Triangulating Point Sets in Orbit Spaces

Manuel Caroli

▶ **To cite this version:**

Manuel Caroli. Triangulating Point Sets in Orbit Spaces. Computer Science [cs]. Université Nice Sophia Antipolis, 2010. English. NNT : . tel-00552215

HAL Id: tel-00552215
https://theses.hal.science/tel-00552215v1

Submitted on 5 Jan 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITE DE NICE – SOPHIA ANTIPOLIS

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

T H E S E

pour obtenir le titre de

Docteur en Sciences

de l'Université de Nice – Sophia Antipolis

Mention : Informatique

présentée et soutenue par

Manuel CAROLI

Triangulating Point Sets in Orbit Spaces

Thèse dirigée par Monique TEILLAUD

soutenue le 10/12/2010

Rapporteurs :	Kurt MEHLHORN	–	MPI für Informatik, Saarbrücken
	John M. SULLIVAN	–	TU Berlin
Examinateurs :	Éric COLIN DE VERDIÈRE	–	Ecole normale supérieure, Paris
	Menelaos KARAVELAS	–	University of Crete
	Jean-Marc SCHLENKER	–	Université Toulouse III
	Monique TEILLAUD	–	INRIA Sophia Antipolis – Méditerranée
	Gert VEGTER	–	University of Groningen
Invité :	Andreas FABRI	–	GeometryFactory Sarl

UNIVERSITY OF NICE – SOPHIA ANTIPOLIS

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

P h D T H E S I S

to obtain the title of

Doctor of Sciences

of the University of Nice – Sophia Antipolis

Specialty: Computer Science

prepared and defended by

Manuel Caroli

Triangulating Point Sets in Orbit Spaces

Advisor: Monique Teillaud

defended on Dec. 10, 2010

Reviewers:	Kurt Mehlhorn	–	MPI für Informatik, Saarbrücken
	John M. Sullivan	–	TU Berlin
Examinators:	Éric Colin de Verdière	–	Ecole normale supérieure, Paris
	Menelaos Karavelas	–	University of Crete
	Jean-Marc Schlenker	–	Université Toulouse III
	Monique Teillaud	–	INRIA Sophia Antipolis – Méditerranée
	Gert Vegter	–	University of Groningen
Invited:	Andreas Fabri	–	GeometryFactory Sarl

Résumé

Dans cette thèse, nous étudions les triangulations définies par un ensemble de points dans des espaces de topologies différentes. Nous proposons une définition générale de la triangulation de Delaunay, valide pour plusieurs classes d'espaces, ainsi qu'un algorithme de construction. Nous fournissons une implantation pour le cas particulier du tore plat tridimensionnel.

Ce travail est motivé à l'origine par le besoin de logiciels calculant des triangulations de Delaunay périodiques, dans de nombreux domaines dont l'astronomie, l'ingénierie des matériaux, le calcul biomédical, la dynamique des fluides, etc. Les triangulations périodiques peuvent être vues comme des triangulations du tore plat. Nous fournissons une définition et nous développons un algorithme incrémentiel efficace pour calculer la triangulation de Delaunay dans le tore plat. L'algorithme est adapté de l'algorithme incrémentiel usuel dans $\mathbb{R}^d$. Au contraire des travaux antérieurs sur les triangulations périodiques, nous évitons de maintenir plusieurs copies périodiques des points, lorsque cela est possible. Le résultat fourni par l'algorithme est toujours une triangulation du tore plat.

Nous présentons une implantation de notre algorithme, à présent disponible publiquement comme un module de la bibliothèque d'algorithmes géométriques CGAL[1].

Nous généralisons les résultats à une classe plus générale d'espaces quotients plats, ainsi qu'à des espaces quotients de courbure constante positive. Enfin, nous considérons le cas du tore double, qui est un exemple de la classe beaucoup plus riche des espaces quotients de courbure négative constante.

Mots-clés : triangulation de Delaunay, espace quotient, complexe simplicial, triangulation périodique, tore plat, revêtement, algorithme incrémentiel, variété euclidienne fermée

[1] www.cgal.org

Abstract

In this work we discuss triangulations of different topological spaces for given point sets. We propose both definitions and algorithms for different classes of spaces and provide an implementation for the specific case of the three-dimensional flat torus.

The work is originally motivated by the need for software computing three-dimensional periodic Delaunay triangulations in numerous domains including astronomy, material engineering, biomedical computing, fluid dynamics etc. Periodic triangulations can be understood as triangulations of the flat torus. We provide a definition and develop an efficient incremental algorithm to compute Delaunay triangulations of the flat torus. The algorithm is a modification of the incremental algorithm for computing Delaunay triangulations in $\mathbb{E}^d$. Unlike previous work on periodic triangulations we avoid maintaining several periodic copies of the input point set whenever possible. Also the output of our algorithm is guaranteed to always be a triangulation of the flat torus. We provide an implementation of our algorithm that has been made available to a broad public as a part of the Computational Geometry Algorithms Library CGAL[2]. We generalize the work on the flat torus onto a more general class of flat orbit spaces as well as orbit spaces of constant positive curvature. We furthermore consider the much richer class of orbit spaces of constant negative curvature.

Keywords: Delaunay triangulation, orbit space, simplicial complex, periodic triangulation, flat torus, covering space, incremental algorithm, closed Euclidean manifold

[2] www.cgal.org

Acknowledgments

The work accomplished in the present thesis would not have been possible without the support of many people that accompanied me during the last three years. It is impossible to mention them all but nevertheless I would like to use these few lines to mention those who influenced me most.

First of all I would like to thank my advisor Monique Teillaud for having proposed this interesting topic and for accepting me as a student. She was always available and gave me a lot of support, it was a real pleasure to be able to work with her.

It is a great honor to me that Kurt Mehlhorn and John M. Sullivan accepted to review my thesis. I am very grateful to the examinators Éric Colin de Verdiére, Menelaos Karavelas, Jean-Marc Schlenker, and Gert Vegter for coming to Sophia Antipolis to attend my defense. I further thank Andreas Fabri for accepting to be a visiting examinator.

I would like to mention Nico Kruithof for having started some initial research. His work helped me significantly to get started with the topic. I also want to thank Mridul Aanjaneya for fruitful discussions on formalizing the problem. Arijit Ghosh was always available for math questions or discussing interesting problems of any kind.

I want to thank Vissarion Fisikopoulos and Mikhail Bogdanov for their efforts to establish compatibility between the CGAL 3D Periodic triangulations and the CGAL Surface mesher and the CGAL Volume mesher, respectively.

During my work I had the occasion to cooperate with research teams from TU Graz, Austria and from the university of Groningen, Netherlands.

I am grateful to the people from Graz, Oswin Aichholzer and his project team (particularly Wolfgang Aigner, Thomas Hackl, Bernhard Kornberger, Birgit Vogtenhuber) as well as Franz Aurenhammer. We had lots of interesting discussions during our mutual visits.

In the scope of the associate team "OrbiCG" we had several mutual visits with Rien van de Weijgaert and Gert Vegter from Groningen. I really appreciated the fruitful discussions both on applications of my code as well as on my research topic itself.

I am very thankful for the support I got from the CGAL project during the whole process of implementation and submission of my package. I appreciated the open atmosphere and the trust that was put in me from the very beginning.

Through the ANR project "Triangles" I had the possibility to get in touch and exchange ideas with French computer scientists and mathematicians on geometry-related topics. Our various meetings were always a positive experience and often provided me with new ideas on specific problems.

Finally, I want to thank the current and former members of the INRIA project teams Géométrica and ABS for all their support and the friendly atmosphere in which I had the pleasure to pass the three years of my PhD studies.

I found a very positive and open international atmosphere both at INRIA as well as outside of INRIA. I am thankful to everybody I met and who enriched my stay of almost four years in France. I will always keep a very good memory of my time in Sophia-Antipolis.

Schließlich möchte ich mich bei meiner Familie bedanken, die mich stets unterstützt und mir ermöglicht hat, meinen bisherigen Weg zu beschreiten.

Contents

Chapter 1

Introduction

Triangulations are considered to be one of the most important structures in Computational Geometry. The Delaunay triangulation is a special type of triangulation that exhibits several very interesting and useful properties. Together with its dual, the Voronoi diagram, it is of great importance to many applications. The Delaunay triangulation and the Voronoi diagram are well-studied; many efficient algorithms are known and many implementations exist. However, most of these results are restricted to Delaunay triangulations in the d-dimensional Euclidean space $\mathbb{E}^d$.

In this work we consider the problem of defining and computing Delaunay triangulations of spaces other than $\mathbb{E}^d$. The introductory section is organized as follows: We first discuss the motivation for our work. Then we give the required background knowledge on Delaunay triangulations and on orbit spaces. After these introductions, we state the problem we consider in this work. Finally we review previous work on Delaunay triangulations and Voronoi diagrams of spaces other than $\mathbb{E}^d$ and present our contributions to the topic.

1.1 Motivation

This work is motivated by needs for computing periodic triangulations in different domains of science. There are several reasons for working in a periodic space: Firstly, the input can be a point set with an inherent periodicity. Another reason is that often the size of the model is too big to run computations on it, so the experiments are run on a small sample, replicated periodically to avoid boundary effects. And finally, there are some interesting mathematical questions on triangulations and meshes, where the use of periodic triangulations can help to gain further insight. In all cases it is desirable to compute the triangulation of a smaller point set containing only one periodic copy of each input point to avoid redundancy in the computation.

We now give a few concrete examples of problems in different fields of science that use periodic triangulations. Some of them have been presented at the "CGAL Prospective Workshop on Geometric Computing in Periodic Spaces"[1], at INRIA Sophia-Antipolis Méditerranée, France, October 2008 and at the workshop "Subdivide and Tile: Triangulating spaces for understanding the world"[2], at the Lorentz Center in Leiden, Netherlands, November 2009.

[1] http://www.cgal.org/Events/PeriodicSpacesWorkshop/
[2] http://www.lorentzcenter.nl/lc/web/2009/357/info.php3?wsid=357

We first consider an example of an inherently periodic problem: In material sciences, researchers develop new materials that should have some specific properties. For example, Moesen et al. [Moe08] develop bone scaffolds, i.e. materials that can be used to replace broken bones. In order to verify the material properties they use finite element simulations. More specifically, they want to compute a Delaunay-based volume mesh of the scaffold. These scaffolds are periodic; in order to mesh the whole scaffold it suffices to mesh only one periodic domain such that the mesh can be replicated and glued together, see Figure 1.1(b). When computing in a periodic setting, two copies of the mesh fit together by construction.

In cosmology, the subfield of astronomy dealing with large-scale structure in the universe, the so-called cosmological principle states: "Viewed on a sufficiently large scale, the properties of the Universe are the same for all observers." [Kee02]. Thus for simulations on the large-scale structure in the universe, a sufficiently large sample of the structure can be used, replicated periodically; see Figure 1.1(a). We have been in contact with astronomers who developed a "Delaunay tessellation Field Estimator" [vdWS07] and who are interested in using periodic Delaunay triangulations and Voronoi diagrams. They are also interested in Betti numbers of periodic alpha shapes [vdWVP+10].

Granular materials show particle properties at micro-scale and continuum-mechanical properties at macro-scale. To understand the relation between both kinds of properties, a method based on a hierarchy of Delaunay triangulations has been developed [Dur08, Kru09]. To avoid boundary effects, the authors use a periodic setting.

Astrophysical simulations often work using a method called *adaptive mesh refinement* that is based on regular grids. This leads to problems when the scaling range should be big. Either some intermediate grid size is used that is not very well adapted for extreme scales or the grid is refined where a smaller scaling is required, which makes the handling of the grid considerably more complicated. Using Voronoi diagrams instead of regular grids seems a well-adapted option in this case. There are applications of this in a periodic setting as well [Dul08].

Similar approaches are used in computational structural biology [Ber09], fluid dynamics [Cam09, Duq], particle dynamics [dFC03], solid mechanics [NG02], biomedical computing [Wei08] etc., this list being far from exhaustive. Some of these applications also ask for different types of periodicities, such as periodicities in less than three directions in $\mathbb{E}^3$ or point sets generated by more complicated isometries, such as reflections or rotations.

There are also mathematical questions, where periodic triangulations can be useful. For example the Kelvin Problem [Tho87]: How can space be partitioned into cells of equal volume with the least area of surface between them? See Figure 1.1(c) for the bitruncated cubic honeycomb, a partition with truncated octahedra proposed by Kelvin himself. There is currently progress made on this topic using periodic Voronoi diagrams [Gab09]. There are more topics, e.g., [Rob06] works on Betti number signatures of point sets in a periodic setting.

1.2 Triangulations in $\mathbb{E}^d$

In this section we recapitulate the definition and the properties of the Delaunay triangulation. We start with underlying concepts such as simplicial complexes and triangulations. Then we introduce Delaunay triangulations, some important properties, and explain in detail the incremental algorithm that is used subsequently.

(a) (b) (c)

Figure 1.1: (a): Large-scale structure of the cosmic web, courtesy R. v. d. Weijgaert. (b): Bone scaffold, courtesy M. Moesen. (c): Bitruncated cubic honeycomb (Image: [Kep]).

1.2.1 Simplicial complexes

We now introduce the notions of simplices and simplicial complexes. For further reading see [Zom05].

We denote the d-dimensional Euclidean space by $\mathbb{E}^d$. The distance between p and q is denoted by $\mathrm{dist}(p,q)$, the length of the vector $\boldsymbol{v} := q - p$ from p to q is denoted by $\|\boldsymbol{v}\|$.

A set $A \subseteq \mathbb{E}^d$ is said to be *convex* if for all points $p, q \in A$ the line segment between p and q is contained in A.

Let $p_0, p_1, \ldots p_k$ be points in $\mathbb{E}^d$, $k \leq d$. The linear combination $\lambda_0 p_0 + \lambda_1 p_1 + \ldots + \lambda_k p_k$ with $\sum_{i=0}^{k} \lambda_i = 1$, $\lambda_i \in \mathbb{R}$, spans an affine variety. The points are said to be *affinely independent* if the affine variety is a space of dimension k.

Definition 1.2.1 (k-simplex)**.** *A* k-simplex σ *in* $\mathbb{E}^d$, $k \leq d$, *is the convex hull of* $k+1$ *affinely independent points* $\mathcal{P}_\sigma = \{p_0, p_1, \ldots, p_k\}$.

A simplex τ defined by $\mathcal{P}_\tau \subseteq \mathcal{P}_\sigma$ is a *face* of σ and has σ as a *coface*. This is denoted by $\sigma \geq \tau$ and $\tau \leq \sigma$. Note that $\sigma \geq \sigma$ and $\sigma \leq \sigma$.

There exist several definitions of simplicial complexes in the literature. Often they restrict to a finite number of simplices [Zom05, RV06]. In the sequel, we deal with infinite simplicial complexes, so, we use the definition given in [Lee00]:

Definition 1.2.2 (Simplicial complex)**.** *A* simplicial complex *is a set* $\mathcal{K}$ *of simplices such that:*

(i). $\sigma \in \mathcal{K}, \tau \leq \sigma \Rightarrow \tau \in \mathcal{K}$

(ii). $\sigma, \sigma' \in \mathcal{K} \Rightarrow \sigma \cap \sigma' \leq \sigma$ *and* $\sigma \cap \sigma' \leq \sigma'$

(iii). Every point in a simplex of $\mathcal{K}$ *has a neighborhood that intersects at most finitely many simplices in* $\mathcal{K}$ *(local finiteness).*

Of course, if $\mathcal{K}$ is finite, then condition (iii) is always fulfilled.

We give some more definitions: Let $\mathcal{K}$ be a simplicial complex. If a subset of $\mathcal{K}$ is a simplicial complex as well, we call it *subcomplex* of $\mathcal{K}$. The *star* of a subset $\mathcal{L} \subseteq \mathcal{K}$ consists

of the cofaces of simplices in $\mathcal{L}$:

$$\mathrm{St}(\mathcal{L}) := \{\sigma \in \mathcal{K} \mid \exists \tau \in \mathcal{L}, \sigma \geq \tau\}.$$

The *closure* $\overline{\mathcal{L}}$ of $\mathcal{L}$ is the smallest subcomplex containing $\mathcal{L}$:

$$\overline{\mathcal{L}} := \{\tau \in \mathcal{K} \mid \exists \sigma \in \mathcal{L}, \tau \leq \sigma\}.$$

The *link* of $\mathcal{L}$ is defined as follows

$$\mathrm{Lk}(\mathcal{L}) := \overline{\mathrm{St}(\mathcal{L})} - \mathrm{St}(\overline{\mathcal{L}}).$$

See Figure 1.2 for an illustration of star, closure, and link. In subsequent discussions, we

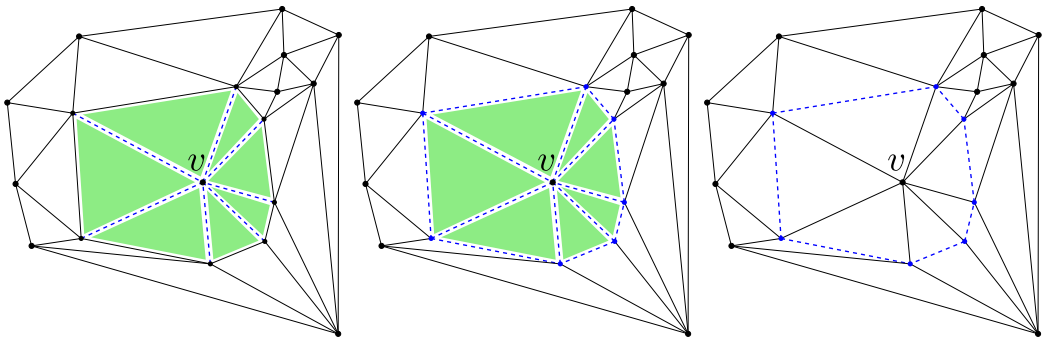


Figure 1.2: Left: $\mathrm{St}(\{v\})$, shaded triangles and dashed segments. Center: $\overline{\mathrm{St}(\{v\})}$, shaded triangles, dashed segments, and incident vertices. Right: $\mathrm{Lk}(\{v\})$, dashed lines and incident vertices.

will be interested in the union of the simplices from a set $\mathcal{L}$, which we denote by $\bigcup \mathcal{L}$. We denote the interior of a simplex σ by $\mathring{\sigma}$. Two simplices σ and τ are said to be *internally disjoint* if their interiors are disjoint, i.e. $\mathring{\sigma} \cap \mathring{\tau} = \emptyset$ (see Figure 1.3).

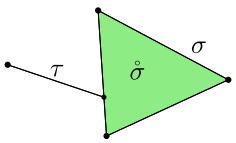


Figure 1.3: σ and τ are *internally disjoint*.

Note that the definition of simplicial complexes and the subsequent definitions are purely combinatorial and do not depend on the embedding space. Thus in order to use these definitions in spaces other than $\mathbb{E}^d$, only the building blocks, the simplices, are required to be defined.

We can now consider triangulations, which are very basic structures in Computational Geometry. Triangulations are widely used in all types of applications, such as finite element methods and meshing. We first give the formal definition.

Definition 1.2.3 (Triangulation)**.** *Let $\mathbb{X}$ be a topological space in which simplices are defined. A* triangulation *of $\mathbb{X}$ is a simplicial complex $\mathcal{K}$ such that $\bigcup \mathcal{K}$ is homeomorphic to $\mathbb{X}$. A* triangulation of $\mathbb{X}$ defined by a point set $\mathcal{S}$ *is a triangulation of $\mathbb{X}$ such that the set of vertices of the triangulation is identical to $\mathcal{S}$.*

In $\mathbb{E}^d$, the union of a finite number of simplices is compact, whereas $\mathbb{E}^d$ itself is not compact. So according to the above definition there is no finite triangulation of $\mathbb{E}^d$. However, we can define triangulations of the one-point-compactification $\mathbb{E}^d \cup \{\infty\}$ of $\mathbb{E}^d$, where combinatorial simplices formed by ∞ and d finite vertices are considered as infinite d-simplices. From now on, when we refer to a finite triangulation of the point set $\mathcal{S}$ in $\mathbb{E}^d$, we actually mean a triangulation of the point set $\mathcal{S} \cup \{\infty\}$ in $\mathbb{E}^d \cup \{\infty\}$.

1.2.2 The Delaunay triangulation

The Delaunay triangulation is named after Boris Delaunay who was one of the first to explore the very interesting and useful properties of this by now well-studied structure in computational geometry [Del34].

In order to define the Delaunay triangulation of $\mathbb{E}^d$ given by a point set, we first need some notation. Let $\mathcal{S}$ be a discrete point set in $\mathbb{E}^d$. Let σ be a d-simplex; the $d+1$ vertices of σ lie on the boundary of a uniquely defined d-ball, called the *circumscribing ball* of σ. If one of the vertices of σ is ∞, then the circumscribing ball of σ is a half-space bounded by the hyperplane defined by the remaining d vertices.

Definition 1.2.4 (Delaunay triangulation)**.** *A d-simplex in a d-dimensional triangulation whose circumscribing ball does not contain any vertex of the triangulation in its interior is said to have the* Delaunay property*. A triangulation of $\mathbb{E}^d$ with vertex set $\mathcal{S}$ is called the* Delaunay triangulation of $\mathbb{E}^d$ defined by $\mathcal{S}$ *if each d-simplex in the triangulation has the Delaunay property.*

We write $DT(\mathcal{S})$ to denote the Delaunay triangulation of $\mathbb{E}^d$ defined by $\mathcal{S}$.

To explore more properties of the Delaunay triangulation we now consider its dual structure, the Voronoi diagram:

Definition 1.2.5 (Voronoi diagram)**.** *The Voronoi cell $Vor(p, \mathcal{S})$ of a point $p \in \mathcal{S}$ consists of all points q in $\mathbb{E}^d$ such that* $\mathrm{dist}(q, p) < \mathrm{dist}(q, p_i)$ *for all $p_i \in \mathcal{S} - \{p\}$.*

The Voronoi diagram $VD(\mathcal{S})$ of the point set $\mathcal{S}$ is the partition of $\mathbb{E}^d$ into the Voronoi cells of the points in $\mathcal{S}$.

See Figure 1.4 for an illustration of a Delaunay triangulation and a Voronoi diagram in the plane. For further reading on Delaunay triangulations and Voronoi diagrams, see [dBvKOS00, BY98, Aur91, AK00].

The dual graph of the Voronoi diagram has the following properties: The sites correspond to the vertices of the dual graph, the Voronoi vertices correspond to cells in the dual graph, the features of intermediate dimension follow implicitly. Such a dual of the Voronoi diagram of $\mathcal{S}$ is called the *Delaunay graph of $\mathcal{S}$*.

We say that $\mathcal{S}$ is in general position if there is no subset of more than $k+1$ points that lie on one k-sphere or in one k-plane for $k < d$. If the points in $\mathcal{S}$ are in general position, then the Delaunay graph is the Delaunay triangulation.

If the points are not in general position, then the subsets of cospherical points form convex polyhedra in the Delaunay graph. In order to transform the Delaunay graph to a Delaunay triangulation, these polyhedra must be triangulated. As the points are cospherical, any triangulation of these polyhedra has the Delaunay property. For practical purposes it is often useful to always have a uniquely defined Delaunay triangulation. For point sets that are not in general position this can be achieved using symbolic perturbation [DT03].

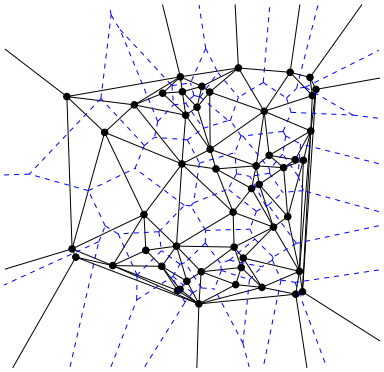


Figure 1.4: Delaunay triangulation (solid lines) and Voronoi diagram (dashed lines) of 50 points in the plane.

The Voronoi diagram can be generalized to a *weighted Voronoi diagram* or *power diagram*, where a real value, the so-called *weight*, is attached to each point in the set $\mathcal{S}$ and the Euclidean distance is replaced by the power distance: Let (p, w_p) and (q, w_q) denote points p and q in $\mathbb{E}^d$ together with their weights w_p and w_q, respectively. Then the power distance is given by

$$\Pi((p, w_p), (q, w_q)) := \mathrm{dist}(p, q)^2 - w_p - w_q.$$

The *weighted Delaunay triangulation* or *regular triangulation* can be defined as the dual of the power diagram in the same way as the Delaunay triangulation is the dual of the Voronoi diagram. Weighted Delaunay triangulations are a generalization of Delaunay triangulations; geometrically, they can be understood as follows: The weighted points (p, w_p) and (q, w_q) can be considered as spheres of radius $\sqrt{w_p}$ and $\sqrt{w_q}$ centered at p and q, respectively. The power product of two weighted points is 0 if and only if the corresponding spheres intersect orthogonally. Thus two such points are said to be *orthogonal*. Analogously, if the power product of two weighted points is strictly negative or positive the weighted points are said to be *superorthogonal* or *suborthogonal*, respectively. In a non-degenerate case, $d + 1$ weighted points admit exactly one common orthogonal weighted point (o, w_o), i.e., (o, w_o) is orthogonal to each of the $d + 1$ weighted points. The $d + 1$ weighted points are said to have the *weighted Delaunay property* if for any other weighted point in $\mathcal{S}$, the power product with (o, w_o) is non-negative. If the input point set is in general position, i.e., if not more than $d + 1$ weighted points have the same common orthogonal point, the weighted Delaunay triangulation is the set of all d-simplices formed by $(d + 1)$-tuples of points having this property. See [DT06b] for a treatment of degenerate cases. Note that if we add the same amount to the weight of each point in $\mathcal{S}$, the weighted Delaunay triangulation does not change. See Figure 1.5 for a weighted Delaunay triangulation. For more on weighted Delaunay triangulations see [ES96].

Most of what we describe in Chapter 2 extends to weighted Delaunay triangulations. Where this is not the case, we treat weighted Delaunay triangulations separately.

Properties of triangulations:
The size of any planar triangulation is linear in the number of vertices, which can be seen easily from the Euler's formula: Let n, e, f denote the number of vertices, edges, and

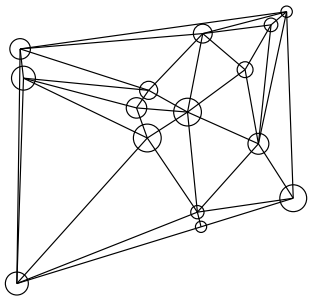


Figure 1.5: A weighted Delaunay triangulation of $\mathbb{E}^2$.

faces in the triangulation, respectively. Then Euler's formula states that $n - e + f = 2$. Furthermore, each face has three edges and each edge corresponds to two faces, thus we have $2e = 3f$. Plugging this into Euler's formula gives $f = 2n - 4 \in \Theta(n)$.

The size of a three-dimensional triangulation can be quadratic in the number of vertices. Let c denote the number of cells and e the number of edges as above. The 3D version of Euler's formula states $n - e + f - c = 0$, which implies $c \in O(e)$. An edge is defined by a pair of vertices, so the number of edges is in $O(n^2)$. There are examples of quadratic sized Delaunay triangulations, so the bound is tight (see Figure 1.6). The size of Delaunay triangulations of $\mathbb{E}^d$ is bounded by $O(n^{\lceil \frac{d}{2} \rceil})$, which is a tight bound [AAD07].

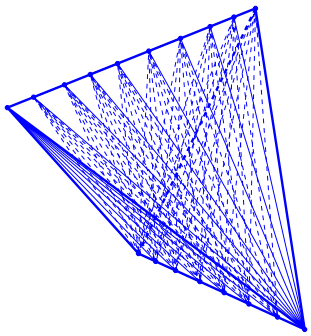


Figure 1.6: A quadratic size Delaunay triangulation of $\mathbb{E}^3$.

1.2.3 Algorithms to compute Delaunay triangulations of $\mathbb{E}^d$

There are various algorithms known for computing the Delaunay triangulation [Bow81, For87, LS80, Wat81]. The first algorithms have been developed for planar Delaunay triangulations. By the duality of the Delaunay triangulation and the Voronoi diagram, algorithms for computing Voronoi diagrams of point sets can be used to compute Delaunay triangulations as well.

In $\mathbb{E}^2$ the size of the Delaunay triangulation is in $O(n)$ as seen above. The lower bound on the complexity of computing it is in $\Omega(n \log n)$ because the sorting problem can be reduced to a Delaunay triangulation computation: Imagine points given on one half of the unit parabola. All input points are part of the convex hull of the triangulation, see Figure 1.7. They can be extracted from the triangulation in increasing order of their x-coordinate in linear time. Thus the construction of the Delaunay triangulation must be in $\Omega(n \log n)$.

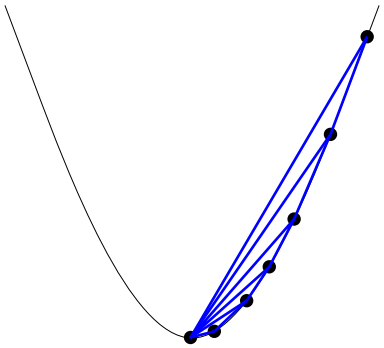


Figure 1.7: Using the Delaunay triangulation algorithm to sort points along x.

We briefly introduce the most commonly used approaches to compute Delaunay triangulations, first for the planar case and then for higher dimensions.

Incremental This approach is efficient in practice and easy to implement. It has first been described by [Bow81] and [Wat81].

Given a point set $\mathcal{S}$, the algorithm starts with a triangulation of $d+1$ points from $\mathcal{S}$ in general position, which is necessarily Delaunay, and inserts the remaining points one by one, restoring the Delaunay property after each point insertion. Each point insertion consists of a *point location* step and a *point insertion* step.

In the naive approach the point location step is in $O(n^{\lceil \frac{d}{2} \rceil})$ in the worst case. The point insertion step inserts the point into the simplex that contains it and restores the Delaunay property. The worst-case complexity of the point insertion step is thus bounded by $O(n^{\lceil \frac{d}{2} \rceil})$. There are n point-location and insertion steps, so the overall complexity is $O(n^{\lceil \frac{d}{2} \rceil + 1})$.

This result can be improved considerably using randomization and a point-location data structure. Randomization means here to insert the points of $\mathcal{S}$ in random order. We now give the results of a randomized worst-case analysis, i.e., we consider a worst-case point set $\mathcal{S}$ and randomized insertion order.

The first results have been achieved for Delaunay triangulations of $\mathbb{E}^2$ only. Using a point-location data-structure the point location step can be improved to an expected complexity of $O(\log n)$. Several such point-location data-structures have been proposed, all requiring expected $O(n)$ space: [CS88] first proposed an offline version, i.e., the point set must be known in advance. [GKS92] introduced an online version and [DMT92a, Dev02] described a fully dynamic approach, i.e. allowing for vertex removal. When using such a point-location data-structure and a randomized insertion order, the expected complexity of the point insertion step is $O(1)$. This yields an overall optimal randomized worst-case complexity of $O(n \log n)$.

The randomized incremental analysis can be easily extended to $\mathbb{E}^3$, resulting in expected time and space complexity $O(n^2)$, which is optimal [GKS92]. It actually even extends to d dimensions. In this case the expected time and space complexity is $O(n^{\lceil \frac{d}{2} \rceil})$ [DMT92a, Dev02]. We describe the details of the incremental algorithm in Section 1.2.4 below.

Divide & Conquer A divide & conquer approach to compute planar Delaunay triangulations has been described by [LS80] and improved by [Dwy87]. It actually computes the Voronoi diagram; the Delaunay triangulation can then be constructed in linear time.

The point set $\mathcal{S}$ is partitioned along the x-axis or the y-axis into two about equally large subsets $\mathcal{S}_1$ and $\mathcal{S}_2$. Then the Voronoi diagrams of $\mathcal{S}_1$ and $\mathcal{S}_2$ are computed recursively. Finally there is a sewing step that computes the Voronoi diagram of $\mathcal{S}$ from the Voronoi diagrams of $\mathcal{S}_1$ and $\mathcal{S}_2$. This algorithm has the optimal worst-case complexity $O(n \log n)$; it does not directly extend to higher dimensions.

Sweeping algorithm This is again an algorithm to compute planar Voronoi diagrams [For87]. It maintains a line that sweeps over the plane and the Voronoi diagram of the already encountered points and the line itself. As the line sweeps the plane, the Voronoi diagram is updated on the fly and new points encountered by the sweep-line are added. This algorithm has optimal worst-case complexity $O(n \log n)$.

1.2.4 The incremental algorithm

The implementation of Delaunay triangulations in the Computational Geometry Algorithms Library CGAL [cga] uses a modified version of the incremental algorithm described by Bowyer [Bow81] and Watson [Wat81]. We now present in more detail the algorithm used by the CGAL implementation [Yvi10, PT10b], on which our algorithm to compute the Delaunay triangulation of different orbit spaces is based.

At first we explain the employed point location strategy together with two optional improvements, one using a point-location data-structure and the other one performing some preprocessing on the point set before constructing the triangulation. Then we present two common solutions to restoring the Delaunay property of the triangulation after having inserted a new point. And finally we show how to remove a vertex from a triangulation.

Point location In order to find the cell that contains a given point, an approach called *remembering stochastic walk* is used [DPT02]. The point location procedure starts at an arbitrary cell c. It chooses a facet f of the cell at random. Let v denote the vertex of c that does not belong to f. If the query point and v lie on different sides of the hyperplane containing f, the point location algorithm proceeds to the neighboring cell on the opposite side of f and does the same again. It remembers the cell that has been visited before in order to save one geometric test. As it chooses f at random amongst the faces of c, it reaches the cell containing the point after a finite number of steps with high probability. Figure 1.8 for an illustration in 2D.

As introduced in Section 1.2.3, several point-location data-structures have been proposed. In CGAL the Delaunay hierarchy [Dev02] is implemented: A logarithmic number of triangulations of gradually smaller subsets of the input point set are maintained. For each of these triangulations there are links between corresponding vertices in the next finer and the next coarser triangulation in the hierarchy. The point location starts in the coarsest triangulation, then goes down to the next finer triangulation and starts the remembering stochastic walk described above at a vertex corresponding to the output of the point location on the coarser triangulation.

If the whole point set is known it advance, it can be ordered in a way that two successive points in the ordering are spatially close. Then the point location can be initialized with

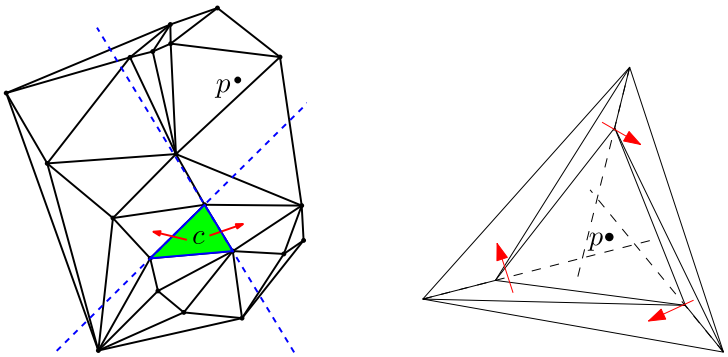


Figure 1.8: Left: We follow one of the arrows chosen at random to get closer to p. Right: Without the random choice of the neighbor to visit the algorithm could end up in an infinite loop in this situation.

the last inserted point. In practice this approach yields very good results. This approach is called *Spatial sorting* [ACR03]. A CGAL implementation for sorting points spatially is available [Del10]; a theoretical analysis can be found in [Buc09].

Point insertion The goal of this step is to restore the Delaunay property when inserting the new point into the triangulation. In the two-dimensional case an approach using so-called *flips* is often used. But it is difficult to extend this approach to higher dimensions [ES96]. We use an alternative approach proposed independently by Bowyer [Bow81] and Watson [Wat81] that extends easily to d dimensions and is well-suited for implementation due to its simplicity.

The point insertion is done in three steps:

1. Identify the cells that must be modified (the *conflict region*).
2. Delete all cells from the conflict region.
3. Fill the hole created in step 2 with cells spanned by the new point and the hole facets.

The conflict region consists of all d-simplices, whose circumscribing balls contain the new point, see Figure 1.10.

To find the conflict region a depth-first search algorithm is used. It starts at the cell containing the new point. For all its neighbors, it is checked whether they belong to the conflict region. If a neighbor is in conflict, all its neighbors but the already visited ones are tested recursively. If a neighbor is not in conflict, the search is stopped.

The conflict region is a star-shaped polytope and the new point is located in its kernel, i.e., the new point can see the whole boundary of the polytope. Furthermore, the boundary of the polytope consists of $(d-1)$-simplices by construction. Thus after deleting the d-simplices in conflict it is possible to fill up the hole by constructing new d-simplices from the $(d-1)$-simplices that are facets of the polytope and the new point. This gives the desired triangulation (see Figure 1.9).

This approach directly extends to the weighted Delaunay triangulation: In this case, the conflict region consists of all d-simplices whose orthogonal spheres are suborthogonal to the sphere centered at the new point and whose squared radius is the weight of the

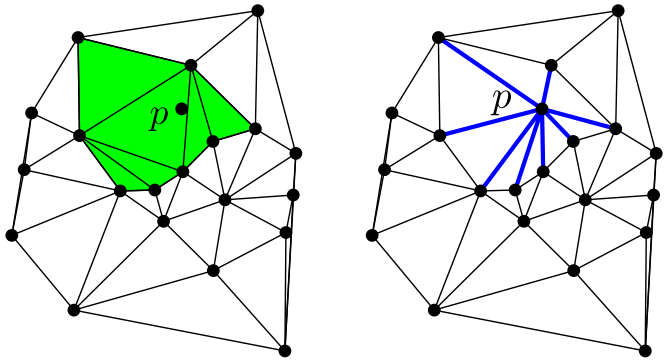


Figure 1.9: Left: The triangles in conflict with p. Right: The star of p in the updated Delaunay triangulation.

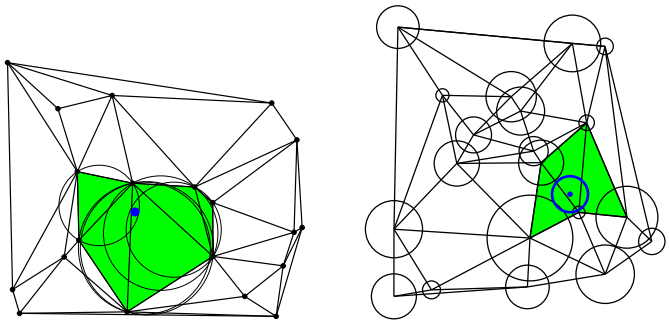


Figure 1.10: Conflict regions for Delaunay triangulation (left) and weighted Delaunay triangulation (right).

new point, see Figure 1.10. The same depth-first search algorithm as for the Delaunay triangulation can be used. The condition for this algorithm to work is that the conflict region must be contractible, which is true for the weighted Delaunay triangulation.

Vertex removal For general triangulations vertex removal is not possible because removing a vertex might leave a hole that cannot be triangulated (e.g. the Schönhardt polyhedron [She98a]).

For the Delaunay triangulation and the weighted Delaunay triangulation a vertex can be removed in the following way: First the vertex and all incident cells are removed from the triangulation. This leaves a hole whose boundary is a polytope with $(d-1)$-simplices as facets. Then the vertices of that polytope are triangulated using the algorithm described above to compute the Delaunay triangulation or the weighted Delaunay triangulation of a point set. The boundary of the hole forms a subcomplex of this triangulation and thus the interior of this polytope can be sewed into the hole.

1.3 Orbit spaces

We now give a short introduction to orbit spaces and other mathematical concepts that will be used throughout the thesis. See [Arm82] for further reading.

A *topological space* is a set $\mathbb{X}$ together with a collection Ω of subsets of $\mathbb{X}$ with the properties

- $\emptyset$ and $\mathbb{X}$ are in Ω
- The union of any collection of sets in Ω is again in Ω.
- The intersection of any finite collection of sets in Ω is again in Ω.

The collection Ω is also called a *topology* on the set $\mathbb{X}$. An element ω of Ω is a *neighborhood* of an element $x \in \mathbb{X}$ if $x \in \omega$. A subset O of $\mathbb{X}$ is called *open* if it is a neighborhood of each of its points. A subset C of $\mathbb{X}$ is called *closed* if its complement is open. A topological space is *discrete* if all elements of $\mathbb{X}$ appear as singleton sets in Ω. Note that in a discrete topological space all the singleton sets are both open and closed. A function f from a topological space $\mathbb{X}$ to a topological space $\mathbb{Y}$ is *continuous* if for each point x of $\mathbb{X}$ and each neighborhood N of $f(x)$ in $\mathbb{Y}$ the preimage $f^{-1}(N)$ is a neighborhood of x in $\mathbb{X}$. Continuous functions are also called *maps*. A *homeomorphism* from a topological space $\mathbb{X}$ to a topological space $\mathbb{Y}$ is a function that is bijective, continuous, and has a continuous inverse. If such a function exists, $\mathbb{X}$ and $\mathbb{Y}$ are called *homeomorphic* or *topologically equivalent*.

We use the notation $\mathcal{G} :=< g_1, \ldots, g_k >$ to denote the group generated by elements $g_1, \ldots, g_k$. Let g, h be elements of $\mathcal{G}$. For two group elements $g, h \in \mathcal{G}$, the group operation that combines g and h is written as gh. A *topological group* is a group together with a topology with the property that both the group operation and the function mapping a group element to its inverse are continuous. A group is *discrete* if it is a topological group with the discrete topology. In this work we only consider *discrete* groups. A group $\mathcal{G}$ is said to be *torsion-free* if the only element of finite order is the identity.

A topological group $\mathcal{G}$ is said to *act* as a group of homeomorphisms on a space $\mathbb{X}$ if its elements are homeomorphisms from $\mathbb{X}$ to $\mathbb{X}$, the group operation is composition of two homeomorphisms, and the following properties are fulfilled:

- $(hg)(x) = h(g(x))$ for all $g, h \in \mathcal{G}$, for all $x \in \mathbb{X}$,
- $e(x) = x$ for all $x \in \mathbb{X}$, where e is the unit element of $\mathcal{G}$,
- the function $\mathcal{G} \times \mathbb{X} \to \mathbb{X}, (g, x) \mapsto g(x)$ is continuous.

Let x be a point of $\mathbb{X}$. The action of $\mathcal{G}$ is *discontinuous* at x if there is a neighborhood U of x such that $\{g \in \mathcal{G} \mid gU \cap U \neq \emptyset\}$ is finite. The action is called discontinuous if it is discontinuous at any $x \in \mathbb{X}$. The set of all images $g(x)$, for all elements g of $\mathcal{G}$, is called the *orbit* of x. From the fact that $\mathcal{G}$ is a group follows that two intersecting orbits are equal, and thus orbits form a partition of $\mathbb{X}$. This induces the following equivalence relation: $x \sim y$ if $x = g(y)$ for some $g \in \mathcal{G}$. The *quotient space* $\mathbb{X}/\mathcal{G}$ is the set of all orbits of $\mathbb{X}$ under the action of $\mathcal{G}$; from now on we use the term *orbit space* for $\mathbb{X}/\mathcal{G}$ because it is more commonly used in geometric contexts and emphasizes the fact that we consider a space of orbits. We call $\mathbb{X}/\mathcal{G}$ the orbit space of $\mathbb{X}$ under the discontinuous action of $\mathcal{G}$. We abuse the denomination by using the following short expressions to denote $\mathbb{X}$ and $\mathcal{G}$ when there are no ambiguities: $\mathbb{X}$ is the *underlying space of* $\mathbb{X}/\mathcal{G}$ and $\mathcal{G}$ is the *group of action defining* $\mathbb{X}/\mathcal{G}$. Let $g \in \mathcal{G}$ and $p \in \mathbb{X}$. We use the short notation gp to denote $g(p)$.

A *fundamental domain* of $\mathcal{G}$ is a subset of $\mathbb{X}$ that contains at least one point of each orbit. One often requires the fundamental domain to be connected and closed and to contain not more than one point of the same orbit in its interior.

We furthermore need the concept of *covering spaces* defined as follows.

Definition 1.3.1 (Covering space)**.** *Let $\mathbb{X}$ be a topological space. A map $\rho : \widetilde{\mathbb{X}} \to \mathbb{X}$ is called a* covering map *and $\mathbb{X}$ is said to be a* covering space *of $\mathbb{X}$ if the following condition holds: For each point $x \in \mathbb{X}$ there is an open neighborhood V, and a decomposition of $\rho^{-1}(V)$ as a family $\{U_\alpha\}$ of pairwise disjoint open subsets of $\widetilde{\mathbb{X}}$, in such a way that $\rho|_{U_\alpha}$ is a homeomorphism for each α. Let h_x denote the cardinality of the family $\{U_\alpha\}$ corresponding to some $x \in \mathbb{X}$. If the maximum $h := \max_{x\in\mathbb{X}} h_x$ is finite, then $\widetilde{\mathbb{X}}$ is called an h-*sheeted covering space *of $\mathbb{X}$.*

A covering space $\widetilde{\mathbb{X}}$ of $\mathbb{X}$ is said to be a *universal* covering space if it is a covering space of all covering spaces of $\mathbb{X}$. For example, a 2-sheeted covering space of a circle is again a circle, see Figure 1.11. The real line is a universal covering space of a circle.

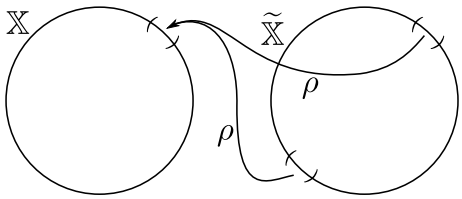


Figure 1.11: Left: A circle. Right: A 2-sheeted covering space.

A space is a *manifold of dimension d* or *d-manifold* if each of its points has a neighborhood homeomorphic to $\mathbb{E}^d$. Note that a torsion-free group does not have fixed points, i.e., $gp \neq hp$ holds for all $p \in \mathbb{E}^d$, $g, h \in \mathcal{G}$ and $g \neq h$. If $\mathbb{X}$ is a manifold and if $\mathcal{G}$ is discrete and torsion-free, then $\mathbb{X}/\mathcal{G}$ is a manifold. In this case the action of $\mathcal{G}$ is also called *properly discontinuous*. If $\mathcal{G}$ is not torsion-free, then $\mathbb{X}/\mathcal{G}$ can have points whose neighborhood is not homeomorphic to $\mathbb{E}^d$, so-called singular points. In this case $\mathbb{X}/\mathcal{G}$ is an *orbifold*, which is a generalization of manifolds [Thu02, BMP03].

When we talk about curvature of manifolds, we mean Riemannian curvature, which is a generalization of Gaussian curvature to d-dimensional manifolds [dC92]. We consider only orbit spaces whose underlying spaces have constant curvature.

1.4 Problem statement

The goal of this work is to give a definition of the Delaunay triangulation of different orbit spaces and to devise efficient algorithms to compute it.

One orbit space we consider is the so-called flat torus. The flat torus can be generated by identifying opposite edges of a square, see Figure 1.12.

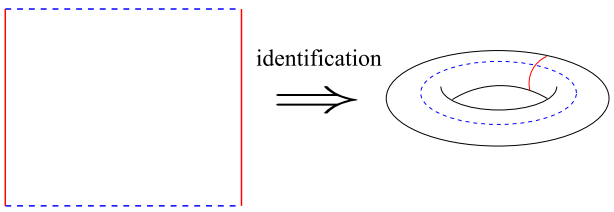


Figure 1.12: Identifying opposite edges of a square yields a torus.

A triangulation of the flat torus is periodic by construction: It can be replicated periodically in x- and y-direction in $\mathbb{E}^2$, see Figure 1.13. In Chapter 2, we describe in more

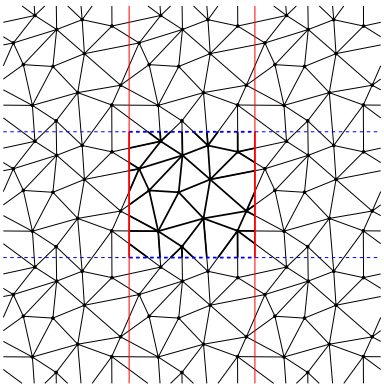


Figure 1.13: A periodic triangulation.

detail how to consider the flat torus as an orbit space.

If the input point set is too small or not well-distributed, it does not define a triangulation. This means that the partition of the flat torus into vertices, segments and triangles is not a simplicial complex and followingly not a triangulation, see Figure 1.14.

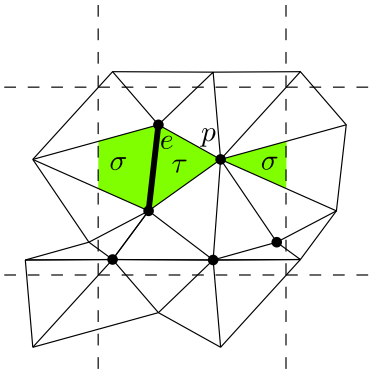


Figure 1.14: This is not a simplicial complex because the intersection of σ and τ is $p \cup e$, which is not a simplex.

Let us insist here on the fact that computing a triangulation, i.e. a *simplicial complex*, is important for several reasons. First, a triangulation is defined as a simplicial complex in the literature [Arm82, DS02, GGL95, Hen79, Lee00, Spa66, Zom05]. Moreover, designing a data structure to efficiently store tessellations that are non-simplicial complexes (e.g. Δ-complexes [Hat01]) would be quite involved. The CGAL 3D triangulation data structure, that we reuse in our implementation, assumes the structure to be a simplicial complex [PT10a]. Even more importantly, algorithms using a triangulation as input rely on the fact that the triangulation is a simplicial complex; this is the case for instance for meshing algorithms [RY07, RY10], as well as algorithms to compute α-shapes, which are actually needed in the periodic case by several applications mentioned at the beginning of this introduction.

One approach to resolve issues arising when computing the Delaunay triangulation of

an orbit space is to use several explicit copies of the input point set. Several examples where this approach is used are presented in Section 1.5. We want to give a number of copies that is sufficiently large to obtain a simplicial complex but small enough such that an implementation of an algorithm is still useful in practice. Such a number depends on both the orbit space and the input point set. We also aim at working without additional copies whenever possible.

1.5 State of the art

In this section we describe existing work on triangulations in spaces other than $\mathbb{E}^d$, especially work in relation to orbit spaces. At first we consider work discussing Delaunay triangulations of different spaces of constant curvature. Then we turn to discussing software packages, and finally we consider papers that treat the problem for more general spaces and in more general settings.

Grima and Márquez describe how to compute triangulations of point sets on surfaces [GM01]. They explicitly treat the cases of the cylinder, the sphere, and the torus. They do not discuss the Delaunay triangulation.

Given a set of points $\mathcal{S}$ on a surface, they define a "pseudo-triangulation" as a maximum set of *segments*, which are the shortest geodesics connecting two points from $\mathcal{S}$ such that no two segments intersect. They do not consider degenerate cases but implicitly assume geodesics to be uniquely defined. The segments of the "pseudo-triangulation" partition the surface into polygonal regions. The authors prove the following results, see also Figure 1.15:

Cylinder If the input points are not all contained in one half-cylinder, then the "pseudo-triangulation" partitions the cylinder into two unbounded regions and many bounded regions that are all triangles.

Sphere If the input points are not all contained in one half-sphere, then the "pseudo-triangulation" partitions the sphere into triangles; this partition is a triangulation.

Torus If at least one quadrant of the fundamental square of the flat torus does not contain any input point, then the "pseudo-triangulation" is not a triangulation.

They do not give an algorithm for computing triangulations of the torus.

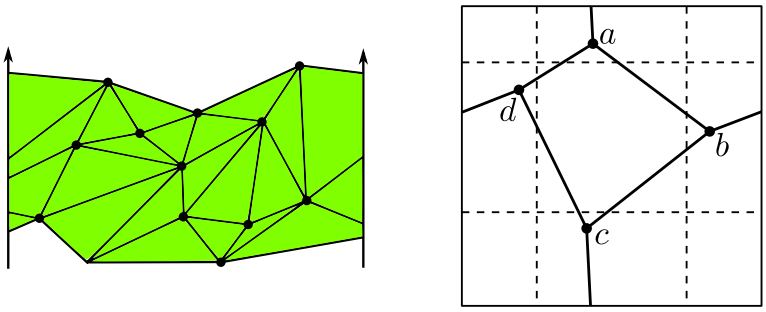


Figure 1.15: Left: Triangulation of the cylinder. Right: "Pseudo-triangulation" of the torus; no more shortest geodesics can be added without introducing intersections. [GM01]

Mazón and Recio [MR97] discuss the computation of Voronoi diagrams on orbifolds. The dual structure of the Voronoi diagram has the Delaunay property, but is not necessarily a simplicial complex. They only consider a quite restricted class of orbifolds: Orbit spaces of $\mathbb{E}^2$ and $\mathbb{S}^2$ under the action of discrete groups of isometries. In the Euclidean case these are the wallpaper groups, the point groups and the Frieze groups, in the spherical case the point groups only [CM57]. Let $\mathcal{G}$ be one such group and $\mathcal{S}$ the input point set. If $\mathcal{G}$ is infinite, then $\mathcal{GS}$, the set of all points in the orbits of the points of $\mathcal{S}$, is infinite. Let $VD(\mathcal{S})$ denote the Voronoi diagram of $\mathcal{S}$ in $\mathbb{E}^2$ or $\mathbb{S}^2$. The authors show that for any $\mathcal{G}$ there is a finite point set $\mathcal{S}^* \subset \mathcal{GS}$ and a fundamental domain $\mathcal{F}$, such that $VD(\mathcal{S}^*) \cap \mathcal{F} = VD(\mathcal{GS}) \cap \mathcal{F}$. This means that $VD(\mathcal{S}^*) \cap \mathcal{F}$ under the action of $\mathcal{G}$ is the infinite Voronoi diagram of $\mathcal{GS}$, see Figure 1.16. The goal here is to compute such a basic

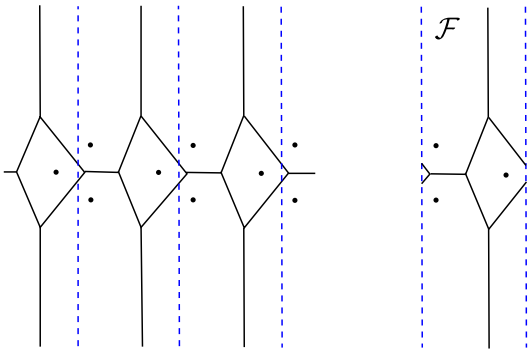


Figure 1.16: Left: Voronoi diagram of three points on a cylinder. Right: One basic building block. Source [MR97]

building block. The authors prove that in the worst case (i.e. for the worst groups) $\mathcal{S}^*$ consists of up to 37 copies of the point set $\mathcal{S}$.

In this approach, the computation of the Voronoi diagram is done in $\mathbb{E}^2$ or $\mathbb{S}^2$, respectively. Thus any of the well-known algorithms of complexity $O(n \log n)$ can be used. As the number of points in $\mathcal{S}^*$ is at most 37 times the number of points in $\mathcal{S}$ this does not change the asymptotic behavior of the algorithm, but it does not appear very practical either, given that the algorithm is slowed down by a factor of at least 37 in the worst case.

Aanjaneya and Teillaud [AT07] examine triangulations of the projective plane, which is an orbit space of $\mathbb{S}^2$ under the action of the group identifying two antipodal points. They use an incremental algorithm and start with an initial triangulation of six of the input points. They represent the triangulation and perform the computations in the projective plane itself. They solve the problems arising due to the fact that the projective plane is not orientable. The triangulations they compute do not have the Delaunay property.

Dolbilin and Huson [DH97b] show that in the case of the three-dimensional flat torus a periodic Delaunay triangulation can be extracted from a Delaunay triangulation of 27 copies of the point set. In Section 4.4.3, we present a part of their proof in more detail. They do not discuss whether the number of required copies can be reduced depending on the input point set. They do not work in the flat torus but consider a fundamental domain given by the Voronoi cell of a single point in the torus. Then they compute the Delaunay triangulation of the convex hull of the input point set inside the fundamental

domain together with its copies in the 26 adjacent fundamental domains in $\mathbb{E}^3$. Their algorithm works in three steps:

1. Let $\mathcal{S}$ be the input set of points in the fundamental cube. Compute the set $\mathcal{S}^*$ of 27 copies of each point in $\mathcal{S}$.
2. Compute the Delaunay triangulation of $\mathcal{S}^*$ in $\mathbb{E}^3$.
3. Extract the simplices from the Delaunay triangulation of $\mathcal{S}^*$ that are part of the infinite periodic Delaunay triangulation.

The output of this algorithm is a set of tetrahedra whose union is a fundamental domain of the three-dimensional flat torus.

Again, as this approach actually computes a Delaunay triangulation in $\mathbb{E}^3$, it can use any of the known algorithms. However, the number of input points to the algorithm is 27 times larger than the number of points in $\mathcal{S}$. This does not change the theoretical complexity of the algorithm, but for practical use it would be nice to reduce the number of required copies of the input points.

Software

In the engineering community, Thompson proposed an algorithm for computing Delaunay triangulations of the three-dimensional flat torus avoiding duplications of points [Tho02]. However, the algorithm heavily relies on the assumption that the input point set is sufficiently large and well-distributed. It starts with an initial "triangulation" of eight vertices close to the corners of the fundamental cube. This initial structure is not a simplicial complex and it is not required to have the Delaunay property. Points are added iteratively using several techniques in order to find inconsistencies due to floating-point errors and work around them [BL95]. The claim is that if the input point set is sufficiently large and well-distributed, then the final output will be a Delaunay triangulation of the three-dimensional flat torus. The claim is supported by experiments on well-distributed data. No proofs are provided. The experiments exhibit a running time of about $O(n^{1.11})$. Unfortunately, the code is not available, so we could not compare it to our implementation.

John M. Sullivan wrote the software package VCS for computing both periodic and non-periodic Voronoi diagrams [Sul]. However, it dates back to 1988 and is not maintained by the author anymore. In case of periodic Voronoi diagrams it works with 27 copies of each input point.

Furthermore, there are other implementations from researchers, adapted to their specific use: For instance, Volker Springel implemented a parallelized version of a two-dimensional periodic mesh generator to run experiments on fluid dynamics [Spr10, Spr09]. It is highly adapted to the used hardware and it is not intended to be available for public use.

More general spaces

Some work on triangulations in more general spaces have been published:

Edelsbrunner and Shah discuss triangulations in general topological spaces [ES97]. However, they assume the space to be embedded in $\mathbb{E}^d$ and they suppose that they are given a sufficiently well-sampled point set. Then they compute the Delaunay triangulation

of the point set in $\mathbb{E}^d$ and define the triangulation of the given topological space to be a restriction of the triangulation in $\mathbb{E}^d$. In his way the restricted Delaunay triangulation is a subcomplex by construction and thus a triangulation. The actual discussion aims at determining whether the restricted triangulation is homotopy equivalent to the given topological space. This topic has been subject to extensive research, see for instance [BO05, Che93, Rup95, She00].

Delgado Friedrichs and Huson discuss orbifold triangulations and crystallographic groups [DH97a]. Their problem is in some sense inverse to the problem we want to discuss: Given a triangulation of a space, find the corresponding space group, i.e. the group $\mathcal{G}$, such that the given triangulation is a triangulation of the orbit space $\mathbb{E}^3/\mathcal{G}$.

1.6 Contributions

We give a mathematically sound definition of the Delaunay triangulation of the three-dimensional flat torus defined by a set of points. We propose a consistent approach of treating point sets that do not define a periodic Delaunay triangulation using covering spaces. We prove conditions to decide whether a periodic Delaunay triangulation of a covering space of the flat torus can be converted to a Delaunay triangulation of the flat torus itself and give simple geometric criteria to verify these conditions.

We present an adaptation of the well-known incremental algorithm in $\mathbb{E}^3$ [Bow81, Wat81] that allows to compute Delaunay triangulations of the three-dimensional flat torus. We focus on the incremental algorithm for several reasons: Its practical efficiency has been proved in particular by the fully dynamic implementation in CGAL [PT10b]; moreover, a dynamic algorithm, allowing to freely insert and remove points, is a necessary ingredient for all meshing algorithms and software based on Delaunay refinement methods (see for instance [She98b, RY07, CDL07]). We extend the above work to different classes of triangulations such as weighted Delaunay triangulations.

We also provide an implementation of the algorithm. The software has been reviewed and accepted by the CGAL Editorial Board for inclusion in version 3.5 of the CGAL library [cga]. This software was demonstrated by a video [CT08]. We are in contact with users of our implementation and there is at least one publication referring it [Sou10].

We discuss extensions that allow for using our code in combination with other CGAL components such as the alpha shape package [DY10] or the surface mesh and volume mesh generators [RY10, RTY10].

We extend parts of our work on Delaunay triangulations of the flat torus to different classes of spaces, such as orbit spaces of the d-dimensional Euclidean space $\mathbb{E}^d$ and the d-dimensional sphere $\mathbb{S}^d$. We present some preliminary ideas on how to extend it also to orbit spaces of the hyperbolic space $\mathbb{H}^d$.

The thesis is organized as follows. In Chapter 2, we discuss periodic triangulations in detail. We give a precise definition as well as an algorithm and prove several important properties. In Chapter 3, we describe the implementation in CGAL, show some experiments and applications. Chapter 4 deals with extending the theory developed in Chapter 2 onto further flat orbit spaces as well as spherical and hyperbolic orbit spaces.

Chapter 2

3D Periodic triangulations

In this chapter, we first show how to define the flat torus as an orbit space. In the main part of the chapter we show how to extend the definition of a Delaunay triangulation to the three-dimensional flat torus and present an algorithm. We extend the algorithm to weighted Delaunay triangulations. We finally prove the correctness of the algorithm and show that it has randomized worst-case optimal complexity of $O(n^2)$.

2.1 The flat torus

At first we give a precise definition of the flat torus that we will consider throughout the rest of the chapter. Then we review some of its well-known properties and establish notation. Finally, we give a definition of simplices in the flat torus.

Let t_x, t_y, and t_z denote the unit translations along the x-, y-, and z-axis in $\mathbb{E}^3$, respectively.

Definition 2.1.1 ($\mathbb{T}^3_{\boldsymbol{c}}$). *Let* $\boldsymbol{c} := (c_x, c_y, c_z) \in (\mathbb{R}\backslash\{0\})^3$ *and* $\mathcal{G}$ *be the group* $< c_x t_x, c_y t_y, c_z t_z >$. *The orbit space* $\mathbb{E}^3/\mathcal{G}$ *is a flat torus and we denote it by* $\mathbb{T}^3_{\boldsymbol{c}}$. *We furthermore denote the projection map by* $\boxed{\pi : \mathbb{E}^3 \to \mathbb{T}^3_{\boldsymbol{c}}}$.

Another commonly used denomination is *Euclidean torus* [Thu97]. Here, flat or Euclidean means that the space has a constant Riemannian curvature zero. It is called torus because it is homeomorphic to the three-dimensional torus embedded in $\mathbb{E}^4$.

The flat torus $\mathbb{T}^3_{\boldsymbol{c}}$ that we consider here has a d-cuboid as fundamental domain. For general flat tori the fundamental domain is not restricted to cuboids but can be a d-dimensional parallelepiped. The general case is treated in Section 4.2.

Note that the orbits of $\mathcal{G}$ are isomorphic to $\mathbb{Z}^3$ and $\mathbb{T}^3 \times \mathbb{Z}^3$ is isomorphic to $\mathbb{E}^3$. $\mathbb{T}^3$ is a metric space with $\mathrm{dist}_{\mathbb{T}}(\pi(p), \pi(q)) := \min \mathrm{dist}(p', q')$ for $p' \sim p, q' \sim q$. Note that π is continuous.

Consider the closed cuboid $[u, u + c_0] \times [v, v + c_1] \times [w, w + c_2]$, which is a fundamental domain for $\mathcal{G}$. The half-open cuboid $\boxed{\mathcal{D}_{\boldsymbol{c}} = [0, c_x) \times [0, c_y) \times [0, c_z)}$ contains exactly one representative of each element of $\mathbb{T}^3_{\boldsymbol{c}}$. We call it *original domain*. We denote coordinate-wise vector multiplication by $*$, i.e. $(a_x, a_y, a_z) * (b_x, b_y, b_z) := (a_x b_x, a_y b_y, a_z b_z)$. Note that this can also be understood as a matrix multiplication where the a and b are the entries

on the diagonals. The map

$$\begin{aligned} \varphi_{\boldsymbol{c}} : \mathcal{D}_{\boldsymbol{c}} \times \mathbb{Z}^3 &\to \mathbb{E}^3 \\ (p, \zeta) &\mapsto p + \boldsymbol{c} * \zeta \end{aligned}$$

is bijective. The longest diagonal of $\mathcal{D}_{\boldsymbol{c}}$ has length $\|\boldsymbol{c}\|$. We say that two points $p_1, p_2 \in \mathbb{E}^3$ are *periodic copies* of each other if they both lie in the same orbit, or equivalently if there is a point $p \in \mathcal{D}_{\boldsymbol{c}}$ such that $p_1, p_2 \in \varphi_{\boldsymbol{c}}(\{p\} \times \mathbb{Z}^3)$.

Now we turn towards the definition of simplices in $\mathbb{T}^3_{\boldsymbol{c}}$. There is no meaningful definition of a convex hull in $\mathbb{T}^3_{\boldsymbol{c}}$ and a tetrahedron is not uniquely defined by four points. We attach with each vertex an integer vector, named *offset*, that specifies one representative out of an orbit (see Figure 2.1). Intuitively, the offsets determine which way a simplex "wraps around" the torus. In the above definition of $\varphi_{\boldsymbol{c}}$, the offsets are the numbers $\zeta \in \mathbb{Z}^3$.

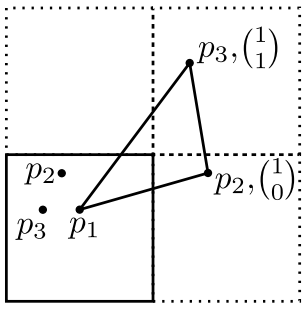


Figure 2.1: (2D illustration) The three points p_1, p_2, and p_3 do not uniquely define a triangle. The drawn triangle is the triangle $(p_1, \binom{0}{0})$, $(p_2, \binom{1}{0})$, $(p_3, \binom{1}{1})$.

As seen in Section 1.2.1 the simplicial complex definition is purely combinatorial and does not depend on the space. Only the simplex definition must be adapted to $\mathbb{T}^3_{\boldsymbol{c}}$. This can be done in a similar way as in [Wil08]:

Definition 2.1.2 (k-simplex in $\mathbb{T}^3_{\boldsymbol{c}}$)**.** *Let $\mathcal{P}$ be a set of $k+1$ ($k \leq 3$) point-offset pairs (p_i, ζ_i) in $\mathcal{D}_{\boldsymbol{c}} \times \mathbb{Z}^3$, $0 \leq i \leq k$. Let* $\mathrm{Ch}(\mathcal{P})$ *denote the convex hull of $\varphi_{\boldsymbol{c}}(\mathcal{P}) = \{p_i + \boldsymbol{c} * \zeta_i \mid 0 \leq i \leq k\}$ in $\mathbb{E}^3$. If the restriction $\pi|_{\mathrm{Ch}(\mathcal{P})}$ of π to the convex hull of $\mathcal{P}$ is injective, the image of* $\mathrm{Ch}(\mathcal{P})$ *by π is called a k-*simplex *in $\mathbb{T}^3_{\boldsymbol{c}}$.*

In other words, the image under π of a simplex in $\mathbb{E}^3$ is a simplex in $\mathbb{T}^3_{\boldsymbol{c}}$ only if it does not self-intersect or touch. Figure 2.2 shows the convex hulls A, B, and C of three point-offset pairs in $[0,1)^2 \times \mathbb{Z}^2$, respectively; $\left(p_1, \binom{0}{2}\right)$ is a point in the orbit of a vertex of A that lies *inside* A. All three vertices of B are in the same orbit.

There are infinitely many sets of point-offset pairs specifying the same simplex. The definition of face and coface is adapted accordingly: Let σ be a k-simplex defined by a set $\mathcal{P}_\sigma \subseteq \mathcal{D}_{\boldsymbol{c}} \times \mathbb{Z}^3$. A simplex τ defined by a set $\mathcal{P}_\tau \subseteq \mathcal{D}_{\boldsymbol{c}} \times \mathbb{Z}^3$ is a *face* of σ and has σ as a *coface* if and only if there is some $\zeta \in \mathbb{Z}^3$ such that $\{(p_i, \zeta_i + \zeta) \mid (p_i, \zeta_i) \in \mathcal{P}_\tau\} \subseteq \mathcal{P}_\sigma$.

2.2 Delaunay triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$

This section is organized as follows: At first we give a definition of the Delaunay triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$. We observe that there are point sets in $\mathbb{T}^3_{\boldsymbol{c}}$ that do not define a Delaunay

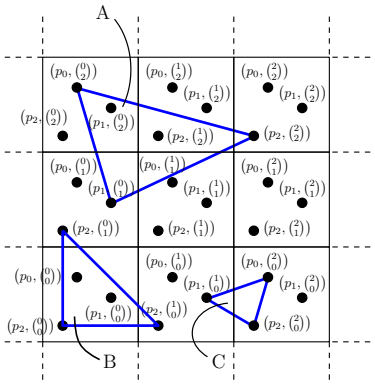


Figure 2.2: (2D illustration) $\pi(A)$ and $\pi(B)$ are not simplices; however, $\pi(C)$ is a simplex.

triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$. Then we give a necessary and sufficient condition for a point set to define a Delaunay triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$ (Theorem 2.2.8). In the second part we discuss how to deal with point sets that do not define a Delaunay triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$.

2.2.1 Definition

Let us recall that a triangulation of a point set $\mathcal{S}$ in $\mathbb{E}^3$ is a simplicial complex with vertex set $\mathcal{S}$. It is a *Delaunay* triangulation if and only if each tetrahedron satisfies the Delaunay property, i.e., its circumscribing ball does not contain any point of $\mathcal{S}$ in its interior. From now on we always assume Delaunay triangulations of $\mathbb{E}^3$ to be uniquely defined, e.g. by using symbolic perturbation as described in Section 1.2.2.

We want to define the Delaunay triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$ for a given point set $\pi(\mathcal{S})$. The idea is to use the projection under π of a Delaunay triangulation of $\mathbb{E}^3$ defined by the infinite periodic point set $\boxed{\mathcal{GS} := \varphi_{\boldsymbol{c}}(\mathcal{S} \times \mathbb{Z}^3)}$. Without loss of generality, we can assume that the points of $\mathcal{S}$ lie in $\mathcal{D}_{\boldsymbol{c}}$.

From now on, (i), (ii) and (iii) always denote the three conditions of Definition 1.2.2 (Simplicial complex).

Lemma 2.2.1. *For any finite point set $\mathcal{S} \subset \mathcal{D}_{\boldsymbol{c}}$, a set of simplices $\mathcal{K}$ in $\mathbb{E}^3$ that fulfills (i), (ii), and the Delaunay property with respect to $\mathcal{GS}$, is a simplicial complex in $\mathbb{E}^3$.*

Proof. We need to show that $\mathcal{K}$ has the local finiteness property (iii).

Assume that there is a vertex v with an infinite number of incident simplices and thus an infinite number of incident edges. Since $\mathcal{S}$ contains only a finite number of points, there must be at least one point q in $\mathcal{S}$ of which infinitely many periodic copies are adjacent to v. The periodic copies of q form a grid, in which the diameter of the largest empty ball is bounded by $\|\boldsymbol{c}\|$. So circumscribing balls of tetrahedra that are cofaces of edges between v and periodic copies of q that are further away than $\|\boldsymbol{c}\|$ cannot be empty. This is a contradiction and hence all vertices in $\mathcal{K}$ are incident to only a finite number of simplices.

Let us now consider a point p in $\mathbb{E}^3$ that is not a vertex in $\mathcal{K}$. If it lies in the interior of a tetrahedron, then it has a neighborhood that intersects only one simplex. If it lies in the interior of a triangle, then it has a neighborhood that intersects three simplices: the triangle and the two incident tetrahedra. Assume now that p lies in the interior of an edge,

then it has a neighborhood intersecting only the triangles and tetrahedra that are incident to the two endpoints of the edge. According to the above discussion, these are only finitely many. □

Since $\mathcal{GS}$ contains points on an infinite grid, any point $p \in \mathbb{E}^3$ is contained in some simplex defined by points in $\mathcal{GS}$. Together with Lemma 2.2.1, this implies that the set of all simplices with points of $\mathcal{GS}$ as vertices and respecting the Delaunay property is a Delaunay triangulation of $\mathbb{E}^3$ and we denote it by $DT(\mathcal{GS})$. Using π we can now define the Delaunay triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$.

Definition 2.2.2 (Delaunay triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$)**.** *Let $DT(\mathcal{GS})$ be a Delaunay triangulation of $\mathcal{GS}$ in $\mathbb{E}^3$. If $\pi(DT(\mathcal{GS}))$ is a simplicial complex in $\mathbb{T}^3_{\boldsymbol{c}}$, then we call it a* Delaunay triangulation *of $\mathbb{T}^3_{\boldsymbol{c}}$ defined by $\mathcal{S}$ and we denote it by $DT_{\mathbb{T}}(\mathcal{S})$.*

There are point sets for which $\pi(DT(\mathcal{GS}))$ is not a simplicial complex, see Figure 1.14 on page 14.

The rest of this section is devoted to studying when $\mathcal{S}$ actually defines a Delaunay triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$. We show that Definition 2.2.2 actually makes sense: We verify that the simplices "match" under π, i.e., that all periodic copies of a simplex in $DT(\mathcal{GS})$ are mapped onto the same simplex in $\mathbb{T}^3_{\boldsymbol{c}}$ under π. We also prove that if $\pi(DT(\mathcal{GS}))$ is a set of simplices, then it fulfills conditions (i) and (iii). Finally, we discuss under which circumstances condition (ii) is fulfilled, which yields the necessary and sufficient condition on $\pi(DT(\mathcal{GS}))$ to be a triangulation, i.e., by Definition 1.2.3 it is a simplicial complex whose union is homeomorphic to $\mathbb{T}^3_{\boldsymbol{c}}$.

Let us start with the first lemma:

Lemma 2.2.3. *If the restriction of π to any simplex in $DT(\mathcal{GS})$ is injective, then $\pi(DT(\mathcal{GS}))$ is a set of internally disjoint simplices in $\mathbb{T}^3_{\boldsymbol{c}}$ that do not contain any point of $\pi(\mathcal{S})$ in their interior.*

Proof. Consider a tetrahedron σ of $DT(\mathcal{GS})$, whose vertices are a four-tuple of points $\mathcal{P}_\sigma \subset \mathcal{GS}$. σ satisfies the Delaunay property, so all periodic copies $\varphi_{\boldsymbol{c}}(\mathcal{P}_\sigma \times \mathbb{Z}^3)$ also have an empty circumscribing ball. This shows that all these periodic copies form tetrahedra of $DT(\mathcal{GS})$.

Note that this is even true in degenerate cases: If we handle degeneracies as in [DT03], then the Delaunay triangulation of a set of cospherical points only depends on their lexicographic order. As translating the set of points does not change their lexicographic order, all periodic copies of that point set are triangulated in the same way.

Followingly, π collapses precisely all the periodic copies of σ onto its equivalence class in $\mathbb{T}^3_{\boldsymbol{c}}$. As any lower-dimensional simplex in $DT(\mathcal{GS})$ is incident to some tetrahedron, and thus is defined by a subset of its vertices, the same holds for simplices of any dimension.

Now the projections under π of two internally disjoint k-dimensional simplices σ and τ in $DT(\mathcal{GS})$ are either equal or internally disjoint for $k \geq 1$, due to the bijectivity of π between both simplices and their respective images. The same argument implies that the interior of a simplex cannot contain any vertex. □

We observe that $\pi(DT(\mathcal{GS}))$ is finite: $DT(\mathcal{GS})$ is locally finite (Lemma 2.2.1), i.e., the star of any vertex is finite. As $\mathcal{S}$ is discrete also $\mathcal{GS}$ is discrete and all tetrahedra have a certain volume larger than some constant. Followingly, there are only finitely many

tetrahedra necessary to fill the original domain $\mathcal{D}_{\boldsymbol{c}}$ and thus $\mathbb{T}^3_{\boldsymbol{c}}$. Finitely many tetrahedra have only finitely many faces so the overall number of simplices in $\pi(DT(\mathcal{GS}))$ is finite, too.

So far we know that if all simplices in $DT(\mathcal{GS})$ are mapped as simplices onto $\mathbb{T}^3_{\boldsymbol{c}}$, then the whole triangulation is mapped onto a set of simplices in $\mathbb{T}^3_{\boldsymbol{c}}$. We now consider the incidence relation.

Observation 2.2.4. *Assume that the restriction of π to any simplex in $DT(\mathcal{GS})$ is injective. If τ is a simplex in $\pi(DT(\mathcal{GS}))$ and $\tau' \leq \tau$, then τ' is a simplex in $\pi(DT(\mathcal{GS}))$. This follows immediately from the fact that incidence relations are maintained by π and from Lemma 2.2.3.*

It only remains to show condition (ii), i.e., the intersection of two simplices σ and τ in $\pi(DT(\mathcal{GS}))$ is another simplex χ that is incident to both σ and τ.

Lemma 2.2.5. *Assume that the restriction of π to any simplex in $DT(\mathcal{GS})$ is injective. Let $\sigma, \tau \in \pi(DT(\mathcal{GS}))$ be any two simplices in $\mathbb{T}^3_{\boldsymbol{c}}$, then $\sigma \cap \tau$ is a set of simplices in $\pi(DT(\mathcal{GS}))$.*

Proof. Without loss of generality, we assume that $\sigma \cap \tau \neq \emptyset$. We show that $\sigma \cap \tau = \bigcup_{p \in \sigma \cap \tau} \chi_p$, where χ_p is a simplex in $\pi(DT(\mathcal{GS}))$. The union is finite because there are only finitely many simplices in $\pi(DT(\mathcal{GS}))$. Consider a point $p \in \sigma \cap \tau$. If p is a vertex of $\pi(DT(\mathcal{GS}))$, then it is not contained in the interior of any other simplex, according to Lemma 2.2.3, and we set $\chi_p = \{p\}$. If p is not a vertex in $\pi(DT(\mathcal{GS}))$, then $p \in \mathring{\sigma}'$ and $p \in \mathring{\tau}'$ for some proper faces $\sigma' \leq \sigma$ and $\tau' \leq \tau$ because σ and τ are internally disjoint (Lemma 2.2.3). Since σ' and τ' are again either internally disjoint or identical, it follows that they are the same face and we set $\chi_p := \sigma' = \tau'$. By condition (i) the simplex χ_p is contained in $\pi(DT(\mathcal{GS}))$. □

Remember that $\bigcup \mathrm{St}(v)$ denotes the union of the simplices in the star of v. We can now formulate the following sufficient condition for $\pi(DT(\mathcal{GS}))$ to be a simplicial complex:

Lemma 2.2.6. *If for all vertices v of $DT(\mathcal{GS})$ the restriction of the projection map $\pi|_{\bigcup \mathrm{St}(v)}$ is injective, then $\pi(DT(\mathcal{GS}))$ forms a simplicial complex.*

Proof. We set $\mathcal{K} = \pi(DT(\mathcal{GS}))$. Let σ be a simplex of $DT(\mathcal{GS})$ and v an incident vertex. Then $\sigma \subseteq \bigcup \mathrm{St}(v)$, thus the restriction of $\pi|_{\bigcup \mathrm{St}(v)}$ to σ is injective as well, and $\mathcal{K}$ is a set of simplices (Lemma 2.2.3).

Conditions (i) and (iii) follow from the above discussion. It remains to show condition (ii): Consider two simplices $\sigma, \tau \in \mathcal{K}$ with $\sigma \cap \tau \neq \emptyset$. By definition of a simplex, there exist sets $\mathcal{P}_\sigma, \mathcal{P}_\tau$ in $\mathcal{D}_{\boldsymbol{c}} \times \mathbb{Z}^3$ such that $\sigma = \pi(\mathrm{Ch}(\mathcal{P}_\sigma))$ and $\tau = \pi(\mathrm{Ch}(\mathcal{P}_\tau))$. From Lemma 2.2.5, we know that $\sigma \cap \tau$ is a set of simplices in $\mathcal{K}$. So there exists a vertex $v \in \sigma \cap \tau$ and $\sigma, \tau \in \mathrm{St}(v)$. By assumption $\pi|_{\bigcup \mathrm{St}(v)}$ is injective, so π is injective on σ and τ, and $\sigma \cap \tau = \pi(\mathrm{Ch}(\mathcal{P}_\sigma)) \cap \pi(\mathrm{Ch}(\mathcal{P}_\tau)) = \pi(\mathrm{Ch}(\mathcal{P}_\sigma \cap \mathcal{P}_\tau))$. Also, the restriction of $\pi|_{\bigcup \mathrm{St}(v)}$ to $\mathrm{Ch}(\mathcal{P}_\sigma \cap \mathcal{P}_\tau)$ is injective. So from Definition 2.1.2, it follows that $\sigma \cap \tau$ is a simplex. Since $\sigma \cap \tau \subseteq \sigma, \tau$, we have $\sigma \cap \tau \leq \sigma, \tau$. □

As the last ingredient for the theorem, we need the following lemma that shows that the Delaunay triangulation as defined in Definition 2.2.2 is actually a triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$ in the sense of Definition 1.2.3.

Lemma 2.2.7. *$|\pi(DT(\mathcal{GS}))|$ is homeomorphic to $\mathbb{T}^3_{\boldsymbol{c}}$.*

Proof. By its construction $|DT(\mathcal{GS})| = \mathbb{E}^3$ and π is surjective. Followingly, $\pi(|DT(\mathcal{GS})|)$ is equal to $\mathbb{T}^3_{\boldsymbol{c}}$. Then, the chain of equalities

$$\begin{aligned}\pi(|DT(\mathcal{GS})|) &= \pi\left(\bigcup_{\sigma\in DT(\mathcal{GS})}\sigma\right) \overset{(1)}{=} \pi\left(\bigcup_{\tau\in\pi(DT(\mathcal{GS}))}\pi^{-1}(\tau)\right)\\ &\overset{(2)}{=} \bigcup_{\tau\in\pi(DT(\mathcal{GS}))}\tau = \bigcup_{\sigma\in DT(\mathcal{GS})}\pi(\sigma) = |\pi(DT(\mathcal{GS}))|\end{aligned}$$

holds with the following arguments:

(1) This step just regroups the simplices in a different order but does not change the set (cf. Lemma 2.2.3).

(2) There is only a finite number of elements in $\pi(DT(\mathcal{GS}))$.

□

We are now ready to prove the main theorem of this section, which gives a necessary and sufficient condition for $\pi(DT(\mathcal{GS}))$ to be a triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$. Let us recall that the *1-skeleton* of a simplicial complex is the subcomplex that consists of all edges and vertices.

Theorem 2.2.8. *Assume that the restriction of π to any simplex in $DT(\mathcal{GS})$ is injective. $\pi(DT(\mathcal{GS}))$ is a triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$ if and only if its 1-skeleton does not contain any cycle of length less than or equal to two.*

Proof. We first show the "if" part. We set $\mathcal{K} = \pi(DT(\mathcal{GS}))$. From Lemma 2.2.3 and Observation 2.2.4, we know that $\mathcal{K}$ is a finite set of simplices that fulfills conditions (i) and (iii). Assume that $\mathcal{K}$ is not a simplicial complex. From Lemma 2.2.6 there is a vertex $v \in \mathcal{K}$ for which $\pi|_{\bigcup \mathrm{St}(v)}$ is not injective. This implies the existence of two different points $p, q \in \bigcup \mathrm{St}(v)$ with $\pi(p) = \pi(q)$. Let σ denote the simplex of $\mathcal{K}$ that contains $\pi(p) = \pi(q)$ in its interior. Then there are two different simplices $\sigma'_{\mathbb{E}} \in \pi^{-1}(\sigma)$ and $\sigma''_{\mathbb{E}} \in \pi^{-1}(\sigma)$ containing p and q, respectively. Thus $\sigma'_{\mathbb{E}}$ and $\sigma''_{\mathbb{E}}$ are both elements of $\overline{\mathrm{St}(v)}$. Let u, w be vertices different from v with $u \leq \sigma'_{\mathbb{E}}$ and $w \leq \sigma''_{\mathbb{E}}$ such that $\pi(u) = \pi(w)$. Such vertices u and w always exist because $\pi(\sigma'_{\mathbb{E}}) = \pi(\sigma''_{\mathbb{E}})$. The vertices u, w are also elements of $\overline{\mathrm{St}(v)}$ and thus there are edges (u, v) and (v, w) in $DT(\mathcal{GS})$. From $\pi(u) = \pi(w)$ follows that the projection of (u, v) and (v, w) under π forms a cycle of length two in $\mathbb{T}^3_{\boldsymbol{c}}$, which contradicts the assumption that $\pi|_{\bigcup \mathrm{St}(v)}$ is injective. So $\mathcal{K}$ must be a simplicial complex. Following Definition 2.2.2, we can now note $DT_{\mathbb{T}}(\mathcal{S}) = \pi(DT(\mathcal{GS}))$. Lemma 2.2.7 shows that $DT_{\mathbb{T}}(\mathcal{S})$ is actually a triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$, which concludes the first part of the proof.

Now we consider the "only if" part. There cannot be any cycles of length one because of the assumption that the restriction of π to any simplex in $DT(\mathcal{GS})$ is injective. Assume $\pi(DT(\mathcal{GS}))$ is a simplicial complex containing two edges σ and τ, $\sigma \neq \tau$, that form a cycle of length two. Then $\sigma \cap \tau$ consists of the two endpoints of the segments σ and τ, which do not form a simplex in $\pi(DT(\mathcal{GS}))$, which contradicts condition (ii) of Definition 1.2.2. □

See Figure 2.3 for an illustration of Lemma 2.2.6 and Theorem 2.2.8.

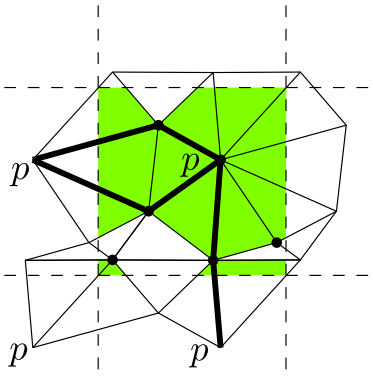


Figure 2.3: (2D illustration) The shaded region is $\varphi_{\boldsymbol{c}}(\mathrm{St}(p) \times \mathbb{Z}^3) \cap \mathcal{D}_{\boldsymbol{c}}$ (cf. Lemma 2.2.6). There are several cycles of length two originating from p (cf. Theorem 2.2.8).

In the proof of Theorem 2.2.8 we have shown that if $\pi|_{\bigcup \mathrm{St}(v)}$ is not injective, then there are cycles of length two in $\pi(DT(\mathcal{GS}))$, which is equivalent to $\pi(DT(\mathcal{GS}))$ not being a simplicial complex. Followingly, this condition is not only sufficient but also necessary in Lemma 2.2.6.

2.2.2 Point sets that do not define a Delaunay triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$

In this section, we explain how we can give a finite representation of the periodic triangulation $DT(\mathcal{GS})$ that is a simplicial complex even if $\pi(DT(\mathcal{GS}))$ is not a simplicial complex. If $\mathcal{S}$ does not define a Delaunay triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$, we compute in a finitely sheeted covering space, see Section 1.3.

Note that $\mathbb{E}^3$ itself with the projection map π as covering map is a *universal covering space* of $\mathbb{T}^3_{\boldsymbol{c}}$, which means that it is a covering space for all covering spaces of $\mathbb{T}^3_{\boldsymbol{c}}$ [Arm82]. However, we cannot use it to compute the Delaunay triangulation because it has an infinite number of sheets. We now construct a finitely sheeted covering space that is sufficiently large so that any point set $\mathcal{P}$ defines a Delaunay triangulation of it.

Let $\boldsymbol{h} = (h_x, h_y, h_z) \in \mathbb{N}^3$. $\mathbb{T}^3_{\boldsymbol{h}*\boldsymbol{c}}$ is a covering space of $\mathbb{T}^3_{\boldsymbol{c}}$ together with the covering map $\rho_{\boldsymbol{h}} := \pi \circ \pi_{\boldsymbol{h}}^{-1}$, where $\pi_{\boldsymbol{h}} : \mathbb{E}^3 \to \mathbb{T}^3_{\boldsymbol{h}*\boldsymbol{c}}$ denotes the projection map of $\mathbb{T}^3_{\boldsymbol{h}*\boldsymbol{c}}$. As $\rho_{\boldsymbol{h}}^{-1}(p)$ for any $p \in \mathbb{T}^3_{\boldsymbol{c}}$ consists of $h_x \cdot h_y \cdot h_z$ different points, $\mathbb{T}^3_{\boldsymbol{h}*\boldsymbol{c}}$ is a $h_x \cdot h_y \cdot h_z$-sheeted covering space. The original domain is $D_{\boldsymbol{h}*\boldsymbol{c}} = [0, h_x c_x) \times [0, h_y c_y) \times [0, h_z c_z)$. If $h_x = h_y = h_z$, we use the notation $\pi_h := \pi_{\boldsymbol{h}}$ with $h := h_x \cdot h_y \cdot h_z$, like for π_{27} in Theorem 2.2.10 below. Note that $\mathbb{T}^3_{\boldsymbol{h}*\boldsymbol{c}}$ is a flat torus again.

Dolbilin and Huson [DH97b] showed that only the points of $\mathcal{GS}$ contained in $\mathcal{D}_{\boldsymbol{c}}$ and the 26 copies that surround it can have an influence on the simplices that are completely contained in $\mathcal{D}_{\boldsymbol{c}}$. We use the ideas of their proof to prove Theorem 2.2.10, so we first give a sketch of their approach. The proof idea is based on considering which parts of the triangulation can be influenced by a point. This is done in three steps using Minkowski sums [dBvKOS00]. Let $F^{(1)}$ denote the Voronoi cell of the origin in the Voronoi diagram of the orbit of the origin under $\mathcal{G}$. Then $F^{(1)}$ is a fundamental domain of $\mathcal{G}$. Let furthermore $\mathcal{Q}$ be a point set in $F^{(1)}$. $A \oplus B$ denotes the Minkowski sum of A and B, and we define $F^{(i)} := F^{(1)} \oplus F^{(i-1)}$.

Lemma 2.2.9 ([DH97b]). *Let σ and τ be simplices in the infinite periodic Delaunay triangulation $DT(\mathcal{GQ})$ such that one of the vertices of σ lies in the center of $F^{(1)}$ and $\tau \cap F^{(1)} \neq \emptyset$ holds. Then*

1. *the center of the circumscribing ball of σ lies in $F^{(1)}$,*
2. *σ is completely contained inside $F^{(2)}$.*
3. *τ is completely contained inside $F^{(3)}$.*

Proof. We briefly sketch the proofs of the three properties.

1. This follows directly from the fact that $F^{(1)}$ is a Voronoi cell of the Voronoi diagram of the orbit of the origin under the action of $\mathcal{G}$: If the center of the circumscribing ball of σ was outside of $F^{(1)}$, then it would be closer to some other point of $\mathcal{GQ}$ contradicting the fact that it is the center of the circumscribing ball of σ.
2. This follows directly from 1, and from the fact that $F^{(2)}$ is the Minkowski sum of $F^{(1)}$ with itself.
3. This follows directly from the definition of $F^{(3)}$.

□

See Figure 2.4 for an illustration of Lemma 2.2.9.

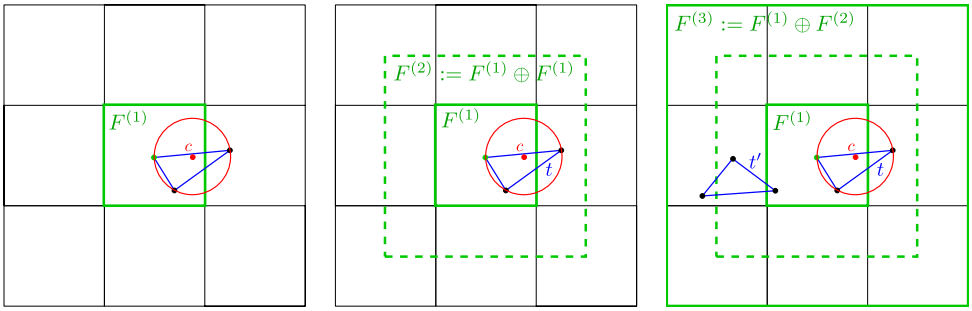


Figure 2.4: (2D illustration) It is sufficient to consider $F^{(3)}$ in order to compute a Delaunay triangulation of the flat torus.

From Lemma 2.2.9 follows that all triangles of the periodic Delaunay triangulation of $\mathcal{Q}$ are contained inside $F^{(3)}$ and so it is sufficient to compute the finite Delaunay triangulation $DT(\mathcal{GQ} \cap F^{(3)})$ in $\mathbb{E}^3$. According to Lemma 2.2.9, all simplices of $DT(\mathcal{GQ} \cap F^{(3)})$ that have at least one vertex in $F^{(1)}$ are simplices of the infinite periodic Delaunay triangulation $DT(\mathcal{GQ})$. Since the set of these simplices covers $F^{(1)}$, applying the action of $\mathcal{G}$ on it yields an infinite periodic partition of $\mathbb{E}^3$. Note that this result directly extends to $\mathbb{E}^d$.

Using the approach of the proof of Lemma 2.2.9, we can show the following:

Theorem 2.2.10. *$\pi_{27}(DT(\mathcal{GS}))$ is a simplicial complex.*

Proof. We show that there are no cycles of length 2 in $\pi_{27}(DT(\mathcal{GS}))$. Let $\mathcal{D}_{\boldsymbol{c}}(i,j,k)$ denote the translation of $\mathcal{D}_{\boldsymbol{c}}$ by $(i \cdot c_x, j \cdot c_y, k \cdot c_z)$, i.e.

$$\mathcal{D}_{\boldsymbol{c}}(i,j,k) := [i \cdot c_x, (i+1) \cdot c_x) \times [j \cdot c_y, (j+1) \cdot c_y) \times [k \cdot c_z, (k+1) \cdot c_z).$$

Assume that there is a cycle of length two in $\pi_{27}(DT(\mathcal{GS}))$. Then there are vertices $v, v', v'' \in DT(\mathcal{GS})$ such that the edges (v, v') and (v', v''), $(v, v') \neq (v', v'')$, are contained in $DT(\mathcal{GS})$ and that $\pi_{27}(v) = \pi_{27}(v'')$. Let $\mathcal{D}_{\boldsymbol{c}}(i, j, k), \mathcal{D}_{\boldsymbol{c}}(i', j', k')$, and $\mathcal{D}_{\boldsymbol{c}}(i'', j'', k'')$ denote the translations of $\mathcal{D}_{\boldsymbol{c}}$ that contain v, v', and v'', respectively. According to Lemma 2.2.9, if a simplex intersects both $\mathcal{D}_{\boldsymbol{c}}(i, j, k)$ and $\mathcal{D}_{\boldsymbol{c}}(i', j', k')$, then $|i - i'| \leq 1$, $|j - j'| \leq 1$, and $|k - k'| \leq 1$. Without loss of generality, we can choose v and v' such that $(i, j, k), (i', j', k') \in \{0, 1\}^3$. Note that $\pi_{27}(\mathcal{D}_{\boldsymbol{c}}(i, j, k)) = \pi_{27}(\mathcal{D}_{\boldsymbol{c}}(i \mod 3, j \mod 3, k \mod 3))$, so at least one of the i'', j'', and k'' must be in $\{0 - 3, 1 - 3, 0 + 3, 1 + 3\}$ for $\pi_{27}(v) = \pi_{27}(v'')$ to hold. This is not possible according to Lemma 2.2.9. □

With the same argumentation as in the proof of Lemma 2.2.7, $|\pi_{27}(DT(\mathcal{GS}))|$ is homeomorphic to the 27-sheeted covering space $\mathbb{T}^3_{\boldsymbol{h}*\boldsymbol{c}}$ with $\boldsymbol{h} = (3, 3, 3)$, which is homeomorphic to $\mathbb{T}^3_{\boldsymbol{c}}$.

We prefer to use the framework of covering spaces, rather than just talk about copies of the points as in [DH97b] because it avoids creating artificial boundaries in the data structure; the adjacency relations are computed for all simplices.

As will be seen in the next section, the incremental algorithm that we present requires a slightly stronger result than Theorem 2.2.10.

2.3 Algorithm

As mentioned in the introduction, there is a strong motivation for reusing the standard incremental algorithm [Bow81] to compute a periodic Delaunay triangulation. Let us give a rough sketch of the algorithm that we propose, before we present it in more detail later.

- We start computing in a finitely-sheeted covering space $\mathbb{T}^3_{\boldsymbol{h}*\boldsymbol{c}}$ of $\mathbb{T}^3_{\boldsymbol{c}}$, with $\boldsymbol{h}$ chosen such that $\pi_{\boldsymbol{h}}(DT(\mathcal{GS}))$ is guaranteed to be a triangulation.
  Theorem 2.2.10 shows that such a covering space always exists: $\boldsymbol{h} = (3, 3, 3)$ is one possible choice. In fact, the algorithm requires a slightly stronger result than Theorem 2.2.10 and thus needs to use other covering spaces if $\mathcal{D}_{\boldsymbol{c}}$ is not a cube.

- In practice, if the point set is large and reasonably well distributed, it is likely that after having inserted all the points of a subset $\mathcal{S}' \subset \mathcal{S}$, all the subsequent $\pi(DT(\mathcal{GS}''))$ for $\mathcal{S}' \subset \mathcal{S}'' \subseteq \mathcal{S}$ are simplicial complexes in $\mathbb{T}^3_{\boldsymbol{c}}$.
  In this case, we discard all periodic copies of simplices of $\pi_{\boldsymbol{h}}(DT(\mathcal{GS}'))$ and switch to computing $\pi(DT(\mathcal{GS}))$ in $\mathbb{T}^3_{\boldsymbol{c}}$ by adding all the points left in $\mathcal{S} \setminus \mathcal{S}'$.

In this way, unlike [DH97b], we avoid duplicating points as soon as this is possible. In cases when $\mathcal{S}$ is a small or badly distributed point set, the algorithm never enters the second phase and returns $\pi_{\boldsymbol{h}}(DT(\mathcal{GS}))$, which is a triangulation of $\mathbb{T}^3_{\boldsymbol{h}*\boldsymbol{c}}$, still homeomorphic to $\mathbb{T}^3_{\boldsymbol{c}}$.

Section 2.3 is organized as follows: We describe the algorithm for the restricted case when $\mathcal{D}_{\boldsymbol{c}}$ is a cube and we prove its correctness. This corresponds to what the current CGAL implementation provides. In the second part we relax the condition on $\mathcal{D}_{\boldsymbol{c}}$, allowing it to be a cuboid and we describe how the algorithm can be adapted to this case. Finally, in Section 2.3.3 we show how to extend the algorithm to compute weighted Delaunay triangulations as well.

Note that, before switching to computing in $\mathbb{T}^3_{\boldsymbol{c}}$, it is not sufficient to test whether $\pi(DT(\mathcal{S}'^{\boldsymbol{c}}))$ is a simplicial complex. Indeed, adding a point could create a cycle of length two (see Figure 2.5). So, a stronger condition is needed before the switch.

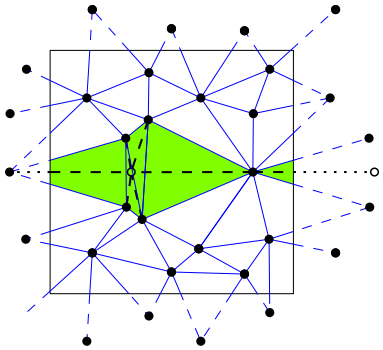


Figure 2.5: (2D illustration) Adding a point in a simplicial complex can create a cycle of length two.

The following observation will be useful in the subsequent proofs:

Observation 2.3.1. *Let Δ denote a tetrahedron in $DT(\mathcal{GS})$ and $\mathcal{B}_\Delta$ its circumscribing ball. If the diameter of $\mathcal{B}_\Delta$ is smaller than $c_{\min} := \min\{c_x, c_y, c_z\}$, then $\pi|_{\mathcal{B}_\Delta}$ is injective, and $\pi|_\Delta$ is injective as well since $\Delta \subset \mathcal{B}_\Delta$. Thus Δ is a simplex in $\mathbb{T}^3_{\boldsymbol{c}}$.*

2.3.1 Cubic domain

If the original domain $\mathcal{D}_{\boldsymbol{c}}$ is a cube with edge length c, the incremental algorithm uses the 27-sheeted covering space $\mathbb{T}^3_{3\boldsymbol{c}}$, where $3\boldsymbol{c} = (3c, 3c, 3c)$. The original domain for this covering space is $\mathcal{D}_{3\boldsymbol{c}}$.

We prove below that once all edges in the triangulation of $\mathbb{T}^3_{3\boldsymbol{c}}$ are shorter than $\frac{1}{\sqrt{6}}\,c$, computing a simplicial complex in $\mathbb{T}^3_{\boldsymbol{c}}$ is possible.

See Algorithm 2.3.1 for a pseudo-code listing of the algorithm. It computes either a Delaunay triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$, as defined in Definition 2.2.2, or a Delaunay triangulation of $\mathbb{T}^3_{3\boldsymbol{c}}$, which is still homeomorphic to $\mathbb{T}^3_{\boldsymbol{c}}$.

To show the correctness of the algorithm, it remains to establish the following two properties:

1. After each insertion, TR_{27} is a Delaunay triangulation of $\mathbb{T}^3_{3\boldsymbol{c}}$. Let us emphasize on the fact that Theorem 2.2.10 cannot be used here because in the inner loop (step 8), the set of points present in TR_{27} does not contain all the periodic copies of p.
   Let p be a point in $\mathcal{D}_{\boldsymbol{c}}$ and $\mathcal{T}_p \subseteq \varphi_{\boldsymbol{c}}(\{p\} \times \mathbb{Z}^3) \cap \mathcal{D}_{3\boldsymbol{c}}$, i.e., $\mathcal{T}_p$ is a subset of the grid of 27 copies of p that lie within $\mathcal{D}_{3\boldsymbol{c}}$. Then TR_{27} is always of the form $\pi_{27}(DT(\mathcal{GS} \cup \mathcal{T}_p^{3\boldsymbol{c}}))$ with $\mathcal{T}_p^{3\boldsymbol{c}} = \varphi_{3\boldsymbol{c}}(\mathcal{T}_p \times \mathbb{Z}^3)$. Lemma 2.3.2 shows that this is a triangulation.

2. If all edges in $\pi_{27}(DT(\mathcal{GS}))$ are shorter than $\frac{1}{\sqrt{6}}\,c$, then we can switch to computing in $\mathbb{T}^3_{\boldsymbol{c}}$. See Criterion 2.3.4 for a proof.

Algorithm 2.3.1 Compute Delaunay triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$ from a point set.

Input: Set $\mathcal{S}$ of points in a cube $\mathcal{D}_{\boldsymbol{c}}$ of edge length $c \in \mathbb{E}^3 \setminus \{0\}$.
Output: $DT_{\mathbb{T}}(\mathcal{S})$ if possible, otherwise $\pi_{27}(DT(\mathcal{GS}))$

1: $\mathcal{S}' \Leftarrow \mathcal{S}$
2: Pop p from $\mathcal{S}'$
3: $\mathcal{S} \Leftarrow \{p\}$
4: $\mathrm{TR}_{27} \Leftarrow \pi_{27}(DT(\varphi_{\boldsymbol{c}}(\{p\} \times \mathbb{Z}^3)))$ `// can be precomputed`
5: **while** the longest edge in TR_{27} is longer than $\frac{1}{\sqrt{6}}c$ **do**
6: Pop p from $\mathcal{S}'$; $\mathcal{S} \Leftarrow \mathcal{S} \cup \{p\}$
7: **for all** $p' \in \{p + \boldsymbol{c} * \zeta \mid \zeta \in \{0,1,2\}^3\}$ **do**
8: Insert p' into TR_{27}
9: **end for** // $\mathrm{TR}_{27} = \pi_{27}(DT(\mathcal{GS}))$
10: **if** $\mathcal{S}' = \emptyset$ **then return** $\mathrm{TR}_{27} = \pi_{27}(DT(\mathcal{GS}))$ `// non-triangulable point set`
11: **end while**
12: Compute $DT_{\mathbb{T}}(\mathcal{S})$ from TR_{27} `// switch to` $\mathbb{T}^3_{\boldsymbol{c}}$
13: Insert all points remaining in $\mathcal{S}'$ into $DT_{\mathbb{T}}(\mathcal{S})$ one by one
14: **return** $DT_{\mathbb{T}}(\mathcal{S})$

Lemma 2.3.2. *Let $\mathcal{S} \subset \mathcal{D}_{\boldsymbol{c}}$ be a finite point set, $p \in \mathcal{D}_{\boldsymbol{c}}$ be a point and $\mathcal{T}_p$ a subset of $\varphi_{\boldsymbol{c}}(\{p\} \times \mathbb{Z}^3) \cap \mathcal{D}_{3\boldsymbol{c}}$. If $\mathcal{D}_{\boldsymbol{c}}$ is a cube, then $\pi_{27}(DT(\mathcal{GS} \cup \mathcal{T}_p^{3\boldsymbol{c}}))$ is a triangulation of $\mathbb{T}^3_{3\boldsymbol{c}}$ with $\mathcal{T}_p^{3\boldsymbol{c}} := \varphi_{3\boldsymbol{c}}(\mathcal{T}_p \times \mathbb{Z}^3)$.*

Proof. We first consider a triangulation defined by only one point, i.e. $\mathcal{S} = \{q\}$ for some $q \in \mathcal{D}_{\boldsymbol{c}}$. Then $\pi_{27}(\mathcal{GS})$ consists of 27 points arranged as a regular grid. Without loss of generality, we can assume $q = (0,0,0)$ because triangulations of point sets are invariant with respect to translations. The point set $\pi_{27}(\mathcal{GS})$ is highly degenerate: There are 27 different empty balls that all have eight points on their boundary. They are centered at $\left(\left(\frac{1}{2},\frac{1}{2},\frac{1}{2}\right) + \zeta\right) * \boldsymbol{c}$ with $\zeta \in \{0,1,2\}^3$ and have radius $r = \frac{\sqrt{3}}{2}\,c \approx 0.866\,c$. Without loss of generality, we now consider the ball $\mathcal{B}$ centered at $\left(\frac{1}{2},\frac{1}{2},\frac{1}{2}\right)$. It is easy to check that the intersection of the image of any pair of these balls under π_{27} consists of one connected set, i.e., that π_{27} restricted to the union of any pair of these balls is injective.

As the length of an edge of the cube $\mathcal{D}_{3\boldsymbol{c}}$ is $3\boldsymbol{c}$ and the diameter of $\mathcal{B}$ is $\sqrt{3}\,c$, an edge of length $(3-\sqrt{3})\,c$ would be enough to introduce a cycle of length two. Let $\mathcal{B}'$ be a largest empty ball given by the points $(2,0,0)$, $(2,0,1)$, $(2,1,0)$, $(2,1,1)$ and $\left(\frac{1+\sqrt{3}}{2},\frac{1}{2},\frac{1}{2}\right)$. It turns out that the radius of $\mathcal{B}'$ is smaller than $0.712\,c$. There exist other such balls along the y- and z-axis but since $\mathcal{D}_{\boldsymbol{c}}$ is a cube, their radii are the same. The width of the overlap between $\mathcal{B}$ and $\mathcal{B}'$ is less than $(2 \cdot 0.712 + \sqrt{3} - 3)c < 0.155\,c$. Then $\pi_{27}|_{\mathcal{B}\cup\mathcal{B}'}$ is not injective and $\pi_{27}(\mathcal{B} \cap \mathcal{B}')$ consists of two disjoint connected components.

Now we have to exploit a special property of Algorithm 2.3.1: $\pi_{27}(\mathcal{GS})$ consists of a regular orthogonal grid of 27 points and $\pi_{27}(\mathcal{T}_p^{3\boldsymbol{c}})$ is a subset of the grid of 27 copies of p. From this we know that after adding a point $p_1 \in \mathcal{T}_p^{3\boldsymbol{c}}$, the next point will differ by some vector in $\boldsymbol{c} * \mathbb{Z}^3$. However, to be able to form a cycle of length two crossing the balls $\mathcal{B}$ and $\mathcal{B}'$ its length along one axis would have to be between $1.266c$ and $1.424c$. This is not possible because $[1.266, 1.424] \cap \mathbb{Z} = \emptyset$, see Figure 2.6. Note that if we move p_1 inside $\mathcal{B}$, then the radius of $\mathcal{B}'$ becomes smaller and thus the overlap range is only a subset of $[1.266, 1.424]$.

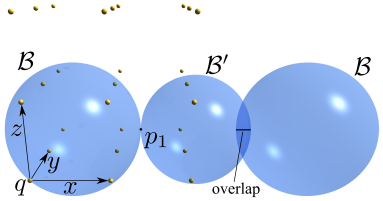


Figure 2.6: No periodic copy of p_1 can lie inside $\mathcal{B} \cap \mathcal{B}'$.

There is no further possibility for a cycle of length two to occur because the shortest diagonals through $\mathcal{D}_{3\boldsymbol{c}}$ have length $\sqrt{18}\,c$ which is larger than $2\sqrt{3}c$.

Using Theorem 2.2.8, this proves that $\pi_{27}(DT(\mathcal{GS} \cup \mathcal{T}_p^{3\boldsymbol{c}}))$ is a triangulation when $\mathcal{S}$ consists of only one point. If $\mathcal{S}$ contains more points, then the empty balls can only be smaller, which even more avoids cycles of length two. □

Now we give a geometric criterion to decide whether $\pi(DT(\mathcal{GS}))$ is a simplicial complex and thus a triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$.

Criterion 2.3.3. *If the diameter of the circumscribing ball of any tetrahedron in $DT(\mathcal{GS})$ is smaller than $\frac{1}{2}\,c$, then $\pi(DT(\mathcal{GT}))$ is a simplicial complex for any finite $\mathcal{T} \subset \mathcal{D}_{\boldsymbol{c}}$ with $\mathcal{S} \subseteq \mathcal{T}$.*

Proof. The edges of a tetrahedron are completely contained in its circumscribing ball and are thus bounded by the ball's diameter. If the diameter of any circumscribing ball is smaller than $\frac{1}{2}\,c$, then all edges in the triangulation are shorter than $\frac{1}{2}\,c$. In order to create a cycle of length two, the sum of the lengths of the two edges needs to be at least c, which is not possible if both edges are shorter than $\frac{1}{2}\,c$. From Observation 2.3.1 and Theorem 2.2.8, it follows that $\pi(DT(\mathcal{GS}))$ is a simplicial complex. If we add more points, the diameter of the largest empty ball cannot become larger. The claim follows. □

We now prove the geometric criterion that is used in practice.

Criterion 2.3.4. *If the 1-skeleton of $DT(\mathcal{GS})$ contains only edges shorter than $\frac{1}{\sqrt{6}}\,c$, where c is the edge length of $\mathcal{D}_{\boldsymbol{c}}$, then $\pi(DT(\mathcal{T}^{\boldsymbol{c}}))$ is a simplicial complex for any finite $\mathcal{T} \subset \mathcal{D}_{\boldsymbol{c}}$ with $\mathcal{S} \subseteq \mathcal{T}$, so it is a triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$.*

Proof. Assume that there is a ball $\mathcal{B}$ of diameter d that does not contain any point of $\mathcal{GS}$ in its interior. Consider the tetrahedron Δ in $DT(\mathcal{GS})$ that contains the center of $\mathcal{B}$. The length of the largest edge of Δ is bounded from below by the edge length of the regular tetrahedron with circumscribing ball $\mathcal{B}$, which is $\frac{2d}{\sqrt{6}}$. So if all edges in $DT(\mathcal{GS})$ are shorter than $\frac{1}{\sqrt{6}}\,c$, then the diameter of any empty ball is smaller than $\frac{1}{2}\,c$. The claim follows from Criterion 2.3.3 and Lemma 2.2.7. □

Note that Criteria 2.3.3 and 2.3.4 are only sufficient, where Criterion 2.3.3 is weaker than Criterion 2.3.4: There are point sets with maximum empty ball diameter shorter than $\frac{1}{2}\,c$ but edges longer than $\frac{1}{\sqrt{6}}\,c$. A more detailed discussion on the two criteria can be found in Section 3.6.7.

Lemma 2.3.2 and Criterion 2.3.4 prove:

Theorem 2.3.5. *Algorithm 2.3.1 is correct, i.e., it always computes a Delaunay triangulation homeomorphic to* $\mathbb{T}^3_{\boldsymbol{c}}$.

A weaker version of Lemma 2.3.2, which will be used in the next section, follows immediately from Criterion 2.3.3:

Corollary 2.3.6. *Let* $\mathcal{S} \subset \mathcal{D}_{\boldsymbol{c}}$, $\mathcal{S} \neq \emptyset$ *and* $\mathcal{T} \subset \mathcal{D}_{4\boldsymbol{c}}$ *be finite point sets. If* $\mathcal{D}_{\boldsymbol{c}}$ *is a cube, then* $\pi_{64}(DT(\mathcal{GS} \cup \mathcal{T}^{4\boldsymbol{c}}))$ *is a triangulation of* $\mathbb{T}^3_{4\boldsymbol{c}}$.

Proof. The largest empty ball in $\mathcal{GS}$ has diameter $\sqrt{3}\,c$. The domain $\mathcal{D}_{4\boldsymbol{c}}$ is a cube of edge length $4c$, which is more than twice $\sqrt{3}\,c$. Thus Criterion 2.3.3 applies. □

Note that unlike Lemma 2.3.2, Corollary 2.3.6 does not require $\mathcal{T}$ to be a subset of a point grid $\varphi_{\boldsymbol{c}}(\{p\} \times \mathbb{Z}^3)$ for some point $p \in \mathcal{D}_{\boldsymbol{c}}$. Even though its result is weaker in that it yields a larger number of periodic copies to consider, it is much easier to prove and thus it generalizes easily to other settings as will be seen below. Before, we describe an approach that avoids computing in $\mathbb{T}^3_{3\boldsymbol{c}}$.

Dummy points In order to avoid computing in $\mathbb{T}^3_{3\boldsymbol{c}}$ at the beginning, the algorithm starts with an initial triangulation of a dummy point set $\mathcal{P}$, chosen such that any superset of $\mathcal{P}$ has a triangulation in $\mathbb{T}^3_{\boldsymbol{c}}$. A possible dummy point set of 36 points is defined as follows:

$$\begin{aligned}\mathcal{P} \;:=\; & \left\{c \cdot (x,y,z) \mid x,y \in \left\{0, \tfrac{1}{3}, \tfrac{2}{3}\right\}, z \in \left\{0, \tfrac{1}{2}\right\}\right\} \\ & \cup \left\{c \cdot \left(\tfrac{1}{6}+x, \tfrac{1}{6}+y, \tfrac{1}{4}+z\right) \mid x,y \in \left\{0, \tfrac{1}{3}, \tfrac{2}{3}\right\}, z \in \left\{0, \tfrac{1}{2}\right\}\right\}\end{aligned}$$

See Figure 2.7 for an illustration of the periodic Delaunay triangulation of the dummy point set.

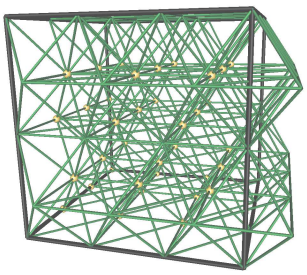


Figure 2.7: The triangulation of the dummy point set.

The diameter of the largest empty ball in the set $\mathcal{P}$ is smaller than $\frac{1}{2}c$ as required by Criterion 2.3.3. The point set $\mathcal{P}$ is easy to describe and sufficiently small for practical use. It would be interesting to investigate further to find the smallest possible point set. In two dimensions it is known that the smallest triangulation of a torus has 7 vertices [Möb86]. For the three-dimensional torus a triangulation of 15 vertices is known [KL84], but it is not known whether this is the smallest possible number of vertices. For both cases the smallest point sets defining a Delaunay triangulation with the diameter of the largest circumscribing ball bounded by $\frac{1}{2}c$ are unknown.

Once all the input points in $\mathcal{S}$ are inserted into the initial triangulation defined by $\mathcal{P}$, the points of $\mathcal{P}$ are removed from the triangulation. By construction, the point set $\mathcal{S} \cup \mathcal{P}$ defines a Delaunay triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$. If the input point set without $\mathcal{P}$ does not define a triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$ anymore, then the triangulation will be converted to $\mathbb{T}^3_{3\boldsymbol{c}}$ during the removal of one of the dummy points.

The only overhead of this approach is the removal of the points in $\mathcal{P}$. There is additionally the potential conversion to $\mathbb{T}^3_{3\boldsymbol{c}}$ but without using the dummy point set the whole triangulation would be computed in $\mathbb{T}^3_{3\boldsymbol{c}}$ in this case, which is likely to be even more expensive. If the input point set is large enough, then the overhead of removing 36 points is negligible. As it completely avoids computing in $\mathbb{T}^3_{3\boldsymbol{c}}$, it is faster than the standard approach.

2.3.2 Non-cubic domain

The above discussion still remains valid if the original domain $\mathcal{D}_{\boldsymbol{c}}$ is a general cuboid, i.e. $\boldsymbol{c} = (c_x, c_y, c_z)$. Only the constants, i.e., the number of sheets of the covering space to start with and the edge length threshold need to be adapted.

Let $\mathbb{T}_{\boldsymbol{h}*\boldsymbol{c}}$ with $\boldsymbol{h} = (h_x, h_y, h_z)$ be a covering space of $\mathbb{T}^3_{\boldsymbol{c}}$ with projection map $\pi_{\boldsymbol{h}*\boldsymbol{c}}$ and original domain $\mathcal{D}_{\boldsymbol{h}*\boldsymbol{c}}$.

We first give a criterion to decide when to switch back to $\mathbb{T}^3_{\boldsymbol{c}}$ in Algorithm 2.3.1 in the case of a non-cubic domain.

Criterion 2.3.7. *If the diameter of the circumscribing ball of any tetrahedron in $DT(\mathcal{GS})$ is smaller than $\frac{1}{2}\,c_{\min}$, where $c_{\min} = \min\{c_x, c_y, c_z\}$, then $\pi(DT(\mathcal{GT}))$ is a simplicial complex for any finite $\mathcal{T} \subset \mathcal{D}_{\boldsymbol{c}}$ with $\mathcal{S} \subseteq \mathcal{T}$.*

Proof. The proof of this criterion is essentially the same as for Criterion 2.3.3: If the diameter of any circumscribing ball is smaller than $\frac{1}{2}\,c_{\min}$, then all edges in $DT(\mathcal{GS})$ are shorter than $\frac{1}{2}\,c_{\min}$. The shortest possible non-trivial cycle in $\mathbb{T}^3_{\boldsymbol{c}}$ has length $c_{\min}$, thus two edges the 1-skeleton of $DT(\mathcal{GS})$ cannot form such a cycle. The claim follows from Theorem 2.2.8. □

Using Criterion 2.3.7, we can give the covering space required in Algorithm 2.3.1.

Lemma 2.3.8. *Let $\mathcal{S} \subset \mathcal{D}_{\boldsymbol{c}}$, $\mathcal{S} \neq \emptyset$ and $\mathcal{T} \subset \mathcal{D}_{\boldsymbol{h}*\boldsymbol{c}}$ be finite point sets. Let $h_i := \left\lceil 2\frac{\|\boldsymbol{c}\|}{c_i} \right\rceil$ for $i = x, y, z$. Then $\pi_{\boldsymbol{h}*\boldsymbol{c}}(DT(\mathcal{GS} \cup \mathcal{T}_p^{\boldsymbol{h}*\boldsymbol{c}}))$ with $\mathcal{T}^{\boldsymbol{h}*\boldsymbol{c}} := \varphi_{\boldsymbol{h}*\boldsymbol{c}}(\mathcal{T} \times \mathbb{Z}^3)$ is a triangulation of $\mathbb{T}^3_{\boldsymbol{h}*\boldsymbol{c}}$.*

Proof. The largest empty ball in $\mathcal{GS}$ has diameter smaller than $\|\boldsymbol{c}\|$. The domain $\mathcal{D}_{\boldsymbol{h}*\boldsymbol{c}}$ is a cuboid of edge lengths $h_i c_i$ for $i = x, y, z$. The condition $h_i = \left\lceil 2\frac{\|\boldsymbol{c}\|}{c_i} \right\rceil$ implies that $h_i c_i \geq 2\|\boldsymbol{c}\|$. Applying Criterion 2.3.7 to the flat torus with original domain $\mathcal{D}_{\boldsymbol{h}*\boldsymbol{c}}$ proves the claim. □

2.3.3 Weighted Delaunay triangulation

The weighted Delaunay triangulation, or regular triangulation, generalizes the Delaunay triangulation when the sites are spheres, also called *weighted points*. We gave a short introduction on weighted Delaunay triangulations in Section 1.2.2.

Like the Delaunay triangulation, the weighted Delaunay triangulation can be defined uniquely, even in degenerate cases [DT06a]. Here, we show that the whole discussion in Section 2.2 works in the same way for weighted Delaunay triangulations. We also prove a geometric criterion using the edge length similar to Criterion 2.3.4, which allows us to generalize Algorithm 2.3.1 to compute a periodic weighted Delaunay triangulation as well.

Let $\mathcal{S}$ be a set of weighted points in $\mathbb{E}^3$, i.e. pairs $(p, w_p) \in \mathbb{E}^3 \times \mathbb{R}$. Let $\mathcal{D}_{\boldsymbol{c}}$ be a cube of edge length c and $\mathcal{S}$ a set of weighted points in $\mathcal{D}_{\boldsymbol{c}}$. Let W and w respectively denote the largest and smallest weight in $\mathcal{S}$. Let $\mathrm{WDT}(\mathcal{GS})$ denote the weighted Delaunay triangulation of $\mathbb{E}^3$ defined by $\mathcal{GS}$.

Criterion 2.3.9. *If the longest edge of* $\mathrm{WDT}(\mathcal{GS})$ *is shorter than* $\sqrt{\frac{1}{6}c^2 - \frac{8}{3}(W-w)}$, *then* $\pi(\mathrm{WDT}(\mathcal{T}^{\boldsymbol{c}}))$ *is a triangulation of* $\mathbb{T}^3_{\boldsymbol{c}}$ *for any finite* $\mathcal{T} \subset \mathcal{D}_{\boldsymbol{c}}$ *with* $\mathcal{S} \subseteq \mathcal{T}$.

Note that if $W = w$, this bound reduces to $\frac{1}{\sqrt{6}}\,c$, which is the edge length threshold for Delaunay triangulations. The proof of this result uses the following auxiliary lemma:

Lemma 2.3.10. *Consider* $\mathrm{WDT}(\mathcal{GS})$. *If the orthogonal weighted point of a tetrahedron has weight* w_o, *then the length of the longest edge of the tetrahedron is not larger than* $2\sqrt{w_o + W - w}$.

Proof. Let (o, w_o) be an orthogonal weighted point and $(p, w_p), (q, w_q)$ be two of the weighted points orthogonal to (o, w_o). Then $\mathrm{dist}(o, p) = \sqrt{w_p + w_o}$. The edge length $\mathrm{dist}(p, q)$ attains its maximum if the segment $[p, q]$ contains o. Then $\mathrm{dist}(p, q) = \sqrt{w_o + w_p} + \sqrt{w_o + w_q}$. This expression is maximized if the weight is maximized. We know that changing the weights of all points in the triangulation by the same amount does not change the triangulation. So we can assume $W - w$ to be the maximum possible weight. The claim follows. See Figure 2.8 for an illustration in 2D. □

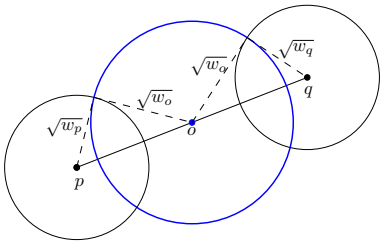


Figure 2.8: (2D illustration) The longest possible edge of a tetrahedron with orthogonal sphere of radius $\sqrt{w_o}$.

Proof of Criterion 2.3.9. Let λ be the length of the longest edge in $\mathrm{WDT}(\mathcal{GS})$. Then the circumradius of any tetrahedron in $\mathrm{WDT}(\mathcal{GS})$ cannot be larger than $\sqrt{\frac{3}{8}}\lambda$. From Lemma 2.3.10 follows that the longest possible edge of a tetrahedron is bounded by $\lambda_{max} := 2\sqrt{\left(\sqrt{\frac{3}{8}}\lambda\right)^2 + W - w}$. If we now choose λ to be smaller than $\sqrt{\frac{1}{6}c^2 - \frac{8}{3}(W-w)}$, then λ_{max} is smaller than $\frac{1}{2}\,c$. Thus no cycle of length two can occur and $\pi(\mathrm{WDT}(\mathcal{GS}))$ is a triangulation according to Theorem 2.2.8 together with the equivalent of Observation 2.3.1

for weighted Delaunay triangulations. Adding further points cannot increase the size of the largest circumscribing ball and so $\pi(\mathrm{WDT}(\mathcal{T}^{\boldsymbol{c}}))$ is a triangulation as well. □

Corollary 2.3.11. *Let η be given as $\eta := \lceil 8{\cdot}(3+\frac{4}{c}(W-w))^{3/2}\rceil$ and $h := \eta^3$. Let $\mathcal{T}_p$ be a subset of the h copies of p in $\mathcal{D}_{\eta\boldsymbol{c}}$ and $T_p^{\eta\boldsymbol{c}} := \varphi_{\eta\boldsymbol{c}}(\mathcal{T}_p \times \mathbb{Z}^3)$. Then $\pi_h(\mathrm{WDT}(\mathcal{GS} \cup \mathcal{T}_p^{\eta\boldsymbol{c}}))$ is a triangulation of $\mathcal{T}^3_{\eta\boldsymbol{c}}$.*

Proof. If $\mathcal{S}$ consists of only one point, then the weighted Delaunay triangulation defined by $\mathcal{GS}$ actually is a Delaunay triangulation because all periodic copies of the point have the same weight. So the largest orthogonal sphere coincides with the largest circumsphere and has radius $\frac{\sqrt{3}}{2}\,c$. With the same argument as in the proof of Criterion 2.3.9 the diameter of the largest orthogonal sphere after adding any other point is bounded by $2{\cdot}\sqrt{(\frac{\sqrt{3}}{2}\,c)^2 + W - w}$. To make sure that this expression is smaller than $2{\cdot}\eta{\cdot}c$ we choose $\eta = \lceil 8{\cdot}(3+\frac{4}{c}(W-w))^{3/2}\rceil$. □

2.4 Analysis

2.4.1 Complexity analysis

In this Section we show that using the Delaunay hierarchy, the randomized worst-case complexity of Algorithm 2.3.1 is asymptotically equal to the one of the Algorithm for computing the Delaunay triangulation of $\mathbb{E}^3$.

Let us first discuss the following two points before we consider the Delaunay hierarchy in more detail: (1) How to test for the length of the longest edge and (2) how to convert from the triangulation of $\mathbb{T}^3_{\boldsymbol{h}*\boldsymbol{c}}$ to the triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$.

(1) We maintain an unsorted data structure $\mathcal{E}$ that references all edges that are longer than the threshold $\frac{1}{\sqrt{6}}\,c_{\min}$. As soon as $\mathcal{E}$ is empty, we know that the longest edge is smaller than the threshold. The total number of edges that are inserted to and removed from $\mathcal{E}$ is at most proportional to the total number of simplices that are created and destroyed during the algorithm. We can have direct access from the simplices to their edges in $\mathcal{E}$. Hence, the maintenance of $\mathcal{E}$ does not change the algorithm complexity.

(2) To convert the triangulation of $\mathbb{T}^3_{\boldsymbol{h}*\boldsymbol{c}}$ to $DT_{\mathbb{T}}(\mathcal{S})$ when we convert to $\mathbb{T}^3_{\boldsymbol{c}}$, we need to iterate over all cells and all vertices to delete all periodic copies, keeping only one; furthermore, we need to update the incidence relations of those tetrahedra whose neighbors have been deleted. This is linear in the size of the triangulation and thus dominated by the main loop.

The Delaunay Hierarchy The overall algorithm is incremental and can be combined with the Delaunay hierarchy [Dev02], see also Section 3.3.9. In a nutshell, the structure is designed for efficient computation of a Delaunay triangulation of $\mathbb{E}^d$; it is built incrementally and has several levels: the intermediate levels store the Delaunay triangulations of an increasing sequence of subsets of the set of input points, while the last level stores the complete triangulation. There are pointers between some vertices in different levels corresponding to the same input point. The structure allows for fast point location in the complete triangulation. We refer the reader to the original paper for a more complete description of this data structure.

This structure can be adapted to our algorithm. Let $\mathbb{T}^3_{\boldsymbol{h*c}}$ be the covering space chosen as presented at the beginning of Section 2.3. Each new level of the hierarchy stores a triangulation of $\mathbb{T}^3_{\boldsymbol{h*c}}$ when it is created, and this triangulation is converted when possible to a triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$. Note that, if a given level l stores a triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$, then the next level $l+1$ is also in $\mathbb{T}^3_{\boldsymbol{c}}$, since it contains more points and thus also stores a triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$ by Criterion 2.3.4. However, some level l can store a triangulation of $\mathbb{T}^3_{\boldsymbol{h*c}}$ while the next level $l+1$ is converted into $\mathbb{T}^3_{\boldsymbol{c}}$. In this case, for all the vertices corresponding to periodic copies of a given input point in that level l, their pointer to the level $l+1$ just all lead to the same vertex corresponding to this non-duplicated point in $\mathbb{T}^3_{\boldsymbol{c}}$.

The randomized analysis of [Dev02] assumes the insertion of points of $\mathcal{S}$ to be performed in a random order. The changes to this analysis, when computing in $\mathbb{T}^3_{\boldsymbol{h*c}}$, are minor: The points are inserted in sets of constant size (the number of periodic copies), and these sets are inserted in random order. The result remains the same.

Theorem 2.4.1. *Algorithm 2.3.1 has optimal randomized worst-case time and space complexity $O(n^2)$.*

Proof. Let $\mathcal{S}$ be the input point set. The vertex set of the Delaunay triangulation in level i is denoted by $\mathcal{S}_i$ and the levels of the hierarchy are numbered from bottom to top, that is $\mathcal{S}_0 = \mathcal{S}$. Let $1/\alpha$ denote the probability that a point is in $\mathcal{S}_{i+1}$ given that it is in $\mathcal{S}_i$. In Algorithm 2.3.1, the points of $\mathcal{S}$ can be inserted in random order but when computing in $\mathbb{T}^3_{3\boldsymbol{c}}$, 27 copies of each point are inserted consecutively. Let $\mathcal{S}'_i$ be the set containing 27 copies of each point of $\mathcal{S}_i$.

The randomized worst-case analysis in [Dev02] shows that the expected cost of the walk in level i is linear in the vertex degree and α, i.e. in $O(\alpha n)$, if the input points are inserted in a random order. The proof is based on the fact that the number of points in $\mathcal{S}_i$ that are closer to a query point q than to any other point in $\mathcal{S}_{i+1}$ is in $O(\alpha)$. This property extends to the case of computing in 27-sheeted covering space. The vertex set of the Delaunay triangulation in level i is $\mathcal{S}'_i$. As for each point of $\mathcal{S}_i$ there are 27 copies in $\mathcal{S}'_i$, the above bound can be at most 27 times bigger, which is still in $O(\alpha)$. Thus the cost of the walk in level i is still in $O(\alpha n)$ and summing up over all levels yields an expected $O(n)$ complexity for one point insertion.

In the beginning of this section, we showed that the maintenance of the structure $\mathcal{E}$ only requires a total number of $O(n^2)$ edges to be inserted and removed. Furthermore, we showed that the conversion from $\mathbb{T}^3_{3\boldsymbol{c}}$ to $\mathbb{T}^3_{\boldsymbol{c}}$ is $O(n^2)$, too, and it is applied only once during the algorithm run.

Also the maximum size of the structure $\mathcal{E}$ is in $O(n^2)$. We insert at most $27n$ points into the Delaunay triangulation, so the asymptotic size of the hierarchy does not change with respect to [Dev02]. □

The n in the bound for the point insertion comes from the worst-case vertex degree in the triangulation. Thus for triangulations with maximum vertex degree smaller than $O(n)$, the results can still be improved. This complexity result also holds for the extensions to non-cubic domains (Section 2.3.2) and to weighted Delaunay triangulations (Section 2.3.3).

2.4.2 Number of sheets

In this section, we give an estimation of the number of points required to switch back to 1-sheeted covering space if the input point set is uniformly distributed.

We first define the notations we use in the following discussion: Let $\mathcal{E}$ be an event and X be a random variable for $\mathcal{E}$. We denote the probability of $\mathcal{E}$ by $p(\mathcal{E})$ and the expected value and the variance of X by $E[X]$ and $V[X]$, respectively. When there is no ambiguity for the random variable we also use $\mu = E[X]$ and $\sigma = \sqrt{V[X]}$ to denote the expected value and the standard deviation, respectively.

We want to predict the number of points required in order to fulfill Criteria 2.3.3 and 2.3.4. More precisely, given a set of points, uniformly distributed in the unit cube, what is the expected value for the number of points such that the largest empty ball's diameter is smaller than $\frac{1}{2}$?

This problem can be modeled as a Poisson process [Mil70, San76]: Consider an infinite random point set Π given by a unit-intensity Poisson process on $\mathbb{E}^3$ and the cube $C := [0, \sqrt[3]{n}]^3$.

The Poisson distribution has the following properties: Let A denote a subset of $\mathbb{E}^3$ of volume $\|A\|$. The probability that A contains exactly k points of Π is given by $p(|A \cap \Pi| = k) = e^{-\|A\|}\frac{\|A\|^k}{k!}$. Then the probability that A does not contain any point of Π is $p(|A \cap \Pi| = 0) = e^{-\|A\|}$.

The expected number of points in a volume of measure $\|A\|$ is then given as

$$E[|A \cap \Pi|] = \sum_{k=0}^{\infty} k \cdot p(|A \cap \Pi| = k) = \sum_{k=0}^{\infty} k \cdot e^{-\|A\|}\frac{\|A\|^k}{k!} = \|A\|$$

By the properties of the Poisson distribution and from the construction of C follows that expected number of points in C is n. Let Π_C denote the set of points of Π that lie inside C, i.e. $\Pi_C := \Pi \cap C$. We denote the number of points in Π_C by n_C.

Let $\mathcal{G}_C$ denote the group of three orthogonal axis-aligned translations with fundamental domain C. Then $DT(\mathcal{G}_C\Pi_C)$ is the infinite periodic Delaunay triangulation defined by Π_C and $DT(\mathcal{G}_C\Pi_C) \cap C$ the set of tetrahedra of $DT(\Pi)$ that are completely contained inside C together with their faces. In the following, we consider $DT(\mathcal{G}_C\Pi_C) \cap C$, this is called "minus sampling". Note that if we consider the point set $\mathcal{G}_C\Pi_C$ instead of Π, we do not change the set of points inside C. So the point set $\mathcal{G}_C\Pi_C \cap C$ still follows the Poisson distribution, a property that is heavily used in the subsequent discussion.

In a similar way as done by [BEY91], we can show the following theorem:

Theorem 2.4.2. *Let C and Π_C be defined as above. Let μ_O denote the expected value of the volume of the largest empty ball in the infinite periodic Delaunay triangulation $DT(\mathcal{G}_C\Pi_C)$. Then the following holds*

$$\frac{0.5 \ln n}{n} \leq \mu_O \leq \frac{5 \ln n + 2}{n}.$$

Proof. We denote by B_x a ball of volume x. A ball B_x is said to be empty if $B_x \cap \Pi = \emptyset$. The probability that $B_{5 \ln n}$ is empty is given by $p(B_{5 \ln n} \text{ empty}) = e^{-5 \ln n} = \frac{1}{n^5}$. Then $p(B_x \text{ empty}) \leq \frac{1}{n^5}$ for all $x \geq 5 \ln n$. We compute the probability of the existence of an empty circumscribing ball of volume at least $5 \ln n$ in $DT(\Pi) \cap C$. A circumscribing ball in $DT(\Pi)$ is defined by 4 points; thus there are $\binom{n_C}{4}$ possibilities to define a tetrahedron with points in $\Pi \cap C$ as vertices. We give an upper bound for the probability that $\binom{n_C}{4} \geq n^4$:

Let X be the random variable that describes the number of points from Π that lie in C. From the fact that X has a Poisson distribution, we know that $\mu = n$ and $\sigma = \sqrt{n}$. We now apply the Cantelli inequality[1] $P(X - \mu \geq k) \leq \frac{\sigma^2}{\sigma^2+k^2}$. Choosing $k = n$ this yields $P(X - n \geq n) \leq \frac{1}{n+1}$, which implies that the probability for $n_C \geq 2n$ is at most $\frac{1}{n+1}$. If $n_C = 2n$, then there are $\binom{2n}{4} < n^4$ possibilities to choose from. Let $\mathcal{B}_C$ denote the set of circumscribing balls defined by all possible 4-tuples of points in $\Pi \cap C$. The probability that one of the balls in $\mathcal{B}_C$ has volume larger than $5 \ln n$ and is empty given $n_C \leq 2n$ is then $p(\exists \text{ empty } B_{5 \ln n} \in \mathcal{B}_C \mid n_C \leq 2n) \leq \binom{2n}{4} \cdot \frac{1}{n^5} < \frac{1}{n+1}$.

So finally, we have the following probabilities:

$$p(\exists \text{ empty } B_{5 \ln n} \in \mathcal{B}_C \mid n_C \leq 2n) \leq \frac{1}{n} \qquad p(n_C \leq 2n) \leq 1$$
$$p(\exists \text{ empty } B_{5 \ln n} \in \mathcal{B}_C \mid n_C \geq 2n) \leq 1 \qquad p(n_C \geq 2n) \leq \tfrac{1}{n}$$

Summing up the conditional probabilities gives $p(\exists \text{ empty } B_{5 \ln n} \in \mathcal{B}_C) \leq \frac{2}{n}$.

Let Y denote the random variable that describes the diameter of the largest empty circumscribing ball in $\mathrm{DT}(\Pi) \cap C$. Then from the above discussion we have the probability $p(Y \geq 5 \ln n) \leq \frac{2}{n}$. The volume of the ball is bounded from above by the volume of C, which is n. Now we can give an upper bound for the expected value of Y. Let $f(x)$ denote the probability density function of the distribution of Y. Then

$$\begin{aligned} E[Y] = \int_0^\infty x f(x) dx &= \int_0^{5 \ln n} x f(x) dx + \int_{5 \ln n}^{n} x f(x) dx + \int_n^\infty x f(x) dx \\ &\leq 5 \ln n \cdot p(Y \leq 5 \ln n) + n \cdot p(Y \geq 5 \ln n) + 0 \leq 5 \ln n + 2 \end{aligned}$$

From [BEY91] we know that $E[Y] \geq 0.5 \ln n$.

The Poisson process we described so far was modeled such that the growth of the cube that contains the points of the triangulation gives the number of points in the cube. When considering periodic triangulations, we want to leave the size of the cube constant, which can be modeled by rescaling Π such that the cube C is rescaled to $[0,1]^3$. The whole discussion remains valid because the Delaunay triangulation is invariant to rescaling.

Let Π' denote the rescaled version of the point set Π, restricted to $[0,1]^3$ and copied periodically onto $\mathbb{E}^3$, i.e. $\Pi' := \{\frac{1}{\sqrt[3]{n}} \cdot p \mid p \in \Pi\}$. Let Y' denote the random variable that corresponds to Y but with respect to the rescaled cube. The rescaling divides the volume of the empty balls by n, so this proves the theorem for $\mu = E[Y']$. □

Let now Y'_d denote the random variable that describes the diameter of the largest empty ball in $DT(\Pi')$. We get the bounds for its expected value directly from the bounds on the empty ball volumes in Theorem 2.4.2:

$$\sqrt[3]{\frac{3}{\pi} \frac{\ln n}{n}} \leq E[Y'_d] \leq \sqrt[3]{\frac{30 \ln n + 12}{\pi \cdot n}} \tag{2.1}$$

Now we can estimate the number of uniformly distributed points required for the expected value of the largest ball diameter to be smaller than $\frac{1}{2}$. Rearranging the left part equation 2.1 yields $24 \leq n$, rearranging the right part gives $n \leq 507$. So for a uniformly distributed random point set the expected value of the size of the largest empty circumscribing ball is smaller than $\frac{1}{2}$ if the point set has more than 507 points.

[1] the one-sided variant of the well-known Chebyshev inequality

Note that the longest edge in a Delaunay triangulation is bounded by the diameter of the largest empty ball. Thus the same analysis works for estimating the number of points required such that the longest edge in the triangulation is shorter than $\frac{1}{\sqrt{6}}$ as required by Criterion 2.3.4. In this case we have to replace the constant of $\frac{1}{2}$ for the ball diameter by $\frac{1}{\sqrt{6}}$. From equation 2.1 then follows $56 \leq n$ and $n \leq 1030$.

In Section 3.6, we present experimental results on the number of points required by Criteria 2.3.3 and 2.3.4.

Chapter 3

Implementation

In this chapter we describe our implementation of 3D periodic triangulations in CGAL. The implementation is based on the 3D triangulations implementation that already existed. At first we give a brief introduction to CGAL. Then we have a more detailed look on the implementation of the 3D triangulations. Afterwards, we present the implementation of the 3D periodic triangulations. Then we give a more detailed analysis of the complexity of the main functions and we show the practical efficiency of our implementation in experiments on both generated and real-world data. Finally, we elaborate on extensions that we implemented additionally.

3.1 Introduction to CGAL

The Computational Geometry Algorithms Library CGAL is a collection of open source implementations of geometric data structures and algorithms [cga]. Its design follows the generic programming paradigm [Aus98, FT06]. This enables the user to easily plug different components together.

In generic programming a *concept* describes the types and operations that must be made available. A class is a *model* of a specific concept if it implements the required operations and types.

Implementations of geometric algorithms in CGAL strictly separate the combinatorial parts of the algorithms from the geometric computations. In this way different implementations of the geometric computations can be plugged into an algorithm implementation. Some basic geometric functionality is available by default in the geometry kernel that we describe below.

One major problem in implementing computational geometry algorithms is that computers do not dispose of a real RAM, i.e., the possibility to perform exact computations on real numbers. Roundoff errors in computations can lead to undefined behavior in the implementation of geometric algorithms. Kettner et al. [KMP+04] have shown that this is not a purely theoretical problem. Generally, there is a tradeoff between efficiency and exactness. The generic programming approach enables the user to choose a suitable number type and plug it into the geometry kernel,

Number types
In principle, the user can provide his own number type to perform the arithmetic operations

required by an algorithm. In CGAL there are several number types available:

- *Built-in floating-point number type (*`double`*)*: The C++ `double` number type is very fast but does not perform exact computations. It conforms to the IEEE standard 754 [Gol91, IEE85, IEE08].
- *Multi-precision floating-point number types*: They achieve a higher precision than `double` but they are slower and still not able to always provide exact computations.
- *Interval arithmetic*: This number type maintains two numbers that are generally of a fast and inexact number type as a lower and upper bound of the number they actually should represent.
- *Rational numbers*: This number type maintains two integer numbers, a numerator and a denominator. As long as the numbers to represent are rational, these number types are exact, at the expense of slower performance.
- *Algebraic numbers*: Such number types can represent all real roots of polynomials with integer coefficients. Unfortunately, computations on algebraic numbers are quite expensive.

Geometry kernels
In geometric algorithms there are two types of geometric computations: (1) predicate evaluations and (2) geometric constructions.

1. *Geometric predicates*: Given a set of geometric objects we want to know whether or not they have a certain property. The set of possible answers of a geometric predicate is small and discrete. Example: Given three points in $\mathbb{E}^2$, decide whether the third point lies to the left or to the right of the line defined by the first two points. There are three possible answers: left, right, exactly on the line.
2. *Geometric construction*: Given a set of geometric objects, compute a new geometric object out of them. Example: Compute the center of the circumscribing circle of three points in $\mathbb{E}^2$.

A CGAL geometry kernel must provide a collection of constant size geometric objects in $\mathbb{E}^2$ and $\mathbb{E}^3$, and geometric predicates and constructions of constant evaluation time. A non-exhaustive list consists of

- Geometric objects: `Point_2, Point_3, Segment_2, Segment_3, Triangle_2, Triangle_3, Tetrahedron_3, Circle_2, Sphere_3, ...`
- Predicates: `Collinear_2, Collinear_3, Equal_2, Equal_3, Orientation_2, Orientation_3, SideOfOrientedCircle_2, SideOfOrientedSphere_3, ...`
- Constructions: `ComputeArea_2, ComputeArea_3, ConstructCircumcenter_2, ConstructCircumcenter_3, ComputeSquaredDistance_2, ComputeSquaredDistance_3, ...`

Note that there are geometric constructions that do not perform any computations, like e.g. `ConstructTriangle_2`. These constructions do not impose any requirements on the number types. In the following we will call this type of constructions *trivial constructions*.

There are different approaches to improve the performance of the kernel. An approach to accelerate the predicate evaluation is described by [FV96]. In a first step the predicate is evaluated using a fast but approximate number type with a known error bound. If the result can be certified using the error bound the predicate evaluation is done. Otherwise it resorts to a slower but exact number type restarting the computation from scratch. This approach is based on the idea that in most cases it is sufficient to use an approximate number type to evaluate the predicate. Only in a few cases the exact evaluation must be used. So the overhead of evaluating a few predicates twice is outweighed by the speed-up from using an approximate number type. The described approach is also called "arithmetic filtering". In CGAL the *filtered kernel* provides filtered predicates for the geometry kernel [BBP01, DP03]. Another approach that concentrates on geometric constructions is called "lazy evaluation" [FP06]. In this case the arithmetic expression that has to be evaluated is not evaluated immediately but only when required.

Note that exact evaluation of predicates is easier than exact computation of constructions, so for algorithms that do not perform any geometric constructions a number type providing only exact predicates can be used to improve the performance.

Geometric algorithms

In the big picture an algorithm implemented in CGAL defines a set of geometric objects, predicates, and constructions it requires. These are provided through a template parameter, the so-called *geometric traits* class. Often the CGAL kernel can serve as traits class but sometimes specialized objects, predicates, or constructions are needed, so a specific traits class is required. The traits class itself takes the number type as a template argument. The user can freely choose the number type to be used and he/she can even exchange the provided traits class with his/her own implementation. Along with this goes the responsibility to choose an appropriate number type depending on, e.g., whether or not geometric constructions are used in the algorithm. See Figure 3.1 for a schematic view on the template parameter hierarchy.

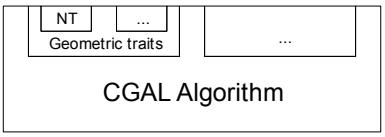


Figure 3.1: The use of templates in a typical CGAL algorithm (NT = number type).

3.2 The CGAL 3D triangulations

This work is based on the CGAL 3D triangulations [PT10b] that have mainly been developed by Sylvain Pion and Monique Teillaud. It computes several types of triangulations of point sets in $\mathbb{E}^3$. CGAL triangulations are stored in the CGAL triangulation data structure [PT10a]. We briefly introduce the implementation with a stress on the parts that are required in the discussion on the implementation of periodic triangulations later on.

Apart from computing 3D triangulations of $\mathbb{E}^3$, there is also functionality for computing Delaunay triangulations and regular triangulations. The algorithm used is incremental, i.e., points are inserted one by one. We now briefly review the software design.

The class `Triangulation_3` implements the functionality to compute and access triangulations of dimension ≤ 3. It is templated by the geometric traits and the data structure. There are two derived classes `Delaunay_Triangulation_3` and `Regular_triangulation_3`. See also Figure 3.2.

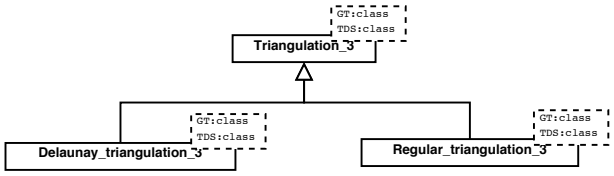


Figure 3.2: Design of the CGAL 3D triangulations, `GT` refers to the geometric traits class and `TDS` to the triangulation data structure.

3.2.1 The triangulation traits

All objects, predicates, and constructions required by the Delaunay triangulation computation are contained in the CGAL Kernel. Thus it is not required to provide a specific geometric traits class do compute Delaunay triangulations.

The geometric traits for computing 3D triangulations must contain

- *Geometric objects*: `Point_3`, `Segment_3`, `Triangle_3`, `Tetrahedron_3`.
- *Geometric predicates*: `CompareXYZ_3`, `Orientation_3`, and a few more for treating degenerate dimensions.
- *Geometric constructions*: The computation of triangulations only requires trivial constructions to construct the geometric objects.

For Delaunay triangulations the predicate `SideOfOrientedSphere_3` is required additionally. It tests for four given points that are not coplanar, whether a fifth point lies inside, outside, or on the sphere defined by the first four points. For the Voronoi output the construction `ConstructCircumcenter_3` is required additionally. And finally, for regular triangulations, the `PowerTest_3` predicate is required instead of the `SideOfOrientedSphere_3` test. Note that `ConstructCircumcenter_3` is the only non-trivial construction and it is only required for the computation of the Voronoi diagram. Thus for computing the Delaunay triangulation it suffices to use a number type that only provides exact predicate evaluation and no exact constructions.

3.2.2 The triangulation data structure

The triangulation data structure implements the data structure to store triangulations of dimension ≤ 3. Formally, it stores a simplicial complex homeomorphic to a three-dimensional compact space without boundaries. As $\mathbb{E}^3$ is not compact, a vertex at infinity v_∞ is added to the point set and the stored complex is a triangulation of $\mathbb{E}^3 \cup \{\infty\}$. Here

we only describe the full-dimensional case, i.e., the case of dimension three because the triangulations of $\mathbb{T}^3_{\boldsymbol{c}}$ that we want to store have always dimension three.

The triangulation data structure stores triangulations in the following way: It stores the *vertices* and the *cells*, where vertices correspond to points in the geometric layer and cells correspond to tetrahedra. Each vertex contains a pointer to one of the cells it is incident to. Additionally it contains the coordinates of the point it corresponds to. The cells store the following information:

- Four pointers to vertices indexed from 0 to 3,
- Four pointers to adjacent cells indexed from 0 to 3, where index i corresponds to the adjacent cell opposite of vertex i.

The lower dimensional simplices, i.e. edges and facets, are implicitly represented as follows: An edge can be accessed through one of the cells it is incident to indicating the two indices that determine the vertices that belong to the edge. A facet can be accessed through one of the two cells it is incident to indicating the index of the vertex that is opposite to the facet, i.e., the only vertex of the cell that does *not* belong to the facet. See Figure 3.3 for an illustration.

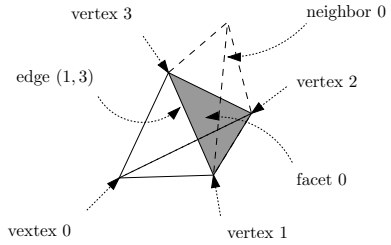


Figure 3.3: Representation of a triangulation in the data structure [PT10a].

The class representing the triangulation data structure is `Triangulation_data_structure_3`. It is templated by a vertex class and a cell class. Default classes are provided but any classes that meet the requirements defined in the respective concepts can be used. From the data structure the vertices and cells are typically accessed using handles. In CGAL `handle` describes a concept that represents pointers. The types to access vertices and cells in the triangulation data structure are named `Vertex_handle` and `Cell_handle`.

3.3 The 3D periodic triangulations

In this section we describe the implementation of the algorithm for computing the periodic Delaunay triangulation of a point set as described in Chapter 2.

We first give naming conventions for the following discussions: We will use the notations $\mathbb{T}^3_{\boldsymbol{c}}$ and $\mathbb{T}^3_{3\boldsymbol{c}}$ as defined in Chapter 2 but assume that $\boldsymbol{c}$ represents a cube of edge length c. Remember that vertices represent orbits of the input points under the action of the group $(\mathbb{Z}^3, +)$ or $(3\mathbb{Z}^3, +)$, when computing in $\mathbb{T}^3_{\boldsymbol{c}}$ or $\mathbb{T}^3_{3\boldsymbol{c}}$, respectively. When computing in $\mathbb{T}^3_{3\boldsymbol{c}}$ we use the following naming conventions. By an *original vertex*, we mean a vertex whose representative in $\mathcal{D}_{3\boldsymbol{c}}$ lies in $\mathcal{D}_{\boldsymbol{c}}$. By a *periodic copy*, we mean a vertex whose representative in $\mathcal{D}_{3\boldsymbol{c}}$ does not lie in $\mathcal{D}_{\boldsymbol{c}}$. See Figure 3.4 for an illustration.

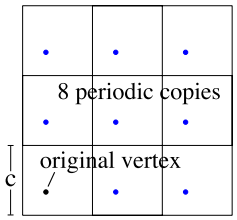


Figure 3.4: (2D illustration) An original vertex with its eight periodic copies in a 9-sheeted covering space of $\mathbb{T}^2_c$.

From Chapter 2 and Section 3.2, we can give the following main differences between $\mathbb{E}^3$ and $\mathbb{T}^3_{\boldsymbol{c}}$ with respect to the implementation:

- use of offsets,
- use of covering spaces,
- no need for a vertex at infinity,
- no need for treating dimension two and smaller.

We now give the overall design of the implementation. Then we present in more detail how the offsets are implemented, how the geometric traits class must be adapted, how the covering spaces are managed and how the different steps of the algorithm must be adapted, such as point location, point insertion and vertex removal. Finally we present additional functionalities, such as access functions specific to $\mathbb{T}^3_{\boldsymbol{c}}$, adaptation of the point location strategies, and dual functions for Voronoi output.

3.3.1 Design

The prefix `Periodic_3` to class names determines the type of periodicity, i.e., it determines that we compute in a space homeomorphic to $\mathbb{T}^3_{\boldsymbol{c}}$, as opposed to other possible periodic spaces, such as orbit spaces of $\mathbb{E}^3$ under the action of a group spanned by two or one translations only, see also Section 5.2.

The design of the CGAL 3D periodic triangulations is very similar to the design of the CGAL 3D triangulations: The class `Periodic_3_triangulation_3` contains all the functionality that is not specific to Delaunay triangulations. That is mainly access functions and the point location functionality. The specialization `Periodic_3_Delaunay_triangulation_3` contains all the Delaunay specific and Voronoi specific functionality. It is possible to add another class `Periodic_3_regular_triangulation_3` as a specialization of `Periodic_3_triangulation_3` in the same way as for the 3D regular triangulations in $\mathbb{E}^3$, see Figure 3.5. Even though there is no conceptual problem, the class `Periodic_3_regular_triangulation_3` has not been implemented.

As in the CGAL 3D triangulations, the periodic triangulation classes have two template parameters: the geometric traits and the triangulation data structure. The first one is described in Section 3.3.3. For the second one the existing implementation of the triangulation data structure can be used: As described in Section 3.2.2, the triangulation data structure stores a simplicial complex without boundary. A triangulation in $\mathbb{T}^3_{\boldsymbol{c}}$ is a simplicial complex homeomorphic to $\mathbb{T}^3_{\boldsymbol{c}}$ and thus without boundary. So we can reuse the

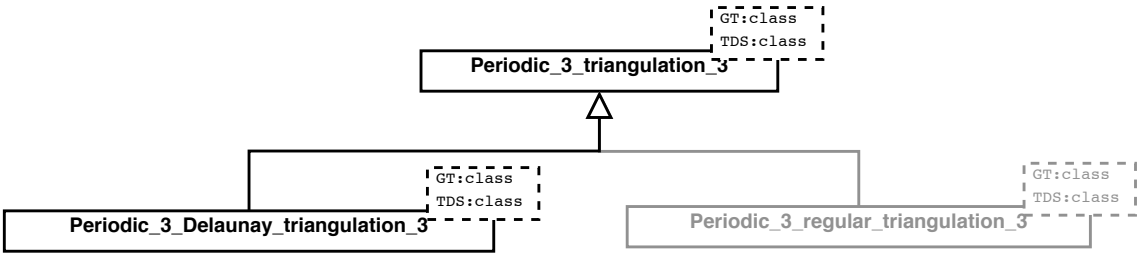


Figure 3.5: Design of the CGAL 3D periodic triangulations, `GT` refers to the geometric traits class and `TDS` to the triangulation data structure.

class `Triangulation_data_structure_3` directly. Only the vertex class and the cell class need to be changed, which is described in Section 3.3.2.

3.3.2 Offsets

As described in Section 2.1, simplices are represented by point-offset pairs, where the points lie in the original domain and the offsets are elements of $\mathbb{Z}^3$. Remember that a point p together with an offset o corresponds to the point $\varphi_{\boldsymbol{c}}(p,o) = p + \boldsymbol{c} * o$ in $\mathbb{E}^3$, i.e., $\{\varphi_{\boldsymbol{c}}(p,o) \mid o \in \mathbb{Z}^3\}$ describes the orbit of p under the action of the group $(\mathbb{Z}^3, +)$. A k-simplex in $\mathbb{T}^3_{\boldsymbol{c}}$ is defined as the projection onto $\mathbb{T}^3_{\boldsymbol{c}}$ of the convex hull of the images of $k+1$ point-offset pairs under $\varphi_{\boldsymbol{c}}$ (Definition 2.1.2). Accordingly, a k-simplex in $\mathbb{T}^3_{3\boldsymbol{c}}$ is given in the same way by the images of $k+1$ point-offset pairs under $\varphi_{3\boldsymbol{c}}$. The triangulation data structure does not know the original domain of the space of triangulation it stores. Offsets are implemented as three-dimensional integer vectors with a comprehensive interface for convenience. We now present the different uses of offsets in the CGAL 3D periodic triangulations.

As described in Section 3.2.2, the triangulation data structure stores only the cells and the vertices of a triangulation. As in the flat torus simplices are defined by point-offset pairs, we must adapt the cell representation of the triangulation data structure as follows. To describe a cell we store four vertices and four offsets to represent a tetrahedron. A vertex represents the orbit of a point of the input point set. The vertex class itself remains unchanged, i.e., it only stores the point of the orbit that lies in the original domain. Note that the offsets must be attached to the cells and cannot be attached to the vertices because a vertex might have different offsets in different cells, see Figure 3.6.

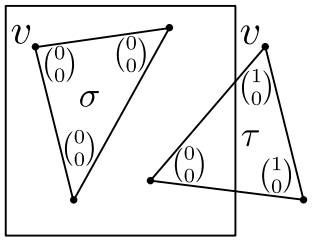


Figure 3.6: (2D illustration) In σ the vertex v has offset $\binom{0}{0}$; in τ the vertex v has offset $\binom{1}{0}$.

Note that for any offset $o \in \mathbb{Z}^3$ the four point-offset pairs (p_i, o_i) and $(p_i, o_i + o)$, $i = 0 \ldots 3$ define the same cell under the respective projection map. For the implementation

we define the following convention in order to have a canonical way of storing cells:

Convention 3.3.1. *Let C be a cell with vertices $v_0, \ldots, v_3$ and corresponding offsets $o_0, \ldots, o_3$ with $o_i = (o_{ix}, o_{iy}, o_{iz})$. Then*

$$\begin{aligned} \min\{o_{0x}, o_{1x}, o_{2x}, o_{3x}\} &= 0 \\ \min\{o_{0y}, o_{1y}, o_{2y}, o_{3y}\} &= 0 \\ \min\{o_{0z}, o_{1z}, o_{2z}, o_{3z}\} &= 0 \end{aligned}$$

Intuitively this means that the representation of the cell in $\mathbb{E}^3$ lies as close as possible towards the origin without allowing for negative offsets. The image under $\varphi_{\boldsymbol{c}}$ of the cell is always entirely contained in the quadrant $x \geq 0$, $y \geq 0$, $z \geq 0$, see Figure 3.7.

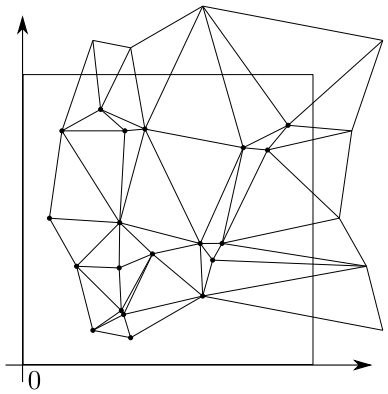


Figure 3.7: (2D illustration) Images of all cells of a periodic triangulation according to Convention 3.3.1.

Note that storing additional information in the cells is critical with respect to memory usage, so we are interested in finding a representation that is as compact as possible. The following lemma shows that one bit per dimension of the space is sufficient.

Lemma 3.3.2. *For Delaunay triangulations of both $\mathbb{T}^3_{\boldsymbol{c}}$ and $\mathbb{T}^3_{3\boldsymbol{c}}$, it is sufficient to encode the offsets using three bits.*

Proof. The claim follows from statement *2* of Lemma 2.2.9: $F^{(2)}$ consists of eight copies of the fundamental cube that can be addressed by offsets using three bits. □

Note that Lemma 3.3.2 actually holds for any torus with a rectangular fundamental domain.

As each cell contains four offsets, which require three bits to be stored according to Lemma 3.3.2, we need to store 12 extra bits per cell. We currently use one `unsigned int` to store the four offsets of the cell. This space-efficient representation is only used to store offsets in the triangulation data structure. For computing with offsets we use a three-dimensional integer vector that can represent a much bigger range of offsets.

For some of the subsequently presented algorithms we need to define neighbor offsets: Consider a Delaunay triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$ or $\mathbb{T}^3_{3\boldsymbol{c}}$. Convention 3.3.1 determines a tetrahedron τ in $\mathbb{E}^3$ to be the canonical representation of a cell c in the Delaunay triangulation. Now let σ be the canonical representation of a cell adjacent to c. Then σ and τ are not necessarily adjacent in $\mathbb{E}^3$, see Figure 3.8.

Definition 3.3.3 (Neighbor offset). *There is an offset ζ such that $\varphi_{\boldsymbol{c}}(\sigma, \zeta)$ is adjacent to τ. We call this offset ζ* neighbor offset *from τ to σ.*

The neighbor offset can be determined as follows: Let v be one of the common vertices of τ and σ. The neighbor offset is the difference between the offset of v in τ and the offset of v in σ, see Figure 3.8.

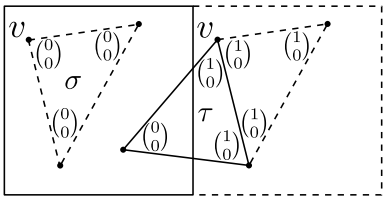


Figure 3.8: (2D illustration) The neighbor offset from τ to σ is $\binom{1}{0}$.

Neighbor offsets are used in the implementation of the remembering stochastic walk for the point location and in the implementation of `find_conflicts` that determines the conflict region of a point.

To deal with points that are not vertices of the triangulation we use the following notation: Let p_v denote the point of the vertex v of the triangulation. By "offset o of a vertex v with respect to a point p" we mean the offset of the endpoint p_v in the edge given by the endpoints $\varphi_{\boldsymbol{c}}(p, 0)$ and $\varphi_{\boldsymbol{c}}(p_v, o)$.

3.3.3 Traits

The `GeometricTraits` concept for periodic triangulations is essentially the same as for CGAL 3D triangulations. The main change is that each predicate and construction can take points and corresponding offsets as arguments additionally to the version that takes only points. In order to provide the new predicates and constructions we have an adapter that works as described by the following pseudo-code listing:

```
bool periodic_predicate(Point p1, ..., Point pk,
                        Offset o1, ..., Offset ok) {
  return predicate(p1 + c * o1, ..., pk + c * ok);
}
```

The `periodic_construction` is defined in the same way. Note that the edge length `c` of the original domain is required to perform the computation of the predicate or construction. The original domain is logically associated with the triangulation, so it is a member of the class `Periodic_3_triangulation_3` and not of the traits class. We endow the traits class with a pointer to the original domain stored in the class `Periodic_3_triangulation_3`.

The predicates and constructions taking only points would not be required by the algorithm but we provide them for efficiency reasons: They can be called if all offsets are zero. In this case no translations due to offsets need to be performed and these predicates and constructions do considerably less arithmetic operations.

Exact evaluation As described in Section 3.2.1, the CGAL 3D triangulations implementation requires exact predicates and no exact constructions. The output is a combinatorial

structure that contains only unmodified input points. Computing the Voronoi diagram is the only case where non-trivial constructions are needed, namely for computing the circumcenter of tetrahedra.

For periodic triangulations the situation is slightly different. During an algorithm run some points have to be translated in order to embed simplices in $\mathbb{E}^3$ for evaluation of geometric predicates, see Figure 3.9. Translating points is a geometric construction, and if

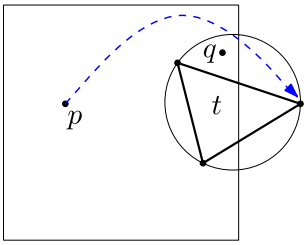


Figure 3.9: (2D illustration) To test on which side of the circumcircle of t a point q lies, p must be translated first.

it is not exact, the correct output of an algorithm run and even its termination cannot be guaranteed. On the other hand we do not want to require exact constructions for efficiency reasons. To avoid constructions, we regroup the translations and predicate evaluations in new predicates. That is why we introduced the predicates taking point-offset pairs as arguments above: The translation of the points is now part of the predicate evaluation. This means the coordinates of the translated points are never represented in memory but only treated symbolically. In this way all the known and highly efficient techniques for exact predicate evaluation can be applied. For the same reason, when computing in $\mathbb{T}^3_{3\boldsymbol{c}}$, the points outside $\mathcal{D}_{\boldsymbol{c}}$ store the coordinates of the corresponding points in $\mathcal{D}_{\boldsymbol{c}}$ and are assigned an offset. The coordinates of points outside of $\mathcal{D}_{\boldsymbol{c}}$ are never computed explicitly.

3.3.4 Covering spaces

Here we describe how our implementation handles covering spaces and how we handle conversion between different covering spaces. Note that in general, triangulations of small point sets require computation in $\mathbb{T}^3_{3\boldsymbol{c}}$, whereas for large point sets we can compute in $\mathbb{T}^3_{\boldsymbol{c}}$. So, one very important implementation issue is that there should not be any overhead for handling covering spaces when computing in $\mathbb{T}^3_{\boldsymbol{c}}$, i.e. without additional copies.

In order to compute in $\mathbb{T}^3_{3\boldsymbol{c}}$, 27 copies of each point are inserted. Remember that in $\mathbb{T}^3_{3\boldsymbol{c}}$, we call vertices with a representative in $\mathcal{D}_{\boldsymbol{c}}$ *original vertices*, and other vertices *periodic copies*.

Copy shift When computing in $\mathbb{T}^3_{3\boldsymbol{c}}$, vertices represent orbits under the action of the group $(3\mathbb{Z}^3, +)$. So a point in $\mathcal{D}_{3\boldsymbol{c}}$ of an orbit represented by a vertex does not necessarily lie in $\mathcal{D}_{\boldsymbol{c}}$. However, the input point set of the algorithm is contained in $\mathcal{D}_{\boldsymbol{c}}$. According to Section 3.3.2 the triangulation data structure stores for each vertex the coordinates of a point in $\mathcal{D}_{3\boldsymbol{c}}$. According to Section 3.3.3, the computation of the point coordinates in $\mathcal{D}_{3\boldsymbol{c}}$ of a periodic copy of an input point is a geometric construction. In order to avoid this geometric construction, we store for each periodic copy the corresponding input point p in $\mathcal{D}_{\boldsymbol{c}}$ and a *copy shift* s, such that $\varphi_{\boldsymbol{c}}(p, s)$ is the periodic copy in $\mathcal{D}_{3\boldsymbol{c}}$. Copy shifts are

only required when computing in $\mathbb{T}^3_{3\boldsymbol{c}}$, so we do not store them in the triangulation data structure to avoid memory overhead. They are stored in a separate data structure in the triangulation class itself as described below. Like offsets, copy shifts are elements of $\mathbb{Z}^3$. In the implementation we reuse the offset class to represent copy shifts.

Data structures Here we describe a data structure to recover the copy shift of a given vertex. This data structure additionally stores the information which vertices are periodic copies of each other, an information that is not contained in the triangulation data structure either. We decided to use a `std::map` with `Vertex_handle` as key type and `std::pair<Vertex_handle,Copy_shift>` as value type to retrieve for each vertex that is not an original vertex its corresponding original vertex and the copy shift. We call this map `covering_map`, because it actually corresponds to the mathematical definition of the covering map, see Definition 1.3.1. It is stored in the class `Periodic_3_triangulation_3`. See Figure 3.10 for an illustration of the use of the `covering_map`.

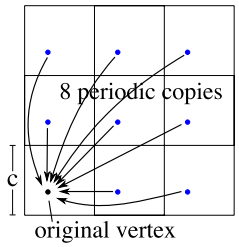


Figure 3.10: (2D illustration) The `covering_map` implements the arrow relation.

For some algorithms, we do not only need to find the original copy from a given vertex but we must also find all periodic copies of a given original vertex. There is another map that stores for each original vertex all its periodic copies. The key type of this map is again `Vertex_handle`, the value type is `std::vector<Vertex_handle>`. Here the copy shifts are implicitly encoded as ternary numbers in the index of the periodic copy in the vector; i.e., the copy with copy shift (i, j, k) is stored as the vector element of index $i \cdot 3^2 + j \cdot 3^1 + k \cdot 3^0$. We call this map `reverse_covering_map`.

Keep track of the edge length We implemented Criterion 2.3.4 in order to decide when to convert the triangulation to $\mathbb{T}^3_{\boldsymbol{c}}$. To avoid overhead when computing in $\mathbb{T}^3_{\boldsymbol{c}}$, we maintain an auxiliary data structure that contains all edges that are too long according to Criterion 2.3.4. This data structure only contains data when computing in $\mathbb{T}^3_{3\boldsymbol{c}}$, otherwise it is empty. It works in the following way: The used data structure is a `std::map` with key type `Vertex_handle` and value type `std::list<Vertex_handle>`. If an edge is too long, one of its vertices is stored as a key and the other one is contained in the corresponding list. We call this data structure `long_edges_list`, see Figure 3.11.

In order not to store edges twice we require a comparison function for `Vertex_handle`s and define the convention that a `Vertex_handle` in a key is always smaller than any of the `Vertex_handle`s in the corresponding list. For the comparison function we compare the memory addresses of the `Vertex_handle`s.

When we are computing in $\mathbb{T}^3_{3\boldsymbol{c}}$, each time we insert a new point or remove a vertex we update the `long_edges_list`. In case of a point insertion we remove all edges of the

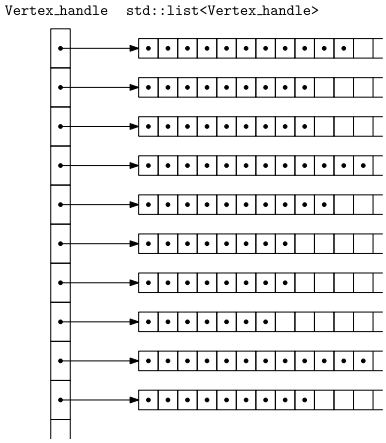


Figure 3.11: The `long_edges_list`.

conflict region that are contained in the `long_edges_list`. Then we check for each edge of the newly inserted cells whether it is longer than the threshold in Criterion 2.3.4 and if so, we insert it into the `long_edges_list`. Note that this approach removes and reinserts the same edges from the boundary of the hole, which is redundant. However, it is quite difficult to test for a given edge whether it lies on the boundary of the hole. It turns out to be faster to remove and reinsert some edges redundantly, rather than testing for all of them, whether they lie on the boundary of the hole.

In case of vertex removal exactly the same approach is used.

Conversion from $\mathbb{T}^3_{3\boldsymbol{c}}$ to $\mathbb{T}^3_{\boldsymbol{c}}$ Once the `long_edges_list` is empty, the current triangulation can be converted to $\mathbb{T}^3_{\boldsymbol{c}}$. To do so, we iterate four times over all cells and once over all vertices (see also Figure 3.12):

Cell

1. Mark all cells that will be deleted. Cells are deleted if they are not canonical with respect to the original domain of $\mathbb{T}^3_{\boldsymbol{c}}$ in the sense of Convention 3.3.1.
2. For all unmarked cells, redirect the neighbor pointers to the new neighbors as shown in Figure 3.12, if they point to a marked cell. The new neighbors can be found using the `covering_map`.
3. Note that vertices will be deleted if they are not original vertices but periodic copies. For all unmarked cells, redirect the vertex handles that point to vertices that are to be deleted to the corresponding original vertices.
4. Delete all marked cells.

Vertex Delete all vertices that are not original vertices.

Conversion from $\mathbb{T}^3_{\boldsymbol{c}}$ to $\mathbb{T}^3_{3\boldsymbol{c}}$ According to Criterion 2.3.4, when computing in $\mathbb{T}^3_{\boldsymbol{c}}$ we will never need to switch back to $\mathbb{T}^3_{3\boldsymbol{c}}$ if we just keep inserting points. However, vertex removal can make it necessary to switch back to $\mathbb{T}^3_{3\boldsymbol{c}}$. So during each vertex removal – even when computing in $\mathbb{T}^3_{\boldsymbol{c}}$ – we need to check that the newly created cells do not have edges that are longer than the threshold in Criterion 2.3.4.

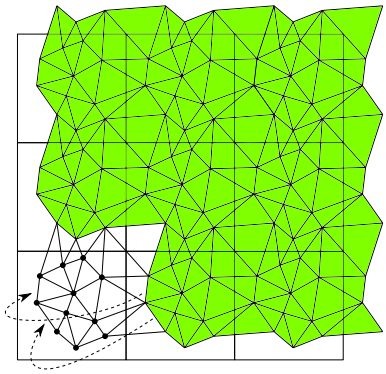


Figure 3.12: (2D illustration) The colored cells will be deleted, the vertices that will be deleted are not drawn. The arrows exemplary indicate the new adjacency relations.

The conversion from $\mathbb{T}^3_{\boldsymbol{c}}$ to $\mathbb{T}^3_{3\boldsymbol{c}}$ turns out to be more complicated than the conversion from $\mathbb{T}^3_{3\boldsymbol{c}}$ to $\mathbb{T}^3_{\boldsymbol{c}}$ described above. It is done in the following way: First create 26 copies of each vertex and construct the `covering_map` and the `reverse_covering_map`. Iterate over all cells and store the vertex offsets and neighbor offsets in some temporary data structures. Now create 26 periodic copies of each cell, obtaining their vertices from the `reverse_covering_map` using the vertex offsets from the temporary data structure. Note that all the original cells are completely contained inside the original domain of $\mathbb{T}^3_{3\boldsymbol{c}}$. Thus they will now turn to cells with all offsets zero, even the ones that had non-zero offsets before. The vertices of non-zero offsets must be replaced by their periodic copies with the respective copy shift according to the `reverse_covering_map`. It remains to set the neighbor relations of both the newly created cells and the original cells, the cell pointers of the vertices, the vertex offsets in the cells and finally to set up the `long_edges_list`.

3.3.5 Point location

The point location is guaranteed to work only for input points given in the original domain.

The remembering stochastic walk described in Section 1.2.4 works in periodic triangulations as well if we use the representations in $\mathbb{E}^3$ of the simplices for the geometric tests. Each time we walk from one cell to the next we need to keep track of the neighbor offsets in order to maintain the offset of the input point with respect to the vertices of the current cell, see Figure 3.13.

The walk is started at a random cell of the triangulation. If this starting cell has a non-empty intersection with the boundary of the fundamental domain, it can happen that it is very close to the point to locate or even contains it but its canonical representation in $\mathbb{E}^3$ according to Convention 3.3.1 appears on the opposite side of the fundamental domain. This can be detected easily by testing whether the starting cell has at least one non-zero offset. In this case – depending on the coordinates of the point – we start the point location at the periodic copy of the cell at the opposite side of the fundamental domain, see Figure 3.14. This prevents the walk from traversing the whole domain, while the starting cell and the cell containing the point are actually close. This case occurs especially when using the spatial sorting, see Section 3.3.9.

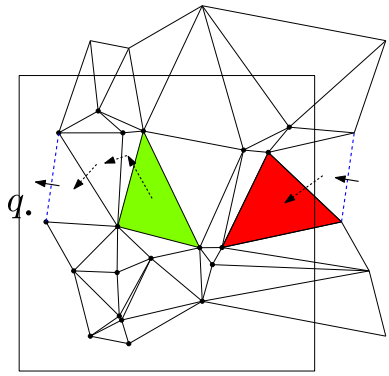


Figure 3.13: (2D illustration) When traversing the dashed edge, the offset of the point with respect to the current cell changes. Green triangle: starting cell, red triangle: cell containing the query point q.

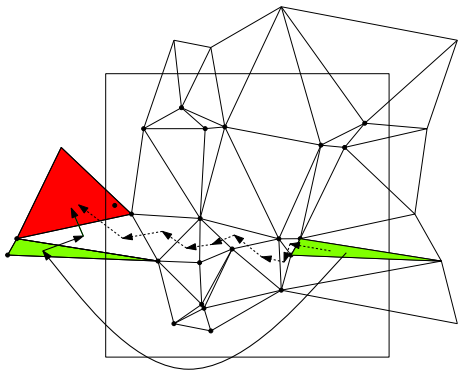


Figure 3.14: (2D illustration) As the point is in the left half of the fundamental domain, we start the walk at a more appropriate periodic copy of the initial cell. Green triangle: starting cell, red triangle: cell containing the query point q.

3.3.6 Point insertion

If we are computing in $\mathbb{T}^3_{\boldsymbol{c}}$, then the point insertion can be implemented in exactly the same way as described in Section 1.2.4. If we work in $\mathbb{T}^3_{3\boldsymbol{c}}$, then each point must be inserted 27 times by the insertion algorithm as described in Algorithm 2.3.1 and additionally we must maintain the maps for the periodic copies of vertices as well as the `long_edges_list`.

As described in Section 1.2.4, the insertion step first identifies the cells that are in conflict with the newly inserted point. Then it deletes these cells, leaving a hole in the data structure that is refilled by new triangles with the new point as a vertex. If the boundary of the hole and the boundary of the fundamental domain have non-empty intersection, some of the new cells filling the hole will have vertices with non-zero offsets (see Figure 3.15). It would be possible to compute these offsets by walking through the cells outside the hole while keeping track of the neighbor offsets. But this approach would be quite costly in terms of running time. We use the following, more time-efficient but less space-efficient approach:

- During the step of identifying cells in conflict with the new point, we compute the offsets of each vertex of each cell in conflict with respect to the new point. These

offsets are temporarily stored in the vertices.

- After having created the new cells, we only need to get the stored offsets from the vertices to correctly set the offsets in the cells.
- After point insertion the offsets that are stored in the vertices have to be cleared off, in order to not interfere with future point insertions.

This approach requires to store one offset in each vertex and thus requires the use of some extra memory. Given that the number of vertices in a triangulation is generally small compared to the number of cells (see Lemma 3.4.1), this memory overhead is acceptable. This is the only case where offsets are attached to vertices.

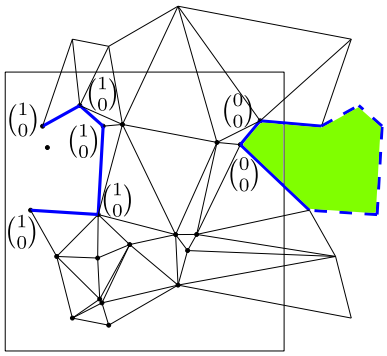


Figure 3.15: (2D illustration) The boldfaced colored lines show the boundary of the hole during insertion of the point. The offsets attached to the hole's vertices are shown; they indicate how to translate the vertices in order to get the shaded polygon.

Note that this approach only works if the hole is homeomorphic to a ball. The hole is the union of all simplices of the star of the newly inserted point. Thus if the structure after the point insertion is a simplicial complex then the hole is homeomorphic to a ball, cf. Lemma 2.2.6 and Theorem 2.2.8.

3.3.7 Vertex removal

In principle, the algorithm for removing vertices works in the same way as described in Section 1.2.4.

Remember that the first step of the vertex removal consists in deleting all tetrahedra that contain the vertex from the triangulation, yielding a hole. By Theorem 2.2.8 (page 24), this hole is always homeomorphic to a ball. The interior of the hole must be triangulated and this triangulation must be sewed in the current data structure. As the hole is homeomorphic to a ball, the triangulation of the hole is computed as the Delaunay triangulation of $\mathbb{E}^3$ defined by the vertices of the hole. This is done using the CGAL 3D triangulation implementation.

There still is an implementation issue: The triangulation of the hole must be sewed into the hole created by deleting the vertex from the periodic triangulation. However, the triangulation data structures of a CGAL triangulation and a CGAL periodic triangulation are not type compatible. That is because for the periodic triangulations different vertex and cell classes for storing the offsets are required. So it is not sufficient to copy the

primitives during the sewing process but they must also be converted to objects of the right type.

Additionally, when computing in $\mathbb{T}^3_{3\boldsymbol{c}}$ the `long_edges_list` needs to be updated as well as the `covering_map` and the `reverse_covering_map`.

When computing in $\mathbb{T}^3_{\boldsymbol{c}}$, we have to verify after each removal that there are no edges that are too long according to Criterion 2.3.4. Otherwise the triangulation has to be converted to $\mathbb{T}^3_{3\boldsymbol{c}}$.

3.3.8 Access

As described in Section 3.2.2, the vertices store the corresponding point coordinates, i.e., the coordinates of the vertex of index `i` in cell `c` can be accessed by `c->vertex(i)->point()`, for `i`$\in \{0,1,2,3\}$. However, in order to project a simplex into $\mathbb{E}^3$, the offsets and the edge length of the original domain must be taken into account. There are two things to consider: (1) The original domain of $\mathbb{T}^3_{\boldsymbol{c}}$ is different from the original domain of $\mathbb{T}^3_{3\boldsymbol{c}}$. (2) When computing in $\mathbb{T}^3_{3\boldsymbol{c}}$, the point coordinates outside of $\mathcal{D}_{\boldsymbol{c}}$ must be computed using the `covering_map`. To hide these technicalities from the user, we introduce a new member function `point` to the periodic triangulations: The call to `c->vertex(i)->point()` can now be replaced by a call to `this->point(c,i)`. Instead of fetching the point coordinates from the triangulation data structure directly, the function `point(c,i)` also fetches the offsets from the data structure and the point coordinates and copy shifts from the `covering_map` if the triangulation is represented in $\mathbb{T}^3_{3\boldsymbol{c}}$.

To access the simplices of the triangulation, all the iterators and circulators of the triangulation data structure are available. However, note that the triangulation data structure does not have the information whether it stores a triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$ or of $\mathbb{T}^3_{3\boldsymbol{c}}$. If the triangulation is represented in $\mathbb{T}^3_{3\boldsymbol{c}}$, the iterator returns 27 copies of each simplex. It is not possible to filter out copies at the data structure level. This also has a mathematical reason: We compute in $\mathbb{T}^3_{3\boldsymbol{c}}$ because the triangulation does not exist in $\mathbb{T}^3_{\boldsymbol{c}}$. If iterators would return only one copy of each simplex, this would lead to inconsistencies, e.g. in the adjacency and incidence relations because returned simplices do not form a triangulation, see Figure 3.16 for an example. For some applications it is useful to have only one copy of each simplex even if the returned set of simplices does not form a triangulation.

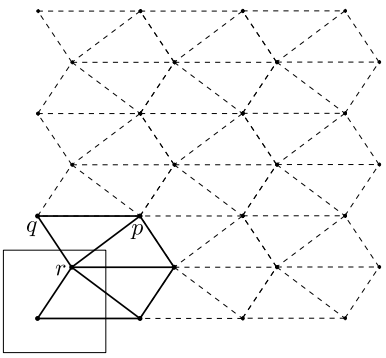


Figure 3.16: (2D illustration) An iterator returning only one periodic copy of each simplex, returns for instance the face pqr but only one of its three adjacent faces.

To provide such functionality without inconsistencies, we have some *geometric iterators*

that only output simplices, without adjacency and incidence relations. Then problems like those shown in Figure 3.16 cannot occur. These geometric iterators exist for all four types of simplices: points, segments, triangles, and tetrahedra. There are four different options to specify which simplices will be returned, see also Figure 3.17:

- `STORED` Iterates over all the simplices that are stored, i.e., it shows the same behavior as the iterators from the triangulation data structure.
- `STORED_COVER_DOMAIN` Iterates over all simplices that have non-empty intersection with the fundamental domain of the flat torus the triangulation is stored in, i.e. 27 copies of the original domain in case of $\mathbb{T}^3_{3\mathbf{c}}$. This means that the simplices that intersect the boundary of the current fundamental domain will be returned several times.
- `UNIQUE` Iterates only over the original copy of each simplex.
- `UNIQUE_COVER_DOMAIN` Iterates only over those simplices that have non-empty intersection with the original domain. As for `STORED_COVER_DOMAIN` boundary simplices will be returned several times.

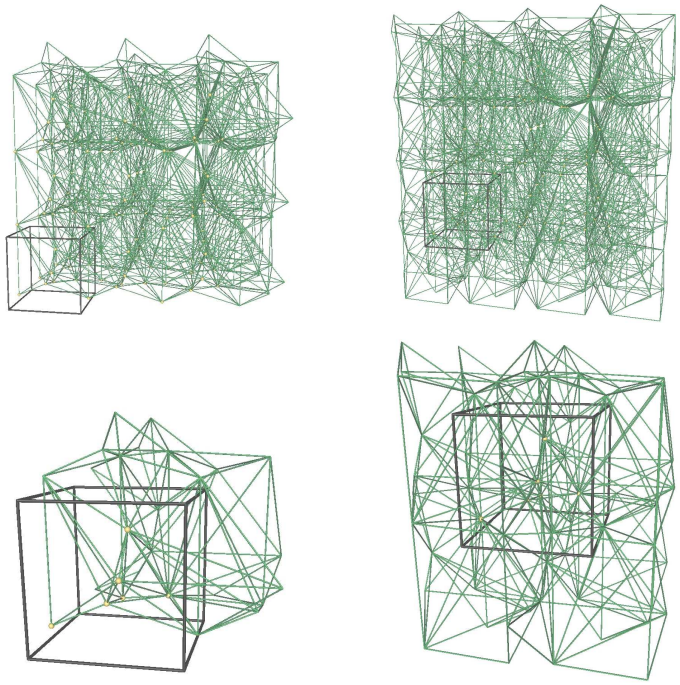


Figure 3.17: The four modes for the geometric iterators. Top left: `STORED`. Top right: `STORED_COVER_DOMAIN`. Bottom left: `UNIQUE`. Bottom right: `UNIQUE_COVER_DOMAIN`. Here the triangulation is represented in $\mathbb{T}^3_{3\mathbf{c}}$ and the tetrahedra output by the iterators are shown.

3.3.9 Optimizations

There are several optimizations available for CGAL 3D triangulation computation as described in Section 1.2.4. These optimizations can be adapted to periodic triangulations.

Hierarchy As described in Section 1.2.4 the triangulation hierarchy maintains a finite number of layers of triangulations, where the upper layers are coarser than the lower layers. To connect the layers, each vertex has a pointer to its corresponding vertex in the layer below (`down`) and in the layer above (`up`) if there is one in the layer above. Note that each vertex has a corresponding vertex in the layer below, so the `down`-pointer always points to some vertex except for vertices in the lowermost layer.

For periodic triangulations it is highly probable that the coarser (upper) layers are represented in $\mathbb{T}^3_{3\boldsymbol{c}}$ while the finer (lower) layers are represented in $\mathbb{T}^3_{\boldsymbol{c}}$. We index the layers from bottom to top, i.e., the lowermost layer has index 0. Let layer i be the uppermost layer represented in $\mathbb{T}^3_{\boldsymbol{c}}$; so layer $i+1$ is represented in $\mathbb{T}^3_{3\boldsymbol{c}}$. Then all the periodic copies of original vertices in layer $i+1$ do not have corresponding vertices in the lower layer i, and thus their `down`-pointer is not set. To avoid undefined `down`-pointers, we let all the `down`-pointers of periodic copies point to the original vertex instead. Periodic copies do not have `up`-pointers as they are not required, see Figure 3.18).

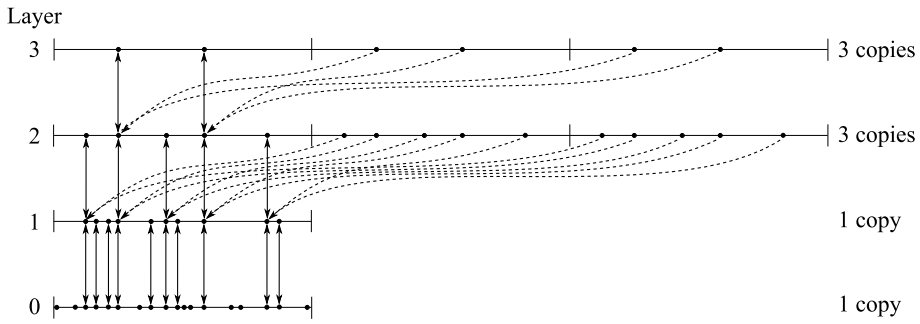


Figure 3.18: (1D illustration) `up`- and `down`-pointers of periodic copies point to the original vertex one layer lower.

Spatial sorting The spatial sorting as described in Section 1.2.4 can be reused without any modification: The output sequence of the spatial sorting has the property that two points that are close in the sequence have small Euclidean distance in $\mathbb{E}^d$. As the input point set for the periodic triangulation lies in $\mathcal{D}_{\boldsymbol{c}}$, which is a subset of $\mathbb{E}^d$, this property remains true.

3.3.10 Additional functionality

We provide some additional functionality, i.e. functionality that is not required to compute the Delaunay triangulation itself but is useful for the user interface.

Force conversion between covering spaces As shown in Section 2.3 the Criterion 2.3.4 is sufficient but not necessary to decide whether the current point set defines a Delaunay triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$. That is, point set can define a triangulation in $\mathbb{T}^3_{\boldsymbol{c}}$ even if Criterion 2.3.4 is not fulfilled. A necessary condition is given by Theorem 2.2.8. However, it is possible that after adding a point to the point set it does not define a triangulation in $\mathbb{T}^3_{\boldsymbol{c}}$ anymore. Nevertheless, for a triangulation that is not going to be changed anymore, it might be useful to convert it from $\mathbb{T}^3_{3\boldsymbol{c}}$ to $\mathbb{T}^3_{\boldsymbol{c}}$. The function `is_triangulation_in_1_sheet()` implements the

test for the condition in Theorem 2.2.8. The functions `convert_to_1_sheeted_covering()` and `convert_to_27_sheeted_covering()` can be used to convert between $\mathbb{T}^3_{\boldsymbol{c}}$ and $\mathbb{T}^3_{3\boldsymbol{c}}$.

Voronoi diagram As in the CGAL 3D Delaunay triangulation, we provide functionality for computing Voronoi diagrams. Note that the computation of Voronoi diagrams requires geometric constructions and thus is exact only if the kernel provides exact constructions.

To return the Voronoi diagram, we only need functions that for a given simplex return its dual. This yields the following types of geometric objects:

Delaunay	*Voronoi*
Cell	Point
Facet	Segment
Edge	Polygon
Vertex	Polyhedron

Note that polygons and polyhedra are not simplices and do not have constant size, so there are no kernel objects in CGAL suited to represent them. In the CGAL 3D Delaunay triangulations only dual functions for facets and cells are available, which is sufficient to describe the Voronoi diagram. The duals of vertices and edges can be constructed easily from this output. In the periodic case the dual functions for facets and cells are sufficient, too, for describing the dual periodic Voronoi diagram. However, for the dual polygons and dual polyhedra some vertices might need to be translated in order to map the polyhedra and polygons into $\mathbb{E}^3$. It is a bit more involved to compute the offsets that determine these translations using the cell and facet dual functions only. Therefore, we also provide dual functions for vertices and edges that use the incidence relations in the Delaunay triangulation to reconstruct the vertex offsets of the polyhedra and polygons. This is more efficient than an ad-hoc approach using the dual functions for facets and cells only. The dual functions for vertices and edges output the respective objects as point lists through an output iterator. As the computation of a Voronoi vertex requires a geometric construction, there is no reason to avoid further geometric constructions for translating points. Thus we return the translated points instead of pairs of points and offsets.

We implement a helper function `periodic_circumcenter` that computes the circumcenter for a given cell and returns the point inside the domain and an offset that specifies the periodic copy of this point that corresponds geometrically to the circumcenter of the canonical representation of the input cell in $\mathbb{E}^3$. This offset cannot be inferred directly from the offsets of the cell because there is no relation between the offset of a circumcenter of a cell and the offsets of the cell itself, see Figure 3.19. Thus geometric tests must be applied to find the right offset in $\{-1, 0, 1\}^3$.

The duality functions then work as follows:

Cell Returns the circumcenter of the cell inside $\mathcal{D}_{\boldsymbol{c}}$ as computed by `periodic_circumcenter`.

Facet Returns the dual edge as a pair of its endpoints represented by point-offset pairs following Convention 3.3.1.

Edge The dual of an edge is a polygon. We return an ordered list of points that define this polygon. Note that these points do not necessarily lie all inside the domain. We calculate the polygon as follows: Circulate over all cells incident to the edge. For each cell compute the circumcenter and the required offset with respect to one of the vertices of the input edge using the helper function `periodic_circumcenter`.

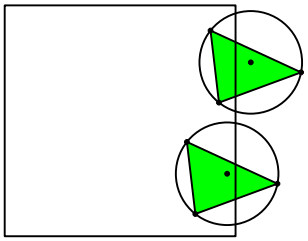


Figure 3.19: (2D illustration) The offsets of a triangle do not determine the offset of its circumcenter.

Vertex The dual of a vertex is a polyhedron. We return a list of points that define this polyhedron as their convex hull. As above, these points do not necessarily lie all inside the domain but are chosen such that their convex hull is a projection of the dual of the vertex into $\mathbb{E}^3$. In order to obtain the polyhedron itself the adjacency relations can be extracted from the Delaunay triangulation.

3.4 Complexity

As seen in Section 2.4.1, Algorithm 2.3.1 has the same randomized worst-case complexity as the algorithm for computing Delaunay triangulations in $\mathbb{E}^3$. However, this is not true for the implementation: As explained in Section 3.3.4, when computing in $\mathbb{T}^3_{3\boldsymbol{c}}$ the coordinates of points that do not lie in $\mathcal{D}_{\boldsymbol{c}}$ are not stored in the triangulation data structure for memory efficiency reasons. They are available through the additional data structure `covering_map` that does not allow for constant time access. Again to avoid memory overhead, when computing in $\mathbb{T}^3_{3\boldsymbol{c}}$ we do not mark the too long edges in the data structure directly but we keep track of them in the `long_edges_list`, as described in Section 3.3.4. As the `covering_map`, this data structure does not allow for constant time access. In this section we examine the impact of the `covering_map` and the `long_edges_list` on the time complexity of the implementation.

The use of these two additional data structures is a choice motivated by practical considerations: For most applications it seems much more important to be highly time and space efficient when computing in $\mathbb{T}^3_{\boldsymbol{c}}$, whereas the efficiency of computation in $\mathbb{T}^3_{3\boldsymbol{c}}$ is less important. In order to achieve the bound of $O(n^2)$ on the complexity as proved in Section 2.4.1, we would need to store the information contained in the `covering_map` and the `long_edges_list` directly in the triangulation data structure to obtain constant time access. However, the memory reserved for the data members required to store this data in the cells and vertices would also be reserved when the triangulation is represented in $\mathbb{T}^3_{\boldsymbol{c}}$, causing a non-negligibly higher memory consumption of the triangulation data structure.

As in Section 2.4.1, we give the expected complexity of the randomized algorithm for a worst-case input. The expected complexity is the expectation value for the complexity over all possible insertion orders.

Let $\mathcal{S}$ be a set of n vertices and let e, and m denote the number of edges, and cells, of the Delaunay triangulation defined by $\mathcal{S}$ as computed by Algorithm 2.3.1. Let N denote the number of vertices and M the number of cells of the triangulation for which Criterion 2.3.4 applies, i.e. for which Algorithm 2.3.1 switches to the $\mathbb{T}^3_{\boldsymbol{c}}$. If the triangulation of n points is

still computed in $\mathbb{T}^3_{3\boldsymbol{c}}$, then $N = n$ and $M = m$. The inclusions $O(1) \subseteq O(N) \subseteq O(n)$ and $O(1) \subseteq O(M) \subseteq O(m)$ hold. The access to vertices in the `covering_map` is in $O(\log N)$, the access to edges in the `long_edges_list` one of the vertices must be found first, which is in $O(\log N)$, and then the second vertex must be searched for in the corresponding list, which is in $O(N)$. Thus the overall complexity for access to the `long_edges_list` is $O(N)$.

Lemma 3.4.1. *Let T be a triangulation of a compact 3-manifold without boundary that has n vertices and m cells. Then the following two statements hold.*

1. *The number of edges of T is given by $e := n + m$.*
2. *The number of vertices is not larger than the number of cells: $n \leq m$.*

Proof. Let f denote the number of facets. The Euler characteristic χ of any 3-manifold without boundary is 0 [Hen79] and is given by the formula

$$\chi = n - e + f - m.$$

Furthermore, for a triangulation we know that $2f = 4m$. Applying this to the above formula yields $0 = n - e + 2m - m$ and thus $e = n + m$, which proves the first statement. Each facet has three edges and each edge is incident to at least three facets, so $3f \geq 3e$. Plugging this into the formula above yields $0 = n - e + f - m \geq n - e + e - m = n - m$ and thus $n \leq m$. □

From Lemma 3.4.1 and well-known triangulation properties (Section 1.2.2) we know that

$$O(1) \subseteq O(n) \subseteq O(m) = O(e) \subseteq O(n^2)$$

and $O(N) \subseteq O(M)$.

Maintenance of the `long_edges_list` In Section 2.4.1, we show that the total number of edges that are inserted into and removed from the `long_edges_list` is proportional to the simplices that are created and destroyed by the algorithm. After the conversion from $\mathbb{T}^3_{3\boldsymbol{c}}$ to $\mathbb{T}^3_{\boldsymbol{c}}$, the `long_edges_list` is not used anymore, so the number of changes to the `long_edges_list` is proportional to N^2. As in our implementation each access to a vertex is in $O(N)$, the maintenance of the `long_edges_list` is in $O(N^3)$.

We now discuss the functionalities presented in Section 3.3.

Conversion from $\mathbb{T}^3_{3\boldsymbol{c}}$ to $\mathbb{T}^3_{\boldsymbol{c}}$ The most expensive steps are the four iterations over all M cells. Three of them require to read copy shifts from the `covering_map`, which is in $O(\log N)$. The overall complexity is $O(M \log N)$.

Conversion from $\mathbb{T}^3_{\boldsymbol{c}}$ to $\mathbb{T}^3_{3\boldsymbol{c}}$ Again, the most expensive steps are the iterations over all cells. Unlike above, in some of these iterations elements need to be inserted or found in the local covering map on cells, which is in $O(\log M)$.

The construction of the `covering_map` is in $O(N \log N)$.

Additionally, in the end we need to iterate over all edges in order to find the edges that are too long. According to Lemma 3.4.1 there are $O(M)$ many edges. The insertion of edges into the `long_edges_list` is in $O(N)$ and thus the overall complexity of the function remains $O(MN)$.

Point location When computing in $\mathbb{T}^3_{3\boldsymbol{c}}$, for the vertices of each visited cell the copy shifts must be computed ($O(\log N)$). In the worst-case $O(M)$ cells must be visited, which yields $O(M \log N)$. When computing in $\mathbb{T}^3_{\boldsymbol{c}}$ the usual bound of $O(m)$ applies.

Point insertion When computing in $\mathbb{T}^3_{3\boldsymbol{c}}$, the subroutine that finds the cells in conflict has to visit $O(N)$ cells in the worst case and it has to get the copy shifts for the vertices of each of these cells, so this takes $O(N \log N)$. Additionally, the `long_edges_list` must be updated for up to $O(N)$ edges, so the overall bound is $O(N^2)$. When computing in $\mathbb{T}^3_{\boldsymbol{c}}$, the complexity remains $O(n)$.

Vertex removal The complexity of the vertex removal is in $O(n^2)$ in the worst case. When computing in $\mathbb{T}^3_{3\boldsymbol{c}}$ the copy shifts for the hole vertices must be extracted from the `covering_map`, which is in $O(N \log N)$ and up to $O(N)$ edges must be removed from the `long_edges_list`, which is in $O(N^2)$, yielding an overall worst-case complexity of $O(N^2)$.

Hierarchy In Section 2.4.1, we discuss that the triangulation hierarchy of [Dev02] can be used for periodic triangulations, too. Here, we analyze the impact of computing in $\mathbb{T}^3_{3\boldsymbol{c}}$ on the complexity of both point location and point insertion when using the Delaunay hierarchy.

Let α denote the ratio between the expected number of points in two adjacent layers. According to [Dev02] the expected complexity of the point location in some layer i is $O(\alpha^{i+1}n)$. According to the discussion on the complexity of the point location, if layer i is represented in $\mathbb{T}^3_{3\boldsymbol{c}}$, the expected complexity of the point location will be $O(\alpha^{i+1}n \log N)$. Let layer I be the lowermost[1] layer that is represented in $\mathbb{T}^3_{3\boldsymbol{c}}$. Then the total complexity of the point location is:

$$\sum_{i=0}^{i<I} \alpha^{i+1}n + \sum_{i=I}^{i=\infty} \alpha^{i+1}n \log N = n \cdot \left(\frac{\alpha(1-\alpha^I)}{1-\alpha} + \frac{\alpha^{I+1}}{1-\alpha} \log N \right) \in O(n \log N)$$

Note that the coefficient $\frac{\alpha^{I+1}}{1-\alpha}$ of $\log N$ will get very small with growing I.

During point insertion additionally the `up` and `down` pointers must be set. For layers represented in $\mathbb{T}^3_{\boldsymbol{c}}$, these two operations are in $O(1)$. For layers represented in $\mathbb{T}^3_{3\boldsymbol{c}}$, the `down` pointers of the periodic copies have to be set to the respective original vertex in the layer below. This requires a query per pointer in the `covering_map`, which is in $O(\log N)$. The probability of inserting a point into layer i is α^i, so the expected number of insertions per point is $\sum_{i=0}^{\infty} \alpha^i \in O(1)$.

As seen above, a point insertion into the periodic Delaunay triangulation is in $O(n + N^2) \subseteq O(nN)$. Inserting into the hierarchy does not change this so the randomized worst-case complexity is $O(n^2 N)$.

Summary See Table 3.1 for an overview of the results of this section.

If N is constant, the asymptotic running time is the same as for computing Delaunay triangulations in $\mathbb{E}^3$, namely in the worst case $O(n^3)$, which can be improved to randomized worst-case $O(n^2)$ by using the hierarchy. In case the point set does not have a triangulation, then $N = n$ and a $\log n$ factor is added, which yields a worst-case complexity in $O(n^3 \log n)$, randomized worst-case $O(n^2 \log n)$ when using the hierarchy.

[1] The layer with the smallest index and largest number of points.

	$\mathbb{E}^3$	$\mathbb{T}^3$
`long_edges_list` maintenance	—	$O(N^3)$
$\mathbb{T}^3_{3\boldsymbol{c}} \to \mathbb{T}^3_{\boldsymbol{c}}$	—	$O(M \log N)$
$\mathbb{T}^3_{\boldsymbol{c}} \to \mathbb{T}^3_{3\boldsymbol{c}}$	—	$O(MN)$
Point location	$O(m)$	$O(m + M \log N)$
Point insertion	$O(n)$	$O(n + N^2)$
Vertex removal	$O(n^2)$	$O(n^2)$
Point location with hierarchy	$O(n)$	$O(n \log N)$
Triangulation	$O(n^3)$	$O(n^3 N)$
T. with hierarchy	$O(n^2)$	$O(n^2 N)$

Table 3.1: Overview of the results of Section 3.4

3.5 Study of an alternative design

In this Section we describe an idea for an alternative design that integrates the CGAL 3D triangulations and the CGAL 3D periodic triangulations. This alternative design has been described in [CKT08]. The motivation for such an approach is that the algorithms for computing triangulations in $\mathbb{E}^3$ and $\mathbb{T}^3_{\boldsymbol{c}}$ are not substantially different. The goal is to duplicate as little functions as possible in the code. However, it turns out that most of the highly optimized functions loses efficiency by generalizing them sufficiently to work for both spaces.

The alternative design is developed taking the following particularities of $\mathbb{E}^3$ and $\mathbb{T}^3$ into account:

$\mathbb{E}^3$: vertex at infinity, degenerate dimensions,

$\mathbb{T}^3$: covering spaces, offsets.

The goal of the alternative design is to make it possible to easily extend the current implementation to different spaces with the least possible redundancy in code. The basic idea of the alternative design is to split the class `Triangulation_3` into three classes related by inheritance (cf. Figure 3.20). The embedding space class provides all functionality that depends on the space. It is now a template parameter of the triangulation, together with the triangulation data structure and the geometric traits.

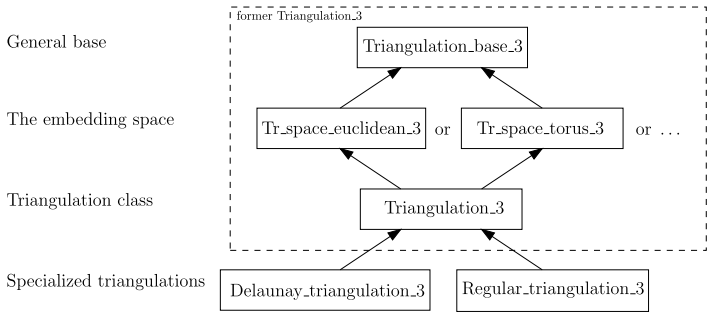


Figure 3.20: The alternative design.

The triangulation data structure and the geometric traits are the same as described in Section 3.3. Let us now list the affected classes in more detail:

General base: This class contains all the functionality that does neither depend on the space, nor on the triangulation type. These are mostly wrapper functions from the triangulation data structure and other general helper functions.

The embedding space: This class is new in the design and is used to handle everything that depends on the embedding space of the triangulation.
There is functionality that depends on both space and triangulation type. In these cases we use visitors [GHJV95] that modify the functionality relevant to the triangulation type in the space class.

Triangulation class: The triangulation class provides the same interface as in the design described in Section 3.3.1, independent of the embedding space. It provides generic algorithms for point location, point insertions and flips. This class is finally specialized to Delaunay triangulation and regular triangulation.

Adding further spaces is quite simple with this design. Only a new space class (and possibly a different geometric traits class) is needed, since the algorithms provided by the triangulation classes are fully generic.

The ideal result would be that all main functions go in the base class and the respective space and triangulation classes contain only visitors that implement space specific and triangulation specific functionality. However, in this way we could only merge about 274 out of 3396 lines of code without losing any efficiency. As the CGAL 3D triangulations are highly optimized and widely used, it is not acceptable to implement modifications that trade genericity against efficiency. This means that almost all the functionality ends up in the space class, which implies that almost all of the code will be duplicated. Given these results we preferred having a completely separate implementation for the 3D periodic triangulations.

3.6 Experiments

In Section 1.5 we introduced three implementations of algorithms to compute the 3D periodic Delaunay triangulation. Unfortunately, most of them are either not publicly available or not maintained anymore. We compare our implementation of the algorithm to compute the 3D periodic Delaunay triangulation to the CGAL implementation for the 3D Delaunay triangulation of $\mathbb{E}^3$. We present some experiments comparing the computation of the Delaunay triangulations of $\mathbb{E}^3$ and $\mathbb{T}^3$ defined by the same set of points in $\mathcal{D}^3_{\boldsymbol{c}}$.

For uniformly distributed input points the computation of the periodic Delaunay triangulation is expected to be slightly slower than the computation of the Delaunay triangulation of $\mathbb{E}^3$ due to overhead caused by managing the offsets and the computation in covering spaces.

At first we show experiments on the construction of the Delaunay triangulation using different configurations of the above described optimizations: with or without use of the spatial sorting and the dummy point set. Another test series restricts on the point insertion in $\mathbb{T}^3_{\boldsymbol{c}}$, which is the most critical part in the Delaunay triangulation construction: We insert the points into a precomputed triangulation in $\mathbb{T}^3_{\boldsymbol{c}}$. In this way we avoid computation in $\mathbb{T}^3_{3\boldsymbol{c}}$ but unlike the use of the dummy points these points are chosen at random and we do not remove them again. Then we briefly present results of the same experiments using the

triangulation hierarchy as point-location data-structure. The following experiment tests the vertex removal functionality. Then we run the Delaunay triangulation construction tests and the point insertion tests using a geometric traits class that is specialized to the case that $\mathcal{D}_{\boldsymbol{c}}$ is the unit cube. And finally we compare the two Criteria of Section 2.3.1. As each of the optimizations we use has different properties, these experiments help to identify specific advantages and shortcomings in our implementation.

3.6.1 Input point sets

We ran experiments on both generated data and real data from astronomy (by courtesy of Prof. Rien van de Weijgaert[2]). In all benchmarks the original domain is the half-open unit cube $[0,1)^3$.

We denote the data sets as follows:

`cube` points uniformly distributed in the half-open unit cube

`sph` points uniformly distributed on a sphere of radius 0.5 centered at $(0.5, 0.5, 0.5)$

`astr` input data from research on astronomy

We now discuss the different experiments in detail. The rows of the tables give the running times for input point sets the number of input points given in the first column. The running time behavior of Delaunay triangulation computation of $\mathbb{E}^3$ and $\mathbb{T}^3_{\boldsymbol{c}}$ is shown in the second and third column, respectively. Furthermore, in the last column the factor by which the computation of the Delaunay triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$ is slower is given. All running times are given in seconds. Experiments were run on an Intel Xeon 8 core at 2.33 Ghz running Fedora 10 and CGAL 3.6. We aborted tests that took longer than 300 seconds; aborted tests are marked as `--` in the tables.

3.6.2 Construction of the Delaunay triangulation

We ran three series of experiments: One with the default configuration, one using the spatial sorting and one using both spatial sorting and the dummy point set.

For each experiment we list the table for the point sets `cube`, `astr`, and `sph` as well as a plot of this data. Note that the plot axes are both scaled logarithmically.

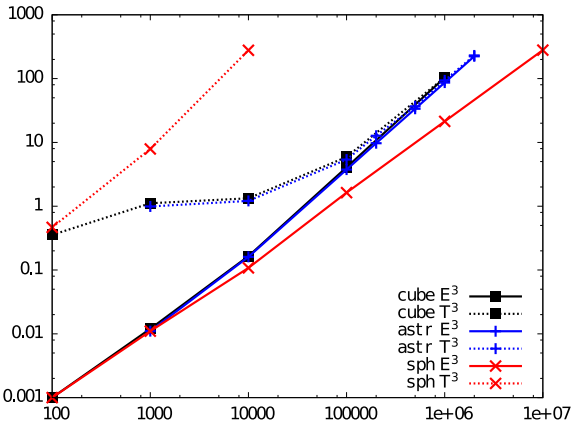
Default configuration

`cube`	$\mathbb{E}^3$	$\mathbb{T}^3_{\boldsymbol{c}}$	factor
10^2	0.001	0.355	354.9
10^3	0.012	1.114	92.85
10^4	0.165	1.332	8.07
10^5	4.068	5.904	1.45
10^6	103.5	104.2	1.01

`astr`	$\mathbb{E}^3$	$\mathbb{T}^3_{\boldsymbol{c}}$	factor
$1 \cdot 10^3$	0.011	0.990	90.00
$1 \cdot 10^4$	0.161	1.212	7.53
$1 \cdot 10^5$	3.753	5.380	1.43
$2 \cdot 10^5$	9.706	12.72	1.31
$5 \cdot 10^5$	33.70	36.73	1.09
$1 \cdot 10^6$	86.70	93.71	1.08
$2 \cdot 10^6$	223.9	230.8	1.03

[2] Kapteyn Institute, Groningen, Netherlands

sph	$\mathbb{E}^3$	$\mathbb{T}^3_c$	factor
10^2	0.001	0.466	465.9
10^3	0.011	7.896	717.9
10^4	0.108	277.9	2574
10^5	1.631	--	--
10^6	21.39	--	--
10^7	279.1	--	--



In this experiment, we see that for the point set `cube` of 10^2 points the slow-down due to the overhead of computing in $\mathbb{T}^3_{3\boldsymbol{c}}$ is quite large. For larger point sets, the slow-down becomes smaller because the initial computation in $\mathbb{T}^3_{3\boldsymbol{c}}$ has less impact on the overall running time. Finally, for point sets of about 10^6 points the computation of periodic triangulations takes about the same time as the triangulation of point sets in $\mathbb{E}^3$. The slow-down becomes negligible.

The experiments `astr` show that the real-world point set is sufficiently well distributed to have a behavior similar to `cube`.

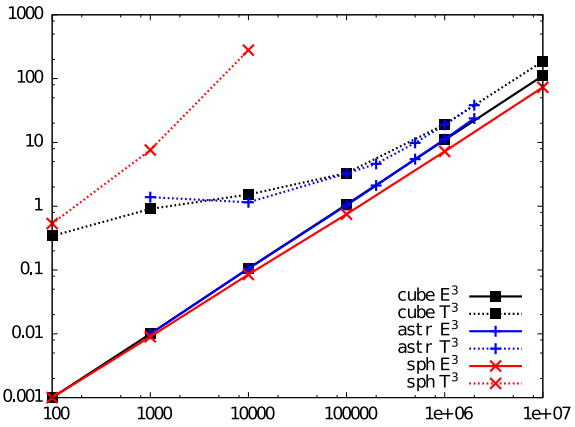
For the `sph` experiment, the large slow-down persists and even grows. Here, the periodic Delaunay triangulation is never represented in $\mathbb{T}^3_{3\boldsymbol{c}}$ because for this point set there are always edges longer than the threshold of Criterion 2.3.4. As the projection of the 1-skeleton of the periodic Delaunay triangulation of the sphere onto $\mathbb{T}^3_{\boldsymbol{c}}$ has in most cases cycles of length 2 we cannot even hope to do much better. Moreover, we are not aware of any case in real-world applications where such a situation would occur. The growing slow-down for larger point sets is due to handling the triangulation of the covering space, where several `std::maps` and additional function calls are involved. This is the price we have to pay in order to avoid as much overhead as possible when computing in $\mathbb{T}^3_{\boldsymbol{c}}$, which is clearly the principal goal.

With spatial sorting

cube	$\mathbb{E}^3$	$\mathbb{T}^3_c$	factor
10^2	0.001	0.342	341.9
10^3	0.010	0.906	90.60
10^4	0.106	1.521	14.35
10^5	1.064	3.309	3.11
10^6	10.95	19.22	1.76
10^7	111.1	186.7	1.68

astr	$\mathbb{E}^3$	$\mathbb{T}^3_c$	factor
$1 \cdot 10^3$	0.010	1.388	138.8
$1 \cdot 10^4$	0.105	1.151	10.96
$1 \cdot 10^5$	1.037	3.255	3.14
$2 \cdot 10^5$	2.112	4.620	2.19
$5 \cdot 10^5$	5.466	9.838	1.80
$1 \cdot 10^6$	11.21	18.89	1.69
$2 \cdot 10^6$	23.77	38.24	1.61

sph	$\mathbb{E}^3$	$\mathbb{T}^3_c$	factor
10^2	0.001	0.535	534.9
10^3	0.009	7.610	845.6
10^4	0.085	279.7	3291
10^5	0.750	--	--
10^6	7.239	--	--
10^7	73.17	--	--



When using the spatial sorting, the behavior of the running times becomes almost linear in the number of points. The principal gain comes from the fact that the point-location step gets accelerated considerably. The point location function has very similar running times whether computing in $\mathbb{E}^3$ or $\mathbb{T}^3_c$. So in the tests of the default configuration, the point location has an attenuating effect on the overall running time difference. In the case of using spatial sorting, computing a periodic triangulation of a large number of well-distributed points is about 1.6 times slower than computing a triangulation of $\mathbb{E}^3$.

The results of the `astr` experiment again agree with the `cube` experiment.

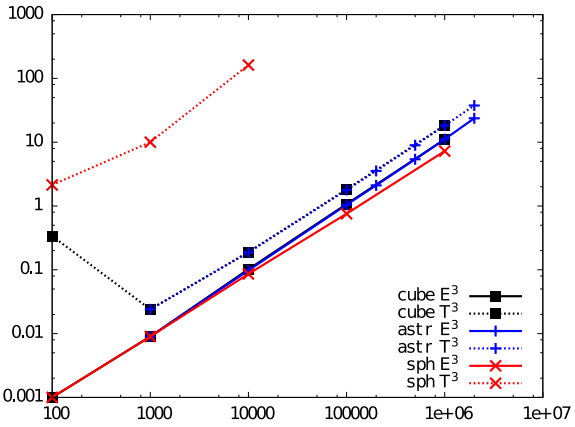
The spatial sorting does not work very efficiently on the set of points on a sphere. In the periodic case its effect is outweighed by the expensive handling of the covering space.

With dummy point set and with spatial sorting

cube	$\mathbb{E}^3$	$\mathbb{T}^3_c$	factor
10^2	0.001	0.339	339.0
10^3	0.009	0.024	2.67
10^4	0.102	0.190	1.86
10^5	1.07	1.82	1.69
10^6	11.0	18.4	1.67

astr	$\mathbb{E}^3$	$\mathbb{T}^3_c$	factor
$1 \cdot 10^3$	0.009	0.024	2.67
$1 \cdot 10^4$	0.098	0.187	1.91
$1 \cdot 10^5$	1.045	1.780	1.70
$2 \cdot 10^5$	2.105	3.545	1.68
$5 \cdot 10^5$	5.402	8.954	1.66
$1 \cdot 10^6$	11.16	18.25	1.64
$2 \cdot 10^6$	23.67	37.62	1.59

sph	$\mathbb{E}^3$	$\mathbb{T}^3_c$	factor
10^2	0.001	2.140	2140
10^3	0.009	10.02	1113
10^4	0.086	161.7	1880
10^5	0.756	--	--
10^6	7.262	--	--



For the `cube` experiment, the big slow-down for the set of 10^2 points is explained by the fact that this point set does not define a Delaunay triangulation of $\mathbb{T}^3_c$ and thus the algorithm must switch to computing in $\mathbb{T}^3_{3c}$ while removing the dummy points. For larger point sets no computation in $\mathbb{T}^3_{3c}$ is required at all, and so the slow-down is much less: The factor is between 1.67 and 2.67. Note that the larger the point set, the less is the slow-down, which can be explained in the following way. In our implementation we provide specialized functions for dealing more efficiently with cells that have all zero offsets. A part of the slow-down in $\mathbb{T}^3_c$ is due to offset manipulations during the point insertion, when cells with non-zero offsets are involved. As the point set is uniformly distributed, the percentage of cells with non-zero offsets, i.e. the number of cells that are located on the boundary of the original domain, is smaller for large point sets than for small point sets, so the relative slowdown decreases when the number of points grows.

For the `sph` experiment with 10^4 points, the slow-down is less than before. That is because this approach avoids inserting points in $\mathbb{T}^3_{3c}$. Only the post-processing of removing the 36 dummy points and converting the triangulation to $\mathbb{T}^3_{3c}$ is expensive. This result suggests that for point sets of more than about 5000 points that do not define a triangulation in $\mathbb{T}^3_c$, it is useful to use the dummy point set.

3.6.3 Point insertion in $\mathbb{T}^3_c$

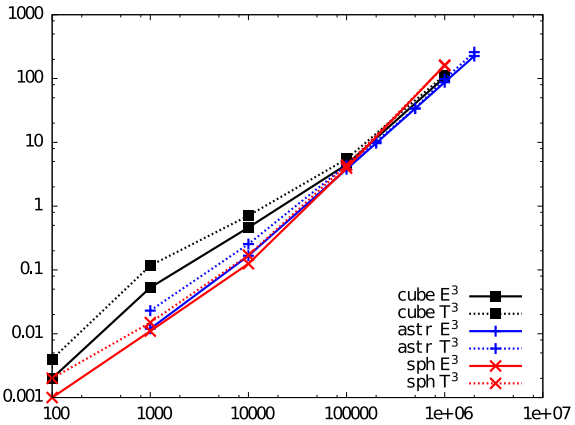
Here, we start with a precomputed triangulation of $\mathbb{T}^3_c$ from 1000 uniformly distributed input points and insert more points. We do this with and without spatial sorting of the input points. The goal of these experiments is to compare the insertion of points in $\mathbb{T}^3_c$ only, i.e. to filter out potential overhead for managing 27 copies or the dummy point set.

Without spatial sorting

`cube`	$\mathbb{E}^3$	$\mathbb{T}^3_c$	factor
10^2	0.002	0.004	2.00
10^3	0.053	0.117	2.21
10^4	0.462	0.708	1.53
10^5	4.479	5.488	1.23
10^6	104.8	113.4	1.08

`astr`	$\mathbb{E}^3$	$\mathbb{T}^3_c$	factor
$1 \cdot 10^3$	0.012	0.023	1.92
$1 \cdot 10^4$	0.167	0.254	1.52
$1 \cdot 10^5$	3.799	4.388	1.15
$2 \cdot 10^5$	9.716	10.32	1.06
$5 \cdot 10^5$	33.76	34.08	1.01
$1 \cdot 10^6$	87.45	98.02	1.12
$2 \cdot 10^6$	225.0	258.8	1.15

`sph`	$\mathbb{E}^3$	$\mathbb{T}^3_c$	factor
10^2	0.001	0.002	2.00
10^3	0.011	0.015	1.36
10^4	0.125	0.170	1.36
10^5	3.940	4.347	1.10
10^6	162.5	158.8	0.98



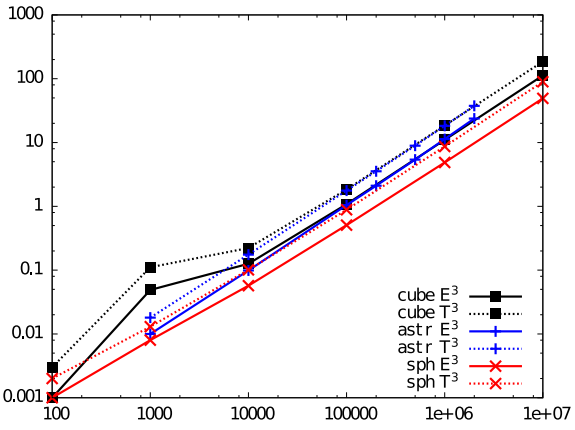
Here, we see even more clearly in all three experiments that the bigger the number of points the less is the slow-down. Note that also the case of points on a sphere can now be computed easily. That is because there are additional points ensuring that all computations can be done in $\mathbb{T}^3_{\boldsymbol{c}}$ rather than $\mathbb{T}^3_{3\boldsymbol{c}}$. These additional points are contained in the final result though.

With spatial sorting

cube	$\mathbb{E}^3$	$\mathbb{T}^3_{\boldsymbol{c}}$	factor
10^2	0.001	0.003	3.00
10^3	0.049	0.111	2.27
10^4	0.126	0.222	1.76
10^5	1.080	1.840	1.70
10^6	11.02	18.51	1.68
10^7	112.5	185.6	1.65

astr	$\mathbb{E}^3$	$\mathbb{T}^3_{\boldsymbol{c}}$	factor
$1 \cdot 10^3$	0.010	0.018	1.80
$1 \cdot 10^4$	0.099	0.175	1.78
$1 \cdot 10^5$	1.045	1.765	1.69
$2 \cdot 10^5$	2.102	3.548	1.69
$5 \cdot 10^5$	5.405	8.920	1.65
$1 \cdot 10^6$	11.32	18.24	1.61
$2 \cdot 10^6$	23.75	37.64	1.59

sph	$\mathbb{E}^3$	$\mathbb{T}^3_{\boldsymbol{c}}$	factor
10^2	0.001	0.002	2.00
10^3	0.008	0.013	1.63
10^4	0.057	0.101	1.77
10^5	0.508	0.884	1.74
10^6	4.872	8.677	1.78
10^7	49.21	89.61	1.82



This experiment confirms the above explanations without adding new results.

3.6.4 The triangulation hierarchy

We ran the same experiments as above using the triangulation hierarchy. In the periodic case, the hierarchy suffers from the fact that there are always layers representing a triangulation of $\mathbb{T}^3_{3\boldsymbol{c}}$. We only give a sample of the results for the point set cube of 10^6 points.

	$\mathbb{E}^3$	$\mathbb{T}^3_{\boldsymbol{c}}$	factor
default	30.57	79.66	2.61
spatial sorting	11.76	33.49	2.85
dummy point set + spatial sorting	11.70	28.96	2.48
point insertion in $\mathbb{T}^3_{\boldsymbol{c}}$	31.21	78.71	2.52
point insertion in $\mathbb{T}^3_{\boldsymbol{c}}$ + spatial sorting	11.78	28.16	2.39

In practice only the triangulation of the uppermost level is represented in $\mathbb{T}^3_{3\boldsymbol{c}}$, so the slow-down is with a factor between 2.39 and 2.85 comparatively small. The fact that the slow-down is similar for all five experiments shows that in $\mathbb{T}^3_{\boldsymbol{c}}$ the triangulation hierarchy shows a similar behavior as in $\mathbb{E}^3$.

3.6.5 Vertex removal

Here we remove 10,000 vertices from a triangulation of the given size.

cube	$\mathbb{E}^3$	$\mathbb{T}^3_{\boldsymbol{c}}$	factor
10^4	1.775	105.6	59.52
10^5	1.426	1.843	1.292
10^6	1.675	2.102	1.255

astr	$\mathbb{E}^3$	$\mathbb{T}^3_{\boldsymbol{c}}$	factor
$1 \cdot 10^4$	1.812	107.7	59.43
$1 \cdot 10^5$	1.394	1.827	1.31
$2 \cdot 10^5$	1.426	1.840	1.29
$5 \cdot 10^5$	1.508	1.923	1.28
$1 \cdot 10^6$	1.636	2.058	1.26
$2 \cdot 10^6$	1.855	2.264	1.22

We see that the running time of removing 10,000 vertices does not much depend on the size of the triangulation. The overhead for the periodic triangulation is comparatively small: It causes a slow-down of a factor between 1.2 and 1.3. Only if the triangulation must be converted to $\mathbb{T}^3_{3\boldsymbol{c}}$ during the removal, the slow-down factor goes up to about 60.

3.6.6 Specific original domain

In the CGAL 3D periodic triangulations, the original domain of the periodic space can be chosen by the user. When computing predicates of point-offset pairs, points must be translated by multiples of the edge length of the original domain. This is numerically more involved than simply fixing the original domain to the unit cube.

We could write a special traits class that permits only the unit cube as original domain. The experiments below show that in this case we gain about 3% on the running time for both 10^6 uniformly distributed points in the cube and the cosmological data set of 10^6 points.

cube	$\mathbb{T}^3_{\boldsymbol{c}}$	$\mathbb{T}^3_{\boldsymbol{c}}$, $\boldsymbol{c} = [0,1)^3$	Speed-up in %
standard	104.2	102.7	1.5%
spatial	19.22	18.45	4.0%
dummy, spatial	18.35	17.80	3.0%
ins $\mathbb{T}^3_{\boldsymbol{c}}$	113.4	111.8	1.4%
ins $\mathbb{T}^3_{\boldsymbol{c}}$, spatial	18.51	17.78	4.0%

astr	$\mathbb{T}^3_{\boldsymbol{c}}$	$\mathbb{T}^3_{\boldsymbol{c}}$, $\boldsymbol{c} = [0,1]^3$	Speed-up in %
standard	93.71	92.78	1.0%
spatial	18.89	18.44	2.4%
dummy, spatial	18.25	17.60	3.5%
ins $\mathbb{T}^3_{\boldsymbol{c}}$	98.02	96.70	1.3%
ins $\mathbb{T}^3_{\boldsymbol{c}}$, spatial	18.24	17.67	3.1%

We see that the speed-up is bigger in the cases when spatial sorting is used. This can be explained by the fact that our specialization to $\boldsymbol{c} = [0,1)^3$ accelerates the predicate evaluation, and the part of the running time used up in predicate evaluation is bigger when using spatial sorting. We decided that the resulting speed-up is not sufficient to include this feature in the public release.

3.6.7 Comparison of the criteria of Section 2.3.1

In the experiments above we have seen that whether computing in $\mathbb{T}^3_{\boldsymbol{c}}$ or $\mathbb{T}^3_{3\boldsymbol{c}}$ has a huge impact on the running times. Here, we test for given point sets after insertion of how many points the triangulation fulfills some of the conditions presented in Chapter 2. We use the following notation for the different conditions:

2c (2-cycle): This is the number of points for which the point set defines a triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$ for the first time. The necessary and sufficient condition of Theorem 2.2.8 is used to compute this property. Note that by adding further points the point set can lose this property again.

bd (ball diameter): This is the number of points for which the point set fulfills the condition of Criterion 2.3.3 for the first time. That is in a triangulation of this point set the diameter of the largest circumscribing ball is smaller than half the edge length of the original domain.

el (edge length): This is the number of points for which the point set fulfills the condition of Criterion 2.3.4, the edge-length criterion.

We ran the tests on the point sets `cube` and `astr`. For the point est `sph`, for instance, these criteria are never fulfilled. We ran each experiment twice. Once on a random insertion order and once using spatial sorting as described in Section 3.3.9.

The results without the use of spatial sorting are shown in the two following tables.

cube	2c	bd	el
10^2	57		
10^3	51	136	250
10^4	50	139	247
10^5	56	153	226
10^6	61	145	230
10^7	55	152	246

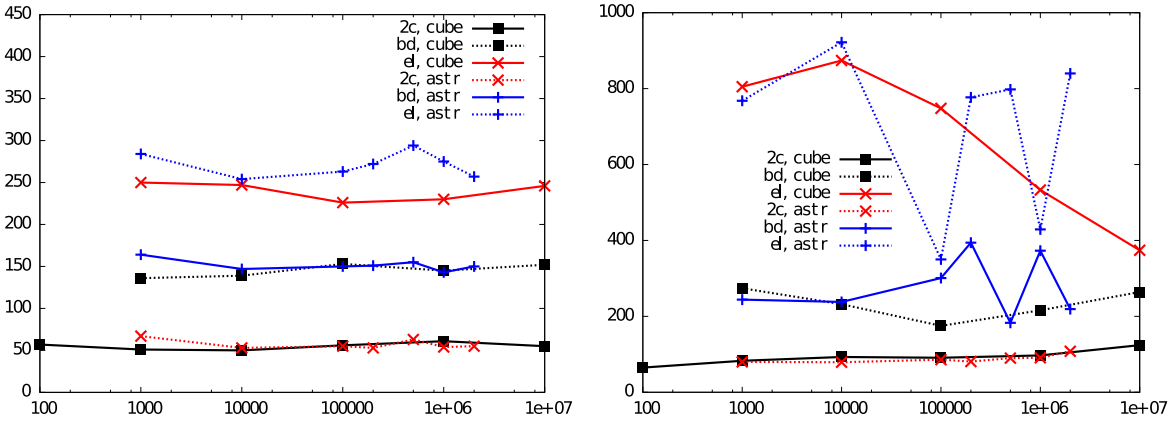
astr	2c	bd	el
$1 \cdot 10^3$	67	164	284
$1 \cdot 10^4$	53	147	254
$1 \cdot 10^5$	55	150	263
$2 \cdot 10^5$	53	151	272
$5 \cdot 10^5$	63	155	294
$1 \cdot 10^6$	54	143	275
$2 \cdot 10^6$	55	150	257

In the two following tables the result with the use of spatial sorting are given.

cube	2c	bd	el
10^2	65		
10^3	83	274	805
10^4	93	232	874
10^5	91	175	748
10^6	97	216	533
10^7	124	264	374

astr	2c	bd	el
$1 \cdot 10^3$	80	244	768
$1 \cdot 10^4$	79	238	922
$1 \cdot 10^5$	86	301	350
$2 \cdot 10^5$	81	394	777
$5 \cdot 10^5$	90	183	798
$1 \cdot 10^6$	91	373	429
$2 \cdot 10^6$	108	219	840

The left plot shows the experiments without spatial sorting, the right plot shows the experiments with spatial sorting.



All tests were run 10 times and the presented numbers are the arithmetic means of all 10 results. This is done because the results of one run can vary due to the randomization in the algorithm.

As Theorem 2.2.8 gives a necessary and sufficient condition on whether the point set defines a Delaunay triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$, the numbers in column 2c can be considered as the best possible result. Note that the condition of Theorem 2.2.8 is not sufficient for our actual purpose, which is to decide whether any superset of the point set has a triangulation in $\mathbb{T}^3_{\boldsymbol{c}}$. We can consider the results of column 2c as lower bounds on the minimum number of points required to switch to $\mathbb{T}^3_{\boldsymbol{c}}$ and use it to measure the quality of our geometric criteria.

The use of spatial sorting increases the required number of points. That is because now the points are inserted in a specific order, thus it takes longer until the point set is sufficiently well-distributed. The unstable results for the point set `astr` show that the number of required points is very sensitive to the input point set.

The tables show that generally point sets of about 60 random points already define a Delaunay triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$. However, it can happen that inserting a point in such a triangulation leads to a point set that does not define a Delaunay triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$ anymore. In general about 150 points are necessary for the largest empty ball diameter to be smaller than half an edge length of the original domain. According to Criterion 2.3.3 such point sets define a Delaunay triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$ even if we add further points. However, computing the circumcenter of a tetrahedron is more expensive than computing the distance of two points. That is why Algorithm 2.3.1 tests for the longest edge length to be smaller than $\frac{1}{\sqrt{6}}\,c$ (see Criterion 2.3.4). As this is a stronger criterion, about 250 to 300 points are required for the algorithm to switch to computing in the 1-sheeted covering space. When the data set becomes large, this number can be considered negligible.

3.7 Applications

Due to the modularity of CGAL it is easily possible to plug different functionalities together. For instance, there are several algorithms that use 3D triangulations, like the alpha shape computation [DY10] or the meshing algorithms [RY10, RTY10].

However, the interface of the implementation of 3D periodic triangulations has some mathematically motivated differences to the interface of the non-periodic triangulations. Thus we need to introduce adapters to make the 3D periodic triangulations compatible to other CGAL algorithms.

3.7.1 Periodic alpha shapes

Alpha shapes have been described in [EKS83, EM94]. This extension has been motivated by the need of astronomers for computing periodic alpha shapes of simulations on the cosmic web [vdWVP+10]. Based on the cosmological principle (Section 1.1), these simulations are run on a comparatively small sample of the cosmic web. However, the boundaries of the sample can have a substantial effect on the topology of its alpha shapes. By computing periodic alpha shapes these effects can be avoided.

Definition 3.7.1 ([EM94]). *Given a point set $\mathcal{S}$ and a parameter α with $0 < \alpha < \infty$, the* alpha shape *of $\mathcal{S}$ is the subset of all simplices of $DT(\mathcal{S})$ that have a circumscribing ball of radius α or smaller that does not have any point of $\mathcal{S}$ in its interior.*

If we replace the Delaunay triangulation by *weighted* Delaunay triangulation, then we get *weighted* alpha shapes. The definition directly extends to periodic triangulations.

The CGAL implementation of alpha shapes takes an implementation of a triangulation algorithm as template parameter. During the construction of the alpha shape, first the 3D (weighted) Delaunay triangulation is computed and then the alpha shape on top of it: It attaches to each simplex the smallest α for which it appears in the alpha shape and provides iterators to output alpha shapes for a given α.

The very same approach works for periodic triangulations. The CGAL 3D periodic triangulation implementation outputs a triangulation on top of which the alpha shape can be computed. In order to do so, the class `Alpha_shape_3` must be instantiated with the class `Periodic_3_Delaunay_triangulation_3` as template parameter. Still, some difficulties arise in the implementation that are solved as follows. Note that periodic weighted alpha shapes are not yet available because there is no implementation of weighted periodic Delaunay triangulations yet.

In order to establish compatibility between the CGAL alpha shape implementation and the CGAL 3D periodic triangulations, we have to redefine some types and functions that exist in the CGAL 3D triangulations but not in the CGAL 3D periodic triangulations. These are mainly functions dealing with the vertex at infinity and degenerate dimensions and turn out to be trivial for the periodic case:

- Type `Finite_[simplex]_iterator`. In the periodic triangulation all simplices are finite, thus this type can be defined to be the same as the `[Simplex]_iterator`.

- Function `dimension`. In the periodic triangulation we are always computing in three dimensions. Thus this function returns always 3.

- Function `is_infinite`: Decide for a given simplex whether it has the vertex at infinity as a vertex. As there is no vertex at infinity in periodic triangulation, this function always returns `false`.
- Function `number_of_finite_[simplices]`: There are no infinite simplices in the periodic triangulation, so this function just returns the output of `number_of_[simplices]`.

Furthermore, as described in Section 3.3.8, we cannot directly access point coordinates of vertices through the triangulation data structure. So in the alpha shape implementation every point coordinate access of the form `cell->vertex(i)->point()` has to be replaced by an expression of the form `this->point(cell,i)`, cf. Section 3.3.8.

Once the periodic alpha shapes are available it is interesting to compute their Betti numbers. This can be accomplished using the algorithm of [DE95] and the work of [Abs09]. Figure 3.21 shows a screenshot of a prototype implementation.

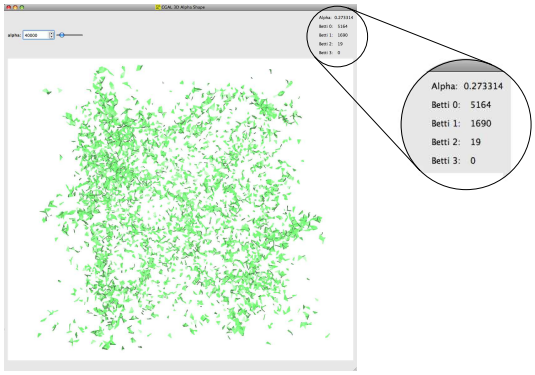


Figure 3.21: Screenshot of a demo prototype for computing periodic alpha shapes and their Betti numbers.

3.7.2 Periodic surface mesher

The results of this section are joint work with Vissarion Fisikopoulos [CFT10].

The CGAL surface mesher provides generation of triangular meshes to approximate smooth surfaces. It implements the meshing algorithm described in [BO05]. For a surface mesh computed by CGAL see Figure 3.22

The algorithm works as follows: It maintains a Delaunay triangulation and a queue of so-called bad surface facets. A facet is surface facet if its dual Voronoi edge intersects the surface. There is a set of so-called refinement criteria that decide whether a facet is good or bad. The algorithm then starts with a Delaunay triangulation of $\mathbb{E}^3$ defined by a set of initial points on the surface. It maintains a queue of bad surface facets and in each step a new point on the surface is added to refine the current bad facet. A *surface Delaunay ball* of a surface facet is a circumscribing ball of this facet that is centered at the intersection of the surface facet's dual Voronoi edge with the surface, see Figure 3.23. One by one the bad surface facets from the queue are refined, that is, the center of their surface Delaunay ball is inserted into the Delaunay triangulation and the new bad surface facets are added to the queue.

The implementation of this algorithm in CGAL is described in [RY10]. It provides the following three refinement criteria as a default.

Figure 3.22: A surface mesh computed by CGAL, model by Visual Computing Lab, Pisa, Italy.

- *Aspect criterion:* A surface facet is good if its minimum angle is larger than some user-defined threshold.
- *Uniform size criterion:* A surface facet is good if the radius of its surface Delaunay ball is smaller than some user-defined threshold.
- *Curvature criterion:* A surface facet is good if the distance of its circumcenter to the center of the Surface Delaunay ball is smaller than some user-defined threshold.

Default values for the thresholds are provided.

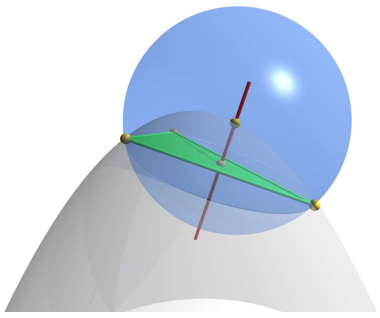


Figure 3.23: The surface Delaunay ball of a surface facet.

The default criteria can be replaced by the user's own criteria, which can be given to it as a template. In the same way the underlying Delaunay triangulation implementation can be exchanged.

In order to compute periodic surface meshes, we plug the periodic Delaunay triangulations in the CGAL surface mesher. When doing so, similar interface problems occur as for the periodic alpha shapes, see Section 3.7.1. Most of them can be resolved in the same way. A problem appears because instead of using the point insertion method of the periodic Delaunay triangulation, the surface mesher first computes the cells in conflict with this point using the method `find_conflicts`. Then it removes these cells and adds new cells using the method `insert_in_hole`. As described in Section 3.3.6, when computing periodic triangulations, the offsets of the vertices lying on the boundary of the hole

with respect to the new point must be stored. This information is directly stored in the vertices and must be cleaned up after each run of `find_conflicts` in order to not cause side effects later on. This clean-up is done by `insert_in_hole`. Thus, when computing a periodic Delaunay triangulation, for each call to `find_conflicts` there must be a corresponding call to `insert_in_hole` and there must not be any calls to `find_conflicts` in between. The surface mesher, however, does not always call `insert_in_hole` after a call of `find_conflicts`. So we use a modified version of `find_conflicts` that cleans up the vertex offset *before* starting to detect cells in conflict.

Only the evaluation of the refinement criteria require geometric computations. So the refinement criteria must be adapted in order to treat the offsets correctly. They are made available to the algorithm through template arguments, so they can be replaced easily. We write a new set of refinement criterion classes for the periodic case. Note that in order to correctly compute translated points from point-offset pairs the refinement criterion classes must have access to the edge length of the original domain of the periodic triangulation. Thus a pointer to the periodic triangulation must be provided to the constructor of the criterion classes. The following modifications must be implemented for the specific criteria.

Aspect criterion
Use the offsets to correctly embed the given facet into $\mathbb{R}^3$ in order to compute its minimum angle, see Figure 3.24.

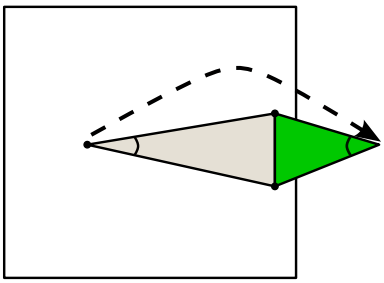


Figure 3.24: (2D illustration) The periodic aspect ratio criterion.

Uniform size criterion
This criterion is more complicated because we have to compute the center of the surface Delaunay ball of a given facet. It is not possible to infer the offset of the center of the surface Delaunay ball from the offsets stored in the triangulation. This can be seen by a similar example as in Figure 3.19 on page 58 for circumscribing balls of cells. We search the offset o that minimizes the distance between one vertex of the facet and the computed center with offset o. The correctness of this approach follows from the fact that we are computing a triangulation using Algorithm 2.3.1, i.e., we know that the biggest empty ball has diameter smaller than half the domain edge length. So one of the periodic copies of the center is closer to the facet vertices than half the domain edge length and all the others are further. In terms of predicates this approach requires several comparisons. See Figure 3.25 for an illustration of the periodic version of this criterion.

Curvature criterion
Here we have the same problem as for the uniform size criterion. We can apply the same

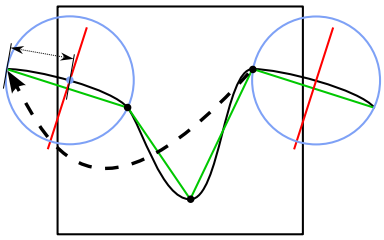


Figure 3.25: (2D illustration) The periodic uniform size criterion.

solution, as well (see Figure 3.26).

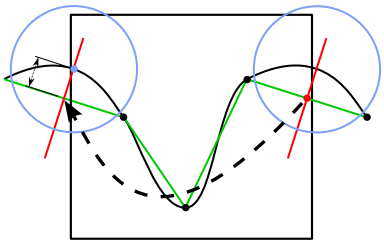


Figure 3.26: (2D illustration) The periodic curvature criterion.

In Figure 3.27 we show some periodic meshes computed with the CGAL surface mesher using the CGAL periodic Delaunay triangulations. The figures show eight copies of each mesh. We see that the copies perfectly agree at the boundaries, which is expected from the way of constructing the meshes.

Figure 3.27 shows some periodic surface meshes: The Schwarz p and the gyroid functions are triply periodic minimal surfaces [Sch70]. The cylinder is not triply periodic but only periodic in one direction. Nevertheless, it can be meshed. We obtained the piecewise defined function from M. Moesen at K. U. Leuven, Belgium, who works on bone scaffolding.

The implementation described in this section is not yet published in CGAL.

3.7.3 Periodic volume mesher

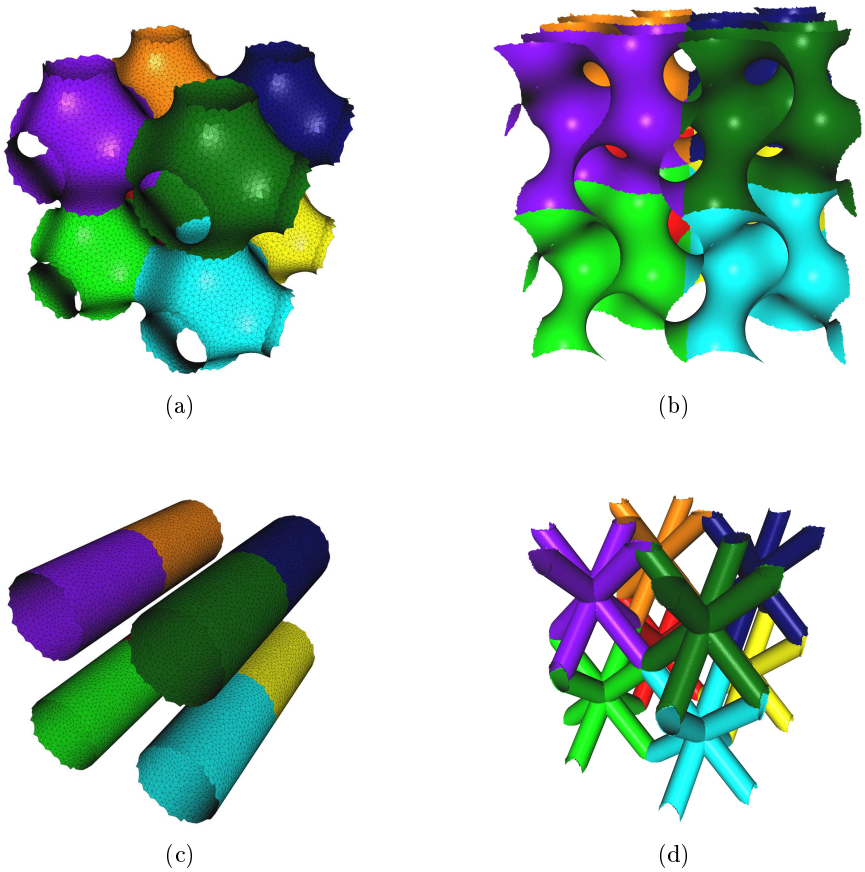
As a continuation of the work on periodic surface meshing described in Section 3.7.2, we also worked on computing periodic volume meshes. This is joint work with Mikhail Bogdanov.

The problems and solutions in this case are very similar to what was described in Section 3.7.2. We have to adapt the same three refinement criteria on the surface triangles and additionally two criteria on the cells in the volume:

- *Radius edge criterion:* upper bound on the ratio between the circumradius and the shortest edge of a tetrahedron
- *Radius criterion:* upper bound on the radius of the circumscribing ball

The computation of the two cell criteria for periodic meshing can be done in the same way as the computation of the facet criteria described above. See Figure 3.28 for examples of periodic volume meshes computed with the CGAL volume mesher and the periodic triangulations.

The implementation described in this section is not yet published in CGAL.



(a) (b)

(c) (d)

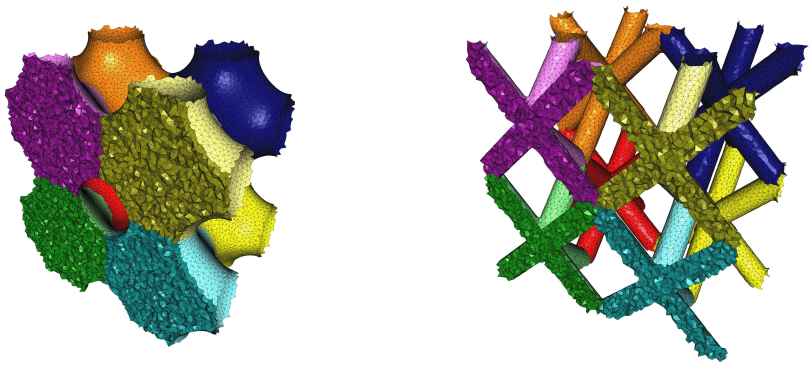
Figure 3.27: Some periodic surface meshes: (a) Schwarz p function, (b) Gyroid function, (c) cylinder, (d) piecewise defined function (courtesy M. Moesen).

3.7.4 Periodic Lloyd algorithm

Given a Voronoi diagram, a Lloyd iteration replaces all the sites by the centroids of their Voronoi cells [Llo82]. The Lloyd algorithm does repeated Lloyd iterations. The Voronoi diagram is known to converge towards a centroidal Voronoi diagram by repeatedly applying Lloyd iterations [DEJ06]. Lloyd iterations are, for instance, used in mesh optimization [Tou09].

Our implementation works as follows: In a first step the Voronoi diagram of all the input points is computed. In each iteration the centroid of each Voronoi cell is computed and each defining site of a cell is moved to the centroid of the same cell. In the implementation, we actually recompute the Voronoi diagram of the centroids from scratch. Using `dual` functions of the periodic Delaunay triangulation, it is straightforward to implement a periodic version of the Lloyd algorithm. The `dual` functions return the cells as convex set of points in $\mathbb{E}^3$, so the centroid computation in $\mathbb{E}^3$ can be reused. Only if a centroid lies outside of $\mathcal{D}_{\boldsymbol{c}}$, it must be translated back inside the original domain. See Figure 3.29 for an illustration of our periodic Lloyd software that is publicly available as a demo in CGAL.

Once we can compute Lloyd iterations on periodic Voronoi diagrams, we are interested in determining whether the number of points has an influence on the minimum and maxi-



(a) (b)

Figure 3.28: Periodic volume meshes: (a) Schwarz p function, (b) piecewise defined function (courtesy M. Moesen).

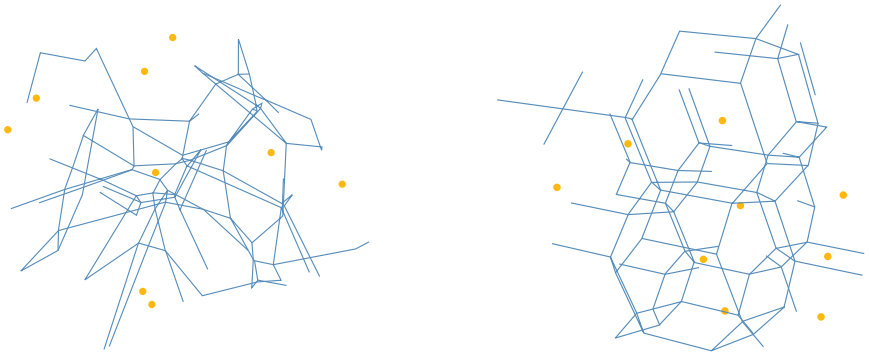


Figure 3.29: Left: Initial periodic Voronoi diagram of 9 random points. Right: Converged centroidal Voronoi diagram after about 25 Lloyd iterations.

mum dihedral angles of the dual periodic Delaunay triangulation. We expect an answer to this question to be useful in mesh optimization [Val]. We did the following experiment: We computed the periodic Delaunay triangulation of a set of n uniformly distributed random points in $\mathcal{D}_{\mathbf{c}}$. Then we ran Lloyd iterations until the minimum and maximum dihedral angle of two consecutive Lloyd iterations did not differ by more than 0.01 for 10 consecutive iterations. We ran 10 independent experiments for each n between 1 and 100, see Figure 3.30.

Unfortunately, the results of this experiment are not very conclusive. For small point sets very good extremal dihedral angles can be attained. However, for bigger point sets the results are much less clear. Especially since the results depend heavily on the starting configuration. For point sets of about 36 points and about 48 points we see that some of the starting configurations can yield very good results.

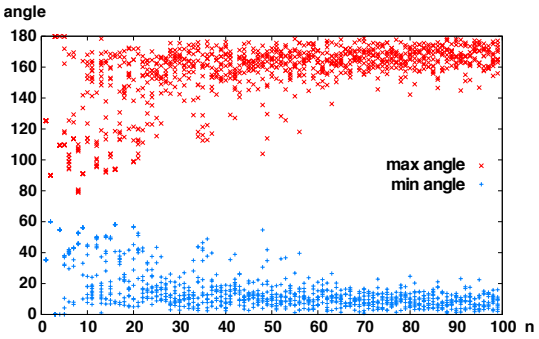


Figure 3.30: Minimum and maximum dihedral angle of a periodic Delaunay triangulation of n points after Lloyd iterations.

3.8 Conclusion

In this chapter, we presented our implementation of 3D periodic triangulations that is publicly available through the open source library CGAL. We described our implementation as well as the similarities and the differences to the CGAL 3D triangulations. We gave an overview on the complexity of the main functions and some interesting functions that are specific to $\mathbb{T}^3$. In experiments on both generated and real-world data we verify the efficiency of our implementation. In fact, for big random point sets our implementation turns out to be only about 1.7 times slower[3] than the computation of the Delaunay triangulation of $\mathbb{E}^3$. This is a great improvement compared to the factor of at least 27 when computing with 27 copies of each point. Finally, we showed some exemplary applications where the 3D periodic triangulations can be used instead of the 3D triangulations, requiring only minor adaptations.

[3] The factor can be improved to 1.6 using a traits class specialized to the unit cube.

Chapter 4

Delaunay triangulations of other spaces

In this chapter, we generalize the results of Chapter 2 to other spaces. More specifically, we consider three classes of orbit spaces:

1. flat spaces, i.e. spaces of constant Riemannian curvature *zero*. These are orbit spaces of $\mathbb{E}^d$,

2. spaces of constant *positive* curvature, i.e. orbit spaces of the d-dimensional sphere $\mathbb{S}^d$,

3. spaces of constant *negative* curvature, i.e. orbit spaces of the d-dimensional hyperbolic space $\mathbb{H}^d$.

The last class is much richer than the two above. We only give a preliminary discussion on the special case of the double torus.

4.1 Preliminaries

A number of definitions in Chapter 2 can be generalized to work on a rather broad class of orbit spaces, including spaces of different curvature. Here we give generalized versions of these definitions that are reused in the subsequent sections.

Let $\mathbb{M}$ be a d-manifold with the following properties:

1. There is a definition of the Delaunay triangulation of $\mathbb{M}$ defined by a set of points in $\mathbb{M}$.

2. An algorithm for computing the Delaunay triangulation of $\mathbb{M}$ from a given point set is known.

In this chapter we discuss the cases of $\mathbb{M} = \mathbb{E}^d$, $\mathbb{M} = \mathbb{S}^d$, and $\mathbb{M} = \mathbb{H}^d$.

Let $\mathcal{G}$ be a discrete group of isometries acting on $\mathbb{M}$ and $\mathbb{X} := \mathbb{M}/\mathcal{G}$ the orbit space of $\mathbb{M}$ under the action of $\mathcal{G}$ with projection map $\pi : \mathbb{M} \to \mathbb{X}$. To be able to consider triangulations of $\mathbb{X}$ according to Definition 1.2.3, we first give a definition for a simplex in $\mathbb{X}$. This is an immediate extension of the definition of simplices in the flat torus (Definition 2.1.2).

Definition 4.1.1 (Simplex in $\mathbb{M}$)**.** *Let σ be a k-simplex in $\mathbb{M}$. If the restriction $\pi|_\sigma$ of π to σ is injective, the image of σ by π is called a* k-simplex *in $\mathbb{X}$.*

Intuitively, this definition requires simplices not to self-intersect in the orbit space. We write $DT(\mathcal{Q})$ to denote the Delaunay triangulation of $\mathbb{M}$ defined by a point set $\mathcal{Q}$. Let $\mathcal{P}$ be a point set in $\mathbb{M}$. We can now adapt Definition 2.2.2 to the Delaunay triangulation of $\mathbb{X}$ defined by $\pi(\mathcal{P})$:

Definition 4.1.2 (Delaunay triangulation of $\mathbb{X}$)**.** *If $\pi(DT(\mathcal{GP}))$ is a triangulation of $\mathbb{X}$ (which subsumes that it is a simplicial complex according to Definition 1.2.2), then we call it the* Delaunay triangulation of $\mathbb{X}$ *defined by $\pi(\mathcal{P})$.*

We introduce some basic notions from group theory that are used later on. Let $\mathcal{G}$ be a group and $\mathcal{H}$ denote a subgroup of $\mathcal{G}$. $\mathcal{H}$ is called *normal* in $\mathcal{G}$ if it is invariant under conjugation, i.e. if for all $h \in \mathcal{H}$ and $g \in \mathcal{G}$, $ghg^{-1} \in \mathcal{H}$. For a group element $g \in \mathcal{G}$, the set $\{gh \mid h \in \mathcal{H}\}$ is called a *coset* of $\mathcal{H}$ in $\mathcal{G}$. The *index of a subgroup* $\mathcal{H}$ in $\mathcal{G}$ is defined as the number of cosets of $\mathcal{H}$ in $\mathcal{G}$.

In the subsequent sections we prove adapted versions of Theorem 2.2.8 and provide some geometric tests to decide whether a given point set defines a triangulation in $\mathbb{X} = \mathbb{M}/\mathcal{G}$, for different classes of spaces $\mathbb{M}$ and groups $\mathcal{G}$.

4.2 Flat spaces

In this section we consider d-dimensional flat spaces. The flat torus discussed in Chapter 2 is a special case of the spaces considered in this section. While the Delaunay triangulation of the flat torus fulfills the needs of many application fields, some of them, like computational biology [Ber09], require more general manifolds that are orbit spaces of $\mathbb{E}^3$ under the action of other crystallographic groups.

We first introduce closed Euclidean d-manifolds and their properties. Section 4.2.2 then studies Delaunay triangulations of closed Euclidean d-manifolds and shows, using the Bieberbach theorem, that there is always a finitely-sheeted covering space of the manifold in which the Delaunay triangulation is defined for any set of points. Section 4.2.3 discusses the generalization of Algorithm 2.3.1 (see page 29) to closed Euclidean manifolds. Most concepts mentioned below are taken from [Thu97].

4.2.1 Closed Euclidean manifolds

A *closed manifold* is a compact manifold without boundary. A d-manifold is called *Euclidean* or *flat* if every point has a neighborhood isometric to a neighborhood in $\mathbb{E}^d$.

A d-dimensional *Bieberbach group* $\mathcal{G}_B$ is a discrete group of isometries of $\mathbb{E}^d$ such that the orbit space $\mathbb{E}^d/\mathcal{G}_B$ is compact. Such groups are also called *crystallographic groups* or *space groups* [Thu97].

Theorem 4.2.1 (Bieberbach [Bie10])**.**

- *Let $\mathcal{G}_B$ be a d-dimensional Bieberbach group. There is a group $\mathcal{G}_T$ of d linearly independent translations that is a normal subgroup of $\mathcal{G}_B$ of finite index. The group $\mathcal{G}_T$ is called a* translational subgroup *of $\mathcal{G}_B$.*

- *For any d, there is only a finite number of d-dimensional Bieberbach groups, up to isomorphism.*

Note that the orbit space $\mathbb{E}^d/\mathcal{G}_B$ is not necessarily a manifold: If $\mathcal{G}_B$ leaves points fixed, these points do not have a neighborhood in $\mathbb{E}^d/\mathcal{G}_B$ that is homeomorphic to a neighborhood in $\mathbb{E}^d$. The orbit space $\mathbb{E}^d/\mathcal{G}_B$ can always be described by the more general concept of an *orbifold* [BMP03, Thu02]. For the orbit space to be a manifold, the group of action defining it must not have fixed points. In other words the group of action must be *torsion-free*, i.e., the identity must be the only element of finite order. If $\mathcal{G}_T$ is a subgroup of d independent translations of $\mathcal{G}_B$, then $\mathbb{E}^d/\mathcal{G}_T$ is a d-torus. It is sufficient to consider torsion-free Bieberbach groups to completely classify closed Euclidean manifolds:

Theorem 4.2.2 ([Thu97])**.** *Any closed Euclidean d-manifold is equal up to diffeomorphism to exactly one orbit space $\mathbb{E}^d/\mathcal{G}_B$, where $\mathcal{G}_B$ is a torsion-free d-dimensional Bieberbach group.*

According to Theorem 4.2.1, there are only finitely many d-dimensional Bieberbach groups, up to isomorphism. In dimension 2 there are 17, in dimension 3 there are 230. The number of Bieberbach groups by dimension is assigned the id A006227 in the On-Line Encyclopedia of Integer Sequences [Slo]; they are known up to dimension 5. The number of *torsion-free* Bieberbach groups is assigned the id A059104; they are known up to dimension 6.

In two dimensions, there are only two torsion-free Bieberbach groups and thus two closed Euclidean manifolds, up to isomorphism: the torus and the Klein bottle. In three dimensions, there are 10 closed Euclidean manifolds, four of which are non-orientable. A classification is given in [HW35] and [Thu97]:

- three linearly independent translations (3-torus)
- one screw motion and two linearly independent translations orthogonal to it. The screw motion can rotate by π, $2\pi/3$, $\pi/2$, $\pi/3$.
- three orthogonal screw motions that rotate by π.
- two linearly independent glide reflections in a plane. There are two different groups generated in this way.
- one glide reflection and one screw motion that rotates by π about an axis parallel to the reflection plane and orthogonal to the translation. Whether the screw axis lies inside the glide reflection plane or not yields two different manifolds.

The first six manifolds are orientable and the four last ones are not because their generators contain reflections.

4.2.2 Triangulations of Closed Euclidean Manifolds

Let $\mathcal{G}_F$ be a torsion-free d-dimensional Bieberbach group, $\mathcal{P}$ a finite point set in $\mathbb{E}^d$, $\mathbb{X} := \mathbb{E}^d/\mathcal{G}_F$ a closed Euclidean manifold with projection map $\pi : \mathbb{E}^d \to \mathbb{X}$, and $DT(\mathcal{G}_F\mathcal{P})$ the Delaunay triangulation of $\mathbb{E}^d$ defined by the infinite point set $\mathcal{G}_F\mathcal{P}$.

For the discussions below we need the following two values:

1. The minimum distance $\delta(\mathcal{G})$ by which a group $\mathcal{G}$ moves a point:

$$\delta(\mathcal{G}) = \min_{p\in\mathbb{E}^d, g\in\mathcal{G}, g\neq 1_\mathcal{G}} \operatorname{dist}(p, gp),$$

   where $1_\mathcal{G}$ denotes the unit element of $\mathcal{G}$. Note that if $\mathcal{G}$ is torsion-free and discrete, then $\delta(\mathcal{G}) > 0$ holds.

2. The diameter $\Delta(\mathcal{S})$ of the largest d-ball $\mathbb{B}$ in $\mathbb{E}^d$ that does not contain any point of a set $\mathcal{S}$ in its interior.

We now generalize Theorem 2.2.8 (page 24):

Theorem 4.2.3. *If the 1-skeleton of $\pi(DT(\mathcal{G}_F\mathcal{P}))$ does not contain cycles of length ≤ 2, then $\pi(DT(\mathcal{G}_F\mathcal{P}))$ is a triangulation of $\mathbb{X}$.*

Most parts of the proof of Theorem 2.2.8 are completely combinatorial and do not depend on the space: The proof of Lemma 2.2.3 is based on the fact that the restriction of π to a simplex is a homeomorphism. Observation 2.2.4 follows directly from Lemma 2.2.3 and the simplicial complex definition. The proofs of Lemmas 2.2.5, 2.2.6, and 2.2.7 are again based on Lemma 2.2.3 and the properties of π. Thus apart from Lemma 2.2.1, the whole discussion of Section 2.2.1 directly generalizes to closed Euclidean d-manifolds. We now prove the generalized version of Lemma 2.2.1:

Lemma 4.2.4. *Let $\mathcal{K}$ be a set of simplices in $\mathbb{E}^d$ whose vertices are exactly the elements of $\mathcal{G}_F\mathcal{P}$, and that fulfills conditions (i) and (ii) of Definition 1.2.2, and the Delaunay property with respect to $\mathcal{G}_F\mathcal{P}$. Then $\mathcal{K}$ satisfies the local finiteness property (iii) as well. Thus, $\mathcal{K}$ is a simplicial complex.*

Proof. We first consider a point p that is a vertex in $\mathcal{K}$. Note that $\delta(\mathcal{G}_F) > 0$ and $\Delta(\mathcal{G}_F\mathcal{P}) < \infty$ hold because $\mathcal{G}_F$ is a torsion-free Bieberbach group. The longest edge of $\mathcal{K}$ is bounded by $\Delta(\mathcal{G}_F\mathcal{P})$ and thus any point incident to p must lie in a ball of radius $\Delta(\mathcal{G}_F\mathcal{P})$ centered at p. As $\mathcal{G}_F$ is discrete, the number of points of $\mathcal{G}_F\mathcal{P}$ that lie inside such a ball is finite. Followingly, p is incident to only finitely many simplices.

Let us now consider a point p in $\mathbb{E}^d$ that is not a vertex in $\mathcal{K}$. Let σ denote the simplex that contains p in its interior and let v_σ denote a vertex of σ. Let $St(v_\sigma)$ denote the set of simplices that v_σ is incident to. Above we have shown that $St(v_\sigma)$ contains only finitely many elements. The set $St(\sigma)$ of simplices that σ is incident to is a subset of $St(v_\sigma)$, thus it is finite. There is a neighborhood $U(p)$ that has non-empty intersection with exactly the elements $St(\sigma)$. □

Criterion 2.3.3 mentions the threshold $\frac{1}{2}$, which depends on the group $\mathcal{G}$. The generalized version of this corollary follows by simple geometric reasoning from Theorem 4.2.3.

Corollary 4.2.5. *If $\Delta(\mathcal{G}_F\mathcal{P}) < \frac{\delta(\mathcal{G}_F)}{2}$, then $\pi(DT(\mathcal{G}_F\mathcal{P}'))$ is a triangulation of $\mathbb{X}$ for any finite $\mathcal{P}' \supseteq \mathcal{P}$.*

For any torsion-free Bieberbach group there are point sets such that the condition of Corollary 4.2.5 is fulfilled, because δ is strictly positive and Δ can be made arbitrarily small by the choice of the point set.

Finally, we give a generalized version of Lemma 2.3.2.

Lemma 4.2.6. *There is a normal subgroup $\mathcal{G}_C$ of $\mathcal{G}_F$ of finite index such that the projection of the Delaunay triangulation of $\mathcal{G}_F\mathcal{P} \cup \mathcal{G}_C Q$ in $\mathbb{E}^d$ onto $\mathbb{X}_C = \mathbb{E}^d/\mathcal{G}_C$ is a triangulation for any finite point set $\mathcal{P}$ in $\mathbb{E}^d$ and any $Q \subseteq \mathcal{G}_F q$ with any $q \in \mathbb{E}^d$.*

Proof. According to Theorem 4.2.1, there is a group $\mathcal{G}_T$ of d linearly independent translations that is a normal subgroup of $\mathcal{G}_F$ with finite index h'. We choose generators $g_1, \ldots, g_d$ of $\mathcal{G}_T$ in the following way: Let g_1 be the shortest translation in $\mathcal{G}_T$. Let g_{i+1} be the shortest translation in $\mathcal{G}_T$ that is linearly independent of the translations $g_1, \ldots, g_i$. Note that $\Delta(\mathcal{G}_T p)$ does not depend on a specific choice of p and thus can be considered constant. We can find an integer coefficient c such that for each g_i the inequality $\operatorname{dist}(p, g_i^c p) > 2\Delta(\mathcal{G}_T p)$ holds for any $p \in \mathbb{E}^d$. The group $\mathcal{G}_C$ generated by $g_1^c, \ldots, g_d^c$ is a subgroup of $\mathcal{G}_T$ of index c^d with the property $\delta(\mathcal{G}_C) > 2\Delta(\mathcal{G}_T p)$ for any $p \in \mathbb{E}^d$. As $\mathcal{G}_T$ is normal in $\mathcal{G}_F$ we have $g g_T g^{-1} \in \mathcal{G}_T$ for each $g \in \mathcal{G}_F, g_T \in \mathcal{G}_T$. By construction of $\mathcal{G}_C$ there is a bijection between the $g_T \in \mathcal{G}_T$ and the $g_C \in \mathcal{G}_C$ given by $g_C = g_T^c$. Now it is easy to see that $\mathcal{G}_C$ is a normal subgroup of $\mathcal{G}_F$ with index $h = h' \cdot c^d$. Note that $\Delta(\mathcal{G}_C \mathcal{G}_F \mathcal{P}) = \Delta(\mathcal{G}_F \mathcal{P}) \leq \Delta(\mathcal{G}_T p)$ for any $p \in \mathbb{E}^d$. Thus $\Delta(\mathcal{G}_C \mathcal{G}_F \mathcal{P}) < \frac{\delta(\mathcal{G}_C)}{2}$ holds and according to Corollary 4.2.5 the projection of the Delaunay triangulation of $\mathcal{G}_C \mathcal{G}_F \mathcal{P} = \mathcal{G}_F \mathcal{P}$ onto $\mathbb{X}_C$ forms a triangulation, which remains true even when adding further points. □

Note that the proof is constructive, i.e., it describes how to construct $\mathcal{G}_C$ from $\mathcal{G}_T$. The group $\mathcal{G}_T$ can be constructed from $\mathcal{G}_F$, e.g. using the Reidemeister-Schreier algorithm [Sim94]. An implementation of the Reidemeister-Schreier algorithm is available, for instance, in GAP [gap]. Lemma 4.2.6 means that there exists a space $\mathbb{X}_C$, in which the point set $\pi(\mathcal{P})$ defines a Delaunay triangulation. The space $\mathbb{X}_C$ is a covering space of $\mathbb{X}$ with a finite number of sheets [Arm82]. Lemma 4.2.6 can also be understood by constructing $\mathbb{X}_C$ from $\mathbb{X}$ directly, as follows.

Each closed Euclidean d-manifold has a d-torus as covering space with a finite number of sheets. This follows from Theorem 4.2.1 as discussed above. A fundamental domain of the d-torus is a d-dimensional hyperparallelepiped. By gluing two of these hyperparallelepipeds together, we get a new covering space that is again a d-torus. We can construct $\mathbb{X}_C$ by gluing as many copies of the fundamental domain as necessary to fulfill the condition in Corollary 4.2.5, i.e. $\Delta(\mathcal{G}_C \mathcal{G}_F \mathcal{P}) = \Delta(\mathcal{G}_F \mathcal{P}) < \frac{\delta(\mathcal{G}_C)}{2}$. See Figure 4.1 for an illustration in two dimensions.

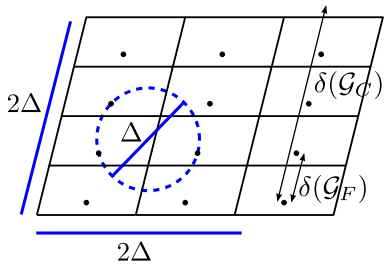


Figure 4.1: Sufficient number of copies of the fundamental domain.

As an example we consider the flat Klein bottle $\mathbb{E}^2/\mathcal{G}_K$, where $\mathcal{G}_K$ is the group generated by a translation g_t and a glide-reflection g_g, that is, a reflection together with a translation parallel to the reflection axis (see Figure 4.2). The group generated by g_t and g_g^2 is a

translational subgroup of $\mathcal{G}_K$ of index 2. Now we can choose a subgroup of this translational subgroup with finite index that fulfills the condition of Lemma 4.2.6 as in Figure 4.1.

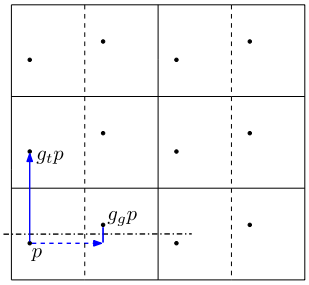


Figure 4.2: A part of the infinite point grid $\mathcal{G}_K p$.

Note that in both Corollary 4.2.5 and Lemma 4.2.6 we deal with the condition of the form $\Delta < \frac{\delta}{2}$. In Corollary 4.2.5 we adapt the point set $\mathcal{P}$ to decrease Δ, in Lemma 4.2.6 we adapt the group $\mathcal{G}_C$ to increase δ.

4.2.3 Algorithm

Algorithm 2.3.1 generalizes to $\mathbb{X} = \mathbb{E}^d/\mathcal{G}_F$ using the results given in the previous section. The algorithm is incremental, i.e., the points of $\mathcal{P}$ are added one by one.

- The algorithm determines $\mathcal{G}_C$ from $\mathcal{G}_F$ as described in Section 4.2.2.
- Then the algorithm starts computing in the h-sheeted covering space $\mathbb{X}_C = \mathbb{E}^d/\mathcal{G}_C$ as in Lemma 4.2.6, inserting h copies per input point (here we call *copy* of a point p an element of its orbit under the action of the quotient group $\mathcal{G}_F/\mathcal{G}_C$, i.e. a point gp, for $g \in \mathcal{G}_F/\mathcal{G}_C$).
- Once the condition of Corollary 4.2.5 is met for the current point set, the algorithm switches to computing in $\mathbb{X}$ and continues to insert each of the remaining points only once.

If $\mathcal{P}$ is such that the condition of Corollary 4.2.5 is never fulfilled, then the algorithm returns the triangulation of the covering space $\mathbb{X}_C$.

Two issues appear, namely, how to store the current triangulation and how to insert a point.

Space-efficient data structure

The triangulation can be stored as a graph in the following way: Full-dimensional simplices are stored with a list of their vertices and neighbors. Each vertex contains the coordinates of the point it corresponds to. Additionally, each d-simplex stores the information on how to map it isometrically into $\mathbb{E}^d$, i.e. an appropriate element of the simplex' preimage under the projection map π. However, this approach is not very space efficient since for large dimensions the number of d-simplices in a triangulation can grow very large. A more space efficient approach is to store the 1-skeleton [BDH09]. In this case, each edge must be described by its two vertices together with their offsets, where the offsets are elements of $\mathcal{G}_C$ or $\mathcal{G}_F$, respectively. A d-simplex can then be constructed by translating edges such that their offsets at common vertices agree, see Figure 4.3.

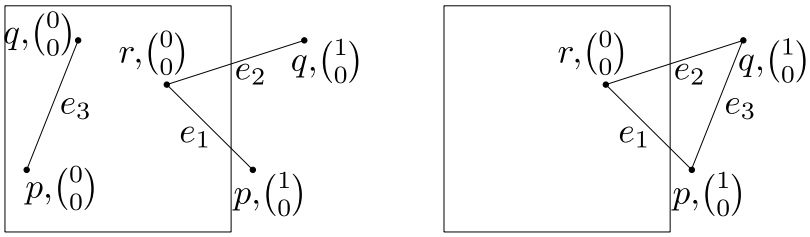


Figure 4.3: Left: Three edges. Right: Add $\binom{1}{0}$ to both offsets of e_3 to form the triangle pqr.

Point insertion

For the point insertion the approach by Bowyer [Bow81] and Watson [Wat81] can be used. If $\mathbb{X}$ is orientable, the insertion routine ensures that data structure only stores positively oriented simplices. If $\mathbb{X}$ is non-orientable, this is not possible. In this case we must apply an orientation test on the preimage under π of the simplex first before testing whether the point lies inside or outside the respective d-ball.

4.2.4 Flat orbifolds

A natural question is how the extend the results to general orbifolds. The results of Section 4.2.2 exclude Bieberbach groups with fixed points. However, from the Bieberbach theorem we know that any orbifold has a finitely sheeted covering space that is a closed Euclidean manifold and on which our approach works. So, while the approach cannot compute a triangulation in the orbifold, it can always compute a Delaunay triangulation in that covering space.

4.3 Spherical spaces

In this section we extend the approach described in Chapter 2 onto orbit spaces of a sphere under the action of a discrete group of isometries. We first discuss Delaunay triangulations of the sphere. Then we identify the orbit spaces that our approach can handle and finally discuss the Delaunay triangulation of a spherical orbifold.

4.3.1 Triangulations of the sphere

Let $\mathbb{S}^d$ denote d-dimensional unit sphere. In order to use Definitions 4.1.1 and 4.1.2, we must give a definition for simplices and the Delaunay triangulation of the d-dimensional sphere. The following definitions are straightforward extensions of the flat case.

Let $\rho : \mathbb{E}^{d+1} - \{0\} \to \mathbb{S}^d$ denote the radial projection, i.e., ρ projects all points of a ray R starting at the origin onto the intersection point of R and $\mathbb{S}^d$.

Definition 4.3.1 (Spherical simplex)**.** *Let $\mathcal{Q}$ be a set of $k+1$ points in $\mathbb{S}^d$ with $k \leq d$. Let $\mathrm{Ch}(\mathcal{Q})$ denote the convex hull of $\mathcal{Q}$ in $\mathbb{E}^{d+1}$. If $\mathrm{Ch}(\mathcal{Q})$ does not contain the origin, we call the image under ρ of $\mathrm{Ch}(\mathcal{Q})$ a* spherical k-simplex.

The circumscribing ball of a spherical d-simplex can be defined in the same way as for $\mathbb{E}^d$ using the spherical metric. It can be constructed as follows: Consider the $(d-1)$-ball

defined by the d vertices of the simplex, lying in a $(d-1)$-hyperplane in $\mathbb{E}^d$. The projection under ρ of this $(d-1)$-ball is the circumscribing ball of the d-simplex.

Definition 4.3.2 (Spherical Delaunay triangulation)**.** *The* Delaunay triangulation *of $\mathcal{P}$ in $\mathbb{S}^d$ is a triangulation of $\mathbb{S}^d$ such that the circumscribing ball of each d-simplex of the triangulation does not have any point of $\mathcal{P}$ in its interior.*

The Delaunay triangulation of the sphere $\mathbb{S}^d$ defined by $\mathcal{P}$ is the convex hull of $\mathcal{P}$ in $\mathbb{E}^{d+1}$. As $\mathcal{P}$ lies in the d-dimensional sphere embedded in $\mathbb{E}^{d+1}$, all points of $\mathcal{P}$ appear in its convex hull in $\mathbb{E}^{d+1}$. It is more efficient to directly compute the Delaunay triangulation in a d-dimensional space instead of resorting to $\mathbb{E}^{d+1}$. [NLC02] describe how to compute a spherical Voronoi diagram in two dimensions by computing two planar Voronoi diagrams. In [CdCL+10] we describe two approaches and an implementation for the case of $d = 2$. These approaches use the algorithm for computing the Delaunay triangulation in $\mathbb{E}^2$ and adapt the predicates accordingly to give the correct results for the Delaunay triangulation of the sphere.

The approaches discussed in [CdCL+10] extend to d dimensions. The predicate to test whether the circumscribing ball of a d-simplex contains a given point can be computed as follows: Let σ be a spherical d-simplex. The vertices of σ define a d-dimensional hyperplane in $\mathbb{E}^{d+1}$. Let p denote the query point. If p and the origin lie on different sides of the hyperplane, then p lies inside the circumscribing ball of σ.

If all the points of $\mathcal{P}$ lie in a half-sphere, $\mathcal{P}$ does not define a Delaunay triangulation of $\mathbb{S}^d$. In order to have a Delaunay triangulation that is homeomorphic to $\mathbb{S}^d$ we can for instance introduce a virtual vertex similar to the vertex at infinity for Delaunay triangulations of $\mathbb{E}^d$, see Section 3.3.

4.3.2 Spherical orbit spaces

In a similar manner as in Section 4.2, our approach works for d-dimensional discrete groups of isometries of the sphere $\mathbb{S}^d$. The group of isometries of the d-dimensional sphere is the orthogonal group $\mathcal{O}(d+1)$. The discrete subgroups of $\mathcal{O}(d+1)$ are finite because the sphere is compact: Otherwise the orbit of a point would be infinite and infinite subsets of a compact space have an accumulation point and are thus not discrete. The discrete subgroups of $\mathcal{O}(d+1)$ are called *point groups* and for each dimension there are infinitely many of them. We briefly discuss the two- and three-dimensional point groups.

There are two types of finite point groups in two dimensions: (1) cyclic groups C_n: generated by a rotation of $360°/n$; (2) dihedral groups: generated by a rotation of $360°/n$ and a reflection; see Figure 4.4 for an illustration.

In three dimensions, there are seven series of point groups, each of which contains an infinite number of groups, and seven more point groups. The seven infinite series all have a rotation of $360°/n$ for any $n > 0$ as one generator. The remaining seven point groups are: the symmetry group of the tetrahedron together with two subgroups as well as the symmetry groups of the octahedron and the icosahedron with one subgroup each. For further reading see [Cox91].

Note that among the three-dimensional point groups, the only group without fixed points is the group that maps each point to its antipodal point on the sphere. The orbit space under the action of this group is the projective plane, which is thus the only two-dimensional spherical manifold besides the sphere itself. All other orbit spaces are actually orbifolds [Sti92].

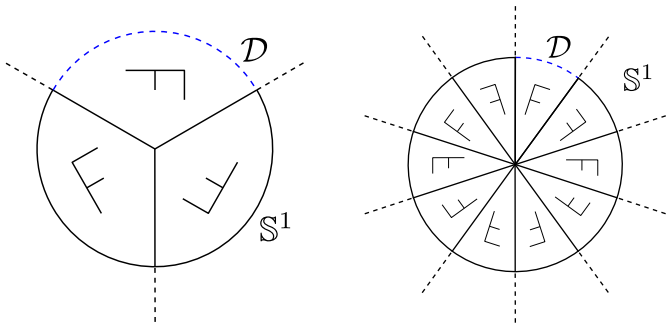


Figure 4.4: Left C_3: rotation of 360°/3. Right D_5: rotation of 360°/5 and reflection. The dashed arcs ($\mathcal{D}$) are the respective fundamental domains.)

4.3.3 Triangulations of spherical orbit spaces

The generalization of Theorem 2.2.8 and Algorithm 2.3.1 is similar to Section 4.2.2. The main difference is that the number of groups to consider is infinite while the groups themselves are finite.

Let $\mathcal{G}_P$ denote a d-dimensional point group, $\mathbb{X} := \mathbb{S}^d/\mathcal{G}_P$ an orbit space with projection map $\pi : \mathbb{S}^d \to \mathbb{X}$, and $DT(\mathcal{G}_P\mathcal{P})$ the Delaunay triangulation of $\mathbb{S}^d$ defined by the finite point set $\mathcal{G}_P\mathcal{P}$.

We now generalize Theorem 2.2.8.

Theorem 4.3.3. *If the 1-skeleton of $\pi(DT(\mathcal{G}_P\mathcal{P}))$ does not contain cycles of length ≤ 2, then $\pi(DT(\mathcal{G}_P\mathcal{P}))$ is a triangulation of $\mathbb{X}$.*

Like for Section 4.2.2, the definitions and lemmas of Section 2.2.1 generalize directly to $\mathbb{X}$, except Lemma 2.2.1: However, the proof of Lemma 2.2.1 turns out to be trivial in the spherical case because $\mathcal{G}_P\mathcal{P}$ is finite and so $DT(\mathcal{G}_P\mathcal{P})$ is finite, too.

As $\mathcal{G}_P\mathcal{P}$ is finite, we can always choose the sphere $\mathbb{S}^d$ as a finite covering space of $\mathbb{X}$, in which the point set $\mathcal{G}_P\mathcal{P}$ defines a triangulation if it contains more than $d+2$ points, not all within one half-sphere.

With the modifications described in Section 4.3.1, the incremental algorithm by Bowyer and Watson works for computing the Delaunay triangulation of $\mathbb{S}^d$. Algorithm 2.3.1 can be modified to work in $\mathbb{X} = \mathbb{S}^d/\mathcal{G}_P$ using the results from Section 4.3.3:

- The algorithm starts computing in the $|\mathcal{G}_P|$-sheeted covering space $\mathbb{S}^d$, inserting $|\mathcal{G}_P|$ copies per input point.
- Once the projection of the current triangulation under π has no cycles of length two and it can be shown that adding further points cannot introduce cycles of length two, the algorithm switches to computing in $\mathbb{X}$.

Note that if $\mathcal{G}_P$ is not torsion-free, then the algorithm returns $DT(\mathcal{G}_P\mathcal{P})$. We now give a geometric criterion similar to Criterion 2.3.3 to decide when to switch back to computing in $\mathbb{X}$ for the case of the real projective plane. We do not have a general criterion.

The real projective plane
Let $\mathcal{G}$ denote the group generated by the isometry that maps a point of $\mathbb{S}^2$ onto its antipodal point. The orbit space $\mathbb{S}^2/\mathcal{G}$ is the real projective plane. We can formulate the following criterion:

Criterion 4.3.4. *If the largest spherical disk in $\mathbb{S}^2$ that does not contain any point of $\mathcal{GP}$ has diameter smaller than $\pi/2$, then $\mathcal{P}$ defines the Delaunay triangulation of the real projective plane $\mathbb{X}/\mathcal{G}$.*

Proof. This proof contains the same ideas as the proof of Criterion 2.3.3, adapted to the spherical case: If the largest empty spherical disk has diameter smaller than $\frac{\pi}{2}$, then the longest edge in the Delaunay triangulation of $\mathcal{GP}$ is shorter then $\frac{\pi}{2}$ and thus a path of two edges is shorter than π and cannot form a two-cycle when projected onto $\mathbb{S}^2/\mathcal{G}$. □

Criterion 4.3.4 can be used by the algorithm described above to decide when to convert the computed Delaunay triangulation from the 2-sheeted covering space $\mathbb{S}^2$ to $\mathbb{S}^2/\mathcal{G}$.

4.4 A hyperbolic space

Here, we discuss how to extend the approach of Chapter 2 to orbit spaces of the hyperbolic plane under the action of a discrete group of hyperbolic isometries. We first give a short introduction on the hyperbolic plane $\mathbb{H}^2$ and on hyperbolic Delaunay triangulations. Then we discuss groups of hyperbolic isometries and extend Theorem 2.2.8 and Algorithm 2.3.1 to this setting. The groups of hyperbolic isometries turn out to be infinite and their theory is much richer than for groups of Euclidean isometries. We discuss three different approaches for the case of the group of the double torus.

4.4.1 The hyperbolic plane $\mathbb{H}^2$

The hyperbolic plane has the property that given a line l and a point p not in l there are infinitely many lines through p that do not intersect l, see Figure 4.5.

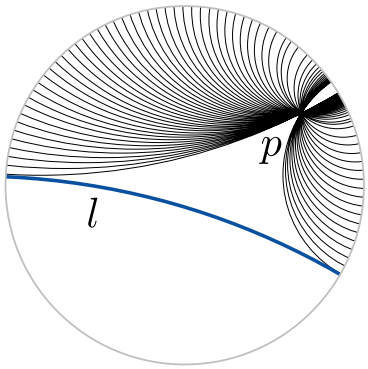


Figure 4.5: Parallel lines in the Poincaré disk: All lines through the point p are parallel to l [pdh].

There are four common models to represent the hyperbolic plane: the Beltrami-Klein Model, the Poincaré disk model, the Poincaré upper halfplane model and the hyperboloid model [Sti92]. We only use the Poincaré disk model in the subsequent discussions.

In the Poincaré disk model, all points of the hyperbolic plane are mapped inside the Euclidean unit disk. The points on the unit circle are points at infinity. Hyperbolic lines correspond to diameters of the unit disk or circular arcs that intersect the unit circle orthogonally. Hyperbolic circles correspond to Euclidean circles, though their centers differ if they are not centered in the origin. The local distance function is given by $ds = \frac{2\sqrt{dx^2+dy^2}}{1-(x^2+y^2)}$. Thus the distance of a point $\binom{x}{y}$ to the origin is $2\tanh^{-1}(\sqrt{x^2+y^2})$.

There are four types of hyperbolic isometries; they can be characterized by their fixed points at infinity. A hyperbolic isometry that has no fixed points is a rotation; if it has exactly one fixed point at infinity it is called limit rotation. Hyperbolic translations are isometries with exactly two fixed points at infinity. These are the three orientation-preserving hyperbolic isometries. The fourth type of hyperbolic isometries are the non-orientation-preserving glide reflections [Sti92]. We now introduce hyperbolic translations in more detail.

The two points left invariant by a translation t define a unique geodesic in the hyperbolic plane. The translation t translates a point along this invariant geodesic. The translation t maps a point of distance d to the geodesic onto another point on the equidistant line of distance d to the geodesic. This is due to the fact that t is an isometry, i.e. $\text{dist}(p,q) = \text{dist}(t(p),t(q))$. Note that if p and q do not have the same distance to the invariant geodesic, then the distance by which they are translated differs, i.e. $\text{dist}(p,t(p)) \neq \text{dist}(q,t(q))$. That is the closer a point p lies to the invariant geodesic the shorter distance $\text{dist}(p,t(p))$ by which they are translated. This is a fundamental difference to the Euclidean plane. For an illustration of hyperbolic translations see Figure 4.6. For instance, in case the two fixed points are not antipodal, note that the distance of a hyperbolic translation along the Euclidean chord is larger than along the hyperbolic geodesic.

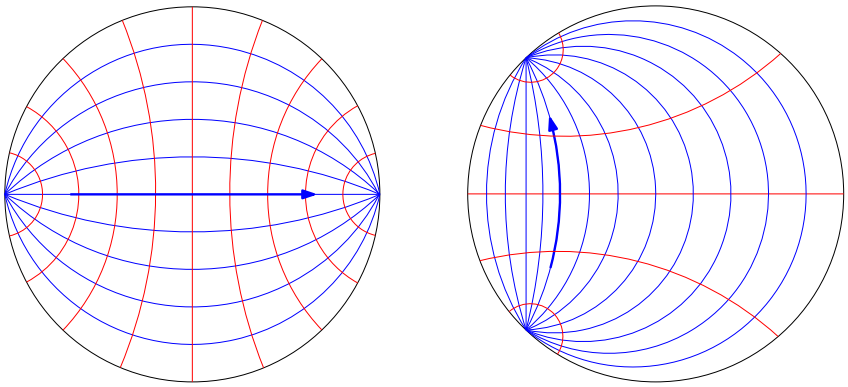


Figure 4.6: Left: Translation along a diameter of the Poincaré disk. Right: Translation along a general geodesic.

As geodesics are unique in the hyperbolic plane, the *hyperbolic convex hull* of a point set $\mathcal{P}$ can be defined in the classical way as the smallest set $\mathcal{S}$ such that the geodesic between two points p and q of $\mathcal{S}$ is entirely contained in $\mathcal{S}$. Then a *hyperbolic edge* or *hyperbolic triangle* is the convex hull of two or three points in $\mathbb{H}^2$, respectively.

In [DMT92b] the authors consider hyperbolic Voronoi diagrams using the property that in the Poincaré models hyperbolic circles are Euclidean circles. In this way, the algorithm for computing the Euclidean Delaunay triangulation can be used directly to compute the

hyperbolic Delaunay triangulation because the empty circles are exactly the same in both cases.

Nielsen and Nock [NN09] describe how to compute hyperbolic Voronoi diagrams using the Beltrami-Klein model. Their approach as well as the work of [DMT92b] can be extended to $\mathbb{H}^d$.

The orientation-preserving discrete groups of isometries on $\mathbb{H}^2$ are called *Fuchsian groups* [Kat92]. Fu and Wang [FW09] describe isometries and discrete isometry subgroups of d-dimensional hyperbolic spaces.

4.4.2 The double torus

Here we only consider the double torus, which is one of the most simple orbit spaces of constant negative curvature.

We first require some more notions from group theory. Let $\mathcal{H}$ be a group and $\mathcal{H}'$ denote a subgroup of $\mathcal{H}$. The *quotient group* $\mathcal{H}/\mathcal{H}'$ is the set of all cosets of $\mathcal{H}'$ in $\mathcal{H}$ with the product of subsets[1] as group operation. A group is said to be *free* if each group element can be written in a unique way as a product of generators. We denote the free group with generators G by $< G >$. Let R denote a subset of $< G >$, called set of *relations*. The quotient group of the free group $< G >$ and the normal subgroup of $< G >$ generated by the relations in R is denoted by $< G \mid R >$. This notation is called *group presentation*.

The double torus can now be constructed as an orbit space under the action of a group generated by four hyperbolic translations. Let a, b, c, and d denote four hyperbolic translations and $\overline{a}, \overline{b}, \overline{c}$, and $\overline{d}$ their respective inverse translations. There are at least two groups acting on $\mathbb{H}^2$ that define a double torus:

$$\begin{aligned} \mathcal{G} &:= < a, b, c, d \mid abcd\overline{a}\overline{b}\overline{c}\overline{d} > \\ \mathcal{G}' &:= < a, b, c, d \mid ab\overline{a}\overline{b}cd\overline{c}\overline{d} > \end{aligned}$$

The groups $\mathcal{G}$ and $\mathcal{G}'$ are isomorphic but not equal [CM57]. The isomorphism can be constructed geometrically using the algorithm described in [VY90]. For more discussions on this see also [AB91, Thr32].

Here we consider the groups for which the fundamental domain is the regular octagon centered at the origin. The generators of $\mathcal{G}$ and $\mathcal{G}'$ respectively map octagon edges onto octagon edges. This can be interpreted as identifying the edges as shown in Figure 4.7.

The generators for $\mathcal{G}$ can be chosen as follows: Let a be the hyperbolic translation that translates points along the Euclidean x-axis by $2\tanh^{-1}\left(\sqrt{\frac{2}{1+\sqrt{2}}}\right)$. Let b, c, d be the same translations rotated by $\pi/4$. The fundamental domain of $\mathcal{G}$ is the regular octagon with angles of $\pi/4$. If we consider the tiling of the hyperbolic plane by such octagons, at each vertex of the tiling eight fundamental domains meet to complete the full angle of 2π.

We do not consider the group $\mathcal{G}'$. The group $\mathcal{G}'$ has the advantage that it is easier to see that the identifications it induces on the octagon yield a double torus. However, its generators are more complicated to deal with.

4.4.3 Triangulations of the double torus

We first extend Theorem 2.2.8 to the case of the double torus. Let $\mathcal{P}$ be a finite point set in $\mathbb{H}^2$, $\mathcal{G}$ the group of the double torus as defined in Section 4.4.2, $\mathbb{H}^2/\mathcal{G}$ the orbit space with

[1] For S, T subsets of $\mathcal{H}$, $ST := \{st \mid s \in S \text{ and } t \in T\}$

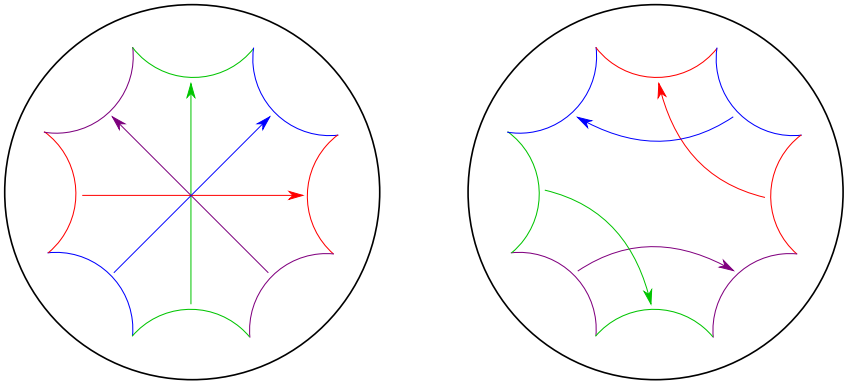


Figure 4.7: Left: Identification scheme for group $\mathcal{G}$. Right: Identification scheme for group $\mathcal{G}'$.

projection map $\pi : \mathbb{H}^2 \to \mathbb{H}^2/\mathcal{G}$, and $DT(\mathcal{GP})$ the Delaunay triangulation of $\mathbb{H}^2$ defined by the infinite periodic point set $\mathcal{GP}$. Then Theorem 2.2.8 can be formulated as follows for the hyperbolic case:

Theorem 4.4.1. *If the 1-skeleton of $\pi(DT(\mathcal{GP}))$ does not contain cycles of length ≤ 2, then $\pi(DT(\mathcal{GP}))$ is a triangulation of $\mathbb{X}$.*

The only missing part in the proof of Theorem 4.4.1 is the local finiteness (Lemma 2.2.1). All the other lemmas are purely combinatorial and only argue using properties of simplices, independent of their embedding.

We now generalize Lemma 2.2.1:

Lemma 4.4.2. *A set of simplices $\mathcal{K}$ that fulfills (i), (ii) (cf. page 3), and the Delaunay property with respect to $\mathcal{GP}$, is a simplicial complex in $\mathbb{H}^2$.*

Proof. In order to show that $\mathcal{K}$ is a simplicial complex it remains to prove that condition (iii) of the simplicial complex definition (local finiteness) is fulfilled. We use a similar argumentation as in the proof of Lemma 2.2.1 (see page 21).

Assume there is a vertex v with an infinite number of incident simplices and thus an infinite number of incident edges. Let t_v denote the hyperbolic translation that moves v to the origin. The Delaunay property is invariant under the action of isometries, so we can apply t_v onto $\mathcal{K}$ without losing the Delaunay property. Thus, without loss of generality we can assume that v lies in the origin. The orbit of v is a point set, in which the diameter of the largest empty disk is bounded by the diameter of the circumscribing circle of the fundamental octagon, which is $\Delta := 2\sinh^{-1}\left(\frac{\sqrt{2(1+\sqrt{2})}}{sin(\pi/8)}\right) \approx 4.90$. So circumscribing circles of hyperbolic triangles that are incident to v and a point in the orbit of a point in $\mathcal{P}$ of distance larger than Δ from v cannot be empty. The disk of radius Δ centered at the origin is compact. From the fact that $\mathcal{G}$ is discrete, it follows that the number of elements of the orbit of a point in $\mathcal{P}$ that lie inside this disk is finite and $\mathcal{P}$ itself is finite, too.

We have shown that there are only finitely many triangles incident to v, which implies directly that the number of edges incident to v is finite, too. □

In the spaces we have discussed so far, we always used covering spaces to resolve this issue of input point sets that do not define a Delaunay triangulation. Below, we discuss

why it is much more difficult and of less interest to construct covering spaces of hyperbolic orbit spaces. We present two alternative approaches:

1. Compute with a finite number of copies of the fundamental domain of the group and compute the Delaunay triangulation of $\mathbb{H}^2$ defined by this finite set of points.
2. Start with a fixed initial point set that is guaranteed to define a Delaunay triangulation of the orbit space and remove these points only after some criterion similar to Criterion 2.3.4 is fulfilled.

Covering spaces

The original motivation for using covering spaces for computing triangulations in the flat torus was that the covering spaces of the flat torus are homeomorphic to the flat torus again. This follows from the Riemann-Hurwitz formula [Har77]: Let $\mathbb{X}$ be a topological space and $\widetilde{\mathbb{X}}$ a k-sheeted covering space. If $\chi(\mathbb{X})$ denotes the Euler characteristic of space $\mathbb{X}$, then the equation $\chi(\mathbb{X}) = k \cdot \chi(\widetilde{\mathbb{X}})$ holds.

Recall that the genus of a surface, i.e. the number of handles, is given by $1 - \frac{1}{2}\chi$. As the Euler characteristic of the flat torus is zero, any covering space of the flat torus is again a flat torus. The double torus has Euler characteristic -2. Followingly, the Euler characteristic decreases when the number of sheets in the cover increases. This corresponds to a larger number of handles.

The above argument on the number of handles can also be understood in a more intuitive way. Let us consider a two-dimensional torus embedded in $\mathbb{E}^3$. The torus has two non-trivial loops. The use of a covering space corresponds to considering a bigger torus with more points on it. Now the non-trivial loops are longer and two-cycles in the triangulation can be avoided. The process of generating a two-sheeted covering space of the double torus corresponds to cutting one non-trivial loop in both double tori and sew the open edges to the other double torus, see Figure 4.8. In this way one handle has been expanded, i.e., one non-trivial loop has become longer, and the other two handles are unchanged. The newly created space has two handles of original loop length and one handle of double loop length. So we have not actually changed the number of handles with small loop length, which was the original goal. Still, it might be possible to cut and glue the loops in a more complicated way in order to avoid introducing new short loops.

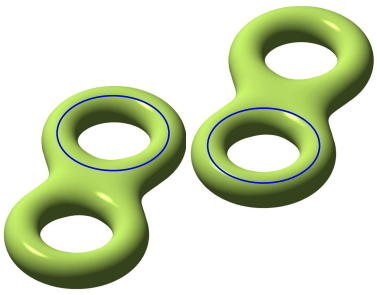


Figure 4.8: Constructing a two-sheeted covering space of the double torus [Ale].

Thus it appears difficult to construct a covering space in which cycles of length two can be avoided. Additionally, the original motivation of using covering spaces, namely that they are topologically equivalent to the original space, is not given anymore.

Copies of the fundamental domain in $\mathbb{H}^2$

Let F denote the fundamental domain of $\mathcal{G}$ centered at the origin and $\mathcal{P} \subset F$ be a finite point set. The diameter of F is $\Delta := 2\sinh^{-1}\left(\frac{\sqrt{2(1+\sqrt{2})}}{sin(\pi/8)}\right) \approx 4.90$, which is the diameter of the largest possible empty circle with respect to $\mathcal{GP}$. So there cannot be any edges longer than Δ in the Delaunay triangulation of $\mathcal{GP}$. Thus points that are further away than $\frac{3}{2}\Delta$ from the origin do not have any influence on the simplices of $DT(\mathcal{GP})$ that intersect F. So we have to consider the subset G of all translations $g \in \mathcal{G}$ such that gF and the disk of radius $\frac{3}{2}\Delta$ centered at the origin have non-empty intersection. We can use Dehn's algorithm in order to identify the set G [Lyn66]. We then compute the Delaunay triangulation of $\mathbb{H}^2$ defined by the point set $G\mathcal{P}$ and extract all simplices that intersect F. Preliminary estimations showed that the number of fundamental domains to consider is far above 400. Thus this approach turns out to be not very practical.

Another idea is to generalize the result of [DH97b] that we describe in Section 2.2.2 to the hyperbolic plane. Unfortunately Minkowski sums do not extend to the hyperbolic case. Nevertheless, we can define a set that is analogous to $F^{(n)}$ in hyperbolic space: Let $F_p^{(1)}$ denote the Voronoi cell of p in the hyperbolic Voronoi diagram of $\mathcal{G}p$. Then $F_p^{(1)}$ is a fundamental domain of $\mathcal{G}$. Now we define $F_p^{(n)} := \bigcup_{q \in F_p} F_q^{(n-1)}$.

Theorem 4.4.3. *Let τ be a simplex in $DT(\mathcal{GP})$ with at least one vertex in $F_p^{(1)}$. Then τ is completely contained inside $F_p^{(3)}$.*

Proof. The proof works in the same way as the proof of Lemma 2.2.9 in [DH97b]. Let σ and τ denote simplices in $DT(\mathcal{GP})$ such that σ has p as one of its vertices and τ has at least one vertex in $F_p^{(1)}$.

1. $F_p^{(1)}$ contains the center c_σ of the circumscribing circle of σ, see Figure 4.9. This is due to the construction of $F_p^{(1)}$ as a Voronoi cell: If c_σ was outside of $F_p^{(1)}$ then a point in the orbit of p would be closer to c_σ than p, which is a contradiction to the fact that c_σ is the circumcenter of a simplex with p as vertex.

2. $\sigma \subset F_p^{(2)}$: Note that from the definition of $F_p^{(1)}$ follows that if $c_\sigma \in F_p^{(1)}$, then $p \in F_{c_\sigma}^{(1)}$, and so $\sigma \subset F_{c_\sigma}^{(1)} \subset F_p^{(2)}$, by definition of $F_p^{(2)}$.

3. Let $q \in F_p^{(1)}$ be a vertex of τ. From the above argumentation we know that $\tau \subset F_q^{(2)}$, and by definition of $F_p^{(3)}$ the inclusion $F_q^{(2)} \subset F_p^{(3)}$, and thus $\tau \subset F_p^{(3)}$ holds.

□

In order to formulate a criterion to decide whether a point set defines a Delaunay triangulation of the double torus, we can have a similar approach as for the flat torus.

Criterion 4.4.4. *If the diameter of the largest circumscribing circle of the triangles of the Delaunay triangulation of $\mathcal{GP} \cap F_p^{(3)}$ that intersect $F_p^{(1)}$ is shorter than $\tanh^{-1}\left(\sqrt{\frac{2}{1+\sqrt{2}}}\right) \approx 1.53$, then any point set $\mathcal{Q}$ with $\mathcal{P} \subseteq \mathcal{Q}$ defines a Delaunay triangulation of the double torus.*

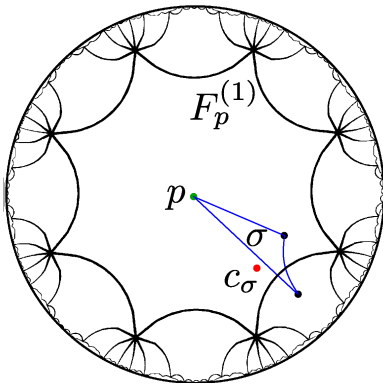


Figure 4.9: c_σ lies in $F_p^{(1)}$.

Proof. From Section 4.4.2, we know that the shortest distance by which a point can be translated under $\mathcal{G}$ is at least $2\tanh^{-1}\left(\sqrt{\frac{2}{1+\sqrt{2}}}\right) \approx 3.06$. So if all circumscribing circles of triangles that intersect $F_p^{(1)}$ are shorter than half this constant, then there cannot be cycles of length two even after adding new points and thus $\mathcal{Q}$ defines a Delaunay triangulation of the double torus according to Theorem 4.4.1. □

Algorithm 4.4.1 describes how to compute Delaunay triangulations of the double torus if possible.

Algorithm 4.4.1 Compute Delaunay triangulation of $\mathbb{H}^2/\mathcal{G}$ defined by the point set $\mathcal{P}$.

Input: Set $\mathcal{P}$ of points in $F_0^{(1)}$
Output: $\pi(DT(\mathcal{GP}))$ if it is a triangulation, $\emptyset$ otherwise

1: $\mathcal{P}' \Leftarrow \mathcal{P}$
2: Pop p from $\mathcal{P}'$
3: $\mathcal{P} \Leftarrow \{p\}$
4: $\mathrm{TR}_{\mathbb{H}} \Leftarrow DT(\mathcal{GP} \cap F_0^{(3)})$
5: **while** the longest edge in $\mathrm{TR}_{\mathbb{H}} \cap F_0^{(1)}$ is longer than $\tanh^{-1}\left(\sqrt{\frac{2}{1+\sqrt{2}}}\right)$ **do**
6: Pop p from $\mathcal{P}'$; $\mathcal{P} \Leftarrow \mathcal{P} \cup \{p\}$
7: **for all** $p' \in \mathcal{G}p \cap F_0^{(3)}$ **do**
8: Insert p' into $\mathrm{TR}_{\mathbb{H}}$
9: **end for** // $\mathrm{TR}_{\mathbb{H}} = DT(\mathcal{GP} \cap F_0^{(3)})$
10: **if** $\mathcal{P}' = \emptyset$
11: **if** $\pi(DT(\mathcal{GQ}))$ **is a triangulation for any** $\mathcal{Q} \supseteq \mathcal{P}$ **then goto** 14
12: **else exit**
13: **end while**
14: Compute $\pi(DT(\mathcal{GP}))$ from $\mathrm{TR}_{\mathbb{H}}$ `// switch to double torus`
15: Insert all points remaining in $\mathcal{P}'$ into $\pi(DT(\mathcal{GP}))$ one by one
16: **return** $\pi(DT(\mathcal{GP}))$

The correctness of the algorithm follows from Theorems 4.4.1 and 4.4.3 as well as Criterion 4.4.4. In case the input set $\mathcal{P}$ does not define a Delaunay triangulation of $\mathbb{H}^2/\mathcal{G}$,

there are actually several options possible. For simplicity we chose to return nothing. In an actual implementation it could be of interest to return the current structure $\mathrm{TR}_{\mathbb{H}}$. It would also be possible to return the simplices from $\mathrm{TR}_{\mathbb{H}}$ that intersect $F_0^{(1)}$ as their images under π form a partition of $\mathbb{H}^2/\mathcal{G}$ according to Theorem 4.4.3.

The main difficulty is to actually compute the point set $\mathcal{PG} \cap F_0^{(3)}$. One feasible approach is to compute the set G of isometries in $\mathcal{G}$ such that $GF_0^{(1)} \supseteq F_0^{(3)}$. We have not done this computation but we can give a simple lower bound on the cardinality of G. We consider the partition of the hyperbolic plane into hyperbolic octagons given by $\mathcal{G}F_0^{(1)}$. Consider the elements of $\mathcal{G}$ that map $F_p^{(1)}$ to one of its neighbors in the octagon partition of $\mathbb{H}^2$ described above. The union of all $F_0^{(1)}$ and all its neighbors clearly forms a subset of $F_p^{(3)}$: There is one neighbor per edge, i.e. eight, and five more neighbors per vertex, i.e. 40. Together with the original copy $F_p^{(1)}$ this makes $1 + 8 + 40 = 49$ copies to consider. Given that the number of 49 copies is only a lower bound on the required number of copies to consider, this approach does not seem to be very feasible in practice.

The approach described above is nevertheless nice because it is extensible to other discrete groups for which $\mathcal{GP} \cap F_0^{(3)}$ can be computed.

Dummy point set.

In this approach, we start with an initial triangulation of a point set $\mathcal{P}_D$ that is chosen such that any point set that contains $\mathcal{P}_D$ defines a Delaunay triangulation of $\mathbb{H}^2/\mathcal{G}$. This is the same approach as described in Section 3.3.9 for the case of the flat torus. If the input point set itself defines a Delaunay triangulation, the points of $\mathcal{P}_D$ can be removed from the triangulation.

Let $\mathcal{P}$ be a point set such that $\mathcal{P}_D \subseteq \mathcal{P}$. We want to prove that $\pi(DT(\mathcal{GP}_D))$ is a Delaunay triangulation of $\mathbb{H}^2/\mathcal{G}$. It is sufficient to show that the restriction of π on two non-disjoint disks that are empty with respect to $\mathcal{GP}_D$ is injective. This implies that the restriction of π on any two empty disks in $\mathcal{GP}$ is injective, too. Then there cannot be cycles of length two and thus $\pi(DT(\mathcal{GP}))$ is a Delaunay triangulation of $\mathbb{H}^2/\mathcal{G}$ according to Theorem 4.4.1.

We propose a point set $\mathcal{P}_D$ of 18 points in Figure 4.10 (9 points inside the fundamental domain, 8 on the border and 1 in the corner). As hyperbolic circumcircles are Euclidean circumcircles, it can be shown that $\pi(DT(\mathcal{P} \cup \mathcal{P}_D))$ is a Delaunay triangulation of $\mathbb{H}^2/\mathcal{G}$.

This approach appears to be the most practical among the presented ones. The drawback is that if the input point set does not define a Delaunay triangulation of $\mathbb{H}^2/\mathcal{G}$, the Delaunay triangulation defined by a modified input point set is returned. This approach generalizes to other groups, if an appropriate dummy point set can be given.

4.4.4 Discussion

We have presented three approaches to handle cases in which the input point set does not define a Delaunay triangulation of the double torus. In the hyperbolic case the motivation for using covering spaces, as done in the case of Euclidean or spherical spaces, is not given anymore. Covering spaces are not homeomorphic to the original space as in the flat torus and the universal covering space is not finite as in spherical orbit spaces. So the natural approach would be to compute in $\mathbb{H}^2$, using as many copies of the input point set as

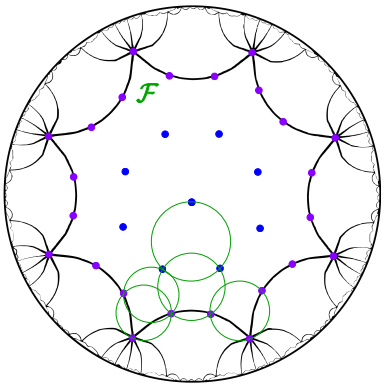


Figure 4.10: Proposed dummy point set $\mathcal{P}_D$ of 18 points for the double torus.

necessary to extract a subset of the triangulation onto which the identifications defined by the group $\mathcal{G}$ can be applied. This is possible, even though the number of required copies turns out to be very large. And finally, instead of changing the space we presented an approach that modifies the point set. This approach is very practical, in case the intended application can cope with 12 extra points in case of small or badly distributed input point sets.

Chapter 5

Conclusion and future work

We presented an approach to compute the Delaunay triangulation of $\mathbb{T}^3_{\boldsymbol{c}}$ defined by a given point set $\mathcal{P}$. The developed algorithm was implemented and is available through the open source library CGAL. Extensions and adaptations of this approach to other flat and spherical orbit spaces and to the double torus of constant (negative) curvature were discussed. During the course of this work some more interesting questions arose, which we present in this chapter.

5.1 Restriction to simplicial complexes

In this work we concentrate on computing simplicial complexes both for the purpose of mathematical soundness of the definition of the Delaunay triangulations we give as well as for practical considerations in the implementation, see also Section 1.4. Nevertheless, it might be interesting to consider tessellations that have the Delaunay property but are not simplicial complexes, i.e. not triangulations. Here, we briefly discuss the problems arising when computing with structures other than simplicial complexes.

Relaxed simplicial complexes
We could relax condition (ii) of the simplicial complex definition (Def. 1.2.2) to the following: Let σ and σ' be simplices in $\mathcal{K}$. Then the intersection $\sigma \cap \sigma'$ is a set of simplices in $\mathcal{K}$. The 1-skeleton of these relaxed simplicial complex could have non-trivial cycles of length two but not of length one, which would contradict the simplex definition (Def. 2.1.2). So the constants on the number of required copies and the edge length could be improved, see Figure 5.1. It seems feasible to design an efficient data structure for relaxed simplicial complexes so this approach might be an interesting direction for further examination. One severe drawback is that algorithms that require the data structure to be a simplicial complex must be modified accordingly. For instance, inserting a new point to the triangulation using the star-hole approach as described in Section 1.2.4 requires the hole to be homeomorphic to a d-ball. Relaxed simplicial complexes do not have this property.

Δ-complexes
Hatcher presents CW-complexes, first introduced by Whitehead in 1949, and describes a more restrictive type of complexes, the so-called Δ-complexes [Hat01]. In the Δ-complex the simplices are represented by maps that are not necessarily injective on the simplex

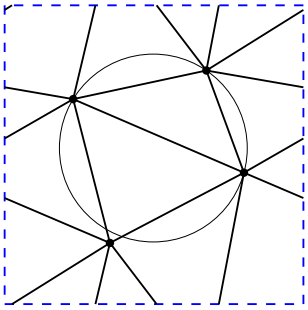


Figure 5.1: A relaxed simplicial complex in $\mathbb{T}^2$ that has the Delaunay property

boundaries, i.e., there can be self-loops. In this way, $\mathbb{T}^2$ can be tessellated by two triangles, three edges and one vertex. However, it is not described how a data structure storing Δ-complexes could be implemented. There is ongoing work on this topic by Colin de Verdière and Teillaud [dVT].

5.2 Restrictions on spaces

Our current approach has some clear limitations on the types of spaces it can handle. Here, we explore the reasons of these restrictions and try to develop ideas how to handle such cases.

Unbounded fundamental domains In Section 4.2, we consider only groups with bounded fundamental domain. Also, the groups of action defining the flat torus and the double torus as well as all groups of isometries on the sphere have bounded fundamental domains.

An example of a group with an unbounded fundamental domain is the group $\mathcal{G}_C$ generated by one translation acting on $\mathbb{E}^2$. The quotient space $\mathbb{E}^2/\mathcal{G}_C$ is the 2D cylinder. Let us assume that $\mathcal{G}_C$ is generated by the unit translation along the x-axis. Let $\mathcal{P}$ be a point set in $\mathbb{E}^2$, then $DT(\mathcal{G}_C\mathcal{P})$, the Delaunay triangulation of $\mathbb{E}^2$ defined by $\mathcal{G}_C\mathcal{P}$, has the following properties:

1. Let p_{max} be the point in $\mathcal{P}$ with largest y-coordinate. Let $t \in \mathcal{G}_C$ be one of the generating unit translations. Then there is an edge between p_{max} and tp_{max} in $DT(\mathcal{G}_C\mathcal{P})$.

2. The union of all simplices in $DT(\mathcal{G}_C\mathcal{P})$ is the convex hull of $\mathcal{G}_C\mathcal{P}$, which is not equal to $\mathbb{E}^2$: All points of larger y-value than p_{max} do not lie in the convex hull.

When projecting $DT(\mathcal{G}_C\mathcal{P})$ onto $\mathbb{E}^2/\mathcal{G}_C$, the first property implies that there is a self-loop in p_{max}. Followingly at least a three-sheeted covering space is necessary to avoid cycles of length two, independent of the point set $\mathcal{P}$, see Figure 5.2.

The second property implies that the projected triangulation is not a triangulation of a cylinder but only of an annulus. This yields a problem for the implementation because the current CGAL implementation cannot handle spaces with boundaries. In the case of $\mathbb{E}^2$ and $\mathbb{E}^3$ a vertex at infinity has been introduced in order to represent a triangulation of the one point compactifications $\mathbb{E}^2 \cup \{\infty\}$ and $\mathbb{E}^2 \cup \{\infty\}$, respectively, see Section 3.2.2. This

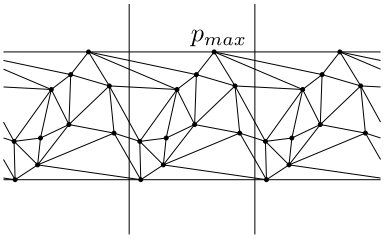


Figure 5.2: Delaunay triangulation of $\mathcal{G}_C\mathcal{P}$.

is not possible for the cylinder because it has two boundaries. One vertex at infinity would be part of a "cap" on both boundaries. Thus its neighborhood would not be homeomorphic to $\mathbb{E}^2$ and so $E^2/\mathcal{G}_C \cup \infty$ would not be a manifold. The option of adding two different vertices at infinity – one for the upper cap and one for the lower cap – is more difficult to handle in terms of implementation.

In three dimensions there are two different types of cylinders: The orbit spaces $\mathbb{E}^3/\mathcal{G}_C$ and $\mathbb{E}^3/\mathcal{G}_{2C}$, where $\mathcal{G}_{2C}$ denotes the group generated by two different unit translations. $\mathbb{E}^3/\mathcal{G}_C$ is a space that is periodic along one axis, $\mathbb{E}^3/\mathcal{G}_{2C}$ is periodic along two axes. For both spaces practical applications exist (see e.g. [Ber09]), so this should be an interesting direction of research to pursue.

A more general formulation of the problem would consider all orbit spaces of $\mathbb{E}^d$ under the action of a Bieberbach group of dimension smaller than d.

5.3 Hyperbolic orbit spaces

Unlike for the case of flat and spherical orbit spaces we do not yet have a general approach for hyperbolic orbit spaces. In Section 4.4, we give some approaches for the double torus that are expected to extend to other orbit spaces of $\mathbb{H}^2$ and even of $\mathbb{H}^d$. However, as the class of discrete groups of hyperbolic isometries is much richer than for the flat or the spherical case, there are still many open questions on this topic. In a next step it would be useful to extend our findings to surfaces of higher genus. Another topic to examine is orbit spaces of higher-dimensional hyperbolic spaces.

Bibliography

[AAD07] Nina Amenta, Dominique Attali, and Olivier Devillers. Complexity of Delaunay triangulation for points on lower-dimensional polyhedra. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms*, pages 1106–1113, 2007. 7

[AB91] R. Aurich and E. B. Bogomolny. Periodic orbits on the regular hyperbolic octagon. *Physica D*, 48:91–101, 1991. 90

[Abs09] Mark A. Abspoel. Computing homology of subcomplexes of the 3-torus. http://scripties.fwn.eldoc.ub.rug.nl/FILES/scripties/Wiskunde/Bachelor/2009/Abspoel.M.A./Mark_Abspoel_WB_2009.pdf, 2009. 72

[ACR03] Nina Amenta, Sunghee Choi, and Günter Rote. Incremental constructions con BRIO. In *Proceedings of the 19th Annual Symposium on Computational Geometry*, pages 211–219, 2003. 10

[AK00] Franz Aurenhammer and Rolf Klein. Voronoi diagrams. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 201–290. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000. 5

[Ale] Oleg Alexandrov. Double torus illustration. http://en.wikipedia.org/wiki/File:Double_torus_illustration.png. 92

[Arm82] Mark A. Armstrong. *Basic Topology*. Springer-Verlag, 1982. 11, 14, 25, 83

[AT07] Mridul Aanjaneya and Monique Teillaud. Triangulating the real projective plane. In *Mathematical Aspects of Computer and Information Sciences*, 2007. 16

[Aur91] Franz Aurenhammer. Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, September 1991. 5

[Aus98] Matthew H. Austern. *Generic Programming and the STL: Using and Extending the C++ Standard Template Library*. Addison-Weasley Longman Publishing Co., Inc., Boston, MA, USA, 1998. 39

[BBP01] Hervé Brönnimann, Christoph Burnikel, and Sylvain Pion. Interval arithmetic yields efficient dynamic filters for computational geometry. *Discrete Applied Mathematics*, 109:25–47, 2001. 41

[BDH09] Jean-Daniel Boissonnat, Olivier Devillers, and Samuel Hornus. Incremental construction of the Delaunay graph in medium dimension. In *Proceedings of the 25th Annual Symposium on Computational Geometry*, pages 208–216, 2009. 84

[Ber09] Julie Bernauer. Talk: Computational structural biology: Periodic triangulations for molecular dynamics. http://www-sop.inria.fr/geometrica/collaborations/OrbiCG/program.html, 2009. 2, 80, 99

[BEY91] Marshall Bern, David Eppstein, and Frances Yao. The expected extremes in a Delaunay triangulation. *International Journal of Computational Geometry and Applications*, 1:79–91, 1991. 36, 37

[Bie10] Ludwig Bieberbach. Über die Bewegungsgruppen des n-dimensionalen euklidischen Raumes mit einem endlichen Fundamentalbereich. *Göttinger Nachrichten*, 75, 1910. 80

[BL95] H. Borouchaki and S. H. Lo. Fast Delaunay triangulations in three dimensions. *Computer Methods in Applied Mechanics and Engineering*, 128:153–167, December 1995. 17

[BMP03] Michel Boileau, Sylvain Maillot, and Joan Porti. *Three-dimensional orbifolds and their geometric structures*. Société mathématique de France, Paris, 2003. 13, 81

[BO05] Jean-Daniel Boissonnat and Steve Oudot. Provably good sampling and meshing of surfaces. *Graphical Models*, 67:405–451, 2005. 18, 72

[Bow81] Adrian Bowyer. Computing Dirichlet tesselations. *The Computer Journal*, 24:162–166, 1981. 7, 8, 9, 10, 18, 27, 85

[Buc09] Kevin Buchin. Constructing Delaunay triangulations along space-filling curves. In *European Symposium on Algorithms*, volume 5757 of *Lecture Notes in Computer Science*, pages 119–130, 2009. 10

[BY98] Jean-Daniel Boissonnat and Mariette Yvinec. *Algorithmic Geometry*. Cambridge University Press, UK, 1998. Translated by Hervé Brönnimann. 5

[Cam09] Daniel Duque Campayo. Talk: Voronoi fluid particle dynamics. http://www-sop.inria.fr/geometrica/collaborations/OrbiCG/program.html, 2009. 2

[CdCL+10] Manuel Caroli, Pedro M. M. de Castro, Sébastien Loriot, Olivier Rouiller, Monique Teillaud, and Camille Wormser. Robust and efficient Delaunay triangulations of points on or close to a sphere. In *9th International Symposium on Experimental Algorithms*, volume 6049 of *Lecture Notes in Computer Science*, pages 462–473, 2010. 86

[CDL07] Siu-Wing Cheng, Tamal K. Dey, and Joshua A. Levine. A practical Delaunay meshing algorithm for a large class of domains. In *Proceedings of the 16th International Meshing Roundtable*, pages 477–494, 2007. 18

[CFT10] Manuel Caroli, Vissarion Fisikopoulos, and Monique Teillaud. Meshing of triply-periodic surfaces in CGAL, 2010. Poster presentation, 7th International Conference on Curves and Surfaces. 72

[cga] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org. 9, 18, 39

[Che93] L. Paul Chew. Guaranteed-quality mesh generation for curved surfaces. In *Proceedings of the 9th Annual Symposium on Computational Geometry*, pages 274–280, 1993. 18

[CKT08] Manuel Caroli, Nico Kruithof, and Monique Teillaud. Decoupling the CGAL 3D triangulations from the underlying space. In *Workshop on Algorithm Engineering and Experiments*, pages 101–108, 2008. 61

[CM57] H. S. M. Coxeter and W. O. J. Moser. *Generators and Relations for Discrete Groups*. Springer-Verlag, Berlin, Heidelberg, New York, Tokyo, 1957. 16, 90

[Cox91] H. S. M. Coxeter. *Regular Complex Polytopes*. Cambridge University Press, Cambridge, England, 2nd edition, 1991. 86

[CS88] Kenneth L. Clarkson and Peter W. Shor. Algorithms for diametral pairs and convex hulls that are optimal, randomized, and incremental. In *Proceedings of the 4th Annual Symposium on Computational Geometry*, pages 12–17, 1988. 8

[CT08] Manuel Caroli and Monique Teillaud. Video: On the computation of 3D periodic triangulations. In *Proceedings of the 24th Annual Symposium on Computational Geometry*, pages 222–223, 2008. 18

[dBvKOS00] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2nd edition, 2000. 5, 25

[dC92] Manfredo Perdigão do Carmo. *Riemannian Geometry*. Birkhäuser, Boston, 1992. 13

[DE95] Cecil Jose A. Delfinado and Herbert Edelsbrunner. An incremental algorithm for Betti numbers of simplicial complexes on the 3-sphere. *Computer Aided Geometric Design*, 12:771–784, 1995. 72

[DEJ06] Qiang Du, Maria Emelianenko, and Lili Ju. Convergence of the Lloyd algorithm for computing centroidal Voronoi tessellations. *SIAM Journal on Numerical Analysis*, 44:102–119, 2006. 76

[Del34] Boris Delaunay. Sur la sphère vide. *Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskih i Estestvennyh Nauk*, 7:793–800, 1934. 5

[Del10] Christophe Delage. Spatial sorting. In *CGAL User and Reference Manual*. CGAL Editorial Board, 3.6 edition, 2010. http://www.cgal.org/Manual/latest/doc_html/cgal_manual/packages.html#Pkg:SpatialSorting. 10

[Dev02] Olivier Devillers. The Delaunay hierarchy. *International Journal of Foundations of Computer Science*, 13:163–180, 2002. 8, 9, 34, 35, 60

[dFC03] Gianni de Fabritiis and Peter V. Coveney. Dynamical geometry for multiscale dissipative particle dynamics. http://xxx.lanl.gov/abs/cond-mat/0301378v1, 2003. 2

[DH97a] Olaf Delgado Friedrichs and Daniel H. Huson. Orbifold triangulations and crystallographic groups. *Periodica Mathematica Hungaria*, 34:29–55, 1997. 18

[DH97b] Nikolai P. Dolbilin and Daniel H. Huson. Periodic Delone tilings. *Periodica Mathematica Hungarica*, 34:1-2:57–64, 1997. 16, 25, 26, 27, 93

[DMT92a] O. Devillers, S. Meiser, and M. Teillaud. Fully dynamic Delaunay triangulation in logarithmic expected time per operation. *Computational Geometry: Theory and Applications*, 2(2):55–80, 1992. 8

[DMT92b] Olivier Devillers, Stefan Meiser, and Monique Teillaud. The space of spheres, a geometric tool to unify duality results on Voronoi diagrams. In *Proceedings of the 4th Canadian Conference on Computational Geometry*, pages 263–268, 1992. Full version available as INRIA Research Report 1620, http://hal.inria.fr/inria-00074941. 89, 90

[DP03] Olivier Devillers and Sylvain Pion. Efficient exact geometric predicates for Delaunay triangulations. In *Proceedings of the 5th Workshop on Algorithm Engineering and Experiments*, pages 37–44, 2003. 41

[DPT02] Olivier Devillers, Sylvain Pion, and Monique Teillaud. Walking in a triangulation. *International Journal of Foundations of Computer Science*, 13:181–199, 2002. 9

[DS02] Robert J. Daverman and Richard B. Sher, editors. *Handbook of Geometric Topology*. Elsevier, Amsterdam, London, Paris, 2002. 14

[DT03] Olivier Devillers and Monique Teillaud. Perturbations and vertex removal in a 3D Delaunay triangulation. In *Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms*, pages 313–319, 2003. 5, 22

[DT06a] Olivier Devillers and Monique Teillaud. Perturbations and vertex removal in Delaunay and regular 3d triangulations. Research Report 5968, INRIA, 2006. 33

[DT06b] Olivier Devillers and Monique Teillaud. Perturbations and vertex removal in Delaunay and regular 3d triangulations. *(To appear in CGTA)*. Research Report 5968, INRIA, 2006. 6

[Dul08] Cornelis Dullemond. Talk: Radiative transfer using unstructured grids. http://www.cgal.org/Events/PeriodicSpacesWorkshop/, 2008. 2

[Duq] Daniel Duque Campayo. Sklogwiki - Boundary conditions. http://www.sklogwiki.org/SklogWiki/index.php/Boundary_conditions. 2

[Dur08] Orencio Duran. Talk: Multi-scale mechanics of granular materials. http://www.cgal.org/Events/PeriodicSpacesWorkshop/, 2008. 2

[dVT] Eric Colin de Verdière and Monique Teillaud. Efficient representations of triangulations of arbitrary dimension. Personal communication. 98

[Dwy87] Rex A. Dwyer. A faster divide-and-conquer algorithm for constructing Delaunay triangulations. *Algorithmica*, 2:137–151, 1987. 9

[DY10] Tran Kai Frank Da and Mariette Yvinec. 3D alpha shapes. In *CGAL User and Reference Manual*. CGAL Editorial Board, 3.6 edition, 2010. http://www.cgal.org/Manual/latest/doc_html/cgal_manual/packages.html#Pkg:AlphaShapes3. 18, 71

[EKS83] Herbert Edelsbrunner, David G. Kirkpatrick, and Raimund Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, IT-29:551–559, 1983. 71

[EM94] Herbert Edelsbrunner and Ernst P. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, January 1994. 71

[ES96] Herbert Edelsbrunner and Nimish R. Shah. Incremental topological flipping works for regular triangulations. *Algorithmica*, 15:223–241, 1996. 6, 10

[ES97] Herbert Edelsbrunner and Nimish R. Shah. Triangulating topological spaces. *International Journal of Computational Geometry and Applications*, 7:365–378, 1997. 17

[For87] Steven J. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987. 7, 9

[FP06] Andreas Fabri and Sylvain Pion. A generic lazy evaluation scheme for exact geometric computations. In *Proceedings of the 2nd Library-Centric Software Design*, 2006. 41

[FT06] Efi Fogel and Monique Teillaud. Generic programming and the CGAL library. In Jean-Daniel Boissonnat and Monique Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*. Springer-Verlag, Mathematics and Visualization, 2006. 39

[FV96] Steven J. Fortune and Christopher J. Van Wyk. Static analysis yields efficient exact integer arithmetic for computational geometry. *ACM Transactions on Graphics*, 15(3):223–248, July 1996. 41

[FW09] Xi Fu and Xiantao Wang. Isometries and discrete isometry subgroups of hyperbolic spaces. *Glasgow Mathematical Journal*, 51:31–38, 2009. 90

[Gab09] Ruggero Gabbrielli. A new counter-example to Kelvin's conjecture on minimal surfaces. *Philosophical Magazine Letters*, 89:483–491, 2009. 2

[gap] Gap - groups, algorithms, programming - a system for computational discrete algebra. http://www.gap-system.org. 83

[GGL95] Ronald L. Graham, Martin Grötschel, and László Lovász, editors. *Handbook of Combinatorics*. Elsevier, Amsterdam, Lausanne, New York, 1995. 14

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley, Reading, MA, 1995. 62

[GKS92] Leonidas J. Guibas, Donald E. Knuth, and Micha Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7:381–413, 1992. 8

[GM01] Clara I. Grima and Alberto Márquez. *Computational Geometry on Surfaces*. Kluwer Academic Publishers, 2001. 15

[Gol91] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, March 1991. 40

[Har77] Robin Hartshorne. *Algebraic Geometry*, volume 52 of *Graduate texts in Mathematics*. Springer-Verlag, New York, 1977. 92

[Hat01] Allan Hatcher. *Algebraic Topology*. Cambridge University Press, 2001. 14, 97

[Hen79] Michael Henle. *A Combinatorial Introduction to Topology*. W. H. Freeman, San Francisco, CA, 1979. 14, 59

[HW35] W. Hantzsche and H. Wendt. Dreidimensionale euklidische Raumformen. *Mathematische Annalen*, 110:593–611, 1935. 81

[IEE85] *IEEE Standard for binary floating point arithmetic, ANSI/IEEE Std* 754 – 1985. New York, NY, 1985. Reprinted in SIGPLAN Notices, 22(2):9–25, 1987. 40

[IEE08] IEEE Task P754. *IEEE 754-2008, Standard for Floating-Point Arithmetic*. IEEE, New York, NY, USA, August 2008. 40

[Kat92] Svetlana Katok. *Fuchsian Groups*. The University of Chicago Press, Chicago and London, 1992. 90

[Kee02] William C. Keel. *The Road to Galaxy Formation*. Springer Praxis Publishing, Chichester, UK, 2002. 2

[Kep] Andrew Kepert. Bitruncated cubic honeycomb. `http://en.wikipedia.org/wiki/File:Truncated_octahedra.jpg`. 3

[KL84] Wolfgang Kühnel and Gunter Lassmann. The rhombidodecahedral tessellation of 3-space and a particular 15-vertex triangulation of the 3-dimensional torus. *manuscripta mathematica*, 49:61–77, 1984. 31

[KMP+04] Lutz Kettner, Kurt Mehlhorn, Sylvain Pion, Stefan Schirra, and Chee Yap. Classroom examples of robustness problems in geometric computations. In *Proceedings of the 12th European Symposium on Algorithms*, volume 3221 of *Lecture Notes in Computer Science*, pages 702–713. Springer-Verlag, 2004. 39

[Kru09] Niels P. Kruyt. Talk: Tessellations and granular materials. http://www-sop.inria.fr/geometrica/collaborations/OrbiCG/program.html, 2009. 2

[Lee00] John M. Lee. *Introduction to Topological Manifolds*. Springer-Verlag, New York, 2000. 3, 14

[Llo82] Stuart P. Lloyd. Least square quantization in PCM. *IEEE Transactions on Information Theory*, 28:129–137, 1982. 76

[LS80] D. T. Lee and B. J. Schachter. Two algorithms for constructing Delaunay triangulations. *International Journal of Computer and Information Sciences*, 9(3):219–242, 1980. 7, 9

[Lyn66] Roger C. Lyndon. On Dehn's algorithm. *Mathematische Annalen*, 166:208–228, 1966. 93

[Mil70] R. E. Miles. On the homogenous planar Poisson point-process. *Mathematical Biosciences*, 6:85–127, 1970. 36

[Möb86] August F. Möbius. Mittheilungen aus Möbius' Nachlass: I. Zur Theorie der Polyëder und der Elementarverwandtschaft. *Gesammelte Werke II*, pages 515–559, 1886. 31

[Moe08] Maarten Moesen. Talk: Periodicity and the design of bone scaffolds. http://www.cgal.org/Events/PeriodicSpacesWorkshop/, 2008. 2

[MR97] Marisa Mazón and Tomás Recio. Voronoi diagrams on orbifolds. *Computational Geometry: Theory and Applications*, 8:219–230, 1997. 16

[NG02] Mikael Nygårds and Peter Gudmundson. Three-dimensional periodic Voronoi grain models and micromechanical FE-simulations of a two-phase steel. *Computational Materials Science*, 24:513–519, 2002. 2

[NLC02] Hyeon-Suk Na, Chung-Nim Lee, and Otfried Cheong. Voronoi diagrams on the sphere. *Computational Geometry: Theory and Applications*, 23:183–194, 2002. 86

[NN09] Frank Nielsen and Richard Nock. Hyperbolic Voronoi diagrams made easy. *Computing Research Repository (CoRR)*, abs/0903.3287, 2009. 90

[pdh] Poincare disc hyperbolic parallel lines. http://en.wikipedia.org/wiki/File:Poincare_disc_hyperbolic_parallel_lines.svg. 88

[PT10a] Sylvain Pion and Monique Teillaud. 3D triangulation data structure. In *CGAL User and Reference Manual*. CGAL Editorial Board, 3.6 edition, 2010. http://www.cgal.org/Manual/latest/doc_html/cgal_manual/packages.html#Pkg:TDS3. 14, 41, 43

[PT10b] Sylvain Pion and Monique Teillaud. 3D triangulations. In *CGAL User and Reference Manual*. CGAL Editorial Board, 3.6 edition, 2010. http://www.cgal.org/Manual/latest/doc_html/cgal_manual/packages.html#Pkg:Triangulation3. 9, 18, 41

[Rob06] Vanessa Robins. Betti number signatures of homogeneous Poisson point processes. *Physical Review E*, 74(061107), 2006. 2

[RTY10] Laurent Rineau, Stéphane Tayeb, and Mariette Yvinec. 3D mesh generation. In *CGAL User and Reference Manual*. CGAL Editorial Board, 3.6 edition, 2010. http://www.cgal.org/Manual/latest/doc_html/cgal_manual/packages.html#Pkg:Mesh_3. 18, 71

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

[RV06] Günter Rote and Gert Vegter. Computational topology: An introduction. In Jean-Daniel Boissonnat and Monique Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*. Springer-Verlag, Mathematics and Visualization, 2006. 3

[RY07] Laurent Rineau and Mariette Yvinec. Meshing 3D domains bounded by piecewise smooth surfaces. In *Proceedings of the 16th International Meshing Roundtable*, pages 443–460, 2007. 14, 18

[RY10] Laurent Rineau and Mariette Yvinec. 3D surface mesh generation. In *CGAL User and Reference Manual*. CGAL Editorial Board, 3.6 edition, 2010. http://www.cgal.org/Manual/latest/doc_html/cgal_manual/packages.html#Pkg:SurfaceMesher3. 14, 18, 71, 72

[San76] Luis A. Santaló. *Integral Geometry and Geometric Probability*. Addison-Wesley, Reading, MA, 1976. 36

[Sch70] Alan Schoen. Infinite periodic minimal surfaces without self-intersection. Technical Note D-5541, NASA, 1970. 75

[She98a] Jonathan R. Shewchuk. A condition guaranteeing the existence of higher-dimensional constrained Delaunay triangulations. In *Proceedings of the 14th Annual Symposium on Computational Geometry*, pages 76–85, 1998. 11

[She98b] Jonathan R. Shewchuk. Tetrahedral mesh generation by Delaunay refinement. In *Proceedings of the 14th Annual Symposium on Computational Geometry*, pages 86–95, 1998. 18

[She00] Jonathan R. Shewchuk. Sweep algorithms for constructing higher-dimensional constrained Delaunay triangulations. In *Proceedings of the 16th Annual Symposium on Computational Geometry*, pages 350–359, 2000. 18

[Sim94] Charles C. Sims. *Computing with Finitely Presented Groups*. Cambridge University Press, Cambridge, UK, 1994. 83

[Slo] Neil J. A. Sloane. The on-line encyclopedia of integer sequences. http://www.research.att.com/~njas/sequences/. 81

[Sou10] Thierry Sousbie. The persistent cosmic web and its filament structure i: Theory and implementation. arXiv:1009.4015, 2010. 18

[Spa66] Edwin H. Spanier. *Algebraic Topology*. McGraw-Hill Book Company, New York, 1966. 14

[Spr09] Volker Springel. Talk: Computational fluid dynamics and Voronoi tessellations. http://www-sop.inria.fr/geometrica/collaborations/OrbiCG/program.html, 2009. 17

[Spr10] Volker Springel. E pur si muove: Galilean-invariant cosmological hydrodynamical simulations on a moving mesh. *Monthly Notices of the Royal Astronomical Society (MNRAS)*, 401:791–851, 2010. 17

[Sti92] John Stillwell. *Geometry of Surfaces*. Springer-Verlag, New York, 1992. 86, 88, 89

[Sul] John M. Sullivan. vcs. http://torus.math.uiuc.edu/jms/software/. 17

[Tho87] William Thomson. On the division of space with minimum partitional area. *Philosophical Magazine*, 24:503, 1887. 2

[Tho02] Karsten E. Thompson. Fast and robust Delaunay tessellation in periodic domains. *International Journal for Numerical Methods in Engineering*, 55:1345–1366, 2002. 17

[Thr32] William Threlfall. Gruppenbilder. *Abhandlungen der Sächsischen Akademie der Wissenschaften, Mathematisch-Physikalische Klasse*, 41:1–59, 1932. 90

[Thu97] William Thurston. *Three-Dimensional Geometry and Topology, Volume 1*. Princeton University Press, New Jersey, 1997. 19, 80, 81

[Thu02] William P. Thurston. *The Geometry and Topology of Three-Manifolds*. http://www.msri.org/publications/books/gt3m/, 2002. 13, 81

[Tou09] Jane Tournois. *Optimisation de maillages*. Thèse de doctorat en sciences, Université de Nice Sophia-Antipolis, Nice, France, 2009. 76

[Val] Sébastien Valette. Personal communcation. http://www.creatis.insa-lyon.fr/~valette/. 77

[vdWS07] Rien van de Weijgaert and Willem Schaap. The cosmic web: Geometric analysis. http://www.astro.rug.nl/~weygaert/tim1publication/weyval2004.pdf, 2007. 2

[vdWVP+10] Rien van de Weijgaert, Gert Vegter, Erwin Platen, Bob Eldering, and Nico Kruithof. Alpha shape topology of the cosmic web. arXiV:1006.2765, 2010. 2, 71

[VY90] Gert Vegter and Chee K. Yap. Computational complexity of combinatorial surfaces. In *Proceedings of the 6th Annual Symposium on Computational Geometry*, pages 102–111, 1990. 90

[Wat81] David F. Watson. Computing the n-dimensional Delaunay tesselation with applications to Voronoi polytopes. *The Computer Journal*, 24(2):167–172, 1981. 7, 8, 9, 10, 18, 85

[Wei08] Dahlia Weiss. Talk: How hydrophobic Buckminsterfullerene affects surrounding water structure. INRIA Geometrica Seminar, http://www-sop.inria.fr/geometrica, March 2008. 2

[Wil08] P M. H. Wilson. *Curved Spaces*. Cambridge University Press, Cambridge, 2008. 20

[Yvi10] Mariette Yvinec. 2D triangulations. In *CGAL User and Reference Manual*. CGAL Editorial Board, 3.6 edition, 2010. http://www.cgal.org/Manual/latest/doc_html/cgal_manual/packages.html#Pkg:Triangulation2. 9

[Zom05] Afra Zomorodian. *Topology for Computing*. Cambridge University Press, Cambridge, 2005. 3, 14