Efficient computation of clipped Voronoi diagram for mesh generation

Dong-Ming Yan, Wenping Wang, Bruno Lévy, Yang Liu

Efficient Computation of Clipped Voronoi Diagram for Mesh Generation

Dong-Ming Yan[a,b,c], Wenping Wang[a], Bruno Lévy[b], Yang Liu[b,d]

[a] *Department of Computer Science, The University of Hong Kong, Pokfulam Road, Hong Kong*
[b] *Project ALICE, INRIA/LORIA, Campus scientifique 615, rue du Jardin Botanique, 54600, Villers les Nancy, France*
[c] *Geometric Modeling and Scientific Visualization Center, KAUST, Thuwal 23955-6900, Kingdom of Saudi Arabia*
[d] *Microsoft Research Asia, Building 2, No. 5 Danling Street, Haidian District, Beijing, 100800, P.R. China*

Abstract

The Voronoi diagram is a fundamental geometric structure widely used in various fields, especially in computer graphics and geometry computing. For a set of points in a compact domain (i.e. a bounded and closed 2D region or a 3D volume), some Voronoi cells of their Voronoi diagram are infinite or partially outside of the domain, but in practice only the parts of the cells inside the domain are needed, as when computing the centroidal Voronoi tessellation. Such a Voronoi diagram confined to a compact domain is called a clipped Voronoi diagram. We present an efficient algorithm to compute the clipped Voronoi diagram for a set of sites with respect to a compact 2D region or a 3D volume. We also apply the proposed method to optimal mesh generation based on the centroidal Voronoi tessellation.

Keywords: clipped Voronoi diagram, Delaunay triangulation, centroidal Voronoi tessellation, mesh generation.

1. Introduction

The Voronoi diagram is a fundamental geometric structure which has numerous applications in various fields, such as shape modeling, motion planning, scientific visualization, geography, chemistry, biology and so on.

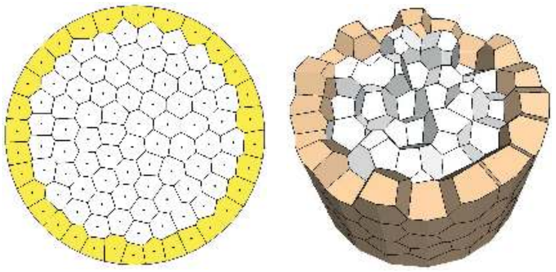
Suppose that a set of sites in a compact domain in $\mathbb{R}^d$ is given. Each site is associated with a Voronoi cell containing all the points in $\mathbb{R}^d$ closer to the site than to any other sites; these cells constitute the Voronoi diagram of the set of sites. Voronoi cells of those sites on the convex hull are infinite, and some of Voronoi cells may be partially outside of the specified domain. However, in many applications one usually needs only the parts of Voronoi cells inside the specific domain. That is, the Voronoi diagram restricted to the given domain, which is defined as the intersection of the Voronoi diagram and the domain, and is therefore called the *clipped Voronoi diagram* [1]. The corresponding Voronoi cells are called the *clipped Voronoi cells* (see Figure 1).

Computing the clipped Voronoi diagram in a convex domain is relatively easy – one just needs to compute the intersection of each Voronoi cell and the domain, both being convex. However, directly computing the clipped Voronoi diagram with respect to a complicated input domain is a difficult problem and there is no efficient solution in the existing literature. There has been no previous work on computing the exact clipped Voronoi diagram for non-convex domains with arbitrary topology. A brute-force implementation would be inefficient because of the complexity of the domain.

The motivation of the work is inspired by the recent work [2, 3]. They showed in [2] that the CVT energy function is C^2-continuous, which can be minimized by the Newton-like algorithm, such as the L-BFGS method presented. In [3], an efficient CVT-based surface remeshing algorithm was presented with an exact algorithm for computing the restricted Voronoi diagram on mesh surfaces. In this paper, we aim at applying the fast CVT remeshing framework to 2D/3D mesh generation. To minimize the CVT energy function, one needs to compute the clipped Voronoi diagram in the input domain for function evaluation and gradient computation (see Section 2).

In this paper, we shall present practical algorithms for computing clipped Voronoi diagrams based on several simple operations. The main idea

Preprint submitted to Computer-Aided Design *October 19, 2011*



(a) (b)

Figure 1: Examples of clipped Voronoi diagram in a circle (a) and a cylinder (b). The clipped Voronoi cells on the boundary are shaded.

of our approach is that instead of computing the intersection of Voronoi diagram and the domain directly, we first detect the Voronoi cells that have intersections with domain boundary and then apply computation for those cells only. We use a simple and efficient algorithm based on connectivity propagation for detecting the cells that intersect with the domain boundary (i.e., polygons in 2D and mesh surfaces in 3D, respectively). We also utilize the presented techniques for mesh generation as applications. The contributions of this paper include :

- introduce new methods for computing the clipped Voronoi diagram in 2D regions (Section 3) and 3D volumes (Section 4);
- present practical algorithms for 2D/3D mesh generation based on the presented clipped Voronoi diagram computation techniques (Section 5).

1.1. Previous work

The properties of the Voronoi diagram have been extensively studied in the past decades. Existing techniques compute the Voronoi diagram for point sites in 2D and 3D Euclidean spaces efficiently. There are several robust implementations that are publicly available, such as CGAL [4] and Qhull [5]. A thorough survey of the Voronoi diagram is out of the scope of this paper, the reader is referred to [6, 7, 8] for details of theories and applications of the Voronoi diagram. We shall restrict our discussion to the approaches of computing the Voronoi diagram restricted to a specific 2D/3D domain and their applications.

Voronoi diagram of surfaces/volumes. It is natural to use the geodesic metric to define the so called Geodesic Voronoi Diagram (GVD) on surfaces. Kunze *et al.* [9] presented a divide-and-conquer algorithm of computing GVD for parametric surfaces. Peyré and Cohen [10] used the fast marching algorithm to compute a discrete approximated GVD on a mesh surface. However, the cost of computing the exact GVD on surfaces is high, for instance, the fast marching method requires to solve the nonlinear Eikonal equation.

The restricted Voronoi diagram (RVD) [11] is defined as the intersection of the 3D Voronoi diagram and the surface, which is applied for computing constrained/restricted CVT on continuous surfaces by Du *et al.* [12]. The concept of the constrained CVT was extended to mesh surfaces in recent work [2, 3] and applied for isotropic surface remeshing. Yan *et al.* [3] proposed an exact algorithm to construct the RVD on mesh surfaces which consist of triangle soups. They processed each triangle independently where a *kd*-tree was used to find the nearest sites of each triangle in order to identify its incident Voronoi cells and compute the intersection. In this paper, we further improve the efficiency of the RVD computation by applying a neighbor propagation approach instead of using *kd*-tree query, assuming the availability of the mesh connectivity information (Section 4.1).

The clipped Voronoi diagram is defined as the intersection of the 3D (resp. 2D) Voronoi diagram and the given 3D volume (resp. a 2D region). Chan *et al.* [1] introduced an output-sensitive algorithm for constructing the 3D clipped Voronoi diagram of a convex polytope. Kyons *et al.* [13] presented an $O(n\log(n))$ algorithm to compute the clipped Voronoi diagram in a 2D square and applied it to network visualization. Yan *et al.* [14] utilized the clipped Voronoi diagram to compute the periodic CVT in 2D periodic space. Hudson *et al.* [15] computed the 3D clipped Voronoi diagram in the bounding box of the sites and used it to improve the time and space complexities of computing the full persistent homological information. However, the handling of non-convex objects was not addressed in these approaches. Existing algorithms used a discrete approximation in specific applications. Hoff III *et al.* [16] proposed a method for computing the discrete generalized Voronoi diagram using graphics hardware. The Voronoi diagram computation was formulated as a clustering problem in the discrete voxel/pixel space. Sud *et al.* [17] presented an

n-body proximity query algorithm based on computing the discrete 2^{nd} order Voronoi diagram on the GPU. GPU-based algorithms were fast but produced only a discrete approximation of the true Voronoi diagram. In this paper, we shall present efficient algorithms to compute the exact clipped Voronoi diagram for both 2D and 3D domains.

Mesh generation. Mesh generation has been extensively studied in meshing community over past decades. The detailed reviews of mesh generation techniques are available in [18, 19]. In the following, we will focus on the work based on Voronoi/Delaunay concepts, which are most related to ours. We also briefly review the main categories of tetrahedral mesh generation techniques.

The concept of Voronoi diagram has been successfully used for meshing and analyzing point data. Amenta *et al.* [20] presented a new surface reconstruction algorithm based on Voronoi filtering. This algorithm has provable guarantees when the sample points of a smooth surface satisfy the *lfs* (local feature size) property. Alliez *et al.* [21] proposed a surface reconstruction algorithm from noisy input data based on the Voronoi-PCA estimation. Leymarie and Kimia introduced the medial scaffold of point cloud data [22], which is a hierarchical representation of the medial axis of 3D objects. Although these works deal with point data, they can be extended further for volumetric meshing.

The medial axis, which is a subset of Voronoi diagram, has been applied in applications such as 2D quadrilateral meshing [23] and 3D hexahedral meshing [24]. Given a closed 2D polygon or 3D triangulated surface as the input domain, a set of dense points is first sampled on the domain boundary and the medial axis/surface is computed directly from the Voronoi diagram of samples. The final mesh is generated by first meshing the medial axis(2D)/surface(3D) and extruding to the domain boundary [25]. The medial axis based method is suitable for models which have well defined medial axis, such as CAD/CAM models, but the medial axis computation is sensitive to noise or small features of the domain boundary.

In this paper, we focus on the tetrahedral meshing as an application of the clipped Voronoi diagram computation (see Section 5). The shape quality and boundary preservation are two main issues of tetrahedral meshing algorithms, since the quality of simplices is crucial to finite element applications. We refer the reader to [26] for the theoretic study of the relationship between element qualities and interpolation error/condition number. In the following, we briefly discuss the main categories of tetrahedral meshing.

- The *octree-based* approaches (e.g. [27, 28]) subdivide the bounding box of input model repeatedly until a pre-specified resolution is reached, then connect those cells to form the tetrahedra. In general, this kind of approaches cannot prevent bad elements near the boundary.

- *Advancing front* methods start from the domain boundary and stuff the interior of the domain progressively, guided by specified heuristic to control the shape/size. Advancing front methods are fast but a high-quality triangulated boundary is required.

- *Delaunay/Voronoi* based approaches generate meshes satisfying Delaunay properties, which maximize the minimal angle of shape elements. Given an input domain, Delaunay/Voronoi based methods repeatedly insert Steiner points into the mesh, until all the elements meet the Delaunay property. This approach aims at generating meshes which conform to the input domain boundary, but often leads to unsatisfied results if the given domain boundary is poorly triangulated. An alternative way is to approximate the boundary instead of conforming, which results the better shape/size quality.

- *Variational approach* is one of the most effective ways of generating isotropic tetrahedral meshes. Recent work includes both CVT-based and ODT-based techniques. The CVT-based approach aims at optimizing the dual Voronoi structure of Delaunay triangulation, while ODT tends to optimize the shape of primal elements [29]. The CVT-based mesh generation has been extensively studied in the literature [12], while ODT was recently introduced to graphics community [30, 31]. One of the main difficulties of both CVT and ODT-based tetrahedral meshing is the boundary conforming issue. Alliez *et al.* [30] used dense quadrature samples to approximate restricted Voronoi cells on mesh surface. Dardenne *et al.* [32] used a discrete version of the CVT to generate tetrahedral meshes from the discrete volume data. The voxels are clustered into n

cells via Lloyd iteration, with each cell corresponding to a site. The tetrahedral mesh is obtained from the connectivity relations of cells. However, such an approach is limited to the resolution of voxels.

2. Problem Formulation

We first provide mathematical definitions and notations, then introduce the main idea of the clipped Voronoi diagram computation.

2.1. Definitions

Definition 2.1. *The Voronoi Diagram of a given set of distinct sites* $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^n$ *in* $\mathbb{R}^d$ *is defined by a collection of Voronoi cells* $\{\Omega_i\}_{i=1}^n$*, where*

$$\Omega_i = \{\mathbf{x} \in \mathbb{R}^d \mid \|\mathbf{x} - \mathbf{x}_i\| \leq \|\mathbf{x} - \mathbf{x}_j\|, \forall j \neq i\}.$$

Each Voronoi cell Ω_i is the intersection of a set of half-spaces, delimited by the bisectors of the Delaunay edges incident to the site $\mathbf{x}_i$.

Definition 2.2. *The Clipped Voronoi Diagram for the sites* $\mathbf{X}$ *with respect to a connected compact domain* Ω *is the intersection of the Voronoi diagram and the domain, denoted as* $\{\Omega_i|_\Omega\}_{i=1}^n$*, where*

$$\Omega_i|_\Omega = \{\mathbf{x} \in \Omega \mid \|\mathbf{x} - \mathbf{x}_i\| \leq \|\mathbf{x} - \mathbf{x}_j\|, \forall j \neq i\}$$

Each clipped Voronoi cell is the intersection of the Voronoi cell Ω_i and the domain Ω, i.e., $\Omega_i|_\Omega = \Omega_i \bigcap \Omega$. We call $\Omega_i|_\Omega$ the *clipped Voronoi cell* with respect to Ω (see Figure 1 for examples).

Definition 2.3. *Centroidal Voronoi Tessellation of a set of distinct sites* $\mathbf{X}$ *with respect to a compact domain* Ω *is the minimizer of the CVT energy function [33] :*

$$F(\mathbf{X}) = \sum_{i=1}^n \int_{\Omega_i|_\Omega} \rho(\mathbf{x}) \|\mathbf{x} - \mathbf{x}_i\|^2 \, \mathrm{d}\sigma. \qquad (1)$$

In the above definition, $\rho(\mathbf{x}) > 0$ is a user-defined density function. The partial derivative of the energy function with respect to each site is given by [34] :

$$\frac{\partial F}{\partial \mathbf{x}_i} = 2m_i(\mathbf{x}_i - \mathbf{x}_i^*), \qquad (2)$$

here $m_i = \int_{\Omega_i|_\Omega} \rho(\mathbf{x}) \, \mathrm{d}\sigma$, and $x_i^* = \frac{\int_{\Omega_i|_\Omega} \rho(\mathbf{x})\mathbf{x} \, \mathrm{d}\sigma}{\int_{\Omega_i|_\Omega} \rho(\mathbf{x}) \, \mathrm{d}\sigma}$ is the centroid of the clipped Voronoi cell $\Omega_i|_\Omega$. We use the L-BFGS method [2] for computing the CVT. The clipped Voronoi diagram is used to assist the function evaluation (Eqn. 1) and the gradient computation (Eqn. 2).

2.2. Algorithm overview

There are two types of clipped Voronoi cells of a clipped Voronoi diagram : *inner Voronoi cells* and *boundary Voronoi cells*, whose corresponding sites are called inner sites and boundary sites, respectively. The *inner Voronoi cells* are entirely contained in the interior of the domain Ω, which can be deduced from the Delaunay triangulation directly. The *boundary Voronoi cells* are those cells that intersect with the domain boundary $\partial\Omega$, as shown in Figure 1. In the following, we will focus on how to compute the boundary Voronoi cells.

To compute a clipped Voronoi diagram with respect to a given domain, we first need to classify the sites into inner and boundary sites, and then compute the clipped Voronoi cells for boundary sites. As discussed above, the boundary cells have intersections with the domain boundary $\partial\Omega$ (i.e., polygons in 2D and mesh surfaces in 3D), which can be found by intersecting the boundary with the Voronoi diagram. We present efficient algorithms for computing the intersection of a Voronoi diagram and 2D polygons or 3D mesh surfaces, respectively. Once the boundary sites are identified, we are able to compute the clipped Voronoi cells efficiently by clipping the domain Ω against boundary Voronoi cells.

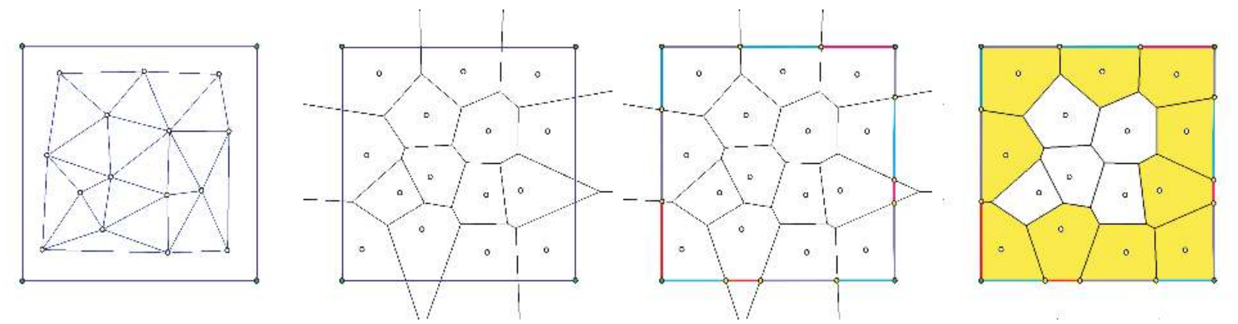
In the following sections, we shall present efficient algorithms for computing clipped Voronoi diagram in 2D (Section 3) and 3D (Section 4) spaces, respectively. Furthermore, we show how to utilize the presented clipped Voronoi diagram computation techniques for practical mesh generation (Section 5).

3. 2D Clipped Voronoi Diagram Computation

Suppose that the input domain Ω is a compact 2D region, whose boundary is represented by a 2D counter-clockwise outer polygon, and several clockwise inner polygons without self-intersections. Assume that the boundary is represented by a set of ordered edge segments $\{e_i\}$. The main steps of our method are illustrated in Figure 2. For a given set of sites inside the given domain, we first compute the Voronoi diagram of the sites. Then we identify the boundary sites and finally compute the clipped Voronoi cells of boundary sites .

3.1. Voronoi diagram construction

We first construct a Delaunay triangulation from input sites $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^n$. The corresponding



(a) (b) (c) (d)

Figure 2: Illustration of main steps for computing clipped Voronoi diagram in 2D. (a) Delaunay triangulation, (b) 2D Voronoi diagram, (c) detect boundary sites, (d) compute clipped Voronoi diagram.

Voronoi diagram $\{\Omega_i\}_{i=1}^n$ is constructed as the dual of the Delaunay triangulation, as defined in Section 2. Each Voronoi cell is stored as a set of bisecting planes, which is used for clipping operations in the following steps.

3.2. Detection of boundary cells

In this step, we shall identify the boundary Voronoi cells by computing the intersection of boundary edges and the Voronoi diagram $\{\Omega_i\}$. We repeatedly find the incident cell-edge pairs with the assistance of an FIFO queue. An incident Voronoi cell of a boundary edge e_i is the cell that intersects with e_i, i.e., a boundary Voronoi cell.

We assign a boolean tag to each boundary edge e_i which indicates whether e_i has been processed or not. This flag is initialized as `false`. Once the edge is visited, the flag is switched to `true`. Starting from an unvisited boundary edge e_i, we first find its nearest incident Voronoi cell Ω_j, then use the barycenter (or midpoint) of e_i to query the nearest site $\mathbf{x}_j$. Any linear search function can be used here for the nearest point query.

The FIFO queue is initialized by the initial incident cell-edge pair (Ω_j, e_i). We repeatedly pop out the cell-edge pair from the queue and compute the intersection of the current Voronoi cell Ω_c and the boundary edge e_c. The intersected segment is denoted as s_c. The current boundary edge is marked as `visited` and the current Voronoi cell is marked as `boundarycell`. We detect new cell-edge pairs by examining the current intersected segment s_c. There are two cases of s_c's endpoints :

(a) if the endpoints of s_c contain a boundary vertex of the current edge e_c (green dots in Figure 2(c)), the adjacent boundary edge who shares the same vertex with e_c is pushed into the queue together with the current Voronoi cell Ω_c;

(b) if the endpoints of s_c contain an intersection point, i.e., the intersection point between a Voronoi edge of Ω_c and e_c (yellow dots in Figure 2(c)), the neighboring Voronoi cell who shares the intersecting Voronoi edge with Ω_c is pushed into the queue together with e_c.

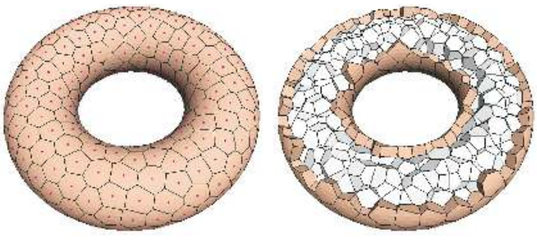
The boundary detection process terminates when all the edges have been visited.

3.3. Computation of clipped Voronoi cells

Once the boundary sites are identified, we compute the clipped Voronoi cells by clipping the domain against their corresponding bounding line segments. A straightforward extension of [3] should first triangulate the boundary polygons and then do computation on the resulting planar mesh, which will be the same as the surface RVD computation described in Section 4.1. Given that the average number of bisectors of 2D Voronoi cells is six [33], it is efficient enough to clip the 2D domain by Voronoi cells directly. Here we simply use the Sutherland-Hodgman clipping algorithm [35] to compute the intersection. More examples of 2D clipped Voronoi diagram are given in Section 6.

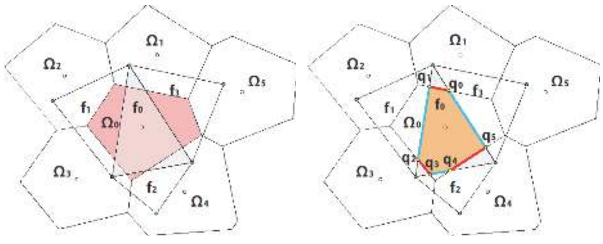
4. 3D Clipped Voronoi Diagram Computation

In this section we describe an efficient algorithm for computing the clipped Voronoi diagram of 3D objects. Suppose that the input volume Ω is given by a tetrahedral mesh $\mathcal{M} = \{\mathcal{V}, \mathcal{T}\}$, where $\mathcal{V} =$



(a) (b)

Figure 3: Illustration of clipped Voronoi diagram computation of 500 sites in a torus. (a) Surface RVD of 227 boundary sites, (b) Clipped Voronoi diagram.



(a) (b)

Figure 4: Illustration of the propagation process. The green points are the vertices of input boundary mesh and the white points are the sites. The yellow points in (b) are the vertices of RVD.

$\{\mathbf{v}_k\}_{k=1}^{n_v}$ is the set of mesh vertices and $\mathcal{T} = \{\mathbf{t}_i\}_{i=1}^{m}$ the set of tetrahedral elements. Each tetrahedron (*tet* for short in the following) $\mathbf{t}_i$ stores the information of its four incident vertices and four adjacent tets. The four vertices are assigned indices $0, 1, 2, 3$ and so are the four adjacent tets. The index of an adjacent tet is the same as the index of the vertex which is opposite to the tet. The boundary of $\mathcal{M}$ is a triangle mesh, denoted as $\mathcal{S} = \{\mathbf{f}_j\}_{j=1}^{n_f}$, which is assumed to be a 2-manifold. Each boundary triangle facet $\mathbf{f}_j$ stores the indices of three neighboring facets and the index of its containing tet. Note that although other types of convex primitives can also be used for domain decomposition, we use tetrahedral mesh here for simplicity.

The 3D clipped Voronoi diagram computation is similar to the 2D counterpart. After constructing the 3D Voronoi diagram $\{\Omega_i\}$ of the sites $\mathbf{X}$ (see Section 3.1), there are two main steps, as illustrated in Figure 3 :

1. detect boundary sites by intersecting Voronoi diagram with the boundary surface $\mathcal{S}$, i.e., compute the surface RVD (Section 4.1);

2. compute the clipped Voronoi cells for all the boundary sites (Section 4.2).

4.1. Detection of boundary sites

For the given set of sites $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^{n}$ and the boundary surface $\mathcal{S} = \{\mathbf{f}_j\}_{j=1}^{n_f}$, the restricted Voronoi diagram (RVD) is defined as the intersection of the 3D Voronoi diagram and the surface $\mathcal{S}$, denoted as $\mathcal{R} = \{\mathcal{R}_i\}_{i=1}^{n}$, where $\mathcal{R}_i = \Omega_i \bigcap \mathcal{S}$ [11]. Each $\mathcal{R}_i$ is called a *restricted Voronoi cell* (RVC). The sites corresponding to non-empty RVCs are regarded as boundary sites.

We use the algorithm presented in [3] for computing the surface RVD. The performance of RVD computation is improved by using a neighbor propagation approach for finding the incident cell-triangles pairs, instead of using a *kd*-tree structure to query the nearest site for each triangle, as shown by our tests.

Now we are going to explain the propagation step (refer to Figure 4). We assign a boolean flag (initialized as `false`) for each boundary triangle at the initialization step. The flag is used to indicate whether a triangle is processed or not. Starting from an unprocessed triangle and one of its incident cells, which is the cell corresponding to the nearest site of the triangle by using the barycenter of the triangle as the query point. Here we assume that a triangle $\mathbf{f}_0$ on $\mathcal{S}$ is the unprocessed triangle and the Voronoi cell Ω_0 is the corresponding cell of the nearest site of $\mathbf{f}_0$, as shown in Figure 4(a). We use an FIFO queue $\mathcal{Q}$ to store all the incident cell-triangle pairs to be processed. To start, the initial pair $\{\mathbf{f}_0, \Omega_0\}$ is pushed into the queue. The algorithm repeatedly pops out the pair in the front of $\mathcal{Q}$ and computes their intersection. During the intersection process, the current triangle is marked as `processed`, new valid pairs are identified and pushed back into $\mathcal{Q}$. The process terminates when $\mathcal{Q}$ is empty and all the triangles are processed.

The key issue now is how to identify all the valid cell-triangle pairs during the intersection. Assume that $\{\mathbf{f}_0, \Omega_0\}$ is popped out from $\mathcal{Q}$, as shown in Figure 4. In this case, we clip $\mathbf{f}_0$ against the bounding planes of Ω_0, which has five bisecting planes, i.e.,$[\mathbf{x}_0, \mathbf{x}_1]$, $[\mathbf{x}_0, \mathbf{x}_2]$..., $[\mathbf{x}_0, \mathbf{x}_5]$. The resulting polygon is represented by $\mathbf{q}_0, \mathbf{q}_1, ..., \mathbf{q}_5$, as shown in Figure 4(b). Since the line segment $\overline{\mathbf{q}_0\mathbf{q}_1}$ is the intersection of $\mathbf{f}_0$ and $[\mathbf{x}_0, \mathbf{x}_1]$, we know that the oppo-

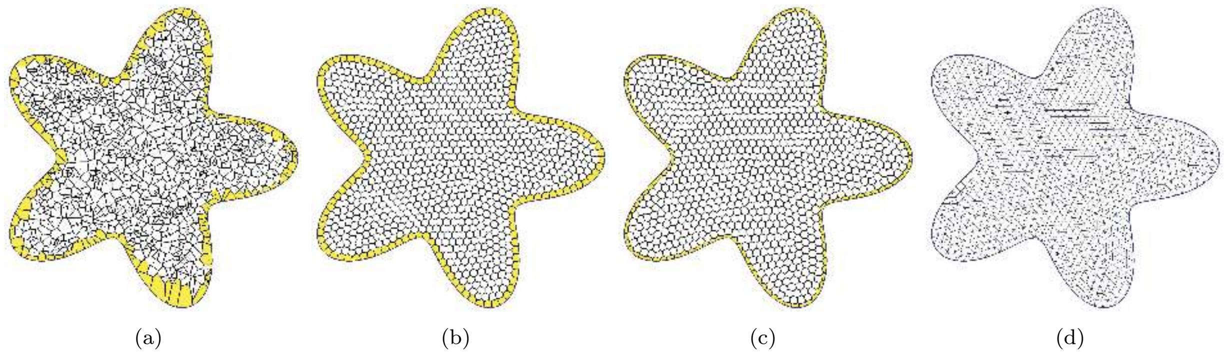


Figure 5: 2D CVT-based meshing. (a) The clipped Voronoi diagram of initial sites; (b) the result of CVT with $\rho = 1$; (c) the result of constrained optimization. Notice that boundary seeds are constrained on the border; (d) the final uniform 2D meshing.

site cell Ω_1 is also an incident cell of $\mathbf{f}_0$, thus the pair $\{\mathbf{f}_0, \Omega_1\}$ is an incident pair. Since the common edge of $[\mathbf{f}_0, \mathbf{f}_1]$ has intersection with Ω_0, the adjacent facet $\mathbf{f}_1$ also has intersection with cell Ω_0, thus the pair $\{\mathbf{f}_1, \Omega_0\}$ is also an incident pair. So is the pair $\{\mathbf{f}_2, \Omega_0\}$. The other incident pairs are found in the same manner. To keep the same pair from being processed multiple times, we store the incident facet indices for each cell. Before pushing a new pair into the queue, we add the facet index to the incident facet index set of the cell. The pair is pushed into the queue only if the facet is not contained in the incident facet set of the cell; otherwise the pair is discarded. At each time after intersection computation, the resulting polygon is associated with the surface RVC of the current site. The surface RVD computation terminates when the queue is empty. Those sites that have non-empty surface RVC are marked as the boundary sites, denoted as $\mathbf{X}_b = \{\mathbf{x}_i | \mathcal{R}_i \neq \emptyset\}$.

4.2. Construction of clipped Voronoi cells

Once the boundary sites $\mathbf{X}_b$ are found, we compute the clipped Voronoi cells for these sites. The computation of boundary Voronoi cells is similar to the surface RVD computation presented in Section 4.1, with the difference that we restrict the computation on boundary cells only. For each boundary cell, we have recorded the indices of its incident boundary triangles. We know that the neighboring tet of each boundary triangle is also incident to the cell. We also store the indices of the incident tet for each boundary cell. The incident tet set is initialized as the neighboring tet of the incident boundary triangle.

We use an FIFO queue to facilitate this process. The queue is initialized by a set of incident cell-tet pairs $(\Omega_i, \mathbf{t}_j)$, which can be obtained from the boundary cell and its initial incident tet set.

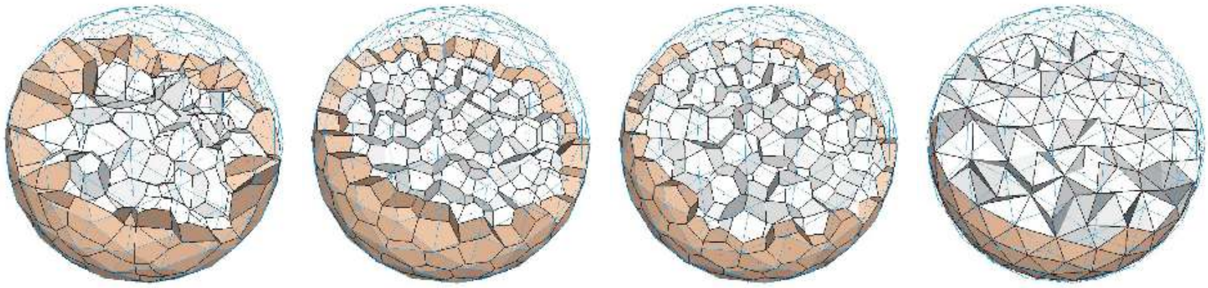
The pair $(\Omega_i, \mathbf{t}_j)$ in front of $\mathcal{Q}$ is popped out repeatedly. We compute the intersection of Ω_i and $\mathbf{t}_j$ again by the Sutherland-Hodgman clipping algorithm [35] and identify new incident pairs at the same time. We clip the tet $\mathbf{t}_j$ by bounding planes of cell Ω_i one by one. If the current bounding plane has intersection with $\mathbf{t}_j$, we check the opposite Voronoi cell Ω_o that shares the current bisecting plane with Ω_i; if Ω_o is a boundary cell and $\mathbf{t}_j$ is not in the incident set of Ω_o, a new pair $(\Omega_o, \mathbf{t}_j)$ is found. We also check the neighboring tets who share the facets clipped by the current bisecting plane. Those tets that are not in the incident set of Ω_i are added to its set, and new pairs are pushed into the queue. After clipping, the resulting polyhedron is associated with the clipped Voronoi cell $\Omega_i|_{\mathcal{M}}$ of site $\mathbf{x}_i$. This process terminates when $\mathcal{Q}$ is empty.

5. Applications for mesh generation

We present two applications of the presented clipped Voronoi diagram computation techniques, including 2D triangular meshing and 3D tetrahedral meshing.

5.1. 2D mesh generation

Triangle mesh generation is a well-known application of CVT optimization. In this section we present such an application based on our 2D clipped Voronoi diagram computation. The input domain



(a) (b) (c) (d)

Figure 6: Illustration of the CVT-based tetrahedral meshing algorithm. The wireframe is the boundary of the input mesh. (a) The clipped Voronoi diagram of the initial sites (the boundary Voronoi cells are shaded); (b) the result of the unconstrained CVT with $\rho = 1$; (c) the result of the constrained optimization. Notice that boundary seeds are constrained on the surface $\mathcal{S}$; (d) the final isotropic tetrahedral meshing result.

Ω is a 2D polygon, which can be single connected or with multiple components. We first sample a set of initial points inside the input domain (Figure 5(a)) and then compute a CVT (Eqn. 1) from this initial sampling (Figure 5(b)). Once we have a set of well distributed samples, we snap the seeds corresponding to boundary Voronoi cells to the boundary and run optimization again, with the boundary seeds constrained on the border (Figure 5(c)). Finally, we keep the primal triangles whose circumscribing centers are inside the domain as the meshing result (5(d)). Our 2D meshing framework also allows the user to insert vertices of input polygon and tag these vertices as fixed. By doing this, the geometric properties of the input domain can be better preserved. More results are given in Section 6.

5.2. Tetrahedral mesh generation

There are three main steps of the CVT-based meshing framework: initialization, iterative optimization, and mesh extraction, which are illustrated by the example in Figure 6.

Initialization. In this step, we build a uniform grid to store the sizing field for adaptive meshing. Following the approach in [30], we first compute the *local feature size* (lfs) for all boundary vertices and then use a fast matching method to construct a sizing field on the grid. This grid is also used for efficient initial sampling (Figure 6(a)). The reader is referred to [30] for details.

Optimization. There are two phases of the global optimization: the unconstrained CVT optimization and the constrained CVT optimization. In the first phase, we optimize the positions of the sites inside the input volume without any constraints, which yields a well-spaced distribution of the sites within the domain, with no sites lying on the boundary surface (Figure 6(b)).

During the second phase of optimization, all the boundary sites will be constrained on the boundary. The partial derivative of the energy function with respect to each boundary site is computed as:

$$\left.\frac{\partial F}{\partial \mathbf{x}_i}\right|_{\mathcal{S}} = \frac{\partial F}{\partial \mathbf{x}_i} - \left[\frac{\partial F}{\partial \mathbf{x}_i} \cdot \mathbf{N}(\mathbf{x}_i)\right]\mathbf{N}(\mathbf{x}_i), \qquad (3)$$

where $\mathbf{N}(\mathbf{x}_i)$ is the unit normal vector of the boundary surface at the boundary site $\mathbf{x}_i$ [2]. The partial derivative with respect to an inner site is still computed by Eqn. 2. Both boundary and inner sites will be optimized simultaneously, applying again the L-BFGS method to minimize the CVT energy function (Figure 6(c)).

Sharp features are preserved in a similar way as how the boundary sites are treated. For example, we project sites on sharp edges on the boundary and allow them to vary only along these edges during the second stage of optimization. For details, please refer to [3] where these steps are described in the context of surface remeshing.

Final mesh extraction. Once the optimization is finished, we extract the tetrahedral cells from the primal Delaunay triangulation (Figure 6(d)). As discussed in [30], the CVT energy cannot eliminate the slivers from the resulting tetrahedral mesh. We perform a post-processing to perturb slivers using the approach of [36]. The results are given in Section 6.

Figure 7: Results of clipped Voronoi diagram computation.

6. Experimental results

Our algorithm is implemented in C++ on both Windows and Linux platform. We use the CGAL library [4] for 2D and 3D Delaunay triangulation and TetGen [37] for background mesh generation when the input 3D domain is given as a closed triangle mesh. All the experimental results are tested on a laptop with 2.4GHz processor and 2GB memory.

Efficiency. We first demonstrate the performance of the proposed clipped VD computation algorithm. The 2D version is very efficient. All the examples shown in this paper take only several milliseconds. To detect the boundary sites, we have implemented a propagation based approach for surface RVD computation. This new implementation of RVD performs better then the previous kd-tree based approach [3] since there is no kd-tree query required, as shown in Figure 8. The performance of the 3D clipped Voronoi diagram computation is demonstrated in Figure 9. We progressively sample the input domain with number of sites from 10 to 6×10^5. Note that the time of surface RVD computation is much less than the Delaunay triangulation, since only a small portion of all the sites are boundary sites. The time cost of the clipped VD computation algorithm is proportional to the total number of incident cell-tet pairs (Section 4.2). Therefore, an input mesh with a small number of tetrahedral elements would help to improve the efficiency. In our experiments, all the input tetrahedral meshes are generated by the robust meshing software TetGen [37] with the conforming boundary. More results of the clipped Voronoi diagram computation of various 3D objects are given in Figure 7 and the timing statistics is given in Table 1.

Model	$\|\mathcal{T}\|$	$\|\mathcal{S}\|$	$\|\mathbf{X}\|$	$\|\mathbf{X}_b\|$	Time
Twoprism	68	30	1k	572	0.2
Bunny	10k	3k	2k	734	1.8
Elk	34.8k	10.4k	2k	1,173	3.1
Block	77.2k	23.4	1k	659	4.7
Homer	16.2k	4,594	10k	2,797	6.3
Rockerarm	212k	60.3k	3k	1,722	12.1
Bust	68.5k	20k	30k	5k	16.2

Table 1: Statistics of clipped Voronoi diagram computation on various models. $|\mathcal{T}|$ is the number of the input tetrahedra. $|\mathcal{S}|$ is the number of the boundary triangles. $|\mathbf{X}|$ is the number of the sites. $|\mathbf{X}_b|$ is the number of the boundary sites. Time (in seconds) is the total time for clipped Voronoi diagram computation, including both Delaunay triangulation and surface RVD computation.

Robustness. We use exact predicates to predicate the side of a vertex against a Voronoi plane during the clipping process. We use Meyer and Pion's FGP predicate generator [38] provided by CGAL in our implementation, as also done in [3]. We did not encounter any numerical issue for all the examples shown in the paper. Our clipped Voronoi diagram is robust even for extreme configurations. We show an example of computing the clipped Voronoi diagram on a sphere in Figure 10. The sites are set to the vertices of the boundary mesh and there is no

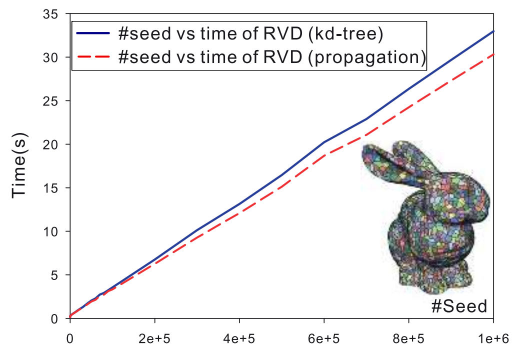


Figure 8: Comparison of the propagation-based surface RVD computation with the kd-tree-based approach.

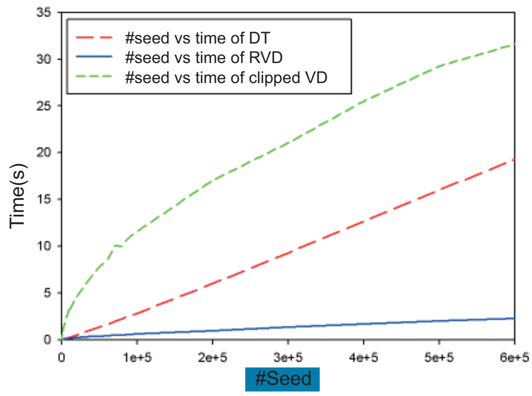


Figure 9: The timing curve of the clipped Voronoi diagram computation against the number of sites on Bone model.

inner site. Furthermore, we give another example of computing clipped Voronoi diagram in a cubic domain. The boundary mesh of the cube is shown in Figure 11(a). We sample the eight corners of the cube as sites, in this case, the bounding planes of Voronoi diagram are passing through the edges of the boundary mesh. The surface RVD and the volume clipped Voronoi diagram are shown in Figure 11(b) and (c), respectively.

2D meshing. We show some 2D mesh generation results based on our fast clipped Voronoi diagram computation. Figure 12 demonstrates that our algorithm works well for multiple connected domains. Figure 13 shows that we insert original vertices of input polygon for the better preservation of the geometric properties.

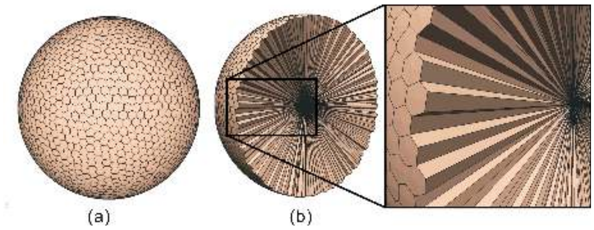


Figure 10: Clipped Voronoi diagram of a sphere. The sites are the vertices of the sphere. (a) The surface RVD, (b) the clipped Voronoi diagram.

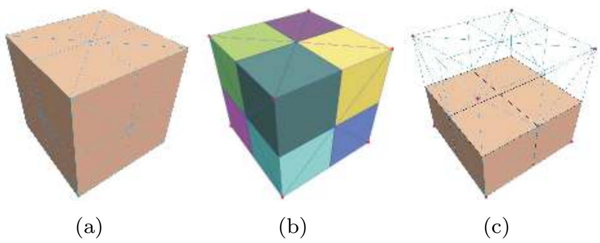


Figure 11: Clipped Voronoi diagram of a cube. Red points represent the sites. (a) The input domain, (b) the surface RVD, (c) the clipped Voronoi diagram.

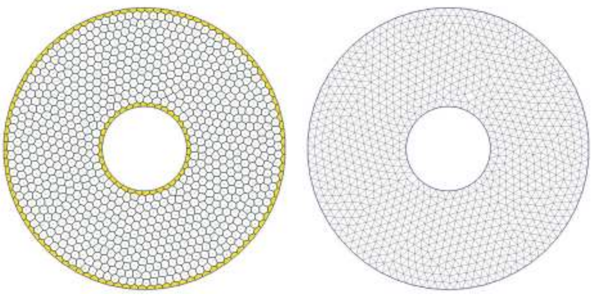


Figure 12: CVT-based 2D mesh generation of a ring.

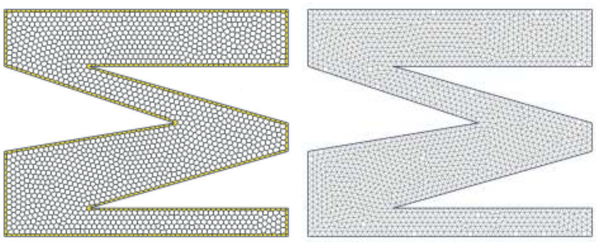


Figure 13: CVT-based 2D mesh generation. The boundary vertices of the input domain are used as constraints.

Tetrahedral meshing. The complete process of the proposed tetrahedral meshing framework is illustrated in Figure 6. Figure 14 (a)&(b) show two adaptive tetrahedral meshing examples, using lfs as the density function [30]. Figure 14 (c)&(d) give two examples with sharp features preserved. Our framework can generate high quality meshes efficiently and robustly. The running time for obtaining final results ranges from seconds to minutes, depending on the size of the input tetrahedral mesh and the desired number of sites.

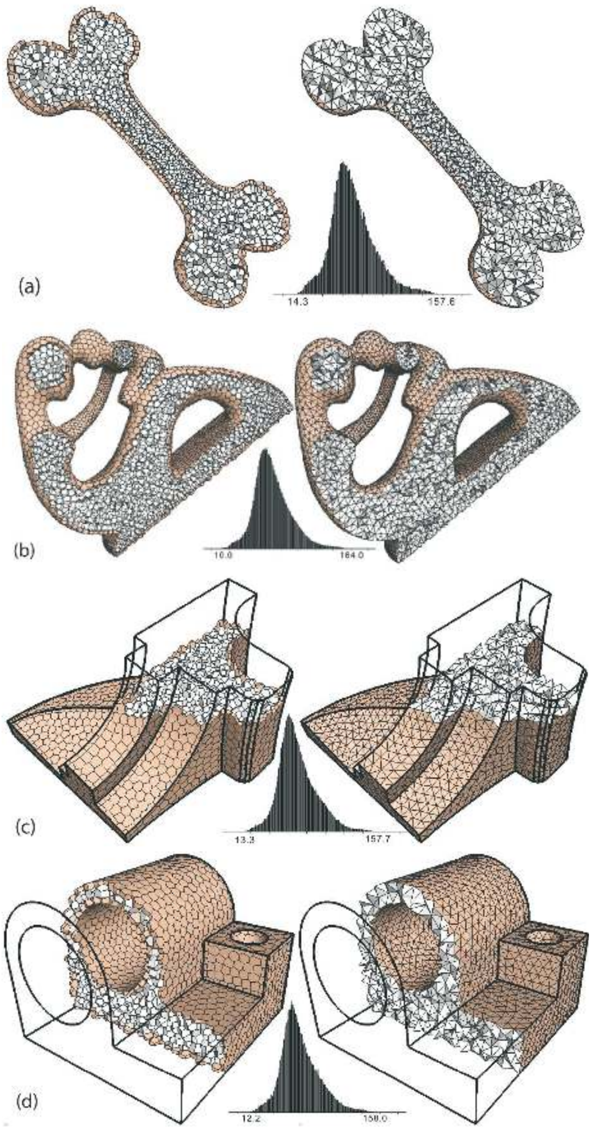


Figure 14: Tetrahedral mesh generation results. The histograms show the angle distribution of the results.

Comparison. We compare our meshing results with the Delaunay refinement approach provided by CGAL [4], as well as a recent work that used a discrete version of clipped Voronoi diagram for tetrahedral mesh generation [32]. Four shape quality measurements are used as in [32], i.e.,

- $Q_1 = \theta_{min}$, the minimal dihedral angle θ_{min} of each tetrahedron;
- $Q_2 = \theta_{max}$, the maximal dihedral angle θ_{max} of each tetrahedron;
- $Q_3 = \frac{3\,r_{in}}{r_{circ}}$, the radius-ratio of each tetrahedral, where r_{in} and r_{circ} are the inscribed/circumscribed radius, respectively;
- $Q_4 = \frac{12\,\sqrt[3]{9\,V^2}}{\sum l_{i,j}^2}$, meshing quality of [39], where V is the volume of the tetrahedron, and $l_{i,j}$ the length of the edge which connects vertices v_i and v_j.

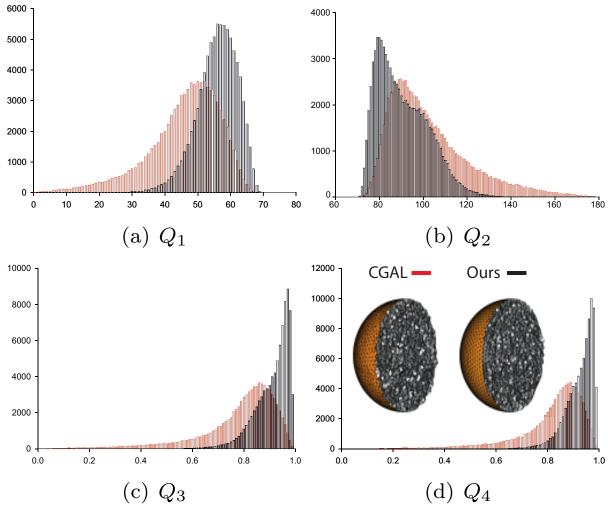
Q_3 and Q_4 are between 0 and 1, where 0 denotes a silver and 1 denotes a regular tetrahedron.

We choose the sphere generated from an isosurface as input domain. The Hausdorff distance (measured by Metro [40]) between the boundary of generated mesh and the input surface (normalized by dividing by the diagonal of bounding box) is 0.049%, which is 3 times smaller than 0.17% reported by [32]. The quality of the tetrahedral mesh is shown in Figure.15 and the comparison of each measurement is given in Table 2. Our approach produces better meshing quality, as well as smaller surface approximation error, attributed to the exact clipped Voronoi diagram computation.

method	$\overline{Q_1}$	$\overline{Q_4}$	$min(Q_1)$	$min(Q_4)$	HDist
[4]	48.11°	0.847	12.05°	0.339	0.054%
[32]	56.32°	0.911	16.31°	0.376	0.170%
ours	56.37°	0.932	24.23°	0.560	0.049%

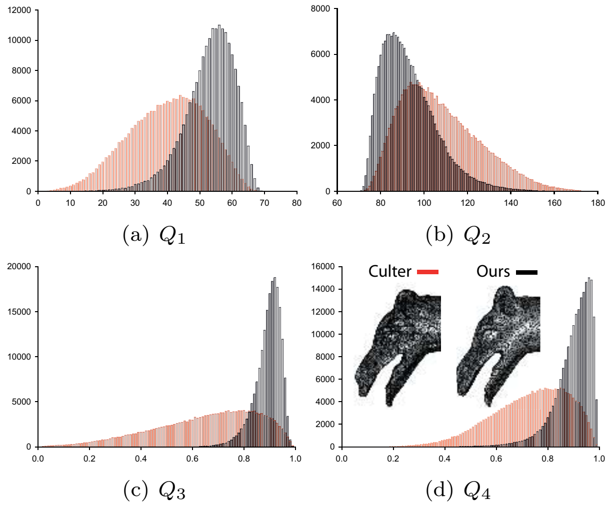
Table 2: Comparison of meshing qualities. `HDist` is the Hausdorff distance between the boundary of generated mesh and the input discretized isosurface.

We also compare our result with an octree-based approach [27]. As shown in Figure 16, the CVT based approach exhibits much better element quality than a standard approach. Our approach outperforms previous work in boundary approximation error (as shown in Figure 17), attributed to the exact clipped Voronoi diagram computation and simultaneous surface remeshing [3].



(a) Q_1 (b) Q_2 (c) Q_3 (d) Q_4

Figure 15: Comparison of the meshing qualities of the sphere with the Delaunay refinement approach implemented in CGAL [4].



(a) Q_1 (b) Q_2 (c) Q_3 (d) Q_4

Figure 16: Comparison with the octree based approach [27]. The resulting tetrahedral mesh has $200k$ tetrahedra.

7. Conclusion

We have presented efficient algorithms for computing the clipped Voronoi diagram for closed 2D and 3D objects, which has been a difficult problem without an efficient solution. As an application, we present a new CVT-based mesh generation algorithm which combines the clipped VD computation and fast CVT optimization.

In the future, we plan to look for more interdisciplinary applications of the clipped Voronoi diagram, such as biology and architecture. Applying our meshing technique to physical simulation applications, and extending the clipped Voronoi diagram to a higher dimension are also interesting directions.

Acknowledgements

We would like to thank anonymous reviewers for their detailed comments and suggestions which greatly improve the manuscript. We also thank one reviewer who pointed out the reference [1]. This work is partially supported by the Research Grant Council of Hong Kong (project no.: 718209 and 718010), the State Key Program of NSFC project (60933008), European Research Council (GOODSHAPE FP7-ERC-StG-205693), and ANR/NSFC (60625202, 60911130368) Program (SHAN Project).

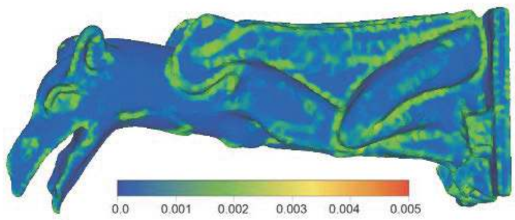


Figure 17: Approximation error of gargoyle model: 50K vertices, 256K tetrahedra, mean/max Hausdorff distance: 0.045%/0.37%. Our approach produces smaller approximation error compared with [30] (mean error: 0.053%) using the same number of vertices.

References

[1] Timothy M. Y. Chan, Jack Snoeyink, and Chee-Keng Yap. Output-sensitive construction of polytopes in four dimensions and clipped Voronoi diagrams in three. In *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 282–291, 1995.

[2] Yang Liu, Wenping Wang, Bruno Lévy, Feng Sun, Dong-Ming Yan, Lin Lu, and Chenglei Yang. On centroidal Voronoi tessellation: Energy smoothness and fast computation. *ACM Trans. on Graphics*, 28(4):Article No. 101, 2009.

[3] Dong-Ming Yan, Bruno Lévy, Yang Liu, Feng Sun, and Wenping Wang. Isotropic remeshing with fast and exact computation of restricted Voronoi diagram. *Computer Graphics Forum (Proceedings of SGP 2009)*, 28(5):1445–1454, 2009.

[4] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.

[5] C. Bradford Barber, David P. Dobkin, and Hannu Huh-

danpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Software*, 22:469–483, 1996.

[6] Franz Aurenhammer. Voronoi diagrams: a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, 1991.

[7] Steven Fortune. Voronoi diagrams and Delaunay triangulations. In *Computing in Euclidean Geometry*, pages 193–233, 1992.

[8] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* Wiley, 2nd edition, 2000.

[9] R. Kunze, F.-E. Wolter, and T. Rausch. Geodesic Voronoi diagrams on parametric surfaces. In *Proceedings of Computer Graphics International 1997*, pages 230–237, 1997.

[10] Gabriel Peyré and Laurent D. Cohen. Geodesic remeshing using front propagation. *Int. J. of Computer Vision*, 69:145–156, 2006.

[11] Herbert Edelsbrunner and Nimish R. Shah. Triangulating topological spaces. *Int. J. Comput. Geometry Appl.*, 7(4):365–378, 1997.

[12] Qiang Du, Max. D. Gunzburger, and Lili Ju. Constrained centroidal Voronoi tesselations for surfaces. *SIAM J. Sci. Comput.*, 24(5):1488–1506, 2003.

[13] Kelly A. Lyons, Henk Meijer, and David Rappaport. Algorithms for cluster busting in anchored graph drawing. *J. Graph Algorithms Appl.*, 2(1):1–24, 1998.

[14] Dong-Ming Yan, Kai Wang, Bruno Lévy, and Laurent Alonso. Computing 2D periodic centroidal Voronoi tessellation. In *Proc. of 8th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD)*, pages 177 – 184, 2011.

[15] Benoit Hudson, Gary L. Miller, Steve Y. Oudot, and Donald R. Sheehy. Topological inference via meshing. In *Proceedings of the 2010 annual symposium on Computational geometry (SOCG)*, pages 277–286, 2010.

[16] Kenneth E. Hoff III, John Keyser, Ming C. Lin, and Dinesh Manocha. Fast computation of generalized Voronoi diagrams using graphics hardware. In *Proceedings of ACM SIGGRAPH 1999*, pages 277–286, 1999.

[17] Avneesh Sud, Naga K. Govindaraju, Russell Gayle, Ilknur Kabul, and Dinesh Manocha. Fast proximity computation among deformable models using discrete Voronoi diagrams. *ACM Trans. on Graphics (Proc. SIGGRAPH)*, 25(3):1144–1153, 2006.

[18] S. Owen. A survey of unstructured mesh generation technology. In *Proceedings of 7th International Meshing Roundtable*, pages 26–28, 1998.

[19] Pascal Jean FREY and Paul-Louis GEORGE. *Mesh Generation: Application to Finite Elements.* Hermés Science, 2000.

[20] Nina Amenta, Marsahll Bern, and Manolis Kamvysselis. A new Voronoi-based surface reconstruction algorithm. In *Proceedings of ACM SIGGRAPH 1998*, pages 415–421, 1998.

[21] Pierre Alliez, David Cohen-Steiner, Yiying Tong, and Mathieu Desbrun. Voronoi-based variational reconstruction of unoriented point sets. In *Proceedings of Symposium on Geometry Processing (SGP 2007)*, pages 39–48, 2007.

[22] F. Leymarie and B. Kimia. The medial scaffold of 3D unorganized point clouds. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(2):313–330, 2007.

[23] S. Yamakawa and K. Shimada. Quad-layer: Layered quadrilateral meshing of narrow two-dimensional domain by bubble packing and chordal axis transformation. *Journal of Mechanical Design*, 124:564–573, 2002.

[24] W. R. Quadros and K. Shimada. Hex-layer: Layered allhex mesh generation on thin section solids via chordal surface transformation. In *Proc. of 11th International Meshing Roundtable*, pages 169–180, 2002.

[25] Peter Sampl. Semi-structured mesh generation based on medial axis. In *Proceedings of the 9th International Meshing Roundtable*, pages 21–32, 2000.

[26] J. R. Shewchuk. What is a good linear element? interpolation, conditioning, and quality measures. In *Proceedings of the 11th International Meshing Roundtable*, pages 115–126, 2002.

[27] Barbara Cutler, Julie Dorsey, , and Leonard McMillan. Simplification and improvement of tetrahedral models for simulation. In *Proceedings of the Eurographics Symposium on Geometry Processing*, pages 93–102, 2004.

[28] Yi-Jun Yang, Jun-Hai Yong, and Jia-Guang Sun. An algorithm for tetrahedral mesh generation based on conforming constrained Delaunay tetrahedralization. *Computers & Graphics*, 29(4):606–615, 2005.

[29] L. Chen and J. Xu. Optimal Delaunay triangulations. *Journal of Computational Mathematics*, 22(2):299–308, 2004.

[30] Pierre Alliez, David Cohen-Steiner, Mariette Yvinec, and Mathieu Desbrun. Variational tetrahedral meshing. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2005)*, 24(3):617–625, 2005.

[31] Jane Tournois, Camille Wormser, Pierre Alliez, and Mathieu Desbrun. Interleaving Delaunay refinement and optimization for practical isotropic tetrahedron mesh generation. *ACM Trans. on Graphics (Proc. SIGGRAPH)*, 28(3):Article No. 75, 2009.

[32] J. Dardenne, S. Valette, N. Siauve, N. Burais, and R. Prost. Variational tetraedral mesh generation from discrete volume data. *The Visual Computer (Proceedings of CGI 2009)*, 25(5):401–410, 2009.

[33] Qiang Du, Vance Faber, and Max Gunzburger. Centroidal Voronoi tessellations: applications and algorithms. *SIAM Review*, 41(4):637–676, 1999.

[34] Masao Iri, Kazuo Murota, and Takao Ohya. A fast Voronoi diagram algorithm with applications to geographical optimization problems. In *Proceedings of the 11th IFIP Conference on System Modelling and Optimization*, pages 273–288, 1984.

[35] Ivan E. Sutherland and Gary W. Hodgman. Reentrant polygon clipping. *Communications of the ACM*, 17(1):32–42, 1974.

[36] Jane Tournois, Rahul Srinivasan, and Pierre Alliez. Perturbing slivers in 3D Delaunay meshes. In *Proceedings of the 18th International Meshing Roundtable*, pages 157–173, 2009.

[37] Hang Si. TetGen: A quality tetrahedral mesh generator and three-dimensional Delaunay triangulator. http://tetgen.berlios.de.

[38] Andreas Meyer and Sylvain Pion. FPG: A code generator for fast and certified geometric predicates. *Real Numbers and Computers (RNC)*, pages 47–60, 2008.

[39] Anwei Liu and Barry Joe. On the shape of tetrahedra from bisection. *mathematics of computation*, 63(207):141–154, 1994.

[40] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2):167–174, 1998.