ϵ-Net Algorithm Implementation on Hyperbolic Surfaces

Vincent Despré, Camille Lanuel, Marc Pouget, Monique Teillaud

ε-Net Algorithm Implementation on Hyperbolic Surfaces

Vincent Despré
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Camille Lanuel
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Marc Pouget
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Monique Teillaud
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Abstract

We propose an implementation, using the CGAL library, of an algorithm to compute ε-nets on hyperbolic surfaces proposed by Despré, Lanuel and Teillaud [18]. We describe the data structure, detail the implemented algorithm and report experimental results on hyperbolic surfaces of genus 2. The implementation differs from the cited algorithm on several aspects. In particular, we use a different data structure, based on combinatorial maps, to represent a triangulation of a surface. We explain how to generate fundamental polygons to represent our input hyperbolic surfaces and the arithmetic issues related to the number type of the coordinates of their vertices.

2012 ACM Subject Classification Theory of computation → Computational geometry; Mathematics of computing → Geometric topology

Keywords and phrases Hyperbolic surface, Delaunay triangulation, Data structure, Combinatorial map, Implementation, CGAL

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.59

Related Version *Full Version*: `https://hal.science/hal-04820353v1` [17]

Supplementary Material *Source code and results*: `https://github.com/camille-lanuel/ESA_2025_implementation_epsilon_net`

Funding This work was partially supported by grant ANR-23-CE48-0017 of the French National Research Agency ANR (project Abysm).

Acknowledgements Experiments presented in this paper 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`).

1 Introduction

Hyperbolic surfaces are well studied in mathematics since hyperbolic geometry is the natural geometry on surfaces of genus larger than one [25]. However, there are long standing open problems that could be experimentally investigated like finding the hyperbolic surface of genus 3 with the longest *systole* (i.e., the length of the shortest non-contractible closed geodesic) or the hyperbolic surface of genus 2 with the smallest diameter.

A few computational tools for hyperbolic surfaces have only been available recently. Iordanov and Teillaud [27] introduced a package in CGAL [28] to construct Delaunay triangulations with Bowyer's incremental algorithm [7], only for the Bolza surface, which is the most symmetric surface of genus two. The authors in [14] proposed an implementation of edge flips to transform any triangulation of a closed hyperbolic surface into the Delaunay

triangulation. Their software [15] also generates surfaces of genus 2. The algorithm does not support insertions of new points.

An ε-net (see Section 2.2) is a natural tool to approximate distances on a surface, therefore it helps to address the aforementioned open problems. Our contribution is the implementation[1] of an ε-net algorithm proposed in [18] inspired from Shewchuk's Delaunay refinement [34]. To the best of our knowledge, it is the first implementation for this problem.

There are two candidate data structures to represent a hyperbolic surface in this context: one focuses on a fundamental domain in the universal cover [18] and the other considers the surface as a combinatorial map [14]. We use an enriched version of the data structure implemented in the CGAL package *2D triangulations on hyperbolic surfaces* [15] based on a combinatorial map.

Our implementation is independent from the genus of the surface. However, the only tractable surfaces that we can currently generate have genus two. Indeed, except for the specific case of the Bolza surface mentioned above, which involves algebraic numbers, so far the only surfaces that are reachable by exact computations are a dense subset of the set of surfaces of genus two given by a domain with rational coordinates. As mentioned in [23], even for generalized Bolza surfaces, the representation of fundamental domains with algebraic numbers in genus higher than two is an obstacle to overcome when implementing.

The algorithm iteratively inserts in the triangulation the circumcenter of a *large triangle*, which is a triangle whose circumradius is greater than ε. Even though the coordinates of a circumcenter lie in an algebraic extension of degree two with respect to the coordinates of the vertices of the triangle, our algorithm only constructs points with exact rational coordinates that approximate the circumcenters. Consequently, our algorithm only uses exact rational arithmetic for the construction of the Delaunay triangulation. The final step is to check that the vertices of this triangulation actually form an ε-net in spite of the approximation of inserted circumcenters; this is the only place where we use algebraic numbers of degree 2. As explained in Section 7.1, it is very hard to sample the set of hyperbolic surfaces with a satisfactory distribution in practice. We use the generator introduced by Dubois et al. [14], which samples a dense subset of the set of surfaces of genus 2 and describes them by rational points (Section 5). Quite noticeably, a random hyperbolic surface is likely to have a reasonably long systole, which is actually the case in our experiments, and our final ε-net check is always successful. We observe in [17, Sec 7.3] that for a surface specifically designed to have a very small systole, the final ε-net check may fail, which means that a higher precision on the rational approximation of the circumcenter would be needed.

The key component of the algorithm, alongside the Delaunay flip part, is the point location. To insert a point in a triangulation of the surface, we work in the triangulation lifted in the hyperbolic plane $\mathbb{H}^2$, and determine the lift of a triangle containing the point (Section 2.1). Similar to the Euclidean plane, there are multiple strategies for traversing the triangulation and locate a point [22]. We tested both the straight and the visibility walks, which demonstrate comparable efficiency for our purposes. Theorem 1 ensures that the visibility walk terminates successfully in the context of finite or periodic Delaunay triangulations of $\mathbb{H}^2$.

The paper is organized as follows: We discuss point location strategies in triangulations in $\mathbb{H}^2$ in Section 3. We detail our data structure in Section 4. The generation of input surfaces and the related arithmetic issues are introduced in Section 5. Our implemented algorithm is presented in Section 6. Our experiments are reported in Section 7.

[1] `https://github.com/camille-lanuel/ESA_2025_implementation_epsilon_net`

2 Background and Notation

2.1 Hyperbolic Surfaces

We refer the reader to textbooks [11, 4, 9] for more details on topology and hyperbolic surfaces. In this paper, we use the term *hyperbolic surface* for a closed (connected, compact, and without boundary) oriented hyperbolic surface. Such a surface S can be seen as the quotient of the hyperbolic plane $\mathbb{H}^2$ under the action of a discrete subgroup Γ of the group of orientation-preserving isometries of $\mathbb{H}^2$. The surface S is locally isometric to $\mathbb{H}^2$ and thus has constant curvature -1. Note that Hilbert's theorem prevents to isometrically embed $\mathbb{H}^2$ or any hyperbolic surface in $\mathbb{R}^3$, thus all our illustrations will be drawn in the Poincaré disk model. The computation of distances in $\mathbb{H}^2$ will be detailed in Section 6.2. We denote as g the genus of S, that is its number of handles, and σ its systole which is the length of the shortest non-contractible closed geodesic.

The action of Γ on $\mathbb{H}^2$ induces a projection $\rho : \mathbb{H}^2 \mapsto \mathbb{H}^2/\Gamma = S$. For $x \in S$, an element $\widetilde{x} \in \Gamma x := \rho^{-1}(x) \subset \mathbb{H}^2$ is called a *lift* of x. Throughout this paper, objects written with a tilde $\widetilde{\cdot}$ are in $\mathbb{H}^2$ while objects on S are written without.

A *fundamental domain* for S is a connected subset of $\mathbb{H}^2$ containing exactly one lift of every point of S, except on its boundary. The *Dirichlet domain* $\mathcal{D}_{\widetilde{b}}$ of a point $\widetilde{b} \in \mathbb{H}^2$ is defined as the closed Voronoi cell of $\widetilde{b}$ in the Voronoi diagram of the point set $\Gamma\widetilde{b}$. It is a fundamental polygon for S: the interiors of two copies of the domain are disjoint, $\Gamma\mathcal{D}_{\widetilde{b}} = \mathbb{H}^2$, and its boundary is made of geodesic segments called *sides*. To each side s of $\mathcal{D}_{\widetilde{b}}$, there is a unique element $\gamma_s \in \Gamma$, called *side pairing*, such that $\gamma_s(s)$ is also a side. The side pairings generate Γ, so that $\mathcal{D}_{\widetilde{b}}$ and its side pairings determine the metric of the surface.

We work with the *Poincaré disk model* in which $\mathbb{H}^2$ is represented by the open unit disk of $\mathbb{C}$ (Figure 1). The geodesics are either diameters of the disk, or circular arcs orthogonal to the unit circle. Hyperbolic circles are Euclidean circles but their centers do not coincide in general.

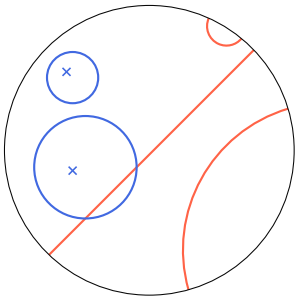

Figure 1 Geodesics (red) and hyperbolic circles (blue) in the Poincaré disk.

For a triangulation T of S, we note $\widetilde{T}$ the infinite periodic triangulation of $\mathbb{H}^2$ whose vertices, edges and faces are all lifts of those of T. Conversely, any vertex, edge or face of $\widetilde{T}$ projects on S as a vertex, edge or face of T. The triangulation T is a *Delaunay triangulation* if $\widetilde{T}$ is a Delaunay triangulation of $\mathbb{H}^2$, that is, no circumcircle of a triangle of $\widetilde{T}$ contains a vertex of $\widetilde{T}$ in its interior.

2.2 ε-Nets

Let us recall definitions [10]. 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 $x \in X$, i.e., the closed balls of radius ε centered at points

of P cover X. It is 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 points of P are pairwise disjoint. If P is both an ε-covering and an ε-packing, then it is an *ε-net*, also known as an $(\varepsilon,\varepsilon)$-Delone set.

When $\varepsilon < \sigma$, we say that S is *ε-thick*. The number of points in an ε-packing of a hyperbolic surface S is upper-bounded by $16(g-1)\left(1/\varepsilon^2 + 1/\sigma^2\right)$, or $16(g-1)/\varepsilon^2$ for an ε-thick surface [18].

2.3 Original Algorithm

Let us summarize the algorithm [18], from which our implementation derives. The algorithm is inspired by Shewchuk's Delaunay refinement [34]. It starts with a Delaunay triangulation of S with a single vertex b. In a nutshell, as long as there is a large triangle in the triangulation, its circumcenter is inserted and the triangulation is updated (see Figure 2): the triangle in which the circumcenter lies is first split into three new triangles, then the Delaunay property is retrieved using edge flips [29, 19]. The set of vertices of the final Delaunay triangulation is an ε-net of S, and all the intermediate sets of vertices are ε-packings.

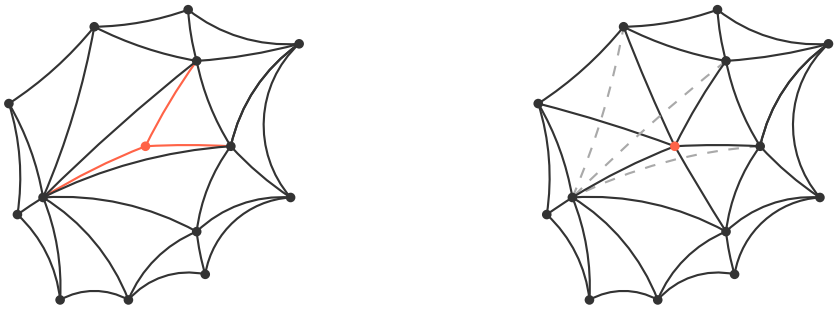

Figure 2 Insertion of a point in a Delaunay triangulation using a flip algorithm.

The algorithm also stores the Dirichlet domain $\mathcal{D}_{\tilde{b}}$ of a lift $\widetilde{b}$ of b together with its side pairings, which can be computed from any fundamental domain for S [16]. The data structure stores the lift in $\mathcal{D}_{\tilde{b}}$ of each vertex of the triangulation (or one of its lifts if it lies on the boundary of $\mathcal{D}_{\tilde{b}}$), and for each triangle, the information needed to retrieve its (at most three) lifts having at least one vertex in $\mathcal{D}_{\tilde{b}}$.

The most crucial step of the algorithm is the location of the circumcenter c_Δ of a triangle Δ in a triangulation of S. To this aim, a lift of c_Δ is located in the lifted triangulation in $\mathbb{H}^2$, as follows. First, the circumcenter $\widetilde{c}_\Delta$ of a lift $\widetilde{\Delta}$ of Δ with at least one vertex in $\mathcal{D}_{\tilde{b}}$ is computed. The point $\widetilde{c}_\Delta$ is a lift of c_Δ that does not necessarily lie in $\mathcal{D}_{\tilde{b}}$. To locate $\widetilde{c}_\Delta$, the algorithm first walks in the tiling $\Gamma\mathcal{D}_{\tilde{b}}$ to find the element $\gamma_c \in \Gamma$ such that $\widetilde{c}_\Delta \in \gamma_c\mathcal{D}_{\tilde{b}}$. The lift $\widetilde{c_b} = \gamma_c^{-1}\widetilde{c}_\Delta$ of c_Δ lying in $\mathcal{D}_{\tilde{b}}$ can then be computed. Second, the triangle in $\mathcal{D}_{\tilde{b}}$ in which $\widetilde{c_b}$ lies is identified by checking, for each triangle, its (at most three) lifts having at least one vertex in $\mathcal{D}_{\tilde{b}}$. It can be shown that the walk in the tiling $\Gamma\mathcal{D}_{\tilde{b}}$ traverses a bounded number of Dirichlet domains. The complexity of the algorithm is bounded by $O\left(N^2\right)$, where N is the number of points in the ε-net.

3 Preliminary Result: Walking in a Triangulation in $\mathbb{H}^2$

As mentioned above, a crucial step of the algorithm relies on finding a triangle containing a given point in a triangulation of $\mathbb{H}^2$ (there is only one such triangle, except for collinear points). There are two classical algorithms in the Euclidean plane: the straight walk and the

visibility walk [22], which can be adapted to the hyperbolic plane. Both methods find the triangle containing a query point $\widetilde{q}$ in a triangulation starting from a vertex $\widetilde{p}$ of a triangle $\widetilde{\Delta}$. The *straight walk* visits all triangles along the geodesic segment $\widetilde{pq}$. The algorithm starts by rotating around $\widetilde{p}$ to find a triangle incident to $\widetilde{p}$ that has an edge intersecting the geodesic segment $\widetilde{pq}$. The *visibility walk* 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 $\widetilde{e}$.

In the Euclidean case, the visibility walk terminates in a Delaunay triangulation [24] but it can loop forever in a non-Delaunay triangulation [13]. The following theorem takes care of the hyperbolic case.

▶ **Theorem 1.** *The visibility walk terminates in a finite or periodic hyperbolic Delaunay triangulation.*

The proof follows the same logic as in the Euclidean case [21] but it requires a new definition of the power of a point with respect to a circle, adapted to the hyperbolic case, which requires to rewrite the details. Due to lack of space, the proof is given in the full version of this article [17, Sec 3].

4 Data Structure

We enrich the data structure of the CGAL package *2D triangulations on hyperbolic surfaces* [15] based on a combinatorial map. A *combinatorial map* is an edge-centered data structure based on *darts*, which are equivalent to half-edges in our setting [30, 12]. A dart can be seen as an oriented edge. 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, as illustrated in Figure 3. Following β_1 pointers, we obtain all the darts of a given face. Following $\beta_1 \circ \beta_2$ combinations of pointers, we obtain all the darts of a given vertex.

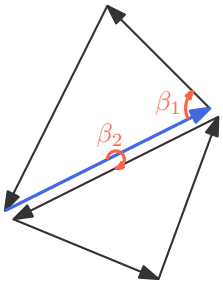

Figure 3 A dart (bold blue) in a 2D combinatorial map, and its pointers.

In the CGAL package *2D triangulations on hyperbolic surfaces* [15], the geometric information of a triangulation T is given as a cross-ratio for each edge, which is associated with darts. The *cross-ratio* of four pairwise distinct points z_1, z_2, z_3, z_4 in the Poincaré disk is defined as $[z_1, z_2, z_3, z_4] = \dfrac{(z_4 - z_2)(z_3 - z_1)}{(z_4 - z_1)(z_3 - z_2)} \in \mathbb{C}$. Let $\widetilde{e} = (\widetilde{p}, \widetilde{r})$ be a lift of an edge e and $\widetilde{q}, \widetilde{s}$ be the remaining vertices of the lifted triangles incident to $\widetilde{e}$, such that $\widetilde{p}, \widetilde{q}, \widetilde{r}, \widetilde{s}$ are oriented counter-clockwise. The cross-ratio of e in T is $R_T(e) = [\widetilde{p}, \widetilde{q}, \widetilde{r}, \widetilde{s}]$. It is invariant under orientation-preserving isometries, so, the cross-ratio of an edge e can be defined from any of its lifts. The imaginary part of $R_T(e)$ is positive if and only if $\widetilde{s}$ lies in the circumdisk of the triangle $(\widetilde{p}, \widetilde{q}, \widetilde{r})$, i.e., if and only if the edge e is Delaunay flippable.

In addition to cross-ratios, the data structure of [15] contains one anchor, which represents a lift of one triangle in the Poincaré disk. The *anchor* is composed of a dart representing the triangle in the combinatorial map, and of the coordinates of the three vertices $\widetilde{p}, \widetilde{q}, \widetilde{r}$ of a lift of the triangle. The dart corresponds to the edge (p, q). Knowing a lift of a triangle, its neighbors can be retrieved using the cross-ratios of its edges: if $(\widetilde{p}, \widetilde{q}, \widetilde{r})$ and $(\widetilde{p}, \widetilde{r}, \widetilde{s})$ are two triangles in $\mathbb{H}^2$ sharing the edge $(\widetilde{p}, \widetilde{r})$, the coordinates of $\widetilde{s}$ can be deduced from $\widetilde{p}, \widetilde{q}, \widetilde{r}$ and $R_T((p, r))$. The anchor can therefore be used to compute a part of $\widetilde{T}$ one step at a time, for example to draw a lift of each triangle of T.

Our algorithm (see Section 6) performs computations with lifts of triangles at each step, in particular to compute a lift of the circumcenter of a triangle and locate it. To have constant time access to a lift of any triangle, we actually store an anchor for *each* face of the triangulation. The three darts of each triangular face are associated with its anchor.

Note that the set of anchors of all faces of T is not necessarily connected.

Updating the Data Structure

In the ε-net algorithm, the data structure is modified by two operations: splitting a triangle into three new triangles, and flipping an edge.

When splitting a triangle Δ, we know the lift $\widetilde{\Delta} = (\widetilde{p}, \widetilde{q}, \widetilde{r})$ given by its anchor and the point $\widetilde{s}$ to be inserted in $\widetilde{\Delta}$. The darts of Δ are kept and three pairs of darts are created (Figure 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. We create three new anchors corresponding to the new triangles $(\widetilde{p}, \widetilde{q}, \widetilde{s})$, $(\widetilde{q}, \widetilde{r}, \widetilde{s})$ and $(\widetilde{r}, \widetilde{p}, \widetilde{s})$ and associate them to the darts of their respective triangles. The cross-ratios of the three new edges are computed from the coordinates of $\widetilde{p}, \widetilde{q}, \widetilde{r}$, and $\widetilde{s}$. The cross-ratios of the edges $(p, q), (q, r)$, and (r, p) must be updated. For the edge (p, q), for instance, we use its non-updated cross-ratio and the vertices $\widetilde{p}, \widetilde{q}$ and $\widetilde{r}$ to compute the coordinates of a lift of the third vertex of its neighboring incident triangle. This step is mandatory because the anchor associated with the adjacent triangle in the combinatorial map may store a non-adjacent lift in $\mathbb{H}^2$. The new value of the cross-ratio can then be computed.

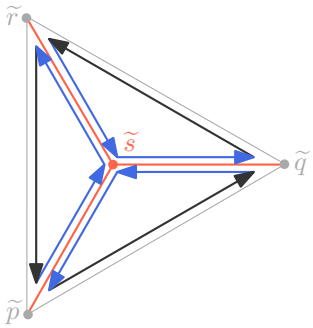

Figure 4 Splitting a triangular face in three in the combinatorial map, new darts are in blue.

To flip an edge, we use the CGAL package [15], which modifies the β_1 pointers and updates the cross-ratios [14]. We still need to update the anchors. To do so, we get the coordinates of the points stored in the anchor of a dart δ of the edge being flipped. We call them $\widetilde{p}, \widetilde{q}, \widetilde{r}$, in such a way that the edge $(\widetilde{r}, \widetilde{p})$ is represented by δ in the combinatorial map. Using the cross-ratio of that edge, we compute the coordinates of $\widetilde{s}$, the third vertex of the other lifted triangle sharing the edge $(\widetilde{r}, \widetilde{p})$ (Figure 5). We then create two new anchors corresponding to

the lifts of the triangles obtained after the flip, $(\widetilde{q}, \widetilde{s}, \widetilde{p})$ and $(\widetilde{q}, \widetilde{s}, \widetilde{r})$ and we associate them to the darts of their respective triangles.

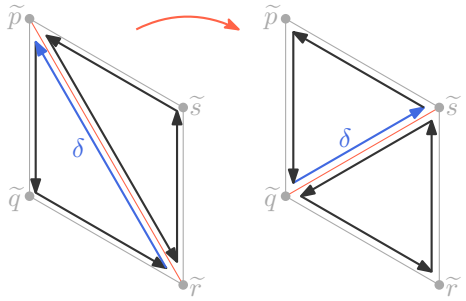

Figure 5 A flip. No darts are added or removed, only adjacencies are modified.

Note that, though these operations remove triangles and create new ones, darts are added but never removed in the data structure.

5 Generation of Input Surfaces

The input of our algorithm is a Delaunay triangulation of a surface with a single vertex. It is constructed from a fundamental domain with all its vertices in $\mathbb{H}^2$ projecting to the same point on the surface. We now explain how to generate such fundamental domains and the arithmetic issues related to the number type of the coordinates of its vertices.

The Teichmüller space $\mathcal{T}_g$, the space of all hyperbolic surfaces of genus g, can be parameterized for instance from a pants decomposition (Fenchel-Nielsen coordinates) [9, § 1.7], its group of isometries (Fricke coordinates) [26, § 2.5], or by a fundamental polygon in $\mathbb{H}^2$ [9, § 6]. Since we aim at computing a triangulation in $\mathbb{H}^2$, starting from a fundamental polygon is the right choice to avoid the use of hyperbolic trigonometric functions. Our algorithm relies on computations of cross-ratios and points in $\mathbb{H}^2$. There is no point in computing with `float` or `double` number types, as this is notoriously unstable.[2] Previous work showed that Delaunay triangulations can be computed on the Bolza surface, represented by a fundamental polygon with very specific algebraic numbers [27, 28]. Even when computing the Delaunay triangulation of points with rational coordinates, the group of the surface leads to algebraic numbers. In practice, algebraic numbers are handled with the CORE library [2] but this approach was shown to fail for generalized Bolza surfaces, already for genus 3 [23].

We now detail a way to generate and compute with generic surfaces of genus 2. Any genus 2 surface has a fundamental domain that is a symmetric octagon centered at the origin [3]. More precisely, three points are first chosen in the upper half of the Poincaré disk. Then a fourth point is 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 Teichmüller space $\mathcal{T}_2$ is thus parameterized by three complex numbers. Restricting to complex numbers with rational real and imaginary parts gives a dense subset of the Teichmüller space $\mathcal{T}_2$ [14]. This means that for any genus 2 surface, we can work on an arbitrary close surface given by a fundamental domain whose vertices have rational coordinates. This allows us to only use exact rational computations for constructing an ε-net (see Section 6.2) and algebraic extensions of degree 2 to check its correctness (see Section 6.4).

[2] `https://www.cgal.org/exact.html`

A Delaunay triangulation of this octagon is computed as in the CGAL package [15, 14]: the eight vertices of the domain actually project on a same point on the surface, which we denote as b. Triangulating this convex octagon in an arbitrary way gives a triangulation of the surface with one vertex. The next step is to compute the cross-ratios of all edges. Then the Delaunay triangulation is computed using flips. The data structure of the CGAL package [15] is a combinatorial map with a cross-ratio on each edge, and one anchor. To generate the input for our ε-net algorithm, we only have to add an anchor for each face of the triangulation. Since the initial triangulation has only one vertex and six faces, the remaining anchors are computed (and associated with the darts of their respective faces) using cross-ratios, by successively computing lifts of adjacent triangles, as detailed in Section 4.

For higher genus surfaces, a construction of fundamental polygons is described in [35, § 6.11], but unfortunately it is not clearly leading to tractable numbers. We are investigating how polygons with rational coordinates could be generated since in such a case our algorithm will have the same robustness properties as those we illustrate in genus 2.

6 The Implemented Algorithm

The data structure is implemented in `Anchored_hyperbolic_surface_triangulation_2`, a class inherited from the CGAL class `Triangulation_on_hyperbolic_surface_2` [15]. The algorithm to compute an ε-net is implemented in the `epsilon_net(epsilon)` method. It maintains a Delaunay triangulation and iteratively inserts circumcenters of large triangles. We first detail our processing of large triangles aiming at minimizing the number of computations of circumcenters (Section 6.1). We then detail the largeness test, the representation of coordinates of points in $\mathbb{H}^2$ by exact rational numbers and the possible issues of the approximation of circumcenters (Section 6.2). The point location is detailed in Section 6.3. Finally, we explain how to certify that the set of vertices of the output triangulation is an ε-net using exact computation in degree two algebraic extensions (Section 6.4).

6.1 Additional Data Structure

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 [17, App A]. The results show that any attempt to maintain the list using criteria involving circumradii, like 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. The only operations on the list are: pop the front dart, and push new darts 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 popped from the front. The list is initialized with one marked dart for each triangle of the input triangulation. Remark that for a reasonable choice of ε, all these triangles are large.

When a dart is popped out from $\mathcal{L}$, if it is marked, we unmark it and the circumradius of the triangle it represents is computed. If the triangle is large, its circumcenter splits the

triangle containing it as detailed in Section 4, and one dart is marked and pushed to the back of $\mathcal{L}$ for each of the three new triangles. Darts are added to $\mathcal{L}$ without checking the size of the circumradius of the corresponding triangle and without checking if it is a Delaunay triangle. The Delaunay property is restored using the flip algorithm described in [14] and implemented in [15]. After a flip, in each of the two new triangles, we maintain the property that only one dart is marked. Indeed, such a new triangle may have 0, 1 or 2 marked darts (Figure 6). If it has none, then we mark one and push it to the back of $\mathcal{L}$; if it has two, then we leave them in $\mathcal{L}$ and unmark one; otherwise there is nothing to do.

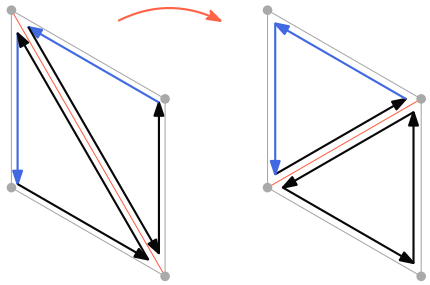

Figure 6 Case when triangles created by a flip have one or two marked darts.

When a non-marked dart is popped out from $\mathcal{L}$, then nothing is done: it means that the triangle it belongs to is represented by another marked dart in $\mathcal{L}$.

The algorithm proceeds with the new front dart until $\mathcal{L}$ is empty. Note that in practice, since darts representing new triangles are always pushed at the end of $\mathcal{L}$, large triangles tend to be close to the front and are processed before triangles with smaller circumradii.

Remark that no element of $\mathcal{L}$ is ever removed from the list, except at the front. As explained in Section 4, when a triangle disappears from the triangulation, its darts stay in the data structure, but their β_1 pointers 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 triangle whose circumradius is not greater than ε when it comes to the front of $\mathcal{L}$.

6.2 Circumcenters and Circumradii

When a marked dart is popped out from $\mathcal{L}$, the circumcenter of a lift of the triangle represented by the dart is first constructed in order to compute its circumradius. The lifted triangle is the one stored in the anchor associated to the dart. The coordinates of a circumcenter of three vertices with rational coordinates are algebraic numbers of degree two [6]. When it is inserted in the triangulation, a circumcenter becomes a vertex. Handling exact circumcenters would thus lead to cascading the algebraic degrees of coordinates, which would result in computations that are known to be impossible to handle in practice.

The coordinates of the circumcenter $\widetilde{c}_\Delta$ of a triangle $\widetilde{\Delta}$ are represented by the type `CGAL::Sqrt_extension`, which handles algebraic numbers of degree two. We do not insert the point $\widetilde{c}_\Delta$ but a rational approximation denoted $\widetilde{c}$. We round each coordinate of $\widetilde{c}_\Delta$ to a double precision number using the `to_double()` method of CGAL and convert it to an exact rational type. More specifically, coordinates of all vertices of our triangulation are represented by the `CGAL::Exact_rational` number type, which is a wrapper for the arbitrary-precision rational type `mpq_t` provided by GMP [20]. All the computations on our data structure are performed using this `CGAL::Exact_rational` number type for points and cross-ratios. Note that the bitsize of the rationals encoding an inserted point is bounded whereas the bitsize of

the rationals encoding a cross-ratio grows when it is updated during a flip.

Let us detail the computation of the largeness test for triangles that we use to avoid as much as possible the use of hyperbolic trigonometric functions. In the Poincaré disk, the distance between two points $\widetilde{u}$ and $\widetilde{v}$ is $d_{\mathbb{H}^2}(\widetilde{u},\widetilde{v}) = \operatorname{arcosh}(1+\lambda(\widetilde{u},\widetilde{v}))$, where $\lambda(\widetilde{u},\widetilde{v}) = \dfrac{2||\widetilde{u}-\widetilde{v}||^2}{(1-||\widetilde{u}||^2)(1-||\widetilde{v}||^2)}$ and $||\cdot||$ is the Euclidean distance. We compute the minimum λ between the approximate circumcenter $\widetilde{c}$ and the three vertices of the anchor. If it is greater than a certified upper bound of $\cosh(\varepsilon)-1$, then we consider the triangle large and insert $\widetilde{c}$ in the triangulation. The certified upper bound on $\cosh(\varepsilon)$ is computed with the Boost interval library [1]. Due to approximations for the computation of $\widetilde{c}$, this process may miss a large triangle and thus does not enforce the ε-covering property. Similarly, even if the point $\widetilde{c}$ that is inserted into the triangulation is at distance at most ε from the vertices of its triangle, it may be at distance smaller than ε from another vertex of the triangulation. We will check these properties in the final step of the algorithm (Section 6.4).

6.3 Point Location

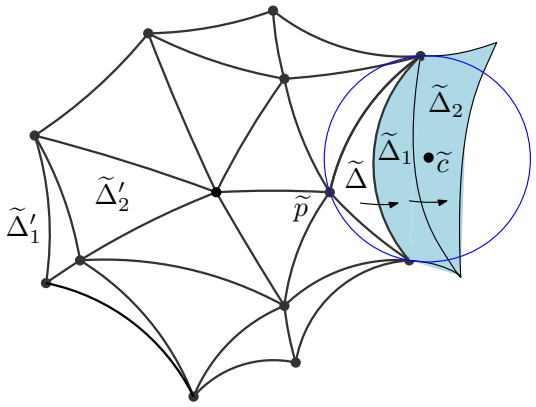

Figure 7 A fundamental domain of the surface is given by the anchors of the triangles (thick black lines). The blue triangles are the lifts constructed by the visibility walk to reach the approximate circumcenter $\widetilde{c}$ of $\widetilde{\Delta}$. Triangles $\widetilde{\Delta}'_1$ and $\widetilde{\Delta}'_2$ are the lifts of Δ_1 and Δ_2 in the domain.

Our implementation maintains a Delaunay triangulation of S using the data structure presented in Section 4. Each point to be inserted is located in this triangulation by locating a lift in the lifted triangulation. We do not store a Dirichlet domain as in the original algorithm (Section 2.3), but instead directly walk in the current triangulation itself.

We tested two walk algorithms: the straight walk and the visibility walk (Figure 7) mentioned in Section 3. In our case, the query point $\widetilde{c}$ is the approximate circumcenter of the lifted triangle $\widetilde{\Delta}$ given by the anchor of the triangle Δ being processed by the algorithm. The walk starts from a vertex $\widetilde{p}$ of the triangle $\widetilde{\Delta}$. Lifts of triangles must be constructed along the walk to find the lifted triangle containing $\widetilde{c}$.

Both walks are based on orientation tests in $\mathbb{H}^2$ with vertices of lifted triangles. The combinatorial map encoding the triangulation provides a direct access to the neighbor Δ' of the triangle Δ adjacent through one of its edges. However, the lift $\widetilde{\Delta}'$ of this triangle given by its anchor may not be adjacent to the lift $\widetilde{\Delta}$. The lift of Δ' that is adjacent to $\widetilde{\Delta}$ is computed from the vertices of $\widetilde{\Delta}$ and the cross-ratio of the common edge as explained in Section 4.

Running the `epsilon_net` method with both walks shows that they perform equivalently (see [17, App B]): the running time of the method remains the same, and they compute a similar number of lifts. The visibility walk does not have to handle the degenerate cases of the straight walk, which may go through a vertex or along an edge; we therefore choose the visibility walk. The bound on the length of the straight walk for the original algorithm, using Dirichlet domains, does not apply to the visibility walk. However we observe in Section 7.2 that the walk has constant length in practice.

6.4 ε-Covering and ε-Packing Checks

As explained in Section 6.2, 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. For every triangle Δ, we compute the exact circumcenter $\widetilde{c}_\Delta$ of its anchor, which has coordinates in a degree two algebraic extension, using the number type `CGAL::Sqrt_extension`. We then compute $\lambda(\widetilde{c}_\Delta, \widetilde{v})$ with the same type (see Section 6.2) for a vertex $\widetilde{v}$ of the triangle, and check that it is less than a certified lower bound on $\cosh(\varepsilon) - 1$. To check the ε-packing property, it is sufficient to check that all Delaunay edges are longer than ε. Since all vertices have rational coordinates, the value $\lambda(\widetilde{v}_i, \widetilde{v}_j)$ for two vertices of an edge is rational and it is checked to be larger than the upper bound on $\cosh(\varepsilon) - 1$ computed with the Boost interval library [1]. When these tests succeed, the vertices of the output triangulation are then certified to be an ε-net of the surface. Otherwise, the failure of these tests means that the double precision used for the approximation of circumcenters was not enough.

Our experiments presented in Section 7 show that for surfaces with a long systole, which is the most common case (see Section 7.1), our algorithm is successful. Using a double precision to set the rational coordinates of approximate circumcenters and to compute the bounds on $\cosh(\varepsilon) - 1$ actually constructs valid ε-nets. On the other hand, we also observe in the full version of this article [17, Sec 7.3] that higher precision would be needed to handle a surface with a very small systole.

Our future work will focus on certifying the largeness test for triangles and check the ε-packing property after each insertion. If the ε-packing check fails, this means that the inserted point was not a good enough approximation of the circumcenter and iteratively refining it (which is possible via the CGAL Algebraic Kernel [5]) would eventually recover the ε-packing property. Note that iterative refinement of the rational bounds on $\cosh(\varepsilon) - 1$ must also be computed (which is possible via the Boost library).

7 Experiments

All the experiments of Section 7.2 are performed on 180 random surfaces of genus 2 generated as explained in Section 5 by uniformly choosing, for the Euclidean distance, three points in the upper half of the Poincaré disk. As explained in Section 7.1, 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. See the full version [17, Sec 7.2] for details. Section 7.3 displays and analyzes visualizations of the output.

The source code and complete data of the experiments are available on GitHub [3].

[3] `https://anonymous.4open.science/r/ESA_2025_submission_implementation_epsilon_net-A7CD/`

7.1 Distribution of Input Surfaces

To experiment with an algorithm in practice, it is important to sample the space of possible input with some distribution. We discuss this issue below and explain why our random generator of surfaces is likely to produce surfaces with long systole.

Well-Distributed Surfaces?

The situation was nicely described by Mirzakhani [31] (though there are more recent results [32]). We just give a flavor here. The moduli space $\mathcal{M}_g$ is the set of all hyperbolic surfaces of genus g. It can be equipped by 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 any of the two metrics. However, today's mathematical literature does not answer this question. The first problem is that there is no known parameterization of $\mathcal{M}_g$, so, we can only work with a parameterization of its universal cover, the Teichmüller space $\mathcal{T}_g$, i.e., the set of so-called marked hyperbolic surfaces. Indeed, in $\mathcal{T}_g$, as opposed to in $\mathcal{M}_g$, applying a non-identity homeomorphism to a hyperbolic surface gives a different element in $\mathcal{T}_g$: the moduli space $\mathcal{M}_g$ is the quotient of $\mathcal{T}_g$ by the mapping class group Mod_g, which is the group of homeomorphisms of the topological surface of genus g. Unfortunately, the known parameterizations of $\mathcal{T}_g$ are not invariant under the action of Mod_g, which is the main reason why sampling $\mathcal{M}_g$ is so intricate. It is not clear how to describe a fundamental domain of $\mathcal{M}_g$ in $\mathcal{T}_g$ in any parameterization of $\mathcal{T}_g$. As of today, the best that can be done is to sample a parameterization of $\mathcal{T}_g$, being aware that this does not lead to a good distribution on $\mathcal{M}_g$.

For completeness, we mention that there are other ways of constructing random surfaces in theory, by randomly gluing hyperbolic ideal triangles together [8, 33]. However, these methods rely on a compactification of the obtained cusped surfaces, which, roughly speaking, boils down to a conformal uniformization of the metric. This process is obviously very far from yielding a practical construction. Additionally, those approaches lead to surfaces with huge genus (typically, several thousands), which cannot be manipulated in practice.

Systole of random surfaces

Remark that $\mathcal{M}_g$ is not a compact set, as a surface can have an arbitrarily small systole: Indeed, in any compact subset of $\mathcal{M}_g$, the systole function $sys(\cdot)$ is bounded. Mirzakhani showed that small systoles are somewhat 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 point of view) is of the order of σ_0^2. This means that the typical case is a surface with a reasonably long systole. It is the case for the random surfaces that we generated, as explained in Section 5. For completeness, we also specifically design a surface with a small systole and experimentally observe the impact in the full version of this article [17, Sec 7.3].

7.2 Main Results

For each of the 180 random surfaces, an input for the ε-net algorithm is computed as a single vertex Delaunay triangulation of the surface, as explained in Section 5. The `epsilon_net` method is run with 50 values of ε on every input, decreasing from 0.5 to 0.01 with a step of 0.01. Table 1 presents an overview of the results.

We first observe that the number of vertices in the computed ε-nets is close to the theoretical upper bound for an ε-thick surface, which is $16(g-1)/\varepsilon^2 = 16/\varepsilon^2$ ($g = 2$ for all surfaces of the experiments). In proportion, the number of vertices is 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.

Table 1 Average number of vertices of the ε-nets.

ε	**0.50**	**0.40**	**0.30**	**0.20**	**0.10**	**0.05**	**0.01**
Avg. # of vertices	34	54	96	216	865	3,454	86,314
$16/\varepsilon^2$	64	100	178	400	1,600	6,400	160,000

Even though, as in the Euclidean case [22], we have no complexity analysis of the visibility walk for the point location, we observe that the walk traverses a (small) constant number of triangles. The walk locates the approximate circumcenter of a triangle in $\mathbb{H}^2$, starting from this triangle. 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 8. On average, 68% of the approximate circumcenters lie in their triangle. Over all surfaces and all tested values of ε, the farthest located points were 4 triangles away from the starting triangle of the walk.

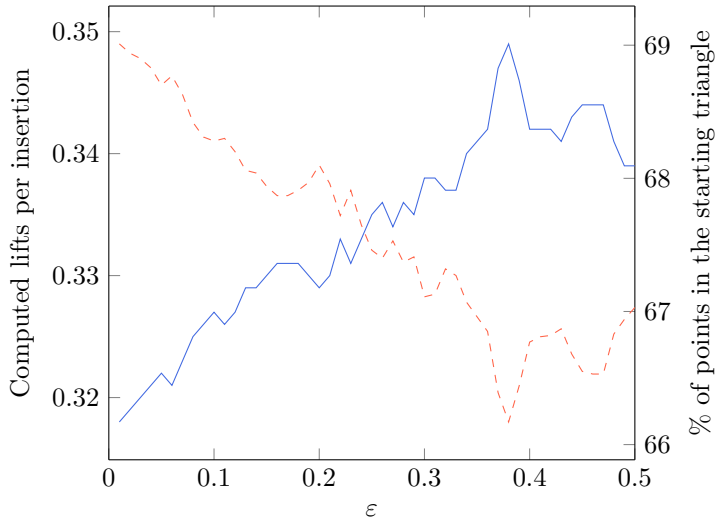

Figure 8 Average number of computed lifted triangles in the walk at each locate query (left, solid), and average percentage of points lying in the initial lifted triangle of the walk (right, dashed).

We counted the number of flips done 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 9. This shows that the number of flips at each insertion is a small constant in practice. The total number of flips is thus, in practice, linear in the number of points, which is in contrast with the theoretical quadratic bound (see Section 2.3).

In the full version of this article [17, App C], we show an order of magnitude for the running times to give an idea of the practical complexity of our implementation. It appears to be slightly faster than quadratic in $1/\varepsilon$.

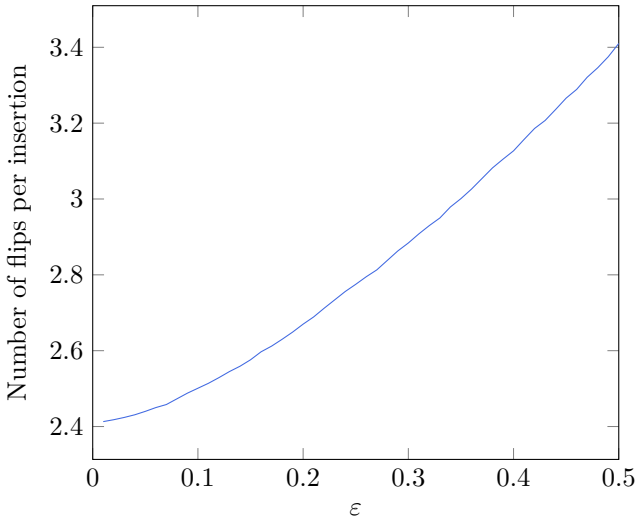

Figure 9 Average number of flips done to recover the Delaunay property after each insertion.

7.3 Visualization of the Output

To visualize an ε-net, we draw a fundamental domain of the surface in the Poincaré disk using its Delaunay triangulation. To do so, it suffices to draw a lift of each triangle in a connected way. We cannot just use the anchors since they would generally not lead to a connected domain. So, we access the anchor of a chosen triangle and build lifts of the other triangles starting from this initial anchor as explained in Section 4. As objects appear smaller near the boundary of the Poincaré disk, we center the drawing at the origin to be able to observe the details. To achieve this, we translate the vertices of the initial anchor such that one of them is 0 before computing the rest of the drawing. If we want similar drawings when running the ε-net algorithm with different values of ε on a given surface, we always use an anchor having a fixed lift $\widetilde{b}$ of b, the unique vertex of the input triangulation (Section 5), as the initial anchor; we then apply the translation that translates $\widetilde{b}$ to 0.

In Figures 10 and 11, the drawings are computed by the lift method of the CGAL package *2D triangulations on hyperbolic surfaces* [15], which uses a weight on edges to order the lifts of triangles of the triangulation T of S. The weight of an edge $(\widetilde{x}, \widetilde{y})$ is defined as $|\widetilde{x}|^2 + |\widetilde{y}|^2$ ($|\cdot|$ being the complex modulus). The drawing is then iteratively computed: given T' the set of triangles that have 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 10 shows the obtained Delaunay triangulation of the surface with a small systole studied in the full version of this article [17, Sec 7.3]. Since the systole is very small, there is a very long collar around the shortest non-contractible closed curve.

The drawings of triangulations shown in Figure 11 naturally look like Dirichlet domains since the weights on the edges ensure that the lifts of triangles entirely contained in $\mathcal{D}_{\widetilde{b}}$, the Dirichlet domain of $\widetilde{b}$, are drawn. This becomes clear when $\widetilde{b}$ is translated to the origin (Figure 12). Since the input of the ε-net algorithm is a Delaunay triangulation with the single vertex b, we obtain $\mathcal{D}_{\widetilde{b}}$ 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* (see Figure 13). We observe that,

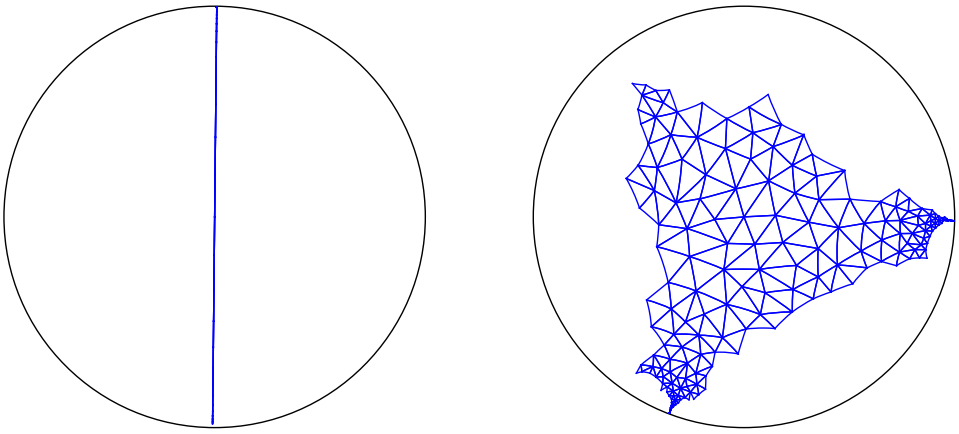

Figure 10 Delaunay triangulation of the computed 0.25-net of the surface with a small systole of [17, Sec 7.3]. Left: Drawing centered at a vertex in the long collar. Right: Drawing centered outside the collar, which appears in the form of "horns" pointing towards the boundary of the Poincaré disk.

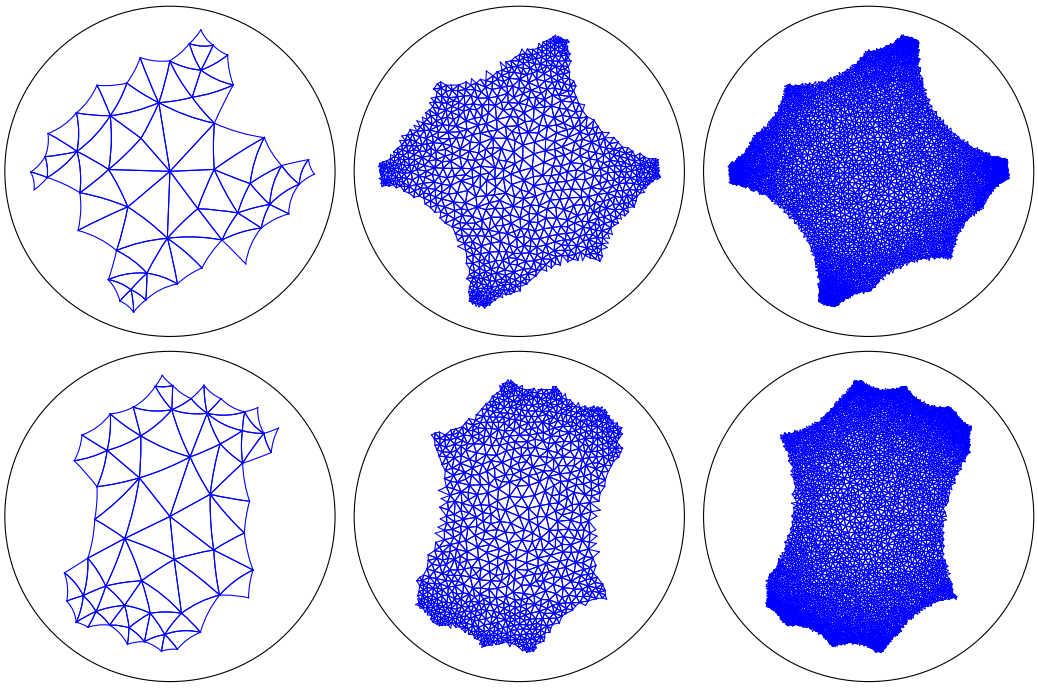

Figure 11 Delaunay triangulations of the computed ε-nets for ε=0.5 (left), ε=0.1 (middle), ε=0.05 (right) on the surfaces of seed 123 (top) and 321 (bottom).

when ε decreases, the combinatorial Dirichlet domain fits the Dirichlet domain $\mathcal{D}_{\tilde{b}}$ better. Formalizing this convergence is an interesting open question.

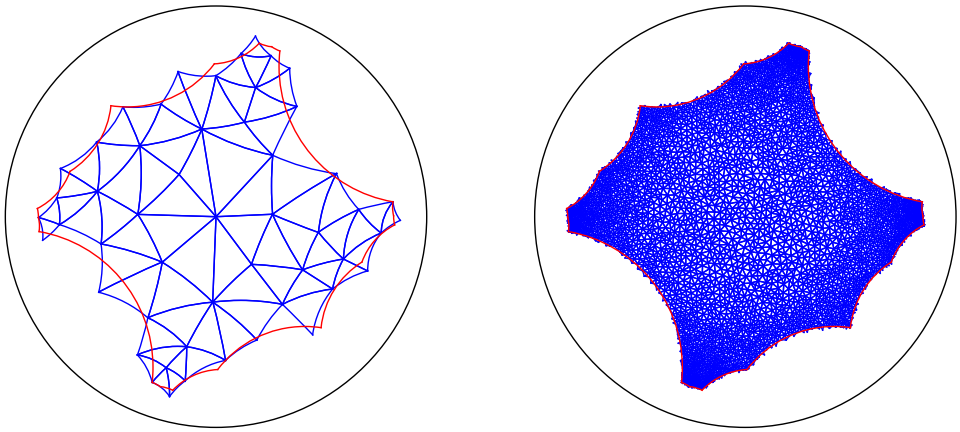

Figure 12 Delaunay triangulations of a 0.5-net (left) and a 0.05-net (right) of the surface of seed 123, and the corresponding Dirichlet domain. Lift computed with the CGAL package [15].

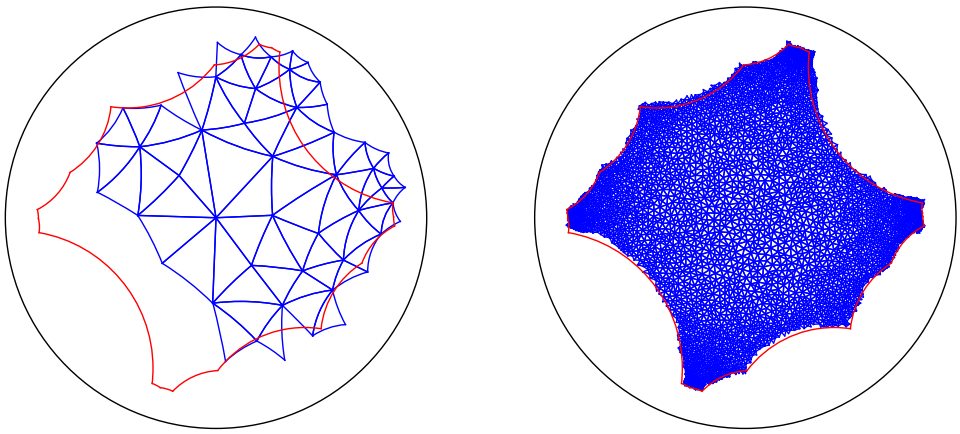

Figure 13 Delaunay triangulations of a 0.5-net (left) and a 0.05-net (right) of the surface of seed 123, and the corresponding Dirichlet domain. Lift computed with a BFS algorithm.

References

1 *BOOST C++ Libraries*. URL: http://www.boost.org.

2 *The CORE library project*. URL: https://cs.nyu.edu/~exact/core_pages/.

3 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.

4 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.

5 Eric Berberich, Michael Hemmer, Michael Kerber, Sylvain Lazard, Luis Peñaranda, and Monique Teillaud. Algebraic kernel. 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#PkgAlgebraicKernelD.

6 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.

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

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

9 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.

10 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.

11 Éric Colin de Verdière. Algorithms for embedded graphs, 2021. Lecture notes. URL: https://monge.univ-mlv.fr/~colinde/cours/all-algo-embedded-graphs.pdf.

12 Guillaume Damiand. Combinatorial maps. 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#PkgCombinatorialMaps.

13 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.

14 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.

15 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.

16 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.

17 Vincent Despré, Camille Lanuel, Marc Pouget, and Monique Teillaud. ε-net algorithm implementation on hyperbolic surfaces. December 2024. URL: https://hal.science/hal-04820353.

18 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.

19 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.

20 GMP development team. GNU MP: The GNU Multiple Precision Arithmetic Library. URL: https://gmplib.org.

21 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.

22 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.

23 Matthijs Ebbens, Iordan Iordanov, Monique Teillaud, and Gert Vegter. Delaunay triangulations of generalized Bolza surfaces. *Journal of Computational Geometry*, 13(1):125–177, 2022. URL: https://hal.inria.fr/hal-03664678, doi:10.20382/jocg.v13i1a5.

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

25 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.

26 Yoichi Imayoshi and Masahiko Taniguchi. *An Introduction to Teichmüller Spaces*. Springer Japan, 1992. URL: http://dx.doi.org/10.1007/978-4-431-68174-8, doi:10.1007/978-4-431-68174-8.

27 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.

28 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.

29 Charles L. Lawson. Software for C^1 surface interpolation. In *Symposium on Mathematical Software*, 1977. NASA Technical Report JPL-PUBL-77-30. URL: https://ntrs.nasa.gov/citations/19770025881.

30 Pascal Lienhardt. Subdivisions of n-dimensional spaces and n-dimensional generalized maps. In *Proceedings 5th Annual Symposium on Computational Geometry*, pages 228–236, 1989. doi:10.1145/73833.73859.

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

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

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

34 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.

35 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.