GPU-Based Computation of Discrete Periodic Centroidal Voronoi Tessellation in Hyperbolic Space

Liang Shuai [a], Xiaohu Guo [a], Miao Jin [b]
[a] *University of Texas at Dallas*
[b] *University of Louisiana at Lafayette*

Abstract

Periodic centroidal Voronoi tessellation (CVT) in hyperbolic space provides a nice theoretical framework for computing the constrained CVT on high-genus (genus > 1) surfaces. This paper addresses two computational issues related to such hyperbolic CVT framework: (1) efficient reduction of unnecessary site copies in neighbor domains on the universal covering space, based on two special rules; (2) GPU-based parallel algorithms to compute a discrete version of the hyperbolic CVT. Our experiments show that with the dramatically reduced number of unnecessary site copies in neighbor domains and the GPU-based parallel algorithms, we significantly speed up the computation of CVT for high-genus surfaces. The proposed discrete hyperbolic CVT guarantees to converge and produces high-quality results.

Key words: Centroidal Voronoi Tessellation, Universal Covering Space, Hyperbolic Space, GPU Algorithm

1. Introduction

The centroidal Voronoi tessellation (CVT) [4] is a special type of Voronoi diagram, where every site coincides exactly with the centroid of its Voronoi cell. The celebrating Gersho's conjecture [8] in 2D, proved by Tóth [7], states that the shape of the Voronoi cells converges to uniform regular hexagons when the CVT is optimized globally. This property inspired many researchers to compute the constrained CVT [5] on surfaces, for applications where a uniform sampling or remeshing of the surface is desired.

Different methods of computing constrained CVT on surfaces can be roughly categorized into two classes: "extrinsic" and "intrinsic" approaches. Extrinsic approaches [5,11,24] compute an Euclidean Voronoi diagram in the ambient 3D space and its intersection of a surface, with sites constrained on the surface. If two regions of the surface are close in the 3D space but far away along the surface, the computed constrained CVT on surface tends to be incorrect [17,18].

Intrinsic approaches [1,17,18], which overcome the above limitations of their extrinsic counterparts, compute the CVT in a 2D parameter domain of a surface, with a density function applied to compensate the introduced area distortion of surface parametrization. To allow sites move freely across artificially cut open surface boundaries on its parameter domain for non-topological disk surfaces, Rong et al. [18] proposed to compute the CVT in a 2D periodic parameter domain, called the *Universal Covering Spaces* of surfaces, which are 2D spaces with constant curvatures – spherical, Euclidean, and hyperbolic spaces.

Computing a CVT in a 2D periodic parameter domain is equivalent to computing a periodic CVT in its corresponding space. Computing a periodic CVT in spherical and Euclidean spaces has been well studied in previous literatures [5,6,25]. Thus the main challenge resides in the efficient computation of the periodic CVT in hyperbolic space. In this paper we propose several strategies to speed up the computation of periodic CVT in hyperbolic space, including two special rules to efficiently reduce the number of site copies, and a GPU-based parallel computation framework of the discrete hyperbolic CVT.

1.1. *Preliminaries*

We first introduce briefly the concept of universal covering space, and then present two definitions related to periodic CVT in hyperbolic space.

A *covering map* is a surjective continuous map π from a topological space $\bar{U}$ to another topological space U such that any point $\mathbf{p}$ in U has a neighborhood $N(\mathbf{p})$ satisfying $\pi^{-1}(N(\mathbf{p}))$ is a collection of disjoint sets, and each set can

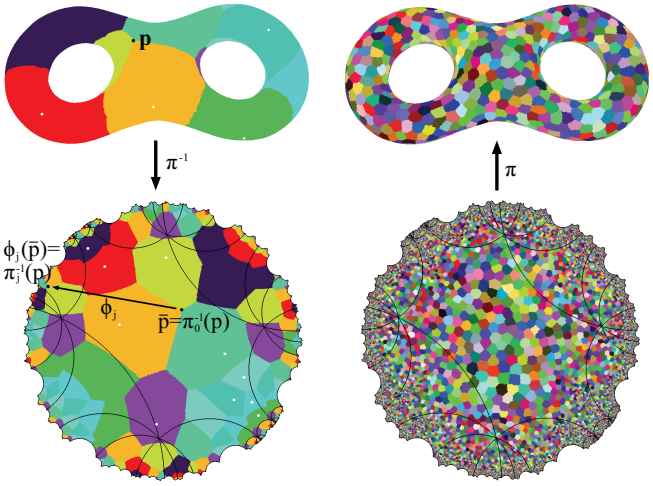


Fig. 1. Left: a point $\mathbf{p}$ on the surface is mapped to $\bar{\mathbf{p}}$ in the center domain by the inverse covering map π_0^{-1}, and $\bar{\mathbf{p}}$ can be mapped to $\pi_j^{-1}(\mathbf{p})$ in the neighbor domain j by the deck transformation ϕ_j; Right: we can get the CVT result on a genus-2 surface by computing a periodic CVT in the hyperbolic UCS.

be homeomorphically mapped onto U by π. $(\bar{U}, \pi)$ is called the *covering space* of U, and sometimes people simply denote it as $\bar{U}$. $(\bar{U}, \pi)$ is the *universal covering space (UCS)* of U if $\bar{U}$ is simply connected. A *deck transformation* $\phi : \bar{U} \to \bar{U}$ keeps the covering map π unchanged: $\pi = \pi \circ \phi$. All deck transformations form a so called *Fuchsian group* G. A *fundamental domain* F of the UCS is a subset of $\bar{U}$ such that $\bar{U} = \cup_{\phi \in G} \phi(F)$. The UCS of a high-genus (genus > 1) surface can be conformally embedded into a 2D hyperbolic space [10]. Figure 1 shows a torus surface and its UCS embedded on hyperbolic plane. We refer readers to Munkres's book [14] for more details.

Definition 1 *Let U be a surface and $(\bar{U}, \pi)$ be its UCS. Given a point set $S = \{\boldsymbol{s}_i \in U \mid i = 1, ..., n\}$ on U, the Voronoi diagram in universal covering space of U induced by S can be defined as the subdivision of $\bar{U}$ into Voronoi cells Ω_i^{ucs}:*

$$\begin{aligned}\Omega_i^{ucs} =& \{\boldsymbol{p} \in \bar{U} \mid d_{\bar{U}}(\boldsymbol{p}, \bar{\boldsymbol{s}}_i) < d_{\bar{U}}(\boldsymbol{p}, \bar{\boldsymbol{s}}_j), \\ & \exists\, \bar{\boldsymbol{s}}_i \in \pi^{-1}(\boldsymbol{s}_i), \forall\, \bar{\boldsymbol{s}}_j \in \pi^{-1}(\boldsymbol{s}_j), \forall\, j \neq i\},\end{aligned} \tag{1}$$

where $d_{\bar{U}}(\cdot, \cdot)$ is the distance in space $\bar{U}$.

Note that in the definition each point (or *site*) $\mathbf{s}_i$ is duplicated into infinite number of copies via π^{-1}. The fundamental domain is periodically repeated in UCS, so a Voronoi diagram in UCS is also a *periodic Voronoi diagram (PVD)*.

Due to its periodicity, the PVD defined in (1) only needs to be computed in one fundamental domain (referred as *center domain* $\bar{U}_0$ later). Define the *neighbor domains* of $\bar{U}_0$ be the fundamental domains which share at least one vertex with $\bar{U}_0$, denoted by $\bar{U}_j$, $j = 1, ..., h$, where $h = 16g^2 - 8g$ is the number of neighbor domains for a genus-g surface. Use $\pi_j^{-1}(\mathbf{s})$ to denote the preimages of site $\mathbf{s}$ in different fundamental domains ($j = 0$ for center domain and $j = 1, ..., h$ for neighbor domains). Let ϕ_j be such a deck transformation that $\phi_j(\bar{\mathbf{s}}) = \pi_j^{-1}(\mathbf{s})$, where $\bar{\mathbf{s}} = \pi_0^{-1}(\mathbf{s})$ (Figure 1). Then we can refer $\phi_j(\bar{\mathbf{s}}_i), j = 1, ..., h$ as the *site copies* of site $\bar{\mathbf{s}}_i = \pi_0^{-1}(\mathbf{s}_i)$ later for convenience.

This paper focuses on PVD in 2D hyperbolic space. There are different models of 2D hyperbolic space, such as Poincaré disk, Klein disk, Poincaré half-plane, and Minkowski model. All these models are equivalent. In this paper we use the Poincaré disk model to visualize the hyperbolic Voronoi diagram (see Figure 1) and Minkowski model to define the centroid of a hyperbolic region [18].

Definition 2 *Given a region Ω on the Minkowski model, and density $\rho(\boldsymbol{p})$ for any point $\boldsymbol{p} \in \Omega$, the* centroid *of Ω is defined as:*

$$\boldsymbol{c} = \frac{1}{\eta} \int_\Omega \rho(\boldsymbol{p}) \boldsymbol{p}\, d\boldsymbol{p}, \tag{2}$$

where

$$\eta = \| \int_\Omega \rho(\boldsymbol{p}) \boldsymbol{p} d\boldsymbol{p} \|_M. \tag{3}$$

$\| \cdot \|_M$ is the Minkowski norm which can be defined through its inner product: $\| \cdot \|_M = \sqrt{\langle \cdot, \cdot \rangle_M}$.

Here the Minkowski inner product is defined as $\langle \mathbf{p}, \mathbf{q} \rangle_M = z_\mathbf{p} z_\mathbf{q} - x_\mathbf{p} x_\mathbf{q} - y_\mathbf{p} y_\mathbf{q}$ for two points $\mathbf{p} = (x_\mathbf{p}, y_\mathbf{p}, z_\mathbf{p})$ and $\mathbf{q} = (x_\mathbf{q}, y_\mathbf{q}, z_\mathbf{q})$ on Minkowski model. Note that their hyperbolic distance can be computed by $d_M(\mathbf{p}, \mathbf{q}) = \cosh^{-1}(\langle \mathbf{p}, \mathbf{q} \rangle_M)$.

Given a set of sites $S = \{\mathbf{s}_i \mid i = 1, ..., n\}$ on Minkowski model, and its Voronoi diagram as $\Omega = \cup \Omega_i$, where Ω_i is the Voronoi cell associated with site $\mathbf{s}_i$, the *hyperbolic CVT energy* is defined as:

$$E(S, \Omega) = \sum_i \int_{\Omega_i} \rho(\mathbf{p}) \cosh(d_M(\mathbf{p}, \mathbf{s}_i))\, d\mathbf{p}. \tag{4}$$

With the above defined Voronoi diagram and centroid in hyperbolic space, the hyperbolic CVT energy is proved to converge with Lloyd's algorithm [18].

1.2. *Motivation and Contribution*

Rong et al. [18] proposed a nice periodic CVT framework in hyperbolic space. However, two computational issues hinder their algorithms from being practical for general high-genus surfaces.

The first issue is that they compute the part of PVD within a center domain from a full site copies of the center domain and its neighbor domains: $\{\pi_j^{-1}(\mathbf{s}_i) \mid \forall \mathbf{s}_i \in S, j = 0, ..., h\}$, and then the intersection of the resulting Voronoi diagram with the center domain. Although the shape of periodic Voronoi cells inside the center domain can be affected by the site copies located in its neighbor domains, there are $16g^2 - 8g$ neighbor domains for a genus-g surface and only a small portion of the site copies in them will affect the Voronoi cells near the boundary of the center domain. In Section 3, We propose two simple rules, which can be computed efficiently, to significantly reduce the number of unnecessary site copies in neighbor domains.

The second issue is that computing a hyperbolic CVT is extremely time-consuming. For example, computing 1000 sites on a genus-3 Sculpture surface takes around 55 seconds for each Lloyd's iteration on a desktop computer with

Core 2 Duo 2.93GHz CPU. We can utilize the parallel computability of the programmable GPU to accelerate this process. We first define a discrete version of the hyperbolic CVT with each triangle represented by its centroid and the constrained CVT approximated with clusters of triangles. Our discrete CVT is similar to Valette et al. [22]. However, it is formulated in a 2D periodic hyperbolic domain, with its discrete CVT energy proved to converge. We then introduce a parallel mesh flooding algorithm to efficiently compute the defined discrete hyperbolic PVD in Section 4.2. We further show that the energy of the discrete hyperbolic CVT is guaranteed to converge under our GPU-based computational framework in Section 5.

The contribution of this paper can be summarized as:

(i) Two computing efficient rules are introduced to reduce the unnecessary site copies in neighbor domains for computing hyperbolic PVD;

(ii) A GPU-based parallel mesh flooding algorithm is proposed to compute the discrete hyperbolic Voronoi diagram.

Both of the two aspects serve for speeding up the computation of hyperbolic CVT.

2. Related Work

In this section, we give a brief review of existing research related to this work.

2.1. *CVT Algorithms*

Before Du et al. [4] introduced the concept of Centroidal Voronoi Tessellation, there has been some studies about the related concepts in different areas. MacQueen [13] proposed one of the earliest CVT algorithms. However, the probabilistic nature makes its convergence very slow. Lloyd proposed a simple and deterministic algorithm [12], whose convergence was proved later by Du et al. [3]. Liu et al. [11] proposed a quasi-Newton method to optimize the CVT energy, by proving that the energy is C^2 continuous.

In all the above research, the CVT is computed in 2D/3D Euclidean space. Du et al. [5,6] computed the spherical CVT by treating it as the constrained CVT on the sphere. Rong et al. [17,18] extended the concept of CVT from the traditional Euclidean and spherical spaces into the unexplored Hyperbolic space [17], and unified the treatment of CVT in the three different spaces, including the CVT energy functions and their convergence [18].

2.2. *CVT on Surfaces*

CVT has been applied for computing uniform tessellation and remeshing of surfaces. Compared with traditional CVT on planar space, such constrained CVT [5] requires all the sites to be constrained on the surface. Using the 3D Euclidean distance as an approximation, Yan et al. [24] computed the CVT in 3D space and then intersect it with the surface. However, this approach can lead to disconnected Voronoi cells if two regions are close in 3D space but far apart along the surface, as shown by Rong et al. [17,18].

An alternative approach is to compute the CVT in the 2D parameter domain of the surface. Surazhsky et al. [21] used a local parametrization approach by projecting the 1-ring neighbors of a site onto the tangential plane, and finding the centroid of the Voronoi cell in the plane. Alliez et al. [1] parameterized a surface to a planar disk and then computed the CVT on the disk. Since a non-topological disk surface has to be cut open to a topological disk along a set of cutting edges, and sites are unable to cross the boundary of the parameterized disk, visible artifacts are left on surface along the cutting edges. This problem was solved later by Rong et al. [17,18] by computing the CVT on a 2D periodic hyperbolic parameterizing domain, where sites can move freely along or across any cutting boundaries.

2.3. *Periodic Voronoi Diagram*

Yan et al. [25] computed the periodic Voronoi diagram in 2D Euclidean space using a reduced set of periodic copies of the input sites. A related dual problem, periodic Delauney triangulation, has been studied by Caroli and Teillaud [2] on the 3D Euclidean space. The computation of the periodic Voronoi diagram in 2D hyperbolic space by Rong et al. [17,18] uses full copies of the sites in both the center domain and its $16g^2 - 8g$ neighbor domains for a genus-g surface. In this paper, we present two special rules to effectively reduce the set of unnecessary site copies of neighbor domains.

2.4. *GPU-Based CVT Computation*

Vasconcelos et al. [23] proposed to use GPU to compute CVT in a 2D plane, by using a predefined mask to estimate the Voronoi cell for each site. Rong et al. [19] presented a GPU-assisted method to compute the constrained CVT on a surface, based on its 2D geometry image. The surface is discretized as pixels on a rectangular domain, and then an efficient Jump Flooding Algorithm [20] is applied to compute the discrete Voronoi diagram. Rong et al.'s geometry-image approach [19] suffers from the same problem as Alliez et al. [1], where the sites cannot move freely across the boundary of the parametrization domain.

3. Reduction of Site Copies

As we discussed earlier, to compute hyperbolic PVD, we only need to compute the hyperbolic Voronoi diagram in the center domain, with necessary site copies in neighbor domains. In this section we introduce two simple rules to effectively improve the computational efficiency of hyperbolic Voronoi diagram in the center domain.

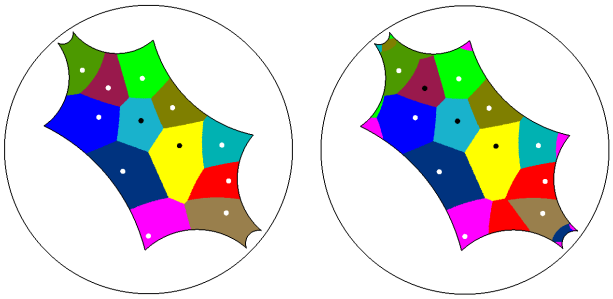
The key idea is that we want to keep only sites of the center domain and the necessary site copies in its neighbor

domains for the computation since the time complexity of computing hyperbolic Voronoi diagram is $O(n \log n)$ where n is the number of sites [15,17]. The challenge is that the exactly necessary site copies - the site copies of neighbor domains with their periodic Voronoi cells intersecting with the center domain - cannot be determined unless the hyperbolic PVD is computed.

Yan et al. [25] shows some simple and effective rules to achieve similar goal for Euclidean PVD. Rules for hyperbolic PVD, compared with its Euclidean counterpart, are more demanding because the number of site copies in neighbor domains increases quadratically with the genus of the surface, and are also more challenging because the deck transformation in 2D hyperbolic space (i.e. Möbius transformation) is less intuitive than rigid transformation in Euclidean plane. Basically, we want to answer the following two questions: 1) which site in center domain needs to be copied into neighbor domains; and 2) which neighbor domain copy of the site is necessary.

Let's refer the set of sites in center domain whose Voronoi cells intersect with the boundary of center domain as *boundary sites*, denoted by S_{bdy}. Since the Voronoi cells of non-boundary sites (or *inner sites*) in a PVD do not touch the boundary of center domain, their copies are not needed. Figure 2 shows a fact that all the inner sites of a non-periodic VD are still inner sites of the corresponding PVD. So all the inner sites of the non-periodic VD do not need to be copied. Then we have the following rule to answer the first question, which is similar to the conclusion in Yan et al.'s 2D Euclidean PVD [25].

First Rule: when we compute a non-periodic Voronoi diagram with sites of the center domain, only the boundary sites need to be copied to neighbor domains for computing the hyperbolic PVD.



(a) (b)

Fig. 2. The (a) non-periodic and (b) periodic Voronoi diagrams in center domain with same input sites. White dots denote boundary sites and black dots denote inner sites.

For the second question, we answer it with the following rule.

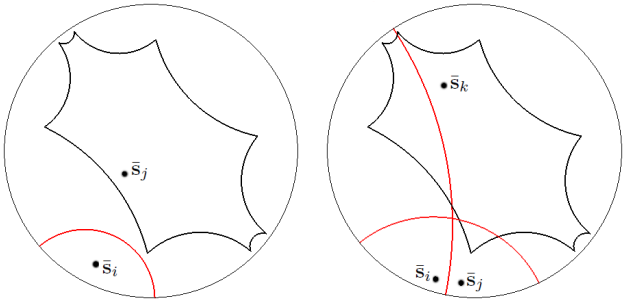
Second Rule: when we compute a non-periodic Voronoi diagram for the boundary sites and their copies in neighbor domains: $S_{bdy} \cup S_{copy}$, where $S_{copy} = \{\phi_j(\bar{\mathbf{s}}_i) | \bar{\mathbf{s}}_i \in S_{bdy}, j = 1, ..., h\}$, sites in S_{copy} with their Voronoi cells having intersection of the center domain are necessary.

The second rule is straightforward, but the computation of a non-periodic Voronoi cell for each site copy in S_{copy} is expensive. So we develop an alternative method to apply the second rule.

Consider a hyperbolic Voronoi diagram on Poincaré disk $\mathbb{D}$, the bisector between two sites $\bar{\mathbf{s}}_i$ and $\bar{\mathbf{s}}_j$ divides the hyperbolic space into two halves, denoted by $H(\bar{\mathbf{s}}_i, \bar{\mathbf{s}}_j)$ and $H(\bar{\mathbf{s}}_j, \bar{\mathbf{s}}_i)$, where

$$H(\bar{\mathbf{s}}_i, \bar{\mathbf{s}}_j) = \{\mathbf{p} \in \mathbb{D} \mid d_P(\bar{\mathbf{s}}_i, \mathbf{p}) \leq d_P(\bar{\mathbf{s}}_j, \mathbf{p})\}.$$

Here $d_P(\cdot, \cdot)$ denotes the hyperbolic distance in Poincaré disk. Then the Voronoi cell of $\bar{\mathbf{s}}_i$ is $\cap_{\forall \bar{\mathbf{s}}_j \neq \bar{\mathbf{s}}_i} H(\bar{\mathbf{s}}_i, \bar{\mathbf{s}}_j)$. If there exists a site $\bar{\mathbf{s}}_j$, and $H(\bar{\mathbf{s}}_i, \bar{\mathbf{s}}_j)$ does not intersect with the center domain, then the Voronoi cell of $\bar{\mathbf{s}}_i$ does not intersect with the center domain either. In this case it is said that $\bar{\mathbf{s}}_j$ *killed* $\bar{\mathbf{s}}_i$ (see Figure 3(a)). By testing the copies of boundary sites and removing the killed ones, we can dramatically reduce the number of unnecessary site copies.



(a) (b)

Fig. 3. Site reduction with the second rule. (a) Site $\bar{\mathbf{s}}_i$ is killed by $\bar{\mathbf{s}}_j$; (b) Site $\bar{\mathbf{s}}_i$ can be killed by the combination of $\bar{\mathbf{s}}_j$ and $\bar{\mathbf{s}}_k$, but it cannot be killed by either $\bar{\mathbf{s}}_j$ or $\bar{\mathbf{s}}_k$ individually. The red arcs are the bisectors between sites.

To test if a half plane $H(\bar{\mathbf{s}}_i, \bar{\mathbf{s}}_j)$ intersects with the center domain, we have the following theorem:

Theorem 1 *Given a hyperbolic polygon and two points* $\mathbf{p}$ *and* $\mathbf{q}$ *outside the polygon, if* $\mathbf{p}$ *has shorter distances to all corners of the polygon than* $\mathbf{q}$*, then* $\mathbf{p}$ *has shorter distances to all points inside the polygon than* $\mathbf{q}$*.*

The proof of Theorem 1 is given in Appendix A. According to this theorem, the distances from the two sites $\bar{\mathbf{s}}_i$ and $\bar{\mathbf{s}}_j$ to all the corners $\{\mathbf{p}_k | k = 1, ..., 4g\}$ of the center domain can determine whether the half plane $H(\bar{\mathbf{s}}_i, \bar{\mathbf{s}}_j)$ intersects with the center domain or not. If $d_P(\bar{\mathbf{s}}_i, \mathbf{p}_k) > d_P(\bar{\mathbf{s}}_j, \mathbf{p}_k)$ for all corners $\mathbf{p}_k$, then $H(\bar{\mathbf{s}}_i, \bar{\mathbf{s}}_j)$ does not intersect with the center domain. Thus the time complexity of such a simple pair-wise test is $O(g)$.

Note that the simplified pair-wise test can miss some unnecessary site copies, when a site $\bar{\mathbf{s}}_i$ can be killed by a combination of a group of other sites $\{\bar{\mathbf{s}}_{i_1} \dots \bar{\mathbf{s}}_{i_k}\}$ instead of any individual one in this group. Figure 3(b) shows an example where $\bar{\mathbf{s}}_i$ can be killed by the combination of $\bar{\mathbf{s}}_j$ and $\bar{\mathbf{s}}_k$, but it cannot be killed by either $\bar{\mathbf{s}}_j$ or $\bar{\mathbf{s}}_k$ individually.

In order to simplify the computation of site-copy-reduction for parallel computation (in Section 4.2.2), we

only perform such pair-wise test. We propose the following two different options for applying the second rule.

Option 1: For any copy of the boundary sites $\bar{\mathbf{s}}_i \in S_{copy}$, we test it against all the other sites $\bar{\mathbf{s}}_j \in S_{bdy} \cup S_{copy}$, $\bar{\mathbf{s}}_j \neq \bar{\mathbf{s}}_i$. If $\bar{\mathbf{s}}_i$ can be killed by $\bar{\mathbf{s}}_j$, we can eliminate $\bar{\mathbf{s}}_i$ from the set of necessary site copies.

Suppose the sites are uniformly distributed, then the average size of $|S_{bdy}|$ becomes $\sqrt{m}$. Thus the size of $|S_{copy}|$ is $\sqrt{m}(16g^2 - 8g)$. The total work of computation in option 1 is $O(m \cdot g^5)$. It will not be efficient even with parallel computation.

So we proposed the second option as follows. Since the boundary of the center domain is a closed loop, and the Voronoi cell of each boundary site $\bar{\mathbf{s}}_i \in S_{bdy}$ will intersect with this boundary loop, we can sort the boundary sites in S_{bdy} according to the order of intersection. Two boundary sites are next to each other if their intersections with the boundary loop are adjacent. Suppose S^{ord}_{bdy} is the list of such ordered boundary sites. Then we can simply test each copy of boundary sites against its neighbors in the ordered list.

Option 2: For any boundary site $\bar{\mathbf{s}}_i \in S^{ord}_{bdy}$, we check if its copy $\phi_j(\bar{\mathbf{s}}_i)$ in neighbor domain j is necessary, by testing $\phi_j(\bar{\mathbf{s}}_i)$ against the set of 5 other sites: $\{\bar{\mathbf{s}}_i, \phi_j(\bar{\mathbf{s}}_{i-1}), \phi_j(\bar{\mathbf{s}}_{i+1}), \phi_{j-1}(\bar{\mathbf{s}}_i), \phi_{j+1}(\bar{\mathbf{s}}_i)\}$, where $\bar{\mathbf{s}}_{i-1}$ and $\bar{\mathbf{s}}_{i+1}$ are two neighbors of $\bar{\mathbf{s}}_i$ in the ordered list S^{ord}_{bdy}, and the neighbor domains $(j+1)$ and $(j-1)$ "sandwich" the neighbor domain j. If $\phi_j(\bar{\mathbf{s}}_i)$ can be killed by any one of them, we eliminate it from the set of necessary site copies.

We can see that the total work of computation in Option 2 is only $O(\sqrt{m}g^3)$. Although Option 2 cannot achieve as many reductions as Option 1, such simple test can be efficiently computed with parallel implementation (Algorithm 2 in Section 4.2.2).

Figure 4 shows the number of site copies (1) before reduction; (2) after first rule; (3) after second rule with Option 1; (4) after second rule with Option 2; and (5) with optimal reduction. We experiment it on four different surfaces with 2000 sites in their center domain. The result of our site-copy-reduction schema depends on the shape of center domain and the position of sites. Figure 5 shows the result of our site-copy-reduction with varying number of sites on the genus-3 Sculpture surface. Section 6.2 shows the detailed comparison of running time with these different options.

4. Discrete Hyperbolic PVD

The computation of UCS on triangular mesh surfaces has been explored by Jin et al. [10]. Thus the concept of PVD in Definition 1 can be directly extended onto triangular meshes. Due to the discrete nature of triangular mesh it is possible to develop GPU-based parallel algorithms running on each individual triangle to speed up the computation of Voronoi diagram. In this section the discrete Voronoi diagrams are defined first. Then some parallel algorithms for computing discrete hyperbolic PVD are introduced and analyzed.

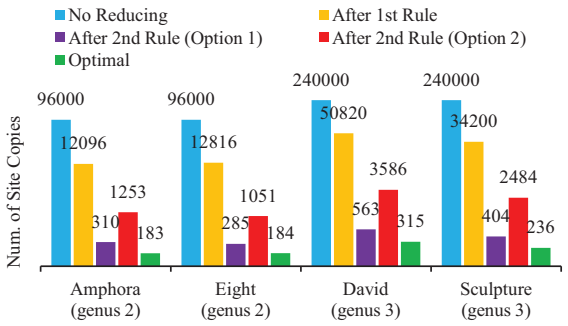


Fig. 4. The number of site copies before and after reduction, with 2000 sites in the center domain of different surfaces. Note that the heights of the bars are scaled by logarithm.

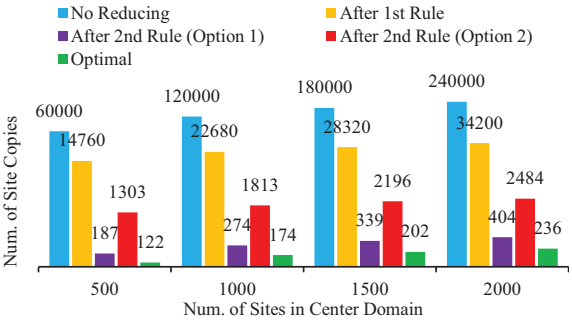


Fig. 5. The number of site copies before and after reduction, with varying number of sites in the center domain of the Sculpture surface. Note that the heights of the bars are scaled by logarithm.

4.1. *Definitions*

Definition 3 *Let $\bar{M} = (\bar{F}, \bar{V})$ be a hyperbolic triangular mesh on Poincaré disk, where $\bar{F}$ is the set of triangular faces and $\bar{V}$ is the set of vertices. The centroid of hyperbolic triangle $f \in \bar{F}$ is denoted by $\boldsymbol{c}_f$. Given a set of sites $S = \{\bar{\boldsymbol{s}}_i \in \bar{M} \mid i = 1, ..., n\}$, the discrete hyperbolic Voronoi diagram on $\bar{M}$ is defined as the union of face sets Ω^{dh}_i:*

$$\Omega^{dh}_i = \{f \in \bar{F} \mid d_P(\boldsymbol{c}_f, \bar{\boldsymbol{s}}_i) < d_P(\boldsymbol{c}_f, \bar{\boldsymbol{s}}_j), \forall j \neq i\}. \quad (5)$$

In the above definition, each triangle of the mesh is represented by its centroid. As will be shown in Section 5, the energy of the discrete hyperbolic CVT is guaranteed to converge when we represent each triangle by its centroid.

Next some notations are given to define the discrete Voronoi diagram in UCS. Let $M = (F, V)$ be a triangular mesh surface with genus g ($g > 1$) embedded in 3D Euclidean space. Let $(\bar{M}, \pi)$ be its UCS, where $\bar{M} = (\bar{F}, \bar{V})$ is also a triangular mesh on the Poincaré disk.

Definition 4 *Given a set of sites $S = \{\boldsymbol{s}_i \mid i = 1, ..., n\}$ on the surface mesh $M = (F, V)$, the discrete hyperbolic periodic Voronoi diagram (DH-PVD) on $\bar{M} = (\bar{F}, \bar{V})$ can be defined as the union of face sets Ω^{dhp}_i:*

$$\begin{aligned}\Omega^{dhp}_i = \{&f \in \bar{F} \mid d_P(\boldsymbol{c}_f, \bar{\boldsymbol{s}}_i) < d_P(\boldsymbol{c}_f, \bar{\boldsymbol{s}}_j), \\ &\exists\, \bar{\boldsymbol{s}}_i \in \pi^{-1}(\boldsymbol{s}_i), \forall\, \bar{\boldsymbol{s}}_j \in \pi^{-1}(\boldsymbol{s}_j), \forall\, j \neq i\}.\end{aligned} \quad (6)$$

This definition is the discrete counterpart of the PVD defined in (1). The method for computing Ω^{dhp}_i is introduced

in the following section.

4.2. *Parallel Computing Method*

The computing of discrete hyperbolic periodic Voronoi diagram (or DH-PVD) follows the same idea as the continuous case (Section 3), i.e. computing the intersection of the center domain and the non-periodic discrete hyperbolic Voronoi diagram. The site-copy-reduction schema introduced in Section 3 is also used. So the computation procedure of DH-PVD consists of 3 steps: (1) in the first step a discrete hyperbolic Voronoi diagram is computed on the center domain by mesh flooding algorithm, using the sites in the center domain only; (2) in the second step we make necessary site copies into the neighbor domains, and use them to update the identification of nearest sites for the boundary triangles of the center domain; (3) in the third step we continue mesh flooding in the center domain using the updated results from Step 2. Eventually we can get DH-PVD after completing Step 3. Figure 6 shows the pipeline of these three steps for computing DH-PVD on the center domain of the genus-2 Amphora surface.

4.2.1. *Mesh Flooding*

In this section a parallel algorithm, mesh flooding, for computing discrete hyperbolic Voronoi diagram is introduced. Assume that the number of mesh triangles is much more than the number of sites, and each triangle contains at most one site. In the beginning the triangles that are nearest to the sites are marked as initial Voronoi cells. Then the Voronoi cells grow concurrently like "fire-spreading" until they reach the actual Voronoi cell boundaries.

We use the same notations as in Definition 3. Let $Nbr(f) = \{f' \mid f \text{ and } f' \text{ share one edge}\}$ be the set of neighbor faces of f. Note that each face has at most three neighbors. Let $C \subseteq F$ be the current working set of triangles that are at the front of "fire-spreading", and $N \subseteq F$ be the next working set that are neighbors of C and will be used for the next round of spreading. C is initialized as:

$$C = \{f \in F \mid f \text{ is the nearest face to site } \mathbf{s}_i, \exists\, \mathbf{s}_i \in S\}. \tag{7}$$

The discrete hyperbolic Voronoi diagram can be represented as a map $v : F \to S$, which is initialized as:

$$v(f) = \begin{cases} \mathbf{s}_i, & \text{if } f \text{ is the nearest face to } \mathbf{s}_i\ ; \\ null, & \text{otherwise.} \end{cases} \tag{8}$$

After initializing v and C, the mesh flooding algorithm described below is executed. When it finishes, v represents the computed discrete hyperbolic Voronoi diagram.

Algorithm 1. Mesh Flooding

Require: C and v are properly initialized

Ensure: v maps each triangle to its nearest site

01. **while** $C \neq \emptyset$ **do**
02. $N \leftarrow \emptyset$
03. **parallelly for each** triangle $f \in C$
04. find the site $\mathbf{s} \in \{v(f') \mid f' \in Nbr(f)\} \cup \{v(f)\}$ that minimizes $d_P(\mathbf{c}_f, \mathbf{s})$, while *null* value in v is ignored.
05. $v(f) \leftarrow \mathbf{s}$
06. **for each** neighbor triangle $f' \in Nbr(f)$
07. put f' into N if $v(f') = null$ or $d_P(\mathbf{c}_{f'}, \mathbf{s}) < d_P(\mathbf{c}_{f'}, v(f'))$
08. **end for**
09. **end for (parallelly)**
10. exchange C and N
11. **end while**

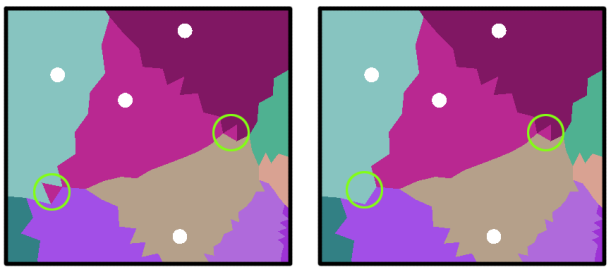


Fig. 7. Left: the accurate discrete Voronoi diagram; Right: the mesh flooding result. Two disconnected triangles are marked by circles.

In Definition 3, each mesh triangle is represented by its centroid and the Voronoi cell is a set of triangles. Such Voronoi cell cannot be guaranteed to be *well-connected*, which means any triangle share at least an edge with others in the same Voronoi cell. Actually some triangles in a Voronoi cell may only connect via one single vertex, or even disconnected. We call a discrete hyperbolic Voronoi diagram *perfect* if all of its Voronoi cells are well-connected. In a non-perfect case, the triangle that does not share at least one edge with its Voronoi cell is called a *disconnected triangle* (see Figure 7). About the correctness of the mesh flooding algorithm, we have the following theorem.

Theorem 2 *Suppose C and v are initialized as in (7) and (8). Let v' represent the discrete hyperbolic Voronoi diagram on mesh $\bar{M} = (\bar{F}, \bar{V})$ induced by site set $\{v(f) \mid f \in C\}$. If v' is perfect, then Algorithm 1 can guarantee $v = v'$.*

This theorem is proved in Appendix B, which means for a perfect discrete hyperbolic Voronoi diagram the algorithm can get a correct result. If the Voronoi diagram is not prefect, there is no guarantee about the correctness of the result on disconnected triangles (see Figure 7). But in practice the ratio of disconnected triangles is rather small. The results in Section 6.1 show that our mesh flooding keeps a low error rate, and the error rate decreases significantly along with CVT iterations.

Following the notations in Definition 4, the center domain of UCS $\bar{M}$ is denoted by $\bar{M}_0 = (\bar{F}_0, \bar{V}_0)$, and its neighbor domains are denoted by $\bar{M}_i = (\bar{F}_i, \bar{V}_i)$, $i = 1, ..., h$, $h = 16g^2 - 8g$. Our goal is to compute the DH-PVD on $\bar{M}$. In Step 1 the mesh flooding algorithm described above is applied to compute a discrete hyperbolic Voronoi diagram on the center domain $\bar{M}_0$. The sites used here are the preim-

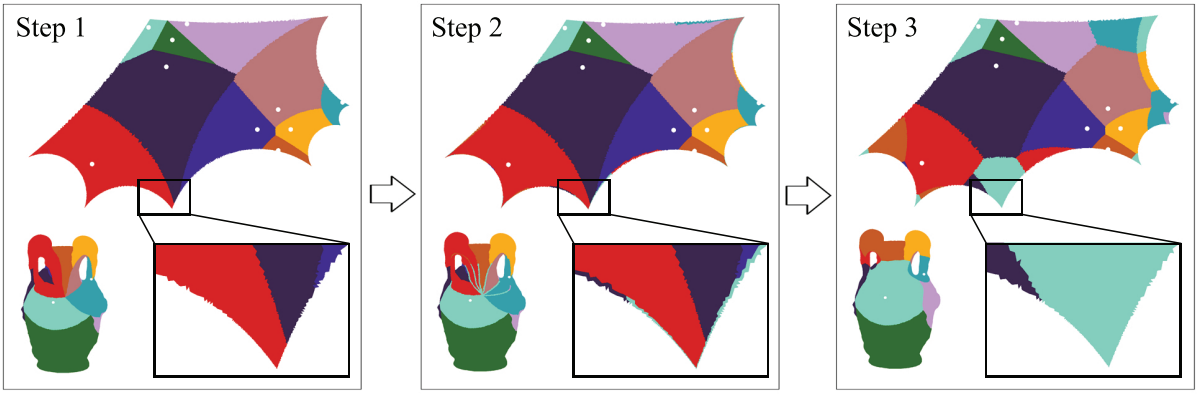


Fig. 6. The pipeline of our parallel computing method for DH-PVD.

ages of S in the center domain under the covering map π_0, denoted by $\bar{S}_0$:

$$\bar{S}_0 = \{\bar{\mathbf{s}}_i \in \bar{M}_0 \mid \pi_0(\bar{\mathbf{s}}_i) = \mathbf{s}_i, \mathbf{s}_i \in S\}.$$

In the following step the necessary site copies in neighbor domains are added to make the Voronoi diagram periodic.

4.2.2. *Updating Boundary Triangles*

In Step 2, the sites that need to be copied, i.e. boundary sites, are identified first. Secondly, the site copies with possibility of having their Voronoi cells intersecting with center domain are selected using the two rules in Section 3. Finally the selected site copies are used to update the v value for all boundary triangles of the center domain.

Let us denote $B = \{f \in \bar{F}_0 \mid f$ has a vertex on boundary of $\bar{M}_0\}$ as the set of boundary triangles of the center domain. Define A_{bdy} to be a list of length $|S|$ indicating if a site is on the boundary of center domain or not: each item will store the site if the corresponding site is on the boundary, otherwise it will store *null*. Note that S_{bdy} and S_{copy} are the two lists storing those boundary sites and their copies in neighbor domains. Let A_{nec} be a list of length $|S_{copy}|$ indicating if a site copy is killed or not: each item will store the site copy if the corresponding site copy is not killed, otherwise it will store *null*. Let S_{nec} be the list of necessary site copies in neighbor domains. Note that S_{bdy} can be computed from A_{bdy} by removing items with *null* values. We call such operation "compacting", which can be computed parallelly with time complexity $O(\log(k))$ where k is the size of original list. Similarly S_{nec} can be computed from A_{nec} by parallel compacting. The reason that we introduce A_{bdy} and A_{nec} is that we want to compute each site parallelly, and each site can independently write its own result into the corresponding item in these lists.

Let C be the current working set as in Step 1. The algorithm for making necessary site copies and updating v at boundary triangles in Step 2 is described below.

Algorithm 2. Updating Boundary Triangles

Require: v is the output of Step 1

Ensure: v is DH-PVD for the boundary triangles in B, and C contains the neighbors of the updated triangles

01. $C \leftarrow \emptyset$, and initialize A_{bdy} with all *null*s
02. **parallelly for each** boundary face $f \in B$
03. $\quad A_{bdy}[v(f)] \leftarrow v(f)$
04. **end for (parallelly)**
05. **parallelly compact** the array A_{bdy} into S_{bdy}
06. make the neighbor copies of S_{bdy} into S_{copy} and initialize the array $A_{nec}[\bar{\mathbf{s}}_i] \leftarrow \bar{\mathbf{s}}_i$, $\forall \bar{\mathbf{s}}_i \in S_{copy}$
07. **parallelly for each** site $\bar{\mathbf{s}}_i \in S_{bdy}$
08. $\quad$ **parallelly for each** neighbor domain j
09. $\quad\quad$ **if** $\phi_j(\bar{\mathbf{s}}_i)$ can be killed using Option 2 (2nd Rule)
10. $\quad\quad\quad A_{nec}[\phi_j(\bar{\mathbf{s}}_i)] \leftarrow null$
11. $\quad\quad$ **end if**
12. $\quad$ **end for (parallelly)**
13. **end for (parallelly)**
14. **parallelly compact** the array A_{nec} into S_{nec}
15. **parallelly for each** boundary face $f \in B$
16. $\quad$ find site $\bar{\mathbf{s}}$ in $\{v(f)\} \cup S_{nec}$ that minimize $d_P(\mathbf{c}_f, \bar{\mathbf{s}})$
17. $\quad$ **if** $v(f) \neq \bar{\mathbf{s}}$
18. $\quad\quad v(f) \leftarrow \bar{\mathbf{s}}$
19. $\quad\quad$ **for each** neighbor triangle $f' \in Nbr(f)$
20. $\quad\quad\quad$ put f' into C if $d_P(\mathbf{c}_{f'}, \bar{\mathbf{s}}) < d_P(\mathbf{c}_{f'}, v(f'))$
21. $\quad\quad$ **end for**
22. $\quad$ **end if**
23. **end for (parallelly)**

Algorithm 2 consists of two phases. In the first phase the necessary site copies are selected using the first rule (lines 2-5) and second rule (lines 6-14) described in Section 3. The second phase (lines 15-23) updates the boundary triangles with the reduced set of site copies.

In Step 3 the mesh flooding algorithm is applied again using the updated values of v and C from Step 2. From the above analysis we know that the result of Step 3 is equal to that of running the mesh flooding algorithm with both the sites on center domain and their copies in neighbor domains, which is exactly the DH-PVD.

4.3. *Complexity Analysis of Parallel Algorithms*

For Algorithm 1, it is easy to see that its flooding time is related to the maximal radius of the Voronoi cell. Suppose

$|F| = n$, $|S| = m$, then the average time complexity of this algorithm is $O(\sqrt{\frac{n}{m}})$, if there are enough GPU cores to achieve full parallelism. The total work of this algorithm is $O(n)$.

For Algorithm 2, without applying the second rule, the upper bound of $|S_{nec}|$ is $\sqrt{m} \cdot (16g^2 - 8g)$. For applying the first rule: lines 2-4 run in parallel, using constant time and having $O(\sqrt{n})$ total work; and line 5 runs in $O(\log(m))$ time, with $O(m \log(m))$ total work. For applying the second rule: lines 7-13 run in parallel, using $O(g)$ time (pairwise testing) and having $O(\sqrt{mg^3})$ total work; and line 14 runs in $O(\log(\sqrt{mg^2}))$ time, with $O(\sqrt{mg^2} \log(\sqrt{mg^2}))$ total work. For updating the boundary triangles (lines 15-23), the time complexity is determined by line 16 – finding the nearest site, which can be achieved in $O(\log(\sqrt{mg^2}))$ time. Thus the total time complexity of Algorithm 2 is $O(\log(m) + g + \log(\sqrt{mg^2}))$, by assuming full parallelism.

5. Discrete Hyperbolic CVT

From the above mentioned results of DH-PVD, we can have the *discrete hyperbolic (periodic) CVT*, by using the Lloyd's algorithm [12]. It has been proved that the hyperbolic CVT minimizes the CVT energy in 2D hyperbolic space in continuous case [18]. The problem we need to discuss is how to extend the hyperbolic CVT energy to the discrete case, and what is the relationship between the defined energy and the discrete hyperbolic CVT.

The *discrete hyperbolic CVT energy* can be defined by changing the region of integration into the discrete one $\Omega_d = \cup \Omega_i^{dh}$ (Definition 3):

$$\begin{aligned} E_d(S, \Omega_d) &= \sum_i \int_{\Omega_i^{dh}} \rho(\mathbf{p}) \cosh(d_M(\mathbf{p}, \mathbf{s}_i))\, d\mathbf{p} \\ &= \sum_i \sum_{f_j \in \Omega_i^{dh}} \int_{f_j} \rho(\mathbf{p}) \cosh(d_M(\mathbf{p}, \mathbf{s}_i))\, d\mathbf{p}. \end{aligned} \tag{9}$$

Note the point $\mathbf{p}$ and site $\mathbf{s}_i$ in (9) are both on Minkowski model. The integral now is defined on each triangle f_j of the mesh, and it can be further simplified if we represent the whole triangle by its centroid $\mathbf{c}_{f_j}$ (Definition 2):

$$\int_{f_j} \rho(\mathbf{p}) \cosh(d_M(\mathbf{p}, \mathbf{s}_i))\, d\mathbf{p} = \eta_{f_j} \langle \mathbf{s}_i, \mathbf{c}_{f_j} \rangle_M, \tag{10}$$

where η_{f_j} is the Minkowski norm: $\eta_{f_j} = \| \int_{f_j} \rho(\mathbf{p}) \mathbf{p} d\mathbf{p} \|_M$, as defined in (3) when computing the centroid of each triangle. Equation (10) is proved in Appendix C.

Now the discrete hyperbolic CVT energy can be written as:

$$E_d(S, \Omega_d) = \sum_i \sum_{f_j \in \Omega_i^{dh}} \eta_{f_j} \langle \mathbf{s}_i, \mathbf{c}_{f_j} \rangle_M. \tag{11}$$

Notice that in (11) the energy E_d is only related to the sites and the centroids of triangles, which makes our discrete version of PVD (in Definition 4) meaningful. Since we are representing each triangle by its centroid when computing DH-PVD, it is easy to see that: if the sites are fixed, the discrete hyperbolic CVT energy is minimized when the tessellation is DH-PVD. Following Rong et al.'s work [18], it is straightforward to prove that in each iteration of Lloyd's algorithm, the discrete hyperbolic CVT energy will decrease monotonically when all the discrete Voronoi diagrams are perfect. Figure 8 shows the decreasing discrete CVT energy of the genus-3 Sculpture surface.

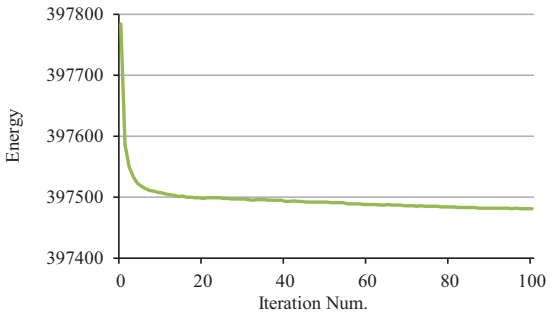


Fig. 8. The discrete CVT energy of the genus-3 Sculpture surface running with Lloyd's algorithm.

6. Implementation and Results

In this paper, we implement the algorithms of DH-PVD with NVIDIA CUDA 4.0. To compute the centroids of discrete Voronoi cells, we use the regional reduction algorithm implemented with NVIDIA Cg 2.0, as suggested by Rong et al. [19]. The hardware platform for running the experiments consists of Intel® Core™2 X6800 CPU (2.93 GHz), 4G DDR2 RAM and NVIDIA® GeForce® GTX 580 GPU with 1.5G GDDR5 video memory.

A restriction that is common for all cluster-based approaches [22] is: the number of sites should be smaller than the number of triangles of the mesh. We can increase the number of triangles by subdividing the input mesh with either linear or Loop subdivision schemes. The initialization of site locations are controlled in the way so that any two sites will not "occupy" the same triangle.

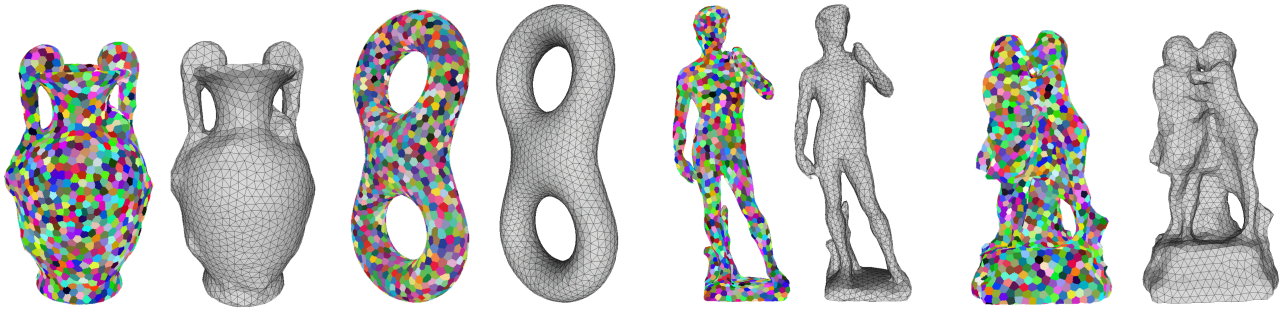
In this section some experimental results are given to illustrate the correctness and effectiveness of our algorithms. Table 1 shows the statistics of the models used in our experiments (see Figure 9). "#Faces Sub." denotes the number of faces after subdivision in pre-processing.

6.1. *Error Rate of Mesh Flooding*

In order to evaluate the correctness of our Mesh Flooding algorithm, a brute force method for computing the accurate discrete Voronoi diagram is also implemented. The *error rate* of our Mesh Flooding result $v : F \to S$ is defined as the ratio:

$$\frac{|\{f \mid v(f) \neq v'(f), \forall f \in F\}|}{|F|}, \tag{12}$$

where $v' : F \to S$ is the accurate discrete Voronoi diagram. Figure 10 shows the error rate of mesh flooding on four different models, with different number of sites sampled



(a) Amphora (b) Eight (c) David (d) Sculpture

Fig. 9. Our CVT results and their dual triangular meshes.

Table 1
Statistics of Models in our Experiment.

Model Name	#Vertices	#Faces	Genus	#Faces Sub.
Amphora	7478	14960	2	239360
Eight	3498	7000	2	448000
David	3002	6013	3	384768
Sculpture	3100	6209	3	397312

randomly. We can note from Figure 11 that the error rate decreases along with Lloyd iterations as the sites become distributed uniformly.

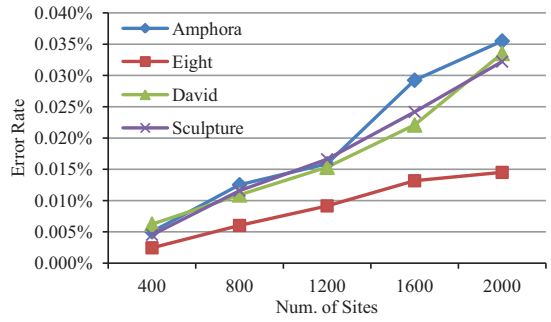


Fig. 10. The error rate of mesh flooding on four different models, with different number of sites sampled randomly.

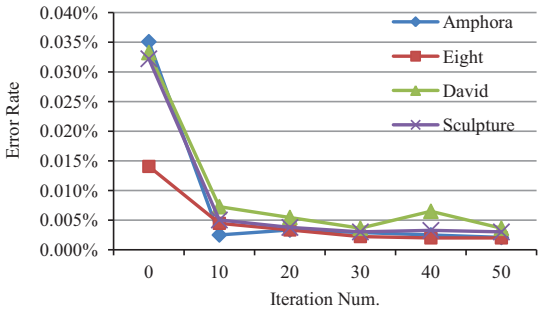


Fig. 11. The error rate of mesh flooding after Lloyd's iterations, with 2000 sites on four different models.

6.2. *Running Time*

We measure the time performance for our mesh flooding algorithms with various options of site-copy-reduction: (1) No reduction; (2) 1st rule only; (3) 2nd rule with Option 1; (4) 2nd rule with Option 2. As can be seen in Figure 12, when the number of sites is small, the Algorithm 2 (using the 2nd rule with Option 2) spends more time on testing the reduction than flooding with full site copies. However, when the number of sites becomes large (> 200), the advantage of site-copy-reduction becomes evident. In this case, the running time of mesh flooding even decrease slightly because the flooding radius of each Voronoi cell decreases. We can also see that the 2nd rule with Option 1 takes too much time on computing site-copy-reduction, although it can achieve very low numbers of site copies. Thus we choose the Option 2 for our Algorithm 2.

Compared with Rong et al.'s continuous algorithm [18], our GPU-based algorithms for discrete hyperbolic CVT is over two orders of magnitude faster for genus-2 surfaces, and over three orders of magnitude faster for genus-3 surfaces. We also compare our performance with Yan et al.'s "extrinsic" approach [24]. Figure 13 shows the comparison of time performance for these three methods.

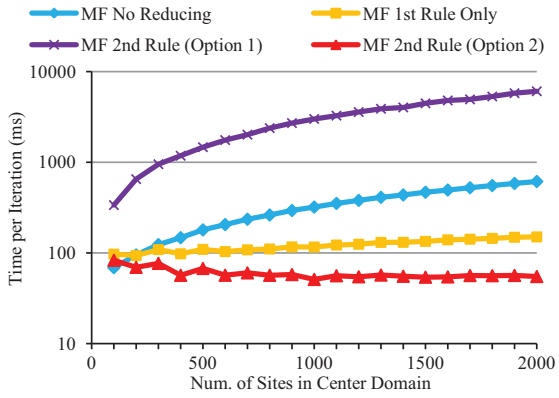


Fig. 12. The running time per iteration comparing the mesh flooding algorithms with different options of site-copy-reduction. Note that the height in vertical axis is scaled by logarithm.

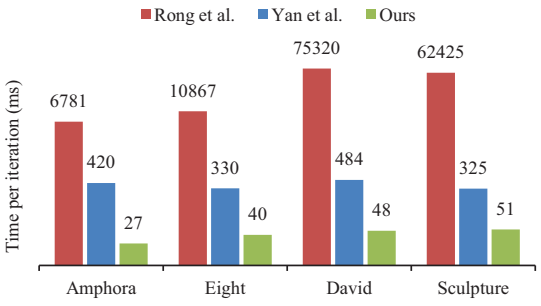


Fig. 13. The running time per iteration comparing our method with Rong et al.'s continuous algorithm [18] and Yan et al.'s "extrinsic" approach [24]. We use 2000 sites in these experiments. Note that the heights of the bars are scaled by logarithm.

6.3. *CVT Quality*

Besides the dramatic improvements on time performance, our GPU-based discrete algorithm can produce high-quality CVT results. Table 2 gives the comparison of quality with those of Rong et al. [18] and Yan et al. [24], given the same initial positions of 2000 sites, and running over the same 100 Lloyd's iterations. Note that the method of Yan et al. [24] is an "extrinsic" approach, and may produce non-manifold vertices for some surfaces, as shown by Rong et al. [18]. In Table 2, we do not include those surfaces that can potentially cause problems for extrinsic approaches.

We measure the CVT quality by the distance d_i of a site to its nearest neighbors: $d_i = \min_{j \neq i} d(\mathbf{s}_i, \mathbf{s}_j)$, where the distance $d(\cdot, \cdot)$ is measured in 3D Euclidean space. The smaller variance of d_i from all the sites ($Var(d_i)$ in Table 2) means better uniformity. We also measure the quality of its dual triangle mesh by the criteria used by Yan et al. [24]: $Q_f = \frac{6}{\sqrt{3}} \frac{A_f}{r_f e_f}$, where A_f is the area of the triangle f, r_f the in-radius of f and e_f the longest edge length of f. Q_{avg} is the average quality of the triangle mesh. We measure θ_{min} as the average of minimal angles of all triangles. We can see that our quality is very close to those of continuous methods [18,24].

Table 2
Comparison of CVT and Dual Mesh Quality

		$Var(d_i)$	Q_{avg}	θ_{min}			$Var(d_i)$	Q_{avg}	θ_{min}
Amphora	Init	0.000122	0.7233	38.81	David	Init	0.000070	0.7145	38.24
	Ours	0.000023	0.8752	50.11		Ours	0.000026	0.8476	47.81
	Rong	0.000017	0.8977	51.77		Rong	0.000022	0.8684	49.35
	Yan	0.000016	0.9140	53.06		Yan	0.000015	0.9057	52.38
Eight	Init	0.000133	0.7145	38.38	Sculpture	Init	0.000098	0.7255	39.07
	Ours	0.000015	0.8762	50.17		Ours	0.000018	0.8880	51.09
	Rong	0.000009	0.8949	51.54		Rong	0.000038	0.8925	51.43
	Yan	0.000008	0.9157	53.18		Yan	0.000010	0.9067	52.45

6.4. *Limitations and Future Work*

First we want to discuss the two effects of poor input triangulation (i.e. triangulation with skinny triangles) to our method. 1) Our CVT schema is based on hyperbolic parameterization [9], and poorly triangulated input meshes may cause the parameterization to fail. Specifically, the parameterization procedure uses Newton's method to compute the hyperbolic uniformization metric for surfaces, which can be written as a linear equation of discrete Lapalce-Beltrami operator. Skinny triangles will increase the condition number of the linear equation, so poor triangulations do affect the convergence speed and numerical stability of the parameterization method. 2) The poor triangulation may also lead to bad VD/CVT quality for our mesh flooding algorithm since the Voronoi cells cannot be well shaped. This disadvantage can be alleviated by simply subdividing the input mesh (Figure 14). Although the significant increase of the number of triangles will slow down the computation speed, our method is still 5 times faster than Yan et al.'s [24] method for the result showing in Figure 14. We will continue to work on the first issue (hyperbolic parameterization with skinny triangles) in the future.

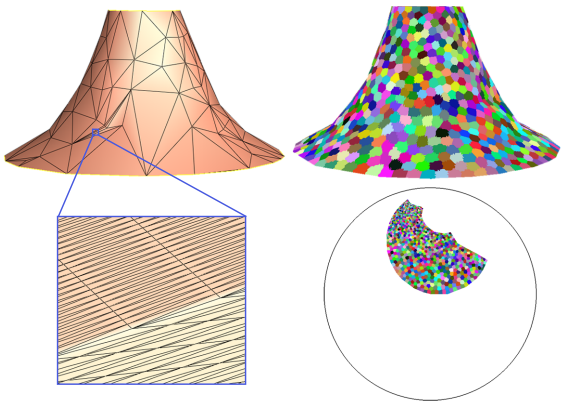


Fig. 14. Top-left: a part of tractricoid with poor triangulation as input mesh; Bottom-left: a zoom view of skinny triangles after subdividing the input mesh (the triangle number is increased by 1024 times); Top-right: the CVT result on the subdivided model; Bottom-right: the corresponding CVT on hyperbolic plane. We use tractricoid because it can be isometrically embedded on hyperbolic plane without computing the uniformization metric.

In Table 2 we can see our result is a little worse than Rong et al.'s [18] due to the approximation brought by discretization. However, we also notice that the CVT and dual mesh quality of both intrinsic methods is a little worse than extrinsic method. Both Rong et al.'s [18] and ours use the square of conformal factor as the density function when computing CVT, expecting the density function to compensate for the distortion of Voronoi cells caused by conformal parameterization. In the future we would like to systematically analyze the influence of conformal distortion to the quality of CVT on surfaces.

Although our current site-copy-reduction schema can significantly reduce the number of site copies down to a num-

ber that is very close to optimum, we are still not sure if there is any efficient algorithm that can directly achieve optimal reduction. We will continue to work on these problems in the future.

7. Conclusion

In this paper, we present a GPU-based parallel algorithm for computing discrete hyperbolic CVT, equipped with an efficient site-copy-reduction schema. This makes our algorithm efficient to compute the constrained CVT on arbitrary high-genus surfaces via their hyperbolic conformal parameterization. Experiments show that our new method is at least two orders of magnitude faster than the previous CPU-based continuous algorithm, while still maintaining the high quality of the resulting CVT on surfaces.

Acknowledgements

The authors would like to thank the anonymous reviewers and Dr. Guodong Rong for their valuable comments and suggestions to improve the paper. Liang Shuai and Xiaohu Guo are partially supported by the National Science Foundation (NSF) under Grant Nos. CCF-0727098, CNS-1012975, and IIS-1149737. Miao Jin is partially supported by NSF under Grant No. CCF-1054996.

References

[1] P. Alliez, E. C. de Verdière, O. Devillers, and M. Isenburg. Centroidal Voronoi diagrams for isotropic surface remeshing. *Graphical Models*, 67(3):204–231, May 2005.

[2] M. Caroli and M. Teillaud. Computing 3D periodic triangulations. In *Proceedings of the 17th European Symposium on Algorithms*, pages 37–48, 2009.

[3] Q. Du, M. Emelianenko, and L. Ju. Convergence of the Lloyd algorithm for computing centroidal Voronoi tessellations. *SIAM Journal on Numerical Analysis*, 44(1):102–119, January 2006.

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

[5] Q. Du, M. D. Gunzburger, and L. Ju. Constrained centroidal Voronoi tessellations for surfaces. *SIAM Journal on Scientific Computing*, 24(5):1488–1506, 2003.

[6] Q. Du and L. Ju. Finite volume methods on spheres and spherical centroidal Voronoi meshes. *SIAM Journal on Numerical Analysis*, 43(4):1673–1692, 2005.

[7] G. Fejes Tóth. A stability criterion to the moment theorem. *Studia Scientiarum Mathematicarum Hungarica*, 38(1-4):209–224, 2001.

[8] A. Gersho. Asymptotically optimal block quantization. *IEEE Transactions on Information Theory*, 25(4):373–380, 1979.

[9] M. Jin, J. Kim, and X. D. Gu. Discrete surface ricci flow: Theory and applications. *Mathematics of Surfaces XII*, 4647:209–232, 2007.

[10] M. Jin, F. Luo, and X. Gu. Computing surface hyperbolic structure and real projective structure. In *Proceedings of the 2006 ACM Symposium on Solid and Physical Modeling*, pages 105–116, 2006.

[11] Y. Liu, W. Wang, B. Lévy, F. Sun, D.-M. Yan, L. Lu, and C. Yang. On centroidal Voronoi tessellation — energy smoothness and fast computation. *ACM Transactions on Graphics*, 28(4):1–17, 2009.

[12] S. P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[13] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[14] J. R. Munkres. *Elements of Algebraic Topology*. Westview Press, 1996.

[15] F. Nielsen and R. Nock. Hyperbolic Voronoi diagrams made easy. In *Proceedings of the 2010 International Conference on Computational Science and Its Applications*, pages 74–80, 2010.

[16] A. Ramsay and R. Richtmyer. *Introduction to Hyperbolic Geometry*. Springer-Verlag, New York, 1995.

[17] G. Rong, M. Jin, and X. Guo. Hyperbolic centroidal Voronoi tessellation. In *Proceedings of the 14th ACM Symposium on Solid and Physical Modeling*, pages 117–126, 2010.

[18] G. Rong, M. Jin, L. Shuai, and X. Guo. Centroidal Voronoi tessellation in universal covering space of manifold surfaces. *Computer Aided Geometric Design*, 28(8):475–496, November 2011.

[19] G. Rong, Y. Liu, W. Wang, X. Yin, X. Gu, and X. Guo. GPU-assisted computation of centroidal Voronoi tessellation. *IEEE Transactions on Visualization and Computer Graphics*, 17(3):345 –356, March 2011.

[20] G. Rong and T.-S. Tan. Jump flooding in GPU with applications to Voronoi diagram and distance transform. In *Proceedings of the ACM Symposium on Interactive 3D Graphics and Games*, pages 109–116, 2006.

[21] V. Surazhsky, P. Alliez, and C. Gotsman. Isotropic remeshing of surfaces: A local parameterization approach. In *Proceedings of the 12th International Meshing Roundtable*, pages 215–224, 2003.

[22] S. Valette, J.-M. Chassery, and R. Prost. Generic remeshing of 3D triangular meshes with metric-dependent discrete Voronoi diagrams. *IEEE Transactions on Visualization and Computer Graphics*, 14(2):369 –381, March-April 2008.

[23] C. N. Vasconcelos, A. Sá, P. C. Carvalho, and M. Gattass. Lloyd's algorithm on GPU. In *Proceedings of the 4th International Symposium on Visual Computing*, pages 953–964, 2008.

[24] D.-M. Yan, B. Lévy, Y. Liu, F. Sun, and W. Wang. Isotropic remeshing with fast and exact computation of restricted Voronoi diagram. *Computer Graphics Forum*, 28(5):1445–1454, 2009.

[25] D.-M. Yan, K. Wang, B. Lévy, and L. Alonso. Computing 2D periodic centroidal Voronoi tessellation. In *Proceedings of the 2011 International Symposium on Voronoi Diagrams in Science and Engineering*, pages 177–184, 2011.

Appendix A. Proof of Theorem 1

To prove Theorem 1, the following lemma is needed:

Lemma 1 *In the hyperbolic plane, given a triangle and two points* $\boldsymbol{p}$ *and* $\boldsymbol{q}$ *outside the triangle, if* $\boldsymbol{p}$ *has shorter distances to all vertices of the triangle than* $\boldsymbol{q}$*, then* $\boldsymbol{p}$ *have shorter distances to all points inside the triangle than* $\boldsymbol{q}$*.*

Proof This lemma can be proved using the Klein disk. According to Nielsen and Nock [15], the bisector between $\mathbf{p}$ and $\mathbf{q}$ is a straight line which partitions the whole Klein disk into two parts. All the vertices of the triangle are in the half space of point $\mathbf{p}$. Since all the edges of the triangle are straight line segments in Klein disk, they are all in the

half space of $\mathbf{p}$. Thus the whole triangle is in the half space of $\mathbf{p}$. □

Then Theorem 1 is proved as follows.

Proof Since any polygon can be triangulated in hyperbolic plane [16], the hyperbolic polygon can be triangulated using its corners. With Lemma 1, this theorem is proved automatically. □

Appendix B. Proof of Theorem 2

For the convenience of proving Theorem 2, the following definition is given first:

Definition 5 *A path $\mathcal{P}(f_a, f_b)$ on triangular mesh $\bar{M} = (\bar{F}, \bar{V})$ from face $f_a \in \bar{F}$ to face $f_b \in \bar{F}$ is a sequence of faces $f_1 f_2 ... f_n$, where $f_1 = f_a$, $f_n = f_b$, $f_i \in \bar{F}, i = 1, ..., n$, and any two consecutive faces in the sequence share one edge. The length of $\mathcal{P}$ is $n - 1$.*

Theorem 2 is proved as follows.

Proof Given the definition of the length of a path, for all the possible paths $\mathcal{P}(f_a, f_b)$ on $\bar{M}$ that connect the two faces f_a and f_b in the same discrete Voronoi cell $v'(f_a) = v'(f_b) = s$, let us denote the shortest path as $\mathcal{P}^s_{min}$. Let us consider $\forall f \in \bar{F}$, and suppose $v'(f) = s$, the nearest face to s is f_s, and the length of the shortest path $|\mathcal{P}^s_{min}(f_s, f)| = n$. If we can prove that $v(f) = s$ after Algorithm 1 is executed, then Theorem 2 is proved.

Let us prove the above statement by mathematical induction on the length of shortest path $|\mathcal{P}^s_{min}(f_s, f)|$.

Base Case:

Prove the statement holds for $|\mathcal{P}^s_{min}(f_s, f)| = 1$.
$\because f_s$ is the nearest face to s,
$\therefore v(f_s) = s$ due to the initialization of v in (8).
$\because |\mathcal{P}^s_{min}(f_s, f)| = 1$,
$\therefore f$ is a neighbor of f_s on mesh $\bar{M}$,
$\therefore v(f) = v(f_s) = s$ due to lines 4 and 5 of Algorithm 1.
$v(f)$ will not change any more due to the fact $v'(f) = s$.
$\therefore v(f) = s$ after Algorithm 1 is executed.

Induction Hypothesis:

Suppose the statement holds for $|\mathcal{P}^s_{min}(f_s, f)| = n$.

Inductive Case:

Prove the statement holds for $|\mathcal{P}^s_{min}(f_s, f)| = n + 1$.
$\because |\mathcal{P}^s_{min}(f_s, f)| = n + 1$,
$\therefore \exists f'$ be a neighbor of f s.t. $|\mathcal{P}^s_{min}(f_s, f')| = n$ and $v'(f') = s$.
By Induction Hypothesis, after Algorithm 1 is executed we have $v(f') = s$.
$\because v(f') = null$ initially,
$\therefore$ there must be a loop iteration assigning s to $v(f')$.
$\because f$ is neighbor of f' and $v'(f) = s$,
$\therefore f$ will be added to the next working set N by line 8 of Algorithm 1 in the same loop iteration, and in next loop iteration $v(f)$ will be set to s by line 17.
$v(f)$ will not change any more due to the fact $v'(f) = s$.
$\therefore v(f) = s$ after Algorithm 1 is executed. □

Appendix C. Proof of Equation (10)

Proof Consider the hyperbolic triangle $\triangle_{\mathbf{c}_{f_j}\mathbf{s}_i\mathbf{p}}$ formed by vertices $\mathbf{c}_{f_j}$, $\mathbf{s}_i$, and $\mathbf{p}$, where $\mathbf{p} \in f_j$. By the hyperbolic law of cosines [16], we have:

$$\begin{aligned}\cosh(d_M(\mathbf{p}, \mathbf{s}_i)) = &\cosh(d_M(\mathbf{p}, \mathbf{c}_{f_j}))\cosh(d_M(\mathbf{s}_i, \mathbf{c}_{f_j})) \\ &- \sinh(d_M(\mathbf{p}, \mathbf{c}_{f_j}))\sinh(d_M(\mathbf{s}_i, \mathbf{c}_{f_j}))\cos\alpha,\end{aligned}$$

where α is the angle of the triangle $\triangle_{\mathbf{c}_{f_j}\mathbf{s}_i\mathbf{p}}$ at corner $\mathbf{c}_{f_j}$.

Let $A = \int_{f_j} \rho(\mathbf{p}) \cosh(d_M(\mathbf{p}, \mathbf{c}_{f_j})) \cosh(d_M(\mathbf{s}_i, \mathbf{c}_{f_j}))\, d\mathbf{p}$

and $B = \int_{f_j} \rho(\mathbf{p}) \sinh(d_M(\mathbf{p}, \mathbf{c}_{f_j})) \sinh(d_M(\mathbf{s}_i, \mathbf{c}_{f_j}))\cos\alpha\, d\mathbf{p}$,

then the l.h.s. of (10) is: $\int_{f_j} \rho(\mathbf{p}) \cosh(d_M(\mathbf{p}, \mathbf{s}_i))\, d\mathbf{p} = A - B$.

$$\begin{aligned}A &= \int_{f_j} \rho(\mathbf{p}) \cosh(d_M(\mathbf{p}, \mathbf{c}_{f_j})) \cosh(d_M(\mathbf{s}_i, \mathbf{c}_{f_j}))\, d\mathbf{p} \\ &= \langle \mathbf{s}_i, \mathbf{c}_{f_j} \rangle_M \cdot \langle \int_{f_j} \rho(\mathbf{p})\mathbf{p}\, d\mathbf{p}, \mathbf{c}_{f_j} \rangle_M = \eta_{f_j} \langle \mathbf{s}_i, \mathbf{c}_{f_j} \rangle_M, \\ B &= \int_{f_j} \rho(\mathbf{p}) \sinh(d_M(\mathbf{p}, \mathbf{c}_{f_j})) \sinh(d_M(\mathbf{s}_i, \mathbf{c}_{f_j}))\cos\alpha\, d\mathbf{p} \\ &= \sinh(d_M(\mathbf{s}_i, \mathbf{c}_{f_j})) \int_{f_j} \rho(\mathbf{p}) \sinh(d_M(\mathbf{p}, \mathbf{c}_{f_j}))\cos\alpha\, d\mathbf{p}.\end{aligned}$$

B remains the same if we re-interpret the integral on Poincaré disk. Let f'_j, α', $\mathbf{c}'_{f_j}$, $\mathbf{s}'_i$ and $\mathbf{p}'$ be the corresponding triangle/angle/point on Poincaré disk w.r.t. f_j, α, $\mathbf{c}_{f_j}$, $\mathbf{s}_i$ and $\mathbf{p}$. Use $d_P(\cdot, \cdot)$ to denote the hyperbolic distance on Poincaré disk. Without loss of generality, assume $\mathbf{c}'_{f_j}$ is on the origin of Poincaré disk, then the coordinate of $\mathbf{c}_{f_j}$ is (0,0,1) on Minkowski model. Use $\|\cdot\|$ to denote the Euclidean norm. It is easy to prove that:

$$\sinh(d_P(\mathbf{p}', \mathbf{c}'_{f_j})) = \frac{2\,\|\mathbf{p}'\|}{1 - \|\mathbf{p}'\|^2}.$$

Denote the coordinate of $\mathbf{p}'$ as $(x_{\mathbf{p}'}, y_{\mathbf{p}'})$ on Poincaré disk, and the coordinate of $\mathbf{p}$ as $(x_{\mathbf{p}}, y_{\mathbf{p}}, z_{\mathbf{p}})$ on Minkowski model, we have:

$$\begin{aligned}x_{\mathbf{p}} = \frac{2\,x_{\mathbf{p}'}}{1 - x^2_{\mathbf{p}'} - y^2_{\mathbf{p}'}} &= \frac{2\,\|\mathbf{p}'\|\cos\alpha'}{1 - \|\mathbf{p}'\|^2} \\ &= \sinh(d_P(\mathbf{p}', \mathbf{c}'_{f_j}))\cos\alpha' \\ &= \sinh(d_M(\mathbf{p}, \mathbf{c}_{f_j}))\cos\alpha.\end{aligned}$$

So the integral part in B can be re-written as:

$$\begin{aligned}&\int_{f_j} \rho(\mathbf{p}) \sinh(d_M(\mathbf{p}, \mathbf{c}_{f_j}))\cos\alpha\, d\mathbf{p} \\ =&\int_{f_j} \rho(\mathbf{p}) x_{\mathbf{p}}\, d\mathbf{p} = \eta_{f_j}\, x_{\mathbf{c}_{f_j}} = 0 \text{ (per our assumption)},\end{aligned}$$

$$\therefore \int_{f_j} \rho(\mathbf{p}) \cosh(d_M(\mathbf{p}, \mathbf{s}_i))\, d\mathbf{p} = A = \eta_{f_j} \langle \mathbf{s}_i, \mathbf{c}_{f_j} \rangle_M.$$

The equation (10) is proved. □