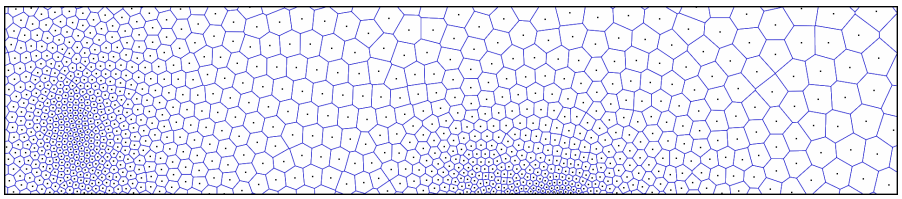
# Delaunay Tessellations and Voronoi Diagrams in CGAL

Pierre Alliez, Christophe Delage, Menelaos I Karavelas, Sylvain Pion,
Monique Teillaud, Mariette Yvinec

# Delaunay Tessellations and Voronoi Diagrams in CGAL

Pierre Alliez[1], Christophe Delage[1], Menelaos I. Karavelas[2,3], Sylvain Pion[1], Monique Teillaud[1], and Mariette Yvinec[1]

[1] INRIA Sophia Antipolis - Méditerranée, FirstName.LastName@sophia.inria.fr
[2] University of Crete, mkaravel@tem.uoc.gr
[3] Foundation for Research and Technology - Hellas



## 1 Introduction

CGAL is a C++ software library of computational geometry algorithms and data structures [CGAa]. Created in 1995, and mostly supported by European research projects, the CGAL library has become a mature Open Source project with approximately half a million lines of code, 10,000 downloads per year, and its availability through the two major Linux distributions Fedora and Debian.

The goal of the CGAL library is to provide easy access to efficient and reliable geometric algorithms to users in industry and academia. The initial motivation for setting up the CGAL library has its roots in the following assessment: While getting a first implementation of a geometric algorithm is a relatively easy task, obtaining a *robust* implementation is tremendously harder. The main reason is inherent to the dual nature of geometric objects, which combine combinatorial data structures and numerical data. Geometric algorithms have to maintain the consistency between combinatorial and numerical components. This is all the more difficult when geometric primitives are in the neighborhood of a degenerate configuration. Indeed a slight

perturbation in numerical data may then yield a drastic change in the combinatorial data structure. Therefore geometric algorithms are highly sensitive to numerical rounding errors and naive implementations, using for instance built-in floating point arithmetic, are bound to fail sooner or later. Robustness is the major concern underlying the development of the CGAL library from the beginning.

More than 50 developers have contributed to CGAL since its creation. The project is managed by an international editorial board whose charge is to review all submitted packages, identify potential contributors, schedule the new releases and ensure the global consistency of the library. In addition to the open source distribution, CGAL is commercialized by the startup company GeometryFactory since 2003. This dual distribution is made available by the Open Source licenses QPL and LGPL, which allow free use of CGAL in Open Source software, in addition to commercial licenses.

The CGAL library provides a rich variety of Voronoi diagrams and Delaunay triangulations. This variety covers several aspects: generators, dimensions and metrics, which we describe in Section 2. One aim of this paper is to present the main paradigms used in CGAL: Generic programming, separation between predicates/constructions and combinatorics, and exact geometric computation (not to be confused with exact arithmetic!). The first two paradigms translate into software design choices, described in Section 4, while the last covers both robustness and efficiency issues, respectively described in Section 6 and 7. Other important aspects of the CGAL library are the interface issues, be they for traversing a tessellation, or for interoperability with other libraries or languages, see Section 5. We present in Section 8 some tessellations at work in the context of surface reconstruction and mesh generation. Section 9 is devoted to some on-going and future work on periodic triangulations (triangulations in periodic spaces), and on high-quality mesh generation with optimized tessellations. Section 10 provides typical numbers in terms of efficiency and scalability for constructing tessellations, and lists the remaining weaknesses. We conclude by listing some of our directions for the future.

## 2 Voronoi diagrams

### 2.1 Affine diagrams

A Voronoi diagram is defined by a set of sites and a distance function. The set of sites is a set $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ of objects in a space $\Sigma$, and the distance function $\delta(x, s_i)$ measures the distance from a point $x$ of $\Sigma$ to a site $s_i$. The Voronoi cell $V(\mathcal{T})$ of a subset $\mathcal{T} \subset \mathcal{S}$, is the set of points in $\Sigma$ which are at equal distance to the sites of $\mathcal{T}$ and closer to any site of $\mathcal{T}$ than to any site in $\mathcal{S} \setminus \mathcal{T}$. The Voronoi diagram $\mathrm{Vor}(\mathcal{S}, \delta)$, simply noted $\mathrm{Vor}(\mathcal{S})$ when there is no ambiguity on the distance function, is the partition of the space $\Sigma$ formed by the non-empty Voronoi cells.

Standard Voronoi diagrams are obtained in $\mathbb{R}^d$ when the sites are points and the distance function is the Euclidean distance function, see Figure 1.

Power diagrams (also called Laguerre diagrams) are obtained when the sites are weighted points in $\mathbb{R}^d \times \mathbb{R}$, i.e., $s_i = (p_i, w_i)$ with $p_i \in \mathbb{R}^d$ and $w_i \in \mathbb{R}$, and the distance function $\delta(x, s_i)$ from a point $x$ to the site $s_i$, is the power of $x$ with respect to the sphere with center $p_i$ and square radius $w_i$:

$$\delta(x, s_i) = (x - p_i)^2 - w_i.$$

Note that the definition of power diagrams accepts negative square radii as well.

The locus of points in space $\Sigma$ that are equidistant to two sites $s_i$ and $s_j$ is called hereafter a *bisector*. Standard Voronoi diagrams and power diagrams are also called affine diagrams (see, e.g., [BWY06]) because their bisectors are affine subspaces of $\mathbb{R}^d$.

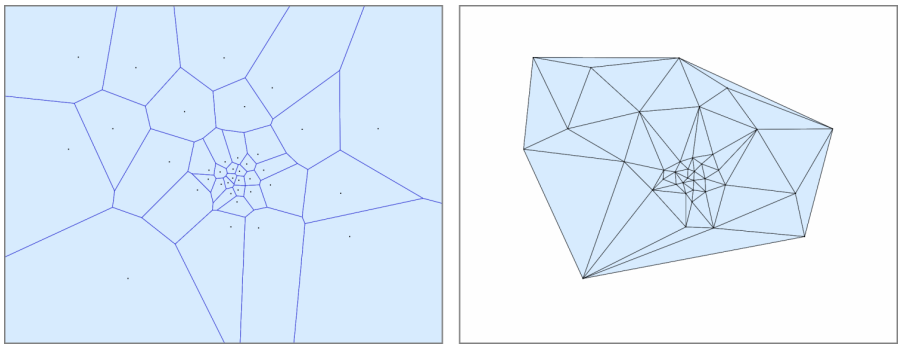


**Fig. 1.** 2D Euclidean Voronoi diagram (left) and its dual Delaunay triangulation (right).

The nerve of a Voronoi diagram is the set of subsets $\mathcal{T} \subset \mathcal{S}$ with a non-empty Voronoi cell. A degenerate configuration for a Voronoi diagram in $\mathbb{R}^d$ happens when there are points in space $\mathbb{R}^d$ equidistant to more than $d+1$ sites. In the absence of degenerate situations, each subset $\mathcal{T}$ in the nerve of $\mathrm{Vor}(\mathcal{S})$ has cardinality at most $d+1$ and can be naturally embedded as a simplex that is the convex hull $\mathrm{conv}(\mathcal{T})$ of $\mathcal{T}$. Affine diagrams share the property that, under the non-degeneracy assumption, the natural embedding of their nerve is a triangulation. Such a dual triangulation is called a Delaunay triangulation in the case of a standard Voronoi diagram and a regular triangulation in the case of a power diagram.

In the CGAL library, Voronoi diagrams and power diagrams are represented through their dual triangulations. The library provides a rich set of functionalities for Delaunay and regular triangulations in $\mathbb{R}^2$ and $\mathbb{R}^3$. The triangulations are incrementally computed and can be dynamically maintained through insertion and deletion of sites. They offer *localization* and *nearest site* queries,

where a localization query looks for the simplex of the triangulation including a given point and a nearest site queries seeks the site closest to a given point. A whole set of functionalities is also available to navigate through the triangulation or to explore a subset like the star or the link of a given vertex. Dual functions provide access to the features of the dual Voronoi or power diagram.

Using the standard lifting map,

$$(p_i, w_i) \in \mathbb{R}^d \times \mathbb{R} \longrightarrow (p_i, p_i^2 - w_i) \in \mathbb{R}^{d+1},$$

Delaunay and regular triangulations in $\mathbb{R}^d$ can be computed as the projection of the lower part of a convex hull in $\mathbb{R}^{d+1}$. The CGAL library offers convex hull computations in any dimension and a reduced interface to Delaunay triangulations and Voronoi diagrams in $\mathbb{R}^d$, including site insertion, localization and nearest site query.

### 2.2 Quadratic diagrams

Möbius diagrams, introduced by Boissonnat and Karavelas [BK03], generalize power diagrams and should appear soon in CGAL, at least in their planar version. Möbius diagrams are defined by sites which are doubly weighted points $s_i = (p_i, \lambda_i, \mu_i) \in \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}$ and the distance function $\delta(x, s_i)$ from point $x \in \mathbb{R}^d$ to the site $s_i$ is:

$$\delta(x, s_i) = \lambda_i (x - p_i)^2 - \mu_i.$$

The bisectors of Möbius diagrams are hyperspheres (see Figure 2) and in fact the class of Möbius diagrams coincides exactly with the class of diagrams whose bisectors are hyperspheres [BWY06].

In a Möbius diagram, the cells do not have to be connected and the dual of a Möbius diagram is not a triangulation of the set of sites. Even more general diagrams, whose bisectors are general conics, may be obtained when each site $s_i$ is equipped with a positive definite symmetric tensor $M_i$ such that the distance $\delta(x, s_i)$ is

$$\delta(x, s_i) = \sqrt{(x - s_i)^t M_i (x - s_i)}.$$

These diagrams are called anisotropic Voronoi diagrams [LS03].

### 2.3 Planar Euclidean diagrams

In the plane, CGAL also provides Euclidean Voronoi diagrams for extended, i.e., non-punctual, sites, namely segment Voronoi diagrams and Apollonius diagrams. In segment Voronoi diagrams, the sites are straight line segments in $\mathbb{R}^2$ and the distance function $\delta(x, s_i)$ from point $x$ to segment $s_i$ is just the Euclidean distance function:

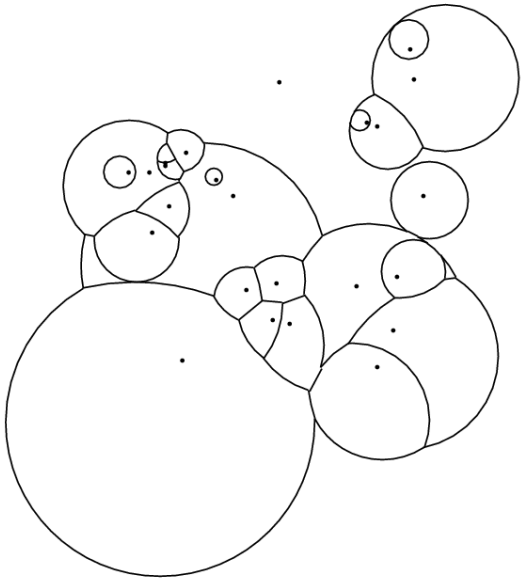


**Fig. 2.** A Möbius diagram.

$$\delta(x, s_i) = \min_{y \in s_i} ||x - y||.$$

The CGAL package for segment Voronoi diagrams accepts as input intersecting segments without restriction, meaning that input segments are allowed to intersect properly, overlap or share some endpoint, see Figure 3. However, to avoid dealing with two-dimensional bisectors and disconnected Voronoi cells the package does not consider the input segments as the sites of the diagram. Instead, the sites of the diagram are related to the subsegments defined by the arrangement of the input segments. Furthermore, the two endpoints and the relative interior of each subsegment are considered as three distinct sites. In such a setting, a segment Voronoi diagram is an instance of an abstract Voronoi diagram as described by Klein [Kle89] and can be computed via a randomized incremental algorithm [KMM93]. The CGAL implementation, fully described in [Kar04], supports on-line insertions and nearest site queries. As an easy alternative to segment Voronoi diagrams, in the case where input segments form the boundary of a polygonal region, CGAL also provides a package to compute the straight skeleton of a polygonal region. This straight skeleton subdivides the polygonal region into subregions which can be described as the intersections of the polygonal region with the Voronoi cells of the Euclidean diagram obtained when using the supporting lines of the input segments as sites.

Apollonius diagrams, also called additively weighted Voronoi diagrams, are closely related to the Euclidean Voronoi diagrams of spheres and have obvious

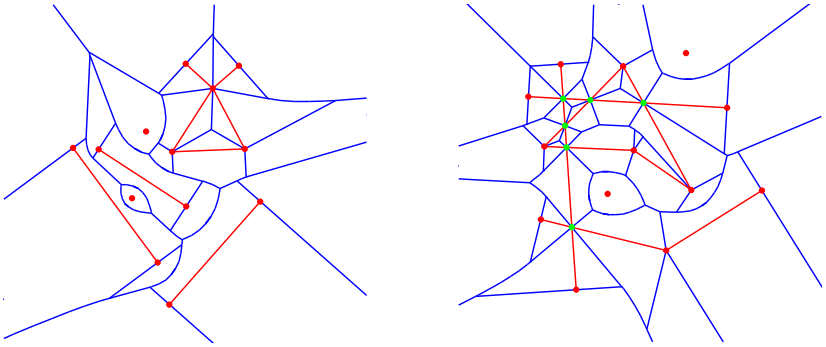


**Fig. 3.** Segment Voronoi diagrams. Left: input segments are sharing endpoints. Right: input segments are intersecting (intersection points are shown in green).

applications in structural biology. Those diagrams have been studied e.g by Kim et al. [KCK05a, KCK$^+$05b, CKK05] and Boissonnat et al. [BD05, BK03]. The sites of Apollonius diagrams are weighted points $s_i = (p_i, w_i)$ in $\mathbb{R}^d \times \mathbb{R}$ and the function measuring the distance from point $x \in \mathbb{R}^d$ to $s_i$ is

$$\delta(x, s_i) = ||x - s_i|| - w_i.$$

Because the diagram is invariant under a uniform translation of the weights of all sites, it may be assumed that all sites have non-negative weights. In that case, the weighted point $(p_i, w_i)$ may be interpreted as a sphere with center $p_i$ and radius $w_i$ and the Apollonius distance is just the signed Euclidean distance from a point $x$ to the sphere $(p_i, w_i)$, where the signed Euclidean distance from a point $x$ to a sphere $(p_i, w_i)$ is the Euclidean distance affected with a negative sign when the point $x$ lies within the ball bounded by the sphere $(p_i, w_i)$ and with a positive sign otherwise. As any other type of Voronoi diagrams, Apollonius diagrams can also be described as the projection of the lower envelope of the set of functions $\delta_i(x) = \delta(x, s_i)$. In the case of Apollonius diagrams, the graph of the functions $\delta_i(x)$ are cones with apexes at the points $(p_i, w_i)$ of $\mathbb{R}^{d+1}$ and aperture angle 45°. In $\mathbb{R}^2$, Apollonius diagrams have hyperbolic bisectors and they belong to the class of abstract Voronoi diagrams (see Figure 4(right)). Therefore, they can be efficiently constructed using an incremental algorithm [KY03]. The CGAL package implements the algorithm described in [KY03]. It supports on-line insertions and deletions, as well as nearest-site queries.

## 3 Constrained triangulations and Voronoi diagrams

The CGAL library also offers constrained triangulations and constrained Delaunay triangulations. Constrained triangulations are tessellations in which

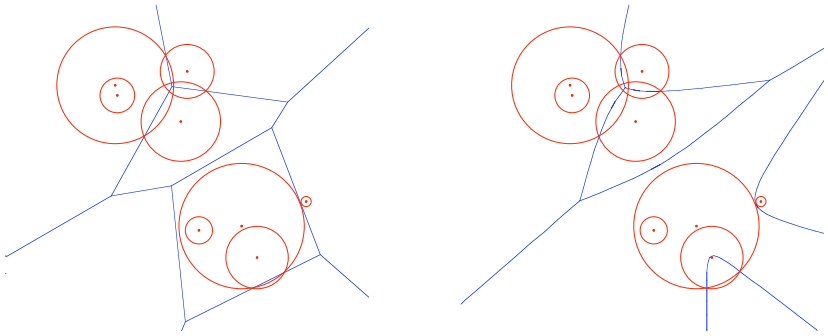


**Fig. 4.** Power diagram (left) and Apollonius diagram (right) for sites corresponding to the same set of circles.

segments of a given set are prescribed to appear as edges. Constrained Delaunay triangulations are constrained triangulations that have the *constrained empty circle property*, a relaxed version of the empty circle property of Delaunay triangulations. The constrained empty circle property considers the set of constraint segments as obstacles to the visibility and requires that any simplex in the triangulation has a circumsphere enclosing no vertex visible from some point in the interior of the simplex. The CGAL packages for constrained triangulation and constrained Delaunay triangulation may handle intersecting, overlapping and partially overlapping input segments without restriction. When constrained segments intersect, the subsegments resulting from the arrangement of input segments are considered as the prescribed edges of the triangulation.

In some applications, it is also required that Voronoi regions do not cross some input constraint segments. Constrained Voronoi diagrams, also called bounded Voronoi diagrams (BVD) [Sei98a] or Voronoi diagrams with barriers [Lin89], have been defined for this purpose. A bounded Voronoi diagram is defined for a set of point sites and a set of constrained segments considered as obstacle to the visibility. The distance $\delta(x, s_i)$ between a point $x$ of $\mathbb{R}^2$ and a site $s_i$ is:

$$\delta(x, s_i) = \begin{cases} ||x - s_i|| & \text{if } x \text{ is visible from } s_i \\ +\infty & \text{otherwise} \end{cases} .$$

Figure 5 illustrates a constrained Delaunay triangulation and its bounded Voronoi diagram.

The notion of face blindness is pivotal for constructing the bounded Voronoi diagram. 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 $\overline{pc} \cap E$

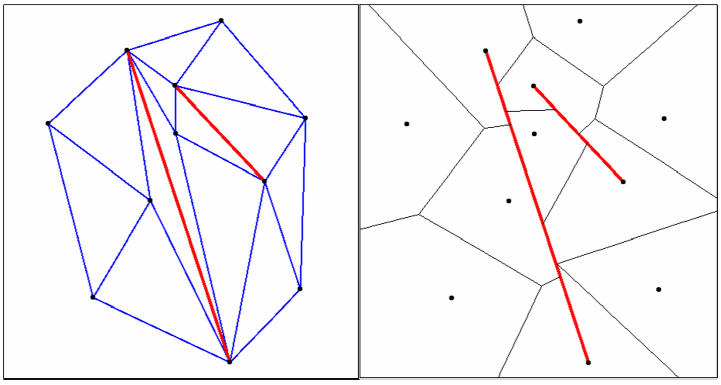


**Fig. 5.** Constrained Delaunay triangulation (left) and its bounded Voronoi diagram (right).

is non-empty. The BVD construction algorithm initially tags all faces of the triangulation as being blind or not blind. It then constructs each cell of the BVD using these tags.

## 4 Software design

### 4.1 Geometric traits and triangulation data structure

One of the most important paradigms used in CGAL to resolve robustness issues arising in geometric code, without dropping computational efficiency, is to establish and maintain a clear-cut separation between the combinatorial aspects of data structures/algorithms, and the numerical computations involved in these algorithms. This distinction shows up in the template parameters of CGAL classes: Triangulation classes have two template parameters called respectively the *triangulation traits* and the *triangulation data structure*.

All numerical issues are encapsulated in the triangulation traits. A model of triangulation traits provides both the geometric primitives, i.e., points, segments, triangles, tetrahedra, etc., handled by the triangulation classes and the operators required on those primitives. In the case of triangulations, the required operators are only predicates testing the sign of polynomial expressions of input points coordinates. Typically, the predicates required to build a Delaunay triangulation in $\mathbb{R}^3$ are the *orientation* predicate which, given four points $p_0, p_1, p_2, p_3$ in $\mathbb{R}^3$, computes the sign of the determinant $orient(p_0, p_1, p_2, p_3)$, and the *in_sphere* predicate which given five points $p_0, p_1, p_2, p_3, p_4$ tests if $p_4$ is enclosed or not by the sphere circumscribed to $p_0, p_1, p_2, p_3$ (see Section 6.1).

The triangulation data structure parameter provides the data structure to store the triangulation. In the default implementation provided by CGAL

for this parameter, the representation of triangulations is mainly based on the vertices and on the full dimensional simplicial cells, i.e., triangles in 2D and tetrahedra in 3D. The triangulation data structure is therefore mainly a container of cells and vertices. The connectivity of the triangulation is encoded in cells and vertices as follows: each cell has a pointer to each of its vertices, and a pointer to each of the adjacent cells; each vertex has a single pointer to one of its incident cells. The faces with intermediate dimensions (i.e. edges in 2D, edges and facets in 3D) are only implicitly encoded through the adjacency relation between cells. Such a representation has the advantage of being compact and to generalize in arbitrary dimension.

CGAL triangulations always encode a triangulation that fills up the whole convex hull of its vertices. Furthermore, they have a fictitious vertex called the *infinite vertex*. The join of the infinite vertex with any facet of the convex hull is considered as a cell of the triangulation. With this convention, the set of faces encoded in a triangulation is a topological sphere and special cases for boundary facets are avoided.

Any Voronoi diagram in the plane with simply connected cells, as it is the case for abstract Voronoi diagrams and in particular for segment Voronoi diagrams and Apollonius diagrams, admit a dual planar graph called the *Delaunay graph*, the nodes of which are in bijection with the Voronoi sites. When there are no degenerate configurations, the Delaunay graph consists of triangular elementary cycles. In the presence of degenerate configurations the Delaunay graph may have more complex faces. Handling the degenerate cases via a perturbation scheme yields a triangulated version of the Delaunay graph by adding a few edges. The CGAL concept for two-dimensional triangulation data structure is able to handle triangulated Delaunay graphs although these planar graphs are not triangulations. There are two aspects in which triangulated Delaunay graphs may differ from a standard triangulation. First, Delaunay graphs may have multiple edges joining the same pair of sites, as well as pairs of adjacent triangles sharing two edges. Secondly, Delaunay graphs may not be embeddable on the plane via straight-line segments. See Figure 6.

### 4.2 The flexibility of CGAL triangulations

As any CGAL data structure, the CGAL triangulations gain their flexibility from their template parameters. Each parameter is associated to a concept, describing the set of types and functions expected from any model of this concept. The concepts are documented.

The geometric traits parameters provide the possibility to run CGAL algorithms or to build CGAL data structures, not only on the geometric primitives offered by the CGAL kernels, but also on user-defined geometric primitives provided that those primitives are included in a model, the so-called *traits class*. For instance, users from GIS may use the flexibility provided by triangulation traits to handle triangulated models of terrains. GIS applications often derive their models of terrains from the Delaunay triangulation of the projections

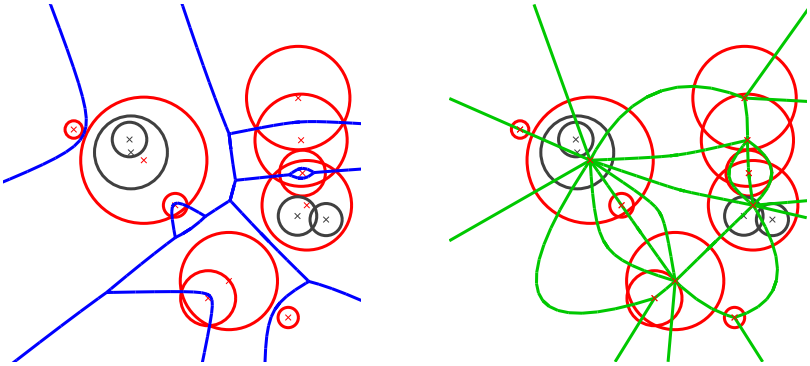


**Fig. 6.** An Apollonius diagram (left) and its triangulated Delaunay graph shown in green (right).

of the terrain vertices on the $xy$-plane. In such a case, the terrain can be represented as a two-dimensional Delaunay triangulation with a triangulation traits providing three-dimensional points and predicates involving only the $x$ and $y$ coordinates of these points.

A great deal of flexibility also arises from the triangulation data structure parameter. In particular the concept for the triangulation data structure includes subconcepts for the models representing the cells and vertices. Therefore the user can easily change or customize the cells and vertices of the triangulation, for instance by adding to these elements colors or any other attributes needed for his/her applications.

## 5 Interfaces

### 5.1 Voronoi interface

The CGAL packages related to 2D Voronoi diagrams actually compute and store the dual Delaunay graph or more precisely its triangulated version. Although a Voronoi diagram and its dual are mathematically equivalent geometric objects, it is typical in applications to prefer one view over the other. To carry the point further, when computing the triangulated version of the Delaunay graph, we *loose* the degenerate-configuration information existent in the input. From the user's perspective, it might be easier to think of and treat the Voronoi diagram directly rather than through its dual representation. To address these issues we provide in CGAL an adaptor that adapts the 2D Delaunay graphs computed by the various CGAL packages to the corresponding 2D Voronoi diagrams seen as two-dimensional arrangements. The objective is to provide a Voronoi diagram interface that has the look and feel of a planar subdivision represented, e.g., via a Double-Connected Edge List

(DCEL) data structure, although the internal representation is a triangulation data structure storing the triangulated Delaunay graph.

The adaptation is straightforward under the non-degeneracy assumption. When degeneracies are present in the input, the triangulated Delaunay graph has artifacts that might be undesirable in the Voronoi diagram view of the computed geometric structure. Suppose for example that we have a set of sites that contains subsets in degenerate position. The dual of the computed triangulated Delaunay graph is a Voronoi diagram in which all vertices have degree 3, and for that purpose we are going to call it a *degree-3 Voronoi diagram*, in order to distinguish it from the true Voronoi diagram of the input sites. A degree-3 Voronoi diagram can have degenerate features, namely Voronoi edges of zero length and/or Voronoi faces of zero area, that do not correspond to the true geometry of the Voronoi diagram.

The way that we treat such issues is by defining an *adaptation policy*. The adaptation policy is responsible for determining which features in the degree-3 Voronoi diagram are to be rejected and which are not. The policy to be used can vary depending on the application or the intended usage of the resulting Voronoi diagram. What we care about is that, firstly, the policy itself is consistent and, secondly, that the adaptation is also done in a consistent manner. The latter is the responsibility of the adaptor we provide, whereas the former is the responsibility of the implementer of a policy. We currently provide two types of adaptation policies, namely the *identity policy* and the *degeneracy removal policy*. The former adapts the triangulated Delaunay graph as is; the resulting Voronoi diagram is in fact a degree-3 Voronoi diagram, which naturally inherits the artifacts present in the triangulated version of the Voronoi diagram. This is the simplest possible adaptation policy and is the most efficient one given the triangulated Delaunay graph: the layer that the adaptor puts on top of the triangulated Delaunay graph is simply a syntactic one, since no geometric computations are performed in addition to those performed by the underlying Delaunay graph. The second type of policy is targeted towards providing to the user a view of the Voronoi diagram that corresponds to the true geometric object. This policy removes the artifacts introduced by either the additional edges introduced in order for the Delaunay graph to consist uniformly of triangles, or the splitting of the input sites into subsites (this is the case for the segment Voronoi diagram, where we split an input segment to three distinct sites, namely its two endpoints and its relative interior).

When the degeneracy removal policy is chosen, we need, within the scope of the adaptor, to identify Voronoi edges of zero length and Voronoi faces of zero area. Depending on whether we want the Voronoi diagram view to be mutable (e.g., allow on-line insertions) or non-mutable (e.g., the input is known beforehand), these tests can be quite expensive when applied to degenerate configurations, which are exactly the ones we want to eliminate. For this purpose we have two possible *adaptation traits*, one that caches the results of the edge/face length/area tests and one that does not. The latter policy is the preferred one under the off-line scenario, whereas the former one

performs better in the on-line scenario. The drawback of the former adaptation policy is that the Delaunay graph adapted needs to support a few additional operations which may not be present.

Currently, in CGAL, we provide adaptation traits and adaptation policies for the 2D Euclidean Voronoi diagram of points, the 2D Apollonius diagram, the 2D Euclidean segment Voronoi diagram and the 2D power diagram.

### 5.2 BGL interface

The BOOST Graph Library (BGL) provides a generic open interface for accessing a graph's structure, while hiding the implementation details [BGL]. This interface is interoperable with many graph algorithms, such that breadth-first search, depth-first search, Kruskal's and Prim's minimum spanning tree algorithms or Dijkstra's shortest paths algorithm. Although the BGL provides some general purpose graph classes conforming to the interface, they are not meant to be exhaustive or exclusive.

In that spirit, and in order to take advantage of the graph algorithms provided along with the BGL, BGL interfaces have been implemented for CGAL's triangulations, polyhedra and arrangements. For Delaunay triangulations in particular, CGAL provides an interface for the Delaunay triangulation that makes it an instance of a BGL undirected graph. The infinite edges of the Delaunay triangulation are by default present, but the user can use BOOST's `filtered_graph` or BGL's property maps in order to make them invisible. CGAL's manual page devoted to BGL provides various examples on how the users can combine the BGL interfaces provided by CGAL with functionality in the BGL.

### 5.3 Interfaces in other languages

CGAL is written in C++. However, to serve communities which may be used to other languages, most of CGAL packages have been interfaced with other languages.

For example, an interface in PYTHON has been realized for many classes in CGAL: 2D and 3D Triangulations and Alpha Shapes, 2D Meshing, Polyhedron, 2D convex hulls, various geometric optimization algorithms, and a large part of CGAL's geometry kernel. It is named `cgal-python` [CGAb]. PYTHON is a modern interpreted language which binds easily to C++ code.

Another example is an interface with SCILAB [SCI], which is a numerical platform similar to MATLAB. The interface is named `CG-LAB`, and is more targeting the engineers' community [CGL]. It provides interfaces to Delaunay triangulations in 2D, 3D and $d$D space; convex hulls in 2D and 3D; Delaunay mesh generator in 2D space; and many others.

Finally, interfaces in JAVA based on JNI (Java Native Interface) are also under development.

### 5.4 Graphical interfaces

Although not the heart of the library, CGAL is also interfaced to some visualization tools, in order to be able to interact easily with geometric data. The main toolkit used in CGAL is QT [QT]. However, some parts of CGAL also use other toolkits for visualization. A notable one is the IPE drawing editor [IPE], toward which many plug-ins interfacing CGAL algorithms have been implemented.

## 6 Robustness issues

The major issue with providing industrial-strength implementations of geometric algorithms is their robustness. There are two main sources of typical non-robustness issues that arise. The first is due to the approximate computations of geometric predicates and constructions using floating-point arithmetic. The second is the proper handling of all particular cases, called degenerate cases, for which we give an illustration below using symbolic perturbations.

We subdivide geometric primitives used by the higher level algorithms, in two categories: the first category is the geometric predicates, which are functions taking some geometric entities as arguments (e.g., points) and returning a result akin to a Boolean or enumerated value. The second category is the geometric constructions, which construct new geometric objects. An example of the former is the classical orientation test for three points in the plane, while an example of the latter is the construction of the circumcenter of three non-collinear points in the plane.

Robustness issues arise when, for example, the approximate computations inside predicates make them return a wrong sign, which later can draw the algorithm to an inconsistent unexpected state. Some examples of failures due to numerical inaccuracies are described in [KMP$^+$04], showing for example cases where the algorithm can loop in localizing a point in a 3D Delaunay triangulation. One general solution which is used throughout CGAL is to rely on the *Exact Geometric Computation paradigm (EGC)*, formulated by Yap and Dubé in [YD94], which states that robustness of the algorithm is ensured if and only if the predicates are evaluated exactly, since the branches in the algorithms are based on the results of predicates.

From the implementation point of view, the exactness of predicates can be achieved using exact multiple precision integer, rational and floating-point numbers. Some wide-spread libraries, like GMP, MPFR and LEDA [GMP, MPF, LED], provide this functionality. CGAL also provides such functionality with its `MP_Float` and `Quotient` classes. In turn, CGAL's predicates and constructions are parameterized by the so-called *number type* that they use. This way it is possible to instantiate the formulas using `double`, `MP_Float`, GMP's `mpq_class`, or other number types that are models of appropriate concepts.

An important aspect here is the study of the algebraic degree of the predicates, the goal being to minimize it, a task that is not at all easy to accomplish. There is, of course, a trade-off to be optimized here, between the number of operations required to compute a predicate and the operations accepted (e.g., usage of square roots can decrease the number of operations required to compute a predicates, but this, on the other hand, necessitates more complicated exact number types). The EGC paradigm, along with *arithmetic* and *geometric filtering* techniques (see Section 7), provides a framework for easy development of geometric software (i.e., development without the need for arithmetic considerations), and at the same time provides robustness through exactness, at a moderate additional cost (see Section 7.1). In the rest of this section we describe briefly how two predicates are computed for two of the Voronoi diagrams provided in CGAL. Our presentation is targeted toward illustrating the variety of the techniques implemented, rather than providing an exhaustive presentation.

### 6.1 Degenerate cases: symbolic perturbations

Computing the 3D Delaunay triangulation of a set of $n$ points involves the computation of two major predicates: the *orientation* predicate and the *in_sphere* predicate.

Let $p_i, p_j, p_k, p_\ell$ be four non-coplanar points in $\mathbb{R}^3$. Point $p_\nu$ has coordinates $(x_\nu, y_\nu, z_\nu)$ for each $\nu$. The orientation predicate for these four points determines whether the fourth point $p_\ell$ lies on the positive or negative halfspace (or on the plane) with respect to the oriented plane defined by the first three points $p_i$, $p_j$ and $p_k$. The orientation predicate is equivalent to determining the sign of the determinant

$$orient(p_i, p_j, p_k, p_\ell) = \begin{vmatrix} 1 & 1 & 1 & 1 \\ x_i & x_j & x_k & x_\ell \\ y_i & y_j & y_k & y_\ell \\ z_i & z_j & z_k & z_\ell \end{vmatrix}.$$

The *in_sphere* predicate involves five points $p_i$, $p_j$, $p_k$, $p_\ell$ and $p_m$, and returns a positive, negative or zero value if $p_m$ lies outside, inside or on the boundary of the ball circumscribing $p_i$, $p_j$, $p_k$ and $p_\ell$. It is well known that the *in_sphere* test can be computed in the following way:

$$in\_sphere(p_i, p_j, p_k, p_\ell, p_m) = \frac{sign(Det(p_i, p_j, p_k, p_\ell, p_m))}{sign(orient(p_i, p_j, p_k, p_\ell))},$$

where

$$Det(p_i, p_j, p_k, p_\ell, p_m) = \begin{vmatrix} 1 & 1 & 1 & 1 & 1 \\ x_i & x_j & x_k & x_\ell & x_m \\ y_i & y_j & y_k & y_\ell & y_m \\ z_i & z_j & z_k & z_\ell & z_m \\ t_i & t_j & t_k & t_\ell & t_m \end{vmatrix},$$

and $t_\nu = x_\nu^2 + y_\nu^2 + z_\nu^2$ for $\nu = i, j, k, \ell, m$.

The $sign(Det(p_i, p_j, p_k, p_\ell, p_m))$ predicate in $\mathbb{R}^3$ can be seen as an orientation predicate in $\mathbb{R}^4$, if each point $p = (x, y, z)$ of $\mathbb{R}^3$ is projected onto a point $\pi(p) = (x, y, z, t)$ on the unit paraboloid $\Pi$ of $\mathbb{R}^4$ with equation $t = x^2+y^2+z^2$ [ES86, DMT92].

While exact predicates allow to detect degenerate configurations, the algorithms must solve special cases explicitly. For Delaunay triangulations, in 2D the situation is degenerate as soon as at least four points are cocircular. In 3D, it is degenerate as soon five points are cospherical. In the sequel, we will use the term cospherical for these two cases.

In fact, in degenerate cases, the Delaunay triangulation is not uniquely defined: any triangulation of the set of cospherical points is a Delaunay triangulation. The computed triangulation must be defined in a non-ambiguous way, to avoid inconsistencies when alternating insertions of points and vertex removals.

This is performed in the CGAL package by the use of a symbolic perturbation, that allows to decide, for each case of cospherical points, which triangulation is chosen [DT03].

The general idea of a symbolic perturbation [EM90, Sei98b] is to replace the original problem by a different one depending on a parameter $\varepsilon$ and such that:

- there exists $\varepsilon_0 > 0$ such that the parameterized problem is in general position for $\varepsilon \in (0, \varepsilon_0]$
- the solution of the parametrized problem has a limit when $\varepsilon$ goes to zero with positive values.
- if the original problem is in general position, the solution of the parameterized problem tends to the solution of the original problem when $\varepsilon$ goes to zero.

The symbolic perturbation method then considers the limit solution of the parametrized problem as the *perturbed solution* of the original problem.

More precisely, let's assume we are facing the *in_sphere* test of five cospherical points $\{p_i, p_j, p_k, p_\ell, p_m\}$. Our perturbation scheme needs to rank somehow the five input points. Then, assuming point $p_i$ has rank $n_i$, the perturbation scheme adds $\varepsilon^{n_i}$ to the fourth coordinate of the projected point $\pi(p_i)$ of $\mathbb{R}^4$. Then, the *orient* test is not perturbed and the outcome of the *in_sphere* test depends on the sign of the perturbed determinant

$$Det_\varepsilon(p_i, p_j, p_k, p_\ell, p_m) = \begin{vmatrix} 1 & 1 & 1 & 1 & 1 \\ x_i & x_j & x_k & x_\ell & x_m \\ y_i & y_j & y_k & y_\ell & y_m \\ z_i & z_j & z_k & z_\ell & z_m \\ t_i+\varepsilon^{n_i} & t_j+\varepsilon^{n_j} & t_k+\varepsilon^{n_k} & t_\ell+\varepsilon^{n_\ell} & t_m+\varepsilon^{n_m} \end{vmatrix}.$$

Developing with respect to the last row yields a polynomial in $\varepsilon$:

$$\begin{aligned} Det_\varepsilon(p_i,p_j,p_k,p_\ell,p_m) = Det(p_i,p_j,p_k,p_\ell,p_m) \\ + orient(p_i,p_j,p_k,p_\ell)\varepsilon^{n_m} - orient(p_i,p_j,p_k,p_m)\varepsilon^{n_\ell} \\ + orient(p_i,p_j,p_\ell,p_m)\varepsilon^{n_k} - orient(p_i,p_k,p_\ell,p_m)\varepsilon^{n_j} \\ + orient(p_j,p_k,p_\ell,p_m)\varepsilon^{n_i}. \end{aligned}$$

It can be shown that the polynomial $Det_\varepsilon$ can never be identically zero. Its sign is the sign of the first non-null coefficient, and depending on this sign, the *in_sphere* predicate will tell whether the point $p_m$ must be considered as being inside or outside the tetrahedron formed by $p_i, p_j, p_k, p_\ell$.

As mentioned above, the perturbation depends only on the way we rank the points. Any ordering of the points is potentially possible. In recent CGAL releases, the lexicographical ordering of the points has been implemented. This ordering has the advantage of being intrinsic to the set of points. It is not invariant under some transformations of the set of points, like e.g., rotations.

The same perturbation scheme can also be used for the case of regular triangulations [DT06] and has been implemented in CGAL.

### 6.2 Predicates of the Apollonius diagram: Algebraic tools to the aid of geometric computations

The predicates involved in the computation of the 2D Apollonius diagram in CGAL are substantially more complicated than the ones for the 2D Voronoi

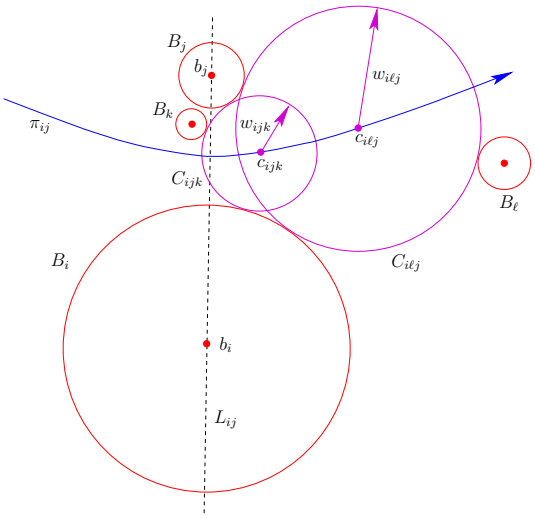



**Fig. 7.** The *Radii_Difference* primitive: if $c_{ijk}$ and $c_{i\ell j}$ lie on the same side of $L_{ij}$, comparing their order on the oriented bisector $\pi_{ij}$ reduces to comparing the radii $w_{ijk}$ and $w_{i\ell j}$.

diagram. It is straightforward to compute the 2D Voronoi diagram using orientation and incircle tests (much like the 3D case in Section 6.1) which are predicates of algebraic degree 3 and 4 respectively, with respect to the input coordinates. In the Apollonius diagram case six predicates are needed, which can be further decomposed to smaller primitives. In the rest of this subsection we briefly describe how one of these primitives is evaluated, namely the *Radii_Difference* primitive (for a detailed description see [KE03] or [EK06]). In particular, we show how, using algebraic tools, we can get a lower algebraic degree for the quantities the sign of which is to be evaluated, as compared to the obvious solution.

The most complicated predicate when computing the Apollonius diagram, is the *Edge_Conflict_Type* predicate: given an edge $\alpha_{ij}^{\mu\nu}$ of the Apollonius diagram, defined by four disks $B_i$, $B_j$, $B_\mu$ and $B_\nu$, and a fifth disk $B_\lambda$, we want to determine if $B_\lambda$ is in conflict with one of the disks centered on $\alpha_{ij}^{\mu\nu}$ (i.e., intersects it) and tangent to both $B_i$ and $B_j$. The *Edge_Conflict_Type* predicate can be resolved easily if we know how to order two points $p$ and $q$ lying on the oriented bisector $\pi_{ij}$ of $B_i$ and $B_j$. $\pi_{ij}$ is either a line (if the radii of $B_i$ and $B_j$ are equal) or a branch of a hyperbola (if the radii of $B_i$ and $B_j$ differ). In the Apollonius diagram context, $p$ and $q$ are defined via two other disks $B_k$ and $B_\ell$, $k \neq \ell$, ($B_k$ and $B_\ell$ can be any of $B_\mu$, $B_\nu$ and $B_\lambda$), see Figure 7. In particular, $p$ is the center $c_{ijk}$ of the ccw-oriented circle $C_{ijk}$ tritangent to $B_i$, $B_j$ and $B_k$ (touching them in that order) and $q$ is the center $c_{i\ell j}$ of the ccw-oriented circle $C_{i\ell j}$ tritangent to $B_i$, $B_\ell$ and $B_j$ (again, touching them in that order).

Let $L_{ij}$ be the line passing through the centers of $B_i$ and $B_j$; if $\pi_{ij}$ is a hyperbolic branch, then $L_{ij}$ is its line of symmetry. If $c_{ijk}$ and $c_{i\ell j}$ lie on different sides of $L_{ij}$, we can determine their order on $\pi_{ij}$ by means of the orientation tests $orient(b_i, b_j, c_{ijk})$ and $orient(b_i, b_j, c_{i\ell j})$, where $b_i$ and $b_j$ are the centers of $B_i$ and $B_j$, respectively. This orientation test is of algebraic degree 14 [EK06]. If $c_{ijk}$ and $c_{i\ell j}$ lie on the same side of $L_{ij}$, determining their order on $\pi_{ij}$ is equivalent to asking for the sign of the difference $w_{ijk} - w_{i\ell j}$, where $w_{ijk}$ and $w_{i\ell j}$ are the radii of the Voronoi circles $C_{ijk}$ and $C_{i\ell j}$ respectively, hence the name for the *Radii_Difference* primitive. The radii $w_{ijk}$ and $w_{i\ell k}$ can be written as:

$$w_{ijk} = \rho_1 - r_i, \qquad w_{i\ell j} = \rho_2 - r_i,$$

where $r_i$ is the radius of $B_i$ and, for $\tau = 1, 2$, $\rho_\tau > 0$, is a real root of an equation of the form

$$f_\tau(x) := \alpha_\tau x_\tau^2 - 2\beta_\tau x_\tau + \gamma_\tau = 0.$$

Clearly, it suffices to compute the sign of the difference $t := \rho_1 - \rho_2$.

The algebraic degrees of $\alpha_\tau, \beta_\tau$ and $\gamma_\tau$ are 4, 5 and 6, respectively. We assume for now that $\alpha_1\alpha_2 > 0$; the case $\alpha_1\alpha_2 = 0$ is discussed towards the end of this subsection. Let $\Delta_\tau = \beta_\tau^2 - \alpha_\tau\gamma_\tau$, be the discriminant of the equation

$f_\tau(x) = 0$, and let $J = \alpha_1\beta_2 - \alpha_2\beta_1$. Then

$$t = \frac{\beta_1 + \sqrt{\Delta_1}}{\alpha_1} - \frac{\beta_2 + \sqrt{\Delta_2}}{\alpha_2} = \frac{-J + \alpha_2\sqrt{\Delta_1} - \alpha_1\sqrt{\Delta_2}}{\alpha_1\alpha_2}.$$

Since $\alpha_1\alpha_2 > 0$, in order to determine the sign of $t$, it suffices to determine the sign of the numerator of $t$. The numerator of $t$ is a quantity of the form $Q_1 := X_0 + X_1\sqrt{Y_1} + X_2\sqrt{Y_2}$, where, in general, $Y_1 \neq Y_2$ and $Y_1, Y_2 > 0$. The algebraic degrees of $X_0$, $X_1$, $X_2$, $Y_1$ and $Y_2$ are 11, 8, 8, 6 and 6, respectively. In order to determine the sign of $Q_1$, we need to determine, in the worst case, the sign of the quantity $(X_0^2 + X_1^2Y_1 - X_2^2Y_2)^2 - 4X_0^2X_1^2Y_2$, which is a degree 36 quantity in the input.

Let us know consider a slightly different approach. Substitute the value of $\rho_1$ in terms of $t$ and $\rho_2$ in $f_1(x) = 0$. We thus get:

$$\alpha_1(t + \rho_2)^2 - 2\beta_1(t + \rho_2) + \gamma_1 = 0,$$

which can be rewritten as a quadratic polynomial in terms of $t$:

$$\bar{f}_1(t) := \bar{\alpha}_1 t^2 + \bar{\beta}_1 t + \bar{\gamma}_1 = 0,$$

where $\bar{\alpha}_1 = \alpha_1$, $\bar{\beta}_1 = f_1'(\rho_2)$ and $\bar{\gamma}_1 = f_1(\rho_2)$. Determining the sign of $t$ reduces to determining the sign of the appropriate root of $\bar{f}_1(t)$. This can be done by applying Descartes' rule of signs to $\bar{f}_1(t)$, which calls for computing the signs of $\bar{\beta}_1$ and $\bar{\gamma}_1$. Both $\bar{\beta}_1$ and $\bar{\gamma}_1$ are expressions of the form $(X_0 + X_1\sqrt{Y})/Z$, where $Z > 0$, and their signs can be evaluated using the following relation:

$$\mathit{sign}(X_0 + X_1\sqrt{Y}) = \mathit{sign}(\mathit{sign}(X_0)X_0^2 + \mathit{sign}(X_1)X_1^2Y).$$

The degrees of $X_0$, $X_1$ and $Y$ are 9, 6 and 6 for $\bar{\beta}_1$, and 14, 11 and 6 for $\bar{\gamma}_1$, respectively. Hence, the highest algebraic degree involved in the evaluation of the signs of the roots of $\bar{f}_1(t)$ is 28, already an improvement over the algebraic expression of degree 36 we found above.

We can further decrease the maximum algebraic degree of the expressions to be evaluated by employing Sturm sequences. Let $(P_i(x))_{0 \leq i \leq M}$, be the Sturm sequence of $f_1(x)$ and $f_1'(x)f_2(x)$. In the absence of (algebraic) degeneracies $M = 4$, i.e., the Sturm sequence consists of five polynomials. In this case we can then determine the sign of $t$ by means of the signs of the quantities $J$, $P_3(\infty)$ (this is $P_3(x)/x^2$ evaluated at $+\infty$) and $P_4$ ($P_4(x)$ is a constant polynomial), where

$$P_3(\infty) = -(\alpha_1 K + 2\alpha_2\Delta_1), \qquad P_4 = (P_3(\infty))^2(4JJ' - G^2),$$

and

$$J' = \beta_1\gamma_2 - \beta_2\gamma_1, \qquad G = \alpha_1\gamma_2 - \alpha_2\gamma_1, \qquad K = \alpha_1\gamma_2 + \alpha_2\gamma_1 - 2\beta_1\beta_2.$$

The algebraic degrees of $J$ and $P_3(\infty)$ are 9 and 14, respectively. The algebraic degree of the second factor of $P_4$ is 20. The expression $4JJ' - G^2$ can be

further factorized to two factors, one of degree 8 and one of degree 12, the latter being a polynomial of 305 monomials: the computation of these factors is very expensive.

It is plausible, however, that the Sturm sequence of $f_1(x)$ and $f_1'(x)f_2(x)$ contains less than five terms. In fact the only possibility is that $(P_i(x))_{0\leq i\leq M}$ contains four instead of five terms, i.e., $M = 3$, and this happens if $P_3(\infty) = -(\alpha_1 K + 2\alpha_2 \Delta_1) = 0$. In this case $P_3(x)$ is the last term of the Sturm sequence, and is a constant polynomial equal to $P_3 = P_3(x) = 2\alpha_1 \Delta_1 J'$. Hence the maximum algebraic degree of the quantities encountered in this case is 11.

In our discussion above it is assumed that $\alpha_1\alpha_2 > 0$. If $\alpha_1 = \alpha_2 = 0$, we only need to test the sign of the quantity $J'$ which is of degree 11. If $\alpha_1 = 0 < \alpha_2$ (the case $\alpha_2 = 0 < \alpha_1$ is symmetric), the Sturm sequence of $f_1(x)$ and $f_1'(x)f_2(x)$ ends with $P_3 = P_3(x) = 4\beta_1^2\alpha_2 f_2(\gamma_1/(2\beta_1))$. The quantity $4\beta_1^2 f_2(\gamma_1/(2\beta_1))$ is of degree 16. It is interesting to note that this quantity arises in a geometrically degenerate configuration: it corresponds to three disks having a common tangent line, or equivalently to three disks the Apollonius circle of which has infinite radius.

The degree 16 we just mentioned is the maximum algebraic degree encountered in the evaluation of the predicates of the Apollonius diagram using the Sturm sequence methodology. Recall that the degree 12 factor of the quantity $4JJ' - G^2$ consists of 305 monomials in the input quantities. As a result its computation involves a dramatically larger number of operations, as compared to computing the quantity $4JJ' - G^2$ directly. For efficiency reasons, in the current CGAL implementation we do not make use of this factorization, that is we evaluate quantities of degree up to 20.

## 7 Efficiency issues

In this section we detail several key aspects which need to be considered to obtain truly efficient implementations of the algorithms.

### 7.1 Arithmetic filtering

As previously mentioned, a critical part of the robustness of the implementation is achieved thanks to the exactness of the geometric predicates and constructions. We are now focusing on the details of their efficient implementation.

Exact predicates can be easily implemented using exact multiple precision integer, rational and floating-point numbers. The main issue with multiple precision number types is their efficiency. Indeed, compared to hardware assisted floating-point types, like `double`, for which arithmetic operations basically take a few cycles, multiple precision numbers require a memory storage which grows with the accumulation of computations (which can be exponential with rational numbers), and the cost of each operation is at least linear in the size

of the operands. In practice, it is reported, e.g., in [DP03], that the impact of naively using these multiple precision number types, while it solves all numerical robustness issues, incurs a penalty of 2 orders of magnitude in terms of efficiency, compared to using `double`s.

People then observed that, in the case of predicates, only the sign of an expression is returned. This means that only the exact sign is required to be computed, not the full bit-wise representation of an expression. Remarks have been made that, given that most of the time floating-point arithmetic returns the correct sign, it should be enough to be able to detect when such floating-point computations have a chance to lead to an incorrect sign. Then, only in these cases, rely on the exact multiple precision costly number types. This scheme is known as *arithmetic filtering*, since it quickly filters out the easy cases using arithmetic tools, and has been developed in a number of ways [FV96, She97, BFS01, DP03, BBP01]. Some of these are implemented in CGAL, such as dynamic filters based on interval arithmetic.

*Interval arithmetic* is used in CGAL as a mean to bound an exact value inside an interval whose bounds are `double`s. This standard tool is relatively fast to compute with, and it is used inside CGAL's `Filtered_predicate` class which allows to easily create filtered predicates following the scheme: first, we evaluate the predicate using interval arithmetic; then, if the sign cannot be concluded because 0 lies in the resulting interval, we recompute the predicate using an exact number type [BBP01]. CGAL does this using template parameterization, and instantiating successively with different number types. This ensures optimal efficiency and ease of use. This method of filtering is quite general, since all arithmetic operations are possible with intervals. The resulting code is typically 3 to 10 times slower than using uncertified `double`s, depending on the algorithm (the share of combinatorial operations versus numerical operations can vary) and the data sets (more degeneracies means slower).

Besides this generic method based on intervals, some critical predicates such as those used in 2D and 3D Delaunay triangulations, benefit from additional layers of filtering, named static and semi-static filtering. The idea is to study the sequence of arithmetic operations used in a predicate, and to derive an error bound on the resulting expression we need the sign of, from bounds on the input arguments. Such a method can be much faster than interval arithmetic, since, basically, the difference between this and pure floating-point is that some bounds are computed at the beginning, which can be somehow cached, and the comparison with zero is replaced with a comparison with an error bound. In practice, this allows to reach close to optimal efficiency, as the slowdown compared to uncertified `double`s tends to be about 25% [DP03]. These methods are however heavier to set up and error prone, which leads to formal proving of some of the methods implemented in CGAL [MP05].

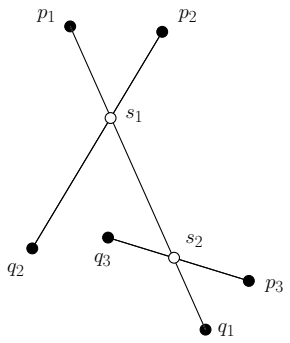



**Fig. 8.** Segment site representation. The point $s_1$ is represented by the four points $p_1$, $q_1$, $p_2$ and $q_2$. The subsegment $p_1s_1$ is represented by the points $p_1$, $q_1$, $p_2$, $q_2$ and a Boolean which is set to *true* to indicate that the first endpoint in not an intersection point. The subsegment $s_1s_2$ is represented by the six points: $p_1$, $q_1$, $p_2$, $q_2$, $p_3$ and $q_3$. The remaining (non-input) points and subsegments in the figure are represented similarly.

### 7.2 Geometric filtering

The segment Voronoi diagram computation poses additional difficulties when we allow the input segments to intersect arbitrarily [Kar04]. If we represent the subsegments via the Cartesian coordinates of their endpoints, we may end up with an exponential blow up on the bit complexity of the Cartesian coordinates of intersection points. Because of this blow up we are unable to bound the algebraic degree of the predicates evaluated during execution of the algorithm, which essentially invalidates any attempt for arithmetic filtering (the arithmetic filters will almost always fail due to the high bit complexity of the geometric objects involved in the predicate evaluation).

To overcome this blow up in the CGAL package for computing the segment Voronoi diagram, we take special care on how segments are represented. More precisely, intersection points are represented by four points, namely the endpoints of the two input segments that define them, whereas segments that have intersection points as endpoints are represented either by four points and a Boolean or by six points, depending on whether only one or both endpoints of the segment are intersection points (see Figure 8). This representation allows us:

1. to have a closed-form representation of the sites defining the Voronoi diagram, thus avoiding cascading,
2. to avoid an exponential blow up on the bit complexity of the coordinates of the intersection points,
3. to use *geometric filtering*, discussed below.

Geometric filtering amounts to performing simple geometric tests exploiting the representation and construction history of the geometric data, as well as the geometric structure inherent in the problem, in order to evaluate predicates in seemingly degenerate configurations. Geometric filtering can be seen as a preprocessing step before performing arithmetic filtering. It can help by eliminating situations in which the arithmetic filter will fail, thus decreasing the number of times we need to evaluate a predicate using exact arithmetic.

Let us consider a simple, yet very illustrative, example of geometric filtering. Suppose we want to determine if two intersection points in the segment Voronoi diagram are identical. We assume here that the input points are represented by `double`s via their Cartesian coordinates. Input segments are represented by their two endpoints, whereas intersection points and the subsegments are represented as described above. In order to determine if the two non-input points are identical we need to compute their Cartesian coordinates and compare them. If the two points are geometrically identical, the answer to our question using double arithmetic may be uncertain (due to numerical errors), in which case we will have to reside to the more expensive exact computation. Instead, before testing the coordinates for equality, we can use the representation of the points to potentially answer the question. More specifically, and this is the geometric filtering part of the computation, we can first test if the defining segments of the two points are the same. If they are not, then we proceed to comparing their coordinates as usual. Testing the defining segments for equality does not involve any arithmetic operations on the input, but rather only comparisons on `double`s.

Geometric filtering techniques, such as the one presented in the paragraph above, is used in all predicate evaluations in the CGAL segment Voronoi diagram package.

### 7.3 Point location

Point location is crucial for the efficiency of the package. It is an important functionality in its own for the user, but it is also a major ingredient of the incremental construction of the triangulation. The CGAL 2D and 3D Delaunay triangulation packages offer two main strategies: a walk though the triangulation, and the use of a hierarchical structure.

*Visibility walk.*

This simple strategy is in fact quite popular for Delaunay triangulations. Let's explain it in 2D. Assume we want to locate the point $p$. We start the walk at an arbitrary triangle. Then, at each step, we move from a triangle $t$ not containing $p$, to a neighbor of $t$ sharing with $t$ an edge $e$ such that the line supporting $e$ separates $t$ from $p$. If there are two such edges, then we move to any of these two neighbors. This generalizes to higher dimensions, replacing the edges by facets.

In the case of the Delaunay triangulation, this walk has been proved to be acyclic and to reach the correct triangle [Ede90, DFNP91]. In the case of an arbitrary triangulation, the walk may loop. A variant, however, the stochastic walk, has been proved to terminate [DPT02]. This variant consists in choosing at random the neighboring triangle of $t$ to proceed to, whenever there are more than one edge of $t$ whose supporting line separates $t$ from $p$.

*Delaunay hierarchy.*

Even if the walk obtains good running times, a more sophisticated strategy is proposed for large data sets. The point location uses an additional hierarchical data structure [Dev02]: the last level of the structure is the Delaunay triangulation $\mathrm{DT}(\mathcal{S})$, and each level $i$ contains the Delaunay triangulation $\mathrm{DT}_i$ of a subset of sites $\mathcal{S}_i \subset \mathcal{S}$.

The subsets of sites $\mathcal{S}_i$ form a decreasing sequence of random subsets of $\mathcal{S}$. The number $k$ of levels is not fixed; for each point, random trials decide its level, and the point with the highest level determines $k$.

A point $p \in \mathcal{S}$ such that $p \in \mathcal{S}_i \subseteq \ldots \subseteq \mathcal{S}_0 = \mathcal{S}$ and $p \notin \mathcal{S}_{i+1}$ has a link to a Delaunay triangle of $\mathrm{DT}_j$ incident to $p$ for all $j$ between 0 and $i$.

To locate a query point $q$, we start at a known vertex $v_{k+1}$ of the highest level $k$. Using the basic location strategy in $\mathrm{DT}_k$ we search for $v_k$, the vertex of $\mathrm{DT}_k$ nearest to $q$. Since $v_k$ is also a vertex of $\mathrm{DT}_{k-1}$, we then start from $v_k$ to search for $v_{k-1}$, the nearest neighbor of $q$ in $\mathrm{DT}_{k-1}$. The search is continued descending the different levels. At each level $i$, the nearest vertex $v_i$ of $q$ in $\mathrm{DT}_i$ is determined.

It has been shown that the construction of the Delaunay hierarchy of a set of $n$ points can be done in expected time $O(n \log n)$ in 2D and $O(n^2)$ in 3D. The space is linear in the size of the triangulation, with a constant factor very close to 1. The walk at each level performs a constant expected number of steps and the expected query time is $O(\log n)$. The expectation is on the randomized sampling and the order of insertion, with no assumption on the point distribution.

In practice, instead of looking for the true nearest vertex $v_i$ of $q$ in $\mathrm{DT}_i$, the point $q$ is located in $\mathrm{DT}_i$ using the basic location strategy, and the vertex of the triangle containing $q$ that is closest to $q$ is used in place of $v_i$. This approximation invalidates the theoretical complexity analysis, but performs very well in practice.

*Voronoi hierarchy.*

A variant of the Delaunay hierarchy is the *Voronoi hierarchy* [KY03]. Conceptually, instead of thinking of the hierarchy as consisting of a series of Delaunay triangulations, we understand the hierarchy as a series of Voronoi diagrams. The Voronoi hierarchy is a more convenient hierarchical data structure when we want to compute the Voronoi diagram for non-point objects. In fact, it is applicable to quite a few types of abstract Voronoi diagrams, thus making it

more general than the Delaunay hierarchy at a moderate additional cost in query time.

In the Delaunay hierarchy the location of the nearest neighbor $v_i$ of the query point $q$ at level $i$ is done by walking from $v_{i+1}$ (the nearest neighbor of $q$ at level $i+1$) along the line connecting $v_{i+1}$ and $q$. In the Voronoi hierarchy we move from $v_{i+1}$ to $v_i$ as follows. Let $V_j$ be the cell of the Voronoi diagram currently visited at level $i$ (initially, $V_j$ is the cell $V_{i+1}$ of $v_{i+1}$). We look at the neighbors of $v_j$: if there exists a neighbor $v_m$ such that $\delta(q, v_m) < \delta(q, v_j)$, we visit $V_m$ and continue as before; otherwise, the distance from $q$ to all neighbors of $v_j$ is larger than the distance $\delta(q, v_j)$, i.e., $v_j$ is the nearest neighbor $v_i$ of $q$ at level $i$.

Moving from a cell $V_j$ to a cell $V_m$ takes linear time in the size of the cell. Finding the next cell to go to can be sped up by using balanced binary trees attached to the Voronoi cells: using these balanced binary trees we can find the next cell to go to in time $O(\log n)$. By means of these trees the nearest-neighbor location query can be performed in $O(\log^2 n)$ expected time, where $n$ is the number of sites already inserted in the Voronoi diagram. In fact, we can compute the two-dimensional Euclidean Voronoi diagram for a set of $n$ (possibly intersecting) disks, disjoint line segments or disjoint convex objects, in $O(n \log^2 n)$ expected time and $O(n)$ expected space [KY03]. For a set of $n$ possibly intersecting line segments, their Voronoi diagram can be computed in $O((m+n)\log n)$ expected time and $O(m+n)$ expected space, where $m = O(n^2)$ is the complexity of the arrangement of the $n$ line segments [Kar04]. As in the case of the Delaunay hierarchy, the expectation is on the randomized sampling for determining the maximum level of the hierarchy in which a site is inserted, and the order of insertion.

In reality, maintaining a balanced binary tree per cell, in a incremental or dynamic setting can be quite complicated. The current CGAL implementation of the segment Voronoi diagram and the Apollonius diagram does not maintain these trees. Although the theoretical results are no longer valid, the Voronoi hierarchy works very well in practice.

### 7.4 Spatial sorting

Sometimes, knowledge about a triangulation is not needed before all input points (or a large set of points) have been inserted to the triangulation. In this case, a clever ordering of the points may improve efficiency in building the triangulation. If the points are sorted along a space-filling curve, a simplex containing the previously inserted point will be a very good start for the visibility walk to locate the new point. Furthermore, part of the data structure that will be accessed during each insertion is likely to have been recently accessed. In practice, ordering points along a space-filling curve prior to insertion induces constant time visibility walk for the location of almost all new sites and, much fewer cache misses.

CGAL implements an even more subtle ordering. A set of $n$ points is randomly subdivided in $\Theta(\log n)$ *rounds*, where the $i^{\text{th}}$ round contains $\Theta(\alpha^i)$ points, $\alpha$ being a constant. This subdivision is done in such a way that the probability of a point participating in round $i$ is proportional to size of this round. Each round is then sorted along a Hilbert curve, using a quicksort-like algorithm. This random subdivision in rounds does not change the expected time complexity for Delaunay triangulations, while giving very good results in practice. See [ACR03] for a comprehensive analysis. We have observed improvements of up to a factor of 2 in 3D (4 in 2D) over a random insertion order.

Spatial sorting can also be seen as a first step toward supporting better streaming representations of very large triangulations as in [ILSS06].

### 7.5 Further scalability of CGAL algorithms

Besides all these improvements which are applicable to most CGAL algorithms, it is also possible to introduce some parallelization in the algorithms. This aspect is more and more important today, as even all last generation laptop computers are provided with multi-core processors, and the trend in future hardware is strongly following this direction. Following the practical work described in [BBK06], we aim at having good support for parallelization in the key CGAL algorithms for triangulations and meshing. Without going into a lot of details, this is performed by carefully splitting the work among all available processors, and managing proper locking of the data structures.

There are other techniques which are important to allow the handling of huge data sets. One such technique is the compact representation of the triangulation in memory. CGAL already provides a relatively good trade-off in this direction thanks to its `Compact_container` class, a generic STL-like container offering the flexibility of a `std::list` (constant time insertions and deletions), together with a memory overhead almost as low as that of an array. More can be done in terms of compact representation and local compression [CDM06], most probably by offering the user a way to reach the best balance between raw speed and memory usage for his/her application.

## 8 Tessellations at work

### 8.1 Alpha-shapes

Assuming a set of points $\mathcal{S}$ in $\mathbb{R}^d$ and a real parameter $\alpha$, the alpha-complex $\mathrm{AC}(\mathcal{S}, \alpha)$ is a subcomplex of the Delaunay triangulation $\mathrm{DT}(\mathcal{S})$ whose definition is based on the radii of empty circumspheres. A circumsphere of a simplex in $\mathrm{DT}(\mathcal{S})$ is a sphere passing through the vertices of this simplex. A circumsphere is said to be empty if it encloses no vertex of the triangulation $\mathrm{DT}(\mathcal{S})$. The characteristic property of Delaunay simplexes is to have empty

circumspheres. The alpha-complex $\mathrm{AC}(\mathcal{S}, \alpha)$ includes all Delaunay simplexes which have an empty circumsphere with square radius not bigger than $\alpha$. The alpha-shape is just the domain covered by simplexes of the alpha-complex. Weighted alpha-complexes and weighted alpha-shapes are defined similarly on top of regular triangulations. Recall that the sites associated to the vertices of a regular triangulation are weighted points which can be considered as spheres as well. The power product of two spheres $\sigma_1$ and $\sigma_2$ with centers $c_1$ and $c_2$ and radii $r_1$ and $r_2$ is defined as:

$$\pi(\sigma_1, \sigma_2) = ||c_1 c_2||^2 - r_1^2 - r_2^2.$$

Two spheres are said to be orthogonal when their power product is zero. Instead of considering the circumspheres of simplexes, the definition of weighted alpha-shapes consider the spheres orthogonal to the sites associated with the simplex vertices. Such an orthogonal sphere is said to be empty when it has a positive power product with any site in the triangulation. The weighted alpha-complex $\mathrm{WAC}(\mathcal{S}, \alpha)$ is the subcomplex of the regular triangulation of $\mathrm{RT}(\mathcal{S})$ including all the simplexes that have an empty orthogonal sphere with square radius not bigger than $\alpha$.

Alpha-shapes and weighted alpha-shapes have been introduced by Edelsbrunner and Mücke [EM94]. Alpha-shapes have also been generalized by Kim et al. [KSK+06] into beta-shapes. Beta-shapes are similar to weighted alpha-shapes, but based on the Apollonius diagram of weighted points instead of their power diagram. These notions are widely used in the context of shape reconstruction from point clouds and find natural applications in computational biology and molecular modeling. In the CGAL library, alpha-shapes are implemented as a data structure attached to the Delaunay or regular triangulations. This data structure allows, for any value of the parameter $\alpha$, (1) to classify any face as included or not in the alpha-complex or (2) to retrieve the whole alpha-complex. It also provides a filtration of the triangulation, where faces are output according to the order they are included in the alpha-complex, as the parameter $\alpha$ is grown from $-\infty$ to $+\infty$.

### 8.2 Meshing and reconstruction

As they encode distance relationships, Voronoi diagrams, Delaunay and regular triangulations are basic tools in the field of meshing and reconstruction. The current tools offered in CGAL for mesh generation are all based upon the Delaunay refinement paradigm pioneered by Chew [Che89] and Ruppert [Rup95].

CGAL currently offers a triangle mesh generator for two-dimensional domains with constraints defined as a planar straight-line graph (PSLG), see Figure 9. CGAL also offers a module to generate surface triangle meshes and a three-dimensional mesh generator able to handle domains bounded by smooth curved surfaces or piecewise smooth curved surfaces. To approximate curved

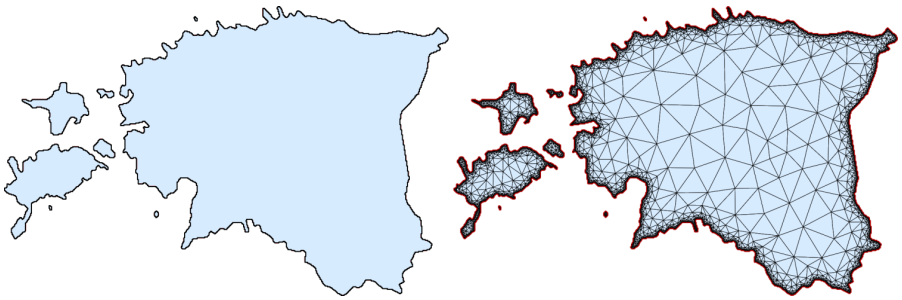


**Fig. 9.** Triangle mesh generation by Delaunay refinement. Left: Map of Estonia provided as input PSLG (7K segments). Right: Mesh generated by adding 7.5K Steiner points (no sizing constraints were provided as input).

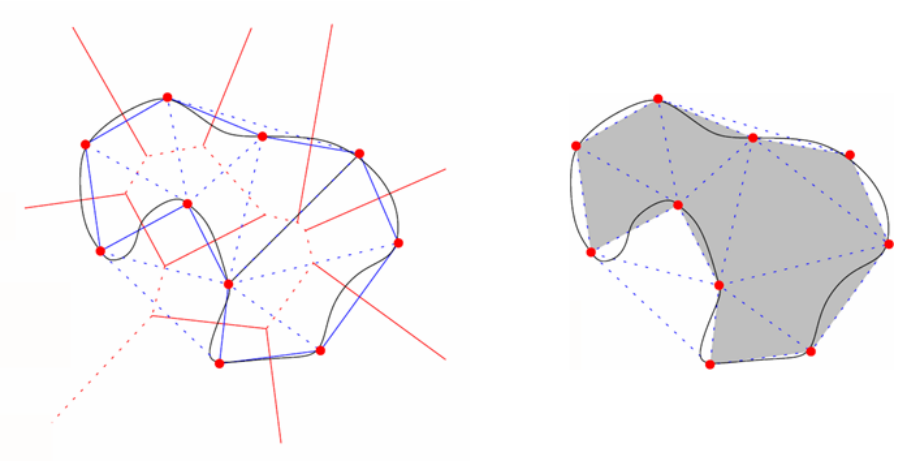


**Fig. 10.** Delaunay triangulation restricted to the input domain boundary (blue solid edges shown on the left) or the input domain (gray triangles shown on the right).

surfaces, the meshing tool uses the notion of restricted Delaunay triangulations. Given a set $\mathcal{S}$ of points and a domain $\Omega$, the Delaunay triangulation of $\mathcal{S}$ restricted to $\Omega$, $\mathrm{DT}_\Omega(\mathcal{S})$, is the subcomplex of $\mathrm{DT}(\mathcal{S})$ formed by the Delaunay faces whose dual Voronoi faces intersect $\Omega$, see Figure 10.

Recent results on the theory of surface sampling [AB99, ACDL00, CDRR04, BO05] have shown that a smooth surface $\Sigma$ is well approximated by the restricted Delaunay triangulation $\mathrm{DT}_\Sigma(\mathcal{S})$ of a dense enough sampling $\mathcal{S}$ on $\Sigma$. In particular, the restricted Delaunay triangulation is a two-dimensional manifold mesh, homeomorphic and even ambient isotopic to the surface. The Hausdorff distance between the surface $\Sigma$ and the restricted Delaunay triangulation $\mathrm{DT}_\Sigma(\mathcal{S})$ goes to zero with increasing density. The restricted Delaunay

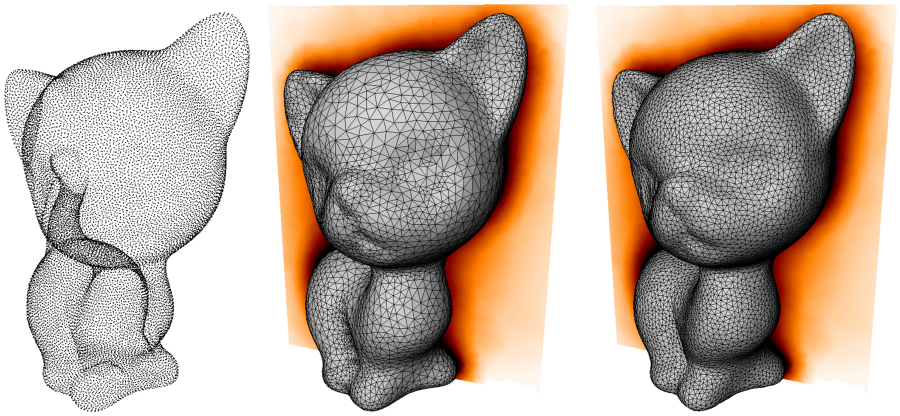


**Fig. 11.** Surface meshing for surface reconstruction from point clouds. Left: Input point set. Middle and Right: Implicit functions derived from the point set, and surface triangle meshes generated by Delaunay refinement and filtering for two different mesh sizing parameters.

triangulation of a dense sample also yields a good approximation for normals, area, curvature and other differential properties. The required density of the sampling is relative to the so-called local feature size $lfs(x)$, defined for each point $x$ of the surface $\Sigma$ as the distance from $x$ to the medial axis of $\mathbb{R}^3 \setminus \Sigma$. The medial axis of $\mathbb{R}^3 \setminus \Sigma$ is the locus of centers of balls included in $\mathbb{R}^3 \setminus \Sigma$ that are maximal with respect to inclusion.

The surface meshing tool of CGAL uses the notion of restricted Delaunay triangulation both to extract the approximating surface mesh from the Delaunay triangulation of the current sampling and to guide the refinement process, yielding a dense enough sampling.

A noticeable feature of this surface and volume mesh generator is that it only requires to know the surface through a geometric oracle which: (1) tells on which side of the surface lies a query point, (2) detects if a query line segment intersects the surface or not, and (3) computes the intersection points if any. Such an oracle is given to the mesh generator as a template parameter. This mechanism provides a mesh generator flexible enough to be applied in a wide variety of situations, ranging from domains defined by implicit surfaces to domains defined by level-sets in 3D gray-scaled images through point-set surfaces. Figure 11 illustrates the output of the surface mesh generator in a surface reconstruction application. An implicit function approximating the distance to the surface is first derived from the input point cloud, and the zero level set of this function is then tessellated. In Figure 12 the output of the CGAL surface mesh generator is compared with the output of the marching cubes algorithm applied to an octree.

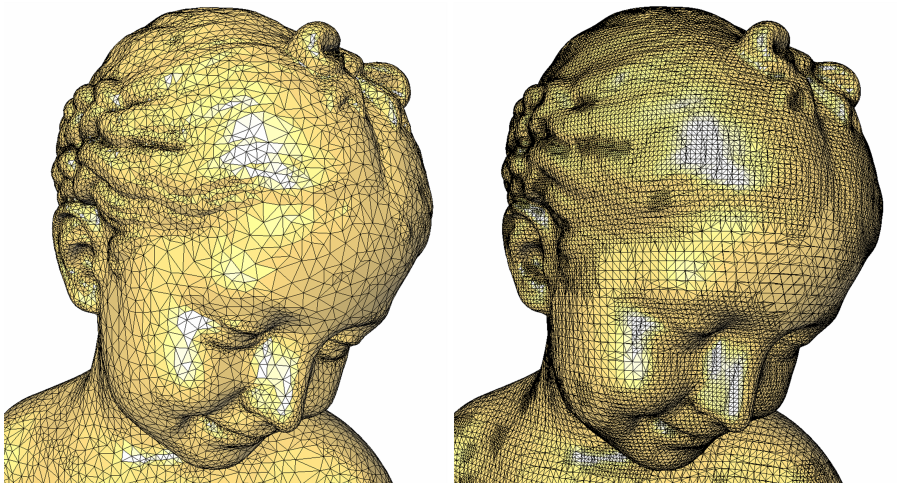


**Fig. 12.** Delaunay-based surface meshing vs. marching cubes in an octree. All triangles of the mesh shown on the left are better shaped compared to the triangles of the mesh to the right, and the mesh contains fewer triangles for the same approximation accuracy.

## 9 On-going and future work

### 9.1 Periodic triangulations and meshes

The scope of computational geometry research has been mostly limited to manipulation of geometric elements in the Euclidean space $\mathbb{R}^d$. Other geometric spaces have hardly, if at all, been considered in the computational geometry literature so far. However, they are relevant to needs in applied domains. In particular, periodic spaces such as the torus and the cylinder in 2D or 3D are the natural spaces in a variety of situations, especially for simulations requiring periodic boundary conditions as it is often the case in fluid dynamics or astronomy.

The standard workaround consists in triangulating an elementary domain, and duplicating some fraction of the points outside that domain to simulate periodicity. This is at the same time inelegant and inefficient: it increases the size of the input set of points to be triangulated, which results in a significant overhead both in time and memory.

Some work was performed to solve theoretical problems to adapt the algorithms initially designed for $\mathbb{R}^3$ to the case of the flat torus $\mathbb{R}^3/\mathcal{Z}^3$ [CT08, CT09]. The package recently released in CGAL allows users to triangulate directly the points in the periodic space and avoid any duplication whenever possible. See Figures 13 and 14 for illustrations.

More work is in progress to compute meshes of periodic surfaces.

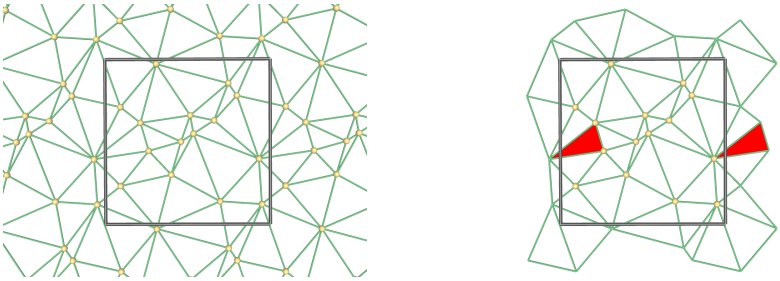


**Fig. 13.** Periodic triangulation in $\mathbb{R}^2$.

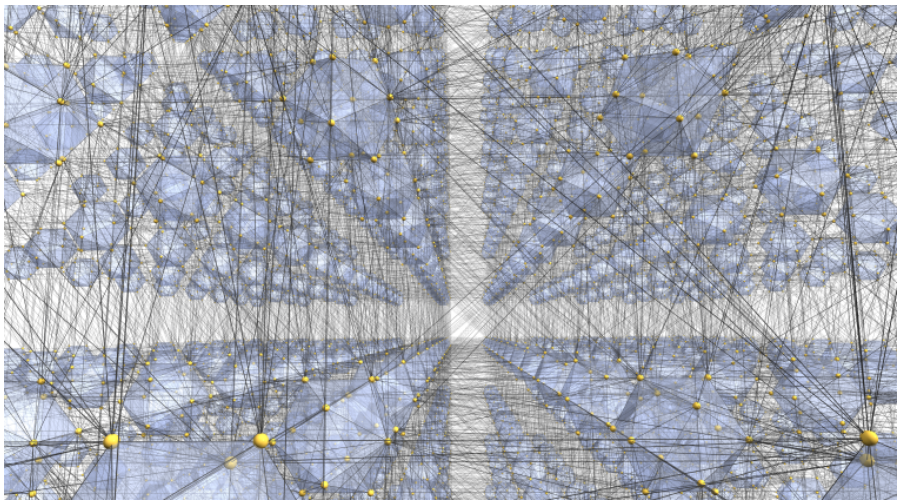


**Fig. 14.** Periodic triangulation in $\mathbb{R}^3$.

### 9.2 Optimized meshing

When high quality meshes are sought after, one approach is to resort to an optimization procedure. Two questions now arise: Which criterion should we optimize? Which degrees of freedom should we use? As the degrees of freedom are both continuous and discrete (vertex positions and mesh connectivity), there is a need for narrowing the space of possible triangulations.

A class of mesh optimization techniques rely on the observation that evenly distributed points in 2D lead to well-shaped triangles [Epp01]. Isotropic 2D meshing can therefore be casted into the problem of isotropic point sampling, which amounts to distributing a set of points on the input domain in as even a manner as possible. One approach to evenly distribute a set of points in 2D is to construct a centroidal Voronoi tessellation [DFG99]. Given a density function defined over a bounded domain $\Omega$, a centroidal Voronoi tessellation (denoted CVT) of $\Omega$ is a class of Voronoi tessellations, where each site happens to coincide with the centroid (i.e., center of mass) of its Voronoi region. The

centroid $\mathbf{c}_i$ of a Voronoi region $V_i$ is calculated as:

$$\mathbf{c}_i = \frac{\int_{V_i} \mathbf{x} \cdot \rho(\mathbf{x})\, \mathrm{d}\mathbf{x}}{\int_{V_i} \rho(\mathbf{x})\, \mathrm{d}\mathbf{x}} \,, \tag{1}$$

where $\rho(\mathbf{x})$ is the density function defined over $V_i$. This structure turns out to have a surprisingly broad range of applications for numerical analysis, location optimization, optimal repartition of resources, cell growth, vector quantization, etc. This follows from the mathematical importance of its relationship with the energy function

$$E(\mathcal{Z}, \mathcal{V}) = \sum_{i=1}^{n} \int_{V_i} \rho(\mathbf{x}) ||\mathbf{x} - \mathbf{z}_i||^2 \, \mathrm{d}\mathbf{x} \,, \tag{2}$$

where $\mathcal{V} = \{V_i\}$ is a partition of $\Omega$ and $\mathcal{Z} = \{z_i\}$. It has been shown [DFG99] that (i) for a given partition $\mathcal{V}$, the energy $E(\mathcal{Z}, \mathcal{V})$ is minimized when $z_i$ is the mass centroid of $V_i$, and (ii) for a given set of centers $\mathcal{Z}$, the energy function $E(\mathcal{Z}, \mathcal{V})$ is minimized when $\mathcal{V}$ is the Voronoi tessellation of the centers.

One way to build a centroidal Voronoi tessellation is to use Lloyd's relaxation method. Lloyd's algorithm is a deterministic, fixed point iteration [Llo82]. Given a density function and an initial set of $n$ sites, Lloyd's algorithm repeats the following two steps until satisfactory convergence has been achieved (see Figure 15):

1. Construct the Voronoi tessellation corresponding to the $n$ sites.
2. Compute the centroids of the $n$ Voronoi regions with respect to the density function, and move the $n$ sites to their respective centroids.

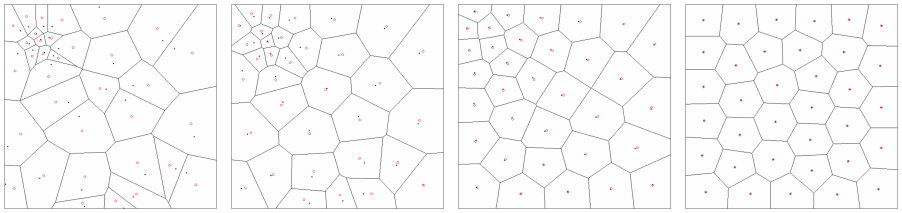


**Fig. 15.** Left: ordinary Voronoi tessellation. Middle left: Voronoi tessellation after one Lloyd iteration. Middle right: Voronoi tessellation after three Lloyd iterations. Right: Centroidal Voronoi tessellation obtained after convergence of the Lloyd iteration. Each generator coincides with the center of mass of its Voronoi cell.

The Lloyd iteration can be applied to the problem of 2D quality mesh generation. Figure 16 illustrates the Lloyd iteration applied to an initial Voronoi diagram, with a variable density map derived from the local feature size of the domain boundary.

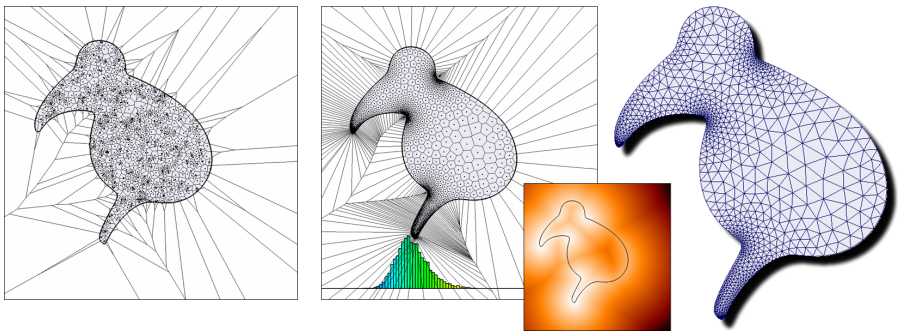


**Fig. 16.** Optimized meshing. Left: Initial Voronoi diagram of a set of points uniformly sampled over the domain. Middle: Voronoi diagram after convergence of the Lloyd iteration, using a variable density function derived from the shape of the domain boundary. The distribution of dual triangle angles is shown. Right: Optimized triangle mesh.

The Lloyd iteration can furthermore be applied to isotropic surface remeshing. One approach is to use a global conformal planar parameterization of the input surface triangle mesh and to apply Lloyd relaxation in the parameter space using a density function designed to compensate for the area distortion due to flattening (see Figure 17).

To alleviate the numerical issues for high isoperimetric distortion, as well as the artificial cuts required for closed or models with non-trivial topology, another approach is to apply the Lloyd relaxation procedure over a set of local overlapping parameterizations [SAG03], see Figure 18.

It has been shown that the energy minimized by the Lloyd iteration corresponds to the volume between a paraboloid and an *underlaid* piecewise linear approximant, which is formed by planar patches tangent to the paraboloid (the lifted Voronoi diagram) [Che05]. In other words, isotropic point sampling can be casted as a function approximation problem in a higher dimension. In 2D, this approach leads to isotropically sampled meshes since it has been shown [She02] that any $L^p$ optimal approximation of a smooth function asymptotically tends to align and shape its elements according to the eigenvectors and eigenvalues of its Hessian: since the Hessian of the isotropic paraboloid function is isotropic, the resulting meshes must have nearly hexagonal Voronoi cells, i.e., nearly equilateral triangles in the dual Delaunay mesh. Unfortunately, such a property does not hold in 3D due to the presence of slivers. We can attribute the slivers to the fact that the energy tends to optimize the compactness of the dual Voronoi cells, but not the compactness of simplexes in the primal Delaunay triangulation: therefore, the presence of a sliver is not penalized by this energy. In other words, minimizing this energy ensures that the vertices in the domain are well-spaced (i.e., isotropic point

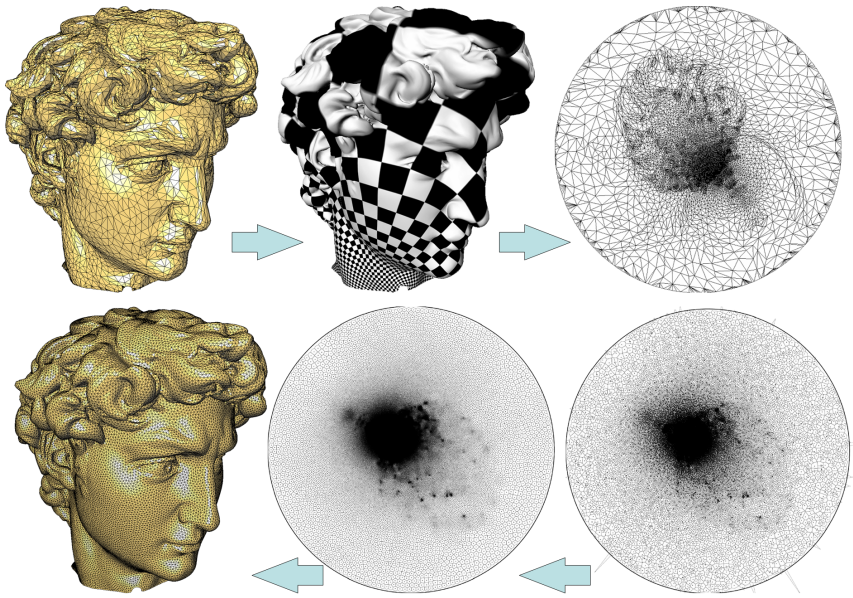


**Fig. 17.** Isotropic remeshing of the Michelangelo David head using global planar conformal parameterization (top). An initial set of generators is randomly generated in parameter space with a variable density so as to compensate for area distortion due to the parameterization (bottom right). The Lloyd iteration is applied in parameter space (bottom middle). The output optimized mesh is uniform once lifted back in 3D (bottom left).

sampling), but having well-spaced vertices guarantees nothing in terms of the quality of the 3D mesh [Epp01].

Chen [Che04] proposed an approach dual to the above in the context of mesh optimization. The main idea is to minimize a dual energy, the volume between a paraboloid and an *overlaid* piecewise linear approximant formed by a linear interpolation of points on the paraboloid, see Figure 19.

Casting the meshing problem as a function approximation problem in a higher dimension, we derive from this approach a variational tetrahedral meshing algorithm [ACSYD05], which alternates between updates of connectivity and vertex positions (see Figures 20 and 21). Connectivity optimization is easily achieved: for a given set of vertex positions, its Delaunay triangulation has the optimal connectivity which minimizes the dual energy. Vertex relocations are computed so as to satisfy a necessary condition for the local optimal $L^1$ approximation of the paraboloid.

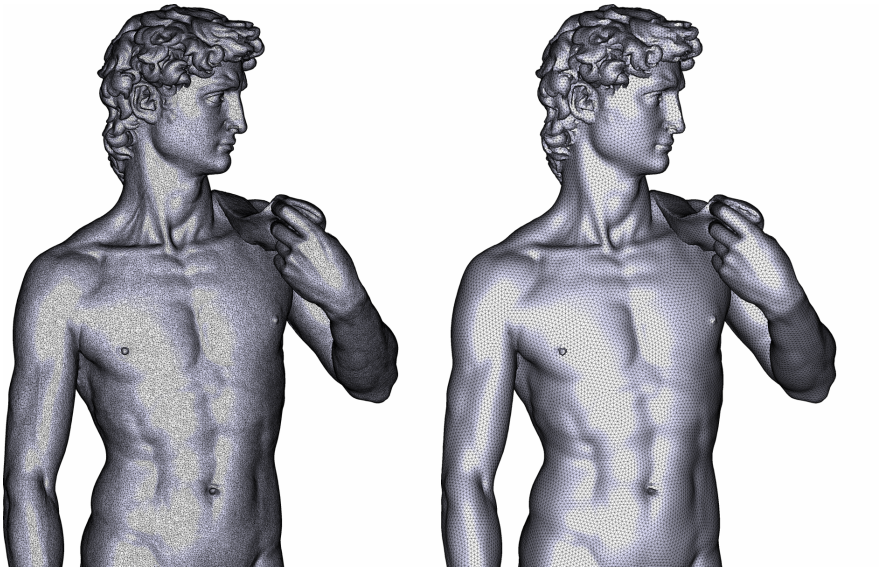


**Fig. 18.** Isotropic remeshing of the Michelangelo David using overlapping parameterizations. Left: Initial triangle surface mesh (350K vertices). Right: Optimized surface mesh (250K vertices).

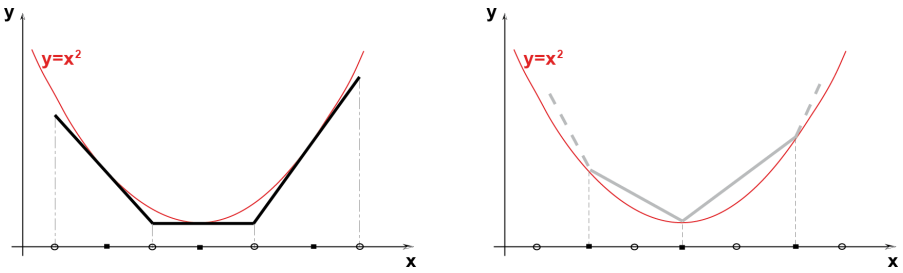



**Fig. 19.** Piecewise linear approximations: The paraboloid can be approximated by an underlaid piecewise linear function (left), or by an overlaid one (right).

## 10 Conclusion

We have presented the current offerings of the CGAL library in terms of Voronoi diagrams and Delaunay tessellations, as well as the main paradigms used, and the design choices made to obtain a unique combination of genericity, robustness and efficiency. In the long run, the exact geometric computation paradigm along with symbolic perturbations for handling degeneracies turns out to be a valid choice for constructing ever complex tessellations. Although the loss of performance is small when parameterizing the algorithms with exact predicates, more work remains to be done in order to elaborate

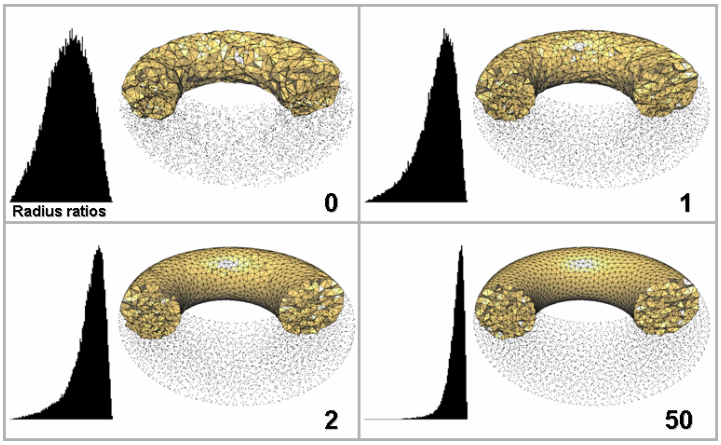



**Fig. 20.** Tetrahedral mesh optimization. The initial mesh is obtained by uniformly sampling the interior of the torus. The mesh and distribution of radius ratios are shown at iterations 1, 2 and 50.

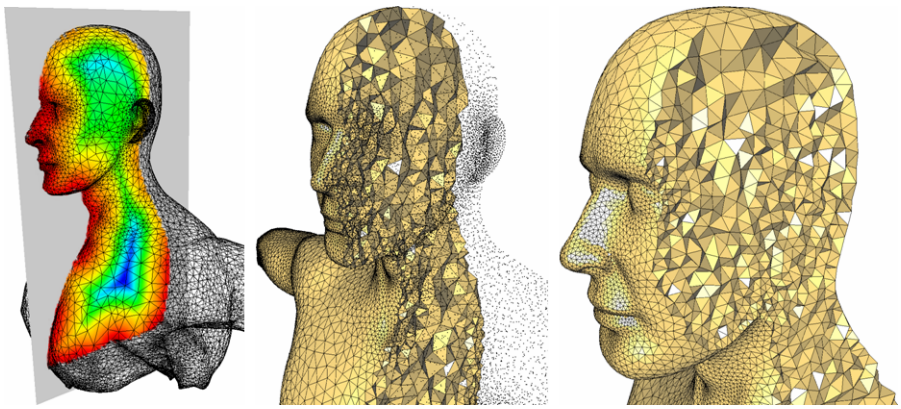


**Fig. 21.** Optimized tetrahedral meshing. Left: Variable sizing function defined inside the domain. The element size is specified in accordance with the local feature size of the domain boundary, and increases with the distance to the boundary. Middle: Cut view of the optimized tetrahedral mesh (80K vertices). All vertices are shown to highlight the increased density near the ear where the local feature size is small. Right: Another cut view of the optimized mesh.

upon efficient cascaded constructions for complex, scaffolded geometric data structures.

The current 3D Delaunay triangulation from CGAL inserts 80K points per second on a Intel Xeon 2.33GHz, when spatial sorting is activated. It consumes 300 MBytes per million points, which limits us to 10M points on a

32-bit computer with 3 GBytes of memory. The maximum number of points increases "only" to 30M points on a 64-bit computer with 16 GBytes, as all pointers used in the data structure consumes twice as much memory (64 vs. 32 bits). We plan to improve both scalability using size-adapted pointers and compact data structures, and efficiency by elaborating upon parallel implementations specialized to modern computer architectures with multi-core CPUs. Furthermore, our preliminary work on spatial sorting can be seen as a first step toward out-of-core implementation.

Although the variety of possible tessellations and Voronoi diagrams is huge, some are particularly relevant for applications. After variations in generators, dimension, metric and space, it is planned to generate ever nicer tessellations using variational techniques aimed at optimizing not only the connectivity and vertex locations as already presented here, but also the metric or even the generators.

## Acknowledgments

This work has been partially supported by the IST Programme of the EU (FET Open) Project under Contract No IST-006413 (ACS - Algorithms for Complex Shapes with Certified Numerics and Topology).

## References

[AB99] Nina Amenta and Marshall Bern. Surface reconstruction by Voronoi filtering. *Discrete Comput. Geom.*, 22(4):481–504, 1999.

[ACDL00] N. Amenta, S. Choi, T. K. Dey, and N. Leekha. A simple algorithm for homeomorphic surface reconstruction. *International Journal of Computational Geometry and Applications*, 12(1):125–141, 2000.

[ACR03] N. Amenta, S. Choi, and G. Rote. Incremental constructions con BRIO. In *Proc. ACM Symposium on Computational Geometry*, pages 211–219, June 2003.

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

[BBK06] Daniel K. Blandford, Guy E. Blelloch, and Clemens Kadow. Engineering a compact parallel Delaunay algorithm in 3D. In *Proceedings of ACM Symposium on Computational Geometry (SoCG)*, June 2006.

[BBP01] H. Brönnimann, C. Burnikel, and S. Pion. Interval arithmetic yields efficient dynamic filters for computational geometry. *Discrete Applied Mathematics*, 109:25–47, 2001.

[BD05] Jean-Daniel Boissonnat and Christophe Delage. Convex hulls and Voronoi diagrams of additively weighted points. In *Proc. 13th European Symposium on Algorithms*, volume 3669 of *Lecture Notes Comput. Sci.*, pages 367–378. Springer-Verlag, 2005.

[BFS01] C. Burnikel, S. Funke, and M. Seel. Exact geometric computation using cascading. *Internat. J. Comput. Geom. Appl.*, 11:245–266, 2001.

[BGL] The BOOST Graph Library. `http://www.boost.org/libs/graph/`.

[BK03] Jean-Daniel Boissonnat and Menelaos I. Karavelas. On the combinatorial complexity of Euclidean Voronoi cells and convex hulls of d-dimensional spheres. In *Proc. 14th ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 305–312, 2003.

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

[BWY06] Jean-Daniel Boissonnat, Camille Wormser, and Mariette Yvinec. Curved Voronoi diagrams. In Jean-Daniel Boissonnat and Monique Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*, pages 67–116. Springer-Verlag, Mathematics and Visualization, 2006.

[CDM06] Luca Castelli Aleardi, Olivier Devillers, and Abdelkrim Mebarki. 2D triangulation representation using stable catalogs. In *Proc. 18th Canad. Conf. Comput. Geom.*, pages 71–74, 2006.

[CDRR04] S.W. Cheng, T.K. Dey, E.A. Ramos, and T. Ray. Sampling and meshing a surface with guaranteed topology and geometry. *Proceedings of the twentieth annual symposium on Computational geometry*, pages 280–289, 2004.

[CGAa] CGAL, Computational Geometry Algorithms Library. `http://www.cgal.org`.

[CGAb] CGAL PYTHON Bindings. `http://cgal-python.gforge.inria.fr/`.

[CGL] CGLAB Toolbox. `http://cglab.gforge.inria.fr/`.

[Che89] P. Chew. Constrained Delaunay triangulations. *Algorithmica*, 4(1):97–108, 1989.

[Che04] Long Chen. Mesh smoothing schemes based on optimal Delaunay triangulations. In *Proceedings of 13th International Meshing Roundtable*, pages 109–120, 2004.

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

[CKK05] Y. Cho, D. Kim, and D.S. Kim. Topology representation for the Voronoi diagram of 3D spheres. *International Journal of CAD/CAM*, 5(1):59–68, 2005.

[CT08] Manuel Caroli and Monique Teillaud. Video: On the computation of 3D periodic triangulations. In *Proc. 24th Annual ACM Symposium on Computational Geometry (SoCG)*, pages 222–223, 2008. `http://www.computational-geometry.org/`.

[CT09] Manuel Caroli and Monique Teillaud. Computing 3D periodic triangulations. In *Proceedings 17th European Symposium on Algorithms*, volume 5757 of *Lecture Notes in Computer Science*, pages 37–48, 2009.

[Dev02] Olivier Devillers. The Delaunay hierarchy. *Internat. J. Found. Comput. Sci.*, 13:163–180, 2002.

[DFG99] Q. Du, V. Faber, and M. Gunzburger. Centroidal Voronoi tesselations: Applications and algorithms. *SIAM Review*, 41(4):637–676, 1999.

[DFNP91] L. De Floriani, B. Falcidieno, G. Nagy, and C. Pienovi. On sorting triangles in a Delaunay tessellation. *Algorithmica*, 6:522–532, 1991.

[DMT92] Olivier Devillers, Stefan Meiser, and Monique Teillaud. The space of spheres, a geometric tool to unify duality results on Voronoi diagrams. In *Proc. 4th Canad. Conf. Comput. Geom.*, pages 263–268, 1992.

[DP03] Olivier Devillers and Sylvain Pion. Efficient exact geometric predicates for Delaunay triangulations. In *Proc. 5th Workshop Algorithm Eng. Exper.*, pages 37–44, 2003.

[DPT02] Olivier Devillers, Sylvain Pion, and Monique Teillaud. Walking in a triangulation. *Internat. J. Found. Comput. Sci.*, 13:181–199, 2002.

[DT03] Olivier Devillers and Monique Teillaud. Perturbations and vertex removal in a 3D Delaunay triangulation. In *Proc. 14th ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 313–319, 2003.

[DT06] Olivier Devillers and Monique Teillaud. Perturbations and vertex removal in Delaunay and regular 3D triangulations. Research Report 5968, INRIA, 2006.

[Ede90] H. Edelsbrunner. An acyclicity theorem for cell complexes in $d$ dimensions. *Combinatorica*, 10(3):251–260, 1990.

[EK06] I. Z. Emiris and M. I. Karavelas. The predicates of the Apollonius diagram: algorithmic analysis and implementation. *Computational Geometry: Theory and Applications, Special Issue on Robust Geometric Algorithms and their Implementations*, 33(1–2):18–57, January 2006.

[EM90] H. Edelsbrunner and E.P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics (TOG)*, 9(1):66–104, 1990.

[EM94] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Trans. Graph.*, 13(1):43–72, January 1994.

[Epp01] D. Eppstein. Global optimization of mesh quality. *Tutorial at the 10th International Meshing Roundtable*, 10, 2001.

[ES86] H. Edelsbrunner and R. Seidel. Voronoi diagrams and arrangements. *Discrete Comput. Geom.*, 1:25–44, 1986.

[FV96] S. Fortune and C. J. Van Wyk. Static analysis yields efficient exact integer arithmetic for computational geometry. *ACM Trans. Graph.*, 15(3):223–248, July 1996.

[GMP] GMP, GNU Multiple Precision Arithmetic Library. `http://www.swox.com/gmp`.

[ILSS06] Martin Isenburg, Yuanxin Liu, Jonathan Shewchuk, and Jack Snoeyink. Streaming computation of Delaunay triangulations. In *Proceedings of SIGGRAPH'06*, pages 1049–1056, July 2006.

[IPE] The Ipe extensible drawing editor. `http://tclab.kaist.ac.kr/ipe/`.

[Kar04] M. I. Karavelas. A robust and efficient implementation for the segment Voronoi diagram. *International Symposium on Voronoi Diagrams in Science and Engineering (VD2004)*, pages 51–62, 2004.

[KCK05a] D.S. Kim, Y. Cho, and D. Kim. Euclidean Voronoi diagram of 3D balls and its computation via tracing edges. *Computer-Aided Design*, 37(13):1412–1424, 2005.

[KCK+05b] D.S. Kim, Y. Cho, D. Kim, S. Kim, J. Bhak, and S.H. Lee. Euclidean Voronoi diagrams of 3D spheres and applications to protein

structure analysis. *Japan Journal of Industrial and Applied Mathematics*, 22(2):251–265, 2005.

[KE03] Menelaos I. Karavelas and Ioannis Z. Emiris. Root comparison techniques applied to computing the additively weighted Voronoi diagram. In *Proc. 14th ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 320–329, 2003.

[Kle89] R. Klein. *Concrete and Abstract Voronoi Diagrams.* Springer, 1989.

[KMM93] R. Klein, K. Mehlhorn, and S. Meiser. Randomized incremental construction of abstract Voronoi diagrams. *Computational Geometry: Theory and Applications*, 3(3):157–184, 1993.

[KMP+04] Lutz Kettner, Kurt Mehlhorn, Sylvain Pion, Stefan Schirra, and Chee Yap. Classroom examples of robustness problems in geometric computations. In *Proc. 12th European Symposium on Algorithms*, volume 3221 of *Lecture Notes Comput. Sci.*, pages 702–713. Springer-Verlag, 2004.

[KSK+06] D.S. Kim, J. Seo, D. Kim, J. Ryu, and C.H. Cho. Three-dimensional beta shapes. *Computer-Aided Design*, 38(11):1179–1191, 2006.

[KY03] Menelaos I. Karavelas and Mariette Yvinec. The Voronoi diagram of planar convex objects. In *Proc. 11th European Symposium on Algorithms*, volume 2832 of *Lecture Notes in Computer Science*, pages 337–348. Springer-Verlag, 2003.

[LED] LEDA, Library for Efficient Data Types and Algorithms. `http://www.algorithmic-solutions.com/enleda.htm`.

[Lin89] A. Lingas. Voronoi diagrams with barriers and their applications. *Inform. Process. Lett.*, 32:191–198, 1989.

[Llo82] S. Lloyd. Least square quantization in PCM. *IEEE Trans. Inform. Theory*, 28:129–137, 1982.

[LS03] François Labelle and Jonathan Richard Shewchuk. Anisotropic Voronoi diagrams and guaranteed-quality anisotropic mesh generation. In *SCG '03: Proceedings of the nineteenth annual symposium on Computational geometry*, pages 191–200, New York, NY, USA, 2003. ACM Press.

[MP05] Guillaume Melquiond and Sylvain Pion. Formal certification of arithmetic filters for geometric predicates. In *Proc. 17th IMACS World Congress on Scientific Computation, Applied Mathematics and Simulation*, 2005.

[MPF] MPFR, Multiple-Precision Floating-Point Computations. `http://www.mpfr.org`.

[QT] QT, Cross-Platform Rich Client Development Framework. `http://www.trolltech.com/products/qt`.

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

[SAG03] Vitaly Surazhsky, Pierre Alliez, and Craig Gotsman. Isotropic Remeshing of Surfaces: a Local Parameterization Approach. In *Proceedings of 12th International Meshing Roundtable*, pages 215–224, 2003.

[SCI] SCILAB, the open source platform for numerical computation. `http://www.scilab.org/`.

[Sei98a] R. Seidel. Constrained Delaunay triangulations and Voronoi diagrams with obstacles. In H. S. Poingratz and W. Schinnerl, editors, *1978–1988 Ten Years IIG*, pages 178–191. IIG-TU Graz, Austria, Report 260, 1998.

[Sei98b] R. Seidel. The nature and meaning of perturbations in geometric computing. *Discrete Comput. Geom.*, 19:1–17, 1998.

[She97] Jonathan Richard Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete Comput. Geom.*, 18(3):305–363, 1997.

[She02] J. R. Shewchuk. What is a good linear element? interpolation, conditioning, and quality measure. In *Proc. of 11th Int. Meshing Roundtable*, pages 115–126, 2002.

[YD94] C. Yap and T. Dubé. The exact computation paradigm. *Computing in Euclidean Geometry*, 1994.