Computer Aided Geometric Design 104 (2023) 102216

Contents lists available at ScienceDirect

Computer Aided Geometric Design

journal homepage: www.elsevier.com/locate/cagd

Accelerating surface remeshing through GPU-based computation of the restricted tangent face ☆

Yuyou Yao, Jingjing Liu, Wenming Wu, Gaofeng Zhang, Benzhu Xu, Liping Zheng *

School of Computer Science and Information Engineering, Hefei University of Technology, Hefei 230601, China

ARTICLE INFO

Article history:
Received 29 July 2022
Received in revised form 5 May 2023
Accepted 12 May 2023
Available online 18 May 2023

Keywords:
Restricted tangent face
Centroidal Voronoi tessellation
Parallel computation
GPU acceleration
Surface remeshing

ABSTRACT

High-quality mesh surfaces are crucial for geometric processing in a variety of applications. To generate these meshes, polyhedral remeshing techniques truncate Voronoi cells of the original surface and yield precise intersections, but the calculation is complicated. Some methods apply auxiliary points to construct Voronoi diagrams to simplify these techniques, thereby extracting co-planar facets to approximate the original surface. However, extracting these approximate facets from the constructed Voronoi diagram makes it inefficient and non-parallelizable. To this end, we propose an efficient GPU method for manifold surface remeshing, where the restricted tangent face (RTF) is utilized to approximate the original surface. By intersecting the pre-clipped Voronoi cell with the tangent plane, this method directly calculates the RTF of each point without any auxiliary points or traversing Voronoi cells. Moreover, to restrict the movement of points, we introduce a projection method based on the *KNN* strategy, where each point is projected onto the triangular facet in the original surface. Owing to the independence and non-interference of the RTF computation and projection of each point, our method is implemented in parallel on the GPU. Experimental results on various mesh surfaces demonstrate the superior performance of our method in the viability, effectiveness, and efficiency.

1. Introduction

Triangle surface mesh is frequently used for three-dimensional (3D) data representation in geometrical modeling (Decker et al., 2021), architectural modeling (Yang et al., 2018), and scientific visualization (Sullivan and Kaszynski, 2019) due to its effectiveness and simplicity. Previous work produces raw meshes by 3D scanning reconstruction (Chen et al., 2021) and computer vision method (Shazeer et al., 2018), etc. However, low-quality raw meshes are unsuitable for subsequent geometric processing in a variety of applications owing to small angles, short edges, irregular vertices (Liang and Zhang, 2011; Engwirda and Ivers, 2014), etc. As an efficient technique for improving the quality of meshes, surface remeshing has received much attention from researchers in the past two decades.

Previous work (Du et al., 2003; Alliez et al., 2005; Ye et al., 2019) for generating meshes has achieved considerable advancements due to the emergence of polyhedral meshing algorithms (Yan et al., 2013; Yan and Wonka, 2015) based on the centroidal Voronoi tessellation (CVT) (Du et al., 1999). These methods either construct Voronoi diagrams on a surface

☆ Editor: Jin Huang.

* Corresponding author.
E-mail address: zhenglp@hfut.edu.cn (L. Zheng).

https://doi.org/10.1016/j.cagd.2023.102216
0167-8396/

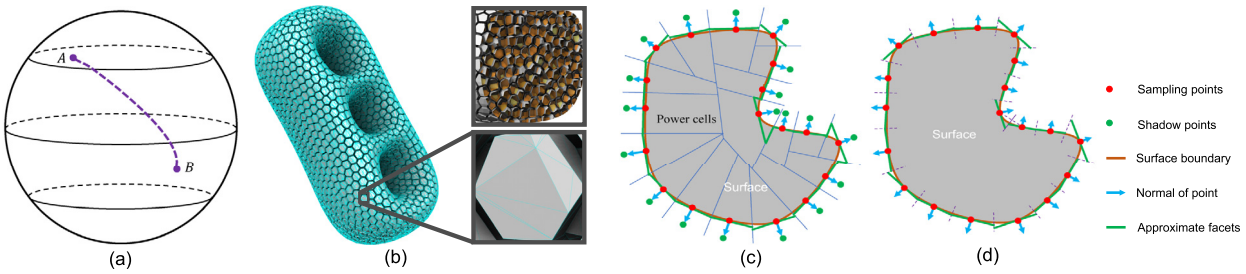


Fig. 1. Exemplification of the geodesic distance, RVD (3D), RPF (2D), and RTF (2D). (a) the geodesic distance between two points on a surface (highlighted with purple line); (b) RVD, intersecting Voronoi cells with the original surface; (c) RPF, extracting the co-facets of the power diagram constructed with shadow points; and (d) RTF, restricted tangent planes without auxiliary points or Voronoi diagram construction. Notably, (b) is generated by the code provided in Lévy and Liu (2010). (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

(e.g., the geodesic CVT (GCVT)) (Fu and Zhou, 2009) or truncate Voronoi cells of the original surface (e.g., the restricted CVT (RCVT), the restricted Voronoi diagram (RVD), etc.) (Yan et al., 2009, 2010), along with CVT optimization techniques (e.g., Lloyd's method or the quasi-Newton method) (Lloyd, 1982; Liu et al., 2009) to produce remeshing surfaces. They can be regarded as exact-CVT-oriented methods as they calculate the geodesic distance or the intersection on the original surface, as shown in Fig. 1(a)(b). However, it is extremely difficult and time-consuming (several seconds per iteration, see Fig. 12) to calculate the precise geodesic distance or the intersection of Voronoi cells with the surface.

Several researchers (Zimmer et al., 2012; Chen et al., 2018) have recently concentrated on yielding high-quality meshes by employing approximate planes, e.g., the restricted power face (RPF) (Xu et al., 2019) in Fig. 1(c). They construct 3D Voronoi diagrams using extra sampling points (e.g., shadow points), filtering the co-planar facets of these boundary Voronoi cells as approximate planes (Abdelkader et al., 2020). These techniques, which are considered approximate-CVT-oriented methods, compute the CVT on these approximation planes during each iteration, producing remeshing results. Compared to Yan et al. (2013), they drastically reduce the necessary calculations and achieve better computational performance owing to the avoidance of truncating Voronoi cells on the original surface. Nevertheless, since they construct some inner Voronoi cells based on these extra sampling points, approximate-CVT-oriented methods consume excessive time and memory. Moreover, filtering these approximate planes by traversing the Voronoi diagrams could be more effective.

Therefore, we concentrate on the limitations of previous work on surface remeshing and take into account parallelization for speedups. In this paper, we propose an efficient GPU method for manifold surface remeshing without truncating Voronoi cells of the original surface. To achieve it, we adapt the concept of approximate CVT and fit the primordial surface using restricted tangent faces (RTF). Unlike existing methods (Xu et al., 2019), we are only interested in these planar facets of randomly initialized sampling points, and auxiliary points (Abdelkader et al., 2020) are unnecessary. Specifically, an efficient method is presented to directly calculate the tangent face of each sampling point instead of traversing the Voronoi diagram (Xu et al., 2019; Abdelkader et al., 2020). This method is well-designed, and the tangent face of each sampling point is calculated independently, which is easily implemented in parallel on the GPU. Thus, the sampling points are relocated to the barycenters of the respective tangent faces. Nevertheless, this may result in optimized points deviating from the original surface. To restrict the movement of sampling points and couple with the tangent face calculations on the GPU, we introduce a projection strategy based on k-nearest-neighbors (*KNN*), where each sampling point is projected onto the triangular facet on the original surface. The tangent facet computation and projection strategy are alternately and iteratively performed to generate the RTF, producing high-quality meshes. We contribute the following:

- a GPU-based method to directly calculate the RTF of sampling points without constructing or traversing Voronoi diagrams with auxiliary points.
- a GPU-based projection strategy via *KNN* method to restrict sampling points moving on the original surface.

The remainder of this paper is organized as follows. Sect. 2 briefly review some CVT-based methods for surface remeshing. The preliminary of the Voronoi diagram, CVT and RPF is provided in Sect. 3. In Sect. 4, we introduce our method for generating high-quality meshes. Sect. 5 presents some remeshing results, and some conclusions are given in Sect. 6.

2. Related work

High-quality meshes are critical in various applications, i.e., geometrical modeling (Decker et al., 2021), scientific visualization (Sullivan and Kaszynski, 2019), etc. Several techniques have been introduced to improve the quality of a given mesh, one of which is surface remeshing (Khan et al., 2022). The raw mesh is fed as input to a series of remeshing algorithms, generating another high-quality mesh. Existing work introduces a mount of remeshing algorithms, i.e., local modification-based remeshing (Wang et al., 2018), segmentation-based remeshing (Khan et al., 2018a), and Delaunay triangulation (DT) based remeshing (Chen and Holst, 2011). An exhaustive review of remeshing techniques could be referred to Khan et al. (2022). Here we only give a brief review of the CVT-based remeshing methods, including two-dimensional (2D) CVT-oriented methods in Sect. 2.1, exact-CVT-oriented methods in Sect. 2.2, and approximate-CVT-oriented methods in Sect. 2.3.

2.1. 2D-CVT-oriented remeshing

The CVT (Du et al., 1999) defines a particular partition of a given domain into several subregions, where each site corresponds to the barycenter of its subregion. Due to its excellent geometric properties, the CVT has been applied for surface remeshing, and a series of algorithms have been proposed (Du et al., 2003; Leung et al., 2015; Herholz et al., 2017; Ye et al., 2019; Yan et al., 2014; Abdelkader et al., 2020).

Alliez et al. (2005) introduce a CVT-based method for isotropic remeshing for triangulated surface meshes. They use a global conformal planar parameterization and apply Lloyd's method in the parametric space using a density function designed to compensate for the area distortion due to flattening. However, the difficulty is that the density function is not specific, which may result in an undesirable remeshing result. Similarly, an adaptive meshing approach for heterogeneous materials is proposed in You et al. (2015), where the CVT and mesh extraction are on the basis of a density function associated with the material distributions. In contrast, this method is only suitable for 2D heterogeneous objects. Besides, two CVT variants for a general planar domain are presented in Khan et al. (2018b), which utilize an extended domain rather than the origin domain for computing the centroids. Still, this method is so limited that it only applies to the planar domain. Additionally, to speed up surface remeshing, Rong et al. (2010) introduce a GPU-assisted construction and optimization for CVT on 2D parameter spaces, which achieves better computation efficiency.

Although the 2D CVT could be easily constructed and optimized, the 2D CVT-based remeshing methods are either challenging to extend to 3D objects (Khan et al., 2018b) or rely on a specific density function (Alliez et al., 2005), resulting in the generated meshes with undesirable quality.

2.2. Exact-CVT-oriented remeshing

Different from the CVT computation in 2D parametric space, some methods try to directly compute the CVT on the 3D original space and then generate the remeshing results. These methods calculate the CVT on a surface (i.e., the GCVT) or truncate the 3D Voronoi diagram of the original surface (i.e., the RVD). Recently, many researchers have worked to design an efficient method for calculating the geodesic distance in the GCVT or the intersection of 3D Voronoi cells with the surface in the RVD.

In view of the GCVT, the crux is calculating the geodesic distance of any two points on the surface. Rong et al. (2011) extend the concept of CVT from 2D space to 3D spherical space and provide a framework to compute the CVT. Zhuang et al. (2014) introduce an interactive method for mesh generation by calculating anisotropic geodesics. To improve the computational efficiency, they present a fast local subdivision method to calculate anisotropic geodesics from existing Euclidean geodesics, producing a comparable mesh for surfaces with sharp features. Wang et al. (2015) present two intrinsic methods to compute the centroid for any geodesic Voronoi diagram (GVD). Though the computational efficiency of the GVD is improved, there is no significant improvement in the generated surface quality. The 2D Possion disk sampling method (Yan and Wonka, 2013) is extended with consideration of geodesic distance by Fu and Zhou (2009). Liu et al. (2016) provide a manifold differential evolution (MDE) method for globally optimizing the energy of GCVT on a surface, yielding meshes of high quality. However, an essential limitation of the GCVT-based remeshing method is that the geodesic distance calculation is too complicated, resulting in inefficient geodesic path computation.

Considering the RVD, the key is the intersection between a Voronoi cell and the primitive surface, as shown in Fig. 1(b). Du et al. (2003) introduce constrained CVT (CCVT) on a surface where each vertex is constrained on the surface and coincides with the barycenter of the Voronoi cell on the surface. However, the local minimum problem of the energy function in this method is still a challenge. Yan et al. (2009) propose another CVT-based method for surface remeshing, called RVD, which computes the exact RVD based on the quasi-Newton method. To eliminate the triangles with small or obtuse angles, they further extend the exact RVD energy function with a penalty function (Yan and Wonka, 2015) to avoid the existence of short edges. However, the limitation of this work is the lacking of termination guarantees for more complex models or sharp features. Additionally, by discretizing the original surface into voxels, Leung et al. (2015) compute the exact Euclidean distance transform (EDT), 3D CVT, and RVD on the GPU, which is highly efficient. Nevertheless, this method is limited by the graphical memory, especially for more complicated models. Recently, a field-aligned method (Du et al., 2018) based on RVD has been introduced for surface remeshing. They minimize an energy function that combines both CVT and the penalty enforced by a six-way rotational symmetry field, but the interpolation is inefficient. Several extensions of the RVD have been presented for surface remeshing in some particular applications, such as the thin-plate models (Wang et al., 2020), signed distance field (Hou et al., 2022) and triangulated surfaces (Sainlot et al., 2017), etc. However, the intersection between Voronoi cells and the primal surface is necessary for these RVD-based remeshing methods, which leads to complex computation and more time consumption.

To summarize, exact-CVT-oriented remeshing methods rely on calculating the geodesic distance between any two points on the surface or the intersection of a Voronoi cell with the surface. Nevertheless, both of these calculations are complex and time-consuming, leading to inefficient remeshing of these methods.

2.3. Approximate-CVT-oriented remeshing

Tangent plane intersection (TPI) (Zimmer et al., 2012) approximates the intersection between Voronoi cells and the original surface, significantly reducing the computation complexity in RVD (Yan et al., 2009). Recently, computing the CVT on an approximation planar has received much attention for surface reconstruction or remeshing.

Chen et al. (2018) extend the CVT to point clouds and generate meshes from high-quality input points. The critical of this method is that the CVT computation is on the point cloud by restricting each Voronoi cell to the underlying surface. However, the underlying surface is approximated by a set of best-fit planes closely related to the number of sampling points. Especially for curved regions, this method produces a poor approximation, resulting in low-quality remeshing results. Xu et al. (2019) introduce the RPF to approximate the intersection in RVD (Yan et al., 2009) to produce meshes. They define a set of shadow points outside the surface boundary, and each is assigned a specific weight. Based on the sampling points and shadow points, they constructed a power diagram from which the RPF of each sampling point could be extracted. However, this method is affected by the location of shadow points, which may cause low-quality or failure remeshing results. Besides, extracting the RPF from the power diagram requires traversing all vertices, which is too complicated and unparalleled. Recently, a robust remeshing approach called VoroCrust (Abdelkader et al., 2020) introduces a placement strategy of points for surface remeshing without Voronoi clipping. The co-planar facets generated by the points on different sides of the boundary are utilized as approximation planes. Nevertheless, extracting the co-planar from the constructed Voronoi diagram, similar to Xu et al. (2019), requires more time and cannot be parallelized. Moreover, this method fails to eliminate short edges and hole filling, affecting the quality of meshes.

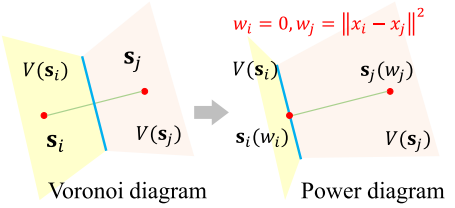
In summary, existing methods employ auxiliary points and sampling points, to construct Voronoi/power diagrams, from which the co-planar facets are extracted as the approximate planes (Xu et al., 2019; Abdelkader et al., 2020). However, this method is complicated, time-consuming, and non-parallelizable. Others take high-quality point clouds without outliers or noises as inputs to generate meshes (Chen et al., 2018). However, this suffers from the number of sampling points and may generate low-quality meshes, especially for those with curved regions, as shown in Fig. 11. With this regard, we borrow the idea of approximate CVT (Xu et al., 2019) and introduce an efficient GPU method for generating high-quality meshes with truncating Voronoi cells, where the RTF is utilized to fit the original surface. To be specific, we directly calculate a CVT on the RTF and optimize the relevant sampling point to its barycenter. Moreover, we introduce a *KNN* based projection strategy to project each sampling point onto the triangular facet in the original surface instead of a best-fit plane affected by the number of sampling points (Chen et al., 2018). Owing to the independence of the RTF computation and projection of each sampling point, our method is implemented in parallel on the GPU.

3. Preliminary

In this section, we provide an overview of the power diagram, CVT and RPF in sequence.

3.1. Voronoi diagram & power diagram

The Voronoi diagram, also called "Voronoi tessellation", defines a spatial subdivision of a given N-dimensional domain $\Omega \subset \mathbb{R}^N$ into several subregions. Given a set $\mathbf{S} = \{\mathbf{s}_i\}_{i=1}^n$ of n distinct points (also called "sites" or "generators") in Ω, the Voronoi diagram divides the domain Ω into n disjoint regions $V(\mathbf{S}) = \{V(\mathbf{s}_i)\}_{i=1}^n$ based on the Euclidean distance, as shown in the inset (left). Each region $V(\mathbf{s}_i)$ of the point $\mathbf{s}_i$, called Voronoi region, is defined as:



$$V(\mathbf{s}_i) = \{\mathbf{s} \in \Omega | \|\mathbf{s} - \mathbf{s}_i\| \leq \|\mathbf{s} - \mathbf{s}_j\|, \forall j \neq i\} \tag{1}$$

where $d(\mathbf{s}, \mathbf{s}_i) = \|\mathbf{s} - \mathbf{s}_i\|$ denote the Euclidean distance of two points $\mathbf{s}$ and $\mathbf{s}_i$.

The Power diagram (Aurenhammer, 1987), as an extension of the Voronoi diagram, introduces weights $\mathbf{W} = \{w_i\}_{i=1}^n$ to points. That is, each point $\mathbf{s}_i$ is associated with a parameter w_i. Each region $P(\mathbf{s}_i)$, called power cell, is redefined as:

$$P(\mathbf{s}_i) = \{\mathbf{s} \in \Omega | \|\mathbf{s} - \mathbf{s}_i\|^2 - w_i \leq \|\mathbf{s} - \mathbf{s}_j\|^2 - w_j, \forall j \neq i\} \tag{2}$$

where $d_p(\mathbf{s}, \mathbf{s}_i) = \|\mathbf{s} - \mathbf{s}_i\|^2 - w_i$ is redefined as the power distance of points $\mathbf{s}$ and $\mathbf{s}_i$. Notably, the power diagram degenerates to the Voronoi diagram when the weights of all points are equal (Aurenhammer et al., 1998).

3.2. CVT

By imposing centroidal constraint to ordinary Voronoi diagram and power diagram, two extensions of the Voronoi diagram can be generated: CVT (Liu et al., 2009) and centroidal power diagram (CPD) (Zheng et al., 2021). Taking the CVT as an example, each point $\mathbf{s}_i$ is located at the barycenter $\mathbf{s}_i^*$ of its Voronoi region $V(\mathbf{s}_i)$, and the barycenter $\mathbf{s}_i^*$ can be calculated as:

$$\mathbf{s}_i^* = \frac{\int_{V(\mathbf{s}_i)} \rho(\mathbf{s})\mathbf{s}d\mathbf{s}}{\int_{V(\mathbf{s}_i)} \mathbf{s}d\mathbf{s}} \tag{3}$$

Equivalently, CVT can be obtained by minimizing the following term:

$$Q(\mathbf{S}) = \sum_{i=1}^{n} \int_{V(\mathbf{s}_i)} \rho(\mathbf{s})\|\mathbf{s} - \mathbf{s}_i\|^2 d\mathbf{s} \tag{4}$$

Lloyd's (Liu et al., 2009) and quasi-Newton methods (De Goes et al., 2012) are the most popular implementations of CVT/CPD. The former is the simplest way to generate CVT, which moves each site to the mass center of its Voronoi cell in each iteration. The latter applies a quasi-Newton-like solver to compute the CVT/CPD. Compared to Lloyd's method with linear convergence, the quasi-Newton method with super-linear convergence is more efficient (Xin et al., 2016).

3.3. RPF

Given a point $\mathbf{s}_i$ on surface $\mathcal{M}$, associated with a shadow point $\mathbf{s}_i^s = \mathbf{s}_i + d \cdot \mathbf{n}_{\mathbf{s}_i}$, where $\mathbf{n}_{\mathbf{s}_i}$ is the normal of $\mathbf{s}_i$ and d is the offset. The weight of the point $\mathbf{s}_i$ is $w_i = 0$, while $w_i^s = \|\mathbf{s}_i - \mathbf{s}_i^s\|^2$ is set for the corresponding shadow site $\mathbf{s}_i^s$. The point $\mathbf{s}_i$ is therefore typically located on the equal power distance facet, and Xu et al. (2019) introduce the RPF $\mathcal{F}_i$:

$$\mathcal{F}_i = \{\tau \,|\, \tau \subset P(\mathbf{s}_i), \tau \subset P(\mathbf{s}_i^s)\} \tag{5}$$

Obviously, the RPF $\mathcal{F}_i$ is defined as the co-planar facet of their power cells $P(\mathbf{s}_i)$ and $P(\mathbf{s}_i^s)$, as illustrated in Fig. 1(c). Based on the CVT framework implemented with CGAL (Fabri and Pion, 2009), they provide a method to compute the RPF (Xu et al., 2019). To be specific, they first compute the regular triangulation of all sampling points and shadow points, thereby constructing the power diagram based on the duals of regular triangulation. To the best of our knowledge, the power diagram construction in this way is not parallelizable. Moreover, the co-planar facet, i.e., RPF, of the respective power cells of each sampling point and the relevant shadow point is highly dependent on the power diagram construction. Extraction of these co-planar facets is performed by traversing all vertices of each power cell only if the power diagram construction is done, which is inefficient.

Contrary to previous work (Xu et al., 2019), these shadow points and their weights in RPF are not necessary in our work. We are only concerned with cutting the tangent planes to generate approximative facets. Therefore, in this paper, we simplify the notion of RPF as RTF, which could be directly computed by truncating Voronoi cells of the tangent planes, as shown in Fig. 1(d). In the absence of misunderstandings, we also utilize $\mathcal{F}_i$ as the RTF of sampling point $\mathbf{s}_i$. In this paper, we provide an efficient GPU-based method to directly compute the RTF without any auxiliary points (e.g., these shadow points in RPF) or power diagram construction, which is described later.

4. Our methodology

The main components of our methodology are 1) the GPU-based computation of the RTF (Sect. 4.2) and 2) the projection strategy to restrict the sampling points moving on the original surface (Sect. 4.3). Note that the *KNN* method serves as the foundation for both modules. Additionally, the CVT-based optimization for the RTF and final mesh extraction (Sect. 4.4) are provided in this section.

4.1. Overall idea

Based on the ideas of TPI (Zimmer et al., 2012) and RPF (Xu et al., 2019), we propose an efficient GPU method for surface remeshing, where the restricted tangent faces are introduced to approximate the original surface. Unlike Xu et al. (2019), our method computes RTF directly from the sampling points and their normal directions without these shadow points or power diagram construction. In addition, a projection strategy is introduced to restrict the movement of sampling points on the original surface to generate a more accurate mesh. Notably, the RTF computation and projection strategy are based on the *KNN* method. Thanks to the open-source *KNN* method (Garcia et al., 2008), the RTF computation and projection strategy are well-designed and implemented in parallel with GPU acceleration. Thus, they are iteratively performed by computing a CVT, and the final meshes are generated from the optimized sampling points.

Given that the input mesh surface $\mathcal{M} = \{\mathcal{V}^m, \mathcal{T}^m\}$ is composed by a set of vertices $\mathcal{V}^m = \{v_i^m\}_{i=1}^{n_v^m}$ and a set of triangles $\mathcal{T}^m = \{t_i^m\}_{i=1}^{n_t^m}$, where n_v^m and n_t^m are the number of vertices and triangle facets. Each triangle facet t_i is represented by three ordered vertices, with indices 0, 1 and 2. Any combination of two vertices forms an edge of t_i. The sampling points set is represented as $\mathbf{S} = \{\mathbf{s}_i\}_{i=1}^n$, and the corresponding RTF set of all sampling points is denoted as $\mathcal{F} = \{\mathcal{F}_i\}_{i=1}^n$. The RTF $\mathcal{F}_i$ of sampling point $\mathbf{s}_i$ is composed by a set of vertices, that is, $\mathcal{F}_i = \mathcal{V}_i^f = \{v_j\}_{j=1}^{n_{\mathcal{F}_i}}$, where $n_{\mathcal{F}_i}$ represents the number of RTF vertices.

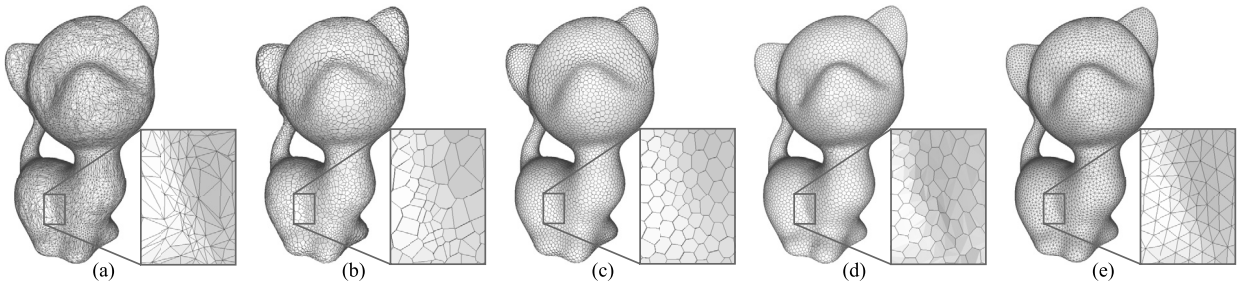


Fig. 2. Overall remeshing process of the *Kitten* model with our method. (a) input mesh (12.5K vertices, 25.0K faces); (b) initial tangent faces of sampling points (10.0K); (c) optimized restricted tangent faces; (c) reconnected restricted tangent faces; and (e) remeshing result (10.0K vertices, 20.0K faces).

Algorithm 1: Remeshing method based on RTF computation.

Input: origin mesh M, number of sampling points n, error termination ϵ, max iteration number max_{iter}
Output: remeshing result M'

Initialization: $\mathbf{S} = \{\mathbf{s}_i\}_{i=1}^{n}$ are randomly sampled from M
while $\delta > \epsilon \ \& \ nb_{iter} < max_{iter}$ **do**
  estimating normals $\mathbf{N} = \{\mathbf{n}_i\}_{i=1}^{n}$ of sampling points $\mathbf{S}$ `// Normal estimation, Sect. 4.2`
  for $\boldsymbol{s}_i \in \boldsymbol{S}$ *in parallel* **do**
    computing the RTF $\mathcal{F}_i$ of the point $\mathbf{s}_i$ `// RTF computation, Algorithm 2, Sect. 4.2`
    calculating the barycenter $\mathbf{b}_i$ of the RTF $\mathcal{F}_i$ `// RTF barycenter, Eq. (7), Sect. 4.2`
    computing $\mathbf{b}_i^p$ by projecting the $\mathbf{b}_i$ to surface M `// projection strategy, Algorithm 3`
    $\delta_i = |\mathbf{s}_i - \mathbf{b}_i^p|$ `// distance error of current point`
    $\mathbf{s}_i \leftarrow \mathbf{b}_i^p$ `// point optimization, Sect. 4.4`
  end
  $\delta = \max_{i=1,2,\cdots,n}\{\delta_i\}$ `// max distance error`
end
Extraction: generating mesh M' from the optimized points $\mathbf{S}$ `// remeshing, Sect. 4.4`

It should be mentioned that the sampling points are randomly sampled from the input surface $\mathcal{M}$ in an initialization stage. The pseudo code of our method for surface remeshing is provided in Algorithm 1, and the remeshing process is given in Fig. 2. Our method consists of the following steps:

a) computing the RTF and barycenter of each sampling point directly; (Sect. 4.2)
b) projecting the barycenter (optimized sampling point) onto the input surface; (Sect. 4.3)
c) extracting triangle meshes from the optimized sampling points. (Sect. 4.4)

In fact, the sampling initialization in our method could be easily extended to weighted sampling or anisotropic sampling (Chen et al., 2018) to capture the surface curvatures.

4.2. RTF computation

Previous work (Xu et al., 2019) computes the RPF based on the CVT framework with CGAL (Fabri and Pion, 2009) on the CPU. The RPF is filtered from the 3D power diagram by traversing the vertices of each power cell. Notably, as the shadow point is determined based on the normal of each sampling point in RPF (Xu et al., 2019), we observe that the RPF of a sampling point resembles its tangent plane. Thus, we simplify the calculation of RPF by computing the RTF in this paper. Based on the 3D clipped framework (Ray et al., 2018), we design a GPU-based method to compute the RTF based on the sampling point and its normal direction. Fortunately, the computation of the barycenter, volume, or integral of Voronoi cell is unnecessary in our method, and we are only interested in the intersection of RTF. Thus, the computational efficiency of the 3D clipped Voronoi diagram framework is further improved in our method.

To be specific, we use a *pre-clipping* for each sampling point $\mathbf{s}_i$ to compute a convex cell $\mathcal{V}_i$ by clipping an initialized bounding box. The *pre-clipping* is based on the orthogonal duals with the k-nearest-neighbors of $\mathbf{s}_i$. Then, we directly compute the equation of a half-space plane, called the RTF plane, by the sampling point $\mathbf{s}_i$ and its normal direction $\mathbf{n}_i$. Thus, the convex cell $\mathcal{V}_i$ is further clipped by the RTF plane to obtain the exact RPF vertices $\mathcal{F}_i = \{v_i^f\}_{i=1}^{n_{\mathcal{F}_i}}$, that is the *RTF-clipping* process. The flow of our RTF computation method is shown in Fig. 3, and the pseudo-code is presented in Algorithm 2. The detailed description of each step in the RTF computation method is given as follows.

KNN & Pre-clipping. The RTF computation method is on the basis of the 3D clipped Voronoi diagram framework (Ray et al., 2018; Liu et al., 2022), in which a key technique is the *KNN* method. The k-nearest-neighbors $\mathbf{S}_i^k$ of each sampling point $\mathbf{s}_i$ are used to clip a convex cell $\mathcal{V}_i$ in the *pre-clipping* process. Some efficient implementations of the *KNN* method are available, such as the multi-threading *KNN* on the CPU (Yan and Qixiang, 2009) or the parallelized *KNN* on the GPU

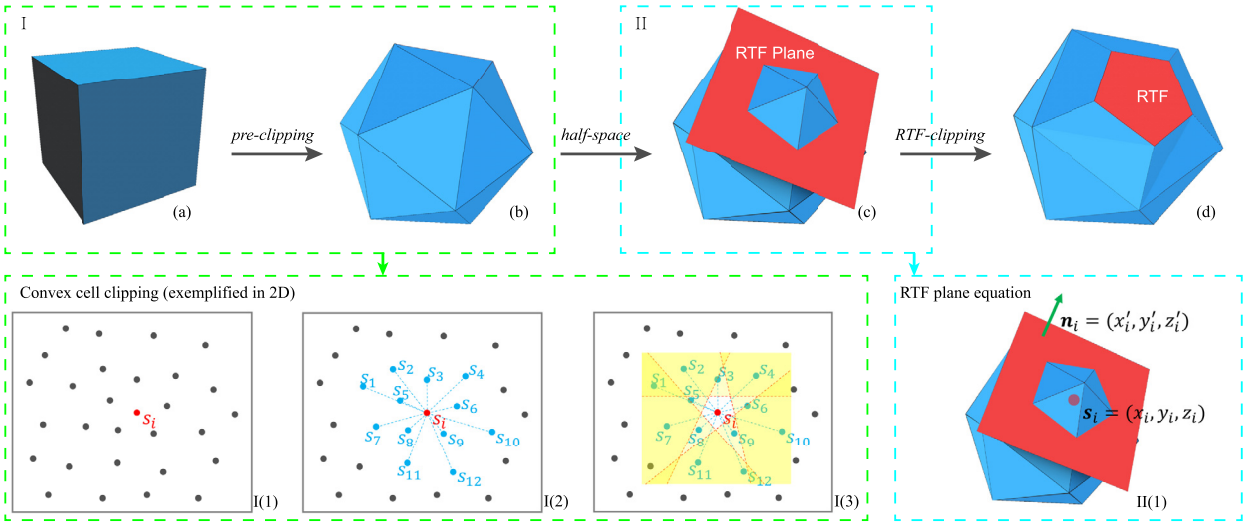


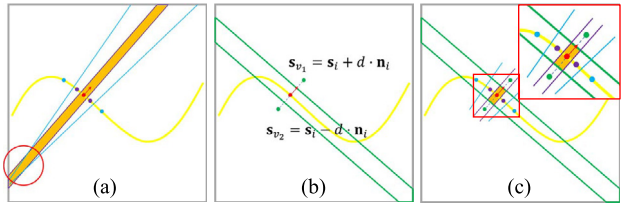
Fig. 3. The RTF computation process of a sampling point $\mathbf{s}_i$ on surface. Top: an overview of the RTF computation; (a) a convex cell initialized with bounding box, (b) the pre-clipped convex cell, (c) the half-space plane (RTF plane), and (d) the obtained RTF (red colored). Bottom: exemplification of the *pre-clipping* in 2D and the half-space plane; I(1) sampling points, I(2) k-nearest-neighbors of $\mathbf{s}_i$, I(3) *pre-clipping* process, and II(1) half-space plane $\mathcal{P}_{\mathcal{F}_i}$.

Algorithm 2: GPU-based computation for RTF.

Input: sampling points $\mathbf{S}$, points normal $\mathbf{N} = \{\mathbf{n}_i\}_{i=1}^n$
Output: RTF $\mathcal{F} = \{\mathcal{F}_i\}_{i=1}^n$
for $\mathbf{s}_i \in \boldsymbol{S}$ *in parallel* **do**
  $\mathcal{V}_i \longleftarrow BoundingBox(\mathbf{s}_i)$ `// initialize a convex cell`
  $\mathbf{S}_i^k \longleftarrow k\text{-}nearest\text{-}neighbors$ `// KNN`
  $\mathcal{V}_i \longleftarrow$ clipping by the orthogonal duals of $\mathbf{S}_i^k$ `// pre-clipping`
  $\mathcal{P}_i^{\mathcal{F}} \longleftarrow$ plane equation of $\mathcal{F}_i$ based on Eq. (6) `// RTF equation`
  $\mathcal{F}_i \longleftarrow$ clipping by the RTF plane $\mathcal{P}_{\mathcal{F}_i}$ `// RTF-clipping`
end

(Garcia et al., 2008). Owing to the purpose of parallelized computing RPF on the GPU, we use the simple and open-source *Brute-and-Force* strategy (Garcia et al., 2008).

Once the k-nearest-neighbors $\mathbf{S}_i^k$ of a sampling point $\mathbf{s}_i$ are obtained, the *pre-clipping* process is performed as follows. The bounding box of the primordial surface is used to establish a convex cell $\mathcal{V}_i$ of the sampling point $\mathbf{s}_i$. After obtaining the k-nearest-neighbors $\mathbf{S}_i^k$, a *pre-clipping* step is then carried out to clip the convex cell $\mathcal{V}_i$ by the orthogonal duals of the sampling point $\mathbf{s}_i$ and its $\mathbf{S}_i^k$. Fig. 3(I(1)–I(3)) illustrates the *pre-clipping* process in two dimensions, that is similar to Ray et al. (2018). However, the k-nearest-neighbors $\mathbf{S}_i^k$ are often close to the RTF plane $\mathcal{P}_{\mathcal{F}_i}$ (but usually not on the RTF plane), which may lead to superfluous clippings and result in a thin polyhedral cell. These meaningless clips do not impact the RTF results, but they lengthen computation times.



In our method, a two-virtual-point clipping proceeds before the *pre-clipping* stage, aiming to produce a suitable convex cell, which is further clipped by the orthogonal duals of its neighbors, as shown in the illustration. For each sampling point $\mathbf{s}_i$ on the original surface, associated with a normal $\mathbf{n}_i$, the two virtual points $\mathbf{s}_{v_1}$ and $\mathbf{s}_{v_2}$ are defined as: $\mathbf{s}_{v_1} = \mathbf{s}_i + d \cdot \mathbf{n}_i$ and $\mathbf{s}_{v_2} = \mathbf{s}_i - d \cdot \mathbf{n}_i$, where d is the offset of virtual points. In fact, due to the orthogonal duals of virtual points and the RTF being parallel, the value of d has no impact on the RTF results, and we set d to 20 in our work. By taking advantage of two-virtual-point clipping, our method avoids the unnecessary intersection (red circle in the illustration (a)) in the subsequent *pre-clipping* stage, improving the RTF computation efficiency.

Normal estimation & RTF plane equation. The normal estimation of each sampling point $\mathbf{s}_i$ on the surface, which is intimately connected to the RTF plane $\mathcal{P}_{\mathcal{F}_i}$, is a crucial step in our methodology. In actuality, the RTF plane equation $\mathcal{P}_{\mathcal{F}_i}$ could be calculated using nothing more than the normal direction and position of sampling point $\mathbf{s}_i$. The underlying idea of the normal estimation is to establish an approximation plane using the k nearest neighbors, whose normal represents the estimated normal of the sampling point $\mathbf{s}_i$.

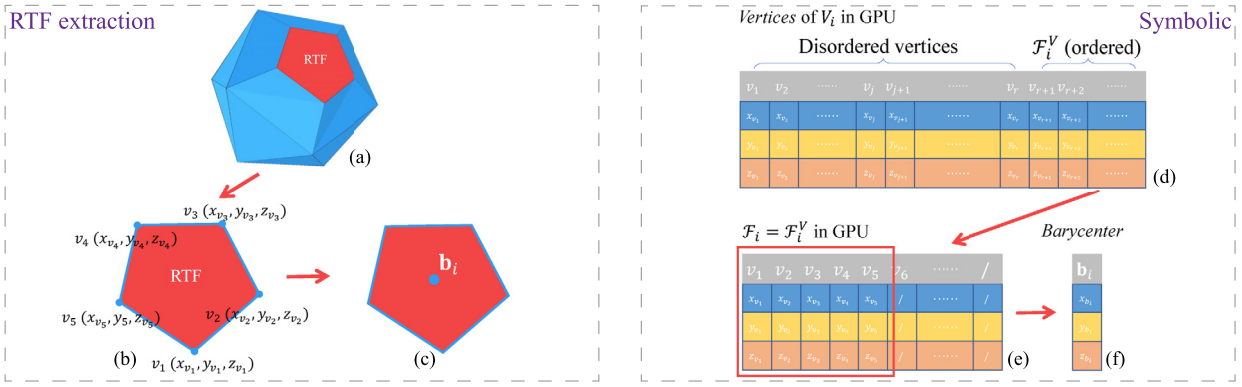


Fig. 4. The RTF extraction and symbolic representation in our method. Left: the RTF extraction process; (a) a convex cell $\mathcal{V}_i$, (b) vertices of the RTF $\mathcal{F}_i$, and (c) the barycenter $\mathbf{b}_i$ of $\mathcal{F}_i$. Right: the symbolic representation in GPU; (d) the vertex storage of $\mathcal{V}_i$, (e) the vertex storage of $\mathcal{F}_i$, and (f) the storage of RTF barycenter $\mathbf{b}_i$.

In our method, we immediately utilize the Point Cloud Library (PCL) (Rusu and Cousins, 2011) to determine the normal direction of each sampling point in $\mathbf{S}$ since PCL has been designed to perform the function of normal estimation. After acquiring the normal directions of sampling points $\mathbf{S}$, we could immediately calculate the related RTF plane equation, as shown in Fig. 3 II(1). To be specific, let $\mathbf{n}_i = (x_i', y_i', z_i')$ represent the normal of the sampling point $\mathbf{s}_i = (x_i, y_i, z_i)$, and the equation of $\mathcal{P}_{\mathcal{F}_i}$ could be calculated as follow:

$$\mathcal{P}_{\mathcal{F}_i}: \quad x_i' \cdot x + y_i' \cdot y + z_i' \cdot z - \mathbf{n}_i \cdot \mathbf{s}_i = 0 \tag{6}$$

RTF-clipping* & *Symbolic representation. A further intersection (called *RTF-clipping*) of the pre-clipped convex cell from the preceding section with the RTF plane yields the precise RTF result that is similar to the *pre-clipping* process. In contrast to the truncation of Voronoi cells of the input mesh producing partial surface (Yan et al., 2009), the intersection in our method creates a bounded plane considerably more efficiently.

The data structure of the 3D clipped Voronoi diagram is fully described in Ray et al. (2018). A convex polyhedron is typically composed of a set of vertices and half-space plane equations. Each vertex is represented by a dual triangle, i.e., IDs of the half-space planes. However, there is neither an explicit storage structure for each facet in a Voronoi cell nor an order storage for the vertices of the Voronoi cell. Thus, additional post-processing is necessary for extracting vertices of a specific facet. Fortunately, we observe that these intersections are ordered in the latest clipping, which could be directly extracted without traversing all vertices of the Voronoi cell. By making use of this observation, a well-designed clipping procedure is described in our method, where the *RTF-clipping* is viewed as the last stage in the RTF computation process. As a result, instead of traversing all vertices in the convex polyhedron $\mathcal{V}_i$, the vertices $\{v_j\}_{j=1}^{n_{\mathcal{F}_i}}$ of an RTF $\mathcal{F}_i$ could be directly derived from the latest incision, as shown in Fig. 4. Moreover, the barycenter $\mathbf{b}_i$ of the RTF $\mathcal{P}_{\mathcal{F}_i}$ could be easily calculated as:

$$\mathbf{b}_i = \frac{1}{n_{\mathcal{F}_i}} \cdot \sum_{j=1}^{n_{\mathcal{F}_i}} v_j \tag{7}$$

4.3. KNN-based projection strategy

As previously explained, we parallelly calculate the RTF barycenters $\mathbf{B}_i$ of sampling points $\mathbf{S} = \{\mathbf{b}_i\}_{i=1}^{n}$ on the original surface $\mathcal{M}$. Thus, in each iteration, the sampling points are updated to their relevant barycenters. This, however, can only produce unconstrained points that may diverge from the original surface $\mathcal{M}$, causing remeshing outputs with significant errors or failures. With this regard, we borrow the notion of pulling back in Chen et al. (2018) and design a *KNN*-based projection strategy to prevent the sampling points from moving off the original surface $\mathcal{M}$.

To achieve this, we directly employ the triangular facets on the original surface rather than the approximate underlying surface derived using a best-fitting plane in Chen et al. (2018). Similar to the previous RTF computation, our objective is to independently project each barycenter $\mathbf{b}_i$ onto the original surface $\mathcal{M}$, which could be accomplished in parallel by GPU acceleration. Due to the successful application of the *KNN* method in the above text, we also utilize it in the projection strategy. Specifically, we firstly seek for the k-nearest-neighbor vertices $\mathbf{N}_i^k$ of each RTF barycenter $\mathbf{b}_i$ from the original surface $\mathcal{M}$, as shown in Fig. 5(a). Then, the triangular facets from $\mathcal{M}$ with at least one vertex v^p in $\mathbf{N}_i^k$ are easily acquired. Thus, the barycenter $\mathbf{b}_i$ is projected onto the triangular facets, and the nearest point is chosen as the projected point $\mathbf{b}_i^p$. The pseudo of the *KNN*-based projection strategy is provided in Algorithm 3. Given the process of projecting the barycenter $\mathbf{b}_i$ onto a triangular facet t_j, two main cases are considered in our method:

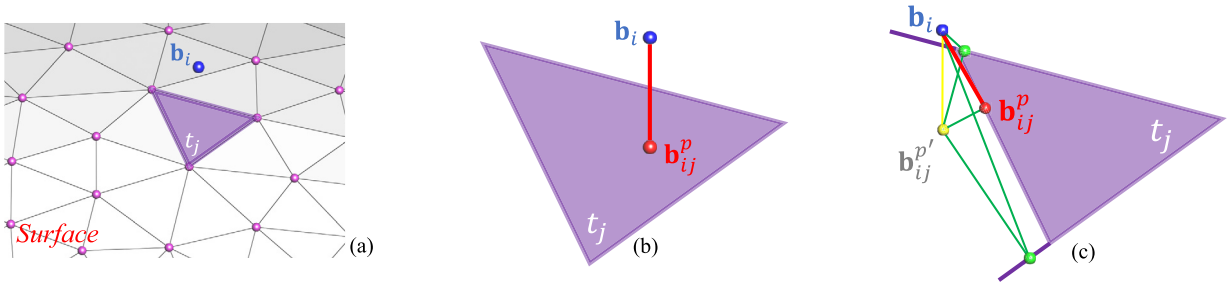


Fig. 5. Illustration of the projection strategy using the k-nearest-neighbor vertices $\mathbf{N}_i^k$, where the target facet is represented by the purple colored triangle t_j. (a) the k-nearest-neighbor vertices (purple dots) from original surface $\mathcal{M}$ of the barycenter $\mathbf{b}_i$ (blue dot); (b) the barycenter $\mathbf{b}_i$ is projected onto the triangle t_j, generating the projected point $b_{ij}^{p'}$; and (c) the barycenter $\mathbf{b}_i$ is projected outside the triangle t_j (yellow dot), and a further projection is utilized to produce the projected point $\mathbf{b}_{ij}^p$ (red dot) with nearest projection distance (red line).

Algorithm 3: *KNN*-based projection strategy.

Input: original mesh surface M, RTF barycenter $\mathbf{B} = \{\mathbf{b}_i\}_{i=1}^n$
Output: projected points $\mathbf{B}^p = \{\mathbf{b}_i^p\}_{i=1}^n$
for $\boldsymbol{b}_i \in \boldsymbol{B}$ *in parallel* **do**
  $\mathbf{N}_i^k \leftarrow k$-nearest-neighbor vertices from original surface $\mathcal{M}$
  for *each triangle* $t_j (\exists v^p \in t_j, v^p \in \boldsymbol{N}_i^k)$ **do**
    $\mathbf{b}_{ij}^p \leftarrow$ the projected point on the triangle t_j of $\mathbf{b}_i$
    $d_{ij}^p \leftarrow$ the projection distance $d(\mathbf{b}_i, \mathbf{b}_{ij}^p)$
  end
  sorting all projection distances $\{d_{ij}^p\}$ in ascending order
  $\mathbf{b}_i^p \leftarrow \mathbf{b}_{ij}^p$ with the minimal projection distance
end

a) the barycenter $\mathbf{b}_i$ is projected inside the triangular facet t_j (including the boundaries), as shown in Fig. 5(b);
b) the barycenter $\mathbf{b}_i$ is projected outside the triangular facet t_j, as shown in Fig. 5(c).

The projected point $\mathbf{b}_{ij}^p$ of the RTF barycenter $\mathbf{b}_i$ on the triangular facet t_j is straightforward in case a). While in case b), the early projection $\mathbf{b}_{ij}^{p'}$ of the barycenter $\mathbf{b}_i$ lies outside of the triangular facet t_j, as shown in the yellow dot in Fig. 5(c). Therefore, a further projection of $\mathbf{b}_{ij}^{p'}$ onto the edges of the triangular facet t_j is required, as shown in the green lines in Fig. 5(c). At this point, we could calculate the normal of t_j through its three vertices, then combine it with one edge $e \in t_j$, to obtain an exact plane $\mathcal{P}^{vir}$. The point $\mathbf{b}_{ij}^{p'}$ is further projected onto the plane $\mathcal{P}^{vir}$, which ensures the projection point falls on the edge e or its extension, as shown in the green and red dots in Fig. 5(c). Only these points on the edges of the triangular facet are taken into account, and the projection point $\mathbf{b}_{ij}^p$ of the RTF barycenter $\mathbf{b}_i$ is determined to be the point with the minimal projection distance, as shown the red dot in Fig. 5(c).

Consequently, each RTF barycenter $\mathbf{b}_i$ could be effectively restricted onto the original surface $\mathcal{M}$ by the two cases a) and b) above. Theoretically, there may be a corner case where the projection of the barycenter $\mathbf{b}_i$ onto the triangular facet t_j in Fig. 5(c) is not on any edge of t. At this time, the barycenter $\mathbf{b}_i$ is away from any vertex of the triangular facet t_j. In our method, this corner case is well avoided due to the usage of the *KNN* method. The reason is that several nearest neighbors $\mathbf{N}_i^k$ of the barycenter $\mathbf{b}_i$ are calculated so that each triangular facet t_j containing a vertex from $\mathbf{N}_i^k$ is not too far from the current barycenter $\mathbf{b}_i$. Notably, the vertex $v^p \in \mathbf{N}_i^k$ is taken as the projection point $\mathbf{b}_{ij}^p$ in the corner case to circumvent exceptions in the implementation of our method.

4.4. Optimization & mesh extraction

To generate high-quality meshes on the basis of the RTF computation, optimization and mesh extraction are two additional steps in our method, which are described in the following.

Optimization. For CVT optimization, the existing methods mainly involve Lloyd's method (Lloyd, 1982) with linear convergence and the quasi-Newton method (Liu et al., 2009) with super-linear convergence, e.g., the L-BFGS method (Xin et al., 2016). For instance, Alliez et al. (2005) optimize each sampling point in 2D parametric space using Lloyd's method. Yan et al. (2009) compute the CVT based on the quasi-Newton method, which is faster than Lloyd's method. However, in our method, there may be gaps or misaligned facets in the RTF of sampling points, as shown in Fig. 2. These misaligned facets

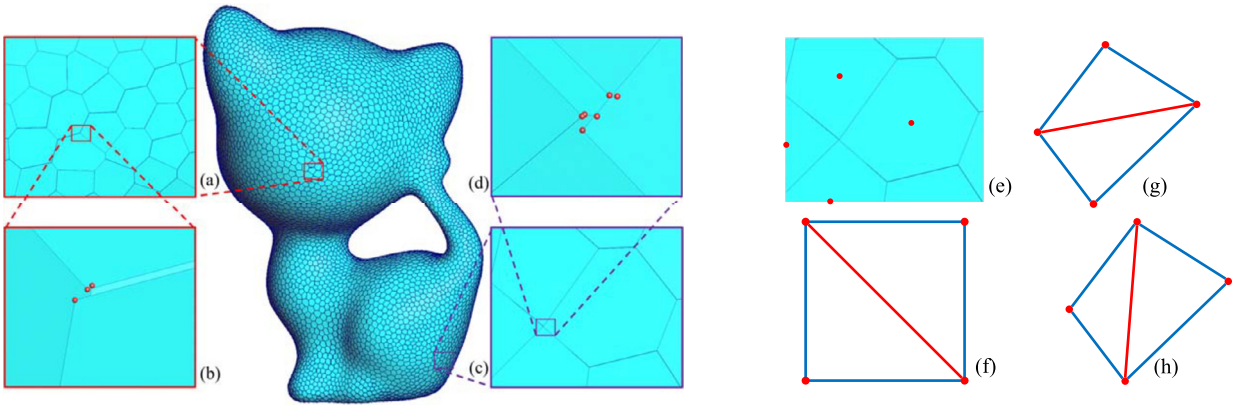


Fig. 6. Exemplification of RTF vertices reconnection in our method. (a) and (b) three co-circular sampling points; (c–e) four co-circular sampling points; (f) triangular meshes of equal quality in four co-circular sampling points; and (g–h) triangular meshes of different quality in four co-circular sampling points.

lead to difficulties in the computation of the laplacian operator matrix. Therefore, we turn to the simple and easy Lloyd's method to compute CVT, that is, the sampling point optimization.

To be more specific, the optimization of sampling points in our method works as follows: 1) the RTF $\mathcal{F}_i$ computation of each sampling point $\mathbf{s}_i$ by Algorithm 2; 2) the barycenter $\mathbf{b}_i$ calculation of each RTF $\mathcal{F}_i$ based on Equation (7); 3) the projection $\mathbf{b}_i^p$ of the barycenter $\mathbf{b}_i$ onto the original surface $\mathcal{M}$ by Algorithm 3; and 4) the update of each sampling point $\mathbf{s}_i$ based on $\mathbf{b}_i^p$. The four steps are performed in parallel on the GPU (lines 5–7 in Algorithm 1), and we also calculate the distance error δ_i between the sampling point $\mathbf{s}_i$ and its projected point $\mathbf{b}_i^p$ (line 8 in Algorithm 1) in each iteration. This optimization process is repeated until the convergence, or the maximum iterations are reached.

Mesh extraction. After the optimization process, we could obtain regularly distributed sampling points. Then, a mesh extraction process is used to produce the triangle meshes. However, as shown in Fig. 2(c), the misaligned facets in the optimized RTF make it almost impossible to directly generate the desired triangle mesh as the connectivity is ambiguous. One straightforward way is the reconnection strategy presented by Xu et al. (2019), where circumcenters for being dual are computed by clustering the neighbors of sampling points, from which the triangle meshes could be generated. Specifically, as shown in Fig. 6(b) and (d), these red dots are merged as two dual circumcenters, and each of them is correlated to several RTFs. Typically, a circumcenter is associated with three RTFs, and the corresponding clustered neighbor sampling points are reconnected as a triangle in our output, as shown in Fig. 6(a–b). In the degenerate cases, there are four co-circular sampling points on the same plane, as shown in Fig. 6(c–e). As shown in Fig. 6(g–h), two triangular divisions could be obtained. The division yielding higher-quality triangles is selected as the output in our method. In particular, when the quality of triangles in two different divisions are of the same quality, either of these two divisions is feasible in our method, as shown in Fig. 6(f).

Another robust way for mesh generation is the RVD-based mesh extraction method in Boltcheva and Lévy (2017), which is provided in Geogram (Lévy and Filbois, 2015). This method takes as input a filtered pointset and connects the input points with triangles by computing their restricted Voronoi diagram. An essential requirement of this method is a high-quality point set, which coincides with the output of the optimization process in our method. Therefore, the regularly distributed sampling points generated by the optimization process taken as the inputs of the RVD-based mesh extraction method is also an efficient way to produce triangular meshes of high quality.

5. Evaluation

In this section, we present various computational results to demonstrate the potency and usefulness of our remeshing method. The proposed algorithms are implemented using C++. The 3D clipped Voronoi diagram framework (Ray et al., 2018) is applied to compute the pre-clipped convex cell, and the normal directions are estimated using PCL 1.8.1. All experiments are performed on a Windows 10 computer with 3.6 GHz Intel (R) Core (TM) i7-9700K GPU with 16 GB memory and an NVIDIA GeForce RTX 2080 Ti with 11 GB memory, using CUDA version 10.0.

GPU specific. The ability of our method to compute each RTF independently is friendly for a GPU implementation. Here we provide some implementation details of our method. Similar to the 3D clipped Voronoi diagram (Ray et al., 2018), the sets for vertices and plane equations are stored in shared memory arrays of constant size max $\#T$ and $\#P$, respectively. The value of $\#T$ is set to 96, and $\#P$ is 64 in our experiments. The number of threads by blocks is set to 16, and the maximum iterations are 160. Moreover, the parameter k of k-nearest-neighbor is crucial in our method, which is described in detail in our experiments. Here, the default values are given as follows: $k_{nor} = 16$ in the *normal estimation* stage, $k_{clip} = 32$ in the *pre-clipping* stage, and $k_{proj} = 16$ in the *KNN-based projection* stage.

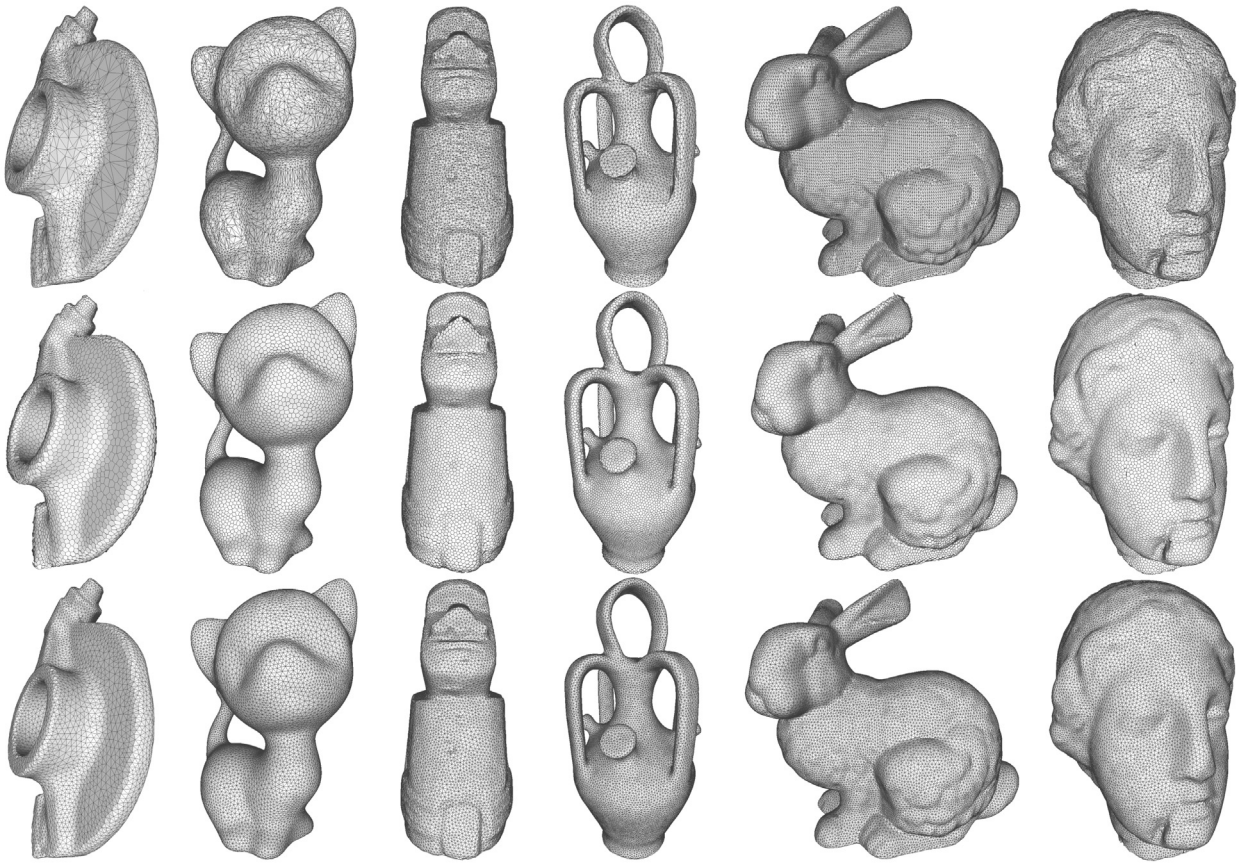


Fig. 7. Remeshing results of various models. From top to bottom: input meshes, optimized RTF and output meshes. From left to right: remeshing of the *Rocker* model with 8.0K vertices, *Kitten* model with 12.0K vertices, *Moai* model with 15.0K vertices, *Botijo* model with 20.0K vertices, *Bunny* model with 25.0K vertices and *Egea* model with 30.0K vertices. The percentage of RTFs with fewer than five or more than seven vertices in each result is reported: 1.400% (112/8000 in *Rocker*), 2.02% (242/12000 in *Kitten*), 2.23% (334/15000 in *Moai*), 1.96% (392/20000 in *Botijo*), 2.36% (591/25000 in *Bunny*), and 3.56% (1067/30000 in *Egea*).

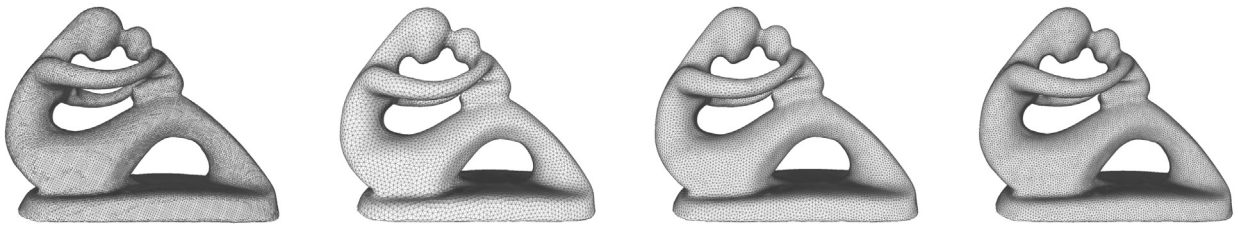


Fig. 8. Remeshing results of the *Fertility* model with different numbers of sampling points. From left to right: input model with 44.0K vertices, result with 10.0K sampling points (18.294 seconds), result with 20.0K sampling points (35.434 seconds), and result with 30.0K sampling points (62.906 seconds).

5.1. Qualitative evaluation

We now provide some computational results to evaluate the feasibility of our remeshing method. Firstly, we conduct an experiment on several models with different numbers of vertices, including the *Rocker*, *Kitten*, *Moai*, *Botijo*, *Bunny* and *Egea* models. The remeshing results of these models with our method are presented in Fig. 7. Then, we also test our method on one model (that is, the *Fertility* model), where the number of sampling points is set to 10.0K, 20.0K, and 30.0K. All sampling points are randomly sampled from the input surface (as shown in Fig. 8(a)), and the output meshes are illustrated in Fig. 8(b–d). According to these results in Fig. 7 and Fig. 8, we can observe that our method is feasible for surface remeshing, and the obtained results are of higher quality than the raw meshes.

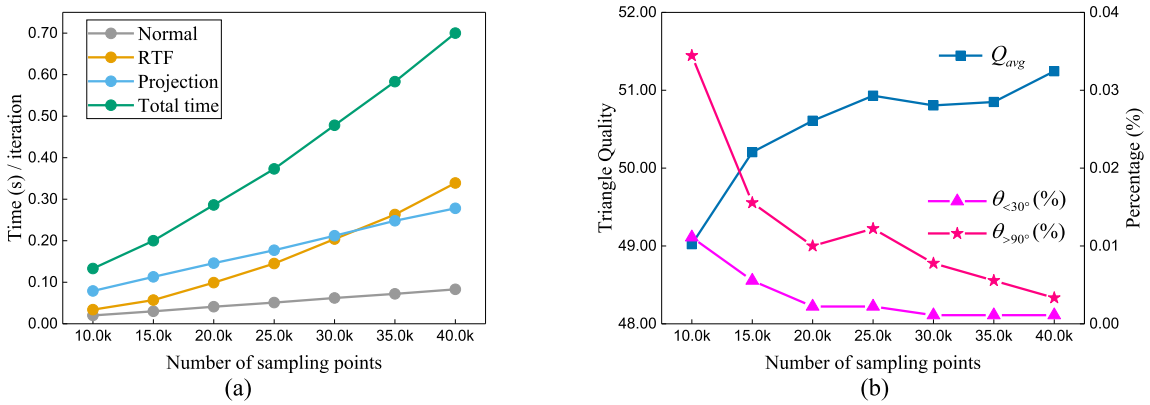


Fig. 9. Computational efficiency of our method with various amount of sampling points. (a) computational timing in each iteration with the *Fertility* model; (b) the corresponding triangle quality of the results in (a).

Table 1
Quality of output meshes with our method, corresponding to these results in Fig. 7.

Model	In/Out	#V [1]	#F [2]	Q_{min}	Q_{avg}	θ_{min}	$\theta_{min.avg}$	$\theta_{<30°}$	$\theta_{>90°}$	d_H [3]	T (s)
Moai	Input	23.2K	46.4K	0.005	0.709	2.672	38.048	0.244	0.356	–	–
	Output	15.0K	30.0K	0.603	0.921	31.477	53.173	0.000	0.001	0.023	22.167
Botijo	Input	10.7K	21.4K	0.026	0.669	1.571	35.332	0.336	0.399	–	–
	Output	20.0K	40.0K	0.597	0.914	32.029	52.670	0.000	0.001	0.041	39.864
Bunny	Input	35.3K	70.6K	0.008	0.715	4.825	36.946	0.068	0.138	–	
	Output	25.0K	50.0K	0.572	0.895	27.112	51.187	0.001	0.003	0.015	57.639
Egea	Input	50.0K	100.0K	0.030	0.631	1.246	36.424	0.029	0.104	–	–
	Output	30.0K	60.0K	0.578	0.897	26.853	51.028	0.003	0.005	0.075	76.481

[1] number of vertices; [2] number of facets; [3] Hausdorff distance ($\times 10^{-2}$).
Notably: the quality of *Rocker*, *Kitten* is referred to Table 2.

5.2. Quantitative evaluation

The quantitative results of some remeshing models using our method are reported in this section. We introduce mesh quality metrics to evaluate the computational efficiency, followed by a quality analysis and computational timings.

Mesh quality metric. The criteria in Khan et al. (2022) is utilized to assess the quality of generated meshes in our experiments. The quality of a triangle t in the mesh surface M is determined as $Q_t = \frac{6}{\sqrt{3}} \cdot \frac{A_t}{S_t E_t}$, where the area, half-perimeter, and the length of the longest edge of the triangle t are represented by A_t, S_t, and E_t, respectively. Thus, Q_{min} and Q_{avg}, which indicate the minimal and average triangle quality of the mesh surface, are utilized to quantify the mesh quality. Similarly, the average of the minimum angles in all triangles is θ_{avg}, and the minimum angle is θ_{min}. The percentage of triangles with minimum angles less than 30° is designated as $\theta_{<30°}$, while the percentage of those with maximum angles of more than 90° is $\theta_{>90°}$. We also evaluate the quality of meshes by the approximation error in terms of Hausdorff distance error d_H, which is normalized by the diameter of the bounding box. Moreover, the efficiency of the remeshing method is crucial to the subsequent geometric processing. Execution time is also taken into account as an important metric in our experiments to assess the efficiency of our method.

Quality analysis. Table 1 illustrates the quality of some output meshes generated by our method, corresponding to these remeshing results in Fig. 7. We can observe from the results in Table 1 that our method is effective for surface remeshing, and the quality of output meshes has significantly improved. Particularly, for the *Rocker*, *Kitten*, and *Botijo* models, the percentage of triangles with minimum angles smaller than 30° is decreased to 0, further demonstrating that our method yields high-quality remeshing results. However, for the special structures in these complex mesh surfaces, such as the ears in the Bunny model, our method yields slightly inferior triangles, which leads to a slightly lower quality of the remeshing results compared to these smooth mesh surfaces.

Efficiency analysis. The quantity of sampling points has an impact on the computational efficiency of our method. To demonstrate computational timings and the triangular quality of the remeshing results, we put the *Fertility* model to the test. Fig. 9(a) displays the computational timing profiles in each iteration, comprising the normal estimation (Normal), RTF and its barycenter computation (RTF), barycenter projection (Projection), and total time. As demonstrated in Fig. 9(b), we also offer the corresponding triangular quality of the remeshing results. According to the remeshing results in Fig. 9(a–b), we can observe that the triangular quality of the related results improves as the number of sampling points gradually grows, and so does the computational timing increases.

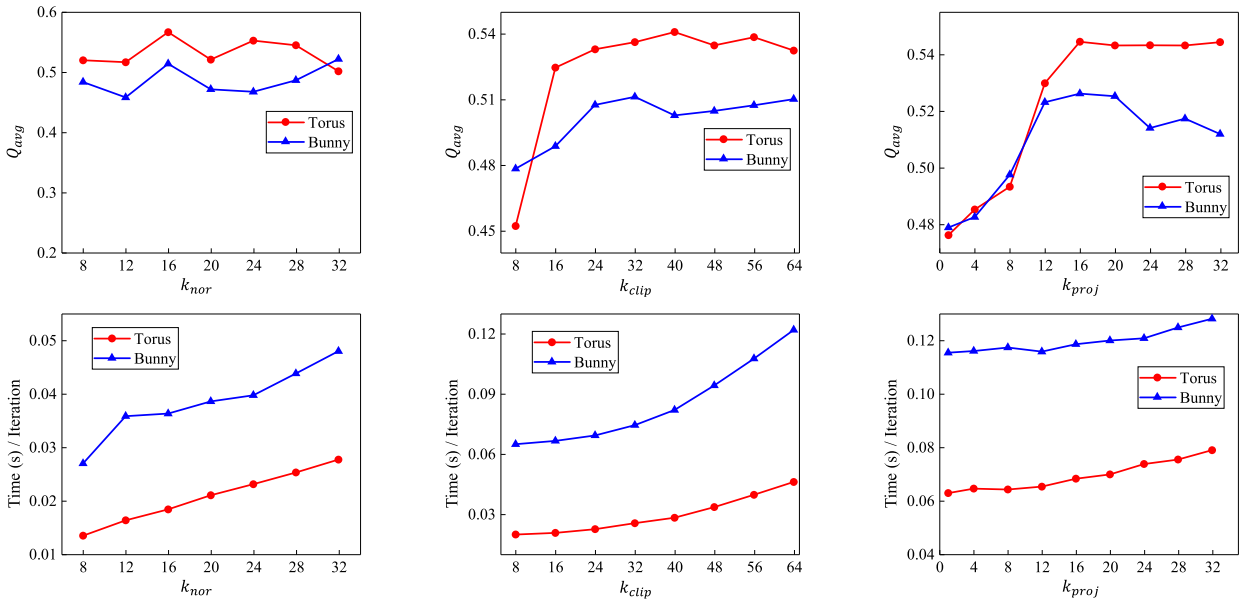


Fig. 10. The impact of *KNN* parameter k at different stages of our method on the *Torus* (10.0K sampling points) and *Bunny* (20.0K sampling points) models, where k_{nor} is used for the normal estimation stage, k_{clip} is applied for the *pre-clipping* stage, and k_{proj} is utilized for the projection stage. Top: the impact of different values of k_{nor}, k_{clip}, and k_{proj} on the quality of generated triangular meshes. Bottom: the impact of different values of k_{nor}, k_{clip}, and k_{proj} on the running time of each iteration in the corresponding stages.

Parameter analysis. A crucial technique is the *KNN* method in our implementation, which is frequently used in different stages, including the normal estimation k_{nor}, the pre-clipping k_{clip}, and the projection strategy k_{proj}. To analyze the effect of different values of these parameters on the generated meshes, a comprehensive experiment is performed, where the number of sampling points is set to 10.0K for the *Torus* model and 20.0K for the *Bunny* model. Notably, when analyzing the effect of one parameter, the others are set to the default values in this experiment. Fig. 10 presents the quality of generated meshes and the relevant running time of each iteration on different values of k_{nor}, k_{clip}, and k_{proj}.

Overall, remeshing with more sampling points results in longer iteration running times. In view of the parameter k_{nor}, the requirement of our method is that more sampling points are used to capture the characteristics of complicated models. Thus, the generated meshes are typical of comparable quality. Considering the effect of the parameter k_{clip}, a small value of the parameter k_{clip} results in an inaccurate convex polyhedron and the respective RTF, leading to a poor-quality mesh. However, the quality of the generated meshes significantly improves and stabilizes as the value of the parameter k_{clip} increases. Finally, when the parameter k_{proj} is taken into account, setting a small value of k_{proj} during the projection stage may lead to the aggregation of sampling points, especially at corners or edges, resulting in the generation of meshes with poor quality.

5.3. Comparison

To further demonstrate the viability and effectiveness of our method, we compare against two exact CVT-oriented remeshing techniques, including the maximal Poisson-disk sampling (MPS) (Guo et al., 2015) and the RVD (Yan et al., 2009), as well as two approximate CVT-oriented remeshing techniques, including the RPF (Xu et al., 2019) and the restricted Voronoi cell (RVC) (Chen et al., 2018) in terms of the quality of generated triangular meshes. Moreover, in view of the computational performance per iteration, we compare against the RVD and RPF running on a CPU, the multi-threading RVC on a multi-core CPU, and the clipped Voronoi diagram (CVD) running on a GPU, respectively.

Notably, despite the fact that the RVD is generally utilized within the quasi-Newton method to speed up the CVT energy calculations, we are solely interested in the intersection between Voronoi cells and the original surface. Using the code provided in Lévy and Liu (2010), we reimplemented the RVD algorithm, and Lloyd's method is applied to relocate these sampling points. Similarly, we reimplemented the CVD method for surface remeshing by replacing the Voronoi diagram computation in RVD (Yan et al., 2009) using the GPU-based construction method in Liu et al. (2022). Moreover, the RPF algorithm (Xu et al., 2019) in this comparison is reimplemented based on the normal estimation on PCL (Rusu and Cousins, 2011), as we only focus on the computation of these approximate facets, that is RPFs.

Remeshing results & quality analysis. We compare the generated meshes with our method against a number of other methods, including MPS (Guo et al., 2015), RVD (Yan et al., 2009), RPF (Xu et al., 2019), RVC (Chen et al., 2018). Fig. 11 presents the results of several remeshing techniques, and Table 2 reports the pertinent mesh qualities. The implementation details for this comparative experiment are as follows: three different raw meshes are taken as inputs for these remeshing

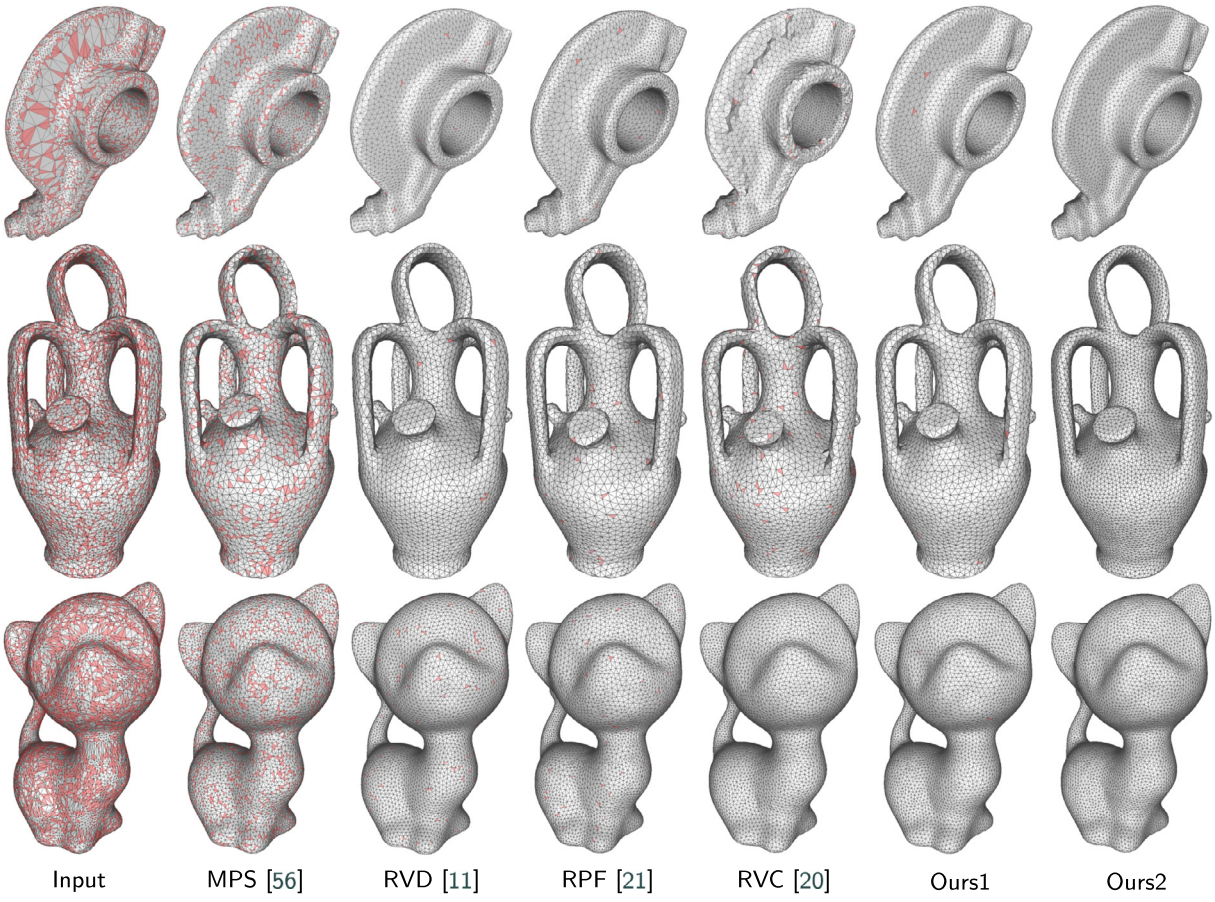


Fig. 11. Comparison results of different remeshing methods, where the colored triangles indicate the facets with lower quality ($\theta_{min} < 33°$). From top to bottom: remeshing results of the *Rocker*, *Botijo*, and *Kitten* models. From left to right: input mesh, remeshing results of MPS (Guo et al., 2015), RVD (Yan et al., 2009), RPF (Xu et al., 2019), RVC (Chen et al., 2018), Ours with less number of vertices (Ours1) and Ours with more number of vertices (Ours2).

methods, i.e., *Rocker*, *Botijo*, and *Kitten*. We initialize some uniformly distributed auxiliary points inside the mesh model to assist the 3D Voronoi diagram construction in the RVD, which is further intersected with the original surface. We apply the original mesh surfaces to the RVC, using their vertices as the original sampling points, even though the RVC requires point clouds as inputs to reconstruct surfaces. Notably, the *Botijo* model with 10.7K vertices in Table 1 is selected as input for the RVC as that with 3.0K in Table 2 results in the surface calculated abnormally. Furthermore, we show two results with our method under a different number of vertices, and the detailed information is provided in Table 2.

The results in Fig. 11 and Table 2 demonstrate the effectiveness of our method for surface remeshing. Compared to MPS (Guo et al., 2015), our method generates meshes of higher quality, but the reported time of MPS is better than ours, as seen in Fig. 11. Instead of truncating boundary Voronoi cells in the RVD (Yan et al., 2009), the original surface is roughly fitted by calculating the tangent planes. Thus, our method significantly reduces the computational complexity and generates meshes with similar or better triangular quality with less time consumption. As illustrated in the first row in Fig. 11, the RVC yields significant approximation error as it pulls the sampling points back to the best-fitting planes to represent the surface, which is influenced by the number of input points. Unlike the RVC, the sampling points are projected onto the triangular facet in the original surface, producing meshes of higher quality, especially smaller Hausdorff distance errors. Nevertheless, influenced by the accuracy of the normal estimation, there are still a few triangles with small or obtuse angles in the remeshing results with our method, which leads to low-quality triangles (as shown in colored facets in Fig. 11).

Efficiency comparison. An essential highlight of our method and its main strength is computational performance. We perform an experiment to track the computation times of different methods with various numbers of sample points on *Kitten* and *Botijo* models. These methods could be typically classified into those running on a CPU, including RVD (Yan et al., 2009) and RPF (Xu et al., 2019), and those running on a multi-core CPU or GPU, including RVC (Chen et al., 2018), CVD (Liu et al., 2022) and our method. Despite the fact that several optimization techniques have been used to optimize the position per sampling point, we only pay attention to the computational performance of these methods in one iteration. The corresponding record timings are shown in Fig. 12, where the number of sampling points is gradually increased from 10.0K to 30.0K. In view of the computation process of one iteration in these methods, the RVD calculates the exact

Table 2
Quality of the output results with different remeshing methods in Fig. 11, including the MPS (Guo et al., 2015), RVD (Yan et al., 2009), RPF (Xu et al., 2019), and Ours, where the red values indicate the best results and the green values represent the second-best results.

Model	Method	$\#V$ [1]	$\#F$ [2]	Q_{min}	Q_{avg}	θ_{min}	$\theta_{min.avg}$	$\theta_{<30^o}$	$\theta_{>90^o}$	d_H [3]	$T(s)$
Rocker	Input	9.4K	18.8K	0.004	0.695	0.240	36.456	0.272	0.350	–	–
	MPS	5.8K	11.6K	0.460	0.795	24.862	44.203	0.009	0.177	0.084	0.235
	RVD	5.8K	11.6K	0.588	0.882	32.926	50.411	0.000	0.018	0.035	117.25
	RPF	5.8K	11.6K	0.442	0.871	27.266	49.069	0.001	0.019	0.034	17.863
	RVC	5.8K	11.6K	0.436	0.894	24.966	51.489	0.003	0.002	0.103	9.867
	Ours1	5.8K	11.6K	0.578	0.884	29.019	51.753	0.000	0.002	0.057	5.680
	Ours2	8.0K	16.0K	0.592	0.902	29.347	52.035	0.000	0.001	0.023	8.289
Botijo	Input	3.0K	6.0K	0.026	0.590	1.571	30.722	0.494	0.554	–	–
	MPS	5.0K	10.0K	0.415	0.797	25.103	44.255	0.008	0.170	0.091	0.219
	RVD	5.0K	10.0K	0.568	0.884	30.405	50.650	0.000	0.018	0.055	188.791
	RPF	5.0K	10.0K	0.489	0.872	26.659	49.295	0.003	0.020	0.083	29.624
	RVC	5.0K	10.0K	0.448	0.826	25.447	46.255	0.003	0.009	0.102	11.371
	Ours1	5.0K	10.0K	0.571	0.894	29.433	51.590	0.000	0.001	0.077	7.534
	Ours2	10.0K	20.0K	0.599	0.913	32.391	52.528	0.000	0.000	0.058	15.724
Kitten	Input	12.5K	25.0K	0.008	0.663	5.123	35.823	0.327	0.467	–	–
	MPS	10.3K	20.6K	0.442	0.793	22.859	43.898	0.014	0.178	0.024	0.469
	RVD	10.2K	20.4K	0.524	0.883	29.438	50.478	0.001	0.019	0.029	223.517
	RPF	10.0K	20.0K	0.459	0.874	27.284	49.507	0.001	0.021	0.022	27.318
	RVC	10.0K	20.0K	0.570	0.878	32.715	50.439	0.000	0.025	0.058	18.263
	Ours1	10.0K	20.0K	0.572	0.908	31.147	52.139	0.000	0.001	0.007	13.334
	Ours2	12.0K	24.0K	0.599	0.912	33.672	52.620	0.000	0.000	0.006	16.651

[1] number of vertices; [2] number of facets; [3] Hausdorff distance ($\times 10^{-2}$).

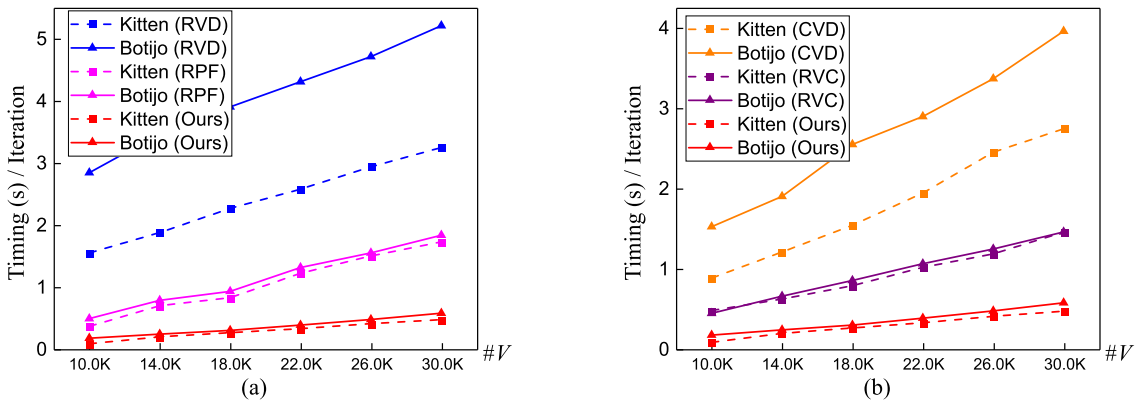


Fig. 12. Comparison of computational efficiency of various methods under different number of sampling points ranging from 10.0K to 30.0K. (a) computational efficiency of methods running on a CPU, including the RVD (Yan et al., 2009) and RPF (Xu et al., 2019); and (b) computational efficiency of methods running on a multi-core CPU or GPU, including the CVD (Liu et al., 2022) and RVC (Chen et al., 2018). Note that the computational efficiency of our method reported in (a) and (b) is the same.

intersection of boundary Voronoi cells and the original surface, which is complicated and time-consuming. Based on the intersection calculation algorithm in RVD, the CVD achieves better computational efficiency with the usage of GPU-accelerated 3D Voronoi diagram construction. Nevertheless, this does not circumvent the calculation of these exact intersections in RVD, which is still time-consuming. The RPF extracts the co-planar facets by traversing the constructed power diagram, achieving better computational efficiency compared to the RVD. The RVC calculates the CVT and pulls the sampling points back to a best-fitting plane and speedups by multi-threading. The RTF computation and the projection strategy in our method are implemented in parallel by taking full advantage of GPU acceleration. Therefore, our method achieves the best computational efficiency compared to other methods.

More qualitative comparison. Furthermore, we compare our method with that proposed by Leung et al. (2015) from a qualitative perspective. The two methods share several features: CVT-driven, extrinsic, and GPU-friendly. However, there are also significant differences between the two methods, and Table 3 illustrates the qualitative comparison results of the two methods at the methodological level. To be specific, in terms of CVT dimension and RVD computation, Leung et al. (2015) compute the 3D CVT and RVD, whereas our method computes the 2D CVT (the plane in 3D space) and RVD computation is unnecessary, which means that our method is more straightforward and more efficient. Considering the CVT computation and GPU acceleration, Leung et al. (2015) compute the exact EDT and CVT on GPU by discretizing the original surface

Table 3
Qualitative comparison to the method proposed by Leung et al. (Leung et al., 2015).

Method	Input	CVT			RVD	Efficiency
		Dimension	Computation	Accuracy		
Leung et al. (Leung et al., 2015)	mesh/non-mesh	3D	voxel-based	poor	necessary	fast
Ours	mesh	2D	clipping-based	good	unnecessary	faster

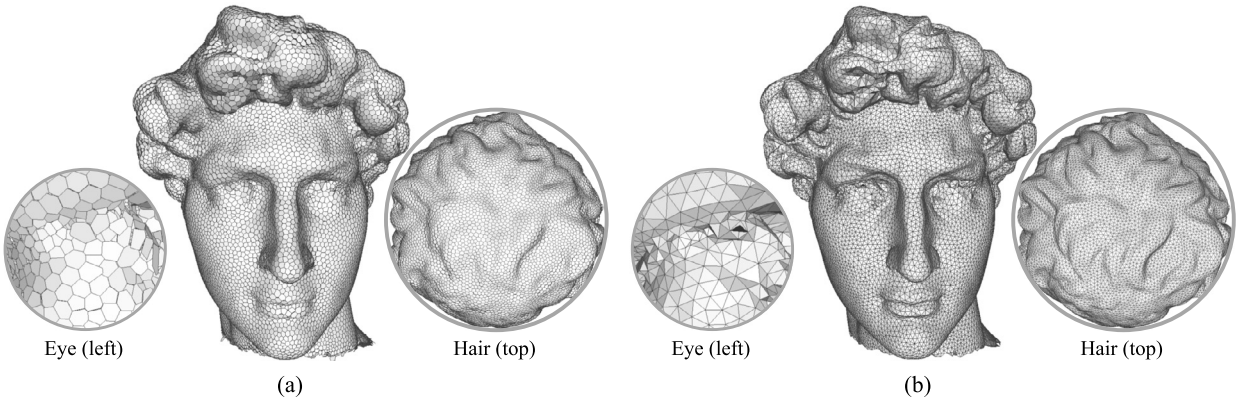


Fig. 13. Remeshing results of our method on a complex model *David Head* with 30.0K sampling points. (a) optimized RTF result; (b) the corresponding triangular mesh.

into voxels with a constant resolution, which is limited by the graphical memory and does not provide sufficient accuracy to guarantee the correctness of the CVT for complex models. In contrast, our method provides a more accurate 2D CVT by clipping the tangent in a resolution-independent manner to approximate the original surface. However, the method proposed by Leung et al. (2015) has advantages in remeshing various surface representations, including implicit, meshes, and point clouds.

5.4. More results

Complex models are commonly used in practical applications, e.g., modeling in computer games, physics simulations, etc. We conduct an experiment on a complex model, i.e., the *David Head*, to further evaluate the capability of our method. The number of sampling points is set to 30.0K in this experiment, and the remeshing results are presented in Fig. 13. To illustrate the RTF and corresponding triangular meshes in detail, we provide additional details in the remeshing results of the *David Head*, e.g., the left eye and the hair on top of the head. From the results in Fig. 13, it could be observed that our method is feasible for complex models, and we could obtain the desired triangular meshes on these relatively smooth regions. Nevertheless, regions with complex features are still challenging for our method, as the normal estimation through these sampling points cannot well-capture these complicated regions, e.g., the left eye of the *David Head*.

6. Conclusion

In this paper, we provide an effective GPU method for producing high-quality remeshing results. Based on the principle of the planar approximation, a parallel method is presented to compute the RTF of each sampling point directly, and each sampling point is further optimized to the barycenter of its RTF. Additionally, we present a *KNN*-based projection strategy to restrict the movement of sampling points on the original surface during the optimization process. Experimental results on a variety of mesh surfaces demonstrate the feasibility and efficiency of our method.

Limitations and future work. Despite the fact that our method offers various techniques for producing high-quality remeshing results, there are still several difficulties in our work. The main limitation is that the proposed method does not have theoretical guarantees, although experimental results demonstrate its validity. Another limitation is the challenge of remeshing complex geometries, especially with few sampling points. A small number of sampling points can hardly capture some details of a complex model, e.g., hands of *Homer* model, etc., which leads to incorrect RTF or even failure remeshing, as shown in Fig. 14(a). Besides, our method introduces the RTF, coupled with the CVT optimization, to yield regularly distributed sampling points, thus generating high-quality meshes. However, the surface curvature is not considered in our work, making it challenging to produce remeshing results with sharp feature preservation or surface curvature adaptation, as shown in Fig. 14(b). Finally, some special topological structures are commonly used in practical applications, e.g., thin-plate or close-plate models. Due to these thin-plate or close-plate structures, the normal estimation of sampling points may be incorrect, resulting in an incorrect RTF result, as shown in Fig. 14(c).

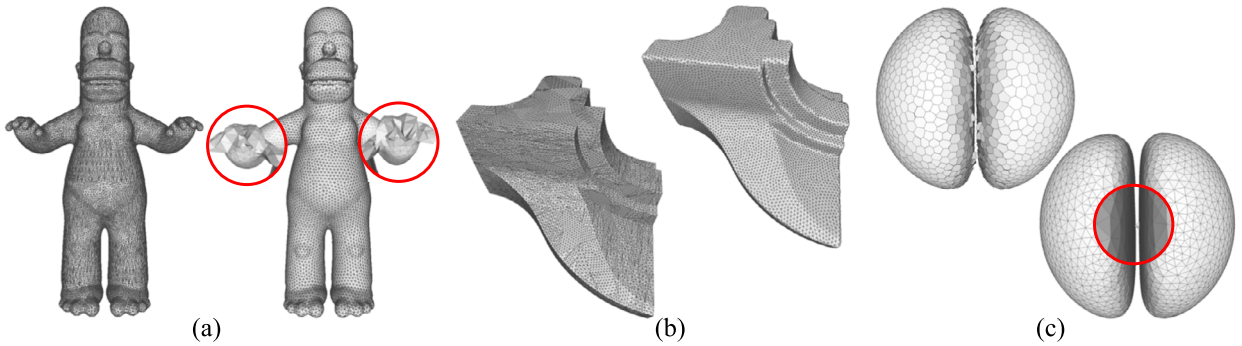


Fig. 14. Exemplifications of several limitations in our work. (a) failure in capturing details of the *Homer* model; (b) remeshing without sharp feature preservation of the *Fandisk* model; and (c) shortcomings in remeshing with close-plate of the *Close Hemisphere* model.

For future work, we are thinking of merging the proposed RTF with power diagrams for non-uniform remeshing. We believe that by giving each sampling point a weight parameter to regulate the area size of the relevant RTF, the surface curvature could be characterized. Additionally, verifying the normal orientation of each sampling point is another way to potentially solve defects in thin-plate or close-plate surface remeshing.

CRediT authorship contribution statement

Yuyou Yao: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft. **Jingjing Liu:** Data curation, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Wenming Wu:** Formal analysis, Methodology, Writing – review & editing. **Gaofeng Zhang:** Formal analysis, Resources, Writing – review & editing. **Benzhu Xu:** Resources, Software, Supervision. **Liping Zheng:** Conceptualization, Funding acquisition, Investigation, Methodology, Project administration, Resources, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgements

The models used in this paper are from the AIM@Shape and courtesy of the Stanford 3D Scanning Repository. We would like to thank Zhonggui Chen and Jianwei Guo for providing their executable programs, and we also thank Bruno Lévy and Yang Liu for the source code of truncating Voronoi cells. This work was supported in part by a grant from the National Natural Science Foundation of China (No. 61972128) and the National Key Research and Development Plan of China (No. 2022YFC3900800).

References

Abdelkader, A., Bajaj, C.L., Ebeida, M.S., Mahmoud, A.H., Mitchell, S.A., Owens, J.D., Rushdi, A.A., 2020. VoroCrust: Voronoi meshing without clipping. ACM Trans. Graph. 39, 1–16.
Alliez, P., De Verdière, É.C., Devillers, O., Isenburg, M., 2005. Centroidal Voronoi diagrams for isotropic surface remeshing. Graph. Models 67, 204–231.
Aurenhammer, F., 1987. Power diagrams: properties, algorithms and applications. SIAM J. Comput. 16, 78–96.
Aurenhammer, F., Hoffmann, F., Aronov, B., 1998. Minkowski-type theorems and least-squares clustering. Algorithmica 20, 61–76.
Boltcheva, D., Lévy, B., 2017. Surface reconstruction by computing restricted Voronoi cells in parallel. Comput. Aided Des. 90, 123–134.
Chen, L., Holst, M., 2011. Efficient mesh optimization schemes based on optimal Delaunay triangulations. Comput. Methods Appl. Mech. Eng. 200, 967–984.
Chen, Y., Ma, Z., Zhou, H., Zhang, W., 2021. 3D print-scan resilient localized mesh watermarking. In: 2021 IEEE International Workshop on Information Forensics and Security. WIFS. IEEE, pp. 1–6.
Chen, Z., Zhang, T., Cao, J., Zhang, Y.J., Wang, C., 2018. Point cloud resampling using centroidal Voronoi tessellation methods. Comput. Aided Des. 102, 12–21.
De Goes, F., Breeden, K., Ostromoukhov, V., Desbrun, M., 2012. Blue noise through optimal transport. ACM Trans. Graph. 31, 1–11.
Decker, N., Lyu, M., Wang, Y., Huang, Q., 2021. Geometric accuracy prediction and improvement for additive manufacturing using triangular mesh shape data. J. Manuf. Sci. Eng. 143.
Du, Q., Faber, V., Gunzburger, M., 1999. Centroidal Voronoi tessellations: applications and algorithms. SIAM Rev. 41, 637–676.
Du, Q., Gunzburger, M.D., Ju, L., 2003. Constrained centroidal Voronoi tessellations for surfaces. SIAM J. Sci. Comput. 24, 1488–1506.
Du, X., Liu, X., Yan, D.-M., Jiang, C., Ye, J., Zhang, H., 2018. Field-aligned isotropic surface remeshing. Comput. Graph. Forum 37, 343–357.

Engwirda, D., Ivers, D., 2014. Face-centred Voronoi refinement for surface mesh generation. Proc. Eng. 82, 8–20.
Fabri, A., Pion, S., 2009. CGAL: the computational geometry algorithms library. In: Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 538–539.
Fu, Y., Zhou, B., 2009. Direct sampling on surfaces for high quality remeshing. Comput. Aided Geom. Des. 26, 711–723.
Garcia, V., Debreuve, E., Barlaud, M., 2008. Fast k nearest neighbor search using GPU. In: 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops. IEEE, pp. 1–6.
Guo, J., Yan, D.-M., Jia, X., Zhang, X., 2015. Efficient maximal Poisson-disk sampling and remeshing on surfaces. Comput. Graph. 46, 72–79.
Herholz, P., Haase, F., Alexa, M., 2017. Diffusion diagrams: Voronoi cells and centroids from diffusion. Comput. Graph. Forum 36, 163–175.
Hou, W., Zong, C., Wang, P., Xin, S., Chen, S., Liu, G., Tu, C., Wang, W., 2022. SDF-RVD: restricted Voronoi diagram on signed distance field. Comput. Aided Des. 144, 103166.
Khan, D., Yan, D.-M., Ding, F., Zhuang, Y., Zhang, X., 2018a. Surface remeshing with robust user-guided segmentation. Comput. Vis. Media 4, 113–122.
Khan, D., Yan, D.-M., Wang, Y., Hu, K., Ye, J., Zhang, X., 2018b. High-quality 2D mesh generation without obtuse and small angles. Comput. Math. Appl. 75, 582–595.
Khan, D., Plopski, A., Fujimoto, Y., Kanbara, M., Jabeen, G., Zhang, Y.J., Zhang, X., Kato, H., 2022. Surface remeshing: a systematic literature review of methods and research directions. IEEE Trans. Vis. Comput. Graph. 28, 1680–1713.
Leung, Y.-S., Wang, X., He, Y., Liu, Y.-J., Wang, C.C., 2015. A unified framework for isotropic meshing based on narrow-band Euclidean distance transformation. Comput. Vis. Media 1, 239–251.
Lévy, B., Filbois, A., 2015. Geogram: a library for geometric algorithms. In: VII International Conference on Adaptive Modeling and Simulation. ADMOS 2015. CIMNE, p. 45.
Lévy, B., Liu, Y., 2010. L_p centroidal Voronoi tessellation and its applications. ACM Trans. Graph. 29, 1–11.
Liang, X., Zhang, Y., 2011. Hexagon-based all-quadrilateral mesh generation with guaranteed angle bounds. Comput. Methods Appl. Mech. Eng. 200, 2005–2020.
Liu, X., Ma, L., Guo, J., Yan, D.-M., 2022. Parallel computation of 3D clipped Voronoi diagrams. IEEE Trans. Vis. Comput. Graph. 28, 1363–1372.
Liu, Y., Wang, W., Lévy, B., Sun, F., Yan, D.-M., Lu, L., Yang, C., 2009. On centroidal Voronoi tessellation—energy smoothness and fast computation. ACM Trans. Graph. 28, 1–17.
Liu, Y.-J., Xu, C.-X., Yi, R., Fan, D., He, Y., 2016. Manifold differential evolution (MDE) a global optimization method for geodesic centroidal Voronoi tessellations on meshes. ACM Trans. Graph. 35, 1–10.
Lloyd, S., 1982. Least squares quantization in PCM. IEEE Trans. Inf. Theory 28, 129–137.
Ray, N., Sokolov, D., Lefebvre, S., Lévy, B., 2018. Meshless Voronoi on the GPU. ACM Trans. Graph. 37, 1–12.
Rong, G., Liu, Y., Wang, W., Yin, X., Gu, D., Guo, X., 2010. GPU-assisted computation of centroidal Voronoi tessellation. IEEE Trans. Vis. Comput. Graph. 17, 345–356.
Rong, G., Jin, M., Shuai, L., Guo, X., 2011. Centroidal Voronoi tessellation in universal covering space of manifold surfaces. Comput. Aided Geom. Des. 28, 475–496.
Rusu, R.B., Cousins, S., 2011. 3D is here: Point Cloud Library (PCL). In: 2011 IEEE International Conference on Robotics and Automation. IEEE, pp. 1–4.
Sainlot, M., Nivoliers, V., Attali, D., 2017. Restricting Voronoi diagrams to meshes using corner validation. Comput. Graph. Forum 36, 81–91.
Shazeer, N., Cheng, Y., Parmar, N., Tran, D., Vaswani, A., Koanantakool, P., Hawkins, P., Lee, H., Hong, M., Young, C., et al., 2018. Mesh-tensorflow: deep learning for supercomputers. Adv. Neural Inf. Process. Syst. 31.
Sullivan, C., Kaszynski, A., 2019. PyVista: 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK). J. Open Sour. Softw. 4, 1450.
Wang, P., Xin, S., Tu, C., Yan, D., Zhou, Y., Zhang, C., 2020. Robustly computing restricted Voronoi diagrams (RVD) on thin-plate models. Comput. Aided Geom. Des. 79, 101848.
Wang, X., Ying, X., Liu, Y.-J., Xin, S.-Q., Wang, W., Gu, X., Mueller-Wittig, W., He, Y., 2015. Intrinsic computation of centroidal Voronoi tessellation (CVT) on meshes. Comput. Aided Des. 58, 51–61.
Wang, Y., Yan, D.-M., Liu, X., Tang, C., Guo, J., Zhang, X., Wonka, P., 2018. Isotropic surface remeshing without large and small angles. IEEE Trans. Vis. Comput. Graph. 25, 2430–2442.
Xin, S.-Q., Lévy, B., Chen, Z., Chu, L., Yu, Y., Tu, C., Wang, W., 2016. Centroidal power diagrams with capacity constraints: computation, applications, and extension. ACM Trans. Graph. 35, 1–12.
Xu, M., Xin, S., Tu, C., 2019. An efficient surface remeshing algorithm based on centroidal power diagram. In: Proceedings of the 2019 7th International Conference on Information Technology: IoT and Smart City, pp. 536–542.
Yan, C., Qixiang, C., 2009. A novel parallel processing for continuous k-nearest neighbor queries. In: 2009 International Conference on Environmental Science and Information Application Technology. IEEE, pp. 593–596.
Yan, D.-M., Wonka, P., 2013. Gap processing for adaptive maximal Poisson-disk sampling. ACM Trans. Graph. 32, 1–15.
Yan, D.-M., Wonka, P., 2015. Non-obtuse remeshing with centroidal Voronoi tessellation. IEEE Trans. Vis. Comput. Graph. 22, 2136–2144.
Yan, D.-M., Lévy, B., Liu, Y., Sun, F., Wang, W., 2009. Isotropic remeshing with fast and exact computation of restricted Voronoi diagram. Comput. Graph. Forum 28, 1445–1454.
Yan, D.-M., Wang, W., Lévy, B., Liu, Y., 2010. Efficient computation of 3D clipped Voronoi diagram. In: International Conference on Geometric Modeling and Processing. Springer, pp. 269–282.
Yan, D.-M., Wang, W., Lévy, B., Liu, Y., 2013. Efficient computation of clipped Voronoi diagram for mesh generation. Comput. Aided Des. 45, 843–852.
Yan, D.-M., Bao, G., Zhang, X., Wonka, P., 2014. Low-resolution remeshing using the localized restricted Voronoi diagram. IEEE Trans. Vis. Comput. Graph. 20, 1418–1427.
Yang, X., Koehl, M., Grussenmeyer, P., 2018. Mesh-to-BIM: from segmented mesh elements to BIM model with limited parameters. In: The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol. 42. Copernicus Publications, pp. 1213–1218.
Ye, Z., Yi, R., Yu, M., Liu, Y.-J., He, Y., 2019. Geodesic centroidal Voronoi tessellations: theories, algorithms and applications. arXiv preprint. arXiv:1907.00523.
You, Y., Kou, X., Tan, S., 2015. Adaptive meshing for finite element analysis of heterogeneous materials. Comput. Aided Des. 62, 176–189.
Zheng, L., Yao, Y., Wu, W., Xu, B., Zhang, G., 2021. A novel computation method of hybrid capacity constrained centroidal power diagram. Comput. Graph. 97, 108–116.
Zhuang, Y., Zou, M., Carr, N., Ju, T., 2014. Anisotropic geodesics for live-wire mesh segmentation. Comput. Graph. Forum 33, 111–120.
Zimmer, H., Campen, M., Herkrath, R., Kobbelt, L., 2012. Variational tangent plane intersection for planar polygonal meshing. In: AAG, pp. 319–332.