Efficient Computation of 3D Clipped Voronoi Diagram

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

The University of Hong Kong, Pokfulam Road, Hong Kong, China
Project Alice, INRIA, Nancy, France
{dmyan,wenping}@cs.hku.hk, {Bruno.Levy,Yang.Liu}@loria.fr

Abstract. The Voronoi diagram is a fundamental geometry structure widely used in various fields, especially in computer graphics and geometry computing. For a set of points in a compact 3D domain (i.e. a finite 3D volume), some Voronoi cells of their Voronoi diagram are infinite, 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 for computing the clipped Voronoi diagram for a set of sites with respect to a compact 3D volume, assuming that the volume is represented as a tetrahedral mesh. We also describe an application of the proposed method to implementing a fast method for optimal tetrahedral mesh generation based on the centroidal Voronoi tessellation.
keywords: Voronoi diagram, Delaunay triangulation, centroidal Voronoi tessellation, tetrahedral meshing.

1 Introduction

The Voronoi diagram (VD) is a fundamental and important geometry structure which has numerous applications in different areas, such as shape modeling [3], motion planning [18], scientific visualization [5], collision detection [19], geography [11], chemistry [16] and so on. For a finite set of sites (points in 3D), each site is associated with a Voronoi cell containing all the points closer to the site than to any other sites; all these cells constitute the Voronoi diagram of the set of sites.

Suppose that a set of sites in a compact domain in 3D are given. The Voronoi cells of those sites that on the boundary of the convex hull of all the sites are infinite. However, in many applications one often needs only the parts of the Voronoi cells inside the domain, as when computing the centroidal Voronoi tessellation. That is, the Voronoi diagram with respect to the given domain is defined as the intersection of the 3D Voronoi diagram and the domain, and is therefore called the *clipped Voronoi diagram*. The corresponding Voronoi cells are called the *clipped Voronoi cells*, see Figure 1 for 2D examples.

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, the operations would be more involved if the domain is non-convex and there has been no previous work on computing exact clipped Voronoi diagram for non-convex domains with arbitrary topology. A brute-force implementation would be inefficient because of the domain complexity.

Contributions: We present an efficient algorithm for computing clipped Voronoi diagrams of arbitrary closed 3D objects. The idea of our approach is to represent the input domain by a set of convex primitives. We use tetrahedron as the basic primitive in this paper – that is, the 3D domain is represented as a 3D tetrahedral mesh. Then the intersection of a 3D Voronoi cell and the input domain is reduced to computing the intersection of a 3D Voronoi cell and a set of tetrahedra, which can be done efficiently. The key to an efficient implementation is assigning each tetrahedron to its incident Voronoi cells, i.e., those Voronoi cells that intersect with the tetrahedron. Then we only need to compute the intersection between the tetrahedron and its incident cells. We identify the incident Voronoi cells for all the tetrahedra using neighborhood propagation.

This work extends our previous work [22] on computing the restricted Voronoi diagram (RVD) of a mesh surface in the following aspects. There we discuss how to compute a Voronoi diagram of sites on a triangle mesh surface by restricting the 3D Voronoi diagram of the sites to the surface, which involves the intersection of 3D Voronoi cells with individual triangles of the mesh surface. Also, in [22], we assume no connectivity information between the triangle elements and the intersection pairs of Voronoi cells and triangle elements are found with the assistance of a *kd*-tree. In this paper, we further improve the efficiency of surface RVD computation algorithm [22] by replacing the *kd*-tree query by a more efficient neighbor propagation approach, assuming the availability of the mesh connectivity information.

1.1 Previous work

A detailed survey of the Voronoi diagram is out of the scope of this paper, the reader is referred to [4, 10, 15] for the properties and applications of the Voronoi diagram. Existing techniques can 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 [1] and Qhull [6].

To speed up the Voronoi diagram computation in specific applications, many researchers focus on computing approximated Voronoi diagram on discrete spaces with the help of the GPU (*Graphical Processing Unit*). Hoff III *et al.* [12] propose a technique for computing discrete generalized Voronoi diagram using graphics hardware. The Voronoi diagram computation is cast into a clustering problem in the discrete voxel/pixel space. Sud *et al.* [19] present an n-body proximity query algorithm based on computing the discrete 2^{nd} order Voronoi diagram on the GPU. GPU based algorithms are fast but produce only a discrete approximation of the true Voronoi diagram.

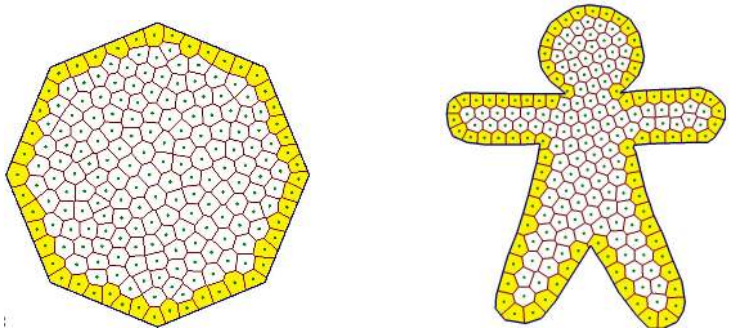
Yan *et al.* [22] present a direct algorithm for computing the restricted Voronoi diagram (RVD) [9] on mesh surfaces. In that method no connectivity information between the triangle facets is assumed, and a *kd*-tree is used to find the nearest sites of each triangle in order to identify its incident Voronoi cells. The incident 3D Voronoi cells of each triangle face are identified starting from the nearest site and the intersection between the triangle and its incident Voronoi cells is computed by Sutherland's clipping algorithm [20]. Let m be the number of triangles and n the number of sites. Then the time complexity of the method in [22] is $O(m \log n)$ assuming that the number of incident cells of each triangle is bounded, since a *kd*-tree is used for the nearest site query.

1.2 Outline

The remainder of this paper is organized as follows: We give the problem formulation in Section 2 and the overview of our algorithm in Section 3. We present our algorithm for computing clipped Voronoi diagram of 3D objects in Section 4. As an application of our algorithm, we present in Section 5 a CVT-based tetrahedral meshing method built on the top of our new method for computing the 3D clipped Voronoi diagram. Experimental results are given in Section 6 and we draw conclusions in Section 7.

2 Problem Formulation

We consider computing the exact clipped Voronoi diagram of closed 3D objects. Figure 1 illustrates the problem with two 2D examples of the clipped Voronoi diagram with a convex domain and a non-convex domain, respectively.



(a) (b)

Fig. 1. Clipped Voronoi diagram on 2D convex (a) and non-convex (b) domains. Shaded cells are boundary Voronoi cells. The number of seeds is 200 for each example.

Given a set of sites $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^n$ in 3D, the Voronoi diagram of $\mathbf{X}$ is defined by a collection of n Voronoi cells $\Omega = \{\Omega_i\}_{i=1}^n$, where

$$\Omega_i = \{\mathbf{x} \in \mathbb{R}^3, \|\mathbf{x} - \mathbf{x}_i\| \leq \|\mathbf{x} - \mathbf{x}_j\|, \forall j \neq i\}.$$

Each Voronoi cell Ω_i is the intersection of a set of 3D half-spaces, delimited by the bisecting planes of the Delaunay edges incident to the site $\mathbf{x}_i$.

Let $\mathcal{M}$ denote the input domain, which is assumed to be a connected compact set in 3D. The clipped Voronoi diagram for the sites $\mathbf{X}$ with respect to $\mathcal{M}$ is defined as the intersection of the 3D Voronoi diagram Ω and $\mathcal{M}$, denoted as $\Omega|_{\mathcal{M}} = \{\Omega_i|_{\mathcal{M}}\}_{i=1}^n$, where

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

which is the intersection of the Voronoi cell Ω_i and $\mathcal{M}$. We call $\Omega_i|_{\mathcal{M}}$ the *clipped Voronoi cell* with respect to $\mathcal{M}$.

3 Algorithm Overview

In this section we describe an efficient algorithm for computing the clipped Voronoi diagram of 3D objects. We suppose that the domain $\mathcal{M}$ is represented as a set of tetrahedra; other types of convex primitives can be used for this decomposition as well.

Suppose that the input domain $\mathcal{M}$ is given by a tetrahedral mesh, that is $\mathcal{M} = \{\mathcal{V}, \mathcal{T}\}$, where $\mathcal{V} = \{\mathbf{v}_k\}_{k=1}^{n_v}$ is the set of mesh vertices and $\mathcal{T} = \{\mathbf{t}_i\}_{i=1}^m$ is 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 2-manifold. Each boundary triangle facet $\mathbf{f}_j$ stores the indices of three neighboring facets and the index of its containing tet.

The clipped Voronoi cells $\{\Omega_i|_{\mathcal{M}}\}$ can be classified into two types: *inner Voronoi cells* that are contained in the interior of $\mathcal{M}$ and *boundary Voronoi cells* that intersect the boundary $\mathcal{S}$ of the domain $\mathcal{M}$. Since the inner Voronoi cells of $\Omega|_{\mathcal{M}}$ are entirely inside the domain, there is no need to clip them against the boundary surface $\mathcal{S}$. So we just need to concentrate on computing the boundary Voronoi cells, see Figure 1 for 2D examples of clipped Voronoi diagrams.

Therefore the problem now is how to identify all the sites whose Voronoi cells intersect the boundary $\mathcal{S}$. To solve this problem, we first compute the surface restricted Voronoi diagram (RVD) for all the sites $\mathbf{X}$. The sites whose Voronoi cells intersect the domain boundary surface are called the *boundary sites* and their cells are the boundary Voronoi cells. So we will just compute the intersection of the boundary Voronoi cells with the domain $\mathcal{M}$. The reader is referred to [21, 22] for details of surface RVD computation. In this paper, we further improve the RVD computation algorithm by replacing the *kd*-tree search in [22] by a more efficient neighbor propagation approach.

4 Clipped Voronoi Diagram Computation

There are three main steps of our algorithm for computing the clipped Voronoi diagram:

1. **Voronoi diagram construction:** This step computes the Delaunay triangulation for the input sites, from which we extract the 3D Voronoi diagram;
2. **Surface RVD computation:** We compute the surface RVD to identify all the boundary Voronoi cells;
3. **Clipped Voronoi cells construction:** The 3D clipped Voronoi cells for all the boundary Voronoi cells are computed.

In the following, we will explain each step in details.

4.1 Voronoi diagram construction

We first build a 3D Delaunay triangulation from input sites $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^n$, using CGAL. The corresponding 3D Voronoi diagram $\Omega = \{\Omega_i\}_{i=1}^n$ is constructed as the dual of the Delaunay triangulation, as defined in Section 2.

4.2 Surface RVD computation

For the given set of sites $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^n$ and the boundary surface $\mathcal{S}$, the restricted Voronoi diagram (RVD) is defined as the intersection of the 3D Voronoi diagram Ω and the surface $\mathcal{S}$, denoted as $\mathbf{R} = \{\mathbf{R}_i\}_{i=1}^n$, where $\mathbf{R}_i = \Omega_i \bigcap \mathcal{S}$ [9]. Each $\mathbf{R}_i$ is called a *restricted Voronoi cell* (RVC). We compute the surface RVD using neighbor propagation, which is faster than searching using the *kd*-tree structure, as shown by our tests.

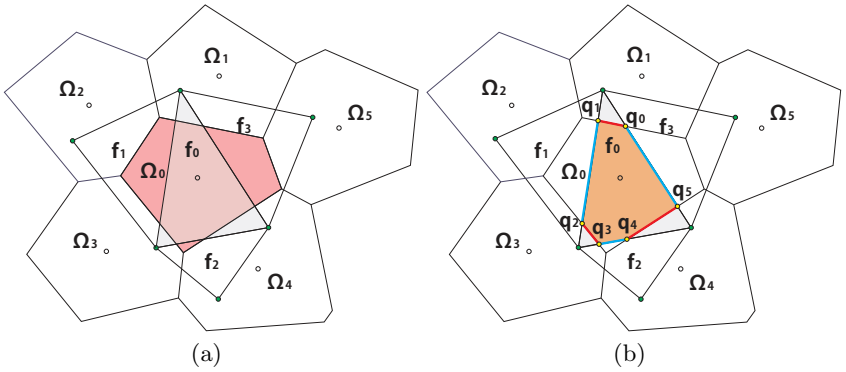


Fig. 2. Illustration of propagation process.

Now we are going to explain the propagation steps. Refer to Figure 2. We start from a seed triangle and one of its incident cells, which can be found by a

linear search function. Here we assume that a triangle $\mathbf{f}_0$ of boundary $\mathcal{S}$ is the seed triangle and the Voronoi cell Ω_0 is the corresponding cell of the nearest site of $\mathbf{f}_0$, as shown in Figure 2(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, new valid pairs are identified and pushed back into $\mathcal{Q}$. The process terminates when $\mathcal{Q}$ is empty.

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 Q, as shown in Figure 2. 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 2(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 opposite 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 indices set of the cell. The pair is pushed into the queue only if the facet is not contained in the incident facets set of the cell; otherwise the pair is discarded. Each time after intersection computation, the resulting polygon is made 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 boundary sites, denoted as $\mathbf{X}_b = \{\mathbf{x}_i | \mathbf{R}_i \neq \emptyset\}$. The pseudo code of the algorithm is given in Algorithm 4.1.

Algorithm 4.1: Surface RVD computation algorithm.

```
input  : sites X, boundary mesh S
output: surface RVD R of X on S
begin
    Ω ← VoronoiDiagram(X) ;
    {f_0, Ω_0} ← FindInitialPair() ;
    queue Q ← {f_0, Ω_0} ;
    while Q ≠ ∅ do
        {f_t, Ω_t} ← Q.pop() ;
        polygon poly ← Intersect(f_t, Ω_t) ;
        R_t ← R_t ⋃ poly ;
        Q.push(NewIncidentPairs(poly)) ;
    end
end
```

4.3 Clipped Voronoi cells construction

Once the boundary sites $\mathbf{X}_b$ are found, we will compute the clipped Voronoi cells for these sites. The boundary Voronoi cells computation is similar to the surface RVD computation presented in Section 4.2, 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 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 Q is popped out repeatedly. We compute the intersection of Ω_i and $\mathbf{t}_j$ again by Sutherland-Hodgman clipping algorithm [20] and identity 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 neighbor tets who share the facets clipped by the current bisecting plane. Those cells 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 made associated with the clipped Voronoi cell $\Omega_i|_{\mathcal{M}}$ of site $\mathbf{x}_i$. This process terminates when Q is empty.

5 Tetrahedral Mesh Generation

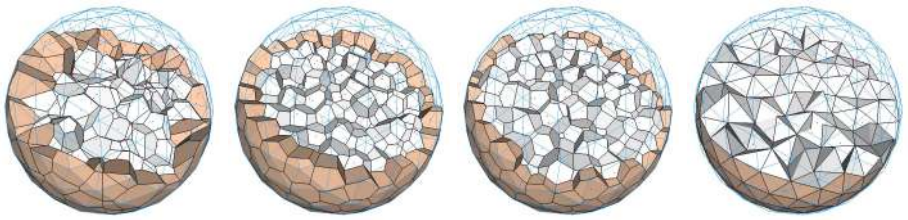
As an application, we implemented an efficient method for tetrahedral meshing based on centroidal Voronoi tessellation (CVT) [7, 8], which utilizes heavily the computation of the 3D clipped Voronoi diagram. The L-BFGS method in [14] for computing CVT is used in our implementation. We will briefly describe this framework and present our experimental results in Section 6.

The centroidal Voronoi tessellation is a special kind of Voronoi tessellation such that each seed $\mathbf{x}_i$ coincides with the mass center of its Voronoi region. In the context of CVT based tetrahedral meshing, the CVT energy function is defined on the input mesh $\mathcal{M}$, *i.e.*,

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

where $\rho(\mathbf{x}) > 0$ is a user-defined density function. When ρ is a constant, we get a uniform CVT. The reader is referred to [7] for preliminaries of CVT and [14] for details of convergency analysis of CVT energy functions. We omit them here since they are not the main contribution of this paper.

There are three steps of the CVT based meshing framework: initialization, iterative optimization, and mesh extraction. Our mesh generation framework is illustrated by the example in Figure 3.



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

Fig. 3. Illustration of the proposed tetrahedral meshing algorithm. The wire frame is the boundary of input mesh. (a) Clipped Voronoi diagarm of initial sites (the boundary Voronoi cells are shaded); (b) result of unconstrained CVT with $\rho = 1$; (c) result of constrained optimization. Notice that boundary seeds are constrained on the surface $\mathcal{S}$; (d) final uniform tetrahedral meshing result.

5.1 Initialization

In this step, we build a uniform grid to store the sizing field for adaptive meshing. Following the approach in [2], 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 3(a)). The reader is referred to [2] for details.

5.2 Optimization

There are two phases of the global optimization part: unconstrained CVT optimization and constrained CVT optimization. In the first phase, we optimize the positions of the sites inside the input volume without any constraint, which yields a well-spaced distribution of the sites within the domain, with no sites lying on the domain boundary surface (Figure 3(b)). In the second phase, we identify all the boundary sites, i.e. those sites whose Voronoi cells intersect the domain boundary surface; we project these sites onto the boundary surface and they will be constrained to the boundary surfaces during the subsequent optimization. Then all the boundary sites and the inner sites are optimized simultaneously, again with respect to the CVT energy function (Figure 3(c)). The details of these steps are explained in the following.

CVT optimization. In the first phase, we use the L-BFGS method [14] to compute the CVT by minimizing the CVT energy function (Eqn. 1). To apply the L-BFGS method to minimize the CVT energy function we need the gradient of the CVT energy function. The partial derivative of the energy function w.r.t. each site is given by the following equation [13]:

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

here $m_i = \int_{\Omega_i|_{\mathcal{M}}} \rho(\mathbf{x})\|\mathbf{x} - \mathbf{x}_i\|^2 \, \mathrm{d}\sigma$, and $\mathbf{x}_i^*$ is the centroid given by

$$x_i^* = \frac{\int_{\Omega_i|_{\mathcal{M}}} \rho(\mathbf{x})\mathbf{x} \, \mathrm{d}\sigma}{\int_{\Omega_i|_{\mathcal{M}}} \rho(\mathbf{x}) \, \mathrm{d}\sigma}. \tag{3}$$

To integrate this function, each clipped Voronoi cell $\Omega_i|_{\mathcal{M}}$ is split into a set of sub-tets $\{\tau_k\}$ by simply connecting the centroid of each clipped polyhedron of $\Omega_i|_{\mathcal{M}}$ with its triangulated facets (see Section 4.3). As discussed in [2], the exact integration of the density function may not improve the quality very much, since the density function is also discretely defined. We use a one-point approximation of the density function at the centroid of each sub-tet of the Voronoi cells, i.e.,

$$x_i^* = \frac{\sum_{\tau_k \in \Omega_i|_{\mathcal{M}}} \rho(\mathbf{c}_k)\mathbf{c}_k \cdot |\tau_k|}{\sum_{\tau_k \in \Omega_i|_{\mathcal{M}}} \rho(\mathbf{c}_k) \cdot |\tau_k|}, \tag{4}$$

where $\mathbf{c}_k$ and $|\tau_k|$ are the centroid and the volume of sub-tet τ_k, respectively.

Once the L-BFGS method is used to compute the updated sites, we need to compute the exact clipped Voronoi cells of these sites in the domain $\mathcal{M}$. Then we start the next iteration until convergence or some termination condition is met. After convergence, all the sites of the boundary Voronoi cells will be projected to the domain boundary surface.

Constrained CVT. During the second phase of optimization, all the boundary sites will be constrained on the boundary. The partial derivative of the energy function w.r.t each boundary site is computed as:

$$\left.\frac{\partial F}{\partial \mathbf{x}_i}\right|_{\mathcal{S}} = \frac{\partial F}{\partial \mathbf{x}_i} - \left[\frac{\partial F}{\partial \mathbf{x}_i} \cdot \mathbf{N}(\mathbf{x}_i)\right] \mathbf{N}(\mathbf{x}_i), \tag{5}$$

where $\mathbf{N}(\mathbf{x}_i)$ is the unit normal vector of the boundary surface at the boundary site $\mathbf{x}_i$ [14]. 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.

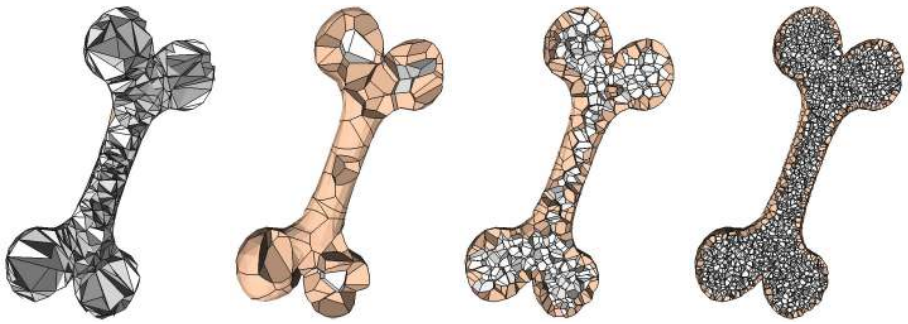
Sharp features are preserved in a similar way as how the boundary sites are treated. For example, we project sites on sharp edges on the boundary and allow them to vary only along these edges during the second stage of optimization. For details, please refer to [22] where these steps are described in the context of surface remeshing.

5.3 Final mesh extraction

After convergence, we shall extract a well-shaped surface triangle remesh for the domain boundary as well as a tetrahedral mesh for the domain interior as the dual of the final CVT. We first compute a boundary remesh $\mathcal{S}'$ from all the boundary seeds $\mathbf{X}_b$ [22]. The final tetrahedral mesh is then extracted by computing a conformal Delaunay triangulation from $\mathcal{S}'$ and all the inner seeds (Figure 3(d)).

6 Experimental results

Our algorithm is implemented in C++ on both Windows and Linux platform. We use CGAL [1] for Delaunay triangulation and TetGen [17] for background mesh generation when the input is given only as a closed boundary mesh. All the experimental results are tested on a laptop with 2.4Ghz processor and 2Gb memory.



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

Fig. 4. (a) Input Bone model (3,368 tets and $1k$ boundary triangles); (b) clipped Voronoi diagram of 100 sites; (c) $1k$ sites; (d) $10k$ sites.

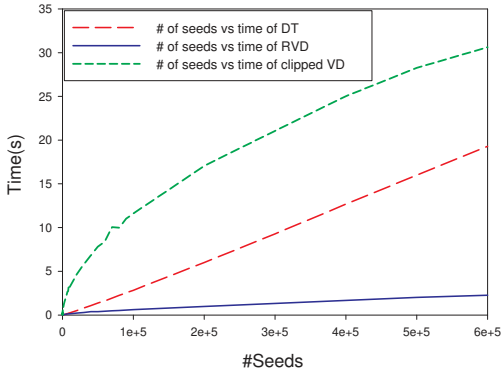


Fig. 5. Timing curve of clipped Voronoi diagram computation against the number of sites on Bone model.

Efficiency. We first demonstrate the performance of the proposed clipped VD computation algorithm. We progressively sample points inside an input tetrahe-

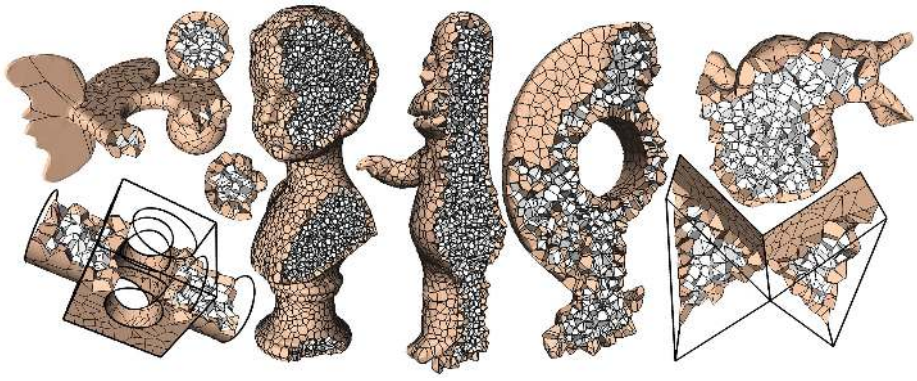


Fig. 6. Results of clipped Voronoi diagram computation.

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

dral mesh, which contains $1k$ boundary triangles and $3,368$ tets. The number of sites increases from 10 to $6 \times e^5$. The results are shown in Figure 4 and the timing curves are shown in Figure 5. From the timing curve in Figure 5, we can see that the time spent on surface RVD computation is much less than Delaunay triangulation, since only a small portion of all the sites are boundary sites. More results of clipped Voronoi diagram computation of various 3D objects are given in Figure 6 and the timing statistics are given in Table 1.

The computational time of the clipped VD computation algorithm is proportional to the total number of incident cell-tet pairs (Section 4.3). Therefore, an input mesh with a small number of tetrahedral elements would help improve the efficiency. In our experiments, all the input tetrahedral meshes are generated by the robust meshing software TetGen [17] with conforming boundary.

Tetrahedral meshing. The complete process of the proposed tetrahedral meshing framework is illustrated in Figure 3. Figure 7 shows two adaptive tetrahedral meshing examples, with lfs as the density function. Figure 8 gives 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

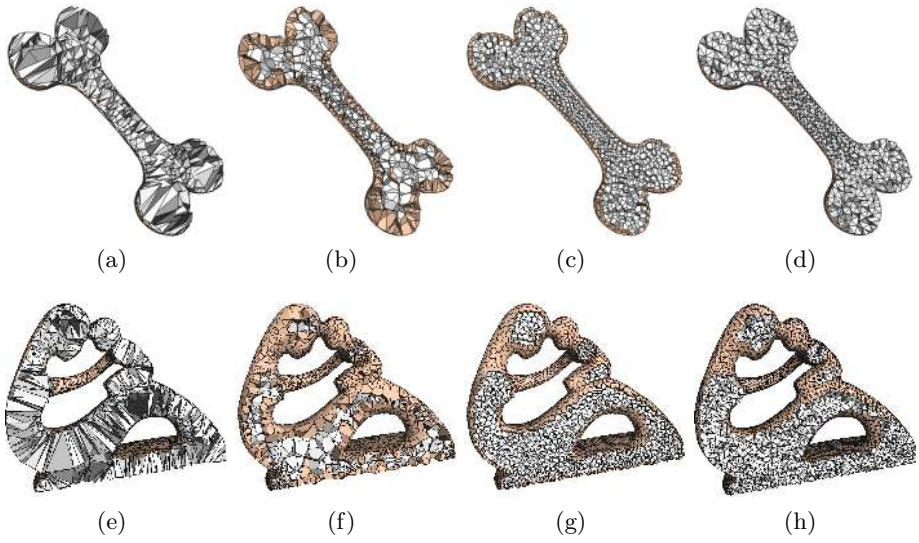


Fig. 7. Adaptive meshing result of Bone (top row) and Fertility (bottom row). (a)&(e) Cut-view of input meshes; (b)&(f) clipped Voronoi diagrams of initial samples; (c)&(g) optimization results; (d)&(h) tetrahedral meshing results.

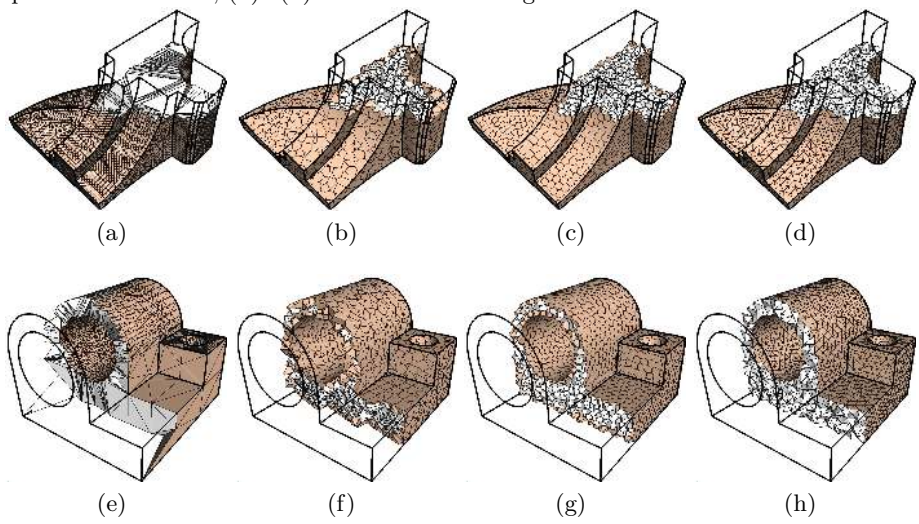


Fig. 8. Uniform meshing result of Fandisk (top row) and Joint (bottom row). (a)&(e) Cut-view of input meshes; (b)&(f) clipped Voronoi diagrams of initial samples; (c)&(g) optimization results; (d)&(h) tetrahedral meshing results with feature preserved.

seconds to minutes, depending on the size of the input tetrahedral mesh and the desired number of sites.

7 Conclusion

We have presented an efficient algorithm for computing clipped Voronoi diagram for closed 3D objects, which has been a difficult problem without an efficient implementation. As an application, we present a new CVT based tetrahedral meshing algorithm which combines our fast clipped VD computation with fast CVT optimization [14]. In the future, we plan to investigate GPU-based methods to further improve the efficiency.

Acknowledgements

The Bunny model is the courtesy of the Stanford 3D Scanning Repository. The other 3D models used in this paper are from AIM@Shape project. We would like to thank Mr. Feng Sun for many helpful discussions during this work.

Dong-Ming Yan and Wenping Wang are partially supported by the General Research Funds (718209, 717808) of Research Grant Council of Hong Kong, NSFC-Microsoft Research Asia co-funded project (60933008), and National 863 High-Tech Program of China (2009AA01Z304). Bruno Lévy and Yang Liu are supported by the European Research Council (GOODSHAPE FP7-ERC-StG-205693).

References

1. CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.
2. 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.
3. Pierre Alliez, Giuliana Ucelli, Craig Gotsman, and Marco Attene. Recent advances in remeshing of surfaces. *Shape Analysis and Structuring*, pages 53–82, 2008.
4. Franz Aurenhammer. Voronoi diagrams: a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, 1991.
5. Michael Balzer and Oliver Deussen. Voronoi treemaps. In *Proceedings of the 2005 ACM Symposium on Software Visualization*, pages 165–172, 2005.
6. C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Software*, 22:469–483, 1996.
7. Qiang Du, Vance Faber, and Max Gunzburger. Centroidal Voronoi tessellations: applications and algorithms. *SIAM Review*, 41(4):637–676, 1999.
8. Qiang Du, Max Gunzburger, and Lili Ju. Advances in studies and applications of centroidal Voronoi tessellations. *Numer. Math. Theor. Meth. Appl.*, 2010. to appear.
9. Herbert Edelsbrunner and Nimish R. Shah. Triangulating topological spaces. *Int. J. Comput. Geometry Appl.*, 7(4):365–378, 1997.

10. Steven Fortune. Voronoi diagrams and Delaunay triangulations. In *Computing in Euclidean Geometry*, pages 193–233, 1992.
11. Christopher M. Gold. What is GIS and what is not? *Transactions in GIS*, 10(4):505–519, 2006.
12. 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.
13. 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.
14. 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 Transactions on Graphics*, 28(4):Article No. 101, 2009.
15. Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley, 2nd edition, 2000.
16. Anne Poupon. Voronoi and Voronoi-related tessellations in studies of protein structure and interaction. *Current Opinion in Structural Biology*, 14(2):233–241, 2004.
17. Hang Si. TetGen: A quality tetrahedral mesh generator and three-dimensional Delaunay triangulator.
18. Avneesh Sud, Erik Andersen, Sean Curtis, Ming C. Lin, and Dinesh Manocha. Real-time path planning in dynamic virtual environments using multiagent navigation graphs. *IEEE Transactions on Visualization and Computer Graphics*, 14(3):526–538, 2008.
19. Avneesh Sud, Naga K. Govindaraju, Russell Gayle, Ilknur Kabul, and Dinesh Manocha. Fast proximity computation among deformable models using discrete Voronoi diagrams. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2006)*, 25(3):1144–1153, 2006.
20. Ivan E. Sutherland and Gary W. Hodgman. Reentrant polygon clipping. *Communications of the ACM*, 17(1):32–42, 1974.
21. Dong-Ming Yan. *Variational Shape Segmentation and Mesh Generation*. Phd dissertation, The University of Hong Kong, 2010.
22. 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.