing with Additively Weighted Voronoi Diagrams. In *Proceedings of the 16th International Meshing Roundtable*, pages 335–346, 2007. 61

[AHMP07] U.A. Acar, B. Hudson, G.L. Miller, and T. Phillips. SVR: Practical engineering of a fast 3D meshing algorithm. In *Proceedings of 16th International Meshing Roundtable*, pages 45–62, 2007. 48

[ATP84] D.A. Anderson, J.C. Tannehill, and R.H. Pletcher. Computational fluid mechanics and heat transfer. *Hemisphere, New York*, pages 132–135, 1984. 5

[ATW09] P. Alliez, S. Tayeb, and C. Wormser. AABB tree. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.5 edition, 2009. 80

[BA76] I. Babuska and A.K. Aziz. On the angle condition in the finite element method. *SIAM Journal on Numerical Analysis*, pages 214–226, 1976. 11

[BCER95] M. Bern, P. Chew, D. Eppstein, and J. Ruppert. Dihedral bounds for mesh generation in high dimensions. In *Proceedings of the 6th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 189–196. Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 1995. 13

[BEG90] M. Bern, D. Eppstein, and J. Gilbert. Provably good mesh generation. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, pages 231–241, 1990. 16, 89

[BGO05] J-D. Boissonnat, L.J. Guibas, and S. Oudot. Learning smooth objects by probing. In *Proceedings of the 21st annual Symposium on Computational Geometry*, pages 198–207. ACM New York, NY, USA, 2005. 14

[BGR88] B.S. Baker, E. Grosse, and C.S. Rafferty. Nonobtuse triangulation of polygons. *Discrete and Computational Geometry*, 3(1):147–168, 1988. 16

[BH96] F.J. Bossen and P.S. Heckbert. A pliant method for anisotropic mesh generation. In *Proceedings of the 5th International Meshing Roundtable*, pages 63–74, oct 1996. 27, 29

[BO05] J-D. Boissonnat and S. Oudot. Provably good sampling and meshing of surfaces. *Graphical Models*, 67(5):405–451, 2005. 9

[BOG02] C. Boivin and C. Ollivier-Gooch. Guaranteed-quality triangular mesh generation for domains with curved boundaries. *International Journal for Numerical Methods in Engineering*, 55:1185–1213, 2002. 48

[BPY09] J-D. Boissonnat, J.P. Pons, and M. Yvinec. From segmented images to good quality meshes using Delaunay refinement. *Lecture Notes In Computer Science*, pages 13–37, 2009. 92

[BS89] R.E. Bank and L.R. Scott. On the conditioning of finite element equations with highly refined meshes. *SIAM Journal on Numerical Analysis*, pages 1383–1394, 1989. 11

[BWY07] J-D. Boissonnat, C. Wormser, and M. Yvinec. Curved Voronoi diagrams. In *Effective Computational Geometry for Curves and Surfaces*, pages 67–116. Springer, 2007. 58, 65, 97

[BYB09] D. Boltcheva, M. Yvinec, and J-D. Boissonnat. Mesh generation from 3d multi-material images. In *MICCAI '09: Proceedings of the 12th International Conference on Medical Image Computing and Computer-Assisted Intervention*, London, UK, 2009. Springer-Verlag. 6, 77, 92

[CC06] A.N. Chernikov and N.P. Chrisochoides. Generalized Delaunay Mesh Refinement: From Scalar to Parallel. In *Proceedings of the 15th International Meshing Roundtable*, pages 563–580, 2006. 16, 29

[CD02] S.W. Cheng and T.K. Dey. Quality meshing with weighted Delaunay refinement. In *Proceedings of the 13th Symposium on Discrete Algorithms*, pages 137–146. Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 2002. 22, 65, 90

[CDA96] S. Cotin, H. Delingette, and N. Ayache. Real time volumetric deformable models for surgery simulation. *Lecture Notes in Computer Science*, 1131:535–540, 1996. 5

[CDE+00] S.W. Cheng, T.K. Dey, H. Edelsbrunner, M.A. Facello, and S.H. Teng. Sliver exudation. *Journal of the ACM (JACM)*, 47(5):883–904, 2000. 12, 22, 49, 63, 65, 96

[CDL07] S.W. Cheng, T.K. Dey, and J.A. Levine. A Practical Delaunay Meshing Algorithm for a Large Class of Domains. In *Proceedings of the 16th International Meshing Roundtable*, pages 477–494. Springer, 2007. 16, 48, 54, 82, 90

[CDM04] B. Cutler, J. Dorsey, and L. McMillan. Simplification and improvement of tetrahedral models for simulation. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 93–102. ACM New York, NY, USA, 2004. 65

[CDR05] S.W. Cheng, T.K. Dey, and T. Ray. Weighted Delaunay refinement for polyhedra with small angles. *Proceedings of the 14th International Meshing Roundtable*, 2005. 22, 65, 90

[CDR07] S.W. Cheng, T.K. Dey, and E.A. Ramos. Delaunay Refinement for Piecewise Smooth Complexes. In *Proceedings of the 18th Symposium on Discrete Algorithms*, pages 1096–1105, 2007. 16, 48, 90

[CDRR04] S.W. Cheng, T.K. Dey, E.A. Ramos, and T. Ray. Quality meshing for polyhedra with Small Angles. *Proceedings of the 20th Symposium on Computational Geometry*, pages 290–299, 2004. 16, 90

[cga] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org. 38, 73, 78, 87

[CGS06] N. Coll, M. Guerrieri, and J.A. Sellares. Mesh Modification Under Local Domain Changes. In *Proceedings of the 15th International Meshing Roundtable*, pages 39–56, 2006. 27, 29

[Che89a] L.P. Chew. Constrained Delaunay triangulations. *Algorithmica*, 4(1):97–108, 1989. 28, 34

[Che89b] L.P. Chew. Guaranteed-quality triangular meshes. Technical report, Dept. of Computer Science, Cornell Univ., 1989. 14, 48

[Che93] L.P. Chew. Guaranteed-quality mesh generation for curved surfaces. In *Proceedings of the 9th Symposium on Computational Geometry*, pages 274–280. ACM New York, NY, USA, 1993. 48

[Che97] L.P. Chew. Guaranteed-quality Delaunay meshing in 3D (short version). In *Proceedings of the 13th Symposium on Computational Geometry*, pages 391–393. ACM New York, NY, USA, 1997. 14, 64

[Che04] L. Chen. Mesh Smoothing Schemes based on Optimal Delaunay Triangulations. In *Proceedings of 13th International Meshing Roundtable*, pages 109–120, 2004. 20, 30, 50, 58, 59

[Che05] L. Chen. *Robust and accurate algorithms for solving anisotropic singularities*. PhD thesis, The Pennsylvania State University, The Graduate school., dec. 2005. 19

[CTS98] S.A. Canann, J.R. Tristano, and M.L. Staten. An approach to combined Laplacian and optimization-based smoothing for triangular, quadrilateral, and quad-dominant meshes. In *Proceedings of the 7th International Meshing Roundtable*, pages 479–494. Citeseer, 1998. 18

[CX04a] L. Chen and J. Xu. Optimal Delaunay triangulation. *Journal of Computational Mathematics*, 22:299–308, 2004. 73

[CX04b] L. Chen and J.C. Xu. Optimal delaunay triangulations. *Journal of Computational Mathematics*, 22(2):299–308, 2004. 50

[DE06a] Q. Du and M. Emelianenko. Acceleration schemes for computing Centroidal Voronoi Tessellations. *Journal of Numerical Linear Algebra and Applications*, 13(2–3):173–192, mar-apr 2006. 20, 30, 94, 95

[DE06b] Q. Du and M. Emelianenko. Uniform convergence of a nonlinear energy-based multilevel quantization scheme via Centroidal Voronoi Tessellations. *submitted to SIAM Journal of Numerical Analysis*, 2006. 30

[DEJ06] Q. Du, M. Emelianenko, and L. Ju. Convergence of the Lloyd algorithm for computing Centroidal Voronoi Tesselations. *SIAM Journal, Numerical Analysis*, 44(1):102–119, 2006. 30, 33

[Del34] B. Delaunay. Sur la sphère vide. *Bulletin de l'Academie des Sciences d'URSS. VII Serie. Classe des Sciences Mathematiques et Naturelles.*, pages 793–800, 1934. 7

[DFG99] Q. Du, V. Faber, and M. Gunzburger. Centroidal Voronoi Tessellations: Applications and Algorithms. *SIAM review*, 41(4):637–676, 1999. 19, 30, 33, 73, 82, 96

[DGJ03] Q. Du, M. Gunzburger, and L. Ju. Constrained Centroidal Voronoi Tessellations for Surfaces. *SIJSSC: SIAM Journal on Scientific and Statistical Computing*, 24, 2003. 34

[DL08] T.K. Dey and J.A. Levine. DelPSC: a Delaunay mesher for piecewise smooth complexes. In *Proceedings of the 24th annual Symposium on Computational Geometry*, pages 220–221. ACM New York, NY, USA, 2008. 96

[dlG95] E.B. de l'Isle and P.L. George. Optimization of tetrahedral meshes. *Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations*, page 97, 1995. 18

[DPS+06] H. Delingette, X. Pennec, L. Soler, J. Marescaux, and N. Ayache. Computational models for image-guided robot-assisted and simulated medical interventions. *Proceedings of the IEEE*, 94(9):1678–1688, 2006. 5

[DW02] Q. Du and D. Wang. Tetrahedral mesh generation and optimization based on centroidal Voronoi tessellations. *International Journal for Numerical Methods in Engineering*, 56:1355–1373, 2002. 19

[DW05] Q. Du and D. Wang. Recent progress in robust and quality Delaunay mesh generation. *Journal of Computational and Applied Mathematics*, 2005. 36

[Ede87] H. Edelsbrunner. *Algorithms in combinatorial geometry*. Springer Verlag, 1987. 65

[EG02] H. Edelsbrunner and D. Guoy. An experimental study of sliver exudation. *Engineering with computers*, 18(3):229–240, 2002. 22, 65

[ELM+00] H. Edelsbrunner, X.Y. Li, G. Miller, A. Stathopoulos, D. Talmor, S.H. Teng, A. ʻUngʻor, and N. Walkington. Smoothing and cleaning up slivers. In *Proceedings of the 22nd annual ACM symposium on Theory of computing*, pages 273–277. ACM New York, NY, USA, 2000. 63

[Epp01] D. Eppstein. Global optimization of mesh quality. In *Tutorial at the 10th International Meshing Roundtable*, volume 10, 2001. 13, 14, 19, 30

[EÜ07] H. Erten and A. Üngör. Triangulations with locally optimal Steiner points. *Eurographics Symposium on Geometry Processing*, pages 1–10, 2007. 29

[FG07] P.J. Frey and P.L. George. *Mesh generation: application to finite elements*. ISTE, 2007. 6, 17

[FGK+00] A. Fabri, G.J. Giezeman, L. Kettner, S. Schirra, and S. Sch "onherr. On the design of CGAL, a Computational Geometry Algorithms Library. *Software Practice and Experience*, 30(11):1167–1202, 2000. 38

[Fie88] D.A. Field. Laplacian smoothing and Delaunay triangulations. *Communications in Applied Numerical Methods*, 4(6):709–712, 1988. 18

[FK94] K. Forsman and L. Kettunen. Tetrahedral mesh generation in convex primitives by maximizing solid angles. *IEEE Transactions on Magnetics*, 30(5 Part 2):3535–3538, 1994. 65

[Fle99] P. Fleischmann. Mesh generation for technology CAD in three dimensions, 1999. 5

[FOG97] L.A. Freitag and C. Ollivier-Gooch. Tetrahedral mesh improvement using swapping and smoothing. *International Journal for Numerical Methods in Engineering*, 40(21):3979–4002, 1997. 49

[Fuc98] A. Fuchs. Automatic grid generation with almost regular Delaunay tetrahedra. In *Proceedings of the 7th International Meshing Roundtable*. Citeseer, 1998. 16

[Geo91] P.L. George. *Automatic mesh generation*. Wiley Chichester, 1991. 18

[Geo04] P-L. George. Tetmesh-GHS3D, Tetrahedral mesh generator. *INRIA User's Manual, INRIA (Institut National de Recherche en Informatique et Automatique), France*, 2004. 83

[HDD+92] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. *Computer Graphics*, 26:71–79, 1992. 5

[Her76] L.R. Herrmann. Laplacian-isoparametric grid generation scheme. *Journal of the Engineering Mechanics Division*, 102(5):749–907, 1976. 18

[Jam76] P. Jamet. Estimations d'erreur pour des éléments finis droits presque dégénérés, RAIRO Anal. *Numer*, 10:46–61, 1976. 11

[Joe91] B. Joe. Delaunay versus max-min solid angle triangulations for 3-dimensional mesh generation. *International Journal for Numerical Methods in Engineering*, 31(5), 1991. 17, 22, 66

[Joe95] B. Joe. Construction of 3-dimensional improved-quality triangulations using local transformations. *SIAM Journal on Scientific Computing*, 16(6):1292–1307, 1995. 65

[KMOD09] L. Kharevych, P. Mullen, H. Owhadi, and M. Desbrun. Numerical coarsening of inhomogeneous elastic materials. In *ACM SIGGRAPH Transactions on Graphics*, volume 28, page 51. ACM, 2009. 5, 6

[KO01] P. Krysl and M. Ortiz. Variational Delaunay approach to the generation of tetrahedral finite element meshes. *International Journal for Numerical Methods in Engineering*, 50(7):1681–1700, 2001. 65

[Kri92] M. Krizek. On the maximum angle condition for linear tetrahedral elements. *SIAM Journal on Numerical Analysis*, 29(2):513–520, 1992. 11

[KS07] B.M. Klingner and J.R. Shewchuk. Aggressive Tetrahedral Mesh Improvement. In *Proceedings of the 16th International Meshing Roundtable*, pages 3–23, 2007. 23, 49, 65

[KTY09] R. Kikuuwe, H. Tabuchi, and M. Yamamoto. An edge-based computationally efficient formulation of Saint Venant-Kirchhoff tetrahedral finite elements. *ACM Transactions on Graphics*, 28(1):1–13, 2009. 5

[Lab06] F. Labelle. Sliver removal by lattice refinement. In *Proceedings of the 22nd annual Symposium on Computational Geometry*, pages 347–356. ACM New York, NY, USA, 2006. 64

[Law77] C.L. Lawson. Software for C^1 surface interpolation. *Mathematical Software*, 3:161–194, 1977. 14

[LH01] C.Y. Liu and C.J. Hwang. New strategy for unstructured mesh generation. *AIAA journal*, 39(6):1078–1085, 2001. 64

[Li00a] X.Y. Li. *Sliver-free 3-Dimensional Delaunay Mesh Generation*. PhD thesis, University of Illinois at Urbana-Champaign, 2000. 12, 21, 25, 63, 64, 66, 69, 72

[Li00b] X.Y. Li. Spacing control and sliver-free Delaunay mesh. In *Proceedings of the 9th International Meshing Roundtable*, pages 295–306, 2000. 64

[Llo82] S. Lloyd. Least square quantization in PCM. *IEEE Transactions on Information Theory*, 28:129–137, 1982. 19, 33

[LM98] B. Lévy and J-L. Mallet. Non-distorted texture mapping for sheared triangulated meshes. In *SIGGRAPH Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 343–352. ACM New York, NY, USA, 1998. 5

[LP01] P. Lindstrom and V. Pascucci. Visualization of large terrains made easy. In *Proceedings of IEEE Visualization*, volume 574, pages 363–370, 2001. 5

[LS07] F. Labelle and J.R Shewchuk. Isosurface Stuffing: Fast tetrahedral meshes with good dihedral angles. In *ACM Transactions on Graphics*, volume 26(3), page 57, 2007. 11, 17, 47, 66

[LT01] X.Y. Li and S.H. Teng. Generating well-shaped Delaunay meshes in 3D. In *Proceedings of the 12th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 28–37. Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 2001. 14, 64

[LWL+09] Y. Liu, W. Wang, B. Lévy, F. Sun, D.M. Yan, L. Lu, and C. Yang. On Centroidal Voronoi Tessellation. Energy smoothness and fast computation. *ACM Transactions on Graphics*, 2009. To appear. 20, 96

[MCP+09] P. Mullen, K. Crane, D. Pavlov, Y. Tong, and M. Desbrun. Energy-Preserving Integrators for Fluid Animation. *ACM/SIGGRAPH Transactions on Graphics*, 28(3), 2009. 5, 6

[Mit94] S.A. Mitchell. Cardinality bounds for triangulations with bounded minimum angle. In *6th Canadian Conference on Computational Geometry*, pages 326–331, 1994. 14

[MPW05] G.L. Miller, S.E. Pav, and N.J. Walkington. When and Why Delaunay refinement algorithms work. *International Journal of Computational Geometry and Applications*, 15(1):25–54, feb 2005. 28

[MTAD09] P. Machado, J. Tournois, P. Alliez, and O. Devillers. Filtering relocations on a Delaunay triangulation. *7th Symposium on Geometry Processing*, 2009. 81

[MV96] S.A. Mitchell and S.A. Vavasis. Quality mesh generation in higher dimensions. 1996. 16

[Nay99] D.J. Naylor. Filling space with tetrahedra. *International Journal for Numerical Methods in Engineering*, (44):1383–1395, 1999. 16

[NCC02] D. Nave, N. Chrisochoides, and L.P. Chew. Guaranteed quality parallel Delaunay refinement for restricted polyhedral domains. *Proceedings of the 18th Symposium on Computational Geometry*, pages 135–144, 2002. 48

[Noc80] J. Nocedal. Updating quasi-Newton matrices with limited storage. *Mathematics of computation*, pages 773–782, 1980. 96

[ORY05] S. Oudot, L. Rineau, and M. Yvinec. Meshing Volumes Bounded by Smooth Surfaces. In *Proceedings of the 14th International Meshing Roundtable*, pages 203–219, 2005. 48

[Owe98] S.J. Owen. A survey of unstructured mesh generation technology. In *Proceedings of the 7th International Meshing Roundtable*, 1998. 6

[PAH+07] H. Pottman, A. Asperl, M. Hofer, A. Kilian, and D. Bentley. *Architectural geometry*. Bentley Institute Press, 2007. 5

[PGH94] V.N. Parthasarathy, C.M. Graichen, and A.F. Hathaway. A comparison of tetrahedron quality measures. *Finite Elements in Analysis and Design*, 15(3):255–261, 1994. 13

[PS04] P.O. Persson and G. Strang. A simple mesh generator in MATLAB. *SIAM Review*, pages 329–346, 2004. 49

[PSB+07] J. Pons, F. Ségonne, J-D. Boissonnat, L. Rineau, M. Yvinec, and R. Keriven. High-quality consistent meshing of multi-label datasets. *Lecture Notes in Computer Science*, 4584:198, 2007. 6, 17, 22, 92

[Rin07] L. Rineau. 2D conforming triangulations and meshes. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.3 edition, 2007. 38

[Rup93] J. Ruppert. A new and simple algorithm for quality 2-dimensional mesh generation. In *Proceedings of the 4th annual ACM-SIAM Symposium*

on Discrete Algorithms, pages 83–92. Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 1993. 14, 89

[Rup95] J. Ruppert. A Delaunay refinement algorithm for quality 2-Dimensional mesh generation. *Journal of Algorithms*, 18(3):548–585, 1995. 14, 15, 28, 89

[RY07] L. Rineau and M. Yvinec. Meshing 3D domains bounded by piecewise smooth surfaces. *Proceedings of the 16th International Meshing Roundtable*, pages 442–460, 2007. 14, 15, 22, 48, 50, 52, 53, 54, 65, 86, 88, 90, 91

[SCL+06] G. Scarella, O. Clatz, S. Lanteri, G. Beaume, S. Oudot, J.P. Pons, S. Piperno, P. Joly, and J. Wiart. Realistic numerical modelling of human head tissue exposure to electromagnetic waves from cellular phones. *Comptes rendus-Physique*, 7(5):501–508, 2006. 17

[Sei88] R. Seidel. Constrained Delaunay triangulations and Voronoi diagrams with obstacles. Technical Report 260, IIG-TU Graz, 1988. 34

[She96] J.R. Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. *Applied Computational Geometry: Towards Geometric Engineering*, 1148:203–222, 1996. 34

[She98] J.R. Shewchuk. Tetrahedral mesh generation by Delaunay refinement. In *Proceedings of the 14th annual Symposium on Computational Geometry*, pages 86–95. ACM New York, NY, USA, 1998. 15

[She00a] J.R. Shewchuk. Mesh generation for domains with small angles. *Proceedings of the 16th annual Symposium on Computational Geometry*, pages 1–10, 2000. 38

[She00b] J.R. Shewchuk. Mesh generation for domains with small angles. In *Proceedings of the 16th Symposium on Computational Geometry*, pages 1–10, 2000. 90

[She02a] J.R. Shewchuk. Constrained Delaunay tetrahedralizations and provably good boundary recovery. In *Proceedings of the 11th International Meshing Roundtable*, pages 193–204. Citeseer, 2002. 14

[She02b] J.R. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications*, 22(1-3):21–74, 2002. 19, 28, 29, 54, 90

[She02c] J.R. Shewchuk. Two discrete optimization algorithms for the topological improvement of tetrahedral meshes. *Unpublished manuscript*, 2002. 65

[She02d] J.R. Shewchuk. What is a good linear element? interpolation, conditioning, and quality measures. In *Proceedings of the 11th International Meshing Roundtable*, pages 115–126, 2002. 6, 11, 29, 39, 58

[Si07] H. Si. TetGen, A Quality Tetrahedral Mesh Generator and 3-Dimensional Delaunay Triangulator, 2007. Available on the web at: http://tetgen.berlios.de/. 82

[SMD97] J.R. Shewchuk, G.L. Miller, and R.O.H. David. *Delaunay refinement mesh generation*. PhD thesis, Carnegie-Mellon University, Pittsburgh, PA, School of Computer Science, 1997. 14

[TAD07] J. Tournois, P. Alliez, and O Devillers. Interleaving delaunay refinement and optimization for 2d triangle mesh generation. In *Proceedings of the 16th International Meshing Roundtable*. Springer-Verlag, october 2007. 27

[Tal97] D. Talmor. *Well-spaced points for numerical methods*. PhD thesis, University of Minnesota, 1997. 73

[Ter05] P. Terdiman. OPCODE 3D Collision Detection library, 2005. http://www.codercorner.com/Opcode.htm. 78

[TSA09] J. Tournois, R. Srinivasan, and P. Alliez. Perturbing slivers in 3D Delaunay meshes. In *Proceedings of the 18th International Meshing Roundtable*, october 2009. 66

[TWAD09] J. Tournois, C. Wormser, P. Alliez, and M. Desbrun. Interleaving Delaunay refinement and optimization for practical isotropic tetrahedron mesh generation. *ACM/SIGGRAPH Transactions on Graphics*, 28(3):75:1–75:9, 2009. 51, 77

[Üng04] A. Üngör. Off-centers: A new type of steiner points for computing size-optimal quality-guaranteed Delaunay triangulations. In *LATIN: Latin American Symposium on Theoretical Informatics*, 2004. 16, 29

[Wat81] D.F. Watson. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *The Computer Journal*, 24(2):167–172, 1981. 14

[WBOL07] J.D. Wendt, W. Baxter, I. Oguz, and M.C. Lin. Finite volume flow simulations on arbitrary domains. *Graphical Models*, 69(1):19 – 32, 2007. 17, 47

[WK02] J. Wu and L. Kobbelt. Fast mesh decimation by multiple-choice techniques. *Vision, Modeling, and Visualization*, pages 241–249, 2002. 50, 55

[WN06] S.J. Wright and J. Nocedal. *Numerical optimization*. Springer, 2006. 96

[YLL+] D-M. Yan, B. Lévy, Y. Liu, F. Sun, and W. Wang. Isotropic remeshing with fast and exact computation of restricted Voronoi diagram. In *ACM/EG Symposium on Geometry Processing / Computer Graphics Forum*. To appear. 20, 96

[YS84] M.A. Yerry and M.S. Shephard. Automatic three-dimensional mesh generation by the modified-octree technique. *International Journal for Numerical Methods in Engineering*, 20(11):1965–1990, 1984. 16

Optimisation de maillages

Résumé :

Dans cette thèse, une approche pratique pour la génération de maillages triangulaires isotropes est proposée. En 2D comme en 3D, l'objectif consiste à mailler un domaine donné, pouvant avoir une géométrie complexe. L'approche présentée consiste à entrelacer des étapes de raffinement de Delaunay et des étapes d'optimisation de maillages dans le but de générer des maillages gradés de qualité. L'utilisateur peut contrôler les caractéristiques du maillage en définissant des critères de taille et de forme des simplexes, ainsi que de topologie et d'approximation. Les méthodes par éléments finis, largement utilisées en simulation, nécessitent des maillages gradés, composés de simplexes bien formés.

Des alternatives aux méthodes de raffinement de Delaunay usuelles sont développées. Les méthodes d'optimisation de maillages proposées permettent d'optimiser la position des sommets intérieurs et de ceux du bord. Les caractéristiques du bord du domaine à mailler, et en particulier des arêtes vives, sont préservées par ces méthodes. En 2D, l'optimisation est basée sur l'algorithme de Lloyd et les diagrammes de Voronoi centrés (CVT). En 3D, une extension naturelle des triangulations de Delaunay optimales (ODT) de Chen, capable d'optimiser la position des sommets du bord du maillage, est introduite. Notre algorithme de maillage tétraédrique est enrichi par une étape de post-traitement permettant d'améliorer de façon significative la qualité des angles dièdres du maillage.

Nous montrons que l'entrelacement d'étapes de raffinement et d'optimisation permet d'obtenir des maillages de meilleure qualité que ceux générés par les méthodes connues en termes d'angles dans les simplexes et de complexité.

Mesh Optimization

Abstract:

In this thesis, a practical approach to isotropic triangle mesh generation is proposed. In 2D as in 3D, the goal is to mesh the interior of a domain that possibly has a complex geometry. The studied approach consists in interleaving Delaunay refinement steps and mesh optimization steps, in order to generate quality graded meshes that satisfy size, shape, topology and approximation user-defined criteria for simplices. Widely used for simulation, finite element methods need graded meshes composed of well-shaped simplices.

Alternatives to usual Delaunay refinement methods are developed. New mesh optimization methods, able to optimize internal and boundary vertices locations and to preserve the input boundary features, including sharp creases, are studied. In 2D, the optimization method is based on the Lloyd iteration and centroidal Voronoi tessellations (CVT). In 3D, a natural extension to optimal Delaunay triangulations (ODT) introduced by Chen that is able to optimize boundary vertices locations, is proposed. Our tetrahedron meshing algorithm is completed with a post-processing step that significantly improves the quality of tetrahedra dihedral angles.

We finally give some experimental evidence that interleaving refinement and optimization results in generating meshes of higher quality than usual methods, in terms of simplices angles and number of vertices.