Parallel Computation of 3D Clipped Voronoi Diagrams

Xiaohan Liu, Lei Ma, Jianwei Guo, and Dong-Ming Yan

Abstract—Computing the Voronoi diagram of a given set of points in a restricted domain (e.g., inside a 2D polygon, on a 3D surface, or within a volume) has many applications. Although existing algorithms can compute 2D and surface Voronoi diagrams in parallel on graphics hardware, computing clipped Voronoi diagrams within volumes remains a challenge. This article proposes an efficient GPU algorithm to tackle this problem. A preprocessing step discretizes the input volume into a tetrahedral mesh. Then, unlike existing approaches which use the bisecting planes of the Voronoi cells to clip the tetrahedra, we use the four planes of each tetrahedron to clip the Voronoi cells. This strategy drastically simplifies the computation, and as a result, it outperforms state-of-the-art CPU methods up to an order of magnitude.

Index Terms—Parallel algorithm, Voronoi diagram, clipping

1 Introduction

Voronoi diagrams of a given set of points (also called *sites* or *generators*) commonly appear in natural environments and man-made structures, and are widely applied in many fields such as engineering, architecture, urban planning, and geography. In past decades, Voronoi diagrams have been extensively used in computer graphics. Since ordinary Voronoi diagrams have unbounded cells corresponding to the generators on the convex hull of its dual Delaunay triangulation [1], one always needs to restrict the Voronoi diagram to a finite domain in practice [2], i.e., to clip the ordinary Voronoi diagram by the domain boundary [3], [4]. Fig. 1 shows several examples of clipped Voronoi diagrams in 2D polygons and 3D surfaces and volumes.

In detail: given a set of Voronoi sites in a compact domain in $\mathbb{R}^3$, the Voronoi cell for a site contains the space in $\mathbb{R}^3$ closer to that site than to any other site. Voronoi cells of sites on the convex hull are unbounded, and some interior cells may be partly outside the domain. In practice, such cells always need further treatment because only the parts of Voronoi cells inside the given domain are useful. Therefore, we refer to the restriction of the Voronoi diagram to the given domain as the *clipped* Voronoi diagram; it is the intersection of the Voronoi diagram and the input domain.

There are efficient algorithms for computing unrestricted Voronoi diagrams [5], and moreover, GPU implementations also exist [6], [7]. However, computing clipped Voronoi diagrams in 3D is time consuming, because it requires frequently computing the intersection between Voronoi cells and the domain. Therefore, many works have targeted improving the computational efficiency of clipped Voronoi diagrams in 3D and related applications. Rong *et al.* [8] compute surface Voronoi diagrams via geometry image parameterization, and use the jump flooding algorithm [6] for the 2D Voronoi diagram computation. Han *et al.* [9] compute restricted Voronoi diagrams on surface meshes by performing polygon clipping on the GPU. Leung *et al.* [10] compute approximate restricted Voronoi diagrams by discretizing the input surface into voxels and computing an exact euclidean distance transform on the GPU. Ray *et al.* [7] compute large scale Voronoi diagrams on the GPU by storing the dual form of a Voronoi cell using a simple triangle mesh, which can be easily updated and computed in each cell in parallel. However, this method can only compute 3D Voronoi diagrams without handling the domain boundary, which is the bottleneck in computing clipped Voronoi diagrams.

This study investigates new methods for improving the efficiency of computing 3D clipped Voronoi diagrams. Our approach uses a preprocessing step which discretizes the input volume into a tetrahedral mesh, and is based on two key observations. First, in various practical applications, the tetrahedra (*tets* for short) and Voronoi cells have a high correlation in local space, especially when $|\mathcal{T}|$ (i.e., the number of tets) is close to $|\mathbf{X}|$ (i.e., the number of Voronoi sites). This observation can be used to advantage in a k-nearest neighbors (k-NN) search strategy [11]. Second, the intersection of a tet with a Voronoi cell can be calculated independently of other tets and cells. Unlike the approaches in [2] and [12] that use bisecting planes of Voronoi cells to clip tets, in our approach, each precomputed cell is clipped by the four planes of each of

- *Xiaohan Liu and Jianwei Guo are with the National Laboratory of Pattern Recognition (NLPR), Institute of Automation, Chinese Academy of Sciences (CASIA), Beijing 100190, China, and also with the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing 100049, China. E-mail: liuxiaohan2017@ia.ac.cn, jianwei.guo@nlpr.ia.ac.cn.*
- *Lei Ma is with the National Engineering Laboratory for Video Technology, Peking University, Beijing 100000, China. E-mail: malei@outlook.com.*
- *Dong-Ming is with the National Laboratory of Pattern Recognition (NLPR), Institute of Automation, Chinese Academy of Sciences (CASIA), Beijing 100190, China, with the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing 100049, China and also with the State Key Laboratory of Hydroscience and Engineering, Tsinghua University, Beijing 100084, China. E-mail: yandongming@gmail.com.*

Manuscript received 4 Mar. 2020; revised 22 July 2020; accepted 24 July 2020. Date of publication 28 July 2020; date of current version 31 Dec. 2021. (Corresponding author: Dong-Ming Yan.) Recommended for acceptance by K. Hormann. Digital Object Identifier no. 10.1109/TVCG.2020.3012288

Fig. 1. Left to right: Clipped Voronoi diagrams in 2D polygons, on 3D surfaces and in volumes in natural environments (top) and man-made structures and designs (bottom).

its neighboring tets. This approach reduces the computational cost significantly. Based on these observations, we present a practical algorithm making best use of the multicore, parallel computing capabilities of modern GPUs. Our parallel GPU implementation is approximately an order of magnitude faster than state-of-the-art implementations. The main contributions of this study are thus:

- an efficient tet-Voronoi cell nearest pairing algorithm based on an independent and efficient tet-Voronoi cell clipping technique, significantly reducing computational cost,
- a linear multi-barycenter update mechanism based on the *atomicAdd* operation, allowing our completely parallel algorithm to make full use of modern GPUs, and
- a heuristic strategy to chose k in k-nearest search, demonstrated to be robust in examples of constructing clipped Voronoi cells.

2 Related Work

Various efficient algorithms for computing Voronoi diagrams and Delaunay triangulations have been developed over past decades. Several robust industrial-quality libraries are widely used, such as CGAL [13], Qhull [14], and Geogram [15]. We refer the readers to [1], [5], [16] for a survey of Voronoi diagrams and their applications. This section focuses on work related to efficient computation of Voronoi diagrams in restricted domains.

2.1 Restricted Voronoi Diagrams on Surfaces

In various applications, if the number of sites is sufficient, the *restricted Voronoi diagram* (RVD) [17] under euclidean distance is a good approximation to the *geodesic Voronoi diagram* (GVD) [18]; the RVD is defined as the intersection of the 3D Voronoi diagram and the mesh surface. Yan *et al.* [2] propose an efficient method for exact computation of the RVD, which requires building the 3D Voronoi diagram and a kd-tree for efficient neighbor search. Later, Lévy and Bonneel [19] improved existing RVD algorithms by removing the requirement for 3D Voronoi diagrams, and only a kd-tree is used to dynamically construct Voronoi cells by nearest neighbor propagation. Yan *et al.* [20] further improved the robustness of RVD algorithms by computing a localized version of restricted Voronoi diagrams (LRVDs), which does not require the kd-tree construction. To make RVD algorithms faster for real-time applications, several GPU implementations have been proposed [8], [9], [21].

2.2 Clipped Voronoi Diagram in Volumes

If the input domain is a closed volume, the Voronoi diagram confined to the volume is called a *clipped Voronoi diagram* [22]. The clipped Voronoi diagram is the intersection of the 3D Voronoi diagram and the input domain. Yan *et al.* [4] first proposed an efficient algorithm to compute the clipped Voronoi diagram of a given set of sites with respect to a compact 2D region or volume. It first tessellates the volume into a tetrahedral mesh. Then, clipped Voronoi cells are computed for those sites whose Voronoi cells intersect the domain boundary. The cells inside the domain do not need to be recomputed, as they are the same as the original Voronoi cells. Because of the robustness of this algorithm, it has been implemented in Geogram [15] as a standard open source tool. Simultaneously, Lévy and Liu [3] proposed another algorithm based on explicitly constructing walls of boundary cells. However, such approaches lack robustness in degenerate cases. Moreover, the above methods are still CPU-based and fail to fully exploit the parallel nature of cell-tet intersection. Here, we propose the first GPU parallel algorithm to tackle this problem.

To fully exploit the computational power of modern graphics hardware, various fundamental geometric algorithms have been devised for GPUs, such as building kd-trees [23], and computing convex hulls [24], Delaunay triangulations [25], and Voronoi diagrams [6], [26]. Rong *et al.* [8] propose a GPU algorithm to compute surface Voronoi diagrams in parametric space using geometry images. In particular, they extend the centroidal Voronoi tessellation (CVT) energy function from euclidean space to spherical and hyperbolic spaces in [27], and propose a parallel version in [28]. Ray *et al.* [7] present an algorithm for computing large scale Voronoi diagrams on GPUs. The key idea is to store the dual form of a Voronoi cell using a simple triangle mesh, which can be easily updated and computed in each cell in parallel. We utilize this algorithm for Voronoi cell construction as the first step of our method. We also extend our previous abstract [29] and propose algorithms for cell clipping and a heuristic strategy for choosing k in the k-NN search, a key parameter of the proposed algorithm.

3 Preliminaries

We first briefly review basic definitions of the Voronoi diagram, the clipped Voronoi diagram, and the CVT.

3.1 Voronoi Diagrams

Given a set of sites $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^n$ in 3D, the Voronoi diagram $\Omega(\mathbf{X})$ is a collection of Voronoi cells $\Omega(\mathbf{X}_i)$ (Ω_i for short), i.e., $\Omega(\mathbf{X}) = \{\Omega_i\}_{i=1}^n$. The *Voronoi cell* of $\mathbf{x}_i$ is defined by

$$\Omega_i = \{\mathbf{x} \in \mathbb{R}^3 | d(\mathbf{x}, \mathbf{x}_i) \leq d(\mathbf{x}, \mathbf{x}_j), \forall j\}, \tag{1}$$

where $d(\cdot, \cdot)$ denotes euclidean distance.

3.2 Clipped Voronoi Diagrams

Given a connected compact domain $\mathcal{M}$ in 3D, the clipped Voronoi diagram for the sites $\mathbf{X}$ with respect to $\mathcal{M}$ is defined as the intersection between the ordinary Voronoi diagram $\Omega(\mathbf{X})$ and the domain $\mathcal{M}$. It is denoted $\Omega|_{\mathcal{M}} = \{\Omega_i|_{\mathcal{M}}\}_{i=1}^{n}$, where

$$\Omega_i|_{\mathcal{M}} = \{\mathbf{x} \in \mathcal{M} | d(\mathbf{x}, \mathbf{x}_i) \leq d(\mathbf{x}, \mathbf{x}_j), \forall j\}. \quad (2)$$

In the above definition, $\Omega_i|_{\mathcal{M}}$ is a *clipped Voronoi cell* with respect to $\mathcal{M}$, and is the intersection of the Voronoi cell Ω_i and the domain $\mathcal{M}$: $\Omega_i|_{\mathcal{M}} = \Omega_i \cap \mathcal{M}$.

3.3 CVT

If the site of each Voronoi cell coincides with its centroid, then the resulting Voronoi diagram is called a *centroidal Voronoi tessellation*. Starting with an initial set of sites, the CVT can be obtained by minimizing the following energy function [30]:

$$E(\mathbf{X}) = \sum_{i=1}^{n} \int_{\Omega_{i|\mathcal{M}}} \rho(\mathbf{x}) ||\mathbf{x} - \mathbf{x}_i||^2 d\mathbf{x}, \quad (3)$$

where $\rho(\mathbf{x}) > 0$ is a user-defined density function over the input domain. In particular, the CVT is uniform when ρ is a constant.

4 Algorithm Overview

We now present an efficient algorithm to compute the clipped Voronoi diagram with respect to the input 3D domain $\mathcal{M}$. To simplify the computation, we first discretize the input volumetric domain into a tetrahedral mesh.

Our algorithm takes a set of sites $\mathbf{X}$ and a tetrahedral mesh $\mathcal{M} = \{\mathcal{V}, \mathcal{T}\}$ as input, where $\mathcal{V} = \{\mathbf{v}_i\}_{i=1}^{n_v}$ is the set of mesh vertices and $\mathcal{T} = \{\mathbf{t}_i\}_{i=1}^{m}$ is the set of tetrahedral elements. Each tet $\mathbf{t}_i$ is represented by four ordered vertices, with indices 0, 1, 2, and 3. Any combination of three vertices forms a triangular face of the tet; its index is that of the vertex opposite the face. The algorithm consists of the following two steps, discussed in detail in the next two sections:

- Compute the indices of the k nearest sites to each site and tet.
- Compute intersections between Voronoi cells and the input tetrahedral mesh.

We consider and compare two alternative strategies to obtain the intersection volume in the second step: (i) first, compute ordinary Voronoi cells, then clip each cell by the half-spaces defined by the faces of tets so that the part of the cell inside each tet can be obtained, or (ii), following previous approaches, tets are used as initial volumes, and are clipped by half-spaces bounded by the bisectors of a site and its k nearest sites.

5 k-NN

Our approach is based on k-nearest-neighbor search. Several efficient CPU and GPU implementations are available. To avoid frequent data transmission between the CPU and GPU, we prefer to perform all computations on the GPU. As our data are in 3D, a grid-based strategy [31] is normally used for computing the k-NN of each site. However, this strategy does not work well for a small number of sites (i.e., fewer than about 14k) and is unable to handle queries from points not in the input point set. Therefore, for fewer than than 14k sites, we use a simple brute-force strategy [11], [32] instead. Thanks to the highly parallel architecture of GPUs, this brute force strategy can achieve comparable performance to the grid-based strategy in the context of our algorithm.

5.1 Brute-Force Strategy

Given a set of reference points $\mathcal{R} = \{r_i\}_{i=1}^{m}$ in a d-dimensional space and a set of query points $\mathcal{Q} = \{q_i\}_{i=1}^{n}$ in the same space, the k nearest reference points to some $q \in \mathcal{Q}$ can be computed as follows:

1) Compute the distance $d_i = d(q, r_i), 1 \leq i \leq m$.
2) Sort the distances $\{d_i\}_{i=1}^{m}$ in ascending order.
3) Output the indices of the k points in $\mathcal{R}$ with the k lowest distances.

Note that each step above is highly parallelizable, so suitable for GPU implementation.

5.2 Grid-Based Strategy

The grid-based strategy takes a set of 3D points $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^{n}$ as input and produces the indices of the k nearest points to each $\mathbf{x} \in \mathbf{X}$ as output. The input points are first embedded in a 3D grid consisting of several cells by rounding their coordinates, so that each cell contains approximately five points on average. The points are sorted in ascending order of the indices of cells so that all points inside a given cell can be quickly retrieved.

Having built the grid structure, we can find concentric cell rings to visit neighbors contained in the cells for each query point. As we do so, a binary max-heap is maintained to store the k candidates. If the distance between the query point and the current neighbor is less than the maximum distance in the heap, the maximum element is replaced by the neighbor and the heap is adjusted. Successive rings of the query point are visited until k elements are present in the heap and the maximum distance is less than the minimum distance to the next ring.

6 Computing Clipped Voronoi Diagrams

To compute the intersection between Voronoi cells and the tetrahedral mesh, we consider two alternative strategies, namely, *cell-tet* and *tet-cell* strategies, illustrated in Fig. 2 and 3. The Sutherland-Hodgman algorithm [33] is applied to clip a cell by a plane.

6.1 Cell-Tet Strategy

The first step in the cell-tet strategy is to compute the ordinary Voronoi diagram. Subsequently, this is clipped by each tet of the tetrahedral mesh to obtain the intersection volume. Details of these two steps are now discussed.

6.1.1 *Voronoi Cell Construction*

The Voronoi cell of a site is the subset of $\mathbb{R}^3$ consisting of all points closer to this site than any other site. Thus, the Voronoi cell Ω_i of $\mathbf{x}_i$ is the intersection of a set of half-spaces: $\Omega_i = \cap_{j \neq i} \Pi^{+}(i, j)$, where $\Pi^{+}(i, j)$ is the half-space bounded by the bisector of $(\mathbf{x}_i, \mathbf{x}_j)$ that contains the site $\mathbf{x}_i$. However, a cell need not to be clipped by all half-spaces because some distant sites do not contribute to the cell. Consider a bounding

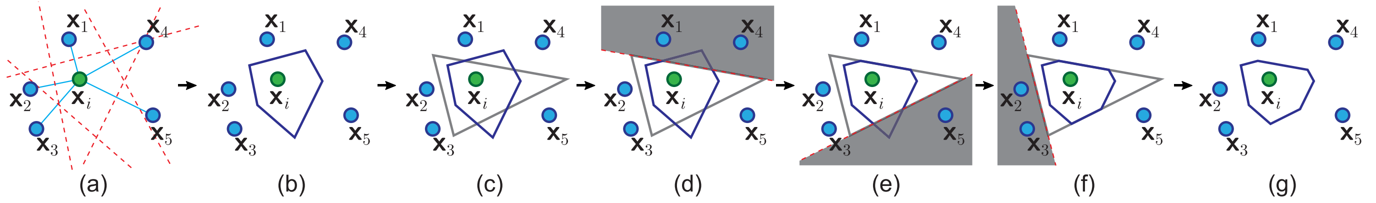

Fig. 2. 2D explanation of cell-tet strategy: (a) the Voronoi cell of $\mathbf{x}_i$ computed by using Algorithm 1, (b) the precomputed Voronoi cell of $\mathbf{x}_i$, (c) a face (i.e., tet in 3D) of which $\mathbf{x}_i$ is one of the k nearest sites, (d–f) the cell clipped by the half-spaces defined by edges 1–3 (i.e., face in 3D), (g) the part of the Voronoi cell inside the face (i.e., tet in 3D).

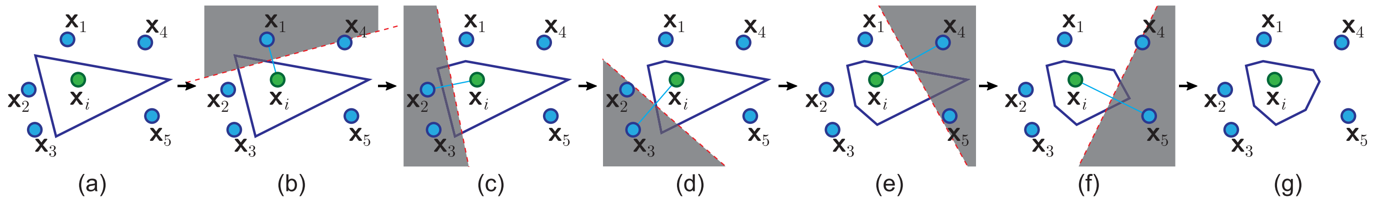

Fig. 3. 2D explanation of the tet-cell strategy: (a) the part initialized by a face (i.e., tet in 3D) of which $\mathbf{x}_i$ is one of the k nearest sites, (b–f) the part clipped by the half-spaces bounded by the bisectors of $\mathbf{x}_i$ and its k_{site} nearest neighbors from near to far, (g) the part of the Voronoi cell inside the face (i.e., tet in 3D).

ball with radius R for the cell computed so far, centered on $\mathbf{x}_i$. A clipping plane will not touch the ball if $d(\mathbf{x}_i, \mathbf{x}_j) > 2R$. The radius R is called the *radius of security* [19]. See Fig. 4. Let $\mathbf{x}_1, \ldots, \mathbf{x}_{10}$ be the 10 nearest neighbors of site $\mathbf{x}_i$, sorted in ascending order of distance to $\mathbf{x}_i$. Each bisector contributing to the cell clips a half-space and forms a Voronoi edge. Sites $\mathbf{x}_7, \ldots, \mathbf{x}_{10}$ are further away from $\mathbf{x}_i$ than $2R$, so do not contribute to the cell, so clipping can stop after reaching the radius of security.

Each Voronoi cell can be constructed independently and in parallel using its nearest neighbors. The maximum number of neighbors used for clipping is denoted k_{site}. Algorithm 1 is used for constructing Voronoi cells, which are stored in global GPU memory for further computation.

Algorithm 1. Compute Voronoi Cells

```
Input : sites $\mathbf{X}$
Output : Voronoi cells $\Omega(\mathbf{X}) = \{\Omega_i\}_{i=1}^n$
 1: find $k_{site}$ nearest sites of each site on the GPU;
 2: for each site $\mathbf{x_i} \in \mathbf{X}$ in parallel do
 3:   $\Omega_i \leftarrow BoundingBox(\mathbf{X})$;
 4:   for $p \leftarrow 1$ to $k_{site}$ do
 5:     $\mathbf{x}_p \leftarrow p$th nearest site of $\mathbf{x}_i$;
 6:     if $SecurityRadiusReached(\Omega_i, \mathbf{x}_p)$ then
 7:       break;
 8:     end if
 9:     $\mathcal{P}_p \leftarrow BisectorPlane(\mathbf{x}_i, \mathbf{x}_p)$;
10:     $\Omega_i \leftarrow ClipByPlane(\Omega_i, \mathcal{P}_p)$;
11:   end for
12:   store $\Omega_i$ in the global memory of the GPU;
13: end for
```

6.1.2 Clipped Voronoi Cell Construction

After computing the ordinary Voronoi diagram, forming tet-cell pairs and obtaining their intersections become the key issues. The centroid $\mathbf{c}_i$ of each tet $\mathbf{t}_i$ is computed, and the k nearest sites of the centroid are queried. Subsequently, for each neighbor site $\mathbf{x}_j$, the tet $\mathbf{t}_i$ and the Voronoi cell Ω_j of $\mathbf{x}_j$ form a tet-cell pair $(\mathbf{t}_i, \Omega_j)$. One thread is created for each tet-cell pair to calculate the part of the cell inside the tet, which is definitely a convex polyhedron. A thread first loads the precomputed cell from global memory to its shared memory, and the cell is clipped by four half-spaces defined by the four faces of the tet. Suppose that $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$ are three vertices of a face, with normal direction $\mathbf{n} = (n_x, n_y, n_z)$ pointing into the tet by the right-hand rule. The half-space equation $\mathcal{P}$ can be defined by one vertex, e.g., $\mathbf{v}_1$, and the normal $\mathbf{n}$

$$\mathbf{n} = (\mathbf{v}_2 - \mathbf{v}_1) \times (\mathbf{v}_3 - \mathbf{v}_1), \tag{4}$$

$$\mathcal{P} : n_x x + n_y y + n_z z - \mathbf{v}_1 \cdot \mathbf{n} > 0. \tag{5}$$

Fig. 5 illustrates the clipping process; pseudo-code is given in Algorithm 2. If the intersection between the tet and the Voronoi cell is not empty, the total volume and barycenter of the clipped Voronoi cell $\Omega_j|_{\mathcal{M}}$ are updated using Algorithm 3 with the help of an *atomicAdd* operation. Note that all of the above calculations are implemented on the GPU to avoid time-consuming CPU-GPU data transmission.

Strategy for Choosing k. Determining the number of neighboring cells which must be clipped by a tet is a key parameter for the proposed algorithm. Assuming that the sampled sites can be regarded as a roughly uniform distribution, a

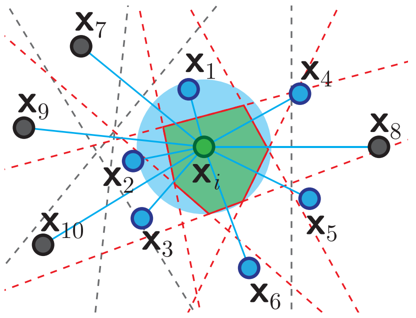

Fig. 4. Voronoi cell (green) of $\mathbf{x}_i$ is the intersection of half-spaces bounded by the bisectors of $\mathbf{x}_i$ and its k_{site} nearest neighbors. Gray sites further from $\mathbf{x}_i$ than twice the radius of the bounding ball (blue) do not contribute to the cell.

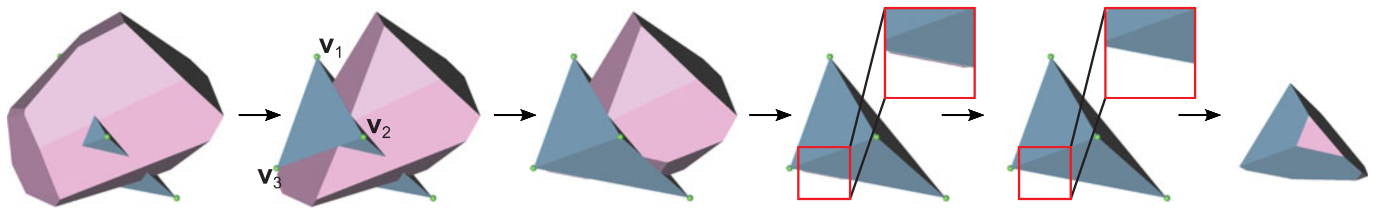

Fig. 5. Clipping a Voronoi cell (pink) by a tet (blue). An ordinary Voronoi cell is clipped by the four faces of the tet in succession. The final part (right) is a subset composed of the faces of the Voronoi cell and tet.

value for k can be determined based on the ratio of the number of sites to the number of tets. To help to apply our algorithm, we give a formula to calculate a default value of k in Eq. (6). However, users can also tune the parameter k manually to adjust the trade-off between efficiency and accuracy.

$$k = \begin{cases} |\mathbf{X}|, & \text{if } |\mathbf{X}| < \lambda \\ \max(\alpha \cdot \lceil \beta \cdot \log_{10}(\gamma \cdot \frac{|\mathbf{X}|}{|\mathcal{T}|} + 1) \rceil, \lambda), & \text{otherwise} \end{cases}. \tag{6}$$

Here, we set $\lambda = 20$, $\alpha = 10$, $\beta = 8$, and $\gamma = 7$ based on our experiments; these value worked well for all of the examples. The coefficient λ is used to control the minimum value of k; d k must be a multiple of coefficient α. For convenience of computation and statistical analysis, α is set to 10. In addition, a scale control coefficient β and a smoothness control coefficient γ are introduced to further adjust growth of k with $|\mathbf{X}|/|\mathcal{T}|$—see the plot of k when $|\mathbf{X}| \geq \lambda$ in Fig. 6. As the advantage of the GPU may be weakened and accuracy may be reduced if k is extremely small, the minimum value of k is set to 20 unless the number of sites is insufficient.

Algorithm 2. Compute Clipped Voronoi Diagrams

```
Input: sites $\mathbf{X}$, tetrahedral mesh $\mathcal{M} = \{\mathcal{V}, \mathcal{T}\}$
Output: the barycenter $\{\mathcal{C}_i\}_{i=1}^n$ and volume $\{V_i\}_{i=1}^n$ of clipped Voronoi cells $\Omega|_\mathcal{M} = \{\Omega_i|_\mathcal{M}\}_{i=1}^n$
 1: Compute the Voronoi cells $\Omega(\mathbf{X}) = \{\Omega_i\}_{i=1}^n$ by Algorithm 1;
 2: $bary\_sum[1..n][x,y,z] \leftarrow 0$; $\{V_i\}_{i=1}^n \leftarrow 0$;
 3: for each tet $\mathbf{t}_i \in \mathcal{T}$ in parallel do
 4:   $\mathbf{c}_i \leftarrow ComputeCentroid(\mathbf{t}_i)$;
 5:   find $k$ nearest sites $\mathbf{N}_i$ of $\mathbf{c}_i$;
 6:   for each site $\mathbf{x}_j \in \mathbf{N}_i$ in parallel do
 7:     $\Omega_j|_{\mathbf{t}_i} \leftarrow \Omega_j$; ▷ load $\Omega_j$ from global memory
 8:     for $p \leftarrow 1$ to 4 do
 9:       $f_p \leftarrow p$th face of $\mathbf{t}_i$;
10:       $\mathcal{P}_p \leftarrow FacePlane(f_p)$;
11:       $\Omega_j|_{\mathbf{t}_i} \leftarrow ClipByPlane(\Omega_j|_{\mathbf{t}_i}, \mathcal{P}_p)$;
12:       if $\Omega_j|_{\mathbf{t}_i} = \phi$ then
13:         break;
14:       end if
15:     end for
16:     if $\Omega_j|_{\mathbf{t}_i} \neq \phi$ then
17:       $UpdateCellAtomically(j, \Omega_j|_{\mathbf{t}_i})$; ▷ Algorithm 3
18:     end if
19:   end for
20: end for
21: for $i \leftarrow 1$ to $n$ in parallel do
22:   $\mathcal{C}_i[x,y,z] \leftarrow bary\_sum[i][x,y,z]/V_i$;
23: end for
```

Algorithm 3. Update the Clipped Voronoi Cell Atomically

```
Input: clipped Voronoi cell's index $j$, a convex polyhedron $\mathbf{P}$
Output: new barycenter $\mathcal{C}_j$ and volume $V_j$
 1: $\mathcal{T}_\mathbf{P} \leftarrow DecomposeIntoTet(\mathbf{P})$;
 2: $bary\_sum\_P[x,y,z] \leftarrow 0$; $vol\_sum\_P \leftarrow 0$;
 3: for each tet $\mathbf{t}_i \in \mathcal{T}_\mathbf{P}$ do
 4:   $\mathbf{c}_i \leftarrow ComputeCentroid(\mathbf{t}_i)$;
 5:   $V_{\mathbf{t}_i} \leftarrow ComputeVolume(\mathbf{t}_i)$
 6:   $bary\_sum\_P[x,y,z] \leftarrow bary\_sum\_P[x,y,z] + \mathbf{c}_i \cdot V_{\mathbf{t}_i}$;
 7:   $vol\_sum\_P \leftarrow vol\_sum\_P + V_{\mathbf{t}_i}$;
 8: end for
 9: $atomicAdd(\mathcal{C}_j, bary\_sum\_P)$;
10: $atomicAdd(V_j, vol\_sum\_P)$;
```

6.2 Tet-Cell Strategy

Note that the cell-tet strategy requires Voronoi cells to be stored in GPU global memory and then loaded them into shared memory. Generally, shared memory is much faster than global memory. Therefore, we propose the alternative tet-cell strategy to compute clipped Voronoi diagrams directly. See Algorithm 4 and Fig. 3.

The same tet-cell nearest pairing algorithm is used as in the cell-tet strategy, the only difference being that each part of the Voronoi cell is initialized by the corresponding tet and clipped by half-spaces bounded by the bisectors directly. Intuitively, this strategy leads to additional clipping when the number of sites is large. The time trade-off between clipping and data transmission is discussed in Section 7.3.

7 Experimental Results

In this section, we evaluate the performance of the proposed algorithm for different numbers of sites and tets and compare it to the state-of-the-art approaches in the open-source library Geogram [15] and hierarchical CVT [34] (HCVT).

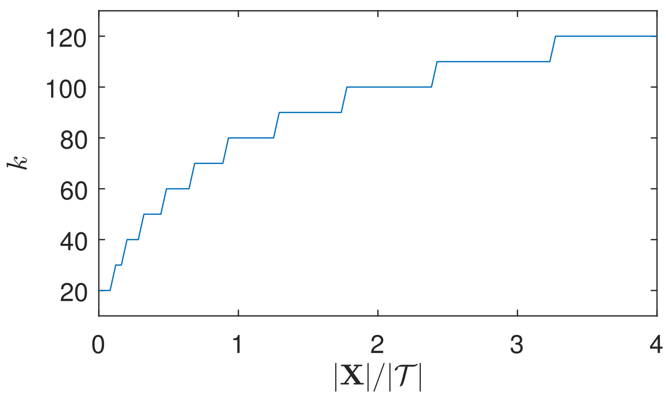

Fig. 6. Stepwise variation of k with $|\mathbf{X}|/|\mathcal{T}|$.

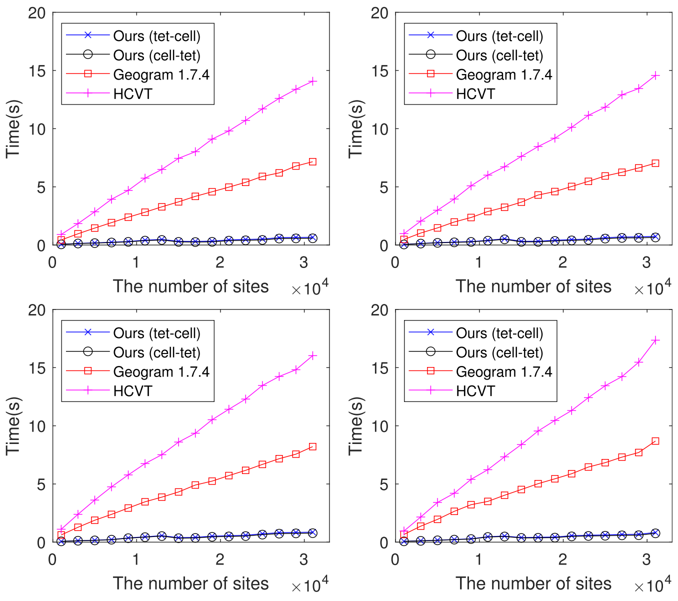

Fig. 7. Comparison between the proposed algorithm, Geogram 1.7.4 [15], and HCVT [34] on the Torus (top-left, 8k tets), Bunny (top-right, 10k tets), Fandisk (bottom-left, 26k tets), and Joint (bottom-right, 31k tets) models for varying numbers of sites. Volume errors are $\leq$ 0.22, 0.13, 0.05, and 0.05 percent, respectively.

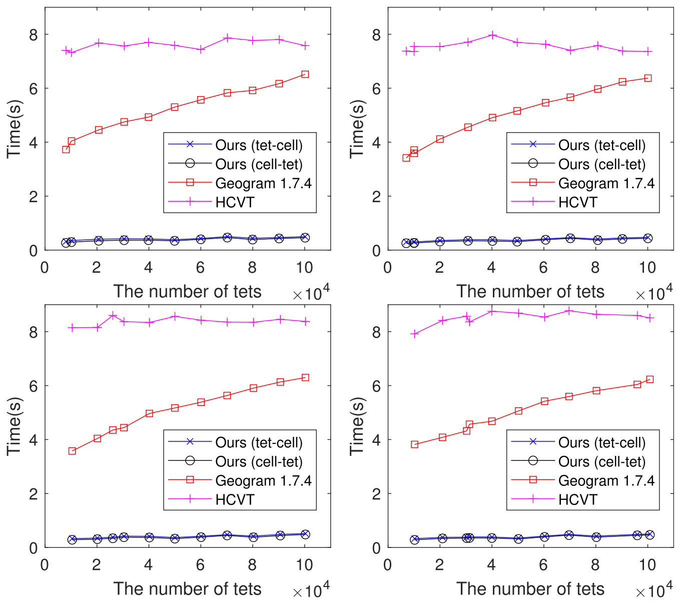

Fig. 8. Comparison between the proposed algorithm, Geogram 1.7.4, and HCVT on the Torus (top-left), Bunny (top-right), Fandisk (bottom-left), and Joint (bottom-right) models, for a fixed number of sites (15k) with varying mesh resolution. Volume errors are $\leq$ 0.09, 0.08, 0.09, and 0.06 percent, respectively.

The proposed algorithm is also applied to Lloyd's algorithm for computing the CVT. Comparisons are made between the tet-cell and the cell-tet strategies in terms of run time and number of clipping operations performed.

Algorithm 4. Compute Clipped Voronoi Diagrams (Tet-Cell)

```
Input : sites $\mathbf{X}$, tetrahedral mesh $\mathcal{M} = \{\mathcal{V}, \mathcal{T}\}$
Output : the barycenter $\{\mathcal{C}_i\}_{i=1}^n$ and volume $\{V_i\}_{i=1}^n$ of clipped Voronoi cells $\Omega|_{\mathcal{M}} = \{\Omega_i|_{\mathcal{M}}\}_{i=1}^n$
1:  find $k_{site}$ nearest sites of each site on the GPU;
2:  $bary\_sum[1..n][x, y, z] \leftarrow 0$; $\{V_i\}_{i=1}^n \leftarrow 0$;
3:  for each tet $\mathbf{t}_i \in \mathcal{T}$ in parallel do
4:    $\mathbf{c}_i \leftarrow ComputeCentroid(\mathbf{t}_i)$;
5:    find $k$ nearest sites $\mathbf{N}_i$ of $\mathbf{c}_i$;
6:    for each site $\mathbf{x}_j \in \mathbf{N}_i$ in parallel do
7:      $\Omega_j|_{\mathbf{t}_i} \leftarrow \mathbf{t}_i$;
8:      for $p \leftarrow 1$ to $k_{site}$ do
9:        $\mathbf{x}_p \leftarrow p$th nearest site of $\mathbf{x}_j$;
10:       if $SecurityRadiusReached(\Omega_j|_{\mathbf{t}_i}, \mathbf{x}_p)$ then
11:         break;
12:       end if
13:       $\mathcal{P}_p \leftarrow BisectorPlane(\mathbf{x}_j, \mathbf{x}_p)$;
14:       $\Omega_j|_{\mathbf{t}_i} \leftarrow ClipByPlane(\Omega_j|_{\mathbf{t}_i}, \mathcal{P}_p)$;
15:       if $\Omega_j|_{\mathbf{t}_i} = \phi$ then
16:         break;
17:       end if
18:     end for
19:     if $\Omega_j|_{\mathbf{t}_i} \neq \phi$ then
20:       $UpdateCellAtomically(j, \Omega_j|_{\mathbf{t}_i})$; $\triangleright$ Algorithm 3
21:     end if
22:   end for
23: end for
24: for $i \leftarrow 1$ to $n$ in parallel do
25:   $\mathcal{C}_i[x, y, z] \leftarrow bary\_sum[i][x, y, z]/V_i$;
26: end for
```

The proposed algorithm was implemented in C++. All results were computed on a Ubuntu 16.04 server with a 2.60 GHz Intel Xeon E5-2690 CPU with 256 GB memory, and an NVIDIA TITAN RTX GPU with 24 GB memory, using CUDA version 10.1. In our experiments, the maximum number of neighbors used to clip a Voronoi cell is set to 90, which is sufficient for randomly distributed sites in most situations. Another important parameter of our algorithm is the number k of neighboring cells to be clipped by a tet, which strongly influences the performance, as does the number of tets. The parameter k is set according to Eq. (6) unless specified otherwise.

7.1 Performance

The efficiency of our algorithm is first demonstrated. Given an input domain represented by a tetrahedral mesh and a set of sites generated randomly inside the domain, we evaluated the time to construct clipped Voronoi diagrams. Each test was repeated 10 times and results were averaged (see Fig. 7). The superior speed of the proposed algorithm is increasingly evident as the number of sampled sites increases from 1k to 31k. We also observe that tet-cell and cell-tet strategies both achieve similar performance. The reason for the small drop in performance between 13k and 15k sites is that grid-based k-NN search does not work well if the number of sites is too small. Thus, we use the brute-force strategy when the number of sites is fewer than 14k.

Results in Fig. 8 are given for a fixed number of sites with varying input mesh size, from 10k to 100k elements. The time taken by our algorithm is almost constant. As the ratio of the number of sites to the number of tets decreases, the default value of k decreases simultaneously to reduce the computational cost, while not affecting accuracy in most situations. Note that the HCVT method uses a surface mesh rather than a tetrahedral mesh as input, so its performance is also independent of the number of tets.

TABLE 1
Results for Computing Clipped Voronoi Diagrams for Various Models

Model	$\|\mathcal{T}\|$	$\|\mathbf{X}\|$	Geogram (s)	HCVT (s)	Ours						
					E_{vol}	k	k-NN (s)	Tet-Cell		Cell-Tet	
								Clip (s)	Total (s)	Clip (s)	Total (s)
Torus	8k	5k	1.48	2.83	0.03%	60	0.14	0.04	**0.18**	0.03	**0.17**
Bunny	10k	10k	2.62	5.30	0.05%	60	0.25	0.06	**0.31**	0.03	**0.28**
Fandisk	26k	15k	4.39	8.50	0.04%	50	0.20	0.12	**0.32**	0.07	**0.27**
Joint	31k	35k	9.31	17.91	0.02%	70	0.57	0.22	**0.79**	0.12	**0.69**
Homer	41k	30k	8.88	16.30	0.05%	50	0.41	0.21	**0.62**	0.11	**0.52**
Gargoyle	199k	10k	6.16	8.70	0.14%	20	0.44	0.20	**0.64**	0.19	**0.63**

$|\mathcal{T}|$ = number of tets, $|\mathbf{X}|$ = number of sites, and E = relative volume error.

However, it generally takes longer than Geogram and our algorithm.

Further results for other models are given in Table 1; they show that our approach can be up to an order of magnitude faster than state-of-the-art CPU methods. We also break down the total running time of our algorithm into its two main components, the time for k-NN search and the time for clipping Voronoi cells. The first component is unchanged for both clipping strategies.

To evaluate the accuracy of the proposed algorithm, we determined the relative volume error $E_{\text{vol}} = |V_{\text{CVD}} - V_{\mathcal{M}}|/V_{\mathcal{M}}$, where V_{CVD} and $V_{\mathcal{M}}$ are the total volume of the clipped Voronoi diagram and the input mesh respectively (we express E_{vol} as a percentage). Specifically, we find that both CPU methods result in low volume error. Thus, we use the cell volume computed by Geogram 1.7.4 as ground truth and visualize the volume error for each individual Voronoi cell (see Fig. 10). The main causes of volume error are:

k too small. If too few neighbors are considered in nearest neighbor search, and the sites are distributed in an uneven manner or slivers exist in the input mesh, some Voronoi cells that intersect a given tet may be missed, reducing the volume of the clipped Voronoi cell.

Unbounded Boundary Cells. Unclipped Voronoi cells whose sites lie on the convex hull are unbounded. As shown

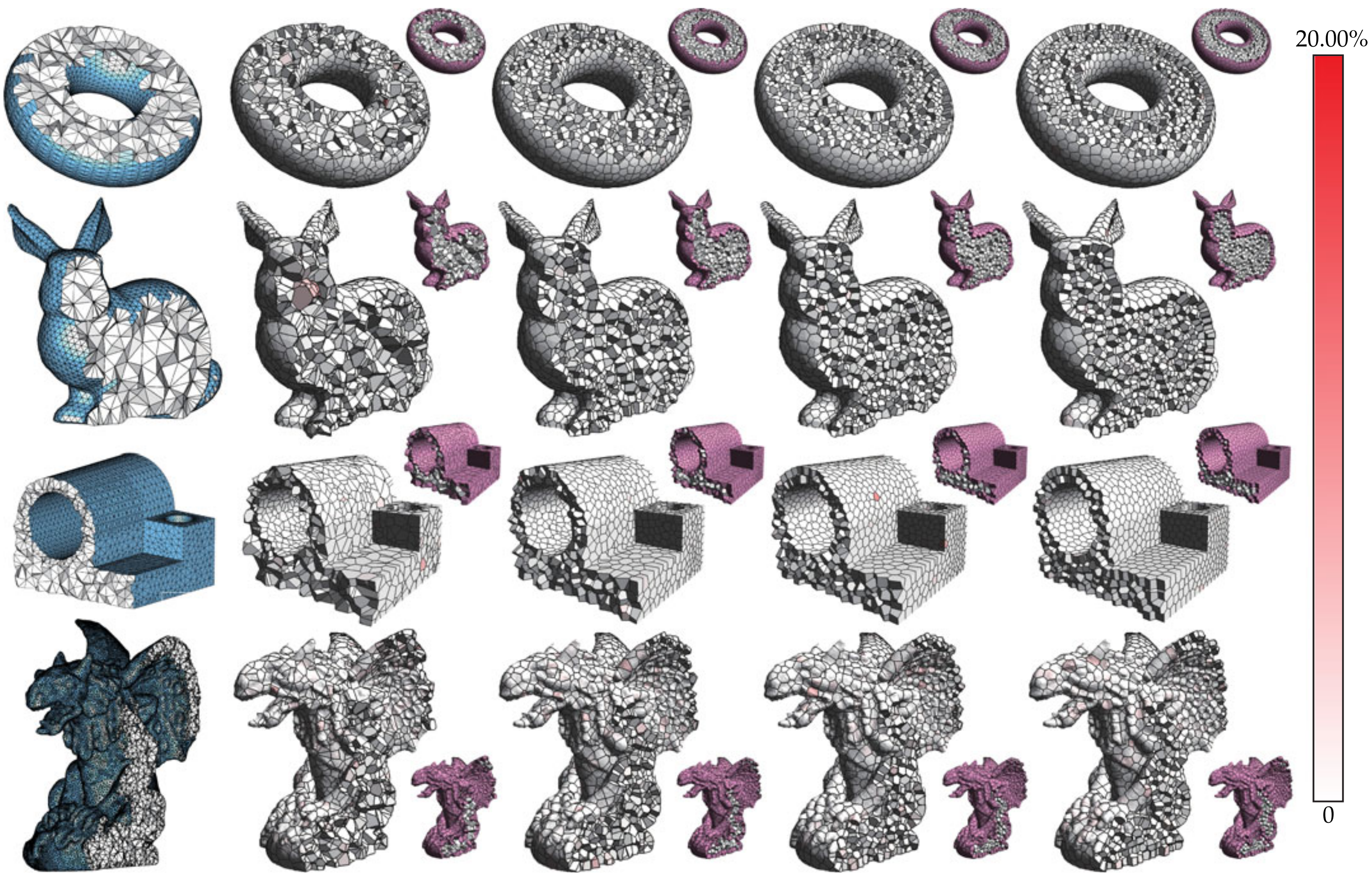

Fig. 9. Results of Lloyd iterations for the Torus (8k tets), Bunny (10k tets), Joint (31k tets), and Gargoyle (199k tets) models, using the proposed method. From left to right: input tetrahedral mesh, clipped Voronoi diagram for 3k sampled sites, results after 5, 20, and, 120 iterations. The total volume errors are $\leq$ 0.05, 0.04, 0.04, and 0.20 percent, respectively. The volume error for each cell is visualized in the figure. The 120 iterations take 10.51s, 9.50s, 10.43s, and 32.65s in total, about an order of magnitude faster than a CPU algorithm.

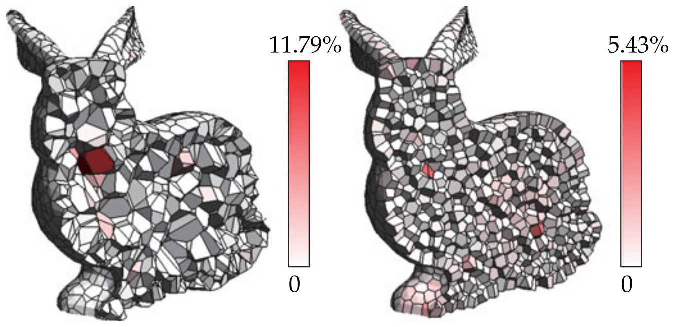

Fig. 10. Volume error for individual Voronoi cells. Volume error generally decreases as the distribution of sites becomes more uniform. ($|\mathcal{T}| = 10\text{k}$, $|\mathbf{X}| = 3\text{k}$, $k = 40$).

in Fig. 11, the unbounded part of the cell may have undesirable intersections with other parts of the input domain. If k is too large or concave parts of the input domain are very close, the clipped Voronoi cell may be too large as a result. To illustrate this kind of error, we set $k = 160$, four times the default value, to produce the result in Fig. 11.

To determine the influence of the parameter k, different values of k were tested in the range 10–120, and computation times and volume errors recorded. As shown in Fig. 12, the computation time for both clipping strategies increases almost linearly with k, while volume errors decrease exponentially. Clearly, k should not be taken too small, to avoid large errors, but taking k too large just expends time for little additional accuracy. In Fig. 12, we set default values of k to 100 and 90 for the Torus model and the Bunny model, respectively, which works well in most situations. Given that each part computed by both strategies is the same theoretically, the difference between the volume errors of the tet-cell and cell-tet strategies is minimal.

7.2 Application to CVT

The centroidal Voronoi tessellation [30] is a particular type of Voronoi tessellation in which each site coincides with the centroid of its Voronoi cell. The CVT has numerous applications in computer graphics, e.g., to meshing, sampling, and stippling. The prevailing method used to compute the CVT is Lloyd's algorithm. It computes the clipped Voronoi diagram of a given set of sites, and then moves each site toward the centroid of its Voronoi cell, iterating these two steps until convergence. Computation of the clipped Voronoi diagrams takes most of the time in this process. Our proposed GPU-based algorithm can thus significantly speed it up while maintaining the volume error within an acceptable range. Fig. 9 shows the results of applying Lloyd's algorithm using a random set of sites inside various input domains. The distribution of sites gradually becomes uniform as iteration proceeds.

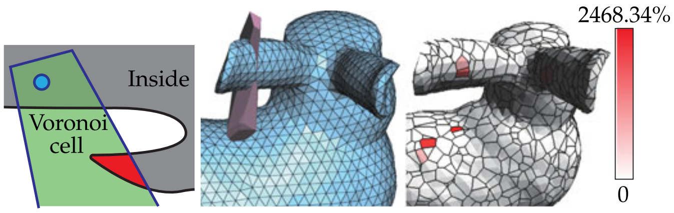

Fig. 11. Left: 2D example showing volume error caused by intersection between the input domain and the unbounded part of the Voronoi cell (red). Center: Part of the bunny's ear is added to the clipped Voronoi cell due to excessive k. Right: Volume error ($|\mathcal{T}| = 10\text{k}$, $|\mathbf{X}| = 3\text{k}$, $k = 160$).

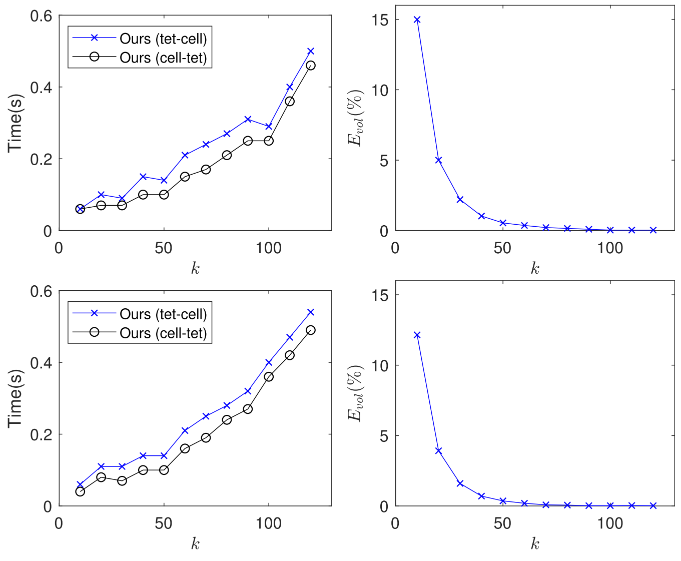

Fig. 12. Influence of k on computation time (left) and volume error (right), for k from 10 to 120, using Torus (top, 8k tets) and Bunny (bottom, 10k tets) models, with 15k sites.

7.3 Tet-Cell Versus Cell-Tet

We now compare the strengths and weaknesses of the two clipping strategies, again using Lloyd's algorithm, by considering running time and the number of clipping operations performed. Fig. 13(left) shows the results of this experiment. The number of clipping operations performed by the tet-cell strategy drops drastically during the first five iterations, whereas the number for the cell-tet strategy remains almost unchanged. The radius of security is reached earlier as the distribution of sites becomes more uniform, which benefits the tet-cell strategy.

Indeed, even though the tet-cell strategy performs more clipping operations than the cell–tet strategy, the former is faster when the number of sites is small (Fig. 13(top)).

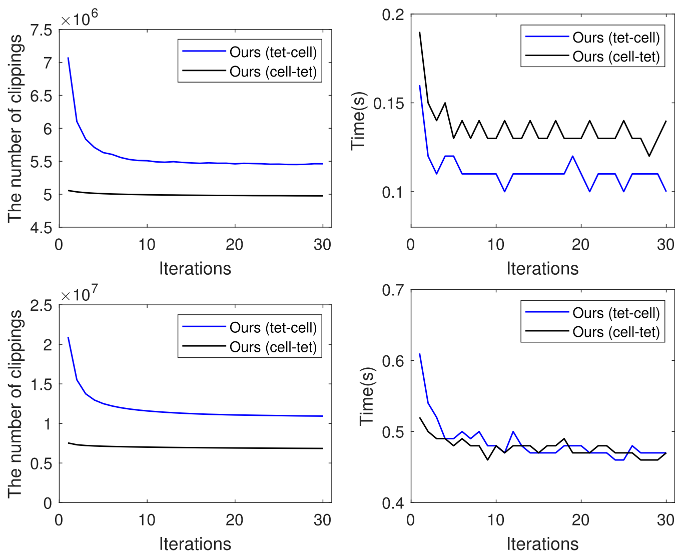

Fig. 13. Comparison of tet-cell and cell-tet strategies on the Homer (41k tets) model with 1k (top) and 30k (bottom) sites, using $k = 50$.

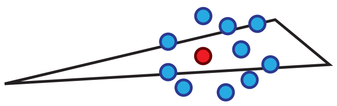
Fig. 14. Sliver and its 10 nearest sites.

Fig. 13(bottom) shows that the computation time of the tet-cell strategy increases and becomes comparable to that of the cell-tet strategy as the number of sites increases. The cell-tet strategy needs to load precomputed Voronoi cells from global memory into shared memory, which is a time-consuming operation. Moreover, the cost of data transmission between global and shared memory accounts for a large proportion of the time when the number of sites is relatively small, and vice versa. Therefore, the tet-cell strategy is to be preferred for cases with a small number of sites, whereas the cell-tet strategy should be used when the number is large.

8 Conclusion

This study proposes an efficient GPU-based algorithm using a tetrahedral mesh to compute 3D clipped Voronoi diagrams. It can be implemented using either a tet-cell or a cell-tet strategy. Both implementations are fast and can be directly used in applications such as CVT. To enhance the usability of the proposed algorithm, a heuristic is given to determine the number of neighboring cells which should be considered when clipping by a tet. Experimental results show that the proposed method can compute 3D clipped Voronoi diagrams effectively, and it is approximately an order of magnitude faster than state-of-the-art CPU methods.

Limitations and Future Work. However, as the proposed algorithm is based on k-NN search, it works well only if the input sites are evenly distributed and the number of slivers in the input tetrahedral mesh is manageable (see Fig. 14). Otherwise, parts of cells may be lost as the nearest pairing algorithm cannot cope with such situations well. We would like to further improve the robustness of our algorithm and explore more applications in the future. Moreover, computing 3D power diagrams on the GPU is a feasible extension of our work.

Acknowledgments

The authors would like to thank the reviewers for their valuable comments which greatly improved the article. This work was supported by the National Natural Science Foundation of China (61772523 and 61802406), the Beijing Natural Science Foundation (L182059), the CCF-Tencent Open Research Fund (RAGR20190105), the Open Research Fund Program of State key Laboratory of Hydroscience and Engineering, Tsinghua University (sklhse-2020-D-07), the Open Project Program of State Key Laboratory of Virtual Reality Technology and Systems, Beihang University (VRLAB2019B02), and the PKU-Baidu Fund (2019BD001).

References

[1] F. Aurenhammer, “Voronoi diagrams - A survey of a fundamental geometric data structure,” *ACM Comput. Surv.*, vol. 23, no. 3, pp. 345–405, 1991.

[2] D.-M. Yan, B. Lévy, Y. Liu, F. Sun, and W. Wang, “Isotropic remeshing with fast and exact computation of restricted Voronoi diagram,” *Comput. Graph. Forum*, vol. 28, no. 5, pp. 1445–1454, 2009.

[3] B. Lévy and Y. Liu, “L_p centroidal Voronoi tesselation and its applications,” *ACM Trans. Graph.*, vol. 29, no. 4, pp. 119:1–119:11, 2010.

[4] D.-M. Yan, W. Wang, B. Lévy, and Y. Liu, “Efficient computation of 3D clipped Voronoi diagram,” in *Proc. Int. Conf. Geometric Model. Process.*, 2010, pp. 269–282.

[5] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu, *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*, vol. 501. Hoboken, NJ, USA: Wiley, 2009.

[6] G. Rong and T. S. Tan, “Jump flooding in GPU with applications to Voronoi diagram and distance transform,” in *Proc. Symp. Interactive 3D Graph. Games*, 2006, pp. 109–116.

[7] N. Ray, D. Sokolov, S. Lefebvre, and B. Lévy, “Meshless Voronoi on the GPU,” *ACM Trans. Graph.*, vol. 37, no. 6, pp. 265:1–265:12, 2018.

[8] G. Rong, Y. Liu, W. Wang, X. Yin, X. Gu, and X. Guo, “GPU-assisted computation of centroidal Voronoi tessellation,” *IEEE Trans. Vis. Comput. Graphics*, vol. 17, no. 3, pp. 345–356, Mar. 2011.

[9] J. Han, D.-M. Yan, L. Wang, and Q. Zhao, “Computing restricted Voronoi diagram on graphics hardware,” in *Proc. 25th Pacific Conf. Comput. Graph. Appl.*, 2017, pp. 23–26.

[10] Y.-S. Leung, X. Wang, Y. He, Y.-J. Liu, and C. C.-L. Wang, “A unified framework for isotropic meshing based on narrowband Euclidean distance transformation,” *Comput. Vis. Media*, vol. 1, pp. 239–251, 2015.

[11] V. Garcia, E. Debreuve, F. Nielsen, and M. Barlaud, “K-nearest neighbor search: Fast GPU-based implementations and application to high-dimensional feature matching,” in *Proc. IEEE Int. Conf. Image Process.*, 2010, pp. 3757–3760.

[12] D.-M. Yan, W. Wang, B. Lévy, and Y. Liu, “Efficient computation of clipped Voronoi diagram for mesh generation,” *Comput.-Aided Des.*, vol. 45, no. 4, pp. 843–852, 2013.

[13] CGAL User and Reference Manual, 4.14.3 ed., 2019. [Online]. Available: https://doc.cgal.org/4.14.3/Manual/packages.html

[14] C. B. Barber, D. P. Dobkin, D. P. Dobkin, and H. Huhdanpaa, “The quickhull algorithm for convex hulls,” *ACM Trans. Math. Softw.*, vol. 22, no. 4, pp. 469–483, 1996.

[15] Geogram: A programming library of geometric algorithms, 2019. [Online]. Available: http://alice.loria.fr/software/geogram/doc/html/index.html

[16] F. Aurenhammer, R. Klein, and D.-T. Lee, *Voronoi Diagrams and Delaunay Triangulations*. Singapore: World Scientific Publishing Company, 2013.

[17] H. Edelsbrunner and N. R. Shah, “Triangulating topological spaces,” *Int. J. Comput. Geometry Appl.*, vol. 7, no. 4, pp. 365–378, 1997.

[18] Q. Du, M. D. Gunzburger, and L. Ju, “Constrained centroidal Voronoi tesselations for surfaces,” *SIAM J. Sci. Comput.*, vol. 24, no. 5, pp. 1488–1506, 2003.

[19] B. Lévy and N. Bonneel, “Variational anisotropic surface meshing with Voronoi parallel linear enumeration,” in *Proc. 21st Int. Meshing Roundtable*, 2012, pp. 349–366.

[20] D.-M. Yan, G. Bao, X. Zhang, and P. Wonka, “Low-resolution remeshing using the localized restricted Voronoi diagram,” *IEEE Trans. Vis. Comput. Graphics*, vol. 20, no. 10, pp. 418–1427, Oct. 2014.

[21] L. Ma, J. Guo, D.-M. Yan, H. Sun, and Y. Chen, “Instant stippling on 3D scenes,” *Comput. Graph. Forum*, vol. 37, no. 7, pp. 255–266, 2018.

[22] T. M. Chan, J. Snoeyink, and C. Yap, “Output-sensitive construction of polytopes in four dimensions and clipped Voronoi diagrams in three,” in *Proc. 6th Annu. ACM-SIAM Symp. Discrete Algorithms*, 1995, pp. 282–291.

[23] K. Zhou, Q. Hou, R. Wang, and B. Guo, “Real-time KD-tree construction on graphics hardware,” *ACM Trans. Graph.*, vol. 27, no. 5, 2008, Art. no. 126.

[24] M. Gao, T.-T. Cao, A. Nanjappa, T.-S. Tan, and Z. Huang, “gHull: A GPU algorithm for 3D convex hull,” *ACM Trans. Math. Softw.*, vol. 40, no. 1, pp. 3:1–3:19, 2013.

[25] T.-T. Cao, A. Nanjappa, M. Gao, and T.-S. Tan, “A GPU accelerated algorithm for 3D Delaunay triangulation,” in *Proc. 18th Meeting ACM SIGGRAPH Symp. Interactive 3D Graph. Games*, 2014, pp. 47–54.

[26] H.-H. Hsieh and W.-K. Tai, “A simple GPU-based approach for 3D Voronoi diagram construction and visualization,” *Simul. Modelling Pract. Theory*, vol. 13, no. 8, pp. 681–692, 2005.

[27] G. Rong, M. Jin, L. Shuai, and X. Guo, “Centroidal Voronoi tessellation in universal covering space of manifold surfaces,” *Comput. Aided Geometric Des.*, vol. 28, no. 8, pp. 475–496, 2011.

[28] L. Shuai, X. Guo, and M. Jin, "GPU-based computation of discrete periodic centroidal Voronoi tessellation in hyperbolic space," *Comput.-Aided Des.*, vol. 45, no. 2, pp. 463–472, 2013.
[29] X. Liu and D.-M. Yan, "Computing 3D clipped Voronoi diagrams on GPU," in *Proc. SIGGRAPH Asia 2019 Posters*, 2019, pp. 9:1–9:2.
[30] Q. Du, V. Faber, and M. Gunzburger, "Centroidal Voronoi tessellations: Applications and algorithms," *SIAM Rev.*, vol. 41, no. 4, pp. 637–676, 1999.
[31] R. Hoetzlein, "Fast fixed-radius nearest neighbors: Interactive million-particle fluids," in *Proc. GPU Technol. Conf.*, vol. 18, 2014, p. 2.
[32] V. Garcia, E. Debreuve, and M. Barlaud, "Fast k nearest neighbor search using GPU," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Workshops*, 2008, pp. 1–6.
[33] I. E. Sutherland and G. W. Hodgman, "Reentrant polygon clipping," *Commun. ACM*, vol. 17, no. 1, pp. 32–42, 1974.
[34] L. Wang, F. Hétroy-Wheeler, and E. Boyer, "A hierarchical approach for regular centroidal Voronoi tessellations," *Comput. Graph. Forum*, vol. 35, no. 1, pp. 152–165, 2016.

Xiaohan Liu received the BS degree in software engineering from the Nanjing University of Aeronautics and Astronautics, China, in 2017. He is currently working toward the MS degree at the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences and the University of Chinese Academy of Sciences, China. His supervisor is Prof. Dong-Ming Yan. His research interests include computer graphics, geometry processing and computer vision.

Lei Ma received the BS degree from Zhejiang University, China, the MS degree from Digital ART (Augmented Reality Tech) Laboratory of Shanghai Jiao Tong University, China, and the PhD degree from the State Key Laboratory of Computer Science, Chinese Academy of Sciences, China. He is currently an associate researcher of computer science at Peking University, China, also working for the National Engineering Lab for Video technology of China. During 2010-2012, he worked for Autodesk China Research and Development Center as a graphic engineer. He was also a co-founder of an AR start-up. His research interests include realistic image synthesis, geometry processing, rendering of complex scenes, virtual reality, and artificial intelligence.

Jianwei Guo received the bachelor's degree from Shandong University, China, in 2011, and the PhD degree in computer science from CASIA, in 2016. He is currently an associate professor with the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, China. His research interests include computer graphics, geometry processing and 3D shape analysis.

Dong-Ming Yan received the bachelor's and master's degrees from Tsinghua University, China, in 2002 and 2005, respectively, and the PhD degree from Hong Kong University, Hong Kong, in 2010. He is currently a professor with the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, China. His research interests include computer graphics, 3D vision, and geometric processing.