ϵ-Net Algorithm Implementation on Hyperbolic Surfaces

Vincent Despré, Camille Lanuel, Marc Pouget, Monique Teillaud

ε-Net Algorithm Implementation on Hyperbolic Surfaces

Vincent Despré[1], Camille Lanuel[1], Marc Pouget[1], and Monique Teillaud[1]

[1]Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

December 2024

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 [DLT24]. 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, using a combinatorial map, to represent a triangulation of a surface. Also for the critical step of locating points on the surface, we use the visibility walk and prove its termination in the hyperbolic setting.

1 Introduction

Hyperbolic surfaces, i.e., surfaces with a constant negative curvature, are well studied in mathematics since hyperbolic geometry is the natural geometry on surfaces of genus larger than one [FK92]. However, there are long standing open problems that could be experimentally investigated like finding the hyperbolic surface of genus 3 with the longest systole (non-contractible shortest curve) 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 [IT17] introduced a package in CGAL [IT19] to construct Delaunay triangulations with Bowyer's incremental algorithm, only for the Bolza surface, which is the most symmetric surface of genus two. The authors in [DDKT22] proposed an implementation of edge flips to transform any triangulation of a closed hyperbolic surface into the Delaunay triangulation. Their software [DDT] also generates surfaces of genus 2. The algorithm does not support insertions of new points.

The aforementioned open problems need computations of distances on a surface and an ε-net (see Section 2.2) is a natural tool to approximate such distances. Our contribution is the implementation of an ε-net algorithm proposed in [DLT24] based on Shewchuk's Delaunay refinement [She02]. 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 [DLT24] and the other considers the surface as a combinatorial map [DDKT22]. We use an enriched version of the data structure implemented in [DDKT22, DDT].

Our implementation is independent from the genus of the surface. However, the only tractable surfaces that we can 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 [EITV22], even for generalized Bolza surfaces, the representation of fundamental domains with algebraic numbers in genus higher than two is an obstacle.

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 are in an algebraic extension of degree two with respect to the coordinates of the vertices of the triangle, our algorithm only constructs approximate points with rational coordinates. Consequently, we have to check the correctness of the output, and this is the only place where we use algebraic numbers of degree 2. This approach is proven successful in our experiments for "typical" surfaces, i.e., surfaces with a large systole. We observe in Section 7.2 that higher precision on the rational approximation of the circumcenter would be needed to handle a surface with a very small systole.

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 $\mathbb{H}^2$ and determine the lift of a triangle containing the point. Similar to the Euclidean plane, there are multiple strategies for traversing the triangulation and locate a point [DPT02]. In our implementation, we tested both the straight and the visibility walks, which demonstrate comparable efficiency for our purposes. To ensure the correctness of our algorithm, we also prove that the visibility walk terminates successfully in the context of finite or periodic Delaunay triangulations of $\mathbb{H}^2$ (Theorem 1).

The paper is organized as follows: The termination of the visibility walk in $\mathbb{H}^2$ is proved in Section 3. We detail our data structure in Section 4 and our implemented algorithm in Section 5. We discuss in Section 6 the issues of generating relevant and tractable hyperbolic surfaces. Our experiments are reported in Section 7.

2 Background and Notation

2.1 Hyperbolic Surfaces

We refer the reader to textbooks [Bea83, Bus10] for more details on 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 $\mathbb{H}^2$. The action of Γ on $\mathbb{H}^2$ induces a projection $\rho : \mathbb{H}^2 \mapsto \mathbb{H}^2/\Gamma = S$. If $x \in S$, 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. The term *copy* refers to an image of an object by an isometry of Γ.

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 $\mathbb{C}$. 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.

If T is a triangulation of S, we note $\widetilde{T}$ the infinite 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.

We denote as g the genus of S and σ its *systole*, i.e., the length of the shortest non-contractible closed geodesic.

2.2 ε-Nets

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*.

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 $16(g-1)/\varepsilon^2$ if $\varepsilon < \sigma$ [DLT24]. If so, we say that S is *ε-thick*.

2.3 Original Algorithm

Let us summarize the original algorithm [DLT24], on which our implementation is based. The algorithm is inspired by Shewchuck's Delaunay refinement [She02]. 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 1): the triangle in which the circumcenter lies is first split into three new triangles, then the Delaunay property is retrieved using edge flips [DST20]. 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.

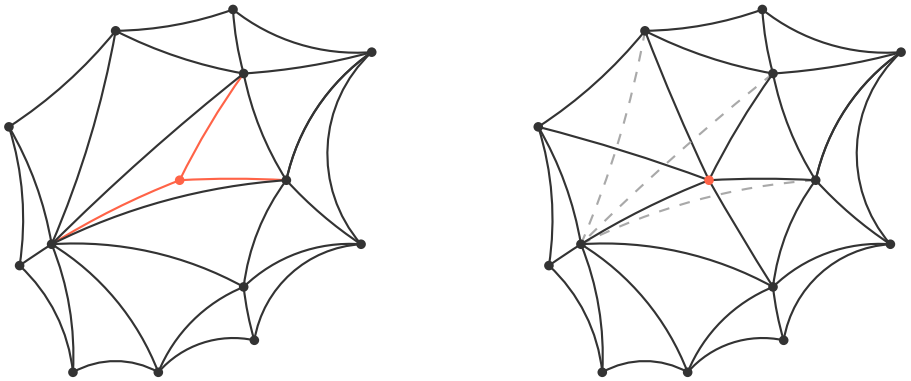

Figure 1: Insertion of a point in a Delaunay triangulation using a flip algorithm.

The algorithm also stores the Dirichlet domain $\mathcal{D}_{\widetilde{b}}$ of a lift $\widetilde{b}$ of b together with its side pairings, which can be computed from any fundamental domain for S [DKPT23]. The data structure stores the lift in $\mathcal{D}_{\widetilde{b}}$ of each vertex of the triangulation (or one of its lifts if it lies on the boundary of $\mathcal{D}_{\widetilde{b}}$), and for each triangle, the information needed to retrieve its (at most three) lifts having at least one vertex in $\mathcal{D}_{\widetilde{b}}$.

The most crucial step of the algorithm is the location of the circumcenter c of a triangle Δ in a triangulation of S. To this aim, a lift of c is located in the lifted triangulation in $\mathbb{H}^2$, as follows. First, the circumcenter $\widetilde{c}$ of a lift $\widetilde{\Delta}$ of Δ with at least one vertex in $\mathcal{D}_{\widetilde{b}}$ is computed. The point $\widetilde{c}$ is a lift of c that does not necessarily lie in $\mathcal{D}_{\widetilde{b}}$. To locate $\widetilde{c}$, the algorithm first finds the copy of $\mathcal{D}_{\widetilde{b}}$ containing it by walking in the tiling $\Gamma\mathcal{D}_{\widetilde{b}}$, i.e. the element $\gamma_c \in \Gamma$ such that $\widetilde{c} \in \gamma_c \mathcal{D}_{\widetilde{b}}$. The lift $\widetilde{c_b} = \gamma_c^{-1}\widetilde{c}$ of c lying in $\mathcal{D}_{\widetilde{b}}$ can then be computed. Second, the triangle in $\mathcal{D}_{\widetilde{b}}$ in which $\widetilde{c_b}$ lies is identified by checking, for each triangle, its (at most three) lifts having at least one vertex in $\mathcal{D}_{\widetilde{b}}$. It can be shown that the walk in the tiling $\Gamma\mathcal{D}_{\widetilde{b}}$ 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). In this section, we drop the $\tilde{\cdot}$ for objects in $\mathbb{H}^2$ in order to keep notation light. There are two classical algorithms in the Euclidean plane: the straight walk and the visibility walk, which can be adapted to the hyperbolic plane. Both methods find the triangle containing a query point q in a triangulation starting from a vertex p of a triangle Δ. The *straight walk* visits all triangles along the geodesic segment pq. The algorithm starts by rotating around p to find a triangle incident to p that has an edge intersecting the geodesic segment pq. The *visibility walk* consists, for each visited triangle not containing q, of moving to a neighbor through an edge e if q and the third vertex of the visited triangle are on different sides of the geodesic line supporting e. In the Euclidean case, the visibility walk terminates in a Delaunay triangulation [Ede90] but it can loop forever in a non-Delaunay triangulation [DFFNP91].

The rest of this section is devoted to proving Theorem 1.

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

A proof of the Euclidean case [DH16] 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 $||qz||^2 - r^2$. In $\mathbb{H}^2$, an equivalent of Pythagoras' theorem is [Bus10, Theorem 2.2.2]: in a right-angled triangle whose hypotenuse has length c and its two other sides have length a and b, we have $\cosh(c) = \cosh(a)\cosh(b)$. For lighter notation, we note the length of a geodesic segment xy as xy instead of $d_{\mathbb{H}^2}(x, y)$. We define the *power* of a point q with respect to a circle $\mathcal{C}(z, r)$ as $\cosh(zq)/\cosh(r)$. It is larger than 1 when q lies outside the circle, smaller than 1 when it lies inside, and equal to 1 when it lies on the circle.

We define the *circle power* of a triangle Δ in $\mathbb{H}^2$ with respect to q as

$$P(\Delta, q) := \frac{\cosh(z_\Delta q)}{\cosh(r_\Delta)},$$

where r_Δ denotes the radius of the circumcircle of Δ and z_Δ its center.

Lemma 2. *During the visibility walk toward the point q in a Delaunay triangulation in $\mathbb{H}^2$, if a triangle Δ is encountered before its neighbor Δ', then $P(\Delta, q) \geqslant P(\Delta', q)$.*

Proof. Denote z and z' the respective centers of the circumcircles of Δ and Δ', and r and r' their respective radii. Call u and v the common vertices of Δ and Δ'. The geodesic line zz' is the bisector of the geodesic segment uv, hence it intersects it perpendicularly at its midpoint m. As illustrated in Figure 2, Hyperbolic Pythagoras' theorem yields

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

When walking toward q, the point q is on the same side of the geodesic uv as Δ' and it is not on uv. The geodesic zz' can be oriented such that it crosses Δ before Δ'. The Delaunay property implies that z appears before z' for this order. Indeed, in the pencil of circles based on the edge uv, if z' appears before z, the part of the circle of center z' to the right of the edge contains the third point of the triangle Δ'. Since this part is included is the circle of center z, Δ' cannot be a Delaunay triangle. To prove that $P(\Delta, q) \geqslant P(\Delta', q)$, we distinguish three cases with respect to the position of the point m on the 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.

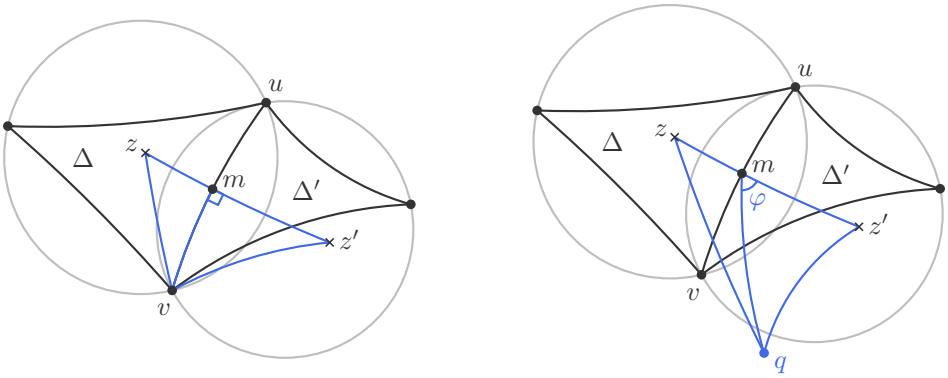

Figure 2: Illustration of Equation 1 (left) and of the first case of the proof (right).

First case: m lies between z and z' This case is illustrated in Figure 2 (right). Define φ as the unsigned angle $\angle(q, m, z')$. In the triangles (z, m, q) and (z', m, q), hyperbolic trigonometry [Bus10, Theorem 2.2.1] yields

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

The angle φ is positive and smaller than $\pi/2$ because q and Δ' lie on the same side of the line uv. Then $\cos(\varphi) > 0$ and $\cos(\pi - \varphi) < 0$ and it follows that $\dfrac{\cosh(zq)}{\cosh(zm)} > \cosh(mq) > \dfrac{\cosh(z'q)}{\cosh(z'm)}$.

We obtain $P(\Delta, q) > P(\Delta', q)$ using Equation 1.

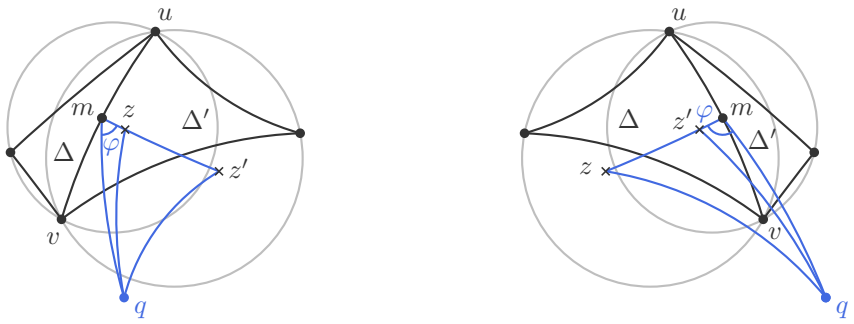

Figure 3: Illustration of the second (left) and third (right) cases of the proof.

Second case: z lies between m and z' This case is illustrated in Figure 3 (left). The same formula now yields

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

It follows that

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

As in the previous case, the angle φ is positive and smaller than $\pi/2$. Moreover, $zm < z'm$ and tanh is increasing. We divide both equations by $\cosh(vm)$ and obtain $P(\Delta, q) > P(\Delta', q)$.

Third case: z' lies between m and z This case is illustrated in Figure 3 (right). In this case, the same formula gives the same equations as above. 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 Delaunay triangulation in $\mathbb{H}^2$ given by the lifts of a triangulation of a surface. Indeed, there is a finite number of circumradii that are thus upper bounded by a value R, and for any triangle Δ' with circumcircle $\mathcal{C}(z', r')$ of the triangulation $P(\Delta, q) \geqslant P(\Delta', q) = \dfrac{\cosh(z'q)}{\cosh(r')} \geqslant \dfrac{\cosh(z'q)}{\cosh(R)}$. This implies that z' is in a bounded region of $\mathbb{H}^2$ so that the walk stays in a finite triangulation.

4 Data Structure

We reuse the data structure provided by Loïc Dubois' GitHub repository [DDKT22, DDT].

A *combinatorial map* is an edge-centered data structure based on *darts*, which are equivalent to half-edges in our setting. 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 4. 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.

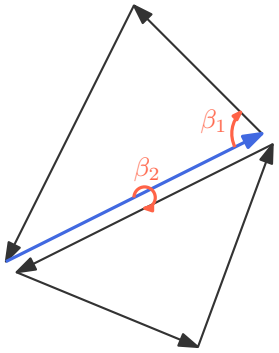

Figure 4: A dart (bold) in a 2D combinatorial map, and its pointers.

In Dubois' work, the geometric information of T is given as a cross-ratio for each edge, which is stored in darts. The *cross-ratio* of four pairwise distinct points z_1, z_2, z_3, z_4 in the Poincaré disk is defined as $[z_1, z_2, z_3, z_4] = \dfrac{(z_4 - z_2)(z_3 - z_1)}{(z_4 - z_1)(z_3 - z_2)} \in \mathbb{C}$. It is invariant under orientation-preserving isometries, so, the cross-ratio of an edge e can be defined from any of its lifts. 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 $R_T(e) = [\widetilde{u_1}, \widetilde{u_2}, \widetilde{u_3}, \widetilde{u_4}]$. The imaginary part of $R_T(e)$ is positive if and only if $\widetilde{u_4}$ lies in the circumdisk of the triangle $(\widetilde{u_1}, \widetilde{u_2}, \widetilde{u_3})$, i.e., if and only if the edge e is Delaunay flippable.

In addition to cross-ratios, Dubois' data structure contains one anchor, which represents a lift of a 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 $\widetilde{v_0}, \widetilde{v_1}, \widetilde{v_2}$ 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 $(\widetilde{a}, \widetilde{b}, \widetilde{c})$ and $(\widetilde{a}, \widetilde{c}, \widetilde{d})$ are two triangles in $\mathbb{H}^2$ sharing the edge $(\widetilde{a}, \widetilde{c})$, the coordinates of $\widetilde{d}$

can be deduced from $\widetilde{a}, \widetilde{b}, \widetilde{c}$ and $R_T(a, c)$. 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 5) 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 given by its anchor $\widetilde{\Delta} = (\widetilde{a}, \widetilde{b}, \widetilde{c})$ and the point $\widetilde{d}$ in $\widetilde{\Delta}$. The darts of Δ are kept and three pairs of darts are created (Figure 5). 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 $(\widetilde{a}, \widetilde{b}, \widetilde{d})$, $(\widetilde{b}, \widetilde{c}, \widetilde{d})$ and $(\widetilde{c}, \widetilde{a}, \widetilde{d})$ and associate them to the darts of their respective triangles. The cross-ratios of the three new edges are computed from the coordinates of $\widetilde{a}, \widetilde{b}, \widetilde{c}$, and $\widetilde{d}$. The cross-ratios of the edges $(a, b), (b, c)$, and (c, a) must be updated. For the edge (a, b), for instance, we use its cross-ratio and the vertices of the anchor $(\widetilde{a}, \widetilde{b}, \widetilde{d})$ of its incident triangle 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 $\mathbb{H}^2$.

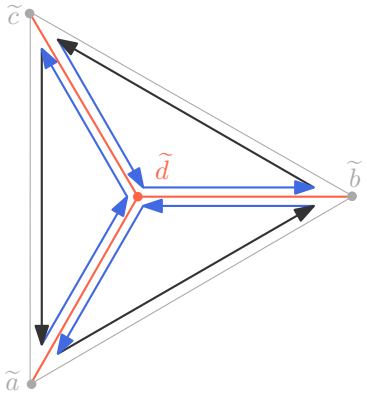

Figure 5: Splitting a triangular face in three in the combinatorial map, new darts are in blue.

To flip an edge, we use Dubois' code, which modifies the β_1 pointers and updates the cross-ratios [DDKT22]. 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 $\widetilde{a}, \widetilde{b}, \widetilde{c}$, in such a way that the edge $(\widetilde{c}, \widetilde{a})$ is represented by δ in the combinatorial map. Using the cross-ratio of that edge, we compute the coordinates of $\widetilde{d}$, the third vertex of the other lifted triangle sharing the edge $(\widetilde{c}, \widetilde{a})$ (see Figure 6). We then create two new anchors corresponding to the lifts of the triangles obtained after the flip, $(\widetilde{b}, \widetilde{d}, \widetilde{a})$ and $(\widetilde{b}, \widetilde{d}, \widetilde{c})$ and we associate them to the darts of their respective triangles.

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

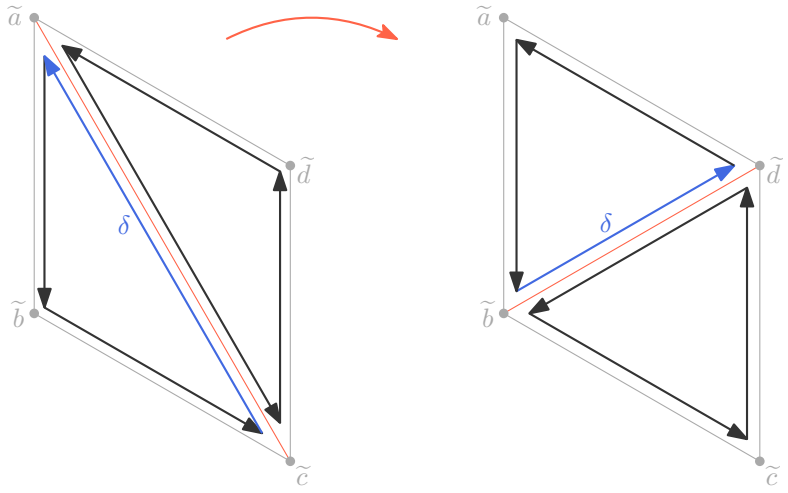

Figure 6: A flip. No darts are added or removed, only adjacencies are modified.

5 The Implemented Algorithm

The data structure is implemented in `Anchored_hyperbolic_surface_triangulation_2`, a class inherited from Dubois' `Hyperbolic_surface_triangulation_2` [DDT]. 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 5.1). 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 5.2). The point location is detailed in Section 5.3. 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 5.4).

5.1 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 computing a circumradius is expensive in practice, so, we avoid such unnecessary computations.

To this aim, we could maintain a set of large triangles. However the insertion of the circumcenter of a large triangle Δ does not only remove Δ from this set, but also other triangles, which may also be large. This implies that we would have done unnecessary expensive circumradii computations for triangles that are removed.

Instead of that set of large triangles, we maintain a list of triangles to be processed and only check whether they are large when necessary. In fact, we maintain a list $\mathcal{L}$ of darts representing triangles. The list is initialized with one dart for each triangle of the input triangulation. The only operations on this list are: pop the front dart, and push new darts at the back. The computation of a circumradius is only performed when a dart is popped from the front.

Instead of that set of large triangles, we maintain a list of triangles to be processed and only check whether they are large when necessary. In fact, we maintain a list $\mathcal{L}$ of darts representing the triangles; each dart has a Boolean mark and we maintain the property that each triangle is represented in $\mathcal{L}$ by exactly one marked dart. The list is initialized with one

marked dart for each triangle of the input triangulation (for a reasonable choice of ε, all these triangles are large). When a dart is popped out of $\mathcal{L}$, 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 a triangle 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. The Delaunay property is restored using the flip algorithm of Despré *et al.* [DDKT22]. After a flip, in each of the two new triangles, we maintain the property that only one dart is marked so that each triangle is represented at most once in the list $\mathcal{L}$. Indeed, such a new triangle may have 0, 1 or 2 marked edges, if it has none, we mark one and push it back in $\mathcal{L}$, if it has two, we unmark one of them and otherwise there is nothing to do.

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 $\mathcal{L}$. A circumradius is computed only when a marked dart is popped out from the front of $\mathcal{L}$; if it is not marked nothing is done: it means that the triangle it belongs to is represented by another dart. The algorithm proceeds with the new front dart until $\mathcal{L}$ is empty.

In the description above, the list $\mathcal{L}$ is not ordered. We ran experiments to check possible orders, in particular, ordering the list according to their circumradii (see Appendix A). It turns out that ordering $\mathcal{L}$ actually does not improve running times. Darts representing new triangles are always pushed at the end of $\mathcal{L}$, so, in practice, large triangles tend to be close to the front and are processed before triangles with smaller circumradii.

5.2 Circumcenters and Circumradii

When a dart is popped out from $\mathcal{L}$, its circumcenter is computed from the vertices of the anchor of that triangle. The coordinates of a circumcenter of three vertices of the triangulation with rational coordinates are algebraic numbers of degree two [BDT14]. 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 lead to computations that are known to be impossible to handle in practice.

The coordinates of the circumcenter $\widetilde{c}_\Delta$ of a triangle Δ are represented by the type `CGAL::Sqrt_extension`, which handles algebraic numbers of degree two. Before inserting $\widetilde{c}_\Delta$, we round it to a double type and convert it to an exact rational type from this double precision number. More specifically, coordinates of points in 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 [dt]. All the computations on our data structure are performed using this `CGAL::Exact_rational` number type for points and cross-ratios.

To avoid as much as possible the use of hyperbolic functions, which cannot be evaluated exactly, we detail the computation of the largeness test for triangles. In the Poincaré disk, the distance between two points $\widetilde{u}$ and $\widetilde{v}$ is $d_{\mathbb{H}^2}(\widetilde{u},\widetilde{v}) = \operatorname{arcosh}(1+\delta(\widetilde{u},\widetilde{v}))$, where $\delta(\widetilde{u},\widetilde{v}) = \dfrac{2||\widetilde{u}-\widetilde{v}||^2}{(1-||\widetilde{u}||^2)(1-||\widetilde{v}||^2)}$ and $||\cdot||$ is the Euclidean distance. We compute the minimum δ between the approximate circumcenter 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 add the approximate circumcenter $\widetilde{c}$ in the triangulation. The certified upper bound on the cosh function is computed with the Boost interval library [Boo]. 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 approximate circumcenter $\widetilde{c}$ 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 on the final output of the algorithm (Section 5.4).

5.3 Point Location

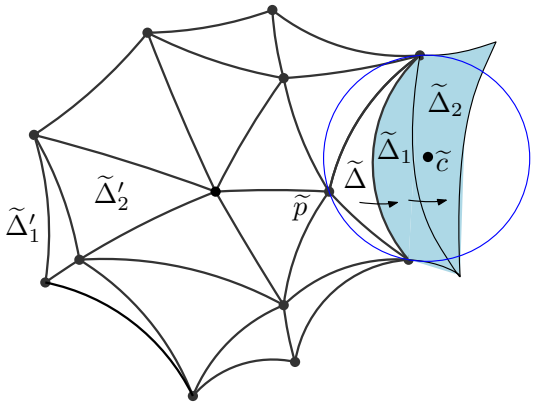

Figure 7: A fundamental domain of the surface is given by the anchors of the triangles (thick black lines). The blue triangles are the lifts constructed by the visibility walk to reach the circumcenter $\widetilde{c}$ of $\widetilde{\Delta}$. Triangles $\widetilde{\Delta}'_1$ and $\widetilde{\Delta}'_2$ 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 locate each new circumcenter by walking in the current triangulation itself.

We tested two walk algorithms: the straight walk and the visibility walk (Figure 7). In our case, the query point $\widetilde{c}$ is the approximate circumcenter of a lift $\widetilde{\Delta}$ of the triangle being processed by the algorithm. The walk starts from a vertex $\widetilde{p}$ of the triangle $\widetilde{\Delta}$. The algorithm uses the adjacency relations in the combinatorial map. In addition, lifts of triangles must be constructed along the walk to find the triangle lift containing $\widetilde{c}$. Both walks are based on orientation tests in $\mathbb{H}^2$ with vertices of lifted triangles. The combinatorial map encoding of the triangulation provides a direct access to the neighbor Δ' of a triangle Δ adjacent through one of its edges. On the other hand, 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 Appendix 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 of the original algorithm on the length of the straight walk, using Dirichlet domains, does not apply to the visibility walk. However we observe, in Section 7.1, that the walk has constant length in practice.

5.4 ε-Covering and ε-Packing Checks

As explained in Section 5.2, the output 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 $\delta(\widetilde{c}_\Delta, \widetilde{v})$ with the same type (see Section 5.2) 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 $\delta(\widetilde{v}_i, \widetilde{v}_j)$ for two vertices of an edge is rational and it is checked to be larger than an upper bound on $\cosh(\varepsilon) - 1$.

An alternative robust algorithm would be to certify the largeness test for triangles and check the ε-packing property after the restoration of the Delaunay property at each insertion. Indeed, the largeness test can done exactly using computations in a degree two extension. Checking the ε-packing property only involves exact rational computations. If it fails, this means that the inserted point was not a good enough approximation of the circumcenter and iteratively refining it would eventually recover the ε-packing property. Note that iterative refinement of the rational bounds on $\cosh(\varepsilon) - 1$ must also be computed.

Our experiments, presented in Section 7, show that for surfaces with a large systole, this more involved algorithm is not necessary: Using a double precision for the approximation of the exact circumcenter and to compute bounds on $\cosh(\varepsilon) - 1$ constructs valid ε-nets. On the other hand, we also observe that higher precision would be needed to handle a surface with a very small systole (Section 7.2).

6 Generation of Input Surfaces and Triangulations

To experiment with an algorithm in practice, it is important to get both relevant and tractable input. For an algorithm computing triangulations in $\mathbb{R}^2$ or $\mathbb{R}^3$ [BDTY00], or even in a quotient space like the flat torus [CT09], authors would typically generate millions of uniformly distributed points, or use data given by some application field. In order to make their implementation robust, they would rely on (filtered) exact predicates on rational coordinates.

However, getting relevant input hyperbolic surfaces, or even *defining* what "relevant" means in this context, is a major challenge. Arithmetic issues are also a major obstacle here, as our algorithm relies on constructions. The next two sections elaborate on these issues.

6.1 Well-Distributed Surfaces?

The situation was nicely described by Mirzakhani [Mir13] (though there are more recent results [Mon22]). 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 all marked hyperbolic surfaces. Indeed, in $\mathcal{T}_g$, as opposed to in $\mathcal{M}_g$, applying a non-trivial homeomorphism to a hyperbolic surface gives a different element in $\mathcal{T}_g$. Thus 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 non-trivial 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 [BM04, Pet17]. 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. 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 "thick" surface (see Section 2.2) with a reasonably long systole. It will be the case in most of our input surfaces. For completeness, we will also enforce a surface with a small systole and experimentally observe the impact in Section 7.2.

6.2 Arithmetic Issues

The 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. Delaunay triangulations could be computed on the very specific Bolza surface, represented by algebraic numbers. There, input points had rational coordinates, but computations with the group of the surface led to numbers whose degrees could be controlled by tailored computations [IT17]. The same approach was shown to fail already for genus 3 [EITV22] in practice with the CORE library [cor]. Our algorithm is generic, the arithmetic issues must be considered independently of a given specific surface. Even more than for standard benchmarks on points in the Euclidean space, using rational numbers is key for running software and go beyond toy cases.

The Teichmüller space $\mathcal{T}_g$ can be parameterized either from a pants decomposition and the Fenchel-Nielsen coordinates or by a fundamental polygon in $\mathbb{H}^2$ and side-pairings [Bus10, § 6]. Since we aim at computing triangulation in $\mathbb{H}^2$, starting from a fundamental polygon is the right choice to avoid the use of hyperbolic trigonometric functions. Any genus 2 surface has a fundamental domain that is a symmetric octagon centered at the origin [ABC+05]. 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 and this construction generically leads to algebraic numbers. For higher genus surfaces, a construction of fundamental polygons is described in [ZVC06, § 6.11], but unfortunately it does not lead to tractable numbers.

Restricting to complex numbers with rational real and imaginary parts gives a dense subset of the Teichmüller space $\mathcal{T}_2$ [DDKT22]. This means that for any genus 2 surface, we can work on an arbitrary close surface given by rational numbers. This allows us to only use exact rational computations for constructing an ε-net (see Section 5.2) and algebraic extensions of degree 2 to check its correctness (see Section 5.4). The eight vertices of the domain actually project on the same point on the surface. 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 [DDKT22]. The data structure produced by Dubois' code 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 to their respective face) using cross-ratios, by successively computing lifts of adjacent triangles, as detailed in Section 4.

7 Experiments

All the experiments of Section 7.1 are performed on 180 random surfaces of genus 2 that are expected to have a large systole (see Section 6.1). Section 7.2 focuses on a surface with a small systole. Section 7.3 displays and analyzes visualizations of the output.

The source code and complete data of the experiments are available on GitHub[1].

7.1 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 6. 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. 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 $16(g-1)/\varepsilon^2 = 16/\varepsilon^2$ since $g = 2$ 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 we do not have a complexity analysis of the visibility walk for the point location, we observe that the walk is traversing a (small) constant number of triangles. The average number of computed triangle lifts during each walk tends to decrease when ε becomes smaller, while the average proportion of points located in the starting triangle lift of the walk tends to increase, as shown in Figure 8. On average, 68% of the approximate circumcenters lie in their triangle lift. Over all surfaces and all tested values of ε, the farthest located points were 4 triangles away from the starting triangle of the walk.

Since in practice, the point location performs a constant number of operations, the complexity of the algorithm is proportional to the total number of flips. We counted the number of flips done to retrieve the Delaunay property of the triangulation after each insertion. On average, it is decreasing when ε becomes smaller, going from 3.41 for $\varepsilon = 0.5$ to 2.41 for $\varepsilon = 0.01$, as shown in Figure 9. It shows that the number of flips at each insertion is in practice a small constant. 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 Appendix C, we show an order of magnitude for the running times to give an idea of the practical complexity of our implementation. Interestingly, it appears to be slightly faster than quadratic in $1/\varepsilon$.

7.2 Surface With a Thin Part

As mentioned in Section 6, surfaces with a small systole are not common. Recall that we generate our input by choosing three random points in the Poincaré disk to form a symmetric octagonal fundamental domain. We designed a surface of genus two 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.

[1] https://anonymous.4open.science/r/SOCG_2025_submission_implementation_epsilon_net-D4C2

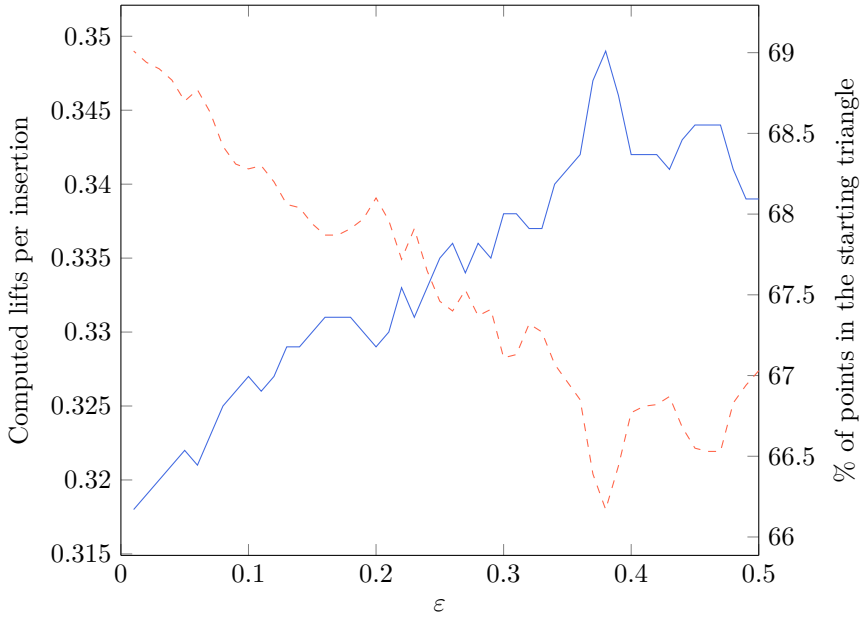

Figure 8: Average number of computed triangle lifts in the walk at each locate query (left, solid), and average percentage of points lying in the initial triangle lift of the walk (right, dashed).

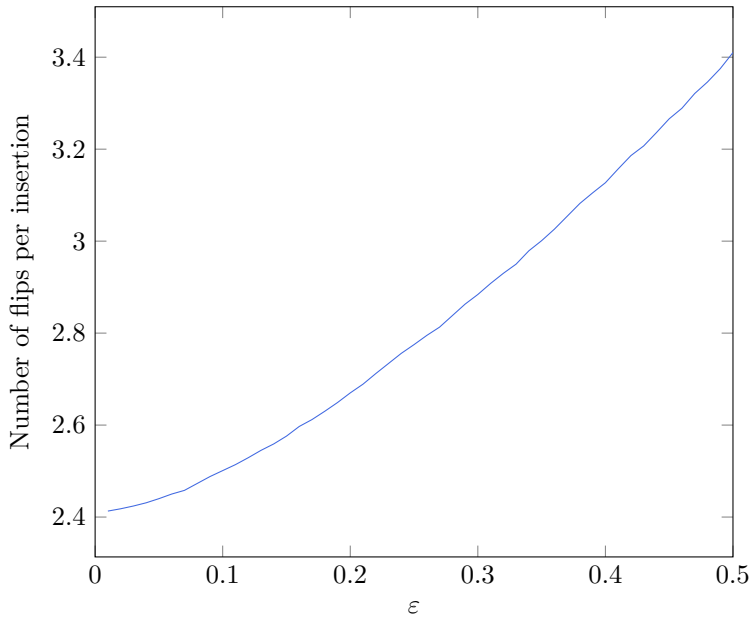

Figure 9: Average number of flips done to recover the Delaunay property after each insertion.

Precision issues The `epsilon_net` method produces an ε-net for $\varepsilon \geqslant 0.22$, which we certify using the method described in Section 5.4. For $\varepsilon = 0.21$, the output is not an ε-covering, which means that the algorithm misses some large triangles. However, it is an $1.05 \times \varepsilon$-covering. This indicates that this problem is likely a precision issue, which motivates the need for a higher precision approximation of the circumcenters (see Section 5.4).

Number of vertices With a systole less than 10^{-4}, the number of points in the output 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 2). However, the observed number of points is far from this bound. The results shows that it actually depends on a quantity $w(\sigma, \varepsilon)$, defined below, instead of $1/\sigma^2$.

The *injectivity radius* $r_x(S)$ of a point $x \in S$ is the supremum of all $r > 0$ such that the open ball of radius r centered at x is an embedded disk in S [Bus10, Chapter 4]. When $\sigma < \varepsilon$, the surface has an ε-thin part defined as $\{x \in S \mid r_x(S) \leqslant \varepsilon\}$. This ε-thin part is composed of one or several cylindrical collars around the closed geodesics of length less than ε. The *width* of the collar around σ is twice the distance between its boundary and σ. It is equal to $w(\sigma, \varepsilon) = 2\,\mathrm{arcosh}(\sinh(\varepsilon/2)/\sinh(\sigma/2))$ [EPV22]. The geometric intuition is that the number of points in an ε-net of a cylindrical collar is linear in its width. In Table 2, we observe that the number of vertices in the output of this surface is proportional to $16/\varepsilon^2 + w(\sigma, \varepsilon)/\varepsilon$.

Table 2: Comparison between the number of vertices in the output 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 on surfaces with large systoles (Section 7.1).

ε	**0.5**	**0.45**	**0.4**	**0.35**	**0.3**	**0.25**
Number of vertices	58	64	75	108	137	179
$0.5383 \times 16/\varepsilon^2 + w(10^{-4}, \varepsilon)/\varepsilon$	54	64	78	98	127	174

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 build a fundamental domain from a triangulation 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 lift one anchor of a triangle and build lifts of the other triangles starting from this anchor. Additionally, we translate one vertex $\widetilde{v}$ of the initial anchor to the origin of the Poincaré disk (and apply the same translation to the 2 other vertices) to obtain a drawing that is better centered in the unit disk. Once the starting triangle lift chosen, the drawing only depends on the order of processing of the adjacent triangles.

In Figures 10 and 11, the drawings are computed by Dubois' code that 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 a computed lift, the next lift being computed is the one in $T \setminus T'$ that shares the edge of least weight with T'. To ensure that the drawings are comparable when we run the ε-net algorithm with different values of ε on a same surface, we make sure that $\widetilde{v}$ is always the same point.

The drawings of triangulations shown in Figure 10 naturally look like Dirichlet domains since the weight on the edges ensures that the lifts of triangles entirely contained in the Dirichlet domain will be drawn. This becomes clear when drawing the Dirichlet domain of $\widetilde{v}$ translated to the origin (Figure 12). Since the input of the ε-net algorithm is a Delaunay triangulation of a single vertex, we obtain a Dirichlet domain by computing the circumcenter of all the lifted triangles incident to $\widetilde{v}$ in the input, 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 actually represents a *combinatorial Dirichlet domain*, see Figure 13. We can observe that, for larger values of ε, many triangles can be drawn outside the Dirichlet domain.

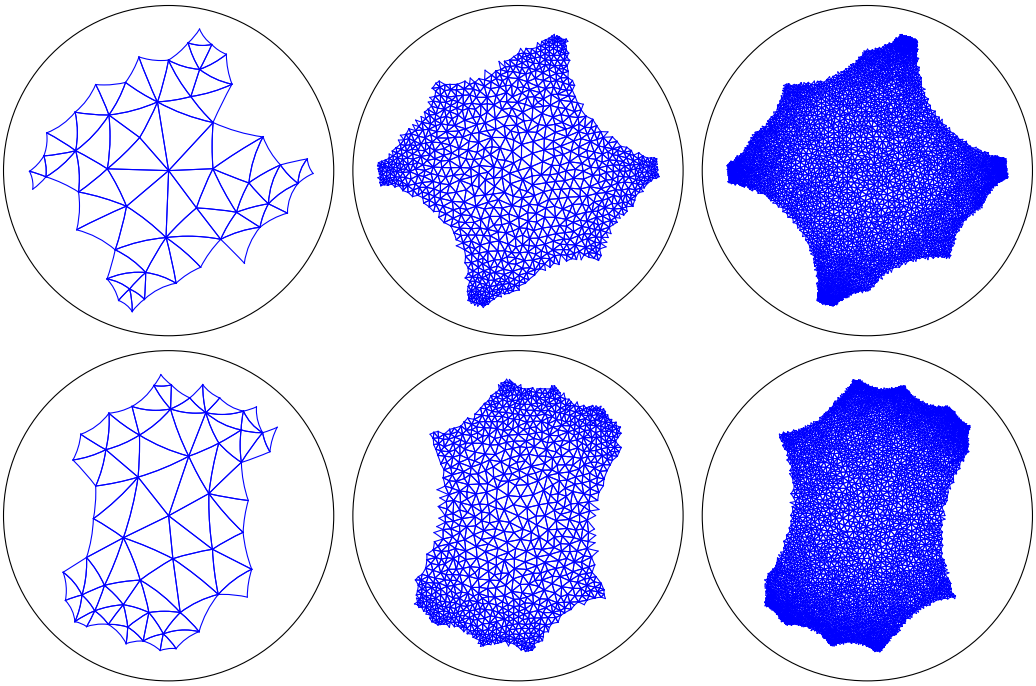

Figure 10: Delaunay triangulations of the computed ε-nets for ε=0.5 (left), ε=0.1 (middle), ε=0.05 (right) on the surfaces of seed 123 (top) and 321 (bottom).

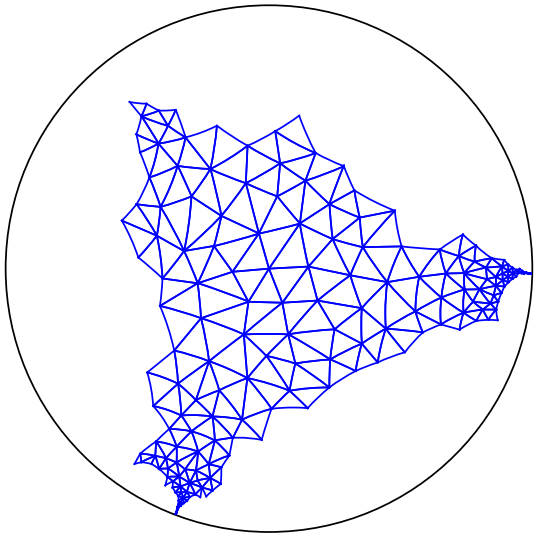

Figure 11: Delaunay triangulation of the computed 0.25-net of the surface with a small systole of Section 7.2. We can observe the collar around the systole in the form of "horns" pointing close to the boundary of the Poincaré disk.

But for smaller values of ε, the lift fits the domain better. It means that considering the length of curves by their number of intersections with the triangulation of an ε-net leads to a combinatorial distance that seems to converge to the hyperbolic metric (up to some constant factor). Making this statement a theorem is an interesting open question.

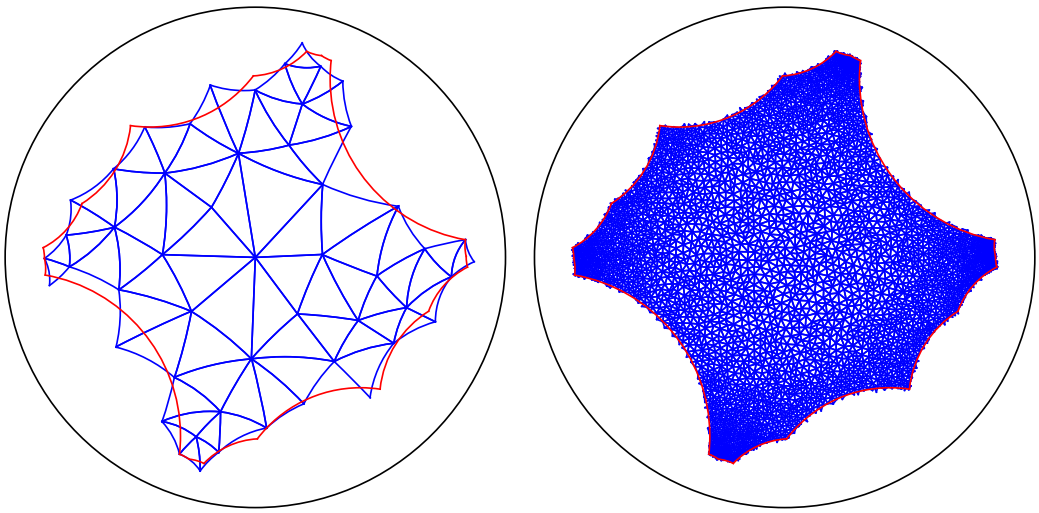

Figure 12: Delaunay triangulations of the computed 0.5-net (left) and the 0.05-net (right) on the surface of seed 123, and the corresponding Dirichlet domain. Lift computed with Dubois' algorithm.

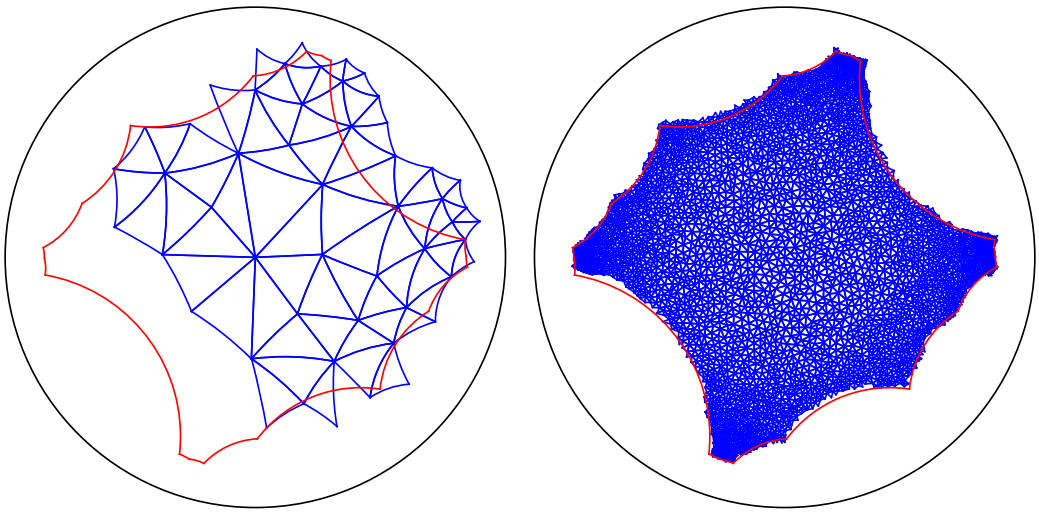

Figure 13: Delaunay triangulations of the computed 0.5-net (left) and 0.05-net (right) on the surface of seed 123, and the corresponding Dirichlet domain. Lift computed with a BFS algorithm.

References

[ABC+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, March 2005. doi:10.1063/1.1850177.

[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.

[BDTY00] Jean-Daniel Boissonnat, Olivier Devillers, Monique Teillaud, and Mariette Yvinec. Triangulations in CGAL. In *Proceedings 16th Annual Symposium on Computational Geometry*, pages 11–18, 2000. URL: https://hal.inria.fr/hal-01179408, doi:10.1145/336154.336165.

[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.

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

[Boo] *BOOST C++ Libraries*. URL: http://www.boost.org.

[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.

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

[CT09] Manuel Caroli and Monique Teillaud. Computing 3D periodic triangulations. In *Proceedings 17th European Symposium on Algorithms*, volume 5757 of *Lecture Notes in Computer Science*, pages 59–70, 2009. URL: https://hal.inria.fr/hal-02954152, doi:10.1007/978-3-642-04128-0_6.

[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.

[DDT] Vincent Despré, Loïc Dubois, and Monique Teillaud. Package: Hyperbolic_surface_triangulation_2. URL: https://github.com/loic-dubois/cgal/tree/Hyperbolic_surface_triangulation_2-dubois/Hyperbolic_surface_triangulation_2.

[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.

[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.

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

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

[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. URL: https://hal.inria.fr/hal-03664678, doi:10.20382/jocg.v13i1a5.

[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.

[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.

[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.

[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.

[ZVC06] Heiner Zieschang, Elmar Vogt, and Hans-Dieter Coldewey. *Surfaces and planar discontinuous groups*, volume 835. Springer, 2006.

A Managing the List $\mathcal{L}$

We compared different ways to manage the list $\mathcal{L}$ of darts representing the triangles to be considered by the algorithm, as mentioned in Section 5. Each of these ways is implemented in an algorithm called *variant*. Variant A is the final algorithm presented in this paper. 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 pushing (or not) one of its darts into $\mathcal{L}$. Variant D is like variant C, but in addition, the darts of the list are ordered according to the circumradius of their triangle, the larger ones being at the front. The results, presented in Table 3, show that any attempt to sort the list in a specific way, or to optimize its memory consumption is counter-productive.

Table 3: 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

These running times 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.

B Point Location Walks Comparison

We compared the performance of the visibility walk and the straight walk within the ε-net algorithm, as mentioned in Section 5.3. 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 triangle lift. These results are summarized in Table 4. Note that, for a given surface and a given parameter ε, the obtained ε-net can be different when using a walk or the other. This is due to implementation details of the walks, leading to darts being treated in a different order in $\mathcal{L}$.

Table 4: 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 triangle lift.

	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

These running times 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.

C Running Time

We give some order of magnitude of the running time of the algorithm. The complete results are summarized by a graph (Figure 14). All the given times are averages over the 180 surfaces used in the experiments. These times are obtained on a cluster 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 $\varepsilon = 0.1$, and 47.46 seconds for $\varepsilon = 0.02$. It takes about 2 minutes and 40 seconds to run the algorithm for $\varepsilon = 0.01$. 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 actual time-complexity of the algorithm is therefore sub-linear in the number of points, which is $O(1/\varepsilon^2)$. This behavior can be explained by the number of lifts computed for the point location and the number of flips at each insertion, which are both decreasing as ε decreases.

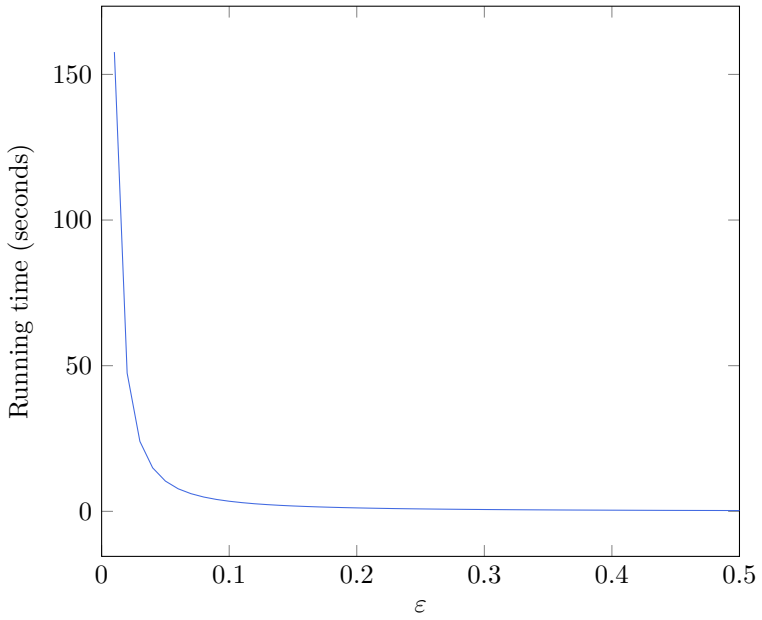

Figure 14: Average running time (seconds) of the ε-net algorithm.