UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Computing an ε-net of a hyperbolic surface

THÈSE

présentée et soutenue publiquement le 26 novembre 2025

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Camille Lanuel

Composition du jury

Président :	Pierrick Gaudry
Rapporteurs :	Guillaume Damiand (CNRS, LIRIS) Lionel Pournin (Université Paris 13, LIPN)
Examinatrices et examinateur :	Myfanwy Evans (Universität Potsdam) Pierrick Gaudry (CNRS, LORIA) Jane Tournois (GeometryFactory)
Invitée :	Monique Teillaud (INRIA, LORIA)
Encadrants :	Vincent Despré (Université de Lorraine, LORIA) Marc Pouget (INRIA, LORIA)

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

Résumé

Les surfaces hyperboliques apparaissent naturellement en mathématiques et font donc l'objet d'études approfondies. Cependant, de nombreuses questions n'ont été résolues que pour des surfaces spécifiques. La mise en œuvre d'algorithmes d'approximation sur les surfaces hyperboliques faciliterait alors l'étude de surfaces plus génériques. La première étape d'un tel algorithme d'approximation est d'approcher la géométrie de la surface avec un ensemble de points bien répartis sur la surface, afin que chaque point de la surface soit proche d'un point de l'ensemble. La notion d'ε-filet répond à cette description.

Dans cette thèse, nous concevons un algorithme pour calculer un ε-filet d'une surface hyperbolique compacte et sans bord. Notre algorithme utilise la technique du raffinement de Delaunay : à partir d'une triangulation de Delaunay à un seul sommet sur la surface, l'algorithme insère itérativement les centres circonscrits des triangles dont le rayon du cercle circonscrit est strictement supérieur au paramètre ε.

Le nombre de points dans un ε-filet d'une surface hyperbolique dépend à la fois de son aire et de sa systole. En particulier, plus la systole est petite, plus le nombre de points est grand. Afin de rompre cette dépendance à la systole sans perdre d'information géométrique importante, nous étendons notre algorithme au calcul de ce que nous appelons un pseudo ε-filet.

Nous détaillons l'implémentation de l'algorithme du calcul d'un ε-filet avec la bibliothèque CGAL. Cette bibliothèque nous permet de faire des calculs robustes et donc de pouvoir utiliser l'ε-filet obtenu comme entrée pour de futurs algorithmes. Nous utilisons des nombres rationnels pour les coordonnées des points afin de garantir la robustesse des calculs tout en restant efficaces. Cela nous contraint cependant à arrondir en rationnels les coordonnées algébriques des points insérés. Nos expériences montrent que la complexité temporelle de l'algorithme d'ε-filet est, en pratique, sous-linéaire par rapport au nombre de points, alors que la complexité dans le pire des cas est quadratique.

Mots-clés: Géométrie hyperbolique, triangulation de Delaunay, géométrie algorithmique.

Abstract

Hyperbolic surfaces naturally appear in mathematics and are therefore intensively studied. However, many questions have been answered only for specific surfaces. The implementation of approximation algorithms on hyperbolic surfaces would then facilitate the study of more generic surfaces. The first step in such an approximation algorithm is to approximate the surface geometry with a set of well-distributed points on the surface, ensuring that every point on the surface is close to a point in the set. The notion of ε-net meets this description.

In this thesis, we design an algorithm to compute an ε-net of a compact hyperbolic surface without boundary. Our algorithm is based on the Delaunay refinement technique: starting from a Delaunay triangulation with a single vertex on the surface, the algorithm iteratively inserts the circumcenters of triangles whose circumcradius is greater than the parameter ε.

The number of points in an ε-net of a hyperbolic surface depends on both its area and its systole. In particular, the smaller the systole, the greater the number of points. To break this dependency on the systole without losing important geometric information, we extend our algorithm to compute what we call pseudo ε-net.

We detail the implementation of the ε-net computation algorithm using the CGAL library. This library allows us to perform robust computations and thus use the resulting ε-mesh as input for future algorithms. We use rational numbers for the coordinates of the points in order to guarantee the robustness of the computations while remaining efficient. However, this forces us to round the algebraic coordinates of the inserted points to rational numbers. Our experiments show that the time-complexity of the ε-net algorithm is, in practice, sub-linear in the number of points, while the worst-case complexity is quadratic.

Keywords: Hyperbolic geometry, Delaunay triangulation, computational geometry.

Remerciements

Je suis arrivée en thèse sans savoir ce qu'est une surface hyperbolique ni coder. Maintenant, je sais. C'est ainsi que je me rends compte de tout le chemin parcouru. Je tiens donc tout d'abord à remercier Monique Teillaud, Vincent Despré et Marc Pouget, qui a repris la direction de ma thèse après le départ à la retraite de Monique. J'ai énormément appris à vos côtés. Merci pour votre confiance, votre patience et vos précieux conseils.

Je remercie également les rapporteurs, Guillaume Damiand et Lionel Pournin d'avoir pris le temps de relire et d'évaluer mon manuscrit, ainsi que les autres membres du jury, Pierrick Gaudry, Jane Tournois et Myfwany Evans, d'avoir accepté d'être examinateurs.

Merci à tous les membres, passés et présents, de l'équipe Gamble. C'était un plaisir de vous avoir comme collègues, mais aussi comme amis pour certains. Merci pour les conversations enrichissantes, tant sur le plan scientifique que personnel, que nous avons eues autour d'un bon repas à la cantine. À propos de la cantine, merci à toute l'équipe de restauration du Loria de m'avoir si bien nourrie pendant ces trois années. (C'est important pour le moral !)

Je pense également à tous les doctorants et postdoctorants du Loria avec qui j'ai parcouru ce chemin. Je pense notamment à Antoine, Yoann, Bastien, Amaury, Mehdi, les Léo (mention spéciale à Léo Valque pour son aide précieuse avec CGAL), Nuwan, Florent, Sarah, Dorian, Marguerite, Niloufar, Gautier, etc. Je souhaite une bonne fin de thèse à celles et ceux d'entre vous qui ne sont pas encore docteurs. Profitez bien du temps qu'il vous reste au Loria, mais méfiez-vous, car il passe plus vite qu'on ne le croit !

Merci Romain pour ton soutien inconditionnel.

Enfin, je remercie chaleureusement ma famille, en particulier Yann, Isabelle, Charlotte et Adrien, de m'avoir encouragé et soutenue tout au long de mon parcours scolaire.

Contents

Introduction

The terminology used in this chapter is explained in Chapter I.

1 Motivation

Computational geometry is a field of computer science that emerged in the late 1970s [dBCvKO08]. It focuses on designing data structures and algorithms to solve geometric problems. In computational geometry algorithms, each step is either a construction or a conditional step based on the evaluation of a predicate. A construction is the computation of a new object, like the intersection point of two lines or the circumcenter of a triangle. A predicate is the evaluation of a binary question such as "are these two segments intersecting?" or "are these three points collinear, oriented clockwise or counterclockwise?", which generally reduces to computing the sign of an algebraic expression.

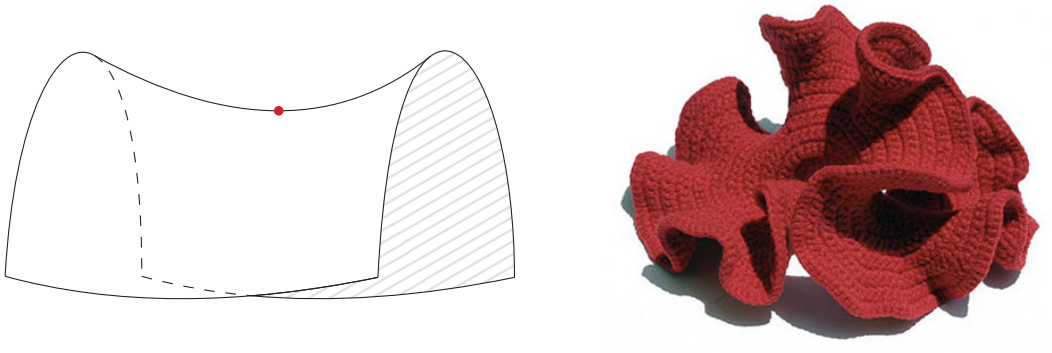

Figure 1: Left: A saddle point (red). Right: A crochet model of the hyperbolic plane, by Diana Taimina (https://theiff.org/oexhibits/oe1e.html).

The study of *hyperbolic geometry* in computational geometry is more recent. It is the geometry of objects with negative curvature, meaning that every point is a saddle point (Figure 1). Hyperbolic geometry was discovered at the end of the 18th century by Gauss, though he never published his work. A few years later, Lobachevsky and Bolyai independently rediscovered it [Rat19, Ch 1]. Many famous mathematicians studied hyperbolic geometry, like Riemann and Poincaré. The latter gave its name to the Poincaré disk (Figure 2), which is the open unit disk of $\mathbb{C}$ equipped with a metric that makes the disk hyperbolic. In the Poincaré disk, lines are circular arcs or diameters of the unit disk. Constructions and predicates are thus algebraically more complicated to compute in the Poincaré disk than in the Euclidean plane. This is a major challenge when it comes to implementing algorithms.

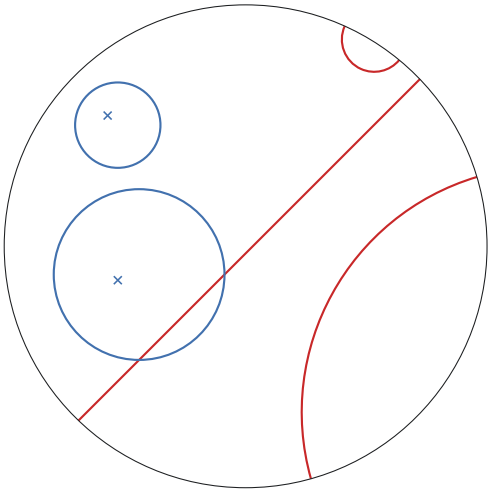

Figure 2: Lines (red) and hyperbolic circles (blue) in the Poincaré disk.

Hyperbolic geometry is unavoidable when studying surfaces (two-dimensional objects) thanks to the *uniformization theorem* [Abi81]. This theorem states that any surface can be cut to be mapped via a one-to-one correspondence to a part of either the sphere, the Euclidean plane or the hyperbolic plane. For example, for a cylinder and the Möbius strip, this is the Euclidean plane, and for the projective plane, it is the sphere. The third case is the most common: any surface of genus (number of "doughnut holes") greater than one can be cut in order to be mapped to a piece of the hyperbolic plane. In other words, almost all surfaces are *hyperbolic*.

A wide variety of problems have been studied in Euclidean computational geometry, from convex hulls to mesh generation. Many of these problems rely on *Delaunay triangulations*. A Delaunay triangulation is a set of triangles such that the circumcircle of each triangle does not contain any vertex of the triangulation in its interior. With such a simple definition, Delaunay triangulations are a key tool in computational geometry. Naturally, they were one of the first problems studied in hyperbolic geometry and on hyperbolic surfaces [BDT14, BTV16, IT17].

The Bolza surface is a hyperbolic surface of genus 2 that can be cut into a regular hyperbolic octagon. It is the most studied hyperbolic surface thanks to its symmetry that eases computations. Generalized Bolza surfaces, which are surfaces of higher genus that can be cut into regular polygons, are also quite well studied. However, many questions that have been answered for the Bolza surface or generalized Bolza surfaces remain open for generic hyperbolic surfaces, for instance finding their diameter [SBPK23] or the length of the shortest non-contractible curve [Ior19, Thm II.17].

To help fill this gap, the aim of this thesis is to move towards the design and implementation of approximation algorithms on hyperbolic surfaces. Such algorithms would be valuable tools to better understand hyperbolic surfaces and explore conjectures. To design an approximation algorithm, the first step is to find a way to approximate the geometry of the surface with a finite set of points that are well-distributed on the surface. This way, any point of the surface is close to a point of this set. In addition to being an input for approximation algorithms, such a set of points could be used in other algorithms. For example, Bowyer's algorithm, an incremental Delaunay triangulation algorithm, has been adapted to the Bolza surface but it requires computing a set dummy points first. This approach could be generalized to any hyperbolic surface by using an ε-net with a well-chosen ε as the set of dummy points.

Computational geometry is useful in many areas such as computer graphics, robotics, mesh generation or computer vision. However, most of the work in this field is theoretical and not many algorithms have been implemented. The CGAL library, the biggest open-source computational geometry library, aims to bridge the gap between theory and practice. It gathers computational geometry data structures and algorithms in a common software library to provide reliable and efficient programs for academic or industrial users. In order to participate in the practical aspects of computational geometry, it was essential for us to implement the algorithm developed during this thesis and integrate it into CGAL.

2 Problem statement

The notion of ε-net [Cla06] corresponds to the desired set of well-distributed points mentioned above. For a parameter $\varepsilon > 0$, an ε-net P of a surface is a subset such that any point of the surface is at distance at most ε from P and two distinct points of P are at least ε apart. The first property ensures that the whole surface is covered, so any point of the surface can be approximated by a point of P. The second property ensures that P is not unnecessarily dense.

We therefore searched for an algorithm to compute ε-nets on hyperbolic surfaces. Since the goal is to use ε-nets as a basis for approximation algorithms intended for practical use, we also aimed to implement the algorithm. The implementation should handle exact computation to be able to reuse the computed ε-net as an input for an approximation algorithm. The CGAL library is thus the perfect tool for the implementation as it is designed to handle exact computations and already integrates some hyperbolic geometry. Moreover, we looked for a way to handle surfaces that have (arbitrarily) long hyperbolic cylinders, leading to an (arbitrarily) large number of points in an ε-net.

3 Contribution

In this thesis, we design and implement an algorithm to compute an ε-net of a compact hyperbolic surface without boundary. Our algorithm uses the Delaunay refinement technique: starting from a Delaunay triangulation with a single vertex on the surface, the algorithm iteratively inserts the circumcenters of triangles whose circumradius is greater than the parameter ε.

Locating a point in a Delaunay triangulation is a crucial step of our algorithm. In Chapter II, we present two classical point location algorithms: the straight walk, which always terminates, and the visibility walk, which we show terminates in our setting (Chapter II).

In Chapter III, we establish bounds on the number of points in an ε-net and describe the ε-net algorithm using two distinct data structures. The first data structure is simpler and facilitates a clear explanation of the algorithm. The second one is more complex but better suited for the implementation of the ε-net algorithm as well as future approximation algorithms. The number of points in an ε-net of a hyperbolic surface depends on its area and on its longest hyperbolic cylinder: the longer the hyperbolic cylinder, the greater the number of points. To break this dependency without losing important geometric information, we extend our algorithm to compute what we call pseudo ε-net.

In Chapter IV, we present an implementation of the ε-net algorithm using the CGAL library (Figure 3). We discuss how our code utilizes the CGAL package for triangulations on hyperbolic surfaces that already exists. We detail each component of the algorithm. In particular, we use rational coordinates for the points. We performed experiments to help us design the implementation and to analyze the behavior of

the algorithm in practice. Our experiments are performed on a large number of surfaces of genus 2 and a limited number of surfaces of greater genus.

The ε-net algorithm with the first data structure (Chapter III) has been presented at the EuroCG workshop in 2024 [DLT24]. Its adaptation to the second data structure, its implementation and the experiments (Chapter III and IV) have been published and presented at the European Symposium on Algorithms in 2025 [DLPT25].

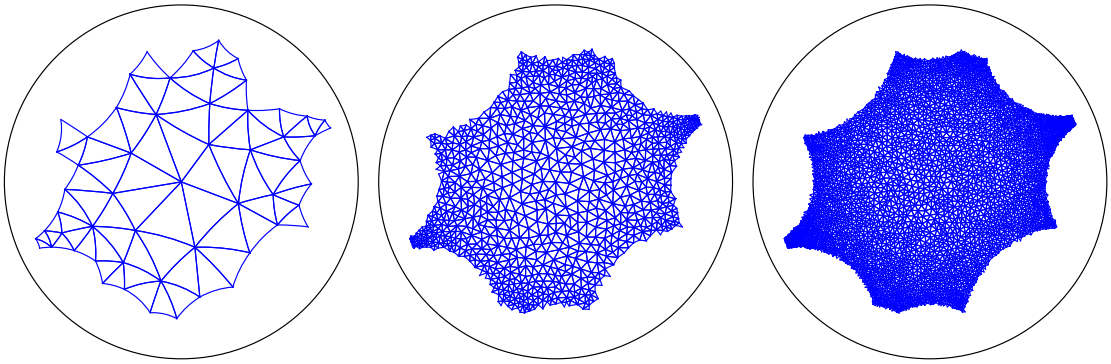

Figure 3: Delaunay triangulations of the computed ε-nets for $\varepsilon = 0.5$ (left), $\varepsilon = 0.1$ (middle), $\varepsilon = 0.05$ (right) of a hyperbolic surface (figure from Section IV.5).

Chapter I

Background

This chapter introduces the foundational concepts used throughout this thesis, starting with the mathematical foundations and ending with computer science aspects. We begin by looking at hyperbolic geometry and hyperbolic surfaces, our object of study. Next, we cover Delaunay triangulations, which are central to our algorithm, and Voronoi diagrams, which also play a crucial role. We conclude the mathematical discussion with ε-nets, the sets of points that we aim to construct on hyperbolic surfaces. On the computer science side, we present two data structures that we use to represent Delaunay triangulations on hyperbolic surfaces. This chapter ends with a presentation of CGAL, the software library that we use to implement our algorithm.

I.1 Hyperbolic geometry

This section introduces hyperbolic surfaces, which are the subject of this thesis. We begin by introducing the hyperbolic plane with an axiomatic approach. We continue with the definition of a hyperbolic surface, seen both as a topological object and as an algebraic object. Finally, we detail two important aspects of hyperbolic surfaces that we will encounter throughout this thesis: fundamental domains and small simple closed geodesics.

I.1.1 Hyperbolic plane

Euclidean geometry is the geometry that is taught in school. Its name comes from Euclid, an ancient Greek mathematician, who described the foundations of this geometry in the *Elements* [EP04]. This geometry relies on five *postulates*, which are statements that are assumed without proof. They are a starting point from which *theorems* are deduced. The five postulates are the following:

1. There is one and only one line segment between any two given points;
2. Any line segment can be extended continuously to a line;
3. There is one and only one circle with any given center and any given radius;
4. All right angles are congruent to one another;

5. (Parallel postulate) Given a line and a point not on the line, there is *exactly one* line through the point that is parallel to the given line.

Let us focus on the parallel postulate (Figure I.1, left). Two lines are *parallel* if they do not intersect. Euclid's parallel postulate was criticized for centuries. Mathematicians believed that it could be derived from the other four and attempted to prove it. At the end of the 18th century, Gauss discovered a new geometry that he called *non-Euclidean geometry* but he never published his work. A few years later, Lobachevsky and Bolyai independently rediscovered Gauss' non-Euclidean geometry [Rat19, Ch 1]. This geometry is what is today called *hyperbolic geometry*.

Hyperbolic geometry is also based on five postulates. The first four are the same as in Euclidean geometry. Only the parallel postulate differs: *given a line and a point not on the line, there are* ***infinitely many*** *lines through the point that are parallel to the given line.* This postulate is difficult to grasp at first. It is illustrated in Figure I.1 (right), but to understand this illustration we first need to introduce the Poincaré disk model of the hyperbolic plane.

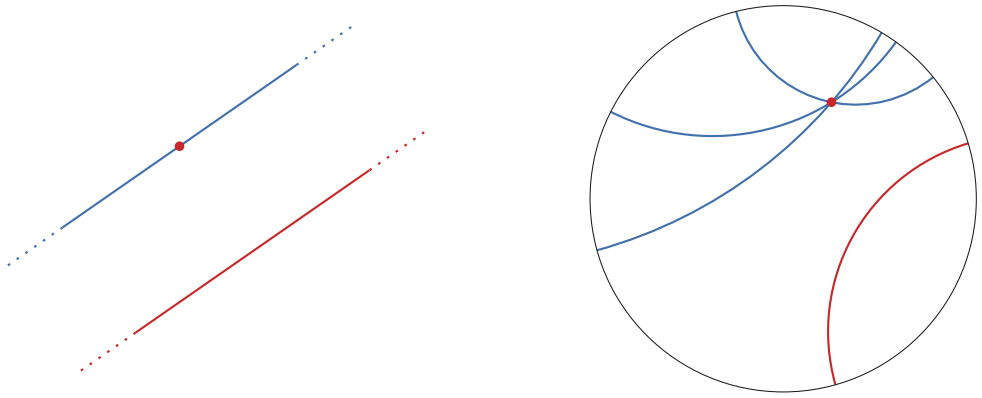

Figure I.1: Illustration of the parallel postulate in Euclidean geometry (left) and hyperbolic geometry (right).

The Poincaré disk

A *metric* is a structure on a manifold that enables the measurement of lengths and angles. A *model of the hyperbolic plane* is a simply connected two-dimensional manifold equipped with a metric such that the five mentioned postulates are satisfied. Such a metric is called *hyperbolic*. There are several models of the hyperbolic plane [FK92, Ch 1], which are all isometric to each other, meaning that there is a distance-preserving map between any two models. The *hyperbolic plane* is, up to isometry, the unique simply connected two-dimensional manifold equipped with a hyperbolic metric [Bor16, Sec 2.1]. The Poincaré half-plane and the Poincaré disk are the most common models of the hyperbolic plane found in textbooks, such as those by Beardon [Bea83] or Buser [Bus10]. In this thesis, we will only work in the Poincaré disk, which will be denoted $\mathbb{H}^2$.

Definition I.1 (The Poincaré disk)**.** The *Poincaré disk* model $\mathbb{H}^2$ of the hyperbolic plane is the open unit disk of the complex plane $\mathbb{C}$, $\{z = x + iy \in \mathbb{C} : |z| = \sqrt{x^2 + y^2} < 1\}$, equipped with a hyperbolic metric.

The unit circle represents points at infinity. Objects then appear smaller when they are closer to the unit circle. Hyperbolic lines, also called *geodesic lines*, are either open diameters of the unit disk or circular

arcs orthogonal to the unit disk. Hyperbolic circles are Euclidean circles but their centers do not coincide in general. The Poincaré disk model is also *conformal*, meaning that hyperbolic angles are the Euclidean angles. These properties are illustrated in Figure I.2.

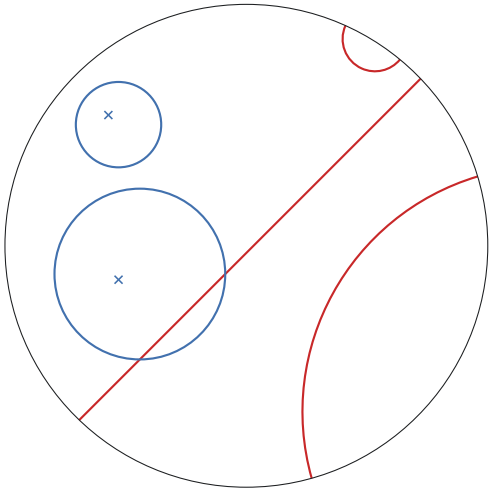

Figure I.2: Geodesic lines (red) and hyperbolic circles (blue) in the Poincaré disk.

Now that we are more familiar with $\mathbb{H}^2$, the illustration of the hyperbolic parallel postulate in Figure I.1 (right) becomes clear.

Some formulas in the hyperbolic plane

In this section, we present some classical formulas of plane hyperbolic geometry, starting with the distance formula.

Theorem I.2 (Distance in $\mathbb{H}^2$ [Rat19, Thm 4.5.1])**.** *The hyperbolic distance between two points $u, v \in \mathbb{H}^2$ is*

$$d_{\mathbb{H}^2}(u, v) = \operatorname{arcosh}\left(1 + \delta(u, v)\right),$$

where

$$\delta(u, v) = \frac{2|u - v|^2}{(1 - |u|^2)(1 - |v|^2)}.$$

Since each model has its own metric, the way to compute hyperbolic lengths and angles from the Euclidean ones can be different from one model to the other. The former formula is only true when working in the Poincaré disk model. However, the formulas involving only *hyperbolic* lengths or angles are independent of the model. In the following, all the lengths and angles are hyperbolic and the word "hyperbolic" will be omitted.

Theorem I.3 (Area & perimeter of a disk [Bea83, Thm 7.2.2])**.** *The area of a disk of radius r is $4\pi \sinh^2(r/2)$. Its perimeter is $2\pi \sinh(r)$.*

Remark. The area and the perimeter of a disk are thus exponential in its radius, while they are respectively quadratic and linear in Euclidean geometry. However, since $\sinh x = x + o(x)$ when $x \to 0$, the area and perimeter of a hyperbolic disk are respectively equivalent to the area and perimeter of a Euclidean circle when its radius tends to zero.

A *hyperbolic polygon* is defined, similarly as a Euclidean polygon, as a closed connected subset of $\mathbb{H}^2$ whose boundary is a Jordan curve made of geodesic segments. Hyperbolic polygons can be surprising for newcomers. The shapes of triangles can be counter-intuitive (Figure I.3) and hyperbolic rectangles do not exist, but right-angled pentagons or hexagons do. One of the key properties of hyperbolic geometry is that the sum of the interior angles of a hyperbolic triangle are less than π. This is a consequence of the following formula.

Theorem I.4 (Area of a triangle [Bea83, Thm 7.13.1]). *The area of a triangle T with interior angles α, β, γ is*

$$\mathcal{A}_{\mathbb{H}^2}(T) = \pi - (\alpha + \beta + \gamma).$$

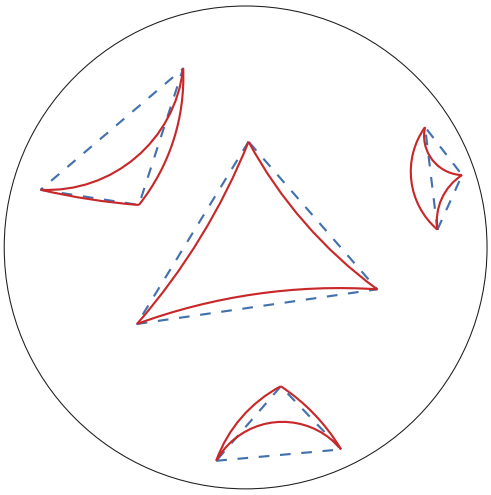

Figure I.3: Comparison between hyperbolic triangles (plain red) and Euclidean triangles (dashed blue).

Since any polygon can be decomposed into triangles, the formula for the area of any polygon follows.

Corollary I.5 (Area of a polygon [Bea83, Thm 7.15.1]). *The area of an n-sided polygon P with interior angles $\theta_1, \ldots, \theta_n$ is*

$$\mathcal{A}_{\mathbb{H}^2}(P) = (n-2)\pi - (\theta_1 + \ldots + \theta_n).$$

The trigonometry in a hyperbolic triangle is analogous to Euclidean trigonometry. All the trigonometry formulas for triangles and other polygons can be found in Buser [Bus10, Ch 2]. In this work, we need hyperbolic versions of Pythagoras' theorem and law of cosines (also known as Al-Kashi's formula). The notation of these two formulas is illustrated in Figure I.4.

Theorem I.6 (Hyperbolic law of cosines [Bus10, Thm 2.2.1(i)]). *In a triangle whose sides are a, b, c and the angle φ is opposite to c, we have*

$$\cosh(c) = \cosh(a)\cosh(b) - \sinh(a)\sinh(b)\cos(\varphi).$$

Hyperbolic Pythagoras' theorem follows from applying the law of cosine with $\varphi = \pi/2$.

Theorem I.7 (Hyperbolic Pythagoras' theorem [Bus10, Thm 2.2.2(i)]). *In a triangle whose sides are a, b, c*

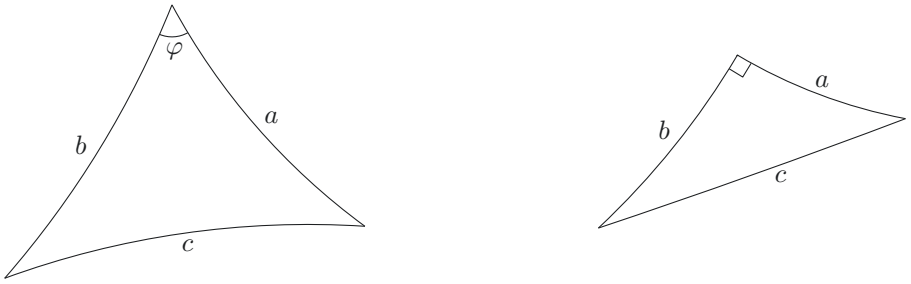

Figure I.4: Illustration of the notations of Theorems I.6 (left) and I.7 (right).

and the angle opposite to c is a right angle, we have

$$\cosh(c) = \cosh(a)\cosh(b).$$

Isometries of the Poincaré disk

An *isometry* $f : (X, d) \to (Y, d')$ is a distance-preserving map between two metric spaces, that is, $d'(f(u), f(v)) = d(u, v)$ for all $u, v \in X$. We are particularly interested in the orientation-preserving isometries of $\mathbb{H}^2$ into itself. These are isometries preserving oriented angles.

Theorem I.8 (Isometries of the Poincaré disk [ADBC+05])**.** *The orientation-preserving isometries of the Poincaré disk are the transformations of the form*

$$\tau : z \mapsto \frac{az + \bar{c}}{cz + \bar{a}}$$

where $a, c \in \mathbb{C}$ and $|a|^2 - |c|^2 = 1$.

Such an isometry τ can also be represented as a 2×2 complex matrix of the form

$$\tau = \begin{pmatrix} a & \bar{c} \\ c & \bar{a} \end{pmatrix} \text{ and } \det(\tau) = 1.$$

The composition of two orientation-preserving isometries of $\mathbb{H}^2$ is then respresented by the product of their matrices.

The set $\mathrm{Isom}^+(\mathbb{H}^2)$ of the orientation-preserving isometries of $\mathbb{H}^2$, equipped with the composition operator, forms a non-Abelian group. The identity of $\mathrm{Isom}^+(\mathbb{H}^2)$ is denoted $\mathbb{1}_\Gamma$.

Every isometry $\tau \in \mathrm{Isom}^+(\mathbb{H}^2)$ extends to the closure $\overline{\mathbb{H}^2}$ of the Poincaré disk as a homeomorphism leaving it invariant. By the Brouwer fixed point theorem, τ has a fixed point in $\overline{\mathbb{H}^2}$. The number of fixed points gives rise to the following classification of $\mathrm{Isom}^+(\mathbb{H}^2)$:

- τ is *elliptic* if it has a fixed point $p \in \mathbb{H}^2$. It then has no fixed points on the unit circle. It is a rotation about p;
- τ is *parabolic* if it has exactly one fixed point on the unit circle;
- τ is *hyperbolic* if it has exactly two fixed points on the unit circle. It then has an invariant geodesic $a_\tau \subset \mathbb{H}^2$ called its *axis* on which it acts by translation. The *displacement length* of τ is the distance

$\ell(\tau)$ between a point on the axis and its image. It is given by $\ell(\tau) = 2\,\mathrm{arcosh}(|\,\mathrm{Tr}(\tau)|/2)$ [Bus10, Eq 9.2.3];

- $\tau = \mathbb{1}_\Gamma$ if it has more than one fixed points in $\mathbb{H}^2$.

I.1.2 Hyperbolic surface

A *surface* is a connected smooth two-dimensional manifold. In this thesis, we only consider compact oriented surfaces without boundary. The *genus* of a surface S is a topological invariant. It represents the number of handles ("doughnut holes") of S. It is defined as the maximum number of non-intersecting simple closed curves that can be cut without causing the resulting surface to be disconnected (Figure I.5). The genus is related to the Euler characteristic χ by the relation $\chi(S) = 2 - 2g$.

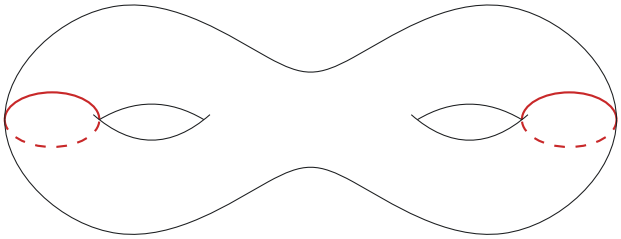

Figure I.5: A genus 2 surface. The two red curves can be cut along without disconnecting the surface.

Definition I.9 (Hyperbolic surface)**.** A *hyperbolic surface* S is a surface equipped with a metric such that S is locally isometric to $\mathbb{H}^2$. In other words, distances and angles on S are locally measured as in $\mathbb{H}^2$. Such a metric is called *hyperbolic*.

Theorem I.10 ([Rat19, 9.3.2])**.** *A surface of genus g admits a hyperbolic metric if and only if $g \geqslant 2$.*

Gaussian curvature

The curvature of a plane curve can be defined in several ways. One of these ways uses osculating circles. Let γ be a smooth curve in the plane. Intuitively, the osculating circle of γ at a point p is the circle that best approximates γ at p. Formally, if $p_1, p_2 \in \gamma$ and $\mathcal{C}(p, p_1, p_2)$ is the circle passing through p, p_1 and p_2, the *osculating circle* of γ at p is the limit $\mathcal{C}(p)$ of $\mathcal{C}(p, p_1, p_2)$ when $p_1 \to p$ and $p_2 \to p$ (Figure I.6). The *curvature* of γ at p is the inverse of the radius of $\mathcal{C}(p)$ [dC76, Sec 1.6]. The *oriented curvature* is obtained by taking into account on which side of γ the circle $\mathcal{C}(p)$ lies.

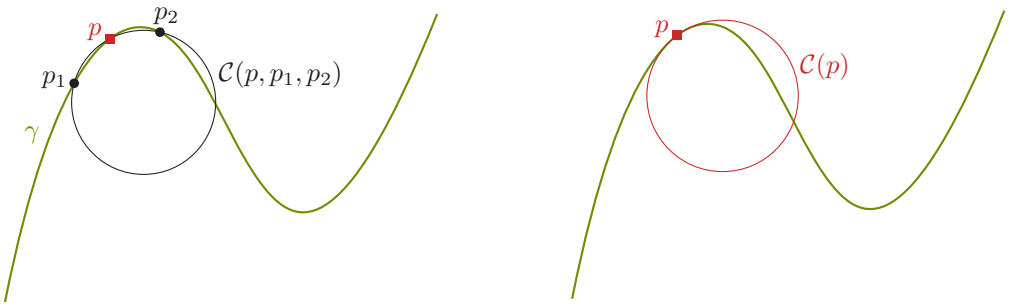

Figure I.6: Illustration of the definition of osculating circle of a plane curve.

Hyperbolic surfaces are surfaces with constant Gaussian curvature -1. This means that every point is a saddle point (Figure I.7). For a surface embedded in $\mathbb{R}^3$, the notion of Gaussian curvature at a point p is defined via *normal planes*, which are the planes containing the normal vector of the surface at p. Each normal plane cuts the surface in a plane curve that has an oriented curvature at p as defined above. The *principal curvatures* at p are the maximum and the minimum of these oriented curvatures, denoted respectively $\kappa_1(p)$ and $\kappa_2(p)$.

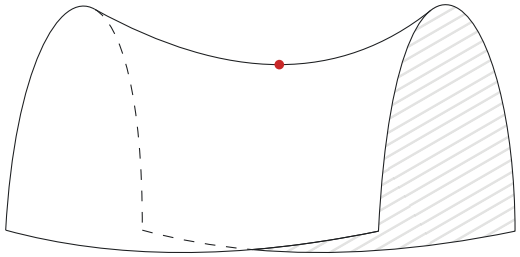

Figure I.7: A saddle point (red).

Definition I.11 (Gaussian curvature [dC76, Sec 3.2])**.** The *Gaussian curvature* of a surface at a point p is

$$K(p) = \kappa_1(p)\kappa_2(p).$$

However, Gauss' *Theorema egregium* (Latin for *remarkable theorem*) states that the notion of Gaussian curvature is intrinsic to the surface: it depends only on its metric and not on its embedding in $\mathbb{R}^3$. The Bertrand–Diguet–Puiseux theorem is an interesting proof of *Theorema egregium*.

Theorem I.12 (Bertrand–Diguet–Puiseux [Ber48])**.** *Let p be a point on a smooth surface and $r > 0$. Let D be the disk of radius r centered at p on the surface. Define $\mathcal{C}(r)$ as its perimeter and $\mathcal{A}(r)$ its area. We have*

$$K(p) = \lim_{r\to 0} 3\frac{2\pi r - \mathcal{C}(r)}{\pi r^3} = \lim_{r\to 0} 12\frac{\pi r^2 - \mathcal{A}(r)}{\pi r^4}.$$

Remark. The numerators of these fractions are respectively the difference of perimeter and area between a Euclidean disk and a disk on the surface. The Gaussian curvature is thus constantly equal to 0 in the Euclidean plane.

This theorem makes it easy to see that the Gaussian curvature of a hyperbolic surface S is -1 at any point. Since S is locally isometric to $\mathbb{H}^2$, D is isometric to a disk in $\mathbb{H}^2$ for r small enough, and we can apply Theorem I.3 to obtain

$$\lim_{r\to 0} 3\frac{2\pi r - \mathcal{C}(r)}{\pi r^3} = \lim_{r\to 0} 6\frac{r - \sinh(r)}{r^3} = \lim_{r\to 0} 6\frac{r - (r - r^3/6 + o(r^3))}{r^3} = -1.$$

The Gauss-Bonnet theorem links the Gaussian curvature with the area and the Euler characteristic χ of a surface. For surfaces with constant curvature, it states the following.

Theorem I.13 (Gauss-Bonnet theorem [dC76, Sec 4.5 (Cor 2)])**.** *The area of a surface $\mathcal{S}$ of constant Gaussian curvature $K \neq 0$ is*

$$\frac{1}{K} \times 2\pi\chi(\mathcal{S}).$$

Recall that the Euler characteristic of a hyperbolic surface of genus g is $2-2g$.

Corollary I.14. *The area of a hyperbolic surface of genus g is*

$$4\pi(g-1).$$

Remark. The area of a hyperbolic surface only depends on its genus. All the hyperbolic surfaces with the same genus have the same area.

Algebraic point of view

A hyperbolic surface is also an algebraic object, more specifically, a quotient of the Poincaré disk by a group of isometries. This algebraic point of view is crucial as it allows us to work in $\mathbb{H}^2$ to do computations on hyperbolic surfaces.

A *path* on a metric space X is a continuous map $f : [0,1] \to X$. By abuse of terminology, the image of a path is also called a path. Two paths f_0, f_1 are *freely homotopic* if one can continuously be deformed into the other. This means that there is a family of paths $f_t : [0,1] \to X$ such that the map $F : [0,1] \times [0,1] \to X$ defined by $F(x,t) = f_t(x)$ is continuous. If f_0 and f_1 also have the same endpoints u and v and $f_t(0) = u$ and $f_t(1) = v$ for all $t \in [0,1]$, then they are *homotopic* (Figure I.8). The relation of homotopy on paths with fixed endpoints is an equivalence relation [Hat02, Prop 1.2]. The equivalence class $[f]$ of a path f under the relation of homotopy is called its *homotopy class*. A path is *homotopically trivial* if it is freely homotopic to a point.

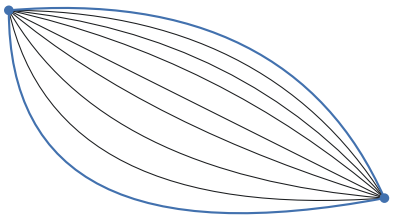

Figure I.8: The two blue paths are homotopic.

A path f is a *loop* if $f(0) = f(1)$. The point $x = f(0)$ is called the *basepoint* of the loop and we say that f is *based* in x. The *composition* $f * g$ of two loops f and g based in x is the loop obtained by going through f and then through g. It is defined as $f * g(t) = f(2t)$ if $t \in [0, 1/2]$ and $f * g(t) = g(2t-1)$ if $t \in [1/2, 1]$. The product of homotopy classes given by $[f][g] = [f * g]$ is well defined and associative. The set $\pi_1(X, x)$ of all homotopy classes of loops based in x is a group with respect to this product [Hat02, Sec 1.1]. It is called the *fundamental group of X at the basepoint x*. If X is path connected, $\pi_1(X, x)$ is, up to isomorphism, independent of the choice of x. It is then called *the fundamental group of X* for short and abbreviated $\pi_1(X)$. This is the case for any surface (compact, connected and without boundary, as assumed above).

Let S be a surface. The *universal cover* $\widetilde{S}$ of S is a simply connected surface equipped with a projection map $\rho : \widetilde{S} \to S$ that is a local isometry. It is unique up to isomorphism [Hat02, Sec 1.3]. The fundamental group $\pi_1(S)$ acts on $\widetilde{S}$ such that $\rho^{-1}(p)$ is the orbit of $\widetilde{p}$ under the action of $\pi_1(S)$. The surface S is thus isomorphic to the quotient $\widetilde{S}/\pi_1(S)$. If S is hyperbolic, its universal cover is the hyperbolic plane and its

fundamental group is isomorphic to a non-Abelian discrete[1] subgroup Γ of $\mathrm{Isom}^+(\mathbb{H}^2)$ (recall Section I.1.1). The surface S is thus isometric to $\mathbb{H}^2/\Gamma$.

Definition I.15 (Lift, projection, copy)**.** With the former notations, a *lift* $\widetilde{p} \in \mathbb{H}^2$ of a point $p \in S$ is an element of the orbit $\rho^{-1}(p) \subset \mathbb{H}^2$. More generally, we call an object $\widetilde{o} \subset \mathbb{H}^2$ a lift of an object $o \subset S$ if $\rho(\widetilde{o}) = o$. Conversely, o is the *projection* of $\widetilde{o}$ if $\rho(\widetilde{o}) = o$. A *copy* of an object in $\mathbb{H}^2$ is its image by an isometry of Γ.

In the remaining of this thesis, S denotes a hyperbolic surface of genus $g \geqslant 2$ seen as a quotient $\mathbb{H}^2/\Gamma$, where Γ is a non-Abelian discrete subgroup of $\mathrm{Isom}^+(\mathbb{H}^2)$.

I.1.3 Fundamental domain

Hilbert showed [Hil01, Tre03] that it is not possible to smoothly represent a hyperbolic surface in Euclidean space while preserving its geometry. A figure like Figure I.5 represents the topology of a hyperbolic surface, but it does not accurately represent its metric. However, a hyperbolic surface can be represented in $\mathbb{H}^2$ by a convex polygon and a set of isometries to glue its sides together.

Before giving a formal definition, we explain the intuition. Let us begin with a simpler example. We consider the flat torus $\mathbb{T}$, a genus 1 surface equipped with a Euclidean metric, which results in its Gaussian curvature being zero everywhere. Figure I.9 (upper left) shows a topological representation of $\mathbb{T}$. Cutting the torus along the curves depicted in the figure, we obtain a parallelogram that can tile the Euclidean plane $\mathbb{R}^2$ (Figure I.9, bottom left). Conversely, $\mathbb{T}$ is obtained by gluing the opposite sides of a parallelogram of $\mathbb{R}^2$.

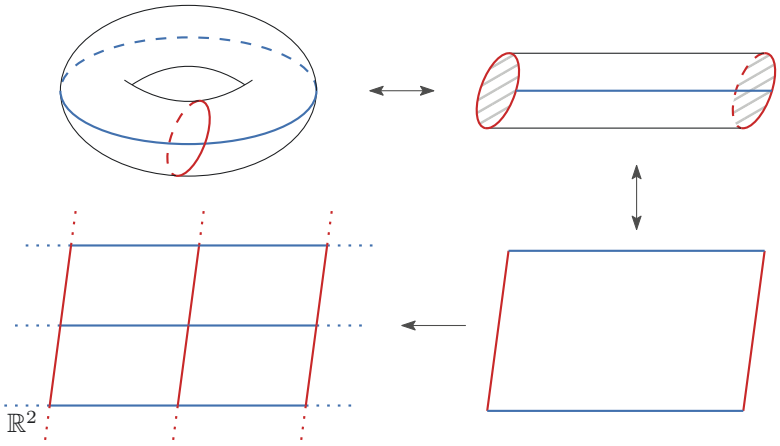

Figure I.9: The flat torus can be cut to obtain a parallelogram that tiles the Euclidean plane $\mathbb{R}^2$.

In the same manner, every hyperbolic surface S can be cut along a set of curves to obtain a hyperbolic polygon (Figure I.10) that tiles $\mathbb{H}^2$. Such a polygon is called a fundamental polygon. Conversely, we obtain S by gluing the paired sides of that polygon.

Definition I.16 (Fundamental domain [Rat19, Sec 6.6])**.** A *fundamental domain* for a hyperbolic surface $S = \mathbb{H}^2/\Gamma$ is a connected open subset $D \subset \mathbb{H}^2$ such that the copies of D are pairwise disjoint and the copies of its closure tessellate $\mathbb{H}^2$. In other words, $\gamma(D) \cap \gamma'(D) = \varnothing$ for all $\gamma \neq \gamma' \in \Gamma$ and $\bigcup_{\gamma\in\Gamma} \gamma(\overline{D}) = \mathbb{H}^2$.

[1] *Discrete* means that for each element $\gamma \in \Gamma$, there exists a neighborhood of γ in which γ is the only element of Γ. The definition is left for completeness, but we will not use it.

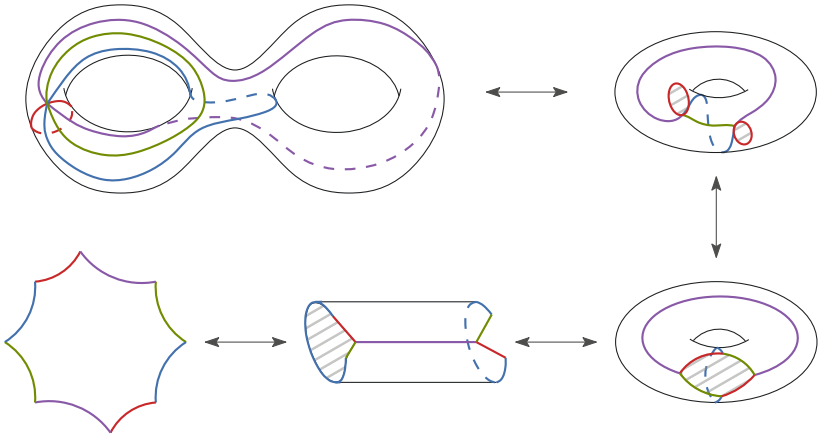

Figure I.10: A hyperbolic surface can be cut to obtain a hyperbolic polygon [Ior19].

Remark. The closure of a fundamental domain contains exactly one lift of each point of S, except on its boundary.

A fundamental domain D is *locally finite* if every compact subset of $\mathbb{H}^2$ meets only a finite number of copies of its closure. This notion plays a key role in the definition of a convex fundamental polygon that follows. It allows these polygons to have the properties that we expect. In particular, any convex and locally finite fundamental domain for a hyperbolic surface S is the interior of a convex polygon [Rat19, Thm 6.7.1].

Definition I.17 (Convex fundamental polygon [Rat19, Sec 6.6])**.** A *convex fundamental polygon* for a hyperbolic surface S is a convex polygon of $\mathbb{H}^2$ such that its interior is a locally finite fundamental domain for S.

Remark. With these definitions, a fundamental domain is open while a convex fundamental polygon is closed (recall the definition of polygon on page 8).

The sides of a convex fundamental polygon P are generally not of the form $P \cap \gamma(P)$ (with $\gamma \in \Gamma \setminus \{\mathbb{1}_\Gamma\}$). When this is the case, P is said to be *exact*, and γ is unique. For every side s of P, there exists a unique isometry $\gamma_s \in \Gamma$ such that $\gamma_s^{-1}(s)$ is also a side s' of P. We say that s' is *paired* to s by γ_s. The set of isometries of Γ pairing the sides of P are called *side-pairings* of P. We will see in Section I.2.3 that Dirichlet domains are exact convex fundamental polygons.

Theorem I.18 ([Rat19, Thm 6.8.3])**.** *Let P be an exact convex fundamental polygon for a hyperbolic surface $S = \mathbb{H}^2/\Gamma$. The group Γ is generated by the side-pairings of P.*

A convex fundamental polygon P is compact since S is compact (by definition). It follows that P has finitely many sides [Rat19, Thm 6.3.6]. The group Γ is then finitely generated. Moreover, the number k of vertices and sides of P is even and satisfies $4g \leqslant k \leqslant 12g - 6$ [DKPT23, Sec 2.3]. This is due to the Euler characteristic of the graph associated with P. The lower bound is attained when all the vertices of P are lifts of the same point on S. There always exists a convex fundamental polygon that attains the lower bound [DKPT23]. The upper bound is attained when all the vertices of the graph associated with P have degree three.

A hyperbolic surface is thus uniquely determined by an exact convex fundamental polygon and its side-pairings.

I.1.4 Small simple closed geodesics

In this section, we focus on the simple closed geodesics of the hyperbolic surface S. These are curves around which the local topology and the geometry of the surface are well understood thanks to the Collar Theorem. In particular, small simple closed geodesics (for some definition of "small") naturally give rise to a decomposition of S into a "thin" and a "thick" part.

The length of a curve on S is denoted by ℓ. The distance on S is denoted by d_S.

A *closed curve* on S is a continuous map from the unit circle to S. By abuse of terminology, its image on S is also called closed curve. Up to reparametrization, a curve is also a path (as defined page 12). A curve is *geodesic* if it locally minimizes distances. For example, a geodesic loop is a geodesic path but not a closed geodesic curve in general (Figure I.11). For short, a closed geodesic curve is called *closed geodesic*. A curve is *simple* if it is injective, and thus its image has no self-intersection.

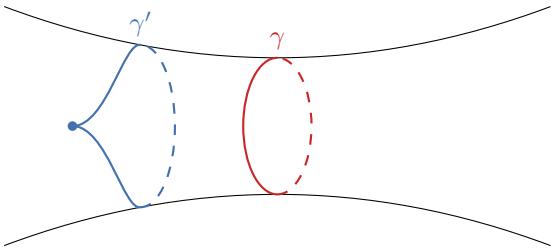

Figure I.11: The geodesic loop γ' is not a closed geodesic but it is freely homotopic to γ, which is a closed geodesic.

Definition I.19 (Systole)**.** The *systole* σ of a hyperbolic surface is the length of its shortest closed geodesic.

By abuse of terminology and notation, any closed geodesic of length σ is also called a systole and is denoted by σ.

Theorem I.20 ([FM12, Prop 1.3])**.** *Let S be a hyperbolic surface. Any homotopically non-trivial closed curve on S is freely homotopic to a unique closed geodesic on S.*

Moreover, there is a one-to-one correspondence between the conjugacy classes in Γ and the *oriented* closed geodesics on S. If γ is a closed geodesic on S associated with a translation τ, then the axis of τ projects on γ and $\ell(\gamma) = \ell(\tau)$ [Bus10, Eq 9.2.4] (hence the choice of the same notation for both).

Definition I.21 (Collar)**.** Let γ be a simple closed geodesic on a hyperbolic surface S. The *collar* of γ is $\mathcal{C}(\gamma) = \{x \in S : d_S(x, \gamma) \leqslant w(\gamma)\}$, where

$$w(\gamma) = \operatorname{arsinh}\left(\frac{1}{\sinh\left(\ell(\gamma)/2\right)}\right)$$

is its *width*.

Remark. The shorter the geodesic, the longer the collar. In particular, $2w(\gamma) = \ell(\gamma)$ if $\ell(\gamma) = 2\operatorname{arsinh} 1$.

Theorem I.22 (Collar Theorem I [Bus10, Thm 4.1.1])**.** *Let S be a hyperbolic surface of genus g. The following holds.*

1. *The collar of a simple closed geodesic γ is homeomorphic to the cylinder $[-w(\gamma), w(\gamma)] \times \mathcal{S}^1$, where $\mathcal{S}^1$ is the unit circle of $\mathbb{R}^2$.*

2. *If two simple closed geodesics are disjoint, their collars are also disjoint.*

3. *There are at most* $3g-3$ *pairwise disjoint simple closed geodesics on* S.

An important consequence of this theorem is that the simple closed geodesics of length at most $2\operatorname{arsinh}1 \approx 1.76$ on S, called *small curves*, are pairwise disjoint. This is because if γ and γ' intersect, and $\ell(\gamma) \leqslant 2\operatorname{arsinh}1$, then γ' must pass through the collar of γ so $\ell(\gamma') \geqslant 2w(\gamma) \geqslant 2\operatorname{arsinh}1$.

The *injectivity radius* $r_p(S)$ of S at a point p is the supremum of all $r>0$ such that the open ball of radius r centered at p, $B(p,r)=\{q\in S : d_S(p,q)<r\}$, is isometric to a disk in $\mathbb{H}^2$. In particular, $B(p,r)$ is a topologically embedded disk on S for all $r \leqslant r_p(S)$. The injectivity radius of S, $r_{inj}(S)$, is the infimum of all the injectivity radii.

Theorem I.23 ([Bus10, Lem 4.1.5]). *For any point* p *on a hyperbolic surface* S, $2r_p(S)=\ell(\mu_p)$ *where* μ_p *is the shortest geodesic loop based at* p. *Moreover,* $2r_{inj}(S)=\sigma$.

Theorem I.24 (Collar Theorem II [Bus10, Thm 4.1.6]). *Let* $\gamma_1,\dots,\gamma_k$ *be all the simple closed geodesics of length at most* $2\operatorname{arsinh}1$ *on a hyperbolic surface* S *of genus* g. *The following holds.*

1. *The geodesics* $\gamma_1,\dots,\gamma_k$ *are pairwise disjoint and* $k\leqslant 3g-3$.

2. *If* $p\in S\setminus(\mathcal{C}(\gamma_1)\cup\dots\cup\mathcal{C}(\gamma_k))$, *then* $r_p(S)>\operatorname{arsinh}1$.

3. *If* $p\in\mathcal{C}(\gamma_i)$ *and* $i\in\{1,\dots,k\}$, *then* $\sinh(r_p(S))=\sinh\left(\ell(\gamma_i)/2\right)\cosh(d_S(p,\gamma_i))$.

In summary, the Collar Theorem states that the collars of the geodesic curves $(\gamma_i)_{1\leqslant i\leqslant k}$ are pairwise disjoint. Moreover, each point $p\in S$ such that $r_p(S)\leqslant\operatorname{arsinh}1$ lies in a collar of some γ_i, but the converse is false in general. This gives rise to the following "thick-thin" decomposition of S.

Definition I.25 (ε-thin & ε-thick part [EPV22]). Let $\varepsilon>0$. The *ε-thin part* of a hyperbolic surface S is

$$S_\varepsilon^{\text{thin}}=\{x\in S : r_x(S)<\varepsilon/2\},$$

and its *ε-thick part* is $S_\varepsilon^{\text{Thick}}=S\setminus S_\varepsilon^{\text{thin}}$.

If $\varepsilon<\sigma$, then $S_\varepsilon^{\text{thin}}$ is empty. We then say that the surface S is *ε-thick*. If $\varepsilon>2\operatorname{arsinh}1$, the Collar Theorem provides no information on the topology of $S_\varepsilon^{\text{thin}}$. The ε-thin part is interesting when $\sigma\leqslant\varepsilon\leqslant 2\operatorname{arsinh}1$. In this case, $S_\varepsilon^{\text{thin}}$ is included in $\bigcup_{1\leqslant i\leqslant k}\mathcal{C}(\gamma_i)$ by Theorem I.24. In fact, $S_\varepsilon^{\text{thin}}$ is the set of points at distance less than $\operatorname{arcosh}(\sinh(\varepsilon/2)/\sinh(\ell(\gamma_i)/2)$ of some γ_i [EPV22, Sec 3.1] (Figure I.12), hence the following definition.

Definition I.26 (ε-collar). Let $0<\varepsilon\leqslant 2\operatorname{arsinh}1$ and γ be a simple closed geodesic of length at most ε on a hyperbolic surface S. The *ε-collar* of γ is $\mathcal{C}(\gamma,\varepsilon)=\{x\in S : d_S(x,\gamma)<w(\gamma,\varepsilon)\}$, where

$$w(\gamma,\varepsilon)=\operatorname{arcosh}\left(\frac{\sinh(\varepsilon/2)}{\sinh\left(\ell(\gamma)/2\right)}\right)$$

is its *width*.

Note that the two former definitions imply that the injectivity radius of any point on the boundary of an ε-collar is $\varepsilon/2$.

Remark. While the cited article uses non-strict inequalities, we use a strict inequality in Definition I.25, which yields a strict inequality in Definition I.26 as well. The only consequence of this change is that the boundary of an ε-collar is included in $S_\varepsilon^{\text{Thick}}$ with our definition.

This definition allows us to restate the above concisely: if $0 < \varepsilon \leqslant 2\operatorname{arsinh} 1$, then

$$S_\varepsilon^{\text{thin}} = \bigcup_{1 \leqslant i \leqslant k} \{\mathcal{C}(\gamma_i, \varepsilon) : \ell(\gamma_i) \leqslant \varepsilon\}.$$

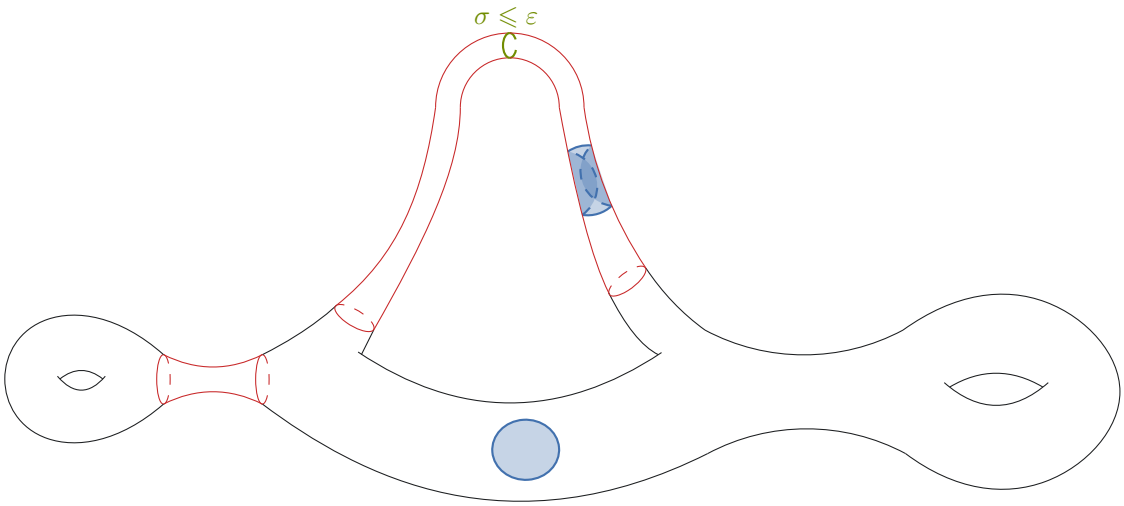

Figure I.12: ε-thick (black) and ε-thin (red) parts of a hyperbolic surface for $\varepsilon \leqslant 2\operatorname{arsinh} 1$. Disks of radius ε are represented in blue. The systole σ appears in green.

I.2 Delaunay triangulation & Voronoi diagram

This section introduces the Delaunay triangulation, the key component of our ε-net algorithm. It is defined in the Euclidean plane, then generalized to the hyperbolic plane and hyperbolic surfaces. The generalization of the Voronoi diagram, the dual of a Delaunay triangulation, to the hyperbolic plane gives rise to a family of convex fundamental polygons known as Dirichlet domains. The section ends with a discussion on the useful properties of both Delaunay triangulations and Dirichlet domains.

I.2.1 Delaunay triangulation in the Euclidean plane

A *triangulation* of a point set $P \subset \mathbb{R}^2$ is a maximal planar subdivision of $\mathbb{R}^2$ whose vertices are the points of P [dBCvKO08, Sec 9.1], meaning that no edge can be added without creating an intersection with an existing edge. Equivalently, a triangulation is the partition of the convex hull of P into triangles. It is both a geometric and a combinatorial object since the subdivision is an embedding of a graph in the plane. When the triangulation is viewed as a geometric object, the terms *point*, *segment* and *triangle* will be used. When it is viewed as a combinatorial object, the terms *vertex*, *edge* and *face* will be used.

While Delaunay triangulations have many properties that make them a key tool in computational geometry, in this thesis, we only need its definition.

Definition I.27 (Delaunay triangulation [dBCvKO08, Thm 9.7])**.** A *Delaunay triangulation* $DT(P)$ of a point set $P \subset \mathbb{R}^2$ is a triangulation of P such that the circumcircle of any triangle is *empty*, that is, it contains no point of P in its interior.

Remark. If no four points of P are cocyclic, then there is a unique Delaunay triangulation of P. Otherwise, there can be several.

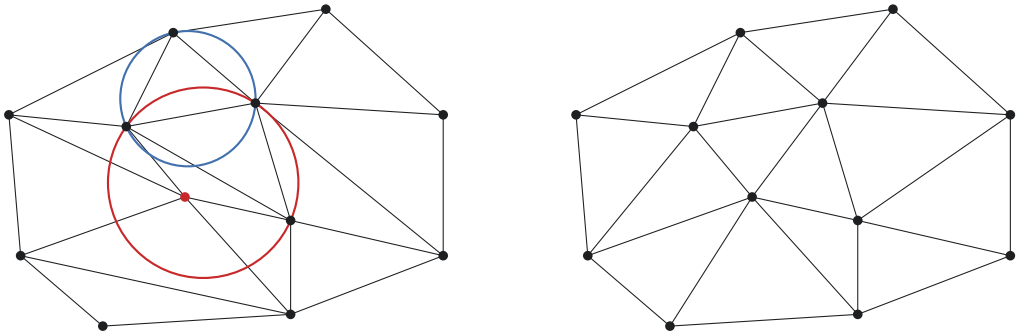

Figure I.13: Left: a non-Delaunay triangulation of a point set P. The blue circle is empty, while the red one is not. Right: the Delaunay triangulation of P.

For a given triangulation, the *global* property of being Delaunay is characterized by a *local* property of its edges.

Definition I.28 (Locally Delaunay edge)**.** Let T be a triangulation in the Euclidean plane. Let $t = (a, b, c)$ and $t' = (c, d, a)$ be two triangles of T sharing the edge (a, c). If the circumcircle of t does not contain d and the circumcircle of t' does not contain b, then the edge (a, c) is *locally Delaunay*.

Theorem I.29 ([dBCvKO08, Thm 9.8])**.** *A triangulation in* $\mathbb{R}^2$ *is Delaunay if and only if all its edges are locally Delaunay.*

The operation of replacing the edge (a, c) with (b, d) is called a *Delaunay flip* or a *flip* for short (Figure I.14). This theorem provides the foundation for the *flip algorithm*, introduced by Lawson in 1977 [Law77]: starting from any triangulation, non-locally Delaunay edges are flipped until all edges are locally Delaunay. The number of edge flips that are needed to transform any triangulation with n vertices into a Delaunay triangulation is $\Theta(n^2)$ [HNU99].

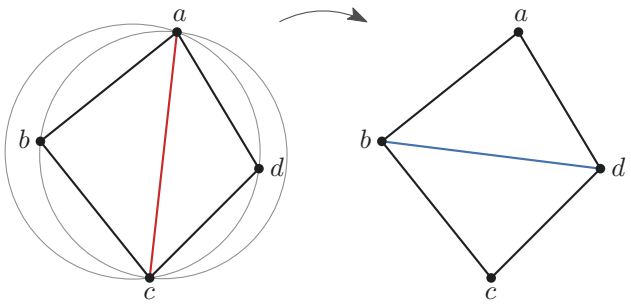

Figure I.14: Delaunay flip of the edge (a, c).

The flip operation is also central to incrementally constructing a Delaunay triangulation. In the *incremental flip algorithm*, vertices are added one at a time, and flips are used to restore the Delaunay property after each insertion.

Another well-known incremental algorithm for constructing Delaunay triangulations is Bowyer's algorithm [Bow81, Wat81]. It works as follows (Figure I.15). Let $P \subset \mathbb{R}^2$ be a set of points. Suppose that the Delaunay triangulation $DT(P)$ is known. Let $p \notin P$ be a point to be inserted in the Delaunay triangulation. To compute $DT(P \cup \{p\})$, the triangles in *conflict* with p are found. These are the triangles whose circumcircle contain p in their interior. They are destroyed, and the resulting "hole" (the *conflict zone*) is retriangulated by creating an edge between p and every vertex on the boundary of the hole.

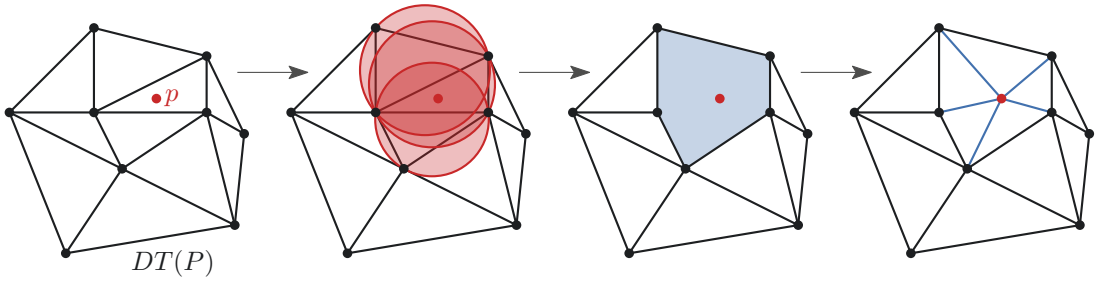

Figure I.15: Insertion of the point p in the Delaunay triangulation $DT(P)$ during Bowyer's algorithm.

I.2.2 Delaunay triangulation on a hyperbolic surface

The definition of a Delaunay triangulation on a hyperbolic surface requires generalizing the concepts of circumcircle and triangulation in $\mathbb{H}^2$. A *triangulation* of a set of points in $\mathbb{H}^2$ is defined similarly to its counterpart in $\mathbb{R}^2$, except that the edges of the planar subdivision are now hyperbolic segments. The *(hyperbolic) circumcircle* of a triangle in $\mathbb{H}^2$ is the restriction to the Poincaré disk of its Euclidean circumcircle. If the hyperbolic circumcircle of a triangle is included in the Poincaré disk, it is a circle in the sense of the set of points at a constant hyperbolic distance from a point in $\mathbb{H}^2$. It is then called a *compact circle* [DST24, Sec 2.2].

Delaunay triangulations are defined in the Poincaré disk $\mathbb{H}^2$ similarly as in the Euclidean plane, considering hyperbolic triangles and circumcircles instead of Euclidean ones. Since hyperbolic circumcircles coincide with Euclidean circles, a hyperbolic Delaunay triangulation of a point set in the open unit disk of $\mathbb{C}$ is obtained from a Euclidean Delaunay triangulation by replacing the Euclidean segments with hyperbolic ones (Figure I.16).

A *geometric triangulation* T of a hyperbolic surface S is an embedded graph that partitions S, whose faces have three (not necessarily distinct) vertices and whose edges are geodesic segments that do not intersect in their interior. For any finite set of points $V \subset S$, we say that T is a triangulation of V if its vertices are the points of V. The *lift* of T is $\rho^{-1}(T) \subset \mathbb{H}^2$, where $\rho : \mathbb{H}^2 \to S$ is the locally isometric projection map defined page 12. We denote it $\widetilde{T}$. It is the infinite triangulation of $\mathbb{H}^2$ formed by all the lifts of all vertices of T such that its projection onto S is T. In particular, the edges of $\widetilde{T}$ are geodesic segments since the projection ρ is a local isometry.

Definition I.30 (Delaunay triangulation of a hyperbolic surface [DST24, Sec 2.3])**.** A geometric triangulation of a hyperbolic surface is a *Delaunay triangulation* if its lift is a Delaunay triangulation in $\mathbb{H}^2$.

If T is a Delaunay triangulation of S, any circumcircle of a triangle of $\widetilde{T}$ is compact. This is because the closed region bounded by a non-compact circle contains a fundamental domain for S. Therefore, a non-compact circle cannot be empty.

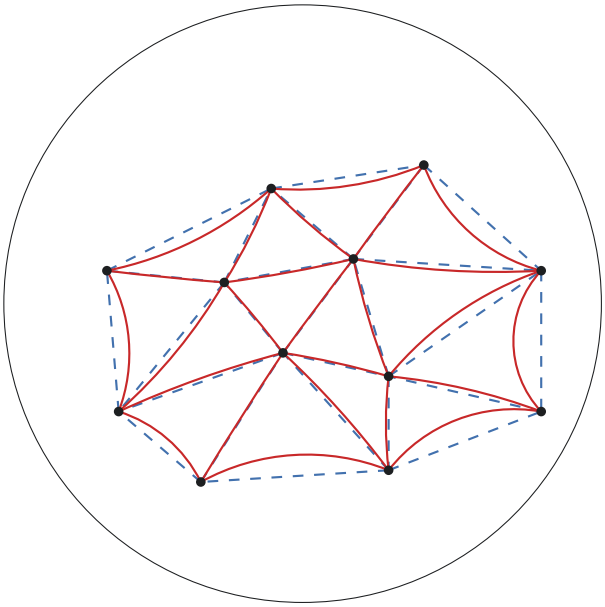

Figure I.16: A hyperbolic Delaunay triangulation obtained by replacing Euclidean segments (dashed blue) with hyperbolic segments (plain red).

Most algorithms to compute Delaunay triangulations in the Euclidean plane do not generalize naturally to hyperbolic surfaces. However, the flip algorithm does generalize to geometric triangulations and is straightforward to implement. We will see in Section I.4 two data structures that support this algorithm.

The following two theorems are trivial for triangulations in the Euclidean plane, but are not obvious for geometric triangulations on hyperbolic surfaces. They are therefore significant for the study of the flip algorithm on hyperbolic surfaces.

Theorem I.31 ([DST24, Lem 4.1]). *Let S be a hyperbolic surface and T be a geometric triangulation of S. The triangulation obtained after flipping an edge of T is also geometric.*

The following theorem gives the maximum number of flips performed in the flip algorithm on a hyperbolic surface. This number of flips depends on the initial geometric triangulation via its longest edge.

Theorem I.32 ([DST24, Thm 5.4]). *Let S be a hyperbolic surface of genus g and $V \subset S$ be a finite set of n points. Starting from any geometric triangulation T of V, the flip algorithm can compute a Delaunay triangulation of V with at most $C_h \Lambda(T)^{6g-4} n^2$ flips, where C_h is a constant depending on the metric h of S and $\Lambda(T)$ is the length of the longest edge(s) of T.*

I.2.3 Voronoi diagram and Dirichlet domain

The Voronoi diagram of a point set $P \subset \mathbb{R}^2$ is the dual structure to the Delaunay triangulation of P.

Definition I.33 (Voronoi diagram [dBCvKO08, Sec 7.1]). Let $P \subset \mathbb{R}^2$ be a point set. The *Voronoi cell* of a point $p \in P$ is the closed region consisting of all points in $\mathbb{R}^2$ that are closer to p than to any other point in P. The *Voronoi diagram* of P is the collection of the Voronoi cells of all the points of P (Figure I.17). The points of P are called *sites*.

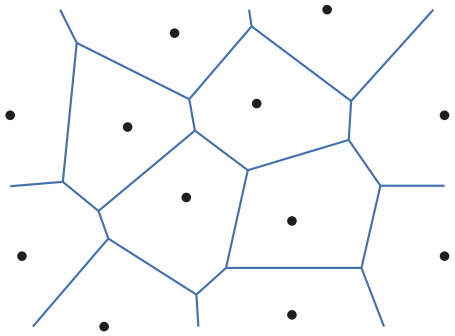

Figure I.17: Voronoi diagram of a point set.

Let x be a vertex of the Voronoi diagram of P. It is incident to (at least) three Voronoi cells of three sites p_1, p_2, p_3. By definition, x is closer to these three sites than to any other site. It follows that x is the center of the circle passing through p_1, p_2, p_3 and there is no other site inside this circle (Figure I.18, left). The triangle (p_1, p_2, p_3) is therefore in a Delaunay triangulation of P. A Delaunay triangulation of P is thus obtained by creating an edge between two points of P whose Voronoi cells are adjacent (Figure I.18, right). Conversely, the Voronoi diagram of P is obtained by tracing the perpendicular bisector of the edges of a Delaunay triangulation of P.

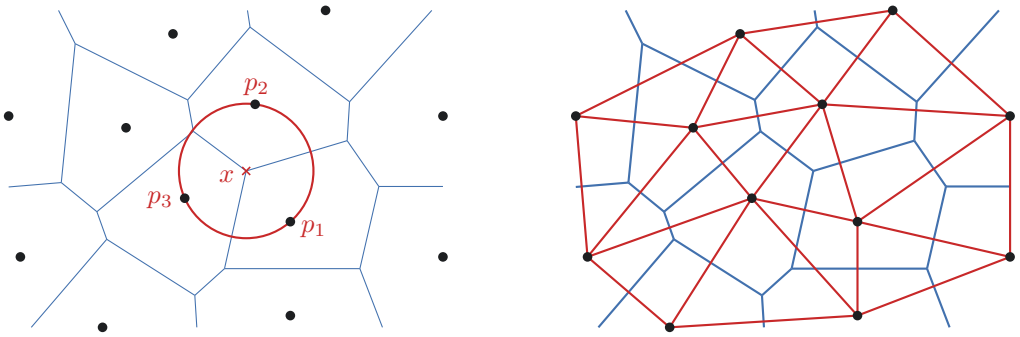

Figure I.18: Left: the circumcircle of p_1, p_2, p_3 is empty. Right: duality of the Voronoi diagram and the Delaunay triangulation.

Like Delaunay triangulations, Voronoi diagrams are generalized to the Poincaré disk using the hyperbolic distance $d_{\mathbb{H}^2}$ instead of the Euclidean one. The Voronoi diagram in $\mathbb{H}^2$ of all the lifts of a given point of a hyperbolic surface gives rise to fundamental domains called Dirichlet domains. Dirichlet domains are considered canonical fundamental domains because they are naturally convex.

Definition I.34 (Dirichlet domain [DKT24, Sec 2])**.** Let $S = \mathbb{H}^2/\Gamma$ be a hyperbolic surface and $\widetilde{x} \in \mathbb{H}^2$ be a lift of a point $x \in S$. The *Dirichlet domain centered at* $\widetilde{x}$ is the Voronoi cell $\mathcal{D}(\widetilde{x})$ of $\widetilde{x}$ in the Voronoi diagram of $\Gamma\widetilde{x} = \{g(\widetilde{x}) : g \in \Gamma\}$, the orbit of x. Therefore, $\mathcal{D}(\widetilde{x}) = \{y \in \mathbb{H}^2 : \forall g \in \Gamma, d_{\mathbb{H}^2}(\widetilde{x}, \widetilde{y}) \leqslant d_{\mathbb{H}^2}(g(\widetilde{x}), \widetilde{y})\}$.

A Dirichlet domain is an exact convex fundamental polygon [Rat19, Thm 6.7.4(2)]. The combinatorics of the domain depend on the choice of the central point $x \in S$, but not on the choice of a lift of x. This is because any two Dirichlet domains centered at lifts of x are equal up to translation by an element of Γ.

I.2.4 Distance paths

A *distance path* on a hyperbolic surface S is a shortest path between two points. It is necessarily a geodesic segment, but not all geodesic segments are distance paths since they minimize distances only locally. The lift (in $\mathbb{H}^2$) of a distance path is also called a distance path. A *half-minimizer* on S is the concatenation of at most two distance paths. A lift in $\mathbb{H}^2$ of a half-minimizer is also called a half-minimizer. One of the key properties of Dirichlet domains is that a distance path can intersect at most twice each side of a Dirichlet domain. This is a consequence of the two following theorems.

Theorem I.35 ([DKT24, Prop 4])**.** *Let S be a hyperbolic surface. Each edge of a Dirichlet domain for S is either a distance path or a half-minimizer.*

Theorem I.36 ([DKT24, Lem 2])**.** *Let S be a hyperbolic surface. A half-minimizer and a distance path that do not have a subarc in common can intersect at most twice.*

There is a similar result for edges of a Delaunay triangulation.

Theorem I.37. *Let S be a hyperbolic surface. An edge of a Delaunay triangulation of S and a distance path that do not have a subarc in common can intersect at most twice.*

Even though this theorem has not been explicitly stated and proved in [DKT24], it is an immediate consequence of this article. This is why we still consider this theorem as background.

Proof. Let e be an edge of a Delaunay triangulation of a hyperbolic surface. It is homotopic with fixed endpoints to a half-minimizer c [DKT24, Prop 3]. The curves e and c form the boundary of a topological disk D on the surface. A distance path γ that does not have a subarc in common with e cannot intersect e without intersecting c. In particular, γ cannot traverse D just by passing through e, otherwise it would not be a distance path. Since γ can intersect c at most twice, it can intersect e at most twice. □

I.3 ε-net

This short section is devoted to introducing ε-nets and their basic properties. Several definitions exist in the literature. In particular, some references [Sut09, Def 14.19] call an ε-net what we call an ε-covering (with a strict inequality in the definition). We stick to the definition used by Clarkson [Cla06] because it is the one that best reflects what we aim to construct on hyperbolic surfaces: a set of well-distributed points on the surface.

Definition I.38 (ε-packing, ε-covering, ε-net [Cla06])**.** Let (X, d) be a metric space and $\varepsilon > 0$. A subset $P \subset X$ is:

- an *ε-covering* if $d(x, P) \leqslant \varepsilon$ for all $x \in X$, that is, the closed balls of radius ε centered at the points of P cover X;
- an *ε-packing* if $d(p, q) \geqslant \varepsilon$ for all $p \neq q \in P$, that is, the open balls of radius $\varepsilon/2$ centered at the points of P are pairwise disjoint;
- an *ε-net* if it is both an ε-covering and an ε-packing.

The following theorem presents some properties of ε-packings and ε-nets of metric spaces. In particular, in a compact metric space such as a hyperbolic surface, there always exists a finite ε-net. However, our algorithm to compute an ε-net of a hyperbolic surface also serves as an alternative, although less direct, proof that finite ε-nets exist for hyperbolic surfaces.

Theorem I.39. *Let (X, d) be a metric space and $\varepsilon > 0$. The following holds.*

1. *An ε-packing is an ε-net if it is maximal.*
2. *If X is compact, any ε-packing of X is finite.*
3. *If X has diameter larger than ε, then it contains an ε-net.*

This theorem seems to be considered as folklore among mathematicians. Since finding a proof from a standard source is difficult, we prove it here. The proofs of items 1 and 3 are a rewriting[2] of Bellissard [Bel, Prop 1]. The proof of 2 is inspired by [Sut09, Prop 14.21].

Proof.

1. Let P be an ε-packing of X. If P is not an ε-covering, then there exists $x \in X$ such that $d(x, P) > \varepsilon$. So $P \cup \{x\}$ is an ε-packing that strictly contains P, meaning that P is not maximal.
2. Let $P \subset X$ be infinite and let $(x_n)_{n\geqslant 0} \subset P$ be a sequence such that $x_n \neq x_m$ for all $n \neq m$. Since X is compact, there exists a convergent subsequence $(x_{\varphi(n)})_{n\geqslant 0}$. In particular, it is a Cauchy sequence, so $d(x_{\varphi(n)}, x_{\varphi(m)}) < \varepsilon$ for all n and m large enough. Therefore, P cannot be an ε-packing. By contraposition, if P is an ε-packing, then P is finite.
3. Since X has diameter larger than ε, it contains two points $x, y \in X$ such that $d(x, y) \geqslant \varepsilon$. The family of ε-packings of X is thus not empty. Moreover, it is partially ordered by inclusion. Consider a chain $(P_i)_{i\in I}$, that is, if $i \neq j$, $P_i \subset P_j$ or $P_j \subset P_i$. Let $P = \bigcup_{i\in I} P_i$. If $x \neq y \in P$, there exists $i, j \in I$ such that $x \in P_i$ and $y \in P_j$. Without loss of generality, suppose that $P_i \subset P_j$. Then both x and y belong in P_j. Because P_j is an ε-packing, we have $d(x, y) \geqslant \varepsilon$. Hence P is an upper bound of the chain $(P_i)_{i\in I}$ in the family of ε-packings of X. By Zorn's lemma, the family of ε-packings of X contains a maximal element, hence an ε-net by 1.

□

Remark. The converse of 1 is true when using a strict inequality in the definition of covering.

I.4 Data structures for triangulations on hyperbolic surfaces

In this section, we discuss the two data structures that we use to represent triangulations on hyperbolic surfaces. The first relies on a fundamental domain where each vertex and each triangle of the triangulation has a lift. The second one represents the triangulation intrinsically on the surface with a combinatorial map. Both efficiently support the flip operation. Each has its strengths and its weaknesses, which will be discussed. For convenience, we have given these data structures descriptive names.

We consider these data structures in the *real RAM model*. This is a model of computing that can compute with real numbers and in which the standard arithmetic operations (such as addition, subtraction, multiplication, division and comparison) are done in constant time.

[2] We allow ourselves to rewrite them instead of simply quoting them because the only online copy of [Bel] that we found is, to this day, accessible only via the Wayback Machine.

I.4.1 The "fundamental domain" data structure

This data structure relies on an exact fundamental polygon in which lies the vertices of the triangulation. It is based on a data structure used to represent triangulations on d-dimensional flat orbifolds [CT16]. It has been adapted for triangulations on the Bolza surface [IT17], which is the most symmetric hyperbolic surface of genus 2. More recently, it was generalized for the study of the flip algorithm on hyperbolic surfaces [DST20].

In this data structure, the hyperbolic surface S is represented by an exact fundamental polygon $\widetilde{F}$ together with its side-pairing isometries. Recall that $\widetilde{F}$ contains exactly one lift of each point on S except on its boundary: if a point on S has a lift on a side s of $\widetilde{F}$, then it has another lift on the side paired to s. We define an *original domain* $\widetilde{F}_o$ as a subset of $\widetilde{F}$ containing exactly one lift of each point of S. It consists of the interior of $\widetilde{F}$ and half of its sides. A triangulation of S is represented by:

- its vertices: a vertex x has constant-time access to its lift $\widetilde{x}$ in $\widetilde{F}_o$ and one of its incident triangles;
- and its triangles: a triangle $t = (x, y, z)$ has constant-time access to its three vertices, its three adjacent triangles, and three isometries $\gamma_{x,t} = \mathbb{1}_\Gamma, \gamma_{y,t}, \gamma_{z,t}$ in Γ defined as follows.

A triangle $t = (x, y, z)$ does not always have a lift entirely included in $\widetilde{F}_o$. However, it always has between one and three lifts with at least one vertex in $\widetilde{F}_o$ (see Figure I.19). Let us choose such a lift and denote it as $\widetilde{t}_o$. Up to a re-indexing of its vertices, $\widetilde{x} \in \widetilde{F}_o$. Then $\gamma_{x,t} = \mathbb{1}_\Gamma$, and $\gamma_{y,t}$ and $\gamma_{z,t}$ are the isometries such that the other two vertices of $\widetilde{t}_o$ are $\gamma_{y,t}(\widetilde{y})$ and $\gamma_{z,t}(\widetilde{z})$. The other lifts of t having at least one vertex in $\widetilde{F}_o$ are retrieved by applying the inverses of these isometries to $\widetilde{t}_o$. The union, on all triangles of the triangulation of S, of their lifts with at least one vertex in $\widetilde{F}_o$ covers the original domain $\widetilde{F}_o$.

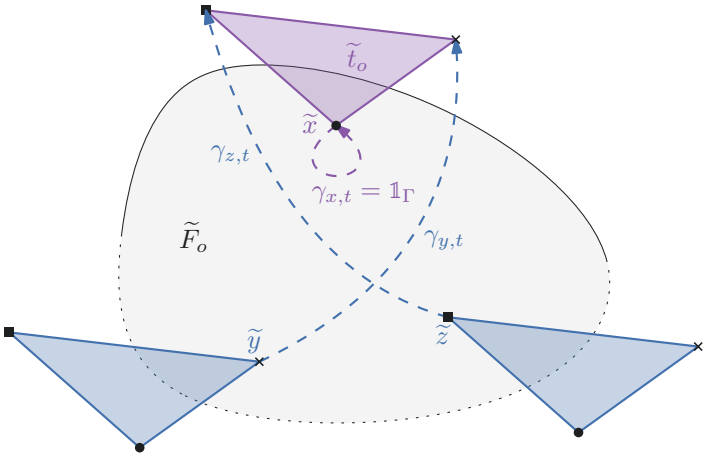

Figure I.19: Example of a triangle t having three lifts with one vertex in $\widetilde{F}_o$. The hyperbolic triangles are schematically represented with straight edges.

To maintain the data structure after the flip of an edge common to two triangles t and t', the lift of $\widetilde{t'}_o$ adjacent to $\widetilde{t}_o$ is first computed using the isometries stored in t and t'. The edge is then flipped. After the flip, a lift having a vertex in $\widetilde{F}_o$ is already known for each new triangle. It remains to compute the isometries of the new triangles, which is easily done using the isometries of the former triangles (Figure I.20). This data structure thus supports the flip operation in constant time.

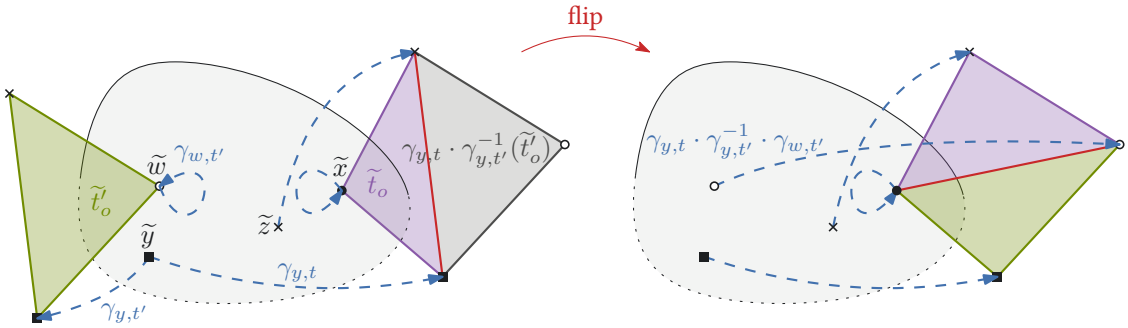

Figure I.20: Updating the data structure after the flip of an edge (red).

This data structure is simple and facilitates the explanation of algorithms. Moreover, it is straightforward to analyze the complexity of algorithms when the fundamental domain is a Dirichlet domain, thanks to the properties discussed in Section I.2.4. However, it is tied to specific representations of objects in $\mathbb{H}^2$, which makes the implementation of algorithms cumbersome. Indeed, exploring the universal cover, for example during a point location algorithm, requires computing copies of the fundamental domain before working on the triangulation. The following data structure does not suffer from this problem.

I.4.2 The "combinatorial map" data structure

This data structure is a combinatorial map augmented with geometrical information. Combinatorial maps were first introduced by Edmonds in 1960 to represent polyhedral surfaces with embedded graphs [Edm60]. However, they only represent the combinatorics of the graphs and the geometric information is missing. In the case of a triangulation on a hyperbolic surface, combinatorial maps have been augmented with geometric information to obtain a data structure suitable to compute flips and retrieve a portion of the lift of the triangulation [DDKT22]. In contrast to the "fundamental domain" data structure, which is tied to specifics lifts of vertices and triangles in $\mathbb{H}^2$, this "combinatorial map" data structure lies intrinsically on the surface. It is convenient for graph algorithms and its implementation is straightforward. This makes it efficient and versatile to use in practice.

A *combinatorial map* is an edge-centered data structure based on *darts*, which are equivalent to half-edges in dimension 2. A dart can be seen as an oriented edge. The combinatorial map only contains darts and pointers between these darts. In dimension 2, each dart has a pointer β_1 to access the dart of the next edge in the same face, and a pointer β_2 to access the dart of the same edge in the adjacent face (Figure I.21). Vertices, edges and faces are retrieved easily: all the darts of a given face are obtained by following β_1 pointers and all the darts of a given vertex are obtained by following $\beta_1 \circ \beta_2$ combinations of pointers.

The geometric information of a triangulation T on the hyperbolic surface S is given as a cross-ratio associated with each edge.

Definition I.40 (Cross-ratio [Aud03, Sec V.6])**.** Let $z_1, z_2, z_3, z_4 \in \mathbb{C}$ be four pairwise distinct points. Their *cross-ratio* is the complex number

$$[z_1, z_2, z_3, z_4] = \frac{(z_4 - z_2)(z_3 - z_1)}{(z_4 - z_1)(z_3 - z_2)}.$$

Theorem I.41 ([Aud03, Prop V.7.6])**.** *Four non-collinear points $z_1, z_2, z_3, z_4 \in \mathbb{C}$ are cocyclic if and only if*

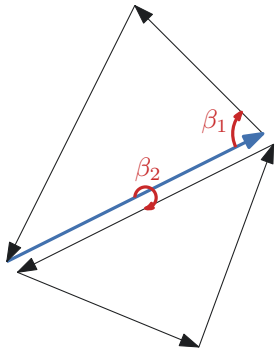

Figure I.21: A dart (bold blue) in a 2D combinatorial map, and its pointers.

$[z_1, z_2, z_3, z_4] \in \mathbb{R}$.

This theorem is a consequence of the geometric interpretation of the argument of the cross-ratio: $\arg([z_1, z_2, z_3, z_4]) = \arg((z_4 - z_2)/(z_4 - z_1)) - \arg((z_3 - z_2)/(z_3 - z_1)) = \alpha - \beta$ with the notations illustrated in Figure I.22. Applying the inscribed angle theorem concludes the proof.

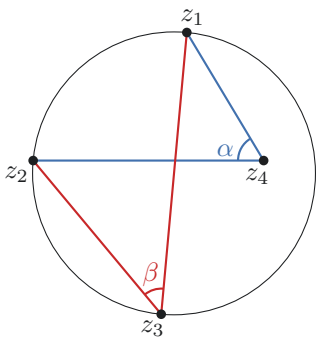

Figure I.22: Interpretation of the argument of the cross-ratio.

Theorem I.42. *Let $z_1, z_2, z_3, z_4 \in \mathbb{C}$. The imaginary part of $[z_1, z_2, z_3, z_4]$ is positive if and only if z_4 lies in the circumdisk of the triangle (z_1, z_2, z_3).*

This theorem is another consequence of the former geometric interpretation because $0 \leqslant \beta < \alpha \leqslant \pi$ when z_4 lies in the circumdisk of the triangle (z_1, z_2, z_3).

The cross-ratio is invariant under homographies, which are the maps of the form $z \mapsto (az+b)/(cz+d)$. In particular, it is invariant under the orientation-preserving isometries of the Poincaré disk $\mathbb{H}^2$ onto itself. The cross-ratio of an edge $e \in T$ can thus be defined from any of its lifts in $\mathbb{H}^2$. Let $\widetilde{e} = (\widetilde{u_1}, \widetilde{u_3})$ be a lift of e and $\widetilde{u_2}, \widetilde{u_4}$ be the remaining vertices of the triangles incident to e, such that $\widetilde{u_1}, \widetilde{u_2}, \widetilde{u_3}, \widetilde{u_4}$ are oriented counter-clockwise. The cross-ratio of e in T is $R_T(e) = [\widetilde{u_1}, \widetilde{u_2}, \widetilde{u_3}, \widetilde{u_4}]$. Theorem I.42 implies that the edge e is locally non-Delaunay if and only if the imaginary part of $R_T(e)$ is positive, which makes the cross-ratio a convenient tool for the flip algorithm.

However, cross-ratios alone are not enough to retrieve a portion of the lift $\widetilde{T}$ of T. In addition to cross-ratios, this data structure contains one anchor, which represents a lift of a triangle in the Poincaré disk. The *anchor* $(\delta, \widetilde{v_0}, \widetilde{v_1}, \widetilde{v_2})$ is composed of a dart δ representing the triangle in the combinatorial map, and of the coordinates of the three vertices $\widetilde{v_0}, \widetilde{v_1}, \widetilde{v_2}$ of a lift of the triangle. The dart corresponds to the edge (v_0, v_1).

Knowing a lift of a triangle, its neighbors can be retrieved using the cross-ratios of its edges: if $(\widetilde{u_1}, \widetilde{u_2}, \widetilde{u_3})$ and $(\widetilde{u_1}, \widetilde{u_3}, \widetilde{u_4})$ are two triangles in $\mathbb{H}^2$ sharing the edge $\widetilde{e} = (\widetilde{u_1}, \widetilde{u_3})$, the coordinates of $\widetilde{u_4}$ can be deduced from the coordinates of $\widetilde{u_1}, \widetilde{u_2}, \widetilde{u_3}$ and $R_T(\widetilde{e})$. The anchor thus serves as a starting point to iteratively compute a portion of $\widetilde{T}$, for example to draw a lift of each triangle of T.

The data structure is easily maintained, in constant time, after the flip of an edge common to two triangles [DDKT22, Sec 3.2]. Let $t = (u_0, u_1, u_2)$ and $t' = (u_0, u_2, u_3)$ be two triangles sharing the edge (u_0, u_2) (Figure I.23). After the flip of this edge, the β_1 pointers of each dart that used to belong to t or t' are updated. The cross-ratio of the newly created edge (u_1, u_3) is computed and the cross-ratios of the edges that have not been flipped, (u_0, u_1), (u_1, u_2), (u_2, u_3) and (u_3, u_0), are updated. These five cross-ratios are computed using formulas involving the values of the cross-ratios of the five edges of t and t' before the flip. Note that it is not necessary to know a lift of these vertices to compute these cross-ratios. If the anchor of the triangulation represents t or t', it is also updated so that it represents one of the new triangles incident to the flipped edge.

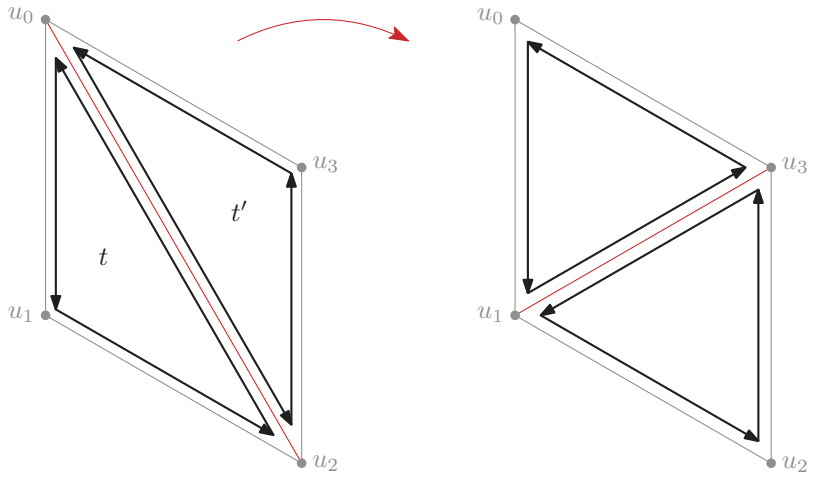

Figure I.23: Notations for the explanation of the update the data structure after a flip.

I.5 The CGAL library

CGAL stands for Computational Geometry Algorithms Library and is pronounced like the French word *cigale*. It is an open-source C++ library of computational geometry algorithms. It includes data structures and algorithms for combinatorics, convex hulls, polyhedra, triangulations, meshes, geometric optimization, and many more [The24]. The CGAL project started in 1996, founded by a consortium of eight European and Israeli research institutes. The goal of the project was to concentrate computational geometry data structures and algorithms in a common software library to provide reliable and efficient programs for academic or industrial users [FT06, Sec 1].

The components (packages) of the CGAL library are distributed under either the LGPL (GNU Lesser General Public License) v3+ or the GPL (GNU General Public License) v3+. A commercial license is provided by the company GeometryFactory[3] since 2003, which also maintains the software and provides support for users.

[3] https://geometryfactory.com/

I.5.1 Paradigms followed by CGAL

In this section, we explain the paradigms followed by the CGAL library. This will allow the reader to become more familiar with CGAL and the implementation details that will be given in Chapter IV.

Object-oriented programming

Object-oriented programming is a programming paradigm centered around objects and classes. A *class* serves as a blueprint for creating objects, defining a user-defined data type. An *object* is an instance of a class, encapsulating data described by *attributes* and behaviors in the form of functions, called *methods*, that can be applied to the objects. The attributes can be of either built-in type provided by the programming language or user-defined type provided by another class. Methods can accept *arguments*, which are similar to variables in mathematical functions. Each class includes one or more *constructor* methods, which initialize an object by setting the initial values of its attributes.

For example, an object of the class `Hyperbolic_isometry_2` [DDPT25], representing a hyperbolic isometry of $\mathbb{H}^2$, has only one attribute, which is a list of its four coefficients (of type `Complex`). A `Hyperbolic_isometry_2` object can be initialized with a constructor taking four `Complex` as arguments. The class provides a method that evaluates the hyperbolic isometry at a given point (of type `Point`). Snippet I.1 shows a syntax example for creating objects and using methods in C++.

Snippet I.1: Syntax example for creating objects and using methods in C++.

```
// construct complex numbers from their real and imaginary parts
Complex zero = Complex(0, 0);
Complex one = Complex(1, 0);
Hyperbolic_isometry_2 hyperbolic_isometry = Hyperbolic_isometry_2(one, zero
    , zero, one);  // construct the identity
Point p = hyperbolic_isometry.evaluate(Point(0.5, 0.5)); // evaluate the
    isometry at a given point and store it in the variable p
```

A fundamental concept in object-oriented programming is inheritance. When a class `B` *inherits* from another class `A`, it gains access to its attributes and methods. We say that `B` is a *child* of `A` and that `A` is the *parent* of `B` [4]. A child class can introduce new attributes and methods that are not present in the parent class. It therefore refines the parent class while extending its functionalities. Additionally, the child can overload any method of the parent by redefining it. This allows it to use its own implementation of the method instead of the one provided by the parent.

Generic programming

Generic programming is an abstraction that is complementary to object-oriented programming. Its purpose is to design algorithms or data structures that do not depend on a specific data type. A generic program runs with any data type which supports the operations used in the program. This avoids having to write the same code for several data types, resulting in a versatile and more maintainable code. For example, an algorithm that returns the maximum of two elements works with any two objects that are of the same

[4] While C++ allows a class to inherit from multiple parents (multiple inheritance), this practice is often discouraged. We therefore refer to *the* parent class to reflect good practice. Many other languages, such as Java and C#, do not support multiple inheritance.

type and can be compared. It can therefore be applied to `int`, `float` or `char` data types (for example), or to any user-defined type that provides a comparison operator.

In C++, generic programming is achieved via the `template` and `typename` keywords. We write `template<typename T>` before a class or a function to indicate that `T` is a generic type. We say that the class or the function is *templated on* `T`. Snippet I.2 shows a minimal example of a generic function that returns the max of two elements of generic type. When no confusion arises, template parameters will be omitted for readability.

Snippet I.2: Minimal example of a generic function `max` that returns the maximum of two elements.

```
#include<iostream>  // to display the results

// T is a generic type
template<typename T> T max(T a, T b) {
    if(a > b)  // works only if '>' is defined for T
        return a;
    else
        return b;
}

int main(){
    // calls max with int type
    std::cout << max(2, 3) << std::endl;  // displays 3
    // calls max with char type
    std::cout << max('b', 'c') << std::endl; // displays c
}
```

Generic programming is at the core of the CGAL library because it is a natural tool to separate the topology and the geometry of objects used in algorithms or data structures [FT06, Sec 3]. This separation is achieved by making them independent of the specific geometry of objects through the use of a geometric traits class as a template in the main classes of the program. A *geometric traits* class implements a set of required behaviors for geometric objects. For instance, it may include functions that determine the orientation of three points or compute the intersection of two lines. This set of requirements is called a *concept*. Any traits class implementing a concept is called a *model* of this concept.

A geometric traits class is templated on a *geometric kernel* class, which provides basic geometric objects, such as points and lines, along with basic operations on them. In some cases, a kernel class can directly serve as a traits class if it fulfills the program's requirements. The kernel classes provided by CGAL may differ in their internal representation of objects or in the different functionalities they offer. For example, the kernel `Circular_kernel_2` provides functionalities on circles and circular arcs that we will need to work in the Poincaré disk.

Every CGAL kernel class operates with a number type, typically provided through a template. The number type can be built-in or supplied by CGAL, some external libraries (CORE [dtb], GMP [dtc], LEDA [dtd]) or defined by the user [HHPS24].

These layers of abstraction make CGAL programs flexible by allowing the user to customize the components of their program to suit their specific requirements. This overview is summarized in Snippet I.3.

Snippet I.3: Typical beginning of a CGAL program.

```
// the "typedef" keyword is used to alias data types to shorter or more
    meaningful names
typedef ANumberTypeClass        Number;  // represents numbers
typedef AKernelClass<Number>    Kernel;  // represents objects and basic
    operations on them
typedef ATraitsClass<Kernel>    Traits;  // implements required behaviors
    of objects
typedef AClass<Traits>          Class;   // uses objects and functions of
    Kernel and Traits

// some code
```

Exact Geometric Computation

The theoretical algorithms developed in computational geometry are most often designed to work in the real RAM model, where basic operations on real numbers are always exact and are executed in constant time. However, these assumptions do not hold in practice because arbitrary real numbers cannot be finitely represented on a machine. Floating point numbers are commonly used to replace real numbers in scientific computing. While using these numbers may produce inexact output due to rounding errors, these minor inaccuracies are often tolerated for many applications. Geometric algorithms are however particularly sensitive to such imprecision because they can cause the program to crash or produce significantly incorrect output. For instance, determining whether two segments intersect is a binary question, where the answer is either correct or entirely wrong, and there is no room for the slightest error.

In geometric algorithms, each step is either a construction or a conditional step based on the evaluation of a predicate. A construction is the *computation* of a new object, like the intersection point of two lines or the circumcenter of a triangle. A *predicate* is the evaluation of a binary question such as "are these two segments intersecting?" or "are these three points collinear, oriented clockwise or counterclockwise?", which generally reduces to computing the sign of an algebraic expression. The Exact Geometric Computation paradigm, followed by CGAL [5], states that the exact computation of these low-level construction and conditional steps guarantees the correctness of higher-level algorithms. In other words, when the basic operations performed by algorithms are guaranteed to be correct, algorithms can be implemented without worrying about precision issues and the resulting programs will produce correct output.

I.5.2 Hyperbolic geometry in CGAL

The first implementations of hyperbolic geometry in CGAL originate from the PhD thesis of Mikhail Bogdanov [BDT14], who implemented Delaunay triangulations and Voronoi diagrams in the Poincaré disk in the package 2D Hyperbolic Delaunay Triangulations [BIT24]. The main class of this package, `Hyperbolic_Delaunay_triangulation_2`, naturally inherits from the Euclidean Delaunay triangulation class `Delaunay_triangulation_2`. This package provides two hyperbolic traits classes that form the foundations of hyperbolic geometry in CGAL. In particular, they provide the construction of the circumcenter of triangles in the Poincaré disk, which is the central operation of our algorithm.

[5] https://www.cgal.org/exact.html

The "fundamental domain" data structure presented in Section I.4.1 has been implemented in the CGAL package 2D Periodic Hyperbolic Triangulations [IT19] by Iordan Iordanov and Monique Teillaud in 2017. This package provides Delaunay triangulations on the Bolza surface [IT17], which is the most symmetric surface of genus 2, characterized by the fact that one of its fundamental polygons is a regular (hyperbolic) octagon. The flip operation is not provided in this package, but the implemented data structure supports the insertion of vertices in the Delaunay triangulation using Bowyer's algorithm [Bow81, Wat81].

The "combinatorial map" data structure presented in Section I.4.2, together with the flip operation and a flip algorithm, has been recently implemented by Loïc Dubois in the CGAL package 2D Triangulations on Hyperbolic Surfaces [DDPT25] that will be part of the next release. However, this package does not include functionality for inserting new vertices into an existing triangulation. It relies on the CGAL Combinatorial Map package, which provides all the functionalities to manage the combinatorics of the data structure.

Chapter II

Walking in a triangulation in $\mathbb{H}^2$

A crucial step of our ε-net algorithm relies on finding a triangle containing a query point in a Delaunay triangulation of $\mathbb{H}^2$. There is only one such triangle, except if the query point lies on a vertex or an edge of the triangulation.

There are two classical algorithms in the Euclidean plane that can be adapted to the hyperbolic plane: the straight walk and the visibility walk. Both methods start from a triangle $\widetilde{t}_0$ of the triangulation to find the query point $\widetilde{q}$. The *straight walk* (Figure II.1, left) visits all triangles along the geodesic segment $\widetilde{pq}$, where $\widetilde{p}$ is a vertex of $\widetilde{t}_0$. The algorithm starts by rotating around $\widetilde{p}$ to find a triangle incident to $\widetilde{p}$ whose edge opposite to $\widetilde{p}$ intersects the geodesic segment $\widetilde{pq}$. The *visibility walk* (Figure II.1, right) consists, for each visited triangle not containing $\widetilde{q}$, of moving to a neighbor through an edge $\widetilde{e}$ if $\widetilde{q}$ and the third vertex of the visited triangle are on different sides of the geodesic line supporting $\widetilde{e}$. Note that the path taken by the visibility walk is not unique because some triangles may have two edges that meet the specified condition. In fact, the straight walk is a special case of the visibility walk.

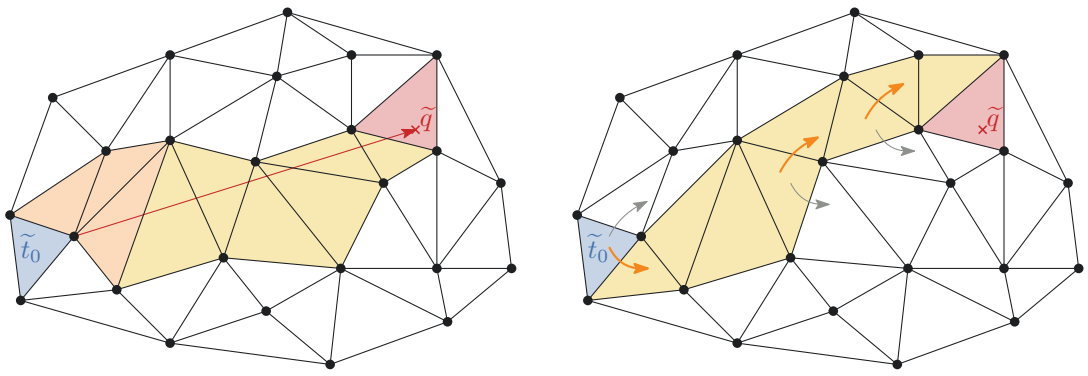

Figure II.1: The two discussed walking strategies in a Euclidean Delaunay triangulation using the same notation as in the description. Left: the straight walk. The triangles visited during the initialization phase are shown in orange. Right: a visibility walk. Arrows indicate the two options available in some triangles.

Since the number of triangles intersecting the line $\widetilde{pq}$ is finite, the straight walk always terminates in any triangulation, but it is more nuanced for the visibility walk. In the Euclidean case, the visibility walk terminates in a Delaunay triangulation [Ede90] but it can loop forever in a non-Delaunay triangulation [dFFNP91]. This short chapter is dedicated to proving the following theorem, which addresses the

hyperbolic case.

Theorem II.1. *The visibility walk terminates in a finite or periodic hyperbolic Delaunay triangulation of* $\mathbb{H}^2$.

For lighter notation, we drop the $\widetilde{\cdot}$ for objects in $\mathbb{H}^2$ and denote the length of a geodesic segment xy as xy instead of $d_{\mathbb{H}^2}(x, y)$ until the end of this chapter.

A proof of the Euclidean case [DH16, Cor 10] relies on the notion of power of a point with respect to a circle. Let $q, z \in \mathbb{R}^2$ and $\mathcal{C}(z, r)$ be the circle of radius $r > 0$ and centered at z. The power of q with respect to $\mathcal{C}(z, r)$ is $||qz||^2 - r^2$. It is positive when q lies outside the circle, negative when it lies inside, and zero when it lies on the circle. When q lies outside $\mathcal{C}(z, r)$, the power is $||qt||^2$ where t is the point at which one of the tangent lines of $\mathcal{C}(z, r)$ passing through q touches the circle (Figure II.2, left). When q lies inside, it is $-||qs||^2$ where s is the point of $\mathcal{C}(z, r)$ such that the triangle (s, q, z) is rectangle at q (Figure II.2, right).

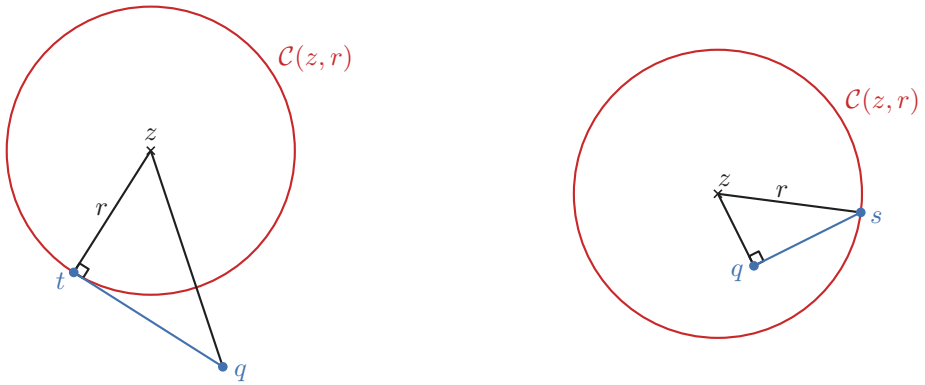

Figure II.2: Interpretation of the power of the point q with respect to the circle $\mathcal{C}(z, r)$ in Euclidean geometry.

This interpretation comes from Pythagoras' theorem. In order to maintain the same interpretation in hyperbolic geometry (recall Hyperbolic Pythagoras' Theorem (Theorem I.7)), we use the following definition for the power of a point with respect to a circle.

Definition II.2 (Power of a point)**.** The *power of a point q with respect to a circle* $\mathcal{C}(z, r)$ is $\cosh(zq)/\cosh(r)$. The *power of a point q with respect to a triangle* $t \in \mathbb{H}^2$ is

$$P(t, q) = \frac{\cosh(z_t q)}{\cosh(r_t)},$$

where r_t denotes the radius of the circumcircle of t and z_t its center.

Remark. The power of a point with respect to a circle is larger than 1 when the point lies outside the circle, smaller than 1 when it lies inside, and equal to 1 when it lies on the circle.

Theorem II.1 is a consequence of the following lemma. This lemma is analogous to the Euclidean case [DH16, Cor 10]. However, the Euclidean version relies on a property of the power of a point whose analog is false in hyperbolic geometry. The proof must therefore be entirely rewritten.

Lemma II.3. *During the visibility walk toward the point q in a Delaunay triangulation in $\mathbb{H}^2$, if a triangle t is encountered before its neighbor t', then $P(t, q) \geqslant P(t', q)$.*

Proof. Let z and z' be the respective centers of the circumcircles of t and t', and r and r' their respective radii. Call u and v the common vertices of t and t'. The geodesic line zz' is the bisector of the geodesic segment uv, therefore they intersect perpendicularly at their midpoint m. As illustrated in Figure II.3(left), Hyperbolic Pythagoras' theorem (Theorem I.7) yields

$$\cosh(r) = \cosh(vm)\cosh(zm) \text{ and } \cosh(r') = \cosh(vm)\cosh(z'm). \tag{II.1}$$

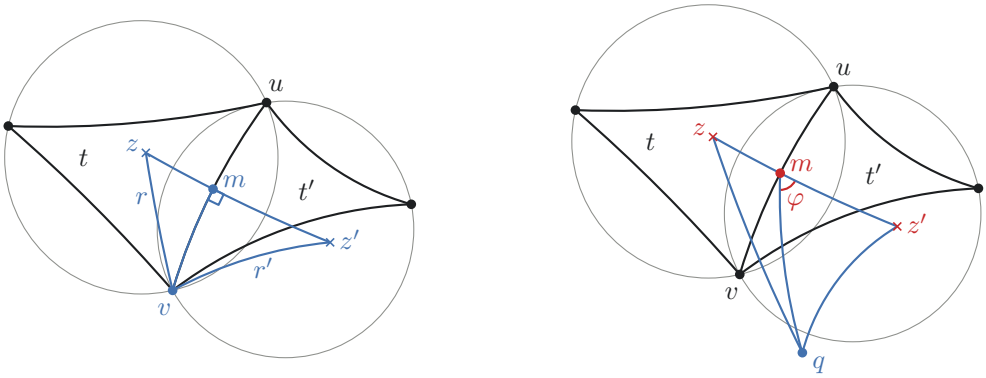

Figure II.3: Illustration of Equation II.1 (left) and of the first case of the proof (right).

When walking toward q, the point q lies on the same side of the geodesic segment uv as t' and it is not on uv. The geodesic line zz' can be oriented such that it crosses t before t'. The Delaunay property implies that z appears before z' for this orientation. Indeed, if z appears after z', then $r = zv > z'v = r'$ so the circumcircle of t contains the third point of t' in its interior. The triangle t is therefore non-Delaunay.

To prove that $P(t, q) \geqslant P(t', q)$, we distinguish three cases with respect to the position of the point m on the geodesic line zz'. Note that when $z = z'$, the inequality is an equality and the two triangles have the same circumcircle. In the following we can thus assume $z \neq z'$ and we will prove that the inequality is strict.

First case: m lies between z and z' This case is illustrated in Figure II.3 (right). Define φ as the unsigned angle $\angle(q, m, z')$. In the triangles (z, m, q) and (z', m, q), the hyperbolic law of cosines (Theorem I.6) yields

$$\begin{cases} \cosh(zq) = -\sinh(zm)\sinh(mq)\cos(\pi - \varphi) + \cosh(zm)\cosh(mq), \\ \cosh(z'q) = -\sinh(z'm)\sinh(mq)\cos(\varphi) + \cosh(z'm)\cosh(mq). \end{cases}$$

Since q and t' lie on the same side of the line uv, $\varphi < \pi/2$. Then, $\cos(\varphi) > 0$ and $\cos(\pi - \varphi) < 0$ and it follows that

$$\frac{\cosh(zq)}{\cosh(zm)} > \cosh(mq) > \frac{\cosh(z'q)}{\cosh(z'm)}.$$

We obtain $P(t, q) > P(t', q)$ using Equation II.1.

Second case: z lies between m and z' This case is illustrated in Figure II.4 (left). The same formula now yields

$$\begin{cases} \cosh(zq) = -\sinh(zm)\sinh(mq)\cos(\varphi) + \cosh(zm)\cosh(mq), \\ \cosh(z'q) = -\sinh(z'm)\sinh(mq)\cos(\varphi) + \cosh(z'm)\cosh(mq). \end{cases}$$

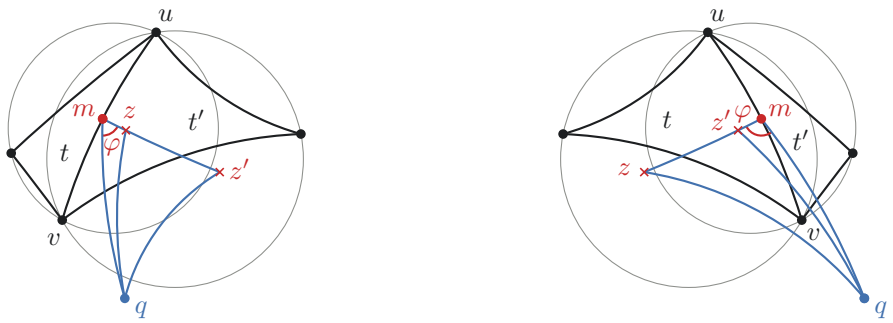

Figure II.4: Illustration of the second (left) and third (right) cases of the proof.

It follows that

$$\begin{cases} \dfrac{\cosh(zq)}{\cosh(zm)} = -\tanh(zm)\sinh(mq)\cos(\varphi) + \cosh(mq), \\ \dfrac{\cosh(z'q)}{\cosh(z'm)} = -\tanh(z'm)\sinh(mq)\cos(\varphi) + \cosh(mq). \end{cases}$$

As in the previous case, the angle $\varphi < \pi/2$. Moreover, $zm < z'm$ and $\tanh$ is increasing. We divide both equations by $\cosh(vm)$ and obtain $P(t,q) > P(t',q)$.

Third case: z' lies between m and z This case is illustrated in Figure II.4 (right). In this case, the same formula yields the same equations as in the previous case. Since $\varphi \in (\pi/2, \pi]$ and $zm > z'm$, we conclude in a similar way. □

The termination of the walk is thus clear when the triangulation is finite. We use a compactness argument for the case of an infinite periodic Delaunay triangulation in $\mathbb{H}^2$ given by the lifts of a triangulation of a hyperbolic surface. Indeed, there is a finite number of circumradii, so they are upper bounded by a value R. For any triangle t with circumcircle $\mathcal{C}(z,r)$ visited during the walk, we have

$$P(t_0,q) \geqslant P(t,q) = \frac{\cosh(zq)}{\cosh(r)} \geqslant \frac{\cosh(zq)}{\cosh(R)},$$

where t_0 is the triangle in which the walk starts. This implies that z is in a bounded region of $\mathbb{H}^2$, so the walk must terminate in a finite number of steps.

Chapter III

The ε-net algorithm

This chapter introduces our algorithm for computing an ε-net of a hyperbolic surface. We start with a presentation of our inspiration: Delaunay refinement algorithms. The core idea is to begin with a Delaunay triangulation of the input surface, initially consisting of a single vertex. We then refine this triangulation by inserting the circumcenter of each Delaunay triangle whose circumradius is larger than ε. We explain our preference for using an incremental flip algorithm over Bowyer's algorithm to maintain the Delaunay triangulation.

Before detailing our algorithm, we establish bounds on the number of points in an ε-net. We then describe our algorithm using the "fundamental domain" data structure (recall Section I.4.1). We prove the termination of the algorithm and analyze its complexity. We detail how the algorithm is adapted to the "combinatorial map" data structure in preparation for implementation (recall Section I.4.2).

We will see that the number of points in an ε-net of a hyperbolic surface depends on its systole: the shorter the systole, the longer the collars and the more points there are. To handle this situation, we extend the ε-net algorithm to compute what we define as a pseudo ε-net.

III.1 Design choices

In this section, we first outline the principle of Delaunay refinement, which is the inspiration for our algorithm. We then explain our preference for using a flip algorithm over Bowyer's algorithm to restore the Delaunay property after each insertion.

III.1.1 Delaunay refinement

Our algorithm to compute an ε-net of a hyperbolic surface is inspired by Delaunay refinement. This is a technique, based on Delaunay triangulations, for generating triangular meshes with nicely shaped triangles. Such triangular meshes are required for applications such as the finite element method, which is used in engineering and mathematical modeling. The ratio between the circumradius and the smallest angle of a triangle is a natural measure of the quality of a Euclidean triangle [She02, Sec 3]. Delaunay refinement algorithms aim to improve this ratio. To achieve this, Chew [Che89] and Ruppert [Rup95] proposed algorithms in which the main operation is the insertion of the circumcenter of a triangle of poor quality in a Delaunay triangulation.

Chew's algorithm applies to bounded regions of the Euclidean plane. It starts from the Delaunay triangulation of a set of points satisfying some conditions on the input region. Then, the circumcenter of a Delaunay triangle of circumradius larger than an input parameter h is inserted and the Delaunay triangulation is recomputed. This process is repeated until the circumradius of all triangles is at most h. Chew showed that this algorithm results in a set of points such that two points are at least h apart, making it an h-packing. Furthermore, every triangle in the output triangulation has a circumradius of at most h; we will later show that this property implies the set is also an h-covering. We adapt this algorithm to a hyperbolic surface in order to obtain an ε-net.

III.1.2 Restoring the Delaunay property

Since our ε-net algorithm consists of iteratively inserting points in a Delaunay triangulation, we need to choose an incremental Delaunay triangulation algorithm to restore the Delaunay property after each insertion. We previously mentioned the incremental flip algorithm and Bowyer's algorithm in Section I.2. In this section, we explain why we do not choose Bowyer's algorithm to manage the Delaunay triangulation throughout our ε-net algorithm.

Bowyer's algorithm has been generalized to the Bolza surface by Iordanov and Teillaud [IT17]. Their implementation is part of CGAL [IT19]. The authors used a data structure similar to the "fundamental domain" data structure (recall Section I.4.1), where the input fundamental domain for the Bolza surface is a regular octagon.

The correctness of Bowyer's algorithm in the Euclidean plane relies on the fact that the conflict zone is always a topological disk. However, when generalizing the algorithm to surfaces, this condition may not always hold (Figure III.1). To address this issue for the Bolza surface, Iordanov and Teillaud used 14 dummy points to ensure that the conflict zone is always a topological disk. These points are used to compute an initial Delaunay triangulation, into which query points are then inserted iteratively. At the end of the algorithm, the dummy points are removed if possible, resulting in the desired Delaunay triangulation of the query points.

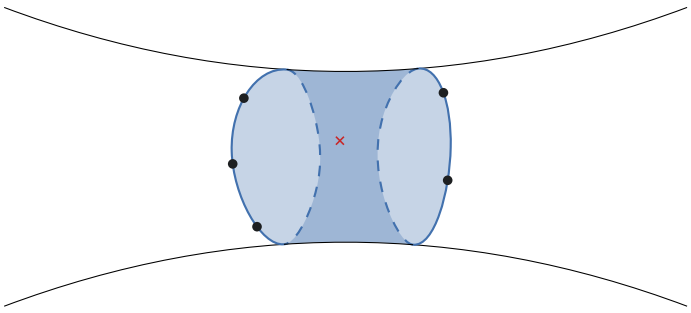

Figure III.1: The conflict zone (colored) of the to-be inserted point (cross) might not be a topological disk on a surface.

There are two reasons why adapting this approach to any hyperbolic surface is challenging. First, the set of dummy points is chosen in order to satisfy a condition involving the systole. The systole is known for the Bolza surface and generalized Bolza surfaces [EITV19], but this is not typically the case for other hyperbolic surfaces. Second, even if we knew the systole of our input surface, it would still be unclear how to choose the appropriate set of dummy points for Bowyer's algorithm to function properly. In fact,

an ε-net with a well-chosen ε could be this set of dummy points. We therefore use the incremental flip algorithm to manage the Delaunay triangulation throughout our ε-net algorithm.

III.2 Number of points in an ε-net

We already know from Theorem I.39 (2) that an ε-packing of a hyperbolic surface S is finite. This section provides an upper bound on the number of points in an ε-packing of S and two lower bounds on the number of points in an ε-covering of S.

Theorem III.1. *Any ε-packing of a closed hyperbolic surface S of genus g and systole σ contains at most*

$$16(g-1)\left(\frac{1}{\varepsilon^2}+\frac{1}{\sigma^2}\right)$$

points. Moreover, if $\varepsilon \leqslant \sigma$, then the upper bound is $16(g-1)/\varepsilon^2$.

Remark. The case when $\varepsilon \leqslant \sigma$ corresponds to the situation where S has no ε-thin part (recall Definition I.25).

Proof. Let P be an ε-packing of S. The open balls of radius $\varepsilon/2$ centered at the points of P on the ε-thick part $S_\varepsilon^{\text{Thick}}$ are isometric to disks in $\mathbb{H}^2$ and are pairwise disjoint, so their union does not cover S. The area of such a disk is $4\pi\sinh^2(\varepsilon/4)$ by Theorem I.3 and the area of S is $4\pi(g-1)$ by Corollary I.14. The number N^T of points in P on the ε-thick part $S_\varepsilon^{\text{Thick}}$ thus satisfies

$$4\pi(g-1) \geqslant N^T 4\pi\sinh^2\left(\frac{\varepsilon}{4}\right) \geqslant N^T\pi\frac{\varepsilon^2}{4}$$

because $\sinh x \geqslant x$ for all $x \geqslant 0$. It follows that

$$N^T \leqslant \frac{16(g-1)}{\varepsilon^2}. \tag{III.1}$$

The open balls of radius $\varepsilon/2$ on the ε-thin part $S_\varepsilon^{\text{thin}}$, if it exists, that is if $\sigma \leqslant \varepsilon$, are also pairwise disjoint, but they are not isometric to disks in $\mathbb{H}^2$. However, by Theorem I.23, the open balls of radius $\sigma/2$ are isometric to disks in $\mathbb{H}^2$. We can apply the reasoning that led to inequality (III.1) for σ instead of ε and obtain a bound on the number of points in P on the thin part $S_\varepsilon^{\text{thin}}$: $N^t \leqslant 16(g-1)/\sigma^2$. The bound on the total number of points of P follows. □

Remark. In fact, the proof shows that $(g-1)\left(1/\sinh^2(\varepsilon/4)+1/\sinh^2(\sigma/4)\right)$ is a more precise upper bound. However, we chose to state the theorem in this simplified form because we are only interested in the order of magnitude of the number of points.

When it comes to the number of points in an ε-covering of S, we provide two lower bounds, one involving g and the other involving σ.

Theorem III.2. *Any ε-covering of a closed hyperbolic surface S of genus g and systole σ contains at least*

$$\frac{g-1}{\sinh^2(\varepsilon/2)} \text{ and } \frac{1}{2\varepsilon}\operatorname{arsinh}\left(\frac{1}{\sinh(\sigma/2)}\right)$$

points.

Remark. For a fixed ε, the right-hand lower bound is equivalent to $\ln(1/\sigma)$ when $\sigma \to 0$.

Proof. Let P be an ε-covering of S. We first prove the left-hand lower bound. The disks of radius ε centered at the points in P cover S. Since their area is at most $4\pi \sinh^2(\varepsilon/2)$ (it is smaller for points on the ε-thin part) and S has area $4\pi(g-1)$, the number N of points in P satisfies

$$4\pi(g-1) \leqslant 4\pi N \sinh^2(\varepsilon/2),$$

hence the left-hand lower bound.

We now prove the right-hand lower bound. Every disk of radius ε centered at a point in P is intersected at most once by a distance path between two points realizing the diameter $\mathrm{Diam}(S)$ of S. The length of the portion of the path included in such a disk is at most 2ε. It follows that $\mathrm{Diam}(S) \leqslant 2\varepsilon N$. By definition of the collar $\mathcal{C}(\sigma)$ of σ, there exists a point x on the boundary of $\mathcal{C}(\sigma)$ and a point $y \in \sigma$ such that $d_S(x,y) = w(\sigma) = \operatorname{arsinh}(1/\sinh(\sigma/2))$. Therefore, $\mathrm{Diam}(S) \geqslant \operatorname{arsinh}(1/\sinh(\sigma/2))$ and the lower bound follows. □

These bounds show that the number of points in an ε-net of a hyperbolic surface depends both on the topology of the surface (via its area, therefore its genus), and on its geometry (via its systole). Moreover, it grows arbitrarily as the systole decreases to zero.

III.3 Using the "fundamental domain" data structure

In this Section, we give the details of our ε-net algorithm using the "fundamental domain" data structure presented in Section I.4.1. We then prove that the algorithm terminates and outputs an ε-net as expected. We then analyze its complexity with respect to the properties of the input surface and the parameter ε.

III.3.1 Description of the algorithm

Let $\varepsilon > 0$ and S be a hyperbolic surface. The input of the algorithm consists of a Delaunay triangulation of S with a single vertex b, together with the Dirichlet domain $\mathcal{D}(\widetilde{b})$ of a lift $\widetilde{b}$ and the group Γ generated by side-pairings. This does not induce any loss of generality because such an input can be computed from any fundamental domain [DKPT23]. The output is a Delaunay triangulation whose set of vertices is an ε-net of S.

Let k be the number of sides of $\mathcal{D}(\widetilde{b})$ and $s_1, \dots, s_k$ be its sides. For a side s_j, the isometry pairing s_j to its paired side is denoted by γ_j. A Delaunay triangulation of a set of points $P \subset S$ is denoted by $DT(P)$.

We use the "fundamental domain" data structure presented in Section I.4.1 to store the triangulation throughout the algorithm. We denote $\widetilde{D}_o$ as the original domain of $\mathcal{D}(\widetilde{b})$ such that every vertex p has constant-time access to its lift $\widetilde{p}_o \in \widetilde{D}_o$ and that, for every triangle t, a corresponding lift $\widetilde{t}_o$ with at least one vertex in $\widetilde{D}_o$ can be computed in constant time via three isometries accessible by t.

In a first step, a set of points is initialized as $P_1 = \{b\}$. At each step $i \geqslant 2$, the algorithm inserts the circumcenter c of a triangle t^ε whose circumradius is greater than ε. We say that t^ε is a *large* triangle. The set of points is updated as $P_i = P_{i-1} \cup \{c\}$ as well as the Delaunay triangulation $DT(P_i)$. To do so, several operations are needed.

First, the circumradius of $\widetilde{t}_o$ for every triangle t of $DT(P_{i-1})$ is computed, until a large triangle t^ε is found[6]. To our knowledge, there is no known formula for directly computing the hyperbolic circumradius of a triangle in the Poincaré disk. It is thus obtained by first computing its circumcenter, as the intersection of the bisectors of its edges [BDT14], and then computing the distance between the circumcenter and one of the vertices of the triangle using Theorem I.2. The circumcenter $\widetilde{c}$ of the lift $\widetilde{t}_o^\varepsilon$ is a lift of c, but it does not necessarily lie in $\widetilde{D}_o$. This can be checked by testing whether $\widetilde{b}$ and $\widetilde{c}$ lie on the same side of the supporting line of each side of $\widetilde{D}_o$.

To insert c into $DT(P_{i-1})$, the lift $\widetilde{c}_o$ of c that lies in $\widetilde{D}_o$ has to be computed. If $\widetilde{c}$ lies in $\widetilde{D}_o$, then $\widetilde{c}_o = \widetilde{c}$. Otherwise, the algorithm walks in the tiling $\{\gamma\widetilde{D}_o\}_{\gamma\in\Gamma}$ of $\mathbb{H}^2$ along the geodesic segment $[\widetilde{x}\widetilde{c}]$, where $\widetilde{x}$ is a vertex of $\widetilde{t}_o^\varepsilon$ in $\widetilde{D}_o$ (Figure III.2). This walk is similar to the straight walk presented in Chapter II, except that the walk is done in the tiling of $\mathbb{H}^2$ by Dirichlet domains instead of a triangulation. The first copy of $\widetilde{D}_o$ traversed by $[\widetilde{x}\widetilde{c}]$ is found by looking for the side $s_{j_1}, j_1 \in \{1,\dots,k\}$ of $\widetilde{D}_o$ intersecting it. Since the sides of $\widetilde{D}_o$ are geodesic segments, this amounts to checking whether two geodesic segments intersect: for two geodesic segments $[\widetilde{x}_1\widetilde{x}_2]$ and $[\widetilde{y}_1\widetilde{y}_2]$, we check whether $\widetilde{x}_1$ and $\widetilde{x}_2$ lie on opposite sides of the supporting geodesic line of $[\widetilde{y}_1\widetilde{y}_2]$, and we run the same test, swapping the roles of x and y. The walk along $[\widetilde{x}\widetilde{c}]$ continues in $\gamma_{j_1}\widetilde{D}_o$, and so on, until the copy $\gamma_{j_n}\dots\gamma_{j_1}\widetilde{D}_o$ containing $\widetilde{c}$ is found. Then $\widetilde{c}_o = \gamma_{j_1}^{-1}\dots\gamma_{j_n}^{-1}\widetilde{c}$. Note that the walk remains well-defined when $[\widetilde{x}\widetilde{c}]$ goes through a vertex of a copy of $\widetilde{D}_o$.

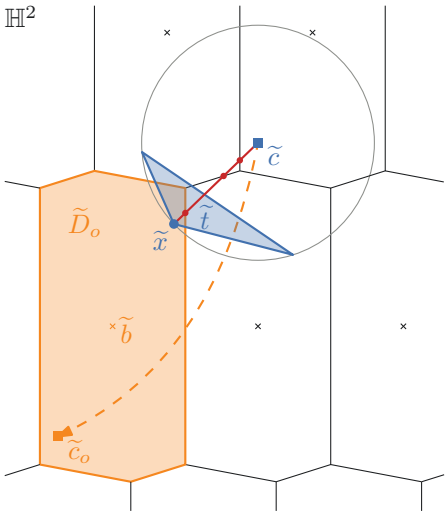

Figure III.2: Schematic representation of the walk in the tiling $\{\gamma\widetilde{D}_o\}_{\gamma\in\Gamma}$ of $\mathbb{H}^2$ to locate the circumcenter of a triangle.

The Delaunay triangulation $DT(P_i)$ of $P_i = P_{i-1}\cup\{c\}$ can then be computed. First, the triangle t_c of $DT(P_{i-1})$ containing c is found by naively checking if $\widetilde{c}_o$ lies in one of the (at most three) lifts of each triangle t in $DT(P_{i-1})$ having a vertex in $\widetilde{D}_o$. This can be done by testing, for each edge, whether $\widetilde{c}_o$ and the third vertex of the triangle lie on the same side of its supporting line. Then t_c is split into three by creating an edge between c and its three vertices. In the data structure, the three isometries stored in each new triangle are $\mathbb{1}_\Gamma$ for c, and the corresponding isometries in t_c for the other two vertices. Then $DT(P_i)$ is computed with a sequence of flips and the data structure is updated as described in Section I.4.1.

[6]Of course, a priority queue could be used to improve the complexity of this search. Since this is not the dominant operation in the algorithm, we accept a linear complexity for simplicity.

III.3.2 Correctness of the algorithm

The termination of the algorithm is straightforward. Each step $i \geqslant 2$ terminates since the triangulation is finite and the geodesic along which the walk takes place in the location process has finite length. At step 1, the ε-packing P_1 consists of one point. At each step $i \geqslant 2$, the new point c is the circumcenter of a large triangle t of $DT(P_{i-1})$. It is then at distance greater than ε from the vertices of t. Moreover, since t is a Delaunay triangle, its circumcircle is empty so c is at distance greater than ε from any other point of P_{i-1}. By induction, $P_i = P_{i-1} \cup \{c\}$ is an ε-packing containing i points. By Theorem III.1, the algorithm must terminate after a finite number $N-1$ of insertions. It returns an ε-packing P_N of cardinality N.

It remains to show that P_N is an ε-covering of S. Let x be a point on S. It lies in a triangle t of $DT(P_N)$. Let $\widetilde{t}$ be a lift of t and $\widetilde{x}$ the lift of x lying in $\widetilde{t}$. The circumdisk of $\widetilde{t}$ has a radius $r \leqslant \varepsilon$. By Lemma III.3 (below), there is a vertex of $\widetilde{t}$ whose distance to $\widetilde{x}$ is at most r. Since this vertex is a lift of a point of P_N, it follows that $d_S(x, P_N) \leqslant \varepsilon$. Therefore, P_N is an ε-covering.

Lemma III.3. *Let $\widetilde{t}$ be a triangle of $\mathbb{H}^2$ and $\widetilde{x} \in \widetilde{t}$. Define r as the radius of the circumcircle of $\widetilde{t}$. There exists a vertex of $\widetilde{t}$ whose distance to $\widetilde{x}$ is at most r.*

This lemma is also true in the Euclidean plane because the following proof uses arguments that work in both the Euclidean and the hyperbolic planes.

Proof. Let $D_{\widetilde{x}}$ be the closed disk of radius r centered in $\widetilde{x}$, $D_{\widetilde{t}}$ be the circumdisk of $\widetilde{t}$, and $\widetilde{c}$ be its circumcenter. If $\widetilde{x} = \widetilde{c}$, the two disks are equal and the result is trivial.

Suppose that $\widetilde{x} \neq \widetilde{c}$. Define $C_{\widetilde{t}}$ (resp. $C_{\widetilde{x}}$) as the circle bounding the disk $D_{\widetilde{t}}$ (resp. $D_{\widetilde{x}}$). The distance between $\widetilde{x}$ and $\widetilde{c}$ is at most r because $\widetilde{x}$ lies in $\widetilde{t}$, so $\widetilde{x} \in D_{\widetilde{t}}$ and $\widetilde{c} \in D_{\widetilde{x}}$. Therefore, the two disks intersect and their intersection is not a singleton. Moreover, $D_{\widetilde{t}} \neq D_{\widetilde{x}}$, so $C_{\widetilde{t}}$ and $C_{\widetilde{x}}$ intersect exactly twice (Figure III.3, left). The three vertices of $\widetilde{t}$ lie on the circle $C_{\widetilde{t}}$. If their distance to $\widetilde{x}$ is greater than r, that is, if they belong to $C_{\widetilde{t}} \setminus D_{\widetilde{x}}$, then the geodesic passing through the two intersection points of the circles separates $\widetilde{x}$ from $\widetilde{t}$. In this case, $\widetilde{x}$ cannot be in $\widetilde{t}$ (Figure III.3, right). Therefore, at least one vertex of $\widetilde{t}$ must lie on $C_{\widetilde{t}} \cap D_{\widetilde{x}}$, so at distance at most r from x. □

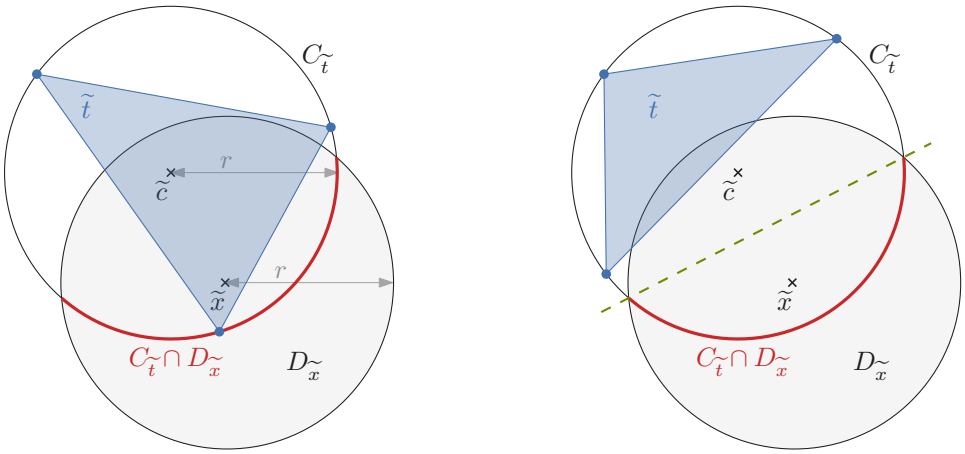

Figure III.3: Left: Illustration of the notations of the proof of Lemma III.3 in Euclidean geometry (for simplicity). Right: If no vertex of $\widetilde{t}$ lies on $C_{\widetilde{t}} \cap D_{\widetilde{x}}$ (bold red), then the dashed geodesic separates $\widetilde{x}$ from $\widetilde{t}$.

III.3.3 Complexity analysis

Since our algorithm involves real numbers, we analyze the complexity of our ε-net algorithm in the *real RAM model*. This is a model of computing in which the standard arithmetic operations (addition, subtraction, multiplication, division and comparison) are performed in constant time, written $O(1)$ time in Landau's notation (also called *Big O* notation).

Theorem III.4. *Let S be a hyperbolic surface and $\varepsilon > 0$. The ε-net algorithm computes an ε-net in*

$$O\left(\frac{1}{\varepsilon^4}\right)$$

time with the "fundamental domain" data structure.

In the proof of this theorem, we detail the complexity analysis in terms of constant-time operations, called *elementary operations*. These details clarify which surface parameters contribute to the constant factor hidden in the Big O notation. We consider the following operations as elementary operations in the real RAM model:

- For a triangle in the data structure, computing one of its (at most three) lifts having a vertex in $\widetilde{D}_o$ (see Section I.4.1 for notations);
- Computing the image of a point by a side-pairing isometry;
- Computing the center or the radius of the circumcircle of a triangle in $\mathbb{H}^2$;
- Deciding if a point lies on the right or the left side of an oriented geodesic segment in $\mathbb{H}^2$;
- Flipping an edge of a triangle (see Section I.4.1 for details on how the data structure is maintained after a flip).

Proof. At the beginning of a step $i \geqslant 2$, P_{i-1} contains $i-1$ points. By a classical reasoning of graph theory, Euler's formula for graphs embedded on surfaces shows that $DT(P_{i-1})$ has $2i+4g-6$ triangles. Since a lift of a triangle must be computed before computing its circumradius, finding a large triangle t^ε costs at most $2(2i+4g-6)$ elementary operations. Computing its circumcenter $\widetilde{c}$ is an elementary operation.

Let k be the number of sides of $\widetilde{D}_o$. Determining whether $\widetilde{c}$ lies in (a given copy of) $\widetilde{D}_o$ thus requires at most k elementary operations. The algorithm tests the copies of $\widetilde{D}_o$ that intersect the geodesic segment $\widetilde{x}\widetilde{c}$. Since $\widetilde{t}_o^{\,\varepsilon}$ is a triangle of $DT\left(\widetilde{P}_{i-1}\right)$, its circumcircle does not contain any other copy of $\widetilde{x}$, so $\widetilde{x}$ is the closest copy of $\widetilde{x}$ to $\widetilde{c}$. The geodesic segment $\widetilde{x}\widetilde{c}$ is thus a distance path. By Theorem I.35 and Theorem I.36, $\widetilde{x}\widetilde{c}$ intersects at most $2k$ sides of copies of $\widetilde{D}_o$. If an intersection occurs at a vertex of degree d of a copy of $\widetilde{D}_o$, then this counts for d intersections. If $\widetilde{x}\widetilde{c}$ has a subarc in common with an edge of a Dirichlet domain, then it must contain it because $\widetilde{x}\widetilde{c}$ is a geodesic segment. Moreover, it can only contain one copy of this edge, otherwise it would not project on a distance path on S. This case counts as 2 intersections and the upper bound of $2k$ intersections remains valid. Additionally, each traversed copy of $\widetilde{D}_o$ must be computed as the image by a side-pairing isometry of the previous copy, which costs at most k elementary operations. Searching the copy of $\widetilde{D}_o$ containing $\widetilde{c}$ thus requires at most $(2k)^2 + k = 4k^2 + k$ elementary operations. Computing $\widetilde{c}_o$ requires at most $2k$ elementary operations since it is obtained as the image of the composition of at most $2k$ side-pairing isometries. Recall that $k \leqslant 12g-6$ (Section I.1.3). The cost of locating $\widetilde{c}$ and computing $\widetilde{c}_o$ is thus at most $4k^2 + 3k \leqslant 4(12g-6)^2 + 3(12g-6)$ elementary operations.

When $\widetilde{c}_o$ is known, finding the triangle $t_c \in DT(P_{i-1})$ that contains c requires at most $12(2i+4g-6)$ elementary operations since it amounts to checking whether it lies, for each triangle, in its (at most) three lifts having a vertex in $\widetilde{D}_o$, and these lifts must be computed. Updating the data structure when splitting the triangle containing c into three is done in eight elementary operations (deleting the triangle that contains c, adding c to the list of vertices, creating 3 triangles and 3 isometries).

Summing the above costs, step i (without the flips) requires at most $28i+576g^2-484g+51$ elementary operations. We sum these costs from $i=2$ to $i=N$ (the number of points in the output ε-net) and add the cost of the last step consisting of checking all the $2N+4g-6$ triangles when they all have circumradius at most ε. Excluding the cost of flips, the algorithm requires at most $14N^2+N(576g^2-484g+69)-576g^2+488g-83$ elementary operations.

The number of flips is counted globally for all the steps in Lemma III.5 below. Since a flip costs one elementary operation, we obtain a total complexity for the ε-net algorithm of at most

$$(8C_h \operatorname{Diam}(S)^{6g-4}+14)N^2+N(576g^2-484g+69)-576g^2+488g-83$$

elementary operations, where C_h is a constant depending on the metric h (see Lemma III.5) of S and $\operatorname{Diam}(S)$ is its diameter. The proof is concluded with Theorem III.1. □

The proof of Theorem III.4 shows that the constant factor hidden in the Big O notation depends on several parameters of the surface: its diameter, its systole (influencing the number of points), its genus, and a constant C_h (see Lemma III.5). The surface parameters hidden in C_h are unfortunately not known. It could involve the parameters mentioned previously or other parameters of the surface.

Lemma III.5. *Let S be a hyperbolic surface of genus g and diameter $\operatorname{Diam}(S)$. The total number of flipped edges during the execution of the ε-net algorithm is at most $8C_h \operatorname{Diam}(S)^{6g-4}N^2$, where C_h is the constant depending on the metric h of S from [DST24, Thm 5.4] (Theorem I.32 in Chapter I) and N is the number of points of the output ε-net.*

The proof of this lemma mimics the proofs in [DST24]. The situation is quite different here, as the points are inserted incrementally and the flips are done at each insertion, whereas all points are known in advance in [DST24], which requires rewriting a complete proof.

Proof. We denote by $T_1 = DT(P_1), T_2, \dots, T_K$ the sequence of triangulations appearing during the ε-net algorithm. For $j \geqslant 1$, T_{j+1} is obtained from T_j either by flipping an edge, or by splitting a triangle into three from a new vertex. Every triangulation is geometric: flipping an edge maintains the property (Theorem I.31) and splitting a triangle clearly maintains it too.

We associate to any triangulation T of S a polyhedral surface Σ in $\mathbb{R}^3$ as in [DST24, Sec 2.3]. We identify the Euclidean plane with the plane $(z=1)$ in $\mathbb{R}^3$ and the Poincaré disk with the unit disk of this plane. Let $\mathcal{S}^2$ be the unit sphere of $\mathbb{R}^3$ and $s_0=(0,0,-1)$ be its South pole. The stereographic projection $\sigma : \mathcal{S}^2 \setminus \{s_0\} \to \mathbb{R}^2$ sends each point $p \neq s_0$ to the intersection of the plane $(z=1)$ and the line going through s_0 and p. For a triangulation T of S, the vertices of the polyhedral surface Σ are the inverse images of the vertices of $\widetilde{T}$ on $\mathcal{S}^2 \setminus \{s_0\}$. Its edges (resp. faces) are the segments (resp. triangles) of $\mathbb{R}^3$ whose vertices are the images of the vertices of the edges (resp. faces) of $\widetilde{T}$. Note that Σ is infinite since $\widetilde{T}$ is infinite. The surface Σ is convex if and only if T is a Delaunay triangulation of S [DST24, Sec 2.3].

For $j \geqslant 1$, let Σ_j be the polyhedral surface associated with T_j. Let us show that Σ_{j+1} *contains* Σ_j for each $j \geqslant 1$, meaning that Σ_{j+1} lies between Σ_j and $\mathcal{S}^2$. The case when T_{j+1} is obtained by flipping a

non-Delaunay edge e of T_j is studied in [DST24, Sec 2.3]: Σ_j is concave at each edge projected from a lift of e, and after the flip Σ_{j+1} is convex at all the new edges. Let us examine the case when T_{j+1} is obtained by splitting a triangle of T_j into three from a new vertex. In this case, T_j is a Delaunay triangulation, so Σ_j is convex and the vertices of Σ_{j+1} contains the vertices of Σ_j. The surface Σ_j is thus contained in Σ_{j+1} as well. As a result, every $\Sigma_{j'}$ with $j' > j$ contains Σ_j. If an edge is flipped at a step i of the algorithm, the corresponding line segment becomes interior to the polyhedral surface and all subsequent polyhedral surfaces, and it can never reappear.

We can now observe that no flip will ever create an edge longer than $8\operatorname{Diam}(S)$. Consider a fundamental domain $\Omega_{\widetilde{b}}$ consisting of one lift of each triangle of T_1 incident to $\widetilde{b}$. For all $\widetilde{x} \in \Omega_{\widetilde{b}}$, $d_{\mathbb{H}^2}(\widetilde{x}, \widetilde{b}) < 2\operatorname{Diam}(S)$. This is because $\widetilde{x}$ belongs to the circumdisk of the triangle $\widetilde{t} \in \widetilde{T_1}$ it lies in, and $\widetilde{b}$ lies on its boundary. That circumdisk must have a radius $r < \operatorname{Diam}(S)$, otherwise it would contain at least one lift of every point of S. In particular, it would contain a lift of a vertex of $\widetilde{t}$ in its interior, which is impossible since $\widetilde{t}$ is a Delaunay triangle. Let e be an edge created by a flip and let $\widetilde{v}$ be the lift of its midpoint v lying in $\Omega_{\widetilde{b}}$. The fundamental domain $\Omega_{\widetilde{b}}$ is strictly included in the disk of radius $4\operatorname{Diam}(S)$ and centered at $\widetilde{v}$: indeed, if $\widetilde{x} \in \Omega_{\widetilde{b}}$ then $d_{\mathbb{H}^2}(\widetilde{x}, \widetilde{v}) \leqslant d_{\mathbb{H}^2}(\widetilde{x}, \widetilde{b}) + d_{\mathbb{H}^2}(\widetilde{b}, \widetilde{v}) < 4\operatorname{Diam}(S)$. The proof of [DST24, Lem 4.2], replacing $2\Lambda(T)$ (where $\Lambda(T)$ is the length of the longest edge in T in the cited article) by $8\operatorname{Diam}(S)$ and Ω with $\Omega_{\widetilde{b}}$, shows that e cannot be longer than $8\operatorname{Diam}(S)$.

We now apply the proof of [DST24, Thm 5.4] with $8\operatorname{Diam}(S)$ instead of $2\Lambda(T)$ to prove that a Delaunay flip algorithm performed on a triangulation of S with n vertices flips at most $8C_h \operatorname{Diam}(S)^{6g-4}n^2$ edges, where C_h is the constant depending on the metric h of S in the cited paper. Since the edges that have been flipped cannot reappear, the number f_i of flipped edges at step i of the ε-net algorithm satisfies $f_i \leqslant 8C_h \operatorname{Diam}(S)^{6g-4}i^2 - \sum_{j=2}^{i-1} f_j$. It follows that

$$\sum_{j=2}^{i} f_j \leqslant 8C_h \operatorname{Diam}(S)^{6g-4}i^2$$

for all $i \in \{2, \dots, N\}$. □

III.4 Using the "combinatorial map" data structure

In preparation for our implementation, we adapt this algorithm to the "combinatorial map" data structure presented in Section I.4.2. Since we would like to use ε-nets as input for future approximation algorithms, which rely on graph algorithms, this data structure makes the implementation of such algorithms straightforward.

First, the "combinatorial map" data structure has to be enriched in order to be convenient and efficient for the ε-net algorithm. Indeed, this data structure only knows the lift of one triangle, stored as an anchor, composed of a dart of this triangle and the coordinates of the three vertices of this lift. However, our algorithm performs computations with lifts of triangles at each step, for instance to compute a lift of the circumcenter of a triangle and to locate it. In order to have constant time access to a lift of any triangle, an anchor is associated with *each* face of the triangulation.

Note that the anchors associated with adjacent triangles do not necessarily correspond to adjacent lifts in $\mathbb{H}^2$. This implies that, when working with a lift of a triangle, if an operation requires the third vertex of an adjacent lift, its coordinates must first be computed using the cross-ratio of their common edge.

III.4.1 Updating the data structure

In the ε-net algorithm, the data structure is modified by two operations: splitting a triangle into three to insert a new vertex, and flipping an edge. We discuss how this enriched "combinatorial map" data structure is maintained during these two operations.

When splitting a triangle t, the lift $\widetilde{t} = (\widetilde{v}_0, \widetilde{v}_1, \widetilde{v}_2)$ given by its anchor and the query point $\widetilde{q}$ to be inserted in $\widetilde{t}$ are known. The darts belonging to t are kept and three pairs of darts are created (Figure III.4). The pointers β_1 and β_2 of all these darts are set or updated to create the three new triangles in the combinatorial map. Three new anchors corresponding to the new triangles (v_0, v_1, q), (v_1, v_2, q) and (v_2, v_0, q) are created and associated with the darts belonging to their respective triangles. The cross-ratios of the three new edges are computed from the coordinates of $\widetilde{v}_0, \widetilde{v}_1, \widetilde{v}_2$ and $\widetilde{q}$. The cross-ratios of the edges (v_0, v_1), (v_1, v_2) and (v_2, v_0) must be updated. For the edge (v_0, v_1), for instance, its (non-updated) cross-ratio and the coordinates of $\widetilde{v}_0, \widetilde{v}_1$ and $\widetilde{q}$ are used to compute the coordinates of a lift of the third vertex of its neighboring incident triangle. The new value of the cross-ratio is then computed.

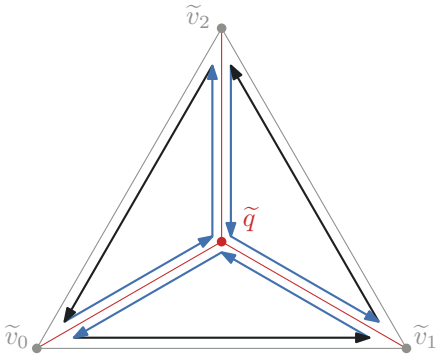

Figure III.4: Splitting a triangular face in three in the combinatorial map. New edges are in red. New darts are in blue.

Flipping an edge is performed similarly as in the original "combinatorial map" data structure when it comes to updating the combinatorial map and the cross-ratios. However, the anchors have to be updated in this augmented data structure. Let δ be a dart belonging to the edge being flipped and $\widetilde{v}_0, \widetilde{v}_1, \widetilde{v}_2$ be the coordinates of the points stored in the anchor associated with the face to which δ belongs. Without loss of generality, suppose that δ belongs to the edge (v_0, v_2) in the combinatorial map. Using the cross-ratio of this edge, we compute the coordinates of $\widetilde{u}$, the third vertex of the neighbor triangle lift that shares the edge $(\widetilde{v}_0, \widetilde{v}_2)$ (Figure III.5). The combinatorial map and the cross-ratios of the edges are updated as in the original "combinatorial map" data structure. Then, two new anchors corresponding to the lifts of the triangles obtained after the flip, (v_0, v_1, u) and (u, v_1, v_2) are associated with their respective triangles.

III.4.2 Point location

Since this data structure is not tied to a fundamental domain, the algorithm directly walks in the lifted triangulation to locate the circumcenter of a triangle. We use the same notations as in Section III.3.1: let t^ε be a triangle of circumradius greater than ε and c be its circumcenter. We call $\widetilde{t^\varepsilon}$ the lift of t^ε stored in its anchor and $\widetilde{c}$ the circumcenter of $\widetilde{t^\varepsilon}$, which is a lift of c. Let $\widetilde{x}$ be a vertex of $\widetilde{t^\varepsilon}$. The algorithm performs a straight walk (recall Chapter II) along the geodesic segment $\widetilde{x}\widetilde{c}$ in the lifted triangulation in $\mathbb{H}^2$. During

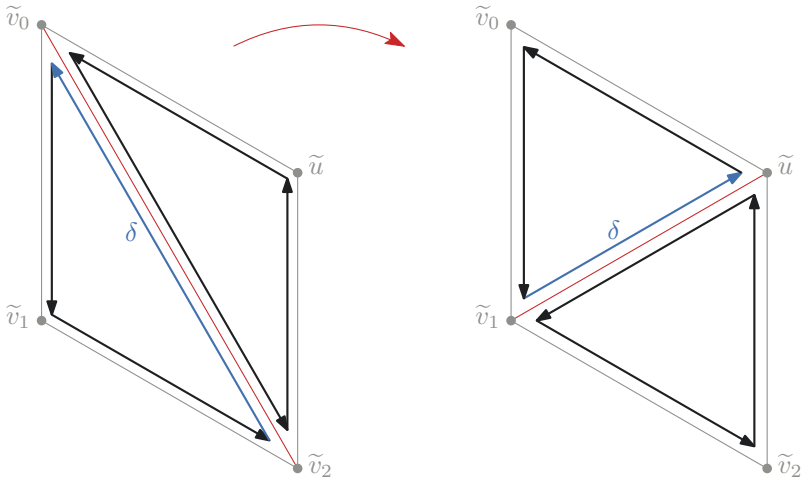

Figure III.5: Flipping an edge in the combinatorial map. No darts are added. Only adjacencies are modified.

the walk, the lift of each involved triangle is computed along the way using the coordinates of the vertices of its predecessor in the walk and the cross-ratio of their common edge, as mentioned in Section I.4.2.

III.4.3 Complexity analysis

As in Section III.3.3, we analyze the complexity of our ε-net algorithm in the real RAM model. However, since the data structure has changed, elementary operations are not the same as in the "fundamental domain" data structure. We consider the following operations as elementary operations in the real RAM model:

- Computing the radius or the center of the circumcircle of a triangle in $\mathbb{H}^2$;
- Deciding if a point lies on the right or the left side of an oriented geodesic segment in $\mathbb{H}^2$;
- Computing the anchor corresponding to a neighbor of a lifted triangle given by an anchor;
- Splitting a triangle in three (see Section III.4.1);
- Flipping an edge of a triangle (see Section III.4.1).

Theorem III.6. *Let S be a hyperbolic surface and $\varepsilon > 0$. The ε-net algorithm computes an ε-net in*

$$O\left(\frac{1}{\varepsilon^4}\right)$$

time with the augmented "combinatorial map" data structure.

Proof. We reuse part of the complexity analysis of Section III.3.3.

Finding a large triangle t^ε costs at most $2i + 4g - 6$ elementary operations at a step $i \geqslant 2$ of the algorithm because it requires to check each triangle at most once and a lift of each triangle is already stored in the data structure. Computing its circumcenter $\widetilde{c}$ is an elementary operation.

Recall from Section III.4.2 that $\widetilde{c}$ is located by performing a straight walk along the geodesic segment $\widetilde{x}\widetilde{c}$, were $\widetilde{x}$ is a vertex of the lift $\widetilde{t^\varepsilon}$ represented by the anchor of t^ε. By Theorem I.37, the geodesic segment

$\widetilde{x}\widetilde{c}$ can intersect each edge of the triangulation at most twice. Moreover, there are $3i + 6g - 9$ edges in the triangulation at step i (with a similar argument as for the number of triangles). The straight walk therefore visits at most $2(3i + 6g - 9)$ lifted triangles. Computing each visited lifted triangle costs one elementary operation. Moreover, at most three orientation tests are performed in each visited triangle to check whether it contains $\widetilde{c}$. Locating $\widetilde{c}$ in the triangulation thus costs at most $18i + 36g - 54$ elementary operations.

Updating the data structure when splitting the triangle containing c into three requires two elementary operations (splitting the triangle in three and adding c to the list of vertices).

Summing the above costs, step i (without the flips) requires at most $20i + 40g - 57$ elementary operations. We summing these costs from $i = 2$ to $i = N$ (the number of points in the output ε-net) and add the cost of the last step consisting of checking all the $2N + 4g - 6$ triangles when they all have circumradius at most ε. Excluding the flips, the algorithm requires at most $10N^2 + N(40g - 45) - 36g + 31$ elementary operations.

The flips are again counted globally for all steps in Lemma III.5. Since a flip costs one elementary operation, we obtain a total complexity for the ε-net algorithm of at most

$$(8C_h \operatorname{Diam}(S)^{6g-4} + 10)N^2 + N(40g - 45) - 36g + 31$$

elementary operations, where C_h is a constant depending on the metric h (see Lemma III.5) of S and $\operatorname{Diam}(S)$ is its diameter. The proof is concluded with Theorem III.1. □

III.5 Handling the thin part

We refer the reader to Section I.1.4 for a reminder on small curves. In particular, the notions of collar, ε-collar, injectivity radius and ε-thick/thin parts will be used here.

We have shown in Section III.2 that the number of points in an ε-net of a hyperbolic surface S depends on the systole σ of the surface. Moreover, it becomes arbitrarily large as σ decreases to zero. This is due to the width of a collar of a small curve, which grows when the curve becomes shorter. It is thus desirable to avoid adding points on these collars. In this section, we explain how to restrict the computation of an ε-net to the ε-thick part $S_\varepsilon^{\text{Thick}}$ of S, resulting in what we call a pseudo ε-net. The number of points in a pseudo ε-net then only depends on the genus of the surface and not on its systole. Since collars are isometric to cylinders, using a pseudo ε-net instead of an ε-net for approximation algorithms would not result in a significant loss of geometric information.

Definition III.7 (Pseudo ε-net)**.** Let S be a hyperbolic surface and $\varepsilon > 0$. A *pseudo ε-net* of S is a finite set of points $P \subset S_\varepsilon^{\text{Thick}}$ such that P is an ε-net of the ε-thick part $S_\varepsilon^{\text{Thick}}$.

III.5.1 Detection of collars

The following lemma defines a quantity l_ε that will help us detect ε-collars in the algorithm.

Lemma III.8. *Let T be the Delaunay triangulation of a set of points P on a hyperbolic surface S, $\varepsilon > 0$ and l_ε be the positive number such that $\cosh(\varepsilon) = \cosh^2(l_\varepsilon/2)$. If P is an ε-packing and if there exists a geodesic loop on S based at a point $p \in P$ that is shorter than l_ε, then T contains a loop edge based at p that is shorter than l_ε.*

Proof. Let γ be the shortest geodesic loop based at p. In $\mathbb{H}^2$, we consider a lift $\widetilde{\gamma}$ between two lifts $\widetilde{p}_1$ and $\widetilde{p}_2$ of p. We will show that the circle $\widetilde{C}$ of diameter the geodesic segment $\widetilde{p}_1\widetilde{p}_2$ is empty.

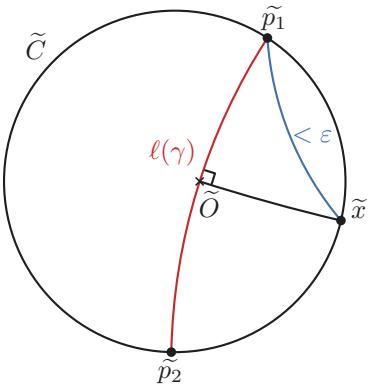

Figure III.6: Illustration of the proof of Lemma III.8.

The points in the disk bounded by $\widetilde{C}$ that are further away from $\widetilde{p}_1$ and $\widetilde{p}_2$ are those that lie both on the bisector of $\widetilde{p}_1$ and $\widetilde{p}_2$ and on $\widetilde{C}$. Let $\widetilde{x}$ be one of these points. The triangle formed by $\widetilde{x}$, $\widetilde{p}_1$ and the center $\widetilde{O}$ of $\widetilde{C}$ has a right angle at $\widetilde{O}$. By Hyperbolic Pythagoras' Theorem (Theorem I.7),

$$\cosh(d_{\mathbb{H}^2}(\widetilde{x},\widetilde{p}_1)) = \cosh^2\left(\frac{\ell(\gamma)}{2}\right) < \cosh^2\left(\frac{l_\varepsilon}{2}\right) = \cosh\varepsilon,$$

so $d_{\mathbb{H}^2}(\widetilde{x},\widetilde{p}_1) < \varepsilon$. Since γ is the shortest curve based at p and P is an ε-packing, $\widetilde{C}$ cannot contain another lift of p nor a lift of another vertex of the Delaunay triangulation. Therefore, a Voronoi edge passes through the midpoint of $\widetilde{p}_1$ and $\widetilde{p}_2$ in the dual Voronoi diagram of the lifted triangulation, so γ is an edge of T. □

Let us rewrite the equation $\cosh^2(l_\varepsilon/2) = \cosh\varepsilon$. Since $\cosh^2(l_\varepsilon/2) = \sinh^2(l_\varepsilon/2) + 1$ and $\cosh\varepsilon = 2\sinh^2(\varepsilon/2) + 1$, we obtain $\sinh^2(l_\varepsilon/2) = 2\sinh^2(\varepsilon/2)$, so $\sinh(l_\varepsilon/2) = \sqrt{2}\sinh(\varepsilon/2)$. In particular, this equation allows us to easily see that $l_\varepsilon > \varepsilon$. Note that we implicitly used the fact that ε and l_ε are positive. We will reuse this fact in the following computations without further mention.

The following lemma will serve as a security tool to ensure the ε-packing property during the algorithm.

Lemma III.9. *Let ρ be a small curve. If $\varepsilon \leqslant \ln\sqrt{2}$, then $\mathcal{C}(\rho,\varepsilon) \subset \mathcal{C}(\rho,l_\varepsilon) \subset \mathcal{C}(\rho)$ and the distance between the boundaries of $\mathcal{C}(\rho,l_\varepsilon)$ and $\mathcal{C}(\rho,\varepsilon)$ is greater than ε (Figure III.7).*

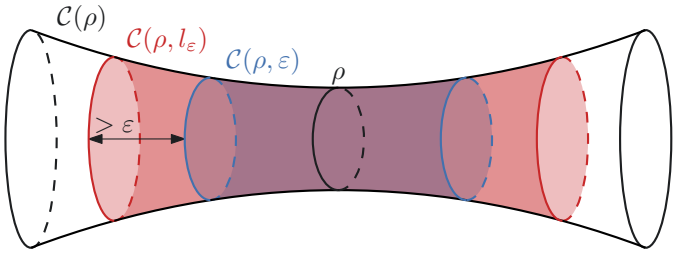

Figure III.7: Illustration of Lemma III.9.

Proof. The collar inclusion follows directly from Theorem I.24(2) combined with the observation that $l_\varepsilon/2 < \operatorname{arsinh} 1$ whenever $\varepsilon \leqslant \ln\sqrt{2}$, completes the argument.

Let d_1 be the distance between ρ and the boundary of $\mathcal{C}(\rho, l_\varepsilon)$, and d_2 be the same distance for $\mathcal{C}(\rho, \varepsilon)$. By Theorem I.24(3), we have $\sinh(l_\varepsilon/2) = \sinh(\ell(\rho)/2)\cosh(d_1)$ and $\sinh(\varepsilon/2) = \sinh(\ell(\rho)/2)\cosh(d_2)$. It follows that:

$$\begin{cases} d_1 = \operatorname{arcosh}\left(\dfrac{\sinh(l_\varepsilon/2)}{\sinh(\ell(\rho)/2)}\right) = \operatorname{arcosh}\left(\dfrac{\sqrt{2}\sinh(\varepsilon/2)}{\sinh(\ell(\rho)/2)}\right), \\ d_2 = \operatorname{arcosh}\left(\dfrac{\sinh(\varepsilon/2)}{\sinh(\ell(\rho)/2)}\right). \end{cases}$$

Since $\operatorname{arcosh} x = \ln\left(x + \sqrt{x^2-1}\right) = \ln x + \ln\left(1 + \sqrt{1 - 1/x^2}\right)$ for all $x > 1$, we obtain

$$\begin{cases} d_1 = \ln\sqrt{2} + \ln\left(\dfrac{\sinh(\varepsilon/2)}{\sinh(\ell(\rho)/2)}\right) + \ln\left(1 + \sqrt{1 - \dfrac{\sinh^2(\ell(\rho)/2)}{2\sinh^2(\varepsilon/2)}}\right), \\ d_2 = \ln\left(\dfrac{\sinh(\varepsilon/2)}{\sinh(\ell(\rho)/2)}\right) + \ln\left(1 + \sqrt{1 - \dfrac{\sinh^2(\ell(\rho)/2)}{\sinh^2(\varepsilon/2)}}\right). \end{cases}$$

Combining both equations yields

$$d_1 - d_2 = \ln\sqrt{2} + \ln\left(1 + \sqrt{1 - \frac{\sinh^2(\ell(\rho)/2)}{2\sinh^2(\varepsilon/2)}}\right) - \ln\left(1 + \sqrt{1 - \frac{\sinh^2(\ell(\rho)/2)}{\sinh^2(\varepsilon/2)}}\right)$$

Therefore, $d_1 - d_2 > \ln\sqrt{2} \geqslant \varepsilon$. □

III.5.2 Description of the algorithm

We describe the computation of a pseudo ε-net with the "fundamental domain" data structure presented in Section I.4.1.

Thanks to Lemmas III.8 and III.9, we have a tool to detect collars assuming that $\varepsilon \leqslant \ln\sqrt{2}$. To compute a pseudo ε-net, we modify the standard ε-net algorithm: the core idea is to halt the algorithm when a point is added on an l_ε-collar and then "remove" the corresponding ε-collar instead of inserting this point. We use a tag to handle what happens in the l_ε-collars: the triangles passing through an ε-collar are tagged to both indicate that the collar has already been processed and to prevent the algorithm from destroying these triangles.

The input is the same as the standard ε-net algorithm: it is a Delaunay triangulation of the single vertex b. See Section III.3.1 for more details on the input. The case where b is in a l_ε-collar requires a preprocessing step that is detailed in the remark after the algorithm.

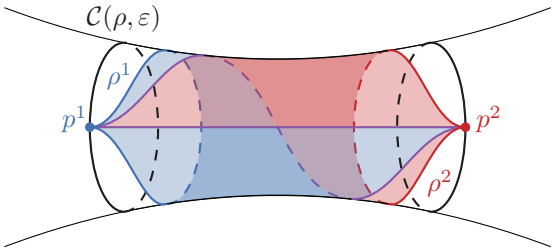

Figure III.8: Step 3 of the algorithm. The two colored triangles are the tagged triangles.

Algorithm III.10.

Input: *Delaunay triangulation of a single vertex b and $0 < \varepsilon \leqslant \ln \sqrt{2}$.*

Output: *Delaunay triangulation of a pseudo ε-net of S.*

1. *Follow the ε-net algorithm until a point p with an attached loop edge shorter than l_ε is found. This is verified at each flip in constant time. The flip algorithm continues even if such a loop edge is found. This loop edge is homotopic to a closed geodesic ρ and $p \in \mathcal{C}(\rho, l_\varepsilon)$.*

2. *If ρ is longer than ε, go back to Step 1. Note that this case could appear only if ρ has length between ε and l_ε. Else, compute the projection p^ρ of p on ρ and locate it in the triangulation.*

   (a) *If p^ρ lies in a tagged triangle, then $\mathcal{C}(\rho, \varepsilon)$ has already been processed[7]. Go back to Step 1.*

   (b) *Else, cancel the step where p was inserted. Insert consecutively two points p^1 and p^2 on each boundary component of $\mathcal{C}(\rho, \varepsilon)$. Go to Step 3.*

3. *After the insertion of p^1 and p^2, the Delaunay triangulation contains two loop edges ρ^1 and ρ^2, respectively based at p^1 and p^2, which are both homotopic to ρ (Figure III.8). Tag the two triangles in $\mathcal{C}(\rho, \varepsilon)$ incident to these loop edges. These triangles are then ignored by the algorithm when searching for a large triangle. Go back to Step 1.*

Remark. The initial point b is in a l_ε-collar if and only if one of the edges of the initial Delaunay triangulation is shorter than l_ε. Without loss of generality, we can suppose that, in this case, it lies on the corresponding small curve. Indeed, if the base point b of the Dirichlet domain $\mathcal{D}(\widetilde{b})$ is in the l_ε-collar of a small curve ρ, we can apply the algorithm proposed in [DKPT23] to obtain a new Dirichlet domain $\mathcal{D}(\widetilde{b'})$ with $b' \in \rho$. To deal with the case where b lies on a closed geodesic shorter than ε, we first insert the points b^1 and b^2 corresponding to the ε-collar of ρ such that the geodesic line b^1b^2 is orthogonal to ρ at b. Then, we remove b from the triangulation while maintaining the Delaunay property. The loop edges based at b^1 and b^2 are already in the triangulation. It suffices to remove the edges incident to b and add the two edges joining b^1 and b^2 (like the purple ones in Figure III.8). Tag the two corresponding triangles as in Step 3.

We need to verify that this algorithm terminates, outputs a pseudo ε-net of S and that all the required computations can be performed in our model. Computations will be handled in the Poincaré disk $\mathbb{H}^2$, in which a lift of a collar $\mathcal{C}(\rho)$ is a region $\{\widetilde{x} \in \mathbb{H}^2 : d_{\mathbb{H}^2}(\widetilde{x}, \widetilde{\rho}) \leqslant w(\rho)\}$, where $\widetilde{\rho}$ is a lift of ρ. Since ρ is a closed geodesic curve on S, $\widetilde{\rho}$ is a geodesic segment of $\mathbb{H}^2$. The two boundary components of the collar lift to two (non-geodesic) circular arcs that are equidistant to $\widetilde{\rho}$. Their supporting circles both intersects the supporting geodesic line of $\widetilde{\rho}$ on the boundary of the Poincaré disk. A lift of $\mathcal{C}(\rho)$ is therefore a piece of what Thurston called a *banana* for obvious visual reasons. The same applies to lifts of ε-collars.

III.5.3 Correctness of the algorithm

At each step, the set of vertices of the Delaunay triangulation is an ε-packing of $S_\varepsilon^{\mathrm{Thick}}$. Similarly to the standard algorithm, when a circumcenter of a large triangle is inserted on $S_\varepsilon^{\mathrm{Thick}}$, it is ε far from all the vertices of the current Delaunay triangulation. Additionally, when a couple $\{p^1, p^2\}$ is inserted, both points are at distance at least ε from the other vertices of the triangulation by Lemma III.9. Note that it is possible

[7] Of course, we could design an optimized method to verify whether a collar has already been processed. Since this method only adds a point location step, we accept this complexity for simplicity.

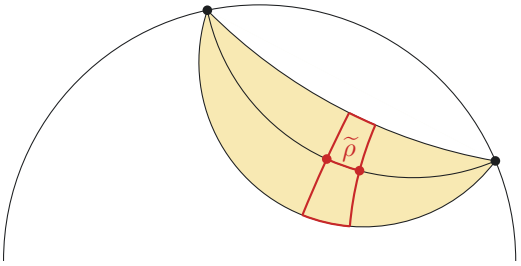

Figure III.9: A banana (yellow) and a lift of a collar of a small curve ρ.

that a couple $\{p^1, p^2\}$ is separated by a distance smaller than ε on S if the corresponding small curve has length close to ε. However, on $S_\varepsilon^{\text{Thick}}$, their distance is at least ε because a path joining them would first have to go out of the corresponding l_ε-collar, whose boundary is at distance greater than ε from either points by Lemma III.9. Moreover, in this case, Lemma III.8 still applies because every vertex that is not p^1 or p^2 is at distance ε from the other vertices.

The following lemma shows that each large triangle that is not tagged will be processed. Together with the same arguments as for the standard ε-net algorithm, this proves that the procedure terminates and that the output is a pseudo ε-net of S.

Lemma III.11. *Let $\varepsilon \leqslant \ln\sqrt{2}$ and S be a hyperbolic surface. Let T be a Delaunay triangulation of S obtained at any step of the pseudo ε-net algorithm. Then the circumcenter of any large triangle that is not tagged is either in $S_\varepsilon^{\text{Thick}}$ or in an ε-collar that has not been handled by the algorithm yet.*

Proof. Let C be the circumcircle of a large triangle t of T that is not tagged. Suppose that its center is in an ε-collar that has already been handled. We will show that this situation leads to a contradiction.

The circle C intersects the boundary of this ε-collar. Let p be the vertex with an attached loop edge of length ε on this boundary component. Consider a lift $\widetilde{C}$ of C. Since it is an empty circle, it passes between the two lifts $\widetilde{p}_1$ and $\widetilde{p}_2$ of p such that the geodesic segment $\widetilde{p}_1\widetilde{p}_2$ is a lift of the loop edge.

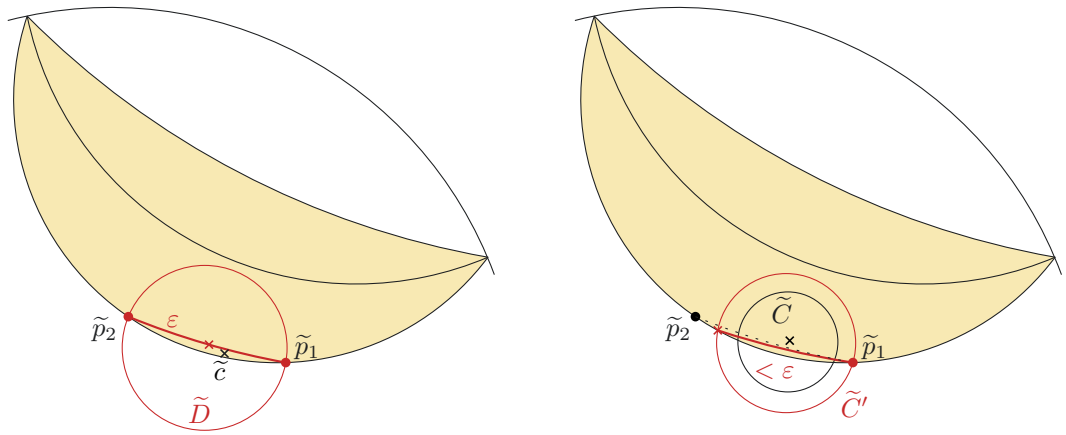

Figure III.10: The two cases of the proof of Lemma III.11. Left: $\widetilde{C}$ is not drawn, but its center is represented as the black cross.

If the center $\widetilde{c}$ of $\widetilde{C}$ lies between the lift of the boundary component of p and the geodesic segment

$\widetilde{p}_1\widetilde{p}_2$, then it lies in the disk $\widetilde{D}$ of diameter $\widetilde{p}_1\widetilde{p}_2$ (Figure III.10, left). The diameter of $\widetilde{D}$ is ε, and since $\widetilde{C}$ has radius larger than ε, it must contain $\widetilde{p}_1$ and $\widetilde{p}_2$ in its interior. This contradicts the fact that C is empty.

Else, let $\widetilde{C}'$ be the circle obtained by increasing the radius of $\widetilde{C}$ until it touches $\widetilde{p}_1$ or $\widetilde{p}_2$ (Figure III.10, right). Up to renumbering, suppose that it is $\widetilde{p}_1$. The intersection between $\widetilde{C}'$ and the lift of the boundary component of the ε-collar yields a chord shorter than ε. The triangle t has at least one vertex that is not p. This vertex has a lift on $\widetilde{C}$ that is in the smaller piece of the interior of $\widetilde{C}'$ delimited by the chord. It is thus at distance less than ε from $\widetilde{p}_1$, which contradicts the fact that the vertices of T form an ε-packing of $S_\varepsilon^{\text{Thick}}$. □

III.5.4 Computation details & complexity analysis

Lemma III.12. *Let T be the Delaunay triangulation of a set of points P on a hyperbolic surface S and $\varepsilon > 0$. If T contains a loop edge shorter than l_ε based at a vertex p, then the lift of an ε-collar containing $\widetilde{p}_o$ can be computed in constant time.*

Proof. All the notations of this Lemma are illustrated in Figure III.11.

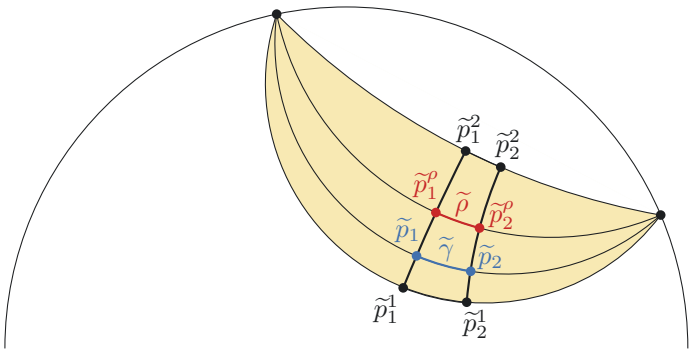

Figure III.11: Illustration of the notations of the proof of Lemma III.12.

Let γ be the loop edge around p, $\widetilde{\gamma}$ be its lift with vertex $\widetilde{p}_1 = \widetilde{p}_o$ and $\widetilde{p}_2$ be the second vertex of $\widetilde{\gamma}$. The translation τ that sends $\widetilde{p}_1$ to $\widetilde{p}_2$ is stored in the data structure. Its fixed points can then be computed in constant time as the two solutions of the quadratic equation $\tau(z) = z, z \in \mathbb{C}$. From this, its axis is computed as the geodesic line whose points at infinity are the fixed points of τ (recall Section I.1.1). The length of the associated curve ρ on S is computed as well: $\ell(\rho) = \ell(\tau) = 2\operatorname{arcosh}(|\operatorname{Tr}(\tau)|/2)$ (recall Section I.1.4).

The two geodesic lines in the Poincaré disk $\mathbb{H}^2$ that are orthogonal to ρ and go through $\widetilde{p}_1$ and $\widetilde{p}_2$ are computed. On those lines, we want to compute the points $\widetilde{p}_1^1, \widetilde{p}_1^2, \widetilde{p}_2^1$ and $\widetilde{p}_2^2$ that are on the boundary of the banana corresponding to $\mathcal{C}(\rho, \varepsilon)$. By Theorem I.24(3), the distances from these points and the points $\widetilde{p}_1^\rho, \widetilde{p}_2^\rho$ that lie on $\widetilde{\rho}$ and on the two orthogonal lines is $\operatorname{arcosh}(\sinh(\varepsilon/2)/\sinh(\ell(\rho)/2))$.

All the computations are performed in $\mathbb{H}^2$ in constant time in the real RAM model. □

We keep the notations from the proof of the previous lemma and show the following:

Lemma III.13. *Let i be the number of vertices in the Delaunay triangulation at step 2 of the algorithm. If $\varepsilon < 2\operatorname{arsinh} 1$, the points p^1 and p^2 that lift to $\widetilde{p}_1^1$ and $\widetilde{p}_1^2$ (respectively) can be located in $O(i)$ time in the Delaunay triangulation.*

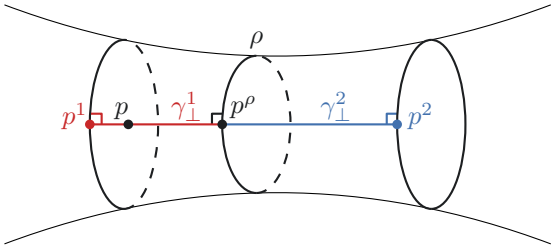

Figure III.12: Illustration of the notations of the proof of Lemma III.13.

Proof. Let p^1 and p^2 be the points of S on which $\widetilde{p}_1^1$ and $\widetilde{p}_1^2$ respectively project. Let p^ρ be the projection of p on ρ, which projects on p_1^ρ. Like for the complexity of the location of a circumcenter (recall the proof of Theorem III.4), we want prove that the orthogonal geodesic segment $\widetilde{\gamma}_\perp^1$ from $\widetilde{p}_1^1$ to $\widetilde{p}_1^\rho$ projects on a shortest path $\gamma_\perp^1$ on S (Figure III.12).

By Theorem I.24, and since $\varepsilon < 2\operatorname{arsinh} 1$, we know that the ε-collars are embedded on S. Then, $\gamma_\perp^1$ is a shortest path inside the collar and no shortest path goes outside, else it would have to pass through the other end of the collar and therefore be longer than $\gamma_\perp^1$. The same holds for the symmetric path $\gamma_\perp^2$. Then, $\widetilde{p}_1^1$ and $\widetilde{p}_1^2$ can be located in the tiling $\{\gamma\widetilde{D}_o\}_{\gamma\in\Gamma}$ from $\widetilde{p}_1$ in $O(g^2)$ time by walking along the geodesic segments $\gamma_\perp^1$ and $\gamma_\perp^2$. Note that the copy in which $\widetilde{p}_1$ lies is already known since it is the lift of the circumcenter p that has been inserted in the triangulation. Their lifts in $\widetilde{D}_o$ are then computed in constant time. Finally, the triangles in which they lie are found in $O(i)$ time (with the same arguments as in the proof of Theorem III.4). □

Theorem III.14. *Let S be a hyperbolic surface given by a Delaunay triangulation on a single vertex (or, equivalently, by a Dirichlet domain) and let $\varepsilon \leqslant \ln\sqrt{2}$ be a positive number. A pseudo ε-net of S can be computed in $O(1/\varepsilon^4)$ time, where the hidden constant depends on S.*

Proof. We keep the same notations as in the previous proofs.

We first prove an upper bound on the number of inserted points. If the algorithm terminates without reaching step two, S is ε-thick by Lemma III.8. The total number of inserted points is then smaller than $16(g-1)/\varepsilon^2$ by Theorem III.1. Otherwise, the situation is similar except that there are no vertices in the ε-collars. For each ε-collar, a point have been inserted and then removed. Two other points have instead been inserted on the two boundary components. Since there are at most $3g-3$ ε-collars, the number of inserted points is then upper bounded by $16(g-1)/\varepsilon^2 + 3g - 3$.

Lemmas III.12 and III.13 imply that inserting a couple $\{p^1, p^2\}$ for a given ε-collar is as costly as inserting a point like in the ε-net algorithm (only the constant changes). Similarly, locating p^ρ at step 2 of the algorithm only increases the constant. We can repeat similar computations as in the proof of Theorem III.4 to obtain a total complexity of the algorithm of $O(1/\varepsilon^4)$ time. □

Remark. The algorithm can be used with $\varepsilon = 2\operatorname{arsinh} 1$ to compute a thick-thin decomposition of the surface. The proof of Theorem III.14 applies to this situation but the result might not be an ε-packing around the boundaries of the ε-collars because Lemma III.9 does not apply in this case.

III.5.5 Lower bound on the systole

We end this section with a result similar to Lemma III.8 that will be useful in Chapter IV to provide a lower bound on the systole of a hyperbolic surface.

Let $0 < \varepsilon < \ln \sqrt{2}$ and P be an ε-net of a hyperbolic surface S. If the systole σ of S is shorter than l_ε, then P has a point on $\mathcal{C}(\sigma, l_\varepsilon)$. By Lemma III.8, there is a loop edge in the Delaunay triangulation of P. By contraposition, if there is no loop edge in the Delaunay triangulation of an ε-net, then $\sigma > l_\varepsilon$.

We thus have a tool to find a lower bound on the systole of S, but it requires that $\varepsilon \leqslant \ln \sqrt{2} \approx 0.35$ (and $l_{\sqrt{2}} \approx 0.48$). The following lemma provides a similar tool but it works for any $\varepsilon \leqslant \operatorname{arsinh} 1 \approx 0.88$.

Lemma III.15. *Let P be an ε-net of a hyperbolic surface S with $\varepsilon < \operatorname{arsinh} 1$. If $\sigma \leqslant \varepsilon$, then there is a loop edge in the Delaunay triangulation of P.*

Proof. Let σ be a systole of S. There is a point $p \in P$ such that $d_S(p, \sigma) \leqslant \varepsilon$ because P is an ε-covering. Since $\varepsilon < \operatorname{arsinh} 1$, we have $\sinh(\varepsilon)\sinh(\varepsilon/2) < 1$, so $\varepsilon < \operatorname{arsinh}(1/\sinh(\varepsilon/2)) \leqslant \operatorname{arsinh}(1/\sinh(\sigma/2))$. Therefore, $p \in \mathcal{C}(\sigma)$. Let γ_p be the shortest geodesic loop based at p and $\ell(\gamma_p)$ be its length. By Theorems I.23 and I.24(3) and since $\sigma \leqslant \varepsilon$, we have $\sinh(\ell(\gamma_p)/2) = \sinh(\sigma/2)\cosh(d_S(p,\sigma)) \leqslant \sinh(\varepsilon/2)\cosh(\varepsilon)$. We obtain

$$\ell(\gamma_p) \leqslant 2 \operatorname{arsinh}\left(\sinh\frac{\varepsilon}{2}\cosh\varepsilon\right). \tag{III.2}$$

Consider a lift $\widetilde{\gamma}_p$ of γ_p in $\mathbb{H}^2$. Its endpoints $\widetilde{p}$ and $\widetilde{p}'$ are lifts of p. Let D be the hyperbolic disk of diameter $\widetilde{\gamma}_p$. We will show that any point $\widetilde{x} \in D$ is at distance less that ε from $\widetilde{p}$ or $\widetilde{p}'$. This is equivalent to showing that $\min(d_{\mathbb{H}^2}(\widetilde{x}, \widetilde{p}), d_{\mathbb{H}^2}(\widetilde{x}, \widetilde{p}')) < \varepsilon$. This will conclude the proof because, since P is an ε-packing, it implies that no lift of any point of P lies in D; then there is a loop edge at p.

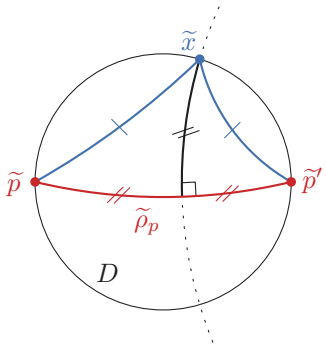

Figure III.13: Case where $\widetilde{x}$ yields the maximum of $\min(d_{\mathbb{H}^2}(\widetilde{x}, \widetilde{p}), d_{\mathbb{H}^2}(\widetilde{x}, \widetilde{p}'))$ over D.

Let us show that $\min(d_{\mathbb{H}^2}(\widetilde{x}, \widetilde{p}), d_{\mathbb{H}^2}(\widetilde{x}, \widetilde{p}')) < \varepsilon$ for all $\widetilde{x} \in D$. The maximum of this minimum is attained when $\widetilde{x}$ lies on the boundary of D and $d_{\mathbb{H}^2}(\widetilde{x}, \widetilde{p}) = d_{\mathbb{H}^2}(\widetilde{x}, \widetilde{p}')$. In this case, $\widetilde{x}$ lies on the perpendicular bisector of $\widetilde{p}$ and $\widetilde{p}'$, which intersects $\widetilde{\gamma}_p$ at a right angle in its middle (Figure III.13). By Hyperbolic Pythagoras' Theorem (Theorem I.7), we have $\cosh(d_{\mathbb{H}^2}(\widetilde{x}, \widetilde{p})) = \cosh^2(\ell(\gamma_p)/2)$. By Equation III.2 and

since $\cosh(\operatorname{arsinh} x) = \sqrt{x^2+1}$ for all $x \in \mathbb{R}$, we obtain

$$\begin{aligned} d_{\mathbb{H}^2}(\widetilde{x}, \widetilde{p}) &= \operatorname{arcosh}\left(\cosh^2\left(\frac{\ell(\gamma_p)}{2}\right)\right) \\ &\leqslant \operatorname{arcosh}\left(\cosh^2\left(\operatorname{arsinh}\left(\sinh\frac{\varepsilon}{2}\cosh\varepsilon\right)\right)\right) \\ &\leqslant \operatorname{arcosh}\left(\sinh^2\left(\frac{\varepsilon}{2}\right)\cosh^2(\varepsilon)+1\right). \end{aligned}$$

By classic identities,

$$\begin{aligned} \operatorname{arcosh}\left(\sinh^2\left(\frac{\varepsilon}{2}\right)\cosh^2(\varepsilon)+1\right) < \varepsilon &\Leftrightarrow \sinh^2\left(\frac{\varepsilon}{2}\right)\cosh^2(\varepsilon)+1 \leqslant \cosh\varepsilon \\ &\Leftrightarrow \cosh^3(\varepsilon) - \cosh^2(\varepsilon) - 2\cosh(\varepsilon) + 2 < 0. \end{aligned}$$

The left-hand side of the inequality is a polynomial in $\cosh\varepsilon$. Its roots are 1, $\sqrt{2}$ and $-\sqrt{2}$. It is negative between 1 and $\sqrt{2}$, that is, for $\varepsilon \in \left(0, \operatorname{arcosh}\sqrt{2}\right)$. Since, $\operatorname{arcosh}\sqrt{2} = \operatorname{arsinh} 1$, this concludes the proof. □

Chapter IV

Implementation

In this chapter, we discuss the implementation of the ε-net algorithm discussed in Section III.4. The pseudo ε-net algorithm detailed in Section III.5 is not implemented.

We begin by presenting the CGAL components used for our implementation (Section IV.1). We continue with discussions on the generation of inputs for the ε-net algorithm (Section IV.2) and on the interrelated choice of template parameters used in practice (Section IV.3). Next, we detail the actual implementation of the ε-net algorithm (Section IV.4), with an emphasis on the choices made to address the encountered technical constraints. We then explain the methods used to visualize the output of the algorithm (Section IV.5). Finally, we conclude with the experiments used to help us design the implementation and to analyze the behavior of the algorithm in practice (Section IV.6).

The source code of our implementation is available on GitHub[8]. It is intended for integration into the CGAL `Triangulation_on_hyperbolic_surface_2` package. After integration, our GitHub repository will be deprecated and our code will be publicly available through the main CGAL repository at `https://github.com/CGAL/cgal`.

IV.1 Design

We implemented the ε-net algorithm with the augmented "combinatorial map" data structure presented in Section III.4. Since we would like to use ε-nets as input for future approximation algorithms that rely on graph algorithms, this data structure makes the implementation of such algorithms straightforward.

Recall that our data structure is a combinatorial map with geometric information on edges and faces. We summarize this data structure here, but we refer the reader to Sections I.4.2 and III.4 for more details. A combinatorial map is made of darts, which can be seen as oriented edges. It also stores pointers between darts to describe adjacency relations. Since a combinatorial map is a purely topological data structure, it has been augmented with information to represent the geometry of the surface: a cross-ratio for each edge and an anchor, representing the lift of a triangle in $\mathbb{H}^2$. This additional information yields a data structure that describes geometric triangulations of hyperbolic surfaces [DDKT22]. This is what we call the "combinatorial map" data structure (recall Section I.4.2). In the data structure for our ε-net algorithm, we additionally associate an anchor with each face of the triangulation in order to have constant-time

[8] `https://github.com/camille-lanuel/cgal/tree/Triangulation_on_hyperbolic_surface_2-Delaunay_triangulation_class-camille-lanuel/Triangulation_on_hyperbolic_surface_2`

access to a lift of any triangle (recall Section III.4).

Our data structure is implemented in the `Delaunay_triangulation_on_hyperbolic_surface_2` class, which inherits from the CGAL `Triangulation_on_hyperbolic_surface_2` class. The suffix "`_2`" indicates dimension two. For readability, we abbreviate these classes as `THS` and `DTHS` respectively.

We refer the reader to I.5 for a reminder on how CGAL works.

IV.1.1 CGAL combinatorial maps

The CGAL `Combinatorial_map` class provides all the functionalities to construct and manipulate combinatorial maps. The class takes an integer (`unsigned int`) as a template parameter to specify the dimension of the combinatorial map. In an instance of `Combinatorial_map`, *cells* such as vertices (0-cells), edges (1-cells) or faces (2-cells) are implicitly represented as sets of darts.

Moreover, information can be associated with cells via *attributes*. An i-attribute stores information for an i-cell. This association is injective: two distinct i-cells are associated with two distinct i-attribute objects and an i-cell may have no associated i-attribute. If an i-attribute a is associated with a given i-cell c, then all the darts of c are associated with a. Attributes are specified as a template parameter when declaring an instance of `Combinatorial_map`.

Since it is often necessary to know whether a dart has been processed during the execution of an algorithm, the class provides Boolean marks for this purpose. This allows the user to retrieve the desired information in constant time. We use this feature regularly in our code.

IV.1.2 CGAL triangulations on hyperbolic surfaces

The "combinatorial map" data structure described in [DDKT22] and in Section I.4.2 has been implemented in the CGAL package `THS` [DDPT25], which will be included in the next CGAL release. This data structure is described in the class of the same name, which is the main class of the package.

The `THS` class takes two template parameters. The first is a traits class that provides complex numbers and points. More details on this traits class will be given in Section IV.3. The second template parameter, `Attributes`, corresponds to the information associated with the cells of the combinatorial map. It defaults to complex numbers associated with edges, but our `DTHS` class uses its own `Attributes`. We added this template parameter to the class in order to be able to use other `Attributes` in our `DTHS` class.

An instance of `THS` is usually constructed from an input fundamental domain. Details on the generation of inputs will be given in Section IV.2. The constructor method sets the three attributes of the instance:

- `combinatorial_map_` (of type `Combinatorial_map<2, Attributes>`);
- `anchor_` (of type `Anchor`), a class-provided object made of a dart and three vertices;
- `has_anchor_` (of type `bool`), a Boolean indicating whether the triangulation has an anchor.

The class provides the option to not set an anchor, which is useful for our `DTHS` class because it has anchors associated with all faces, so this attribute does not need to be maintained.

IV.1.3 Overview of the `DTHS` class

While the `THS` class implements a flip algorithm, it does not aim to provide full support for Delaunay triangulations on hyperbolic surfaces. The `Delaunay_triangulation_on_hyperbolic_surface_2`

class fulfills this role. This class implements the data structure of the ε-net algorithm, as well as Delaunay triangulation functionalities and utility classes. The main functionalities implemented in the DTHS class are:

- locating a point in the Delaunay triangulation,
- inserting a point in the Delaunay triangulation,
- the ε-net algorithm.

Their implementation is detailed in Section IV.4.

The DTHS class is templated by a traits class, which is its only template parameter. It inherits from the THS class with `Delaunay_triangulation_attributes<Traits>` as the `Attributes` template parameter. The `Delaunay_triangulation_attributes` type defines the information associated with cells of the combinatorial map as a `Complex` (a complex number that represents a cross-ratio) for edges and an `Anchor` (as defined above) for faces. The DTHS class is declared as in Listing IV.1.

Snippet IV.1: Declaration of the `Delaunay_triangulation_on_hyperbolic_surface_2` class

```
template<class Traits>
class Delaunay_triangulation_on_hyperbolic_surface_2:
public Triangulation_on_hyperbolic_surface_2<Traits,
    Delaunay_triangulation_attributes<Traits>>
{
// class code
}
```

The construction of an instance of DTHS typically takes a fundamental domain as input, given as an instance of `Hyperbolic_fundamental_domain_2<Traits>`. The constructor method first calls the analogous constructor method of the THS class, which naively triangulates the domain and sets the `combinatorial_map_` attribute, a cross-ratio for each edge and the `anchor_` attribute. The triangulation is then made Delaunay using the flip algorithm implemented in the THS class. An anchor is set for each face using a breadth-first search algorithm to gradually compute an anchor for each face of the triangulation. Finally, the `anchor_` attribute is deleted and `has_anchor_` is set to `false`.

IV.2 Input surfaces

IV.2.1 Generation

Recall from Section III.3.1 that the input for the ε-net algorithm is a Delaunay triangulation of a hyperbolic surface with a single vertex. As we saw in Section IV.1.3, an instance of DTHS can be constructed with an input fundamental domain. A valid input for the ε-net algorithm can therefore be obtained by constructing an instance of DTHS from a fundamental domain whose vertices in $\mathbb{H}^2$ project to the same point on the surface. Such a fundamental domain is generated via the `Hyperbolic_fundamental_domain_factory_2` class from the `Triangulation_on_hyperbolic_surface_2` package [DDPT25]. Before giving details about how the factory class works, we discuss arithmetic issues.

Our algorithm relies on computations of cross-ratios and points in $\mathbb{H}^2$. Standard floating-point number types like `float` or `double` are unsuitable for these computation, as this is notoriously unstable. Iordanov

and Teillaud's work [IT17, IT19] showed that Delaunay triangulations can be computed on the Bolza surface, represented by a fundamental polygon with very specific algebraic numbers. Even when computing the Delaunay triangulation of points with rational coordinates, the fundamental group of the Bolza surface leads to algebraic numbers. In practice, algebraic numbers are handled with the CORE library [dtb] but this approach was shown to fail for generalized Bolza surfaces, even for genus 3 [EITV22]. Therefore, avoiding the use of algebraic numbers wherever possible is a key constraint.

We now detail how the factory class generates fundamental domains. To be more precise, it generates random fundamental octagons for hyperbolic surfaces of genus 2. Any genus 2 surface has a fundamental domain that is a symmetric octagon centered at the origin [ADBC+05]. The generation of such an octagon works as follows: three points are first chosen at random in the upper half of the Poincaré disk. A fourth point is then computed so that the octagon formed by these four points and the four symmetric points with respect to the origin is a fundamental domain. The eight vertices of the domain project to the same point on the surface, as explained on page I.1.3.

It has been shown that any surface of genus 2 can be approximated by a surface described by a fundamental octagon whose vertices have rational coordinates [DDKT22]. Restricting to such surfaces does therefore not induce any significant loss of generality. For higher genus surfaces, a construction of fundamental polygons is described in [ZVC80, Sec 6.11], but unfortunately it is not clearly leading to tractable numbers. Vincent Despré and Marc Pouget have managed to generate a few fundamental polygons with rational coordinates for hyperbolic surfaces of genus higher than two, but not enough to run experiments on a significant number of such surfaces.

For completeness, we mention that there are other ways of constructing random surfaces in theory, by randomly gluing hyperbolic ideal triangles together [BM04, Pet17]. However, these methods do not yield a practical construction. Additionally, they lead to surfaces with huge genus (typically, several thousands), which cannot be manipulated in practice.

IV.2.2 Distribution

We can generate hyperbolic surfaces of genus 2 by randomly choosing three complex numbers in the Poincaré disk, but are they well distributed? The situation was nicely described by Mirzakhani [Mir13] (though there are more recent results [Mon22]). We just summarize it here. The moduli space $\mathcal{M}_g$ is the set of all hyperbolic surfaces of genus g. It can be equipped with two natural metrics: the Teichmüller distance and the Weil-Petersson one. Roughly speaking, they measure the deformation between two hyperbolic metrics: the first one considers the supremum and the second the average. Ideally, we would like to obtain a uniform sampling of $\mathcal{M}_g$ for either of the two metrics. Unfortunately, the current mathematical literature does not answer this question. To this day, we can only recognize that the fundamental domain factory described above does not lead to a uniform distribution on $\mathcal{M}_g$.

Moreover, we can expect generated surfaces to have a long systole. Mirzakhani showed that small systoles are rare in $\mathcal{M}_g$: the probability to obtain a surface with a systole smaller than some $\sigma_0 > 0$ using a uniform distribution for the Weil-Petersson distance is of the order of σ_0^2. This means that the typical case is a surface with a reasonably long systole. For completeness, we experiment with a specifically designed surface with a small systole.

IV.3 Choice of template parameters

We refer the reader to Section I.5.1 for a reminder on generic programming in CGAL.

The `Hyperbolic_surface_traits_2` class is passed as the `Traits` template parameter to both the `THS` and `DTHS` classes. It is the only model implementing the `HyperbolicSurfaceTraits_2` concept required by these two classes. This traits class is itself templated on one of the two existing Delaunay triangulation traits classes. The first one, `Hyperbolic_Delaunay_triangulation_traits_2`, guarantees exact constructions of Delaunay triangulations and Voronoi diagrams for input points with algebraic coordinates. However, we do not use it in practice because it leads to unreasonably long computation time. The second one, `Hyperbolic_Delaunay_triangulation_CK_traits_2`, guarantees exact constructions for input points with rational coordinates. It is faster but requires that all the input points have rational coordinates. For performance reasons, we want to use this traits class. However, this constrains us to adapt the ε-net algorithm in order to have input points with rational coordinates: we have to round the algebraic coordinates of each to-be-inserted circumcenter to rational coordinates. This rounding process is explained in Section IV.4.2.

As mentioned in Section IV.2, we want to avoid the use of algebraic numbers and of the CORE library [dtb]. Since working with surfaces described by fundamental domains whose vertices have rational coordinates does not induce any significant loss of generality, we choose to work with rational numbers. CGAL provides several rational number types based on the CORE, GMP or LEDA libraries. Additionally, the CGAL number type `Exact_rational` is a wrapper for either of these number types, depending on which library is configured by the user.

IV.4 Implementation of the ε-net algorithm

This section provides details about the `epsilon_net` method of the `DTHS` class, which implements the ε-net algorithm described in Section III.4. Note that the part of the algorithm that handles the thin part (Section III.5) is not implemented.

In the remainder of this section, we suppose that we are working on the Delaunay triangulation of a hyperbolic surface S, encapsulated in an instance of `DTHS`, and that we are given a parameter $\varepsilon > 0$. As in previous chapters, we use $\widetilde{\cdot}$ for objects in $\mathbb{H}^2$, while objects on S are noted without.

IV.4.1 Overview

We first summarize the implementation of the `epsilon_net` method. The parameter $\varepsilon > 0$ is given to the `epsilon_net` method as a `double`, the C++ number type representing double-precision floating-point numbers. A precondition for the method is that the initial triangulation must already be an ε-packing. It returns a Boolean value indicating whether the set of vertices is guaranteed to be an ε-net.

We recall that a triangle is *large* if its circumradius is greater than ε.

1. Initialize a list $\mathcal{L}$, representing triangles to be processed, with every triangle of the initial Delaunay triangulation (Section IV.4.7).

2. If $\mathcal{L}$ is not empty, remove the first triangle t of $\mathcal{L}$. If t is large (Section IV.4.2), go to 3; else, go back to 2. If $\mathcal{L}$ is empty, go to 10.

3. Approximate the circumcenter $\widetilde{c}_t$ of its lift represented by its anchor to a point $\widetilde{c}$ with rational coordinates (Section IV.4.2).
4. Locate $\widetilde{c}$ in the lifted triangulation (Section IV.4.3).
5. Insert c in the triangulation by splitting the triangle it lies in (Section IV.4.4).
6. Add each new triangle to $\mathcal{L}$ (Section IV.4.7) and each flippable edge of each new triangle to a list of flippable edges (Section IV.4.5).
7. Restore the Delaunay property using the list of flippable edges (Section IV.4.5).
8. Add the triangles modified by flips to $\mathcal{L}$ (Section IV.4.7).
9. Go back to 2.
10. Return a Boolean that is set to true if the set of vertices is guaranteed to be an ε-net (Section IV.4.6).

Preliminary remark As discussed in Section III.4, in the data structure, the anchors associated with adjacent triangles on the surface do not necessarily represent adjacent lifts in $\mathbb{H}^2$. Many methods require knowledge of the four vertices of a quadrilateral formed by two adjacent lifts of adjacent triangles. In practice, we access the lift represented by the anchor associated with one of these triangles and compute the fourth vertex of the quadrilateral. This is done using the cross-ratio of the diagonal edge and the three vertex values stored in the anchor. It is unnecessary to explicitly check whether the two lifts are adjacent in $\mathbb{H}^2$, because the computation needed to check adjacency is equivalent to directly computing the fourth vertex of the quadrilateral.

IV.4.2 Circumcenter and circumradius

The circumcenter $\widetilde{c}_t$ of a triangle $\widetilde{t} \in \mathbb{H}^2$ is computed by the traits class as the intersection of two perpendicular bisectors of the triangle. It is thus the intersection point of two circular arcs. When the vertices of $\widetilde{t}$ have rational coordinates, the coordinates of $\widetilde{c}_t$ are algebraic numbers of degree two that can be written as $a + b\sqrt{r}$, with $a, b, r \in \mathbb{Q}$.

The CGAL number type `Sqrt_extension` handles exact computation with algebraic numbers of degree two. Arithmetic operations between two such numbers are supported as long as they are both in the same extension. This is the case for both coordinates of $\widetilde{c}_t$ [BDT14]. It is thus possible to compute $\delta(\widetilde{c}_t, \widetilde{v})$ for a vertex $\widetilde{v}$ of $\widetilde{t}$ (recall Theorem I.2). To test if a triangle is large, $\delta(\widetilde{c}_t, \widetilde{v})$ is compared to a certified upper bound of $\cosh \varepsilon - 1$ (computed with the Boost interval library [dta]). If it is greater than the certified upper bound, t is guaranteed to be a large triangle. Note that this process may miss a large triangle if its circumradius is greater than ε but smaller than the upper bound.

Recall from Section IV.3 that the `Hyperbolic_Delaunay_triangulation_CK_traits_2` class used in our implementation requires that the vertices of the Delaunay triangulation have rational coordinates. This implies that $\widetilde{c}_t$ has to be rounded to a point with rational coordinates before being inserted into the triangulation. Each coordinate of $\widetilde{c}_t$ is rounded to a double precision number using the `to_double()` function of CGAL and is then converted to the rational number type chosen by the user.

Additionally, we can increase the precision of this rounding method by rounding the coordinates of $\widetilde{c}_t$ to a `CGAL::Gmpfr`, a fixed precision floating-point number based on the MPFR library, instead of

rounding them to a `double`. However, this method requires the user to use the `CGAL::Gmpq` number type, an arbitrary precision rational number based on the GMP library, for the coordinates of the vertices. The precision of the `CGAL::Gmpfr` numbers involved is $p \times 53$, where p is an integer given to the `epsilon_net` method and 53 is the precision of a `double`. The rounding method using the `double` number type is used only when the program uses a rational number type different from `CGAL::Gmpq`.

Note that these rounding methods do not certify that this rational approximation of $\widetilde{c}_t$ is at distance at most ε from other vertices of the triangulation, and thus its insertion into the triangulation may not result in an ε-packing.

We observe in Section IV.6 that the rounding method using the `double` number type actually constructs valid ε-nets for surfaces of low genus and with a reasonably long systole, but higher precision is needed for surfaces of higher genus or with a small systole.

We discuss how the ε-covering and ε-packing properties are checked in Section IV.4.6.

IV.4.3 Point location

The `DTHS` class provides methods to locate a query point, given as a point in $\mathbb{H}^2$, in the Delaunay triangulation of S encapsulated in a `DTHS` instance.

We only mentioned the straight walk algorithm in the description of the ε-net algorithm (recall Section III.4.2). This is because the number of visited triangles during the straight walk algorithm is easy to study, while it is unknown for the visibility walk even in the Euclidean case [DPT02, Sec 2]. Since we have shown in Chapter II that the visibility walk terminates in a periodic hyperbolic Delaunay triangulation of $\mathbb{H}^2$, we implemented both the straight walk and the visibility walk. We compare their efficiency in practice within the ε-net algorithm in Section IV.6.

These two walk algorithms are implemented in the `DTHS` class in the `locate_straight_walk` and `locate_visibility_walk` methods. The `locate_visibility_walk` method is declared as shown in Snippet IV.2. The `locate_straight_walk` method is declared similarly.

Snippet IV.2: Declaration of the `locate_visibility_walk` method.

```
template<class Traits>
typename Delaunay_triangulation_on_hyperbolic_surface_2<Traits>::Anchor
Delaunay_triangulation_on_hyperbolic_surface_2<Traits>::
locate_visibility_walk(Point const & query, Locate_type & lt, unsigned & li
    , unsigned & ld, Anchor const & hint)
{
    // code of the method
}
```

Both methods obviously take the query point and an anchor, representing the triangle of $\mathbb{H}^2$ in which the walk starts, as parameters. The walks are performed in the lifted triangulation in $\mathbb{H}^2$ by successively computing lifts of triangles along the way. They both return the anchor representing the lift of the triangle of $\mathbb{H}^2$ in which the query point is located. Moreover, three variables are passed as parameters. These variables are modified during the execution of the method to be associated with useful information about the point location. The `lt` variable represents the locate type of the query point, that is, whether the point is located in a face, on an edge or on a vertex of the returned triangle. When the query point lies on an

edge or a vertex, the variable `li` represents its index. The variable `ld` (for "locate distance") represents the number of visited triangles during the walk. Its value is zero if the query point lies in the starting triangle.

Side of a geodesic segment

Both walk algorithms rely on a predicate that determines the orientation of a query point $\widetilde{q} \in \mathbb{H}^2$ relative to an oriented geodesic line in $\mathbb{H}^2$, that is, whether it lies on the right side, left side or on the geodesic. The Delaunay triangulation traits class provides a predicate that determines the position of the query point relative to the disk defined by a geodesic line, that is, whether it lies inside, outside or on its boundary. This is not exactly what we need here, but we use this predicate to implement the one that we need.

Let $\widetilde{x}, \widetilde{y} \in \mathbb{H}^2$. When the geodesic line is a circular arc of a disk $\widetilde{D}$; if $\widetilde{q}$ lies outside of $\widetilde{D}$, then the orientation of $\widetilde{q}$ relative to $\widetilde{x}\widetilde{y}$ depends on the orientation of $\widetilde{x}, \widetilde{y}$ and the origin of the Poincaré disk: $\widetilde{q}$ lies on the left side of $\widetilde{x}\widetilde{y}$ if $\widetilde{x}, \widetilde{y}$ and the origin are oriented counterclockwise and it lies on the right side if they are oriented clockwise (Figure IV.1, left). If $\widetilde{q}$ lies inside $\widetilde{D}$, it is the converse: $\widetilde{q}$ lies on the right side of $\widetilde{x}\widetilde{y}$ if $\widetilde{x}, \widetilde{y}$ and the origin are oriented counterclockwise and it lies on the left if they are oriented clockwise (Figure IV.1, right). Determining orientation of $\widetilde{q}$ relative to $\widetilde{x}\widetilde{y}$ is trivial when the geodesic line $\widetilde{x}\widetilde{y}$ is a Euclidean line, that is, when $\widetilde{x}, \widetilde{y}$ and the origin are collinear.

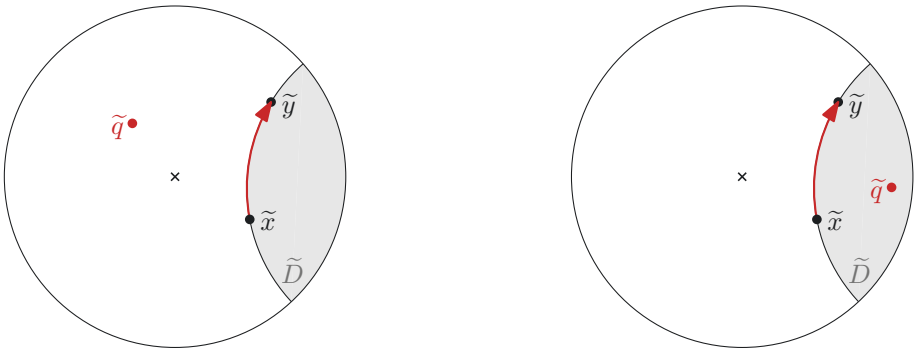

Figure IV.1: Orientation of $\widetilde{q}$ relative to the oriented geodesic $\widetilde{x}\widetilde{y}$.

The straight walk algorithm

The straight walk method follows the pseudo-code algorithm described in [DPT02, Appendix].

We again denote $\widetilde{q}$ the query point to be located in the triangulation. The algorithm starts with an initialization phase (Figure IV.2). Three points $\widetilde{p}_0, \widetilde{r}, \widetilde{l}$ are initialized with the three vertices of the starting anchor. A dart δ is initialized as the dart of the starting anchor, which corresponds to the edge (p_0, r). The goal of the initialization phase is to rotate around $\widetilde{p}_0$, updating $\widetilde{r}$ and $\widetilde{l}$ along the way, until the geodesic line $\widetilde{p}_0\widetilde{q}$ intersects the geodesic segment $\widetilde{r}\widetilde{l}$. In other words, we want $\widetilde{r}$ to be on the right side of the oriented geodesic line $\widetilde{p}_0\widetilde{q}$ and $\widetilde{l}$ to be on its left side. The rotation is clockwise or counterclockwise depending on the initial orientation of the points. For example, in Figure IV.2, the rotation is counterclockwise because $\widetilde{l}$ lies on the right side of the geodesic $\widetilde{p}_0\widetilde{q}$. The dart δ moves in the combinatorial map so that it corresponds to the edge (p_0, l). Then, the new value of $\widetilde{l}$ is computed from the cross-ratio of the edge (p_0, l) and the values of $\widetilde{p}_0, \widetilde{r}$ and $\widetilde{l}$. Finally, $\widetilde{r}$ is updated to the older value of $\widetilde{l}$. This step is repeated until $\widetilde{l}$ no longer lies on the right side of $\widetilde{p}_0\widetilde{q}$.

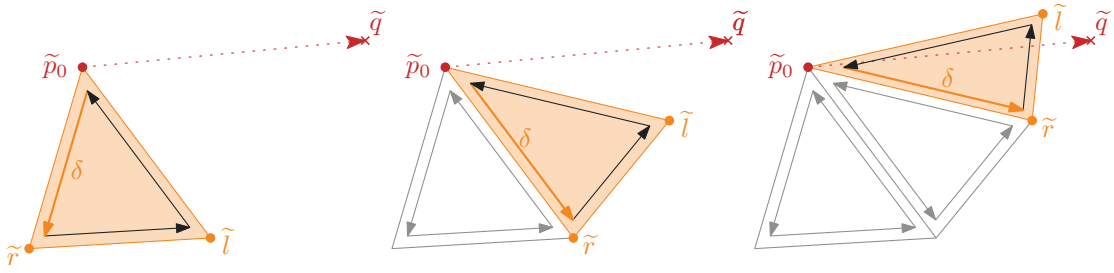

Figure IV.2: Initialization phase of the straight walk algorithm.

When the initialization phase ends, the walk along the geodesic $\widetilde{p}_0\widetilde{q}$ begins (Figure IV.3). The dart δ moves so that it corresponds to the edge (r, l). A point $\widetilde{p}$ is initialized as $\widetilde{p}_0$. Next, the fourth point $\widetilde{s}$ of the quadrilateral with vertices $\widetilde{r}, \widetilde{p}, \widetilde{l}$ and diagonal $(\widetilde{r}, \widetilde{l})$ is computed. This is done by using the values of these points and the cross-ratio of the edge (r, l) associated with δ. If $\widetilde{s}$ lies on the right side of the oriented geodesic $\widetilde{p}_0\widetilde{q}$, then $\widetilde{p}_0\widetilde{q}$ intersects the geodesic segment $\widetilde{l}\widetilde{s}$. The algorithm moves to the neighboring triangle through this segment. Otherwise, it moves to the neighboring triangle through the segment $\widetilde{r}\widetilde{s}$. The values of $\widetilde{r}, \widetilde{l}, \widetilde{p}$ and δ are updated accordingly. This step is repeated while $\widetilde{q}$ lies on the right side of the geodesic $\widetilde{r}\widetilde{l}$.

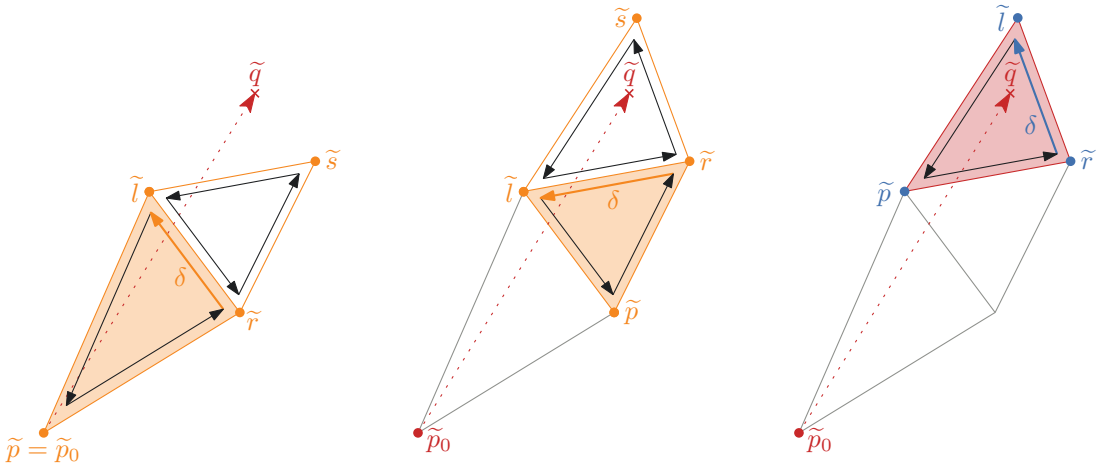

Figure IV.3: Core of the straight walk algorithm. Hyperbolic triangles are schematically represented with straight edges. Right: the returned anchor is in blue.

In the `locate_straight_walk` method, the locate distance `ld` is initialized to zero and is incremented each time δ changes face. At the end of the method, the precise locate type and locate index are computed in order to assign the final values of `lt` and `li`. Finally, the anchor $(\delta, \widetilde{r}, \widetilde{l}, \widetilde{p})$ is returned.

The visibility walk algorithm

The visibility walk algorithm starts with an initialization phase during which `ld` is set to zero and orientation tests are performed to determine whether the query point $\widetilde{q}$ lies in the starting triangle. The variables `lt` and `li` are updated accordingly. If the query point lies in the starting triangle, then the algorithm returns the corresponding anchor and terminates. Otherwise, `li` corresponds to the index i of the first edge that will be traversed by the visibility walk: $\widetilde{q}$ and the third vertex of the triangle lie on different sides of the

i-th edge of the triangle.

Three points $\widetilde{a}, \widetilde{b}, \widetilde{c}$ and a dart δ are initialized as in Figure IV.4: $\widetilde{c}, \widetilde{a}, \widetilde{b}$ respectively correspond to $\widetilde{v}_i, \widetilde{v}_{i+1}$ and $\widetilde{v}_{i+2}$, the vertices of the starting anchor (with indices taken modulo 3); and δ corresponds to the edge (c, a).

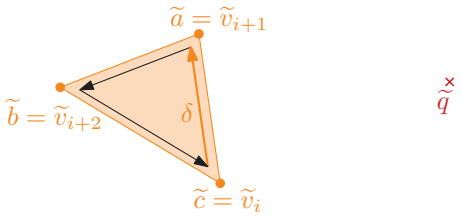

Figure IV.4: Initialization phase of the visibility walk algorithm. Hyperbolic triangles are schematically represented with straight edges.

Then, the visibility walk begins (Figure IV.5). The fourth point $\widetilde{d}$ of the quadrilateral with vertices $\widetilde{c}, \widetilde{a}, \widetilde{b}$ and diagonal $(\widetilde{c}, \widetilde{a})$ is computed from the cross-ratio of the edge (c, a) associated with δ and the values of these three vertices. If $\widetilde{q}$ lies on the right side of the oriented geodesic $\widetilde{c}\widetilde{d}$, then the algorithm goes through the edge $(\widetilde{c}, \widetilde{d})$. Otherwise, if $\widetilde{q}$ lies on the left side of the oriented geodesic $\widetilde{a}\widetilde{d}$, it goes through the edge $(\widetilde{a}, \widetilde{d})$. Note that the order in which these geodesic edges are considered is an arbitrary choice. The values of $\widetilde{a}, \widetilde{b}, \widetilde{c}$ are updated accordingly and δ moves in the adjacent face to correspond to the updated edge $(\widetilde{c}, \widetilde{a})$. If neither of these conditions is met, this means that $\widetilde{q}$ lies in the triangle $(\widetilde{a}, \widetilde{c}, \widetilde{d})$.

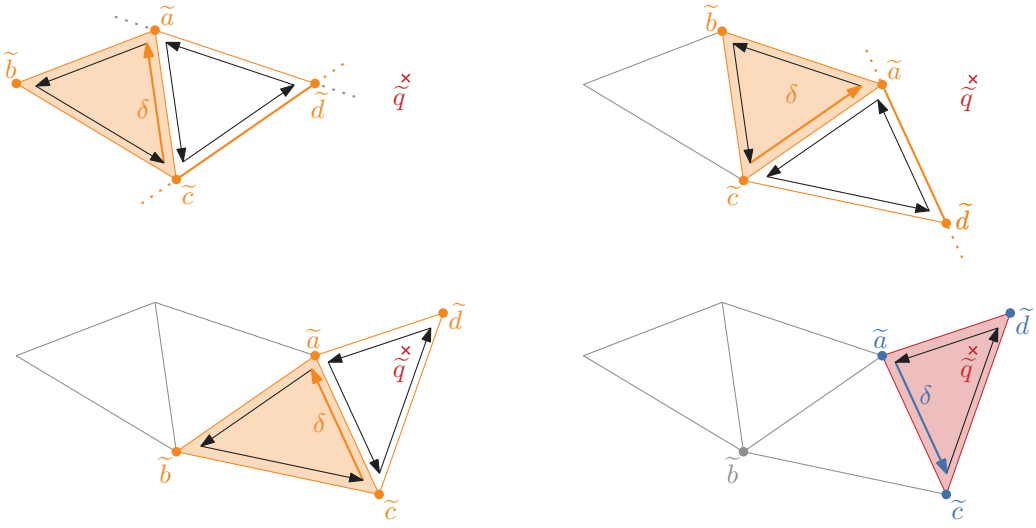

Figure IV.5: Core of the visibility walk algorithm. Hyperbolic triangles are schematically represented with straight edges. Bottom right: the returned anchor is in blue.

During the `locate_visibility_walk` method, the locate distance `ld` is incremented each time δ changes face. At the end of the method, if the query point is not in the starting triangle, the precise locate type and locate index are computed a second time in order to assign the final values of `lt` and `li`. Finally, δ moves to the neighboring face and the anchor $(\delta, \widetilde{a}, \widetilde{c}, \widetilde{d})$ is returned.

IV.4.4 Vertex insertion

The insertion of a vertex $\widetilde{q}$ in the triangulation by splitting the triangle $\widetilde{t}$ in which the point has to be inserted is handled by the `split_insert` method. It returns a list containing the anchor of each new triangle.

If $\widetilde{q}$ is inserted in the interior of $\widetilde{t}$, then $\widetilde{t}$ is split in three (Figure IV.6, left). Three pairs of darts are created and their adjacencies are set. The adjacencies of the darts that used to belong to t are updated. It remains to update or set the attributes of edges (cross-ratios) and faces (anchors).

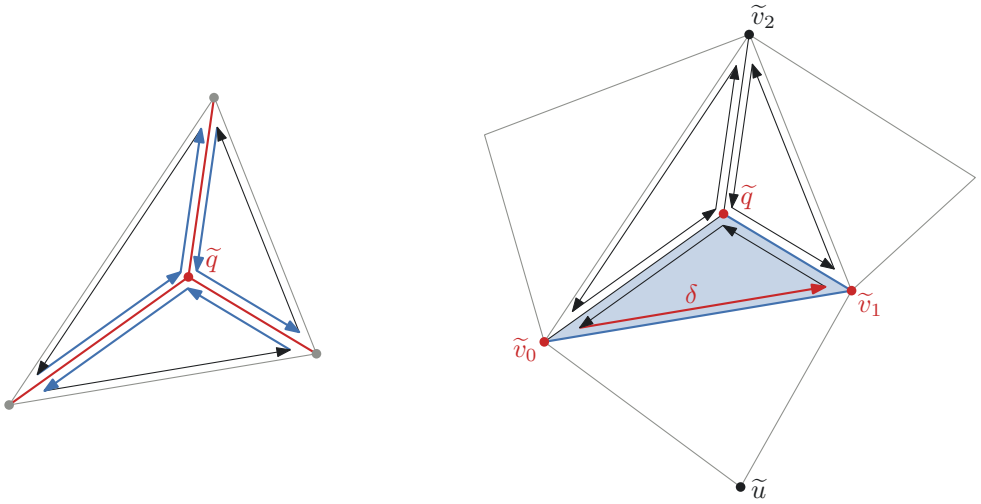

Figure IV.6: Insertion of a vertex in a face. Left: New edges are in red. New darts are in blue. Right: The blue elements are cells whose attributes are set or updated when the triangle $(\widetilde{v}_0, \widetilde{v}_1, \widetilde{q})$ is handled.

Let $(\delta, \widetilde{v}_0, \widetilde{v}_1, \widetilde{v}_2)$ be the anchor representing $\widetilde{t}$. The triangle $(\widetilde{v}_0, \widetilde{v}_1, \widetilde{q})$ is handled first (Figure IV.6, right). The fourth vertex $\widetilde{u}$ of the quadrilateral formed by $\widetilde{t}$ and its neighbor through the edge $(\widetilde{v}_0, \widetilde{v}_1)$ is computed. This is done using the cross-ratio of the edge (v_0, v_1) associated with δ and the vertex values of $\widetilde{v}_0, \widetilde{v}_1$ and $\widetilde{v}_2$. The cross-ratio of the edge (v_0, v_1) is updated with the value $[\widetilde{q}, \widetilde{v}_0, \widetilde{u}, \widetilde{v}_1]$ (Definition I.40). Recall from Section IV.1.1 that the cross-ratio is actually associated with both darts corresponding to this edge (here, δ and its opposite dart). The cross-ratio $[\widetilde{q}, \widetilde{v}_0, \widetilde{v}_1, \widetilde{v}_2]$ is associated with the new edge $(\widetilde{q}, \widetilde{v}_1)$. The anchor $(\delta, \widetilde{v}_0, \widetilde{v}_1, \widetilde{q})$ is associated with the face of the corresponding triangle. The dart δ moves to correspond to the edge $(\widetilde{v}_1, \widetilde{v}_2)$ and the triangle $(\widetilde{v}_1, \widetilde{v}_2, \widetilde{q})$ is handled similarly. The triangle $(\widetilde{v}_2, \widetilde{v}_0, \widetilde{q})$ is handled last. Note that only two cross-ratios are set or updated when handling each new triangle. The last one is set when the neighbor triangle that shares this edge is handled.

If $\widetilde{q}$ is inserted on an edge $\widetilde{e}$ of $\widetilde{t}$, then $\widetilde{t}$ and its neighbor $\widetilde{t}'$ sharing $\widetilde{e}$ are both split in two, as shown in Figure IV.7. Six darts are created (a pair for each new edge, and an additional dart for the edge that is split in two) and their adjacencies are set. The adjacencies of the darts that used to belong to t are updated. The attributes of edges and faces are handled similarly to the insertion in a face.

The `split_insert` method is hidden from the user. On the user side, the insertion of a vertex in the Delaunay triangulation is taken care of by the `insert` method of the `DTHS` class. It first calls the `split_insert` method. The Delaunay property is then restored by the `restore_Delaunay` method (Section IV.4.5).

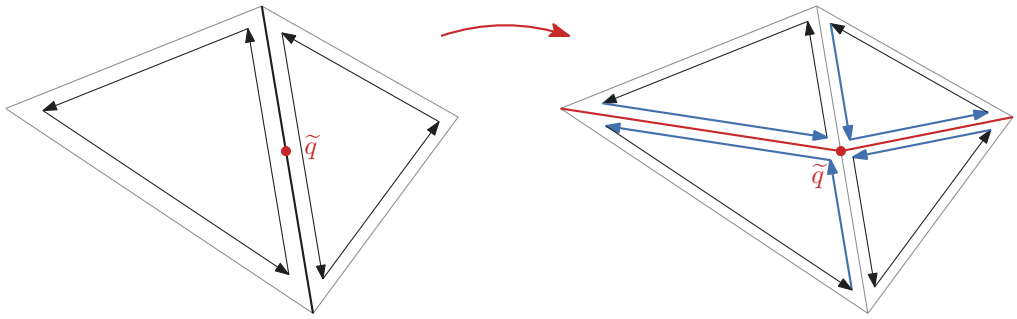

Figure IV.7: Insertion of a vertex on an edge. New edges are in red. New darts are in blue.

IV.4.5 Delaunay property restoration

The flip operation is implemented in the `flip` method of the `DTHS` class. This method takes a dart corresponding to the edge to be flipped as a parameter. It updates the anchors while the combinatorial map and cross-ratios are updated by the `THS` `flip` method, which is called internally. We refer the reader to Section III.4.1 for more details.

After the insertion of a vertex in the triangulation, the Delaunay property is restored using flips. A list of darts corresponding to edges that are flippable is initialized by the `insert` method: for each flippable edge of the new triangles, one of its darts is added to the list. Recall from Section I.4.2 that an edge is flippable if and only if the imaginary part of its cross-ratio is positive. This list is passed to the `restore_Delaunay` method. Each edge represented by a dart in the list is flipped. Its corresponding dart is removed from the list. After each flip, any edge that has become flippable has one of its darts added to the list. These edges are among those whose cross-ratios have been modified after the flip, that is, the edges of the quadrilateral formed by the two triangles adjacent to the flipped edge. Since the cross-ratios of the edges represented by a dart in the list can be modified by the flip of another edge, the `restore_Delaunay` method checks if the edge that is processed is still flippable before actually flipping it. This double-check requires minimal computation time because it only involves checking the sign of a number that is stored in the data structure. The `restore_Delaunay` method returns the total number of flipped edges.

As explained in Section IV.4.1, the `epsilon_net` method maintains a list of triangles to be processed. This necessitates knowing which triangles have been created after the insertion of a new vertex and which edges have been flipped to restore the Delaunay property. The `restore_Delaunay` method therefore updates a list of darts corresponding to the flipped edges. Instead of using the `insert` method, the `epsilon_net` method reuses the `locate`, `split_insert` and `restore_Delaunay` methods to fetch the information needed to maintain the list of triangles.

IV.4.6 Verification of ε-net properties

Our algorithm is robust in the sense that the output triangulation is the Delaunay triangulation of a set of points with rational coordinates. On the other hand, the vertices of this triangulation may not be an ε-net, due to approximations of circumcenters.

To check the ε-covering property, it is sufficient to check that there is no large triangle. The test is similar to that in Section IV.4.2. For every triangle t, the exact circumcenter $\widetilde{c}_t$ of the lift $\widetilde{t}$ represented by its anchor is computed. Then, $\delta(\widetilde{c}_t, \widetilde{v})$ is computed, where $\widetilde{v}$ is a vertex of $\widetilde{t}$. If it is less than a certified

lower bound on $\cosh \varepsilon - 1$ (computed with the Boost interval library [dta]), t is guaranteed to not be a large triangle. Note that triangles whose circumradius is at most ε but greater than the lower bound would be incorrectly detected as large. This verification algorithm can therefore produce false negatives, but never false positives: when it returns true, the set of vertices is guaranteed to be an ε-covering.

To check the ε-packing property, it is sufficient to check that all Delaunay edges are longer than ε. This works because any vertex shares a Delaunay edge with the vertex closest to it. Indeed, if $\widetilde{u}$ is a vertex of the lifted triangulation and $\widetilde{v}$ is its closest neighbor, then the circle whose diameter is the geodesic segment $\widetilde{u}\widetilde{v}$ does not contain any vertex in its interior or on its boundary. Therefore, a Voronoi edge passes between these two vertices in the dual Voronoi diagram (recall I.2.3), meaning they share a Delaunay edge. Checking whether an edge is longer than ε is done by comparing $\delta(\widetilde{u}, \widetilde{v})$, for a lift $\widetilde{u}\widetilde{v}$ of each edge (u, v), with a certified lower bound on $\cosh \varepsilon - 1$ (computed with the Boost interval library [dta]). When it is less than the lower bound, the edge is guaranteed to be shorter than ε. Of course, edges whose vertices project to the same point on S are ignored, as they do not violate the ε-packing property. Note that edges shorter than ε but longer than the lower bound could be incorrectly detected as longer than ε. This verification algorithm can therefore produce false negatives, but never false positives: when it returns true, the set of vertices is guaranteed to be an ε-packing.

Our experiments presented in Section IV.6 show that for surfaces of low genus with a long systole, our algorithm constructs valid ε-nets: using a double precision to set the rational coordinates of approximate circumcenters and computing bounds on $\cosh \varepsilon - 1$ are sufficiently accurate approximations. On the other hand, we also observe that higher precision would be needed to handle surfaces with a very small systole or with a higher genus.

IV.4.7 Management of the list $\mathcal{L}$

At each step of the algorithm, a large triangle is considered. In the original algorithm, all triangles of the current triangulation are checked until a large one is found. Consequently, all triangles that are not affected by an insertion will be checked again for the next insertion. This choice has no effect on the theoretical complexity of the algorithm, but it is time-consuming in practice. Instead, we maintain a list of triangles to be processed by the algorithm. We ran experiments to guide our choice towards an effective way of maintaining this list. The details of these experiments can be found in Section IV.6.1. The results show that any attempt to maintain the list using criteria involving circumradii, such as storing only large triangles or ordering them according to their circumradius, is highly inefficient.

More precisely, we maintain a list $\mathcal{L}$ of darts representing the triangles to be processed. It is implemented as a C++ `std::list`, which supports constant time insertion and removal of elements from anywhere in the list. The only operations on the list are: remove the *front* dart, and add a new dart to the *back*. Each dart has a mark represented by a Boolean and we maintain the property that each triangle represented in $\mathcal{L}$ has exactly one marked dart: a triangle can have several darts in $\mathcal{L}$, but only one is marked. The computation of a circumradius is *only* performed when a *marked* dart is removed. Nothing is done when a non-marked dart is removed.

The list $\mathcal{L}$ is initialized with one marked dart for each triangle of the input triangulation. Note that for a reasonable choice of ε, all the triangles of the input triangulation are large.

When a dart is removed from $\mathcal{L}$, if it is marked, it is unmarked and the circumradius of the triangle it represents is computed. If the triangle is large, its approximate circumcenter splits the triangle containing

it (Section IV.4.4). For each of the new triangles, one of its darts is marked and added to $\mathcal{L}$. Darts are added to $\mathcal{L}$ without checking the circumradius of the corresponding triangle and without checking whether it is a Delaunay triangle. The Delaunay property is restored using the flip algorithm (Section IV.4.5). After a flip, for each created triangle, one of its darts is marked and added to $\mathcal{L}$. Its two other darts are then unmarked if they were previously marked. Indeed, such a triangle may have zero, one or two marked darts (Figure IV.8).

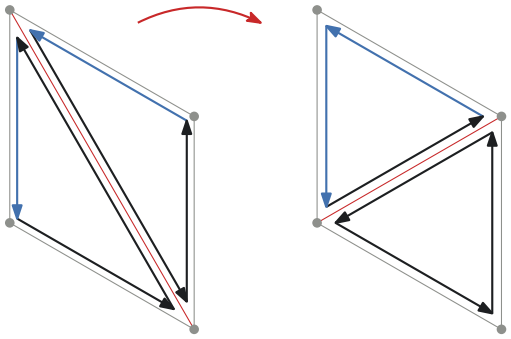

Figure IV.8: Case when triangles created by a flip have zero or two marked darts (blue).

The algorithm proceeds with the new front dart until $\mathcal{L}$ is empty. In practice, since darts representing new triangles are always added to the back of $\mathcal{L}$, large triangles tend to be close to the front and are processed before triangles with smaller circumradius.

Note that no element of $\mathcal{L}$ is ever removed from the list, except at the front. As explained in Section III.4.1, when a triangle disappears from the triangulation, its darts stay in the data structure, but their adjacencies are modified. So, it can be the case that a dart represented a large triangle when it was inserted in $\mathcal{L}$, but it represents a non-large triangle when it reaches the front of $\mathcal{L}$.

IV.5 Visualization of the output

To visualize the Delaunay triangulation of the surface S encapsulated in a `DTHS` instance, we draw a fundamental domain of S in the Poincaré disk. It suffices to draw a lift of each triangle in a connected way. We cannot directly use the anchors since this might not lead to a connected domain. Instead, we access the anchor of a chosen triangle and build lifts of the other triangles starting from this initial anchor as explained in Section I.4.2. Since objects appear smaller near the boundary of the Poincaré disk, the drawing is centered at the origin to be able to observe the details. To achieve this, the vertices of the initial anchor are translated such that one of them is the origin. The rest of the drawing is then computed from this translated lifted triangle. To get similar drawings when running the ε-net algorithm with distinct values of ε on a given surface, the initial anchor is always chosen as one of the anchors having a fixed lift $\widetilde{b}$ of b, the unique vertex of the input triangulation, as one of its vertices. This vertex is the one that is translated to the origin.

In Figure IV.9, the drawings are computed by the `lift` method of the `THS` class. It uses a weight on edges to choose the order in which the lifts of triangles are computed. The weight of an edge $(\widetilde{u}, \widetilde{v})$ is defined as $|\widetilde{u}|^2 + |\widetilde{v}|^2$ ($|\cdot|$ being the complex modulus). The drawing is then iteratively computed: given T the set of all triangles and T' the set of triangles that have already been lifted, the next triangle to be lifted is the one in $T \setminus T'$ that is incident to the edge of least weight in T'.

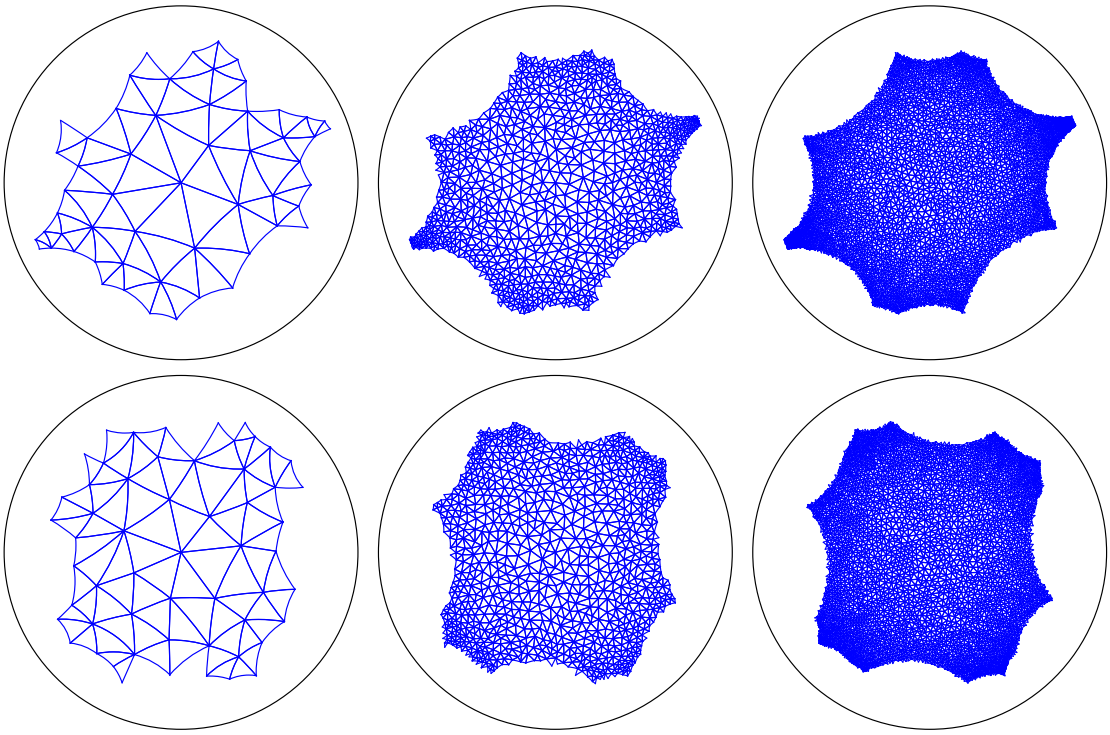

Figure IV.9: Delaunay triangulations of the computed ε-nets for $\varepsilon = 0.5$ (left), $\varepsilon = 0.1$ (middle), $\varepsilon = 0.05$ (right) of the genus 2 surfaces with seeds 123 (top row) and 321 (bottom row).

The Delaunay triangulations of ε-nets drawn in Figure IV.9 naturally look like Dirichlet domains (recall Definition I.34) because the weights on the edges ensure that the lifts of triangles entirely contained in the Dirichlet domain $\mathcal{D}(\widetilde{b})$ are drawn. This becomes clear when drawing the translation of $\mathcal{D}(\widetilde{b})$ with $\widetilde{b}$ translated to the origin (Figure IV.10, top row). Since the input of the ε-net algorithm is a Delaunay triangulation with the single vertex b, $\mathcal{D}(\widetilde{b})$ is obtained by computing the circumcenter of all the lifted triangles incident to $\widetilde{b}$ in the input triangulation (before running the ε-net algorithm).

An alternative drawing is obtained by ordering the lifts of the triangles around the initial anchor following a Breadth First Search (BFS) algorithm on the adjacency graph. Such a drawing represents a *combinatorial Dirichlet domain* (Figure IV.10, bottom row). We observe that, when ε decreases, the combinatorial Dirichlet domain of the Delaunay triangulation of an ε-net fits the Dirichlet domain $\mathcal{D}(\widetilde{b})$ better. Formalizing this convergence is an interesting open question.

IV.6 Experiments

In this section, we present the experiments conducted on the `epsilon_net` method. First, we present the experiments used to design an efficient algorithm (Section IV.6.1). We then analyze the behavior of the algorithm throughout different metrics over 180 surfaces of genus 2 (Section IV.6.2). We then look at other kinds of surfaces: surfaces of genus greater than two (Section IV.6.4) and a surface with a thin part (Section IV.6.3).

Since these results were obtained, minor changes have been applied to the code. This could result in minor changes in the numerical values. Indeed, any slight modification that impacts the order in which

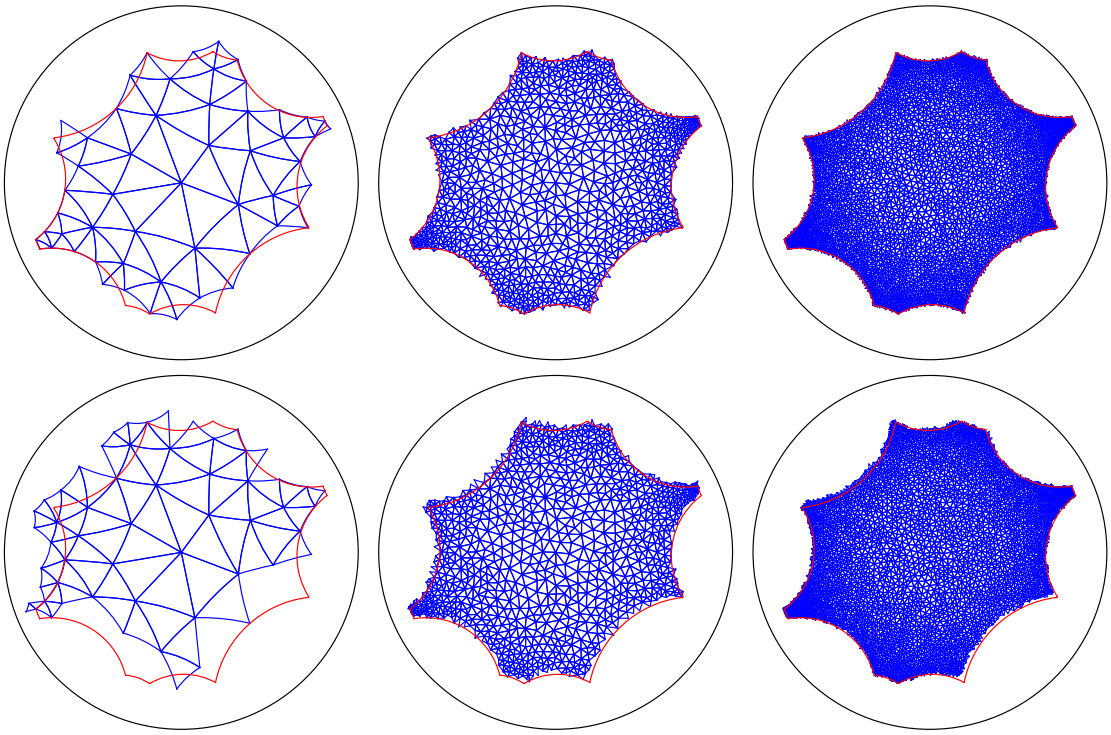

Figure IV.10: Delaunay triangulations of the computed ε-nets for $\varepsilon = 0.5$ (left), $\varepsilon = 0.1$ (middle), $\varepsilon = 0.05$ (right) of the genus 2 surface with seed 123, and the corresponding Dirichlet domain. Top row: lift of the triangulation computed with the `lift` method. Bottom row: lift of the triangulation computed with a BFS algorithm.

triangles are processed can result in the algorithm inserting a few more or fewer vertices. However, the conclusions remain identical.

IV.6.1 Experiments for design choices

The running times of the experiments of this section were obtained on a Dell Precision 3571 laptop equipped with an Intel i7-12700H CPU and 32 GB of RAM. The average is taken over the first 100 surfaces used in the experiments of Section IV.6.2.

Point location

We compared the performance of the visibility walk and the straight walk within the ε-net algorithm, as mentioned in Section IV.4.3. We measured both the average running time of the `epsilon_net` method using either walk, and the number of computed lifts when the point was not in the starting lifted triangle. These results are summarized in Table IV.1. Note that for a given surface and a given parameter ε, the obtained ε-net can be different when using one walk or the other. This is due to implementation details of the walks, leading to darts being treated in a different order in $\mathcal{L}$ (recall Section IV.4.7).

The results show that both walks behave similarly within the ε-net algorithm. The choice of walk can therefore be made arbitrarily. In the remainder of these experiments, the `epsilon_net` method used the visibility walk for point location.

Table IV.1: For each point location algorithm: average running time in the ε-net algorithm, and average number of computed lifts when the point is not in the starting lifted triangle.

	ε	0.5	0.4	0.3	0.2	0.1	0.05
Avg. running time (s)	Straight	0.187	0.278	0.436	0.815	2.340	6.922
	Visibility	0.189	0.275	0.432	0.815	2.372	7.016
Avg. # computed lifts	Straight	1.022	1.024	1.023	1.025	1.026	1.028
	Visibility	1.025	1.029	1.027	1.030	1.028	1.029

Management of the list $\mathcal{L}$

We compared different strategies to manage the list $\mathcal{L}$ of darts representing the triangles to be considered by the algorithm, as mentioned in Section IV.4.7. Each of these strategies is implemented in an algorithm called *variant*. Variant A is the final algorithm presented in this thesis. Variant B is variant A but with darts pushed at the front of $\mathcal{L}$ instead of at the back. In variant C, the algorithm checks whether a triangle is large before deciding whether to push one of its darts into $\mathcal{L}$. Variant D is similar to variant C, but in addition, the darts of the list are ordered in a priority queue according to the circumradius of their triangle, the larger ones being at the front. The results, presented in Table IV.2, show that any attempt to sort the list in a specific way, or to optimize its memory consumption is counterproductive.

Table IV.2: Average running time (seconds) of each variant of the implementation.

Variant/ε	0.5	0.2	0.1	0.05
A	0.19	0.81	2.34	6.92
B	0.31	2.34	9.9	40.59
C	0.36	1.61	4.8	15.44
D	0.49	2.29	8.31	49.93

IV.6.2 Main results

Experiments presented in this section were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see `https://www.grid5000.fr`).

All the experiments of this section were performed on 180 random surfaces[9] of genus 2 generated by the `Hyperbolic_fundamental_domain_factory_2` (recall Section IV.2) by choosing three points uniformly at random, with respect to the Euclidean distance, in the upper half of the Poincaré disk. For each of the 180 random surfaces, an input for the ε-net algorithm was computed as a single vertex Delaunay triangulation of the surface. The `epsilon_net` method was run with 50 values of ε on every input, decreasing from 0.5 to 0.01 with a step of 0.01. All the outputs are certified ε-nets when rounding circumcenters using the `to_double` function.

As explained in Section IV.2.2, these surfaces are expected to have a long systole. Experiments show that 173 of these surfaces have a systole greater than 0.5, while the 7 others have a systole greater than 0.21. Section III.5.5 details how these bounds were obtained.

[9]This number of 180 surfaces was chosen so that the experiments could run on a significant number of surfaces within a reasonable time. The experiments were run on a Grid'5000 node equipped with 18 cores. By parallelizing the execution with 10 surfaces per node, each experiment took between one and two hours to complete.

We considered three metrics: the number of points in the output, the number of triangles visited to locate the approximate circumcenters and the number of flips performed to restore the Delaunay property after each insertion. The number of points serves as a sanity check. The latter two metrics are useful for understanding the behavior of the algorithm in practice. The theoretical complexity of this algorithm is $O(N^2) = O(1/\varepsilon^4)$, where N is the number of vertices in the output (recall Theorem III.6). However, this complexity is due to the point location and flip algorithms, both of which using $O(N^2)$ elementary operations in total. We might reasonably expect to achieve a better performance in practice, with approximate circumcenters located close to their parent triangles and only a few flips to restore the Delaunay property.

Number of points

We first compare the number of points in the computed ε-nets to the theoretical bounds for an ε-thick surface of genus $g = 2$ (recall Section III.2). Among the two lower bounds, we use the one that does not involve the systole: $(g-1)/\sinh^2(\varepsilon/2) = 1/\sinh^2(\varepsilon/2)$. The number of points represents 217% of the lower bound on average, with a standard deviation of 7%. The minimum is 190% and the maximum is 257% over all tested surfaces and values of ε. When it comes to the upper bound, which is $16(g-1)/\varepsilon^2 = 16/\varepsilon^2$, the number of points represents 54% of the upper bound on average, with a standard deviation of 2%. The minimum is 47% and the maximum is 63% over all tested surfaces and values of ε. The number of points in the output is therefore about halfway between the theoretical bounds, slightly closer to the upper bound than the lower bound. Table IV.3 presents an overview of the results.

Table IV.3: Average number of points in the obtained ε-nets of the 180 studied genus 2 surfaces (rounded to the nearest integer).

ε	0.50	0.40	0.30	0.20	0.10	0.05	0.01
Average number of points	34	54	96	216	865	3,454	86,314
Lower bound $(1/\sinh^2(\varepsilon/2))$	16	25	44	100	400	1,600	40,000
Upper bound $(16/\varepsilon^2)$	64	100	178	400	1,600	6,400	160,000

Point location analysis

As mentioned in Section IV.6.1, we used the visibility walk for these experiments. Although we have no complexity analysis of the visibility walk for the point location (as in the Euclidean case [DPT02]), we observe that the walk traverses a (small) constant number of triangles in practice. Recall that each walk begins from the triangle whose approximate circumcenter is being located. The average number of computed lifted triangles during each walk tends to decrease when ε becomes smaller, while the average proportion of approximate circumcenters located in their own triangle tends to increase, as shown in Figure IV.11. On average, 68% of the approximate circumcenters lie in their triangle. Over all surfaces and all tested values of ε, the maximum observed walk distance was 4 triangles from the starting triangle.

Number of flips

We counted the number of flips needed to restore the Delaunay property of the triangulation after each split. On average, it decreases when ε becomes smaller, going from 3.41 for $\varepsilon = 0.5$ to 2.41 for $\varepsilon = 0.01$, as shown in Figure IV.12. This shows that the number of flips at each insertion is a small constant in practice.

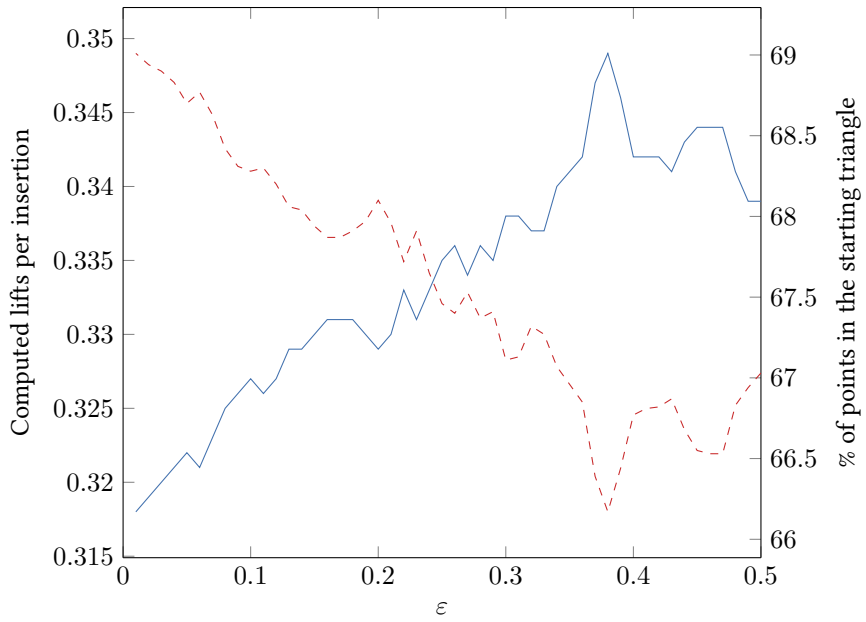

Figure IV.11: Average number of computed lifted triangles in the walk at each locate query (left, solid), and average percentage of points lying in the initial lifted triangle of the walk (right, dashed).

The total number of flips is thus, in practice, linear in the number of points, which is in contrast with the theoretical quadratic bound of Lemma III.5.

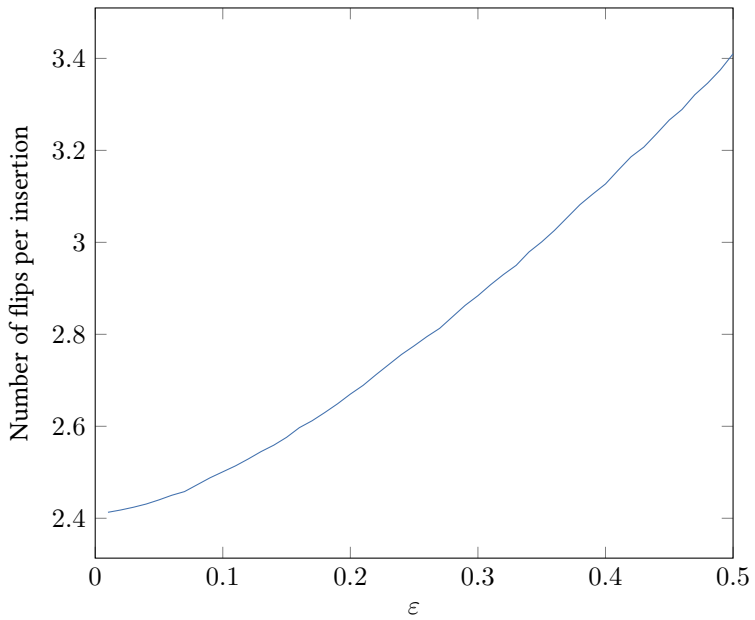

Figure IV.12: Average number of flips done to recover the Delaunay property after each insertion.

Running time

For completeness, we provide an order of magnitude of the running time of the algorithm. The complete results are summarized in Figure IV.13. All the given times are averages over the 180 surfaces used in the above experiments. These times were obtained on the Grid'5000 testbed on a node equipped with an Intel Xeon Gold 5220 CPU and 96 GB of RAM.

The average running time of the algorithm is less than 1 second for $\varepsilon \geqslant 0.23$. It is 3.47 seconds for $\varepsilon = 0.1$ (5.34×10^{-4} seconds per vertex), and 47.46 seconds for $\varepsilon = 0.02$ (3.09×10^{-4} seconds per vertex). It takes about 2 minutes and 40 seconds to run the algorithm for $\varepsilon = 0.01$ (2.55×10^{-4} seconds per vertex). The average running time of the algorithm fits the function $f(\varepsilon) = 0.0901/\varepsilon^{1.5962}$ with an r^2 greater than 0.999. The empirical running time of the algorithm is therefore sub-linear in the number of points, which is $O(1/\varepsilon^2)$ (recall Section III.2). This behavior is explained by our previous observations: both the number of lifts computed per point location and the number of flips per insertion decrease as ε decreases.

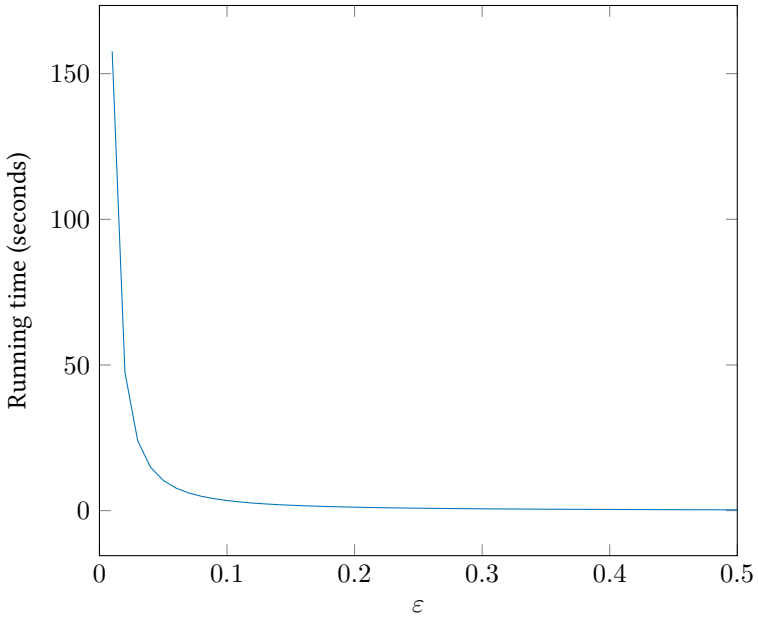

Figure IV.13: Average running time (seconds) of the ε-net algorithm.

IV.6.3 Surface with a small systole

As mentioned in Section IV.2.2, surfaces with a small systole are not common. Recall that our input is generated by choosing three random points in the Poincaré disk to form a symmetric octagonal fundamental domain. We designed a surface of genus 2 with a systole less than 10^{-4} by explicitly choosing two of these points very close together so that the corresponding side of the domain projects onto a small geodesic loop on the surface. The length of this geodesic is then an upper bound on the systole. Figure IV.14 shows the obtained Delaunay triangulation for this surface.

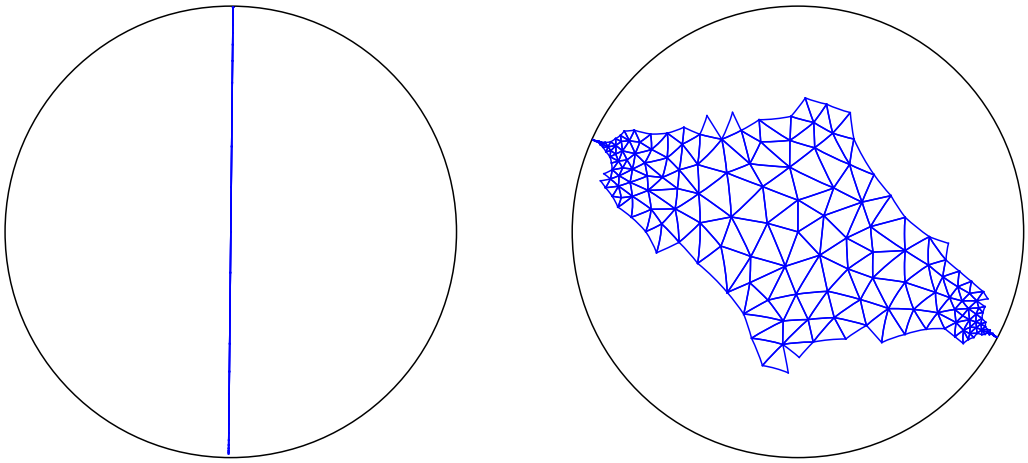

Figure IV.14: Delaunay triangulation of the computed 0.25-net of the surface with a small systole. Left: drawing centered at a vertex in the long collar. Right: drawing centered outside the collar, which appears in the form of "horns" pointing towards the boundary of the Poincaré disk.

Precision

When rounding circumcenters using the `to_double` function, the `epsilon_net` method produces an ε-net for $\varepsilon \geqslant 0.21$, which we certify using the method described in Section IV.4.6. For $\varepsilon = 0.2$, the output is a 0.2-covering but not a 0.2-packing. The shortest non-loop edge has length approximately 0.19943. This indicates that one or several circumcenters are rounded to points that violate the ε-packing property of the set of vertices. However, when rounding circumcenters using the `CGAL::Gmpfr` number type with precision 106 (twice the precision of a `double`), the `epsilon_net` method produces valid ε-nets for values of ε down to 0.1 (the lowest tested value of ε).

Number of vertices

With a systole less than 10^{-4}, the number of points in an ε-net of this surface is upper bounded by a number of the order of 10^9 according to the theoretical upper bound of $16(1/\varepsilon^2 + 1/\sigma^2)$ (see Section III.2). On the other hand, the lower bound of $1/2\varepsilon \times \operatorname{arsinh}(1/\sinh(\sigma/2))$ should double when ε is divided by 2. However, the observed number of points is far from these bounds. The results show that the number of points actually depends on the width of the ε-collar of the systole, rather than on $1/\sigma^2$. The geometric intuition is that the number of points in an ε-net of an ε-collar is linear in its width.

Recall from Definition I.21 that the width of the ε-collar of the systole σ is defined as $w(\sigma, \varepsilon) = 2\operatorname{arcosh}(\sinh(\varepsilon/2)/\sinh(\sigma/2))$. In Table IV.4, we observe that the number of vertices in the output of this surface is proportional to $16/\varepsilon^2 + w(10^{-4}, \varepsilon)/\varepsilon$.

Table IV.4: Comparison between the number of vertices in the obtained ε-nets and a proportion of $16/\varepsilon^2 + w(\sigma, \varepsilon)/\varepsilon$. The coefficient (0.5383) used is the proportion of the upper bound observed in our experiments for genus 2 surfaces with long systoles (Section IV.6.2).

ε	0.50	0.45	0.40	0.35	0.30	0.25
Number of points	58	64	75	108	137	179
$0.5383(16/\varepsilon^2 + w(10^{-4}, \varepsilon)/\varepsilon)$	54	64	78	98	127	174

IV.6.4 Surfaces of higher genus

As mentioned in Section IV.2, a few fundamental domains for hyperbolic surfaces of genus more than two have been generated by Vincent Despré and Marc Pouget. These domains are fundamental polygons whose vertices project to the same vertex on the surface. These domains can thus be handled similarly as the domains for genus 2 surfaces generated by the factory.

Despré and Pouget's generation process is, at the moment, not mature enough to generate a large number of fundamental domains. Moreover, we still do not know whether these surfaces are representative of the full moduli space. Therefore, we do not study these surfaces in detail. We only perform sanity checks by verifying whether the output is an ε-net and observing its number of points.

Genus 3 surfaces

We study three surfaces of genus 3. Their systoles have length at least 0.49.

When rounding circumcenters using the `double` number type, the `epsilon_net` method produces a valid ε-net on all three surfaces for values of ε down to 0.05 (the lowest tested value of ε). The outputs for $\varepsilon = 0.25$ are shown in Figure IV.15.

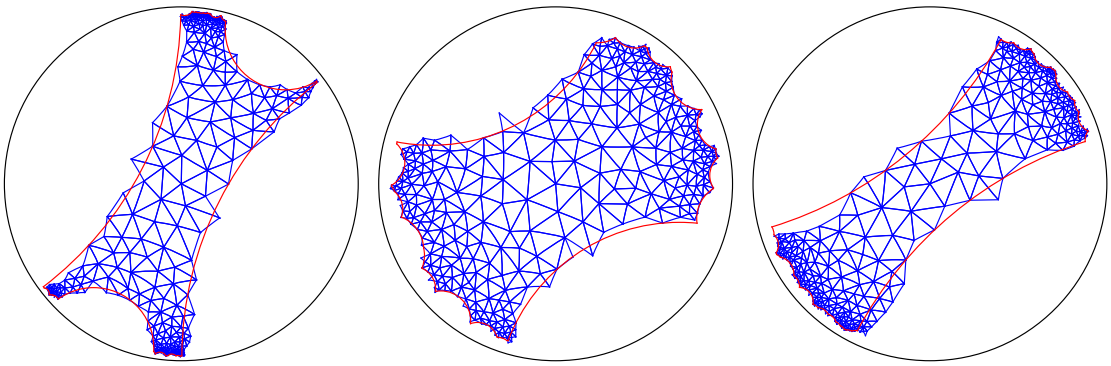

Figure IV.15: Delaunay triangulation of the computed 0.25-net of the genus 3 surfaces of Section IV.6.4 with the corresponding Dirichlet domain. The lift of triangulation is computed with the `lift` method.

The number of points in the output ε-nets follows a similar pattern as genus 2 surfaces presented in Section IV.6.2. The average number of vertices is actually on average 200% the observed number of vertices for genus 2 surfaces. It represents 218% of the lower bound $(g-1)/\sinh^2(\varepsilon/2) = 2/\sinh^2(\varepsilon/2)$ and 54% of the upper bound $16(g-1)/\varepsilon^2 = 32/\varepsilon^2$. The results are summarized in Table IV.5.

Table IV.5: Average number of points in the obtained ε-nets of the three studied genus 3 surfaces (rounded to the nearest unit).

ε	0.50	0.40	0.30	0.20	0.10
Average number of points	70	108	193	428	1727
Lower bound $(2/\sinh^2(\varepsilon/2))$	31	49	88	199	799
Upper bound $(32/\varepsilon^2)$	128	200	356	800	3,200

Higher genus surfaces

We study one surface for each genus up to 9.

When rounding circumcenters using the `double` number type, the `epsilon_net` method does not result in an ε-net for surfaces of genus at least 5. For example, for the genus 5 surface with $\varepsilon = 0.5$ and $\varepsilon = 0.25$, the obtained sets of points are ε-coverings, but not ε-packings. For these two values of ε, the shortest non-loop edges have respective lengths 0.493132 and 0.249986. This indicates that one or several circumcenters are rounded to points that violate the ε-packing property of the set of vertices. The situation worsens for the genus 7 surface: the output for $\varepsilon = 0.5$ is neither a 0.5-covering, nor a 0.5-packing. The shortest non-loop edge has length about 0.12. The output has only 81 vertices (compared to an expected number of 204). This indicates that many large triangles are missed. Indeed, while the largeness test is done with exact computation (recall Section IV.4.2), the algorithm removes the dart of each processed large triangle in the list $\mathcal{L}$ (recall Section IV.4.7). So it could happen that a lifted circumcenter $\widetilde{c}$ of a large triangle t is rounded to a point that is far away from the exact value of $\widetilde{c}$, and the insertion of this approximate circumcenter do not affect t. Then, the dart representing t is removed from $\mathcal{L}$, but t is still large and may never be broken during the execution of the algorithm.

However, when rounding circumcenters using the `CGAL::Gmpfr` number type with precision 106 (twice the precision of a `double`), the `epsilon_net` method produces valid ε-nets for values of ε down to 0.1 (the lowest tested value of ε) and for surfaces with genus up to 8. For the genus 9 surface, a precision of at least 159 (three times the precision of a `double`) must be chosen to obtain a valid ε-net. This shows that the precision of the approximation of circumcenters is the cause of the former failings and that increasing the precision solves this issue. Figure IV.16 shows the output for the surfaces of genus 5, 7 and 9.

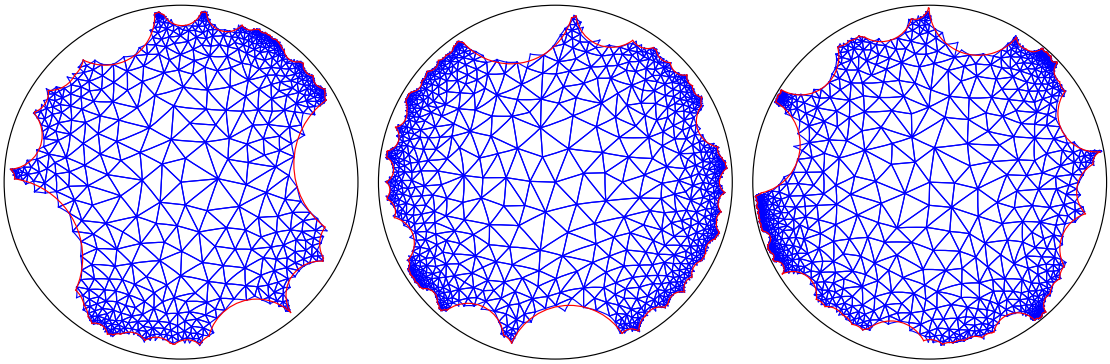

Figure IV.16: Delaunay triangulation of the computed 0.25-net of the genus 5, 7 and 9 surfaces of Section IV.6.4 with their corresponding Dirichlet domains. The lifts of the triangulations are computed with the `lift` method.

We also remark that the algorithm takes three minutes to run on the genus 9 surface with $\varepsilon = 0.5$. Therefore, it would be unreasonable to run the algorithm on a surface with a genus significantly higher or with ε significantly lower.

Conclusion

In this thesis, we have designed and implemented an algorithm to compute an ε-net of a hyperbolic surface. The motivation behind this work is to use ε-nets as input for other algorithms, particularly approximation algorithms.

Our algorithm is based on the Delaunay refinement technique: starting from a Delaunay triangulation with a single vertex on the surface, the algorithm inserts, one by one, the circumcenters of triangles whose circumciradius is greater than the parameter ε. Each insertion involves locating the point in the triangulation, splitting the triangle containing it, and restoring the Delaunay property with a flip algorithm.

We began by providing upper and lower bounds on the number of points in an ε-net (Section III.2). The number of points depends on the area of the surface (and therefore its genus) and on its systole σ when $\sigma < \varepsilon$. Indeed, the topology of a hyperbolic surface around a small simple closed geodesic is a cylinder that can be arbitrarily long as the curve becomes arbitrarily short. Our experiments show that our bounds are the correct order of magnitude for surfaces with a systole larger than ε: the number of points is of the order of $1/\varepsilon^2$ (Section IV.6.2). However, an experiment on a surface with a small systole shows that the upper bound is overestimated for such a surface: the number of points in the ε-collar of σ appears to scale linearly with its width rather than with $1/\sigma^2$ (Section IV.6.3). This suggests that the theoretical upper bound could be refined.

To handle surfaces with a small systole, we extended the ε-net algorithm to compute what we call a pseudo ε-net, which is an ε-net on the thick part of the surface (Section III.5). This algorithm could be adapted to the implemented data structure for integration into our existing code. The majority of the algorithm is straightforward to adapt, except the removal of a point that has been inserted in the data structure. Additionally, all the small curves of length at most ε are computed during the pseudo ε-net algorithm. Therefore, this algorithm can be used to compute all the small curves using $\varepsilon = 2\operatorname{arsinh} 1$. However, the complexity in terms of surface parameters still needs to be analyzed.

We explained the details of the ε-net algorithm with two data structures (Chapter III). The first one allows us to easily explain the algorithm. The second one is more complex but better suited for the implementation of the ε-net algorithm as well as future approximation algorithms. The complexity of the algorithm in both data structures is $O(1/\varepsilon^4)$. This complexity is due to the point location phase and the flip algorithm to retrieve the Delaunay property after each insertion. Experiments indicate that, in practice, the point location phase and the flip algorithm to restore the Delaunay property after each insertion are performed in constant time, resulting in an observed time complexity of $O(1/\varepsilon^2)$ (Section IV.6.2). Although improving the worst-case complexity of $O(1/\varepsilon^4)$ is unlikely, proving that the average-case complexity of the algorithm is $O(1/\varepsilon^2)$ would support the observed time complexity.

We presented our implementation of the ε-net algorithm with the CGAL library, which is designed

for exact computations (Chapter IV). This ensures that the obtained ε-net can be used as input for future algorithms. However, our implementation requires all of the involved numbers to be rational. We thus have to round the inserted circumcenters to points with rational coordinates. Depending on the user's settings, this is done by converting the algebraic coordinates of the circumcenter to a double-precision or arbitrary-precision floating-point number, and then to a rational number type. Experiments show that the latter method produces valid ε-nets for surfaces of any genus when choosing a large enough precision. In particular, the required precision for the output to be a valid ε-net increases with the genus.

An immediate continuation of this thesis would be to perform additional experiments with surfaces of higher genus once the factory for generating surfaces of any genus is ready. This would allow us to confirm the results observed for genus 2 surfaces.

An other direction for future work would be to pursue the original motivation for this thesis: designing and implementing approximation algorithms. We could begin with an algorithm to approximate distances on a hyperbolic surface and extend it to approximate its diameter. Moreover, computing an ε-net could serve as a preprocessing step for other algorithms. For instance, Iordanov's adaptation of Bowyer's algorithm on the Bolza surface [Ior19] could be generalized to other hyperbolic surfaces by using an ε-net with a well-chosen ε as the set of dummy points.

Additionally, as mentioned in Section IV.5, a combinatorial Dirichlet domain corresponding to the Delaunay triangulation of an ε-net of a hyperbolic surface appears to fit a Dirichlet domain as ε decreases to zero. This suggests that the metric induced by the graph of the Delaunay triangulation of an ε-net converges in some sense to the metric of the surface when ε decreases to zero. Formalizing and proving this convergence is an interesting open question.

Bibliography

[Abi81] William Abikoff. The uniformization theorem. *The American Mathematical Monthly*, 88(8):574–592, 1981. URL: http://www.jstor.org/stable/2320507.

[ADBC+05] Aline Aigon-Dupuy, Peter Buser, Michel Cibils, Alfred F. Künzle, and Frank Steiner. Hyperbolic octagons and Teichmüller space in genus 2. *Journal of Mathematical Physics*, 46(3):033513, 02 2005. doi:10.1063/1.1850177.

[Aud03] Michèle Audin. *Geometry*. Springer Berlin Heidelberg, 2003. doi:10.1007/978-3-642-56127-6.

[BDT14] Mikhail Bogdanov, Olivier Devillers, and Monique Teillaud. Hyperbolic Delaunay complexes and Voronoi diagrams made practical. *Journal of Computational Geometry*, 5(1):56–85, March 2014. doi:10.20382/jocg.v5i1a4.

[Bea83] Alan F. Beardon. *The Geometry of Discrete Groups*. Graduate Texts in Mathematics. Springer New York, 1st edition, 1983. doi:10.1007/978-1-4612-1146-4.

[Bel] Jean Bellissard. The structure of Delone sets. URL: https://web.archive.org/web/20240426074223fw_/https://jeanbel.math.gatech.edu/Publi/deloneJMP21.pdf.

[Ber48] Joseph Bertrand. Démonstration d'un théorème de M. Gauss. *Journal de Mathématiques Pures et Appliquées*, 1e série, 13:80–82, 1848. URL: http://www.numdam.org/item/JMPA_1848_1_13__80_0/.

[BIT24] Mikhail Bogdanov, Iordan Iordanov, and Monique Teillaud. 2D hyperbolic Delaunay triangulations. In *CGAL User and Reference Manual*. CGAL Editorial Board, 6.0.1 edition, 2024. URL: https://doc.cgal.org/6.0.1/Manual/packages.html#PkgHyperbolicTriangulation2.

[BM04] Robert Brooks and Eran Makover. Random construction of Riemann surfaces. *Journal of Differential Geometry*, 68(1):121–157, 2004.

[Bor16] David Borthwick. *Spectral Theory of Infinite-Area Hyperbolic Surfaces*, volume 318 of *Progress in Mathematics*. Springer International Publishing, 2016. doi:10.1007/978-3-319-33877-4.

[Bow81] A. Bowyer. Computing Dirichlet tessellations. *The Computer Journal*, 24(2):162–166, February 1981. doi:10.1093/comjnl/24.2.162.

[BTV16] Mikhail Bogdanov, Monique Teillaud, and Gert Vegter. Delaunay triangulations on orientable surfaces of low genus. In *32nd International Symposium on Computational Geometry (SoCG)*, 2016. doi:10.4230/LIPIcs.SoCG.2016.20.

[Bus10] Peter Buser. *Geometry and Spectra of Compact Riemann Surfaces*. Modern Birkhäuser Classics. Birkhäuser Boston, 1st edition, 2010. doi:10.1007/978-0-8176-4992-0.

[Che89] L. Paul Chew. Guaranteed-quality triangular meshes. Technical report, Cornell University, April 1989. URL: https://hdl.handle.net/1813/6899.

[Cla06] Kenneth L. Clarkson. Building triangulations using ϵ-nets. In *38th annual ACM Symposium on Theory of Computing (STOC)*, pages 326–335. Association for Computing Machinery, May 2006. Extended abstract (long paper available at https://kenclarkson.org/enet_tris/p.pdf). doi:10.1145/1132516.1132564.

[CT16] Manuel Caroli and Monique Teillaud. Delaunay triangulations of closed Euclidean d-orbifolds. *Discrete and Computational Geometry*, 55(4):827, 2016. URL: https://inria.hal.science/hal-01294409, doi:10.1007/s00454-016-9782-6.

[dBCvKO08] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer, Berlin, Heidelberg, 2008. URL: http://link.springer.com/10.1007/978-3-540-77974-2, doi:10.1007/978-3-540-77974-2.

[dC76] Manfredo Perdigão do Carmo. *Differential geometry of curves and surfaces*. Prentice Hall, Englewood Cliffs, 1976.

[DDKT22] Vincent Despré, Loïc Dubois, Benedikt Kolbe, and Monique Teillaud. Experimental analysis of Delaunay flip algorithms on genus two hyperbolic surfaces, December 2022. URL: https://hal.inria.fr/hal-03462834.

[DDPT25] Vincent Despré, Loïc Dubois, Marc Pouget, and Monique Teillaud. 2D triangulations on hyperbolic surfaces. In *CGAL User and Reference Manual*. CGAL Editorial Board, 6.1 edition, 2025. URL: https://doc.cgal.org/6.1/Manual/packages.html#PkgHyperbolicSurfaceTriangulation2.

[dFFNP91] Leila de Floriani, Bianca Falcidieno, George Nagy, and Caterina Pienovi. On sorting triangles in a Delaunay tessellation. *Algorithmica*, 6:522–532, June 1991. doi:10.1007/BF01759057.

[DH16] Olivier Devillers and Ross Hemsley. The worst visibility walk in a random Delaunay triangulation is $O(\sqrt{n})$. *Journal of Computational Geometry*, 7(1):332–359, July 2016. doi:10.20382/jocg.v7i1a16.

[DKPT23] Vincent Despré, Benedikt Kolbe, Hugo Parlier, and Monique Teillaud. Computing a Dirichlet domain for a hyperbolic surface. In *39th International Symposium on Computational Geometry (SoCG)*, volume 258, pages 27:1–27:15, 2023. doi:10.4230/LIPIcs.SoCG.2023.27.

[DKT24] Vincent Despré, Benedikt Kolbe, and Monique Teillaud. Representing infinite periodic hyperbolic Delaunay triangulations using finitely many Dirichlet domains. *Discrete & Computational Geometry*, 72(1):1–28, July 2024. doi:10.1007/s00454-024-00653-x.

[DLPT25] Vincent Despré, Camille Lanuel, Marc Pouget, and Monique Teillaud. ε-net algorithm implementation on hyperbolic surfaces. *LIPIcs, Volume 351, ESA 2025*, 351:61:1–61:18, 2025. doi:10.4230/LIPICS.ESA.2025.61.

[DLT24] Vincent Despré, Camille Lanuel, and Monique Teillaud. Computing an ε-net of a closed hyperbolic surface. In *40th European Workshop on Computational Geometry (EuroCG'24)*, pages 22:1–22:8, 2024. URL: https://eurocg2024.math.uoi.gr/data/uploads/paper_22.pdf.

[DPT02] Olivier Devillers, Sylvain Pion, and Monique Teillaud. Walking in a triangulation. *International Journal of Foundations of Computer Science*, 13:181–199, 2002. doi:10.1142/S0129054102001047.

[DST20] Vincent Despré, Jean-Marc Schlenker, and Monique Teillaud. Flipping geometric triangulations on hyperbolic surfaces. In *36th International Symposium on Computational Geometry (SoCG)*, volume 164, pages 35:1–35:16, June 2020. Final version to appear in JoCG https://jocg.org/. doi:10.4230/LIPIcs.SoCG.2020.35.

[DST24] Vincent Despré, Jean-Marc Schlenker, and Monique Teillaud. Flipping geometric triangulations on hyperbolic surfaces. *Journal of Computational Geometry*, 15(1):203–223, December 2024. doi:10.20382/jocg.v15i1a8.

[dta] BOOST development team. *BOOST C++ Libraries*. URL: http://www.boost.org.

[dtb] CORE development team. *The CORE library project*. URL: https://cs.nyu.edu/~exact/core_pages/.

[dtc] GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*. URL: https://gmplib.org.

[dtd] LEDA development team. *The LEDA user manual*. URL: https://www.algorithmic-solutions.info/leda_manual/.

[Ede90] Herbert Edelsbrunner. An acyclicity theorem for cell complexes in d dimensions. *Combinatorica*, 10(3):251–260, 1990. doi:10.1007/BF02122779.

[Edm60] John Robert Edmonds. *A Combinatorial Representation for Oriented Combinatorial Surfaces*. PhD thesis, University of Maryland, 1960. URL: http://drum.lib.umd.edu/handle/1903/24820, doi:10.13016/DAW5-MVLA.

[EITV19] Matthijs Ebbens, Iordan Iordanov, Monique Teillaud, and Gert Vegter. Delaunay triangulations of symmetric hyperbolic surfaces. In *35th European Workshop on Computational Geometry (EuroCG)*, March 2019. URL: https://inria.hal.science/hal-02940717.

[EITV22] Matthijs Ebbens, Iordan Iordanov, Monique Teillaud, and Gert Vegter. Delaunay triangulations of generalized Bolza surfaces. *Journal of Computational Geometry*, 13(1):125–177, 2022. doi:10.20382/jocg.v13i1a5.

[EP04] Euclide and François Peyrard. *Les Élémens de géométrie d'Euclide, traduits littéralement et suivis d'un Traité du Cercle, du Cylindre, du Cône et de la Sphère ; de la mesure des Surfaces et des Solides ; avec des Notes*. F. Louis (Paris), 1804. ark:/12148/bpt6k110982q. URL: https://gallica.bnf.fr/ark:/12148/bpt6k110982q.

[EPV22] Matthijs Ebbens, Hugo Parlier, and Gert Vegter. Minimal Delaunay triangulations of hyperbolic surfaces. *Discrete & Computational Geometry*, 69(2):568–592, February 2022. doi:10.1007/s00454-022-00373-0.

[FK92] Hershel M. Farkas and Irwin Kra. *Riemann Surfaces*, volume 71 of *Graduate Texts in Mathematics*. Springer, New York, NY, 1992. doi:10.1007/978-1-4612-2034-3.

[FM12] Benson Farb and Dan Margalit. *A primer on mapping class groups*. Princeton mathematical series. Princeton University Press, 2012.

[FT06] Efi Fogel and Monique Teillaud. Appendix - generic programming and the CGAL library. In Jean-Daniel Boissonnat and Monique Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*, pages 313–320. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. URL: https://inria.hal.science/hal-01053388, doi:10.1007/978-3-540-33259-6_8.

[Hat02] Allen Hatcher. *Algebraic topology*. Cambridge University Press, Cambridge ; New York, 2002.

[HHPS24] Michael Hemmer, Susan Hert, Sylvain Pion, and Stefan Schirra. Number types. In *CGAL User and Reference Manual*. CGAL Editorial Board, 6.0.1 edition, 2024. URL: https://doc.cgal.org/6.0.1/Manual/packages.html#PkgNumberTypes.

[Hil01] David Hilbert. Ueber Flächen von constanter Gaussscher Krümmung. *Transactions of the American Mathematical Society*, 2(1):87–99, 1901. doi:10.1090/S0002-9947-1901-1500557-5.

[HNU99] F. Hurtado, M. Noy, and J. Urrutia. Flipping edges in triangulations. *Discrete & Computational Geometry*, 22(3):333–346, October 1999. doi:10.1007/PL00009464.

[Ior19] Iordan Iordanov. *Delaunay triangulations of a family of symmetric hyperbolic surfaces in practice*. PhD thesis, Université de Lorraine, March 2019. URL: https://inria.hal.science/tel-02072155.

[IT17] Iordan Iordanov and Monique Teillaud. Implementing Delaunay triangulations of the Bolza surface. In *33rd International Symposium on Computational Geometry (SoCG)*, pages 44:1–44:15, July 2017. doi:10.4230/LIPIcs.SoCG.2017.44.

[IT19] Iordan Iordanov and Monique Teillaud. 2D periodic hyperbolic triangulations. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.14 edition, 2019. URL: https://doc.cgal.org/6.0.1/Manual/packages.html#PkgPeriodic4HyperbolicTriangulation2.

[Law77] C. L. Lawson. Software for c1 surface interpolation. In John R. Rice, editor, *Mathematical Software*, pages 161–194. Academic Press, January 1977. doi:10.1016/B978-0-12-587260-7.50011-X.

[Mir13] Maryam Mirzakhani. Growth of Weil–Petersson volumes and random hyperbolic surface of large genus. *Journal of Differential Geometry*, 94(2):267–300, 2013.

[Mon22] Laura Monk. Benjamini–Schramm convergence and spectra of random hyperbolic surfaces of high genus. *Analysis & PDE*, 15(3):727–752, 2022.

[Pet17] Bram Petri. Random regular graphs and the systole of a random surface. *Journal of Topology*, 10(1):211–267, 2017.

[Rat19] John G. Ratcliffe. *Foundations of Hyperbolic Manifolds*. Graduate Texts in Mathematics. Springer International Publishing, 3rd edition, 2019. doi:10.1007/978-3-030-31597-9.

[Rup95] J. Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, 18(3):548–585, May 1995. URL: https://linkinghub.elsevier.com/retrieve/pii/S0196677485710218, doi:10.1006/jagm.1995.1021.

[SBPK23] Huck Stepanyants, Alan Beardon, Jeremy Paton, and Dmitri Krioukov. Diameter of compact Riemann surfaces, 2023. URL: https://arxiv.org/abs/2301.10844.

[She02] Jonathan Richard Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications*, 22(1):21–74, May 2002. doi:10.1016/S0925-7721(01)00047-5.

[Sut09] Wilson A Sutherland. *Introduction to Metric and Topological Spaces*. Oxford University Press, 06 2009. doi:10.1093/oso/9780199563074.001.0001.

[The24] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 6.0.1 edition, 2024. URL: https://doc.cgal.org/6.0.1/Manual/packages.html.

[Tre03] Andrejs Treibergs. The hyperbolic plane and its immersions into $\mathbb{R}^3$, 2003. URL: https://www.math.utah.edu/~treiberg/Hilbert/Hilber.pdf.

[Wat81] D. F. Watson. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *The Computer Journal*, 24(2):167–172, February 1981. doi:10.1093/comjnl/24.2.167.

[ZVC80] Heiner Zieschang, Elmar Vogt, and Hans-Dieter Coldewey. *Surfaces and Planar Discontinuous Groups*, volume 835 of *Lecture Notes in Mathematics*. Springer Berlin Heidelberg, 1980. doi:10.1007/bfb0089692.

Appendix A

Résumé en Français

A.1 Introduction

La *géométrie algorithmique* est un sous-domaine de l'informatique qui a émergé à la fin des années 1970 [dBCvKO08]. Elle se concentre sur la conception de structures de données et d'algorithmes pour résoudre des problèmes géométriques. Dans les algorithmes de géométrie algorithmique, chaque étape est soit une construction, soit une étape conditionnelle basée sur l'évaluation d'un prédicat. Une construction consiste à calculer un nouvel objet, comme le point d'intersection de deux droites ou le centre du cercle circonscrit d'un triangle. Un prédicat est l'évaluation d'une question binaire telle que «ces deux segments s'intersectent-ils ?» ou «ces trois points sont-ils alignés, orientés dans le sens horaire ou anti-horaire ?», ce qui revient généralement à calculer le signe d'une expression algébrique.

L'étude de la *géométrie hyperbolique* en géométrie algorithmique est plus récente. Il s'agit de la géométrie des objets à courbure négative, ce qui signifie que chaque point est un point-selle. La géométrie hyperbolique a été découverte à la fin du XVIIIe siècle par Gauss, bien qu'il n'ait jamais publié ses travaux. Quelques années plus tard, Lobatchevski et Bolyai l'ont redécouverte indépendamment [Rat19, Ch 1]. De nombreux mathématiciens célèbres, comme Riemann et Poincaré, ont étudié la géométrie hyperbolique. Ce dernier a donné son nom au disque de Poincaré, qui est le disque unité ouvert de $\mathbb{C}$ muni d'une métrique rendant le disque hyperbolique. Dans le disque de Poincaré, les droites sont des arcs de cercle ou des diamètres du disque unité. Les constructions et les prédicats sont donc algébriquement plus compliqués à calculer dans le disque de Poincaré que dans le plan euclidien. C'est un défi majeur lorsqu'il s'agit d'implémenter des algorithmes.

La géométrie hyperbolique est incontournable lors de l'étude des surfaces (objets de dimension 2) grâce au *théorème d'uniformisation* [Abi81]. Ce théorème stipule que toute surface peut être découpée pour être mise en bijection avec une partie de la sphère, du plan euclidien ou du plan hyperbolique. Par exemple, pour un cylindre et le ruban de Möbius, il s'agit du plan euclidien, et pour le plan projectif, il s'agit de la sphère. Le troisième cas est le plus courant : toute surface de genre (nombre de «trous») supérieur à deux peut être découpée afin d'être mise en bijection avec un morceau du plan hyperbolique. En d'autres termes, presque toutes les surfaces sont *hyperboliques*.

Une grande variété de problèmes ont été étudiés en géométrie algorithmique euclidienne, allant des enveloppes convexes à la génération de maillages. Beaucoup de ces problèmes reposent sur les *triangulations de Delaunay*. Une triangulation de Delaunay est un ensemble de triangles tel que le cercle

circonscrit de chaque triangle ne contient aucun sommet de la triangulation à l'intérieur. Avec une définition aussi simple, les triangulations de Delaunay sont un outil essentiel en géométrie algorithmique. Naturellement, elles furent l'un des premiers problèmes étudiés en géométrie hyperbolique et sur les surfaces hyperboliques [BDT14, BTV16, IT17].

La surface de Bolza est une surface hyperbolique de genre 2 qui peut être découpée en un octogone hyperbolique régulier. C'est la surface hyperbolique la plus étudiée grâce à sa symétrie qui facilite les calculs. Les surfaces de Bolza généralisées, qui sont des surfaces de genre supérieur pouvant être découpées en polygones réguliers, sont également assez bien étudiées. Cependant, de nombreuses questions résolues pour la surface de Bolza ou les surfaces de Bolza généralisées restent ouvertes pour des surfaces hyperboliques génériques, par exemple trouver leur diamètre [SBPK23] ou la longueur de la courbe non contractile la plus courte [Ior19, Thm II.17].

Pour aider à combler cette lacune, l'objectif de cette thèse est de progresser vers la conception et l'implémentation d'algorithmes d'approximation sur des surfaces hyperboliques. De tels algorithmes seraient des outils précieux pour mieux comprendre les surfaces hyperboliques et explorer des conjectures. Pour concevoir un algorithme d'approximation, la première étape consiste à trouver un moyen d'approcher la géométrie de la surface à l'aide d'un ensemble fini de points bien répartis sur celle-ci. Ainsi, tout point de la surface est proche d'un point de cet ensemble. En plus d'être une entrée pour les algorithmes d'approximation, un tel ensemble de points pourrait servir dans d'autres algorithmes. Par exemple, l'algorithme de Bowyer, un algorithme incrémental de triangulation de Delaunay, a été adapté à la surface de Bolza mais nécessite d'abord le calcul d'un ensemble de points factices. Cette approche pourrait être généralisée à n'importe quelle surface hyperbolique en utilisant un ε-filet avec un ε bien choisi comme ensemble de points factices.

La géométrie algorithmique est utile dans de nombreux domaines tels que l'informatique graphique, la robotique, la génération de maillages ou la vision par ordinateur. Cependant, la plupart des travaux dans ce domaine sont théoriques et peu d'algorithmes ont été implémentés. La bibliothèque CGAL, la plus grande bibliothèque open-source de géométrie algorithmique, vise à combler le fossé entre théorie et pratique. Elle rassemble des structures de données et des algorithmes de géométrie algorithmique dans une bibliothèque logicielle commune pour fournir des programmes fiables et efficaces aux utilisateurs académiques ou industriels. Afin de participer aux aspects pratiques de la géométrie algorithmique, il était essentiel pour nous d'implémenter l'algorithme développé au cours de cette thèse et de l'intégrer dans CGAL.

La notion d'ε-filet [Cla06] correspond à l'ensemble souhaité de points bien distribués mentionné ci-dessus. Pour un paramètre $\varepsilon > 0$, un ε-filet P d'une surface est un sous-ensemble tel que tout point de la surface est à une distance d'au plus ε de P et que deux points distincts de P sont distants d'au moins ε. La première propriété assure que toute la surface est couverte, de sorte que tout point de la surface peut être approximé par un point de P. La seconde propriété assure que P n'est pas inutilement dense.

Nous avons donc cherché un algorithme permettant de calculer des ε-filets sur des surfaces hyperboliques. Comme l'objectif est d'utiliser les ε-filets comme base pour des algorithmes d'approximation destinés à une utilisation pratique, nous avons également visé l'implémentation de l'algorithme. Cette implémentation doit gérer le calcul exact pour pouvoir réutiliser l'ε-filet calculé comme entrée pour un algorithme d'approximation. La bibliothèque CGAL est donc l'outil parfait pour l'implémentation car elle est conçue pour gérer les calculs exacts et intègre déjà de la géométrie hyperbolique. De plus, nous avons cherché un moyen de gérer des surfaces ayant des cylindres hyperboliques (arbitrairement) longs,

conduisant à un nombre (arbitrairement) grand de points dans un ε-filet.

A.2 Contributions

Dans cette thèse, nous avons conçu et implémenté un algorithme permettant de calculer un ε-filet d'une surface hyperbolique. La motivation derrière ce travail est d'utiliser ces ε-filets comme entrée pour d'autres algorithmes, en particulier des algorithmes d'approximation.

A.2.1 Description de l'algorithme

Notre algorithme est basé sur la technique de raffinement de Delaunay : à partir d'une triangulation de Delaunay avec un seul sommet sur la surface, l'algorithme insère, un par un, les centres circonscrits des triangles dont le rayon du cercle circonscrit est strictement supérieur au paramètre ε. Chaque insertion implique de localiser le point dans la triangulation, de diviser le triangle qui le contient et de restaurer la propriété de Delaunay à l'aide d'un algorithme de bascule d'arête (flip).

Nous avons d'abord fourni des bornes supérieures et inférieures sur le nombre de points dans un ε-filet (Section III.2). Le nombre de points dépend de l'aire de la surface (et donc de son genre) et de sa systole σ lorsque $\sigma < \varepsilon$. En effet, la topologie d'une surface hyperbolique autour d'une petite géodésique fermée simple est un cylindre qui peut être arbitrairement long à mesure que la courbe devient arbitrairement courte.

Nous détaillons l'algorithme de calcul d'un ε-filet avec deux structures de données (Chapitre III). La première est basée sur une représentation de la surface par un domaine de Dirichlet. Elle nous permet d'expliquer facilement l'algorithme. La seconde est plus complexe mais mieux adaptée à l'implémentation de l'algorithme de calcul d'un ε-filet ainsi qu'à de futurs algorithmes d'approximation. Il s'agit de représenter la triangulation de la surface par une carte combinatoire, à laquelle on ajoute des informations géométriques: un birapport pour chaque arête, et une ancre (représentant un relevé d'un triangle) pour chaque face.

La complexité de l'algorithme dans les deux structures de données est de l'ordre de $O(1/\varepsilon^4)$ pour une surface donnée. La constante dans la notation O varie selon la structure de données utilisée. Dans les deux cas, cette complexité est due à la phase de localisation du point à insérer et à l'algorithme de bascule permettant de retrouver la propriété de Delaunay après chaque insertion.

A.2.2 Pseudo ε-net

Puisque le nombre de points dans un ε-filet augmente lorsque la systole diminue, nous voulons rompre cette dépendance. Pour y parvenir, nous étendons l'algorithme de calcul d'un ε-filet afin de calculer ce que nous appelons un pseudo ε-filet. Il s'agit d'un ε-filet sur la partie épaisse de la surface (Section III.5). De plus, toutes les petites courbes de longueur au plus ε sont calculées durant l'algorithme de pseudo ε-filet. Par conséquent, cet algorithme peut être utilisé pour calculer toutes les petites courbes en utilisant $\varepsilon = 2 \operatorname{arsinh} 1$. Cependant, la complexité en termes des paramètres de la surface reste à analyser.

A.2.3 Implémentation

Nous implémentons l'algorithme de calcul d'un ε-filet d'une surface hyperbolique à l'aide de la bibliothèque CGAL, qui est conçue pour les calculs exacts (Chapitre IV). Cela garantit que le ε-filet obtenu peut être utilisé

comme entrée pour de futurs algorithmes. Notre implémentation repose sur le paquet de triangulations de surfaces hyperboliques déjà présent dans CGAL. Ce paquet implémente une structure de données basée sur une carte combinatoire, avec un birapport pour chaque arrête et une ancre pour l'ensemble de la triangulation. Nous réutilisons cette structure de données, via l'héritage, pour rajouter une ancre pour chaque face de la triangulation.

Le plus grand défi de cette implémentation est de garantir un algorithme à la fois robuste et efficace. Nous devons donc éviter les racines carrées en cascade causées par les centres de cercles circonscrits. Le paquet de triangulations sur des surfaces hyperboliques permet d'engendrer des domaines fondamentaux de surfaces hyperboliques de genre 2 dont les coordonnées des sommets sont des nombres rationnels. Vincent Despré et Marc Pouget ont également mis au point une méthode permettant d'engendrer des domaines fondamentaux de surfaces de genre supérieur. Ces surfaces sont denses dans l'ensemble des surfaces hyperboliques. Il n'y a donc pas de perte significative de généralité à les utiliser. Pour éviter les racines carrées en cascade causées par les centres circonscrits, nous arrondissons leurs coordonnées en rationnels au lieu d'insérer le centre exact. Cependant, cet arrondi ne garantit pas que les propriétés de l'ε-filet sont conservées à chaque insertion. Au lieu de vérifier cela à chaque insertion, ce qui serait coûteux, nous vérifions à la fin de l'algorithme si le résultat est un ε-filet valide.

Les coordonnées des centres circonscrits sont d'abord arrondies vers des nombres flottants à précision fixe, puis vers un type de nombre rationnel exact. Cela permet d'éviter que la taille des nombres n'augmente au cours de l'algorithme. La précision est choisie par l'utilisateur au moment d'exécuter l'algorithme. Si le résultat n'est pas un ε-filet valide, alors l'utilisateur peut augmenter la précision jusqu'à ce que le résultat soit un ε-filet valide.

A.2.4 Expériences

Nous expérimentons l'algorithme de calcul d'un ε-filet sur un grand nombre de surfaces de genre 2 et un petit nombre de surfaces de genre supérieur, ainsi qu'un surface avec une très petite systole (Section IV.6).

Les expériences montrent que notre méthode d'approximation des centres circonscrits par des points à coordonnées rationnelles permet d'obtenir des ε-filets valides pour des surfaces de tout genre, à condition de choisir une précision suffisante. En particulier, la précision requise pour que la sortie soit un ε-filet valide augmente avec le genre.

Nos expériences montrent que nos bornes sur le nombre de points d'un ε-filet sont du bon ordre de grandeur pour les surfaces dont la systole est supérieure ε : le nombre de points est de l'ordre de $1/\varepsilon^2$ (Section IV.6.2). Cependant, une expérience sur une surface avec une petite systole montre que la borne supérieure est surestimée pour une telle surface : le nombre de points dans le ε-collier de σ semble être linéaire par rapport à sa largeur $w(\sigma, \varepsilon)$ plutôt que d'être $1/\sigma^2$ (Section IV.6.3). Cela suggère que la borne supérieure théorique pourrait être affinée.

Les expériences indiquent que, dans la pratique, la phase de localisation du point à insérer et l'algorithme de basculement pour restaurer la propriété de Delaunay après chaque insertion s'effectuent en temps constant, ce qui se traduit par en une complexité temporelle observée de l'ordre de $O(1/\varepsilon^2)$ (Section IV.6.2).

Nous pouvons visualiser l'ε-filet obtenu en calculant un relevé de chaque triangle dans le plan hyperbolique de manière connexe (Section IV.5). Cela peut se faire en explorant les triangles avec un algorithme de parcours en largeur. La forme obtenue, que nous appelons domaine de Dirichlet combinatoire, semble converger vers un domaine de Dirichlet de la surface à mesure que ε décroît. Cela suggère que la métrique

induite par le graphe de la triangulation de Delaunay d'un ε-filet converge en un certain sens vers la métrique de la surface lorsque ε décroît.

A.3 Perspectives

Une suite immédiate de cette thèse consisterait à effectuer des expériences supplémentaires avec des surfaces de genre plus élevé, une fois que le générateur permettant de créer des surfaces de tout genre sera prêt. Cela nous permettrait de confirmer les résultats observés pour les surfaces de genre 2. De plus, bien qu'améliorer la complexité dans le pire cas de $O(1/\varepsilon^4)$ soit peu probable, prouver que la complexité moyenne de l'algorithme est de l'odre de $O(1/\varepsilon^2)$ étayerait la complexité temporelle observée.

L'algorithme de calcul d'un ε-filet pourrait également être adapté à la structure de données implémentée pour une intégration dans notre code existant. La majeure partie de l'algorithme est simple à adapter, excepté la suppression d'un point dans la structure de données.

Une autre piste pour les travaux futurs serait de poursuivre la motivation originale de cette thèse, à savoir la conception et l'implémentation d'algorithmes d'approximation. Nous pourrions commencer par un algorithme permettant d'approximer les distances sur une surface hyperbolique, puis l'étendre afin d'approximer son diamètre. De plus, le calcul d'un ε-filet pourrait servir d'étape de pré-traitement pour d'autres algorithmes. Par exemple, l'adaptation par Iordanov de l'algorithme de Bowyer sur la surface de Bolza [Ior19] pourrait être généralisée à d'autres surfaces hyperboliques en utilisant un ε-filet avec un ε bien choisi comme ensemble de points fictifs.

De plus, comme mentionné dans la Section IV.5, un domaine de Dirichlet combinatoire correspondant à la triangulation de Delaunay d'un ε-filet d'une surface hyperbolique semble correspondre à un domaine de Dirichlet lorsque ε décroît vers zéro. Cela suggère que la métrique induite par le graphe de la triangulation de Delaunay d'un ε-filet converge en un certain sens vers la métrique de la surface lorsque ε décroît vers zéro. Formaliser et prouver cette convergence constitue une question ouverte intéressante.