# Towards Voronoi Diagrams of Surface Patches

Pengfei Wang, Jiantao Song, Lei Wang, Shiqing Xin, Dong-Ming Yan, Shuangmin Chen, Changhe Tu, Wenping Wang

**Abstract**—Extraction of a high-fidelity 3D medial axis is a crucial operation in CAD. When dealing with a polygonal model as input, ensuring accuracy and tidiness becomes challenging due to discretization errors inherent in the mesh surface. Commonly, existing approaches yield medial-axis surfaces with various artifacts, including zigzag boundaries, bumpy surfaces, unwanted spikes, and non-smooth stitching curves. Considering that the surface of a CAD model can be easily decomposed into a collection of surface patches, its 3D medial axis can be extracted by computing the Voronoi diagram of these surface patches, where each surface patch serves as a generator. However, no solver currently exists for accurately computing such an extended Voronoi diagram. Under the assumption that each generator defines a linear distance field over a sufficiently small range, our approach operates by tetrahedralizing the region of interest and computing the medial axis within each tetrahedral element. Just as SurfaceVoronoi computes surface-based Voronoi diagrams by cutting a 3D prism with 3D planes (each plane encodes a linear field in a triangle), the key operation in this paper is to conduct the hyperplane cutting process in 4D, where each hyperplane encodes a linear field in a tetrahedron. In comparison with the state-of-the-art, our algorithm produces better outcomes. Furthermore, it can also be used to compute the offset surface.

**Index Terms**—digital geometry processing, CAD models, medial axis (MA), Voronoi diagram, piecewise linear field.

## 1 Introduction

Voronoi diagrams serve the purpose of partitioning a given space into subregions based on proximity. Beyond their direct applications in proximity queries and collision detection [1, 2], Voronoi diagrams find utility in a diverse range of fields, including surface reconstruction [3], robot motion planning [4], non-photorealistic rendering [5], surface simplification [6], mesh generation [7], and shape analysis [8], among others.

Voronoi diagrams showcase numerous variants depending on specific domains, metrics, and generator types, with the most commonly employed version defined in Euclidean spaces using point generators. In digital geometry processing, a fundamental research task involves partitioning a 2-manifold surface into curved Voronoi cells based on geodesic distances. In this scenario, the 2-manifold surface serves as the domain, the geodesic distance functions as the metric, and the user-specified point set acts as the generators. A recent advancement by Xin et al. [9] introduces an extensible approach known as SurfaceVoronoi for computing Voronoi diagrams on surfaces and their variants. SurfaceVoronoi operates under the assumption that mesh triangles are small in size and that the triangle-wide geodesic distance field, provided by a single generator, can be considered linear. Leveraging this assumption, SurfaceVoronoi enables each generator to simultaneously propagate distances until all contributing generators for each triangle are identified. Ultimately, for each triangle, SurfaceVoronoi calculates the surface-restricted Voronoi structure by elevating each 2D linear field to a 3D plane and extracting the lower envelope of a roof-like structure.

This paper explores the 3D Voronoi diagram of a set of non-intersecting surface patches. We aim to enhance SurfaceVoronoi to tackle this challenge, recognizing that the extension must contend with the increase in dimensions compared to surface-restricted Voronoi diagrams, which are inherently 2D. In practical applications, the Voronoi diagram of surface patches or even 3D objects proves significantly useful for understanding how objects interact or relate spatially. For instance, Zhao et al. [10] proposed using bisectors between two 3D objects to describe their topological relationships. In the realm of robotics and autonomous systems, the extended Voronoi diagram facilitates the rapid identification of safe paths to avoid collisions or obstacles between 3D objects [11].

While the resulting distance field from a single surface-patch generator can be arbitrarily complex in the entire $\mathbb{R}^3$ space, it can be as simple as a linear function when confined to a small range. This allows us to encode the distance field within a small tetrahedral cell, contributed by a single generator, using a straightforward quadruple. In its nature, this representation signifies a 4D plane, with the first three dimensions denoting coordinates and the fourth dimension illustrating distance variation. Initially, we generate the initial 4D volume rooted at a tetrahedral element by sweeping the base tetrahedron along the fourth dimension. Our approach begins with the tetrahedralization of the space of interest. Similar to SurfaceVoronoi [9], the first stage involves propagating straight-line distances from the generators until no generator can offer a smaller distance for any tetrahedral element. Subsequently, we preserve the surviving generators and their corresponding linear distance fields for each tetrahedron. The second stage involves decomposing each tetrahedron into sub-domains through a sequence of 4D hyperplane cutting operations. Finally, the lower envelope of the 4D roof-like structure, when projected back into 3D, defines the decomposition configuration of the base tetrahedron. Given that the surface of a CAD model can be readily decomposed into a collection of simple patches, we innovatively apply the extended Voronoi diagram to

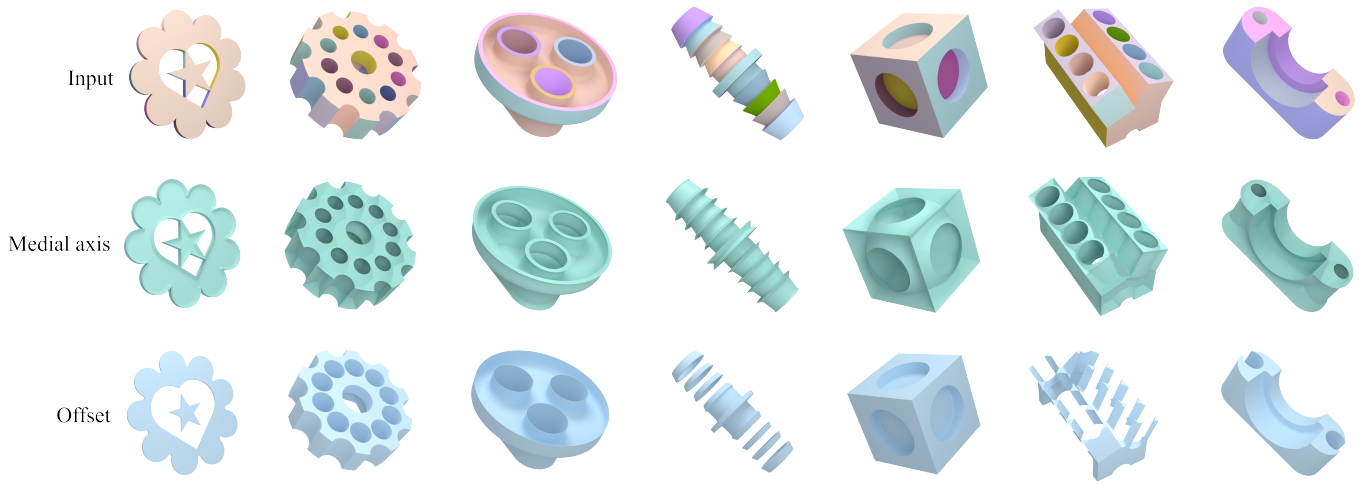



Fig. 1. This paper suggests computing the medial axis of CAD models via Voronoi diagrams of surface patches, treating each patch as an individual computational unit. Interestingly, this computational technique can also be used to calculate offsets. Top: Input polygonal models (surface patches are visualized in a color-coded style). Middle: Medial-axis surfaces. Bottom: Inward offset surfaces.

compute medial-axis surfaces. Extensive experimental results demonstrate that our medial-axis extraction algorithm significantly outperforms the state-of-the-art in terms of accuracy and noise insensitivity. Furthermore, our algorithm can even be used to compute the offset surface.

Our contributions are three-fold:

- We extend SurfaceVoronoi to compute the Voronoi diagram of a collection of surface patches, addressing a challenging task in past research.
- The fundamental operations are elevated from 3D to 4D, enabling the computation of the extended Voronoi diagram, confined within a tetrahedron, through a sequence of 4D hyperplane cutting.
- We innovatively apply the new algorithm to compute medial-axis surfaces and demonstrate its superior performance. Additionally, we discuss more potential application scenarios.

## 2 RELATED WORKS

### 2.1 Conventional Voronoi Diagrams

Suppose that we have a set of generators $\mathcal{S} = \{s_i\}_{i=1}^n$ in a given domain $\Omega$ that is equipped with a metric function $\mathbf{D}$, the Voronoi diagram involves partitioning $\Omega$ into regions such that the generator $s_i$ dominates a region

$$\{x \in \Omega \mid \mathbf{D}(s_i, x) \leq \mathbf{D}(s_j, x), j \neq i\}. \tag{1}$$

The most prevalent version assumes that $\Omega$ represents Euclidean spaces, and the generators are exclusively points. The definition of Voronoi diagrams may vary with domains, metrics, and generator types. Voronoi diagrams, as a fundamental tool, have found widespread applications in computer graphics [12, 13, 14, 15] and image processing [16, 17].

There is a substantial body of literature on the computation of Voronoi diagrams, particularly for computing the Voronoi diagram of point-type generators in Euclidean spaces. The most commonly used methods include the divide-and-conquer scheme [18], the incremental construction method [19], and Fortune's sweep line algorithm [20]. Moreover, it should be noted that the lifting technique transforms the Voronoi diagram problem into a convex hull problem by elevating the computation to a higher-dimensional space [21]. Subsequently, parallelization techniques have been applied to the construction of Voronoi diagrams [22, 23, 24].

Voronoi diagrams find diverse applications in digital geometry processing. One common application involves defining the influence area of a mesh vertex, relying on the principles of Voronoi diagrams. Choi et al. [25] utilized Voronoi diagrams in a novel 3D printing approach. This method, compared to traditional lattice methods, effectively reduces stress concentration and eliminates the need for additional support structures. Xu et al. [13] leveraged Voronoi diagrams to predict normals for undirected point clouds. To tackle the challenge of polygon meshes being unsuitable for learning-based applications, Maruani et al. [26] introduced a unique and differentiable surface representation using Voronoi diagrams.

### 2.2 Extended Voronoi Diagrams

In practical applications, Voronoi diagrams may have variant versions based on specific requirements. For instance, generators may include line segments [27, 28, 29]. Xu et al. [30] investigated the computation of geodesic Voronoi diagrams on a mesh surface, employing polylines as generators. Zong et al. [12] adopted the concept of the Voronoi diagram between triangles and proposed an efficient point-to-mesh distance query algorithm. While Voronoi diagrams with line-segment or surface-patch generators can be reduced to a standard Voronoi diagram by sampling generators into a finite set of points, the results are less accurate, and computation is time-consuming [31].

Conceptually, geodesic distances drive the Voronoi decomposition of a curved surface, but computing geodesic Voronoi diagrams is generally time-consuming [32]. The Restricted Voronoi Diagram (RVD) [33, 34] considers surface-restricted decomposition based on the proximity between surface points. However, the dual of the RVD may not be a manifold triangle mesh. Wang et al. [15] proposed a set of provably effective strategies for addressing this issue. Xin et

al. [9] introduced a triangle-based lifting technique, enabling the computation of various surface-based Voronoi diagrams. This method allows for the use of different metrics to measure the distance between two points, such as exact geodesic distances [35] or Euclidean distances. In addition, there are some methods based on vector heat [36] and PDE-based [37] approaches for computing geodesic Voronoi diagrams. These methods are computationally fast but sensitive to the quality of triangulation.

There is a deep link between Voronoi diagrams and medial axis surfaces [31]. Most of the existing approaches [38, 39] for computing medial-axis surfaces require sampling points from the surface and computing a conventional Voronoi diagram as part of the initialization process. However, the discrete sampling operation can be time-consuming and computationally demanding, especially when the number of sample points is large. Additionally, it may compromise the quality of the resulting medial-axis surfaces or even lead to failure on certain thin-plate models. Yan et al. [40] observed the medial axis of a voxel shape can be accurately approximated by the interior Voronoi diagram of the boundary vertices, referred to as the voxel core. Their approach demonstrates exceptional accuracy in approximating the medial axis of smooth shapes, ensuring topological correctness when given a sufficiently high-resolution voxelization of the shape. Wang et al. [41] observed that the surface-restricted power cell of each medial sphere indicates the tangential surface regions in contact, aiding in classifying a medial sphere as being on a medial sheet, a seam, or a junction. To the best of our knowledge, this method represents the state-of-the-art in extracting the medial axis surface of a CAD model. In this paper, we extend SurfaceVoronoi [9] to address this challenging problem and demonstrate its utilities in computing medial-axis surfaces and offset surfaces of a CAD model.

## 3 Methodology

### 3.1 Review on SurfaceVoronoi

SurfaceVoronoi [9] processes a polygonal surface as its input and operates under the assumption that the geodesic distance field at the triangle level, provided by a single generator, can be treated as linear. The SurfaceVoronoi algorithm primarily consists of two stages: distance over-propagation and incremental half-plane cutting.

3.1.0.1 Distance over-propagation.: Given a set of source points $\mathcal{S} = \{s_i\}_{i=1}^n$ on a triangle mesh surface, the algorithm allows each generator to propagate distances simultaneously while maintaining priorities through a priority queue. Distances propagate across triangles, and each triangle retains a list of surviving generators.

Without loss of generality, let $f = \triangle v_1v_2v_3$ represent one of the triangles of the surface. The distances of $v_1, v_2, v_3$ are initialized to $\infty$. When a new generator propagates its distances to $f$, the decision of whether the generator should be kept in $f$ is based on competition with existing generators. Let $s_1, s_2, \cdots, s_k$ be the surviving generators in $f$, and $s_{k+1}$ be the new generator. If there exists a surviving generator $s_i$ $(1 \leq i \leq k)$ such that:

$$\begin{aligned} \mathbf{D}(s_{k+1}, v_1) &\geq \mathbf{D}(s_i, v_1), \\ \mathbf{D}(s_{k+1}, v_2) &\geq \mathbf{D}(s_i, v_2), \\ \mathbf{D}(s_{k+1}, v_3) &\geq \mathbf{D}(s_i, v_3), \end{aligned} \tag{2}$$

then $s_{k+1}$ is labeled as an invalid generator for $f$, preventing its propagation to neighboring faces. At the end of the distance over-propagation stage, each triangle accumulates a list of surviving generators and corresponding distance triples.

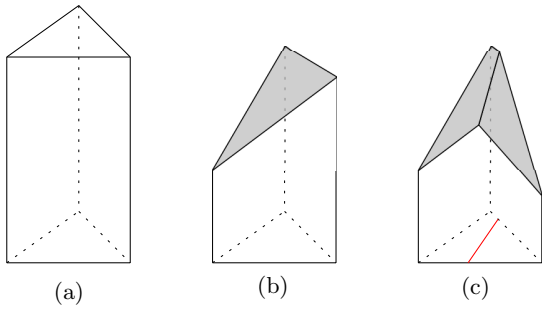



Fig. 2. SurfaceVoronoi involves a step of incremental plane cutting, as illustrated in (a) the original triangular prism, (b) after one cutting operation, and (c) after two cutting operations. The red-colored line segment represents the triangle-restricted Voronoi diagram.

3.1.0.2 Incremental half-plane cutting.: It is straightforward to map $f = \triangle v_1v_2v_3$ onto a 2D plane, with the new vertex coordinates being

$$(x_1, y_1), (x_2, y_2), (x_3, y_3). \tag{3}$$

Each surviving generator of $f$ defines a lifting 3D plane $\pi$ passing through

$$(x_1, y_1, d_1), (x_2, y_2, d_2), (x_3, y_3, d_3), \tag{4}$$

where $d_1, d_2, d_3$ are the distances from the generator to $v_1, v_2, v_3$, respectively. $\pi$ can be implicitly represented as an equation:

$$d = ax + by + c. \tag{5}$$

The lower envelope of the planes established by the surviving generators of $f$ shapes a roof-like structure, using $\triangle v_1v_2v_3$ as the base. Consequently, through a series of plane-cutting operations, the roof-like structure can be defined, revealing the $f$-restricted Voronoi diagram; Refer to Fig. 2.

### 3.2 Linear Representation of Scalar Field in a Tetrahedron

Similar to SurfaceVoronoi, we assume that the tetrahedron-range distance field for a single source $s_i$ (potentially a surface patch in this paper) undergoes linear changes. Let $t$ be a tetrahedron with four vertices

$$v_1(x_1, y_1, z_1), v_2(x_2, y_2, z_2), v_3(x_3, y_3, z_3), v_4(x_4, y_4, z_4), \tag{6}$$

and the distance values given by the generator $s_i$ be $d_1^i, d_2^i, d_3^i, d_4^i$, respectively. The linear change restricted in $t$ can be characterized by an equation

$$d = a_ix + b_iy + c_iz + w_i, \tag{7}$$

where $a_i, b_i, c_i, w_i$ can be found by solving

$$\begin{pmatrix} x_{v_1} & y_{v_1} & z_{v_1} & 1 \\ x_{v_2} & y_{v_2} & z_{v_2} & 1 \\ x_{v_3} & y_{v_3} & z_{v_3} & 1 \\ x_{v_4} & y_{v_4} & z_{v_4} & 1 \end{pmatrix} \begin{pmatrix} a_i \\ b_i \\ c_i \\ w_i \end{pmatrix} = \begin{pmatrix} d_1^i \\ d_2^i \\ d_3^i \\ d_4^i \end{pmatrix}. \quad (8)$$

The coefficient matrix is invertible as long as the tetrahedron is not degenerate, ensuring the existence and uniqueness of the solution.

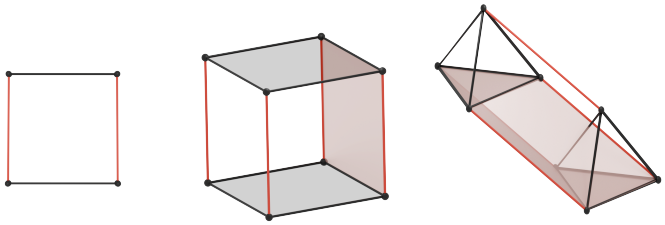
To this end, Eqn. (7) defines a hyperplane in 4D, where the first three dimensions represent coordinates, and the fourth dimension illustrates distance variation. We assume that the tetrahedron $t$ has two surviving generators $s_i$ and $s_j$. The intersection of their corresponding hyperplanes can be represented as follows:

$$\begin{cases} a_i x + b_i y + c_i z + w_i = d \\ a_j x + b_j y + c_j z + w_j = d. \end{cases} \quad (9)$$

By subtracting the two equations, $d$ is eliminated:

$$(a_i - a_j)x + (b_i - b_j)y + (c_i - c_j)z + (w_i - w_j) = 0, \quad (10)$$

which indicates that the intersection between 4D linear fields defines a 3D plane.



(a) (b) (c)

Fig. 3. Just as a rectangle (a) and a box (b) can be generated by sweeping a line segment and a rectangle along an additional dimension, respectively, we envision the creation of an initial 4D triangular prism by sweeping a 3D tetrahedron along the fourth dimension (c). In (b) and (c), a 2D side face and a 3D side face are visualized in brown.

### 3.3 Geometric View on Tetrahedron-range Linear Scalar Field

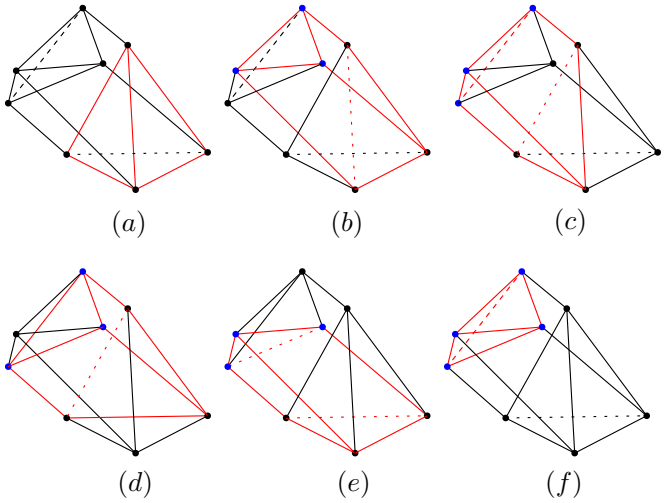
Fig. 3 illustrates the generation of a 4D triangular prism by sweeping a 3D tetrahedron $t$ along the fourth dimension (extending a triangular prism from 3D to 4D). We need to initialize a 4D triangular prism for each 3D tetrahedron. As $d \geq 0$ in our context, we elevate $t$ to 4D and consider it as the bottom face of the 4D triangular prism:

$$(x_1, y_1, z_1, 0), (x_2, y_2, z_2, 0), (x_3, y_3, z_3, 0), (x_4, y_4, z_4, 0). \quad (11)$$

The top face of the 4D triangular prism is:

$$(x_1, y_1, z_1, \infty), (x_2, y_2, z_2, \infty), (x_3, y_3, z_3, \infty), (x_4, y_4, z_4, \infty). \quad (12)$$

The 4D triangular prism has four vertical edges, respectively connecting $(x_i, y_i, z_i, 0)$ and $(x_i, y_i, z_i, \infty)$, $i = 1, 2, 3, 4$. It is bounded by a total of six hyperplanes, *i.e.*, the top, the bottom, and four side tetrahedral faces. In 4D space, each vertex is formed by the intersection of four hyperplanes. Taking the initial 4D triangular prism as an example, each vertex is created by the intersection of three hyperplanes generated by the prism's three side faces, along with either the base face or the top face. Similarly, edges in 4D space are formed through a comparable process. We illustrate how an initial 4D triangular prism is intersected by a 4D hyperplane, as shown in Figure 4.



(a) (b) (c) (d) (e) (f)

Fig. 4. Illustration of how an initial 4D triangular prism is intersected by a 4D hyperplane: (a) Base face. (b-e) Four 3D side faces generated by sweeping the tetrahedral sides along the fourth dimension. (f) Top face.

## 4 ALGORITHM

By considering a set of generators (typically surface patches) in $\mathbb{R}^3$ as input, we consider computing the Voronoi diagram for these generators. Our algorithm begins with a tetrahedralization of the input, allowing the computation of the complex Voronoi diagram within a tetrahedron range. Subsequently, the Voronoi diagram is calculated through distance over-propagation and incremental hyperplane cutting. The individual steps are detailed in the following subsections.

### 4.1 Tetrahedralization

Our algorithm operates within a tetrahedralized space. In general, the computation of the medial axis should be confined to the interior volume enclosed by the input surface. Therefore, the triangles of the original triangle mesh must be embedded into the tetrahedralization results, and each tetrahedron must be sufficiently small. We use fTetWild [42] for this purpose. Additional evaluations regarding how the size of the tetrahedralization affects accuracy can be found in Section 5.2.

### 4.2 Incremental Hyperplane Cutting

At this point, we assume that the distance between any tetrahedral vertex and any surface patch has been computed, with the distance metric being the Euclidean distance. This task will be addressed in the next subsection. Following this, we discuss the hyperplane cutting operation within a tetrahedral element.

Given a quadruple $(x, y, z, d)$, the quadruple is considered to be on the upper side of the hyperplane

$$\pi_i : d = a_i x + b_i y + c_i z + w_i$$

if and only if

$$d \geq a_i x + b_i y + c_i z + w_i.$$

To trace the lower envelope, it is essential to maintain the combination structure, comprising vertices, edges (connecting two vertices), polygonal faces (formed by three or more edges in a loop), polyhedral faces (having four or more polygonal faces topologically equivalent to a polyhedron), and the 4D convex volume. In our implementation, we must simultaneously retain vertices, edges, polygonal faces, and polyhedral faces.

When a new hyperplane $\pi$ is introduced, the process begins by eliminating existing vertices, edges, polygonal faces, and polyhedral faces above $\pi$ through simple tagging. The next step involves identifying edges that intersect with $\pi$. For the edge $e = v_1v_2$, if $\pi(v_1) = d_{v_1} - (ax_{v_1} + by_{v_1} + cz_{v_1} + w) > 0$ and $\pi(v_2) = d_{v_2} - (ax_{v_2} + by_{v_2} + cz_{v_2} + w) < 0$, a new vertex $v$ is obtained using the formula:

$$v = \frac{\pi(v_1)}{\pi(v_1) - \pi(v_2)} v_2 - \frac{\pi(v_2)}{\pi(v_1) - \pi(v_2)} v_1. \tag{13}$$

Following this, it is necessary to update the influenced polygonal faces and polyhedral faces intersecting with $\pi$. To elaborate further, if a polygonal face intersects with the hyperplane $\pi$ at two points, then connecting these intersections is necessary to form a new edge. Additionally, if $\pi$ intersects a sequence of polygonal faces, connecting these intersections in a circular order results in the formation of a new 2D side face. Following a similar process, 3D side faces can also be computed. Finally, each 3D side face is determined by two 4D hyperplanes. If the two hyperplanes that define a 3D side face do not belong to the initial six hyperplanes, the resulting side face contributes to the lower envelope. The lower envelope is then projected back onto the bottom tetrahedron by eliminating the fourth dimension, resulting in a Voronoi-like structure. It is also worth noting that the final structure within a tetrahedral element consists of polygons.

Furthermore, although several computational geometry libraries (e.g., CGAL [43]) provide robust geometric predicates and kernels, indiscriminately replacing double with exact data types may incur substantial algorithmic inefficiencies. Thus, we need to develop robust algorithmic implementations.

4.2.0.1 Handling numerical issues.: When computing the intersection between a hyperplane and an edge with two endpoints $v_1, v_2$, it is crucial to determine the side of each endpoint. According to Eqn. (13), if $\pi(v_1)$ and $\pi(v_2)$ are close to 0, there may be a numerical issue. Therefore, we introduce a tolerance $\epsilon$ to measure the degree to which $\pi(v_1)$ and $\pi(v_2)$ are close to 0. We handle the numerical issues by considering the following 8 cases:

1) $\left|\pi(v_1)\right| \geq \epsilon$, $\left|\pi(v_2)\right| \geq \epsilon$ and $\pi(v_1) \times \pi(v_2) < 0$: Compute the intersection following Eqn. (13).
2) $\pi(v_1) \geq \epsilon$ and $\pi(v_2) \geq \epsilon$: Deem the segment $v_1v_2$ to be above $\pi$ and discard the segment.
3) $\pi(v_1) \leq -\epsilon$ and $\pi(v_2) \leq -\epsilon$: Allow the segment $v_1v_2$ to survive.
4) $\pi(v_1) \geq \epsilon$ and $\left|\pi(v_2)\right| < \epsilon$: Deem the segment $v_1v_2$ to be above $\pi$ and discard the segment.
5) $\pi(v_1) \leq -\epsilon$ and $\left|\pi(v_2)\right| < \epsilon$: Compute the intersection following Eqn. (13).
6) $\left|\pi(v_1)\right| < \epsilon$ and $\pi(v_2) \geq \epsilon$: Deem the segment $v_1v_2$ to be above $\pi$ and discard the segment.
7) $\left|\pi(v_1)\right| < \epsilon$ and $\pi(v_2) \leq -\epsilon$: Compute the intersection following Eqn. (13).
8) $\left|\pi(v_1)\right| < \epsilon$ and $\left|\pi(v_2)\right| < \epsilon$: Deem the segment $v_1v_2$ to be above $\pi$ and discard the segment.

In our experimental setting, the value of $\epsilon$ is set to $10^{-9}$. Based on our tests, most numerical issues can be addressed in this way. Despite its practical usefulness, introducing tolerance may not consistently resolve all numerical issues. For example, we have identified some rare cases where a hyperplane intersects with existing planes in more than two points. In the event of such occurrences, we need to seek a more robust implementation.

4.2.0.2 Robust implementation.: Considering that in our algorithm, two key operations include:

1) Identifying on which side a point is located with regard to a 4D plane.
2) Finding the intersection between an edge and a 4D plane.

To overcome possible error accumulation, we encode each intersection $\mathbf{v} \triangleq (x, y, z, d)^T$ with four 4D planes, similar to [44], rather than directly using the approximate coordinates. Each of the four 4D planes has an implicit form:

$$\pi_i : \mathbf{g}_i^T \begin{pmatrix} x \\ y \\ z \\ d \end{pmatrix} + w_i = 0, \quad i = 1, 2, 3, 4. \tag{14}$$

To this end, the point $\mathbf{v}$ satisfies

$$\mathbf{A}\mathbf{v} + \mathbf{w} = \mathbf{0}, \tag{15}$$

where $\mathbf{A} = \{\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3, \mathbf{g}_4\}^T$ and $\mathbf{w} = \{w_1, w_2, w_3, w_4\}^T$.

To determine the side of $v$ with regard to a new plane $\pi$, we need to check whether $\mathbf{g}^T\mathbf{v} + w$ is positive or not, where $\pi$ is defined by $\mathbf{g}$ and $w$. In other words, we need to compute the sign of the following expression:

$$\mathbf{g}^T\mathbf{A}^{-1}(-\mathbf{w}) + w. \tag{16}$$

If the value of the above expression is greater than or equal to zero, a new vertex will be generated. Unlike the approximate incremental cutting method, we do not explicitly compute the positions of new vertices (i.e., intersections between line segments and hyperplanes). Instead, we encode the new vertex by the four hyperplanes that define it. This technique avoids the need to explicitly compute intersection points. We also recommend using an exact numerical data type (e.g., Gmpq in CGAL) to ensure computational accuracy. It is important to note that using an exact data type to directly compute intersections can result in increasingly complex rational representations, which may significantly raise computational costs. However, encoding each intersection using four hyperplanes effectively mitigates this issue.

### 4.3 Distance Field Propagation

In SurfaceVoronoi, a priority queue is utilized to maintain the morphology of wavefronts propagating from near to far, allowing for the simultaneous handling of distances between different generators and triangles. In this process, the computation of the geodesic distance and the distance propagation are coupled. In our scenario, however, the computation of the medial axis involves only Euclidean distances. Therefore, we adopt a different strategy for inferring distances. It requires the following steps to finish the distance computation.

**Initialization.** We create a BVH structure for the entire surface $\mathcal{S}$. Additionally, we create a BVH structure for each of its constituent patches $\gamma_i$, enabling efficient distance queries. Each patch $\gamma_i$ is composed of triangles, with the surface $\mathcal{S}$ being the union of all such patches $\{\gamma_i\}$. After that, with the support of the BVH of $\mathcal{S}$, we find the closest point $v'$ belonging to $\mathcal{S}$ for each tetrahedral vertex $v$. At the same time, we record the surface patch that accommodates $v'$.

**Distance query.** For each tetrahedron with four vertices $v_1$,$v_2$,$v_3$,$v_4$, we perform the following operations:

Step 1. Initialize a standard queue $\mathcal{Q}$ to accommodate $v_1, v_2, v_3, v_4$, and the surface patch set $\Gamma$ to include the surface patches that provide distances to $v_1, v_2, v_3, v_4$.

Step 2. Take out the top vertex $v$ in $\mathcal{Q}$ and query the minimum distance from $v$ to $\mathcal{S}$ using the corresponding BVH. If $v$'s nearest surface patch, say, $\gamma'$, does not belong to $\Gamma$, perform hyperplane cutting using the 4D plane defined by $\gamma'$. Push all newly generated vertices to $\mathcal{Q}$.

Step 3. If $\mathcal{Q}$ is empty, the process terminates; otherwise, go to Step 2.

Obviously, the above algorithm does not depend on the order in which the tetrahedra are visited. Therefore, it naturally lends itself to a parallel implementation.

### 4.4 Error Analysis

In this section, we provide an upper bound for the error introduced by the linear approximation of a distance field within a tetrahedron.

We denote any point inside the tetrahedron as $\mathbf{x}$, with $d(\mathbf{x})$ representing the true distance value, and $\tilde{d}(\mathbf{x})$ representing the distance value obtained through linear approximation. They satisfy $|\nabla d(\mathbf{x})| = 1$ and $|\tilde{d}(\mathbf{x})| \leq 1$, respectively. The error function is defined as:

$$e(\mathbf{x}) = d(\mathbf{x}) - \tilde{d}(\mathbf{x}). \tag{17}$$

Our goal is to estimate the upper bound of $|e(\mathbf{x})|$. The gradient of the error function is given by:

$$\nabla e(\mathbf{x}) = \nabla d(\mathbf{x}) - \nabla \tilde{d}(\mathbf{x}). \tag{18}$$

According to the properties of gradients, we have:

$$\begin{aligned} |\nabla e(\mathbf{x})| &= |\nabla d(\mathbf{x}) - \nabla \tilde{d}(\mathbf{x})| \\ &\leq |\nabla d(\mathbf{x})| + |\nabla \tilde{d}(\mathbf{x})| \\ &\leq 1 + 1 = 2. \end{aligned} \tag{19}$$

Let $\mathbf{v}_t = \{v_i\}_{i=1}^4$ represent the four vertices of the tetrahedron. Then, the distance error for any vertex within the tetrahedron satisfies

$$|e(\mathbf{x})| \ \leq \ \min_{v \in \mathbf{v}_t} 2 * \|x - v\|$$

Therefore, if we denote the radius of the circumscribed sphere of the tetrahedron by $h$, the maximum error between the linear distance field and the exact distance field within the tetrahedron is $2h$. Thus, in general, smaller and well-shaped tetrahedra yield more accurate linear approximation results. However, it is worth noting that the accuracy of the results is independent of the resolution of the triangular mesh. Specifically, when a surface patch is nearly planar, our approach is particularly accurate because the true distance field is linear, which aligns well with our linear approximation.

## 5 EVALUATION

Our algorithm was implemented using C++ on a platform equipped with a 3.4 GHz AMD Ryzen 9 5950X 16-Core CPU, 64GB of memory, and the Windows 11 operating system. We utilized double precision for data representation and employed the numerical issue handling method described in the paper. Most of the models used in the experiments are sourced from the ABC Dataset [45].

### 5.1 Variant Voronoi Diagrams

Our method supports the computation of various Voronoi diagram variants. In Fig. 5, we use four identical Koala models as generators and compute four different Voronoi diagrams, namely:

- Ordinary Voronoi Diagram (VD),
- Power Diagram [46],
- Additively Weighted Voronoi Diagram [47],
- Multiplicatively Weighted Voronoi Diagram [48].

In the following, we briefly introduce their definitions except for the most traditional version, *i.e.*, Ordinary Voronoi Diagram.

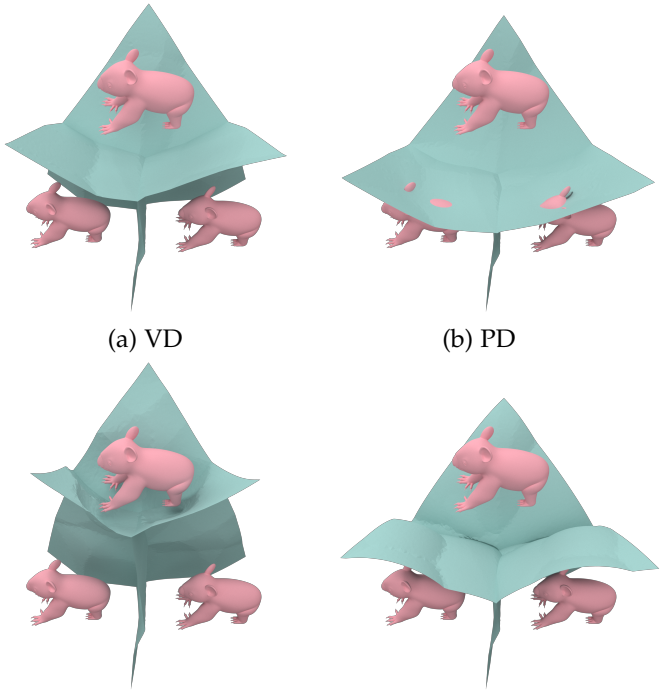
**Power Diagram (PD)** considers both the proximity to points and the influence of weights or powers associated with the points. They have applications in areas such as facility location optimization, where the weight or power of a point represents its importance or capacity. Power diagrams allow for more flexible and customized partitioning of space. The control region of $s_i$ is defined as

$$Cell_{\text{PD}}(s_i) = \{p \mid \mathbf{D}(s_i, p)^2 - w_i^2 \leq \mathbf{D}(s_j, p)^2 - w_j^2, i \neq j\}.$$

**Additively Weighted VD (AWVD)** extends the concept of Voronoi diagrams by assigning weights to represent attributes or values associated with the points. AWVD is useful for spatial interpolation, density estimation, and decision-making problems where the combined influence of multiple factors is considered. The control region of $s_i$ is defined as

$$Cell_{\text{AWVD}}(s_i) = \{p \mid \mathbf{D}(s_i, p) + w_i \leq \mathbf{D}(s_j, p) + w_j, i \neq j\}.$$

**Multiplicatively Weighted VD (MWVD)** incorporates weights as multiplicative factors instead of additive factors. This variation is particularly relevant in applications



(a) VD (b) PD (c) AWVD (d) MWVD

Fig. 5. Variant versions of Voronoi diagram. Here we take four identical Koala models as generators.

where the weights represent scaling factors or proportional relationships. MWVD has applications in fields such as computational physics, where the weights may represent physical properties or scaling factors for interactions. The control region Cell of $s_i$ is defined as

$$Cell_{\mathrm{MWVD}}(s_i) = \{p \mid \mathbf{D}(s_i, p) \cdot w_i \leq \mathbf{D}(s_j, p) \cdot w_j, i \neq j\}.$$

## 5.2 Medial Axis

5.2.0.1 Relevant approaches for comparison.: Numerous approaches [40, 41, 49] have been proposed for the medial-axis surfaces problem. The relevant approaches for comparison include:

1) VoxelCore: Yan et al. [40] suggested identifying core voxels (deemed to be located on the medial-axis surface) from all voxels.
2) MATFP: Wang et al. [41] introduced a method for computing the medial axis of CAD models. MATFP preserves both external and internal features, but the features have to be captured by a seam tracing algorithm.

MATFP requires a step of pre-detecting sharp edges and corners of a CAD model. It initializes with a sampling approach and optimizes their positions before constructing a medial mesh from the updated sphere candidates using restricted regular triangulation. To the best of our knowledge, MATFP represents the state-of-the-art in this field. Similar to MATFP, our approach assumes that the CAD model has been pre-decomposed into multiple surface patches by [50]. Additionally, our algorithm employs fTetWild [42] to tetrahedralize the interior of the input model.

TABLE 1
Time statistics (in seconds) of medial axis calculation for the models shown in Fig. 6.

| | | model 1 | model 2 | model 3 | model 4 | model 5 |
|---|---|---|---|---|---|---|
| VoxelCore | | 1.726 | 1.587 | 0.576 | 2.351 | 1.111 |
| MATFP | Medial Mesh Initialization | 4.153 | 16.877 | 3.588 | 16.980 | 115.272 |
| | Thinning | 1.537 | 186.675 | 2.010 | 88.692 | 1643.8 |
| Ours | fTetwild | 12.013 | 31.969 | 13.531 | 43.297 | 28.812 |
| | Propagation, Cutting | 2.359 | 4.156 | 1.646 | 5.969 | 5.282 |

5.2.0.2 Visual comparison.: Fig. 6 presents a visual comparison among VoxelCore [40], MATFP [41], and our approach. It can be observed that VoxelCore has at least two disadvantages. Firstly, when the resolution of voxels is insufficient, the accuracy of VoxelCore is significantly reduced. Secondly, the resulting medial axis exhibits many spikes, failing to fully capture the sharp features of CAD models. The results produced by MATFP successfully capture both the external and internal features of the medial axis. Nonetheless, the method involves a step of tracking stitching curves on the medial axis, during which the positions of the sampled points need to be relocated. This can potentially lead to numerous wrinkles on the surface of the medial axis and a significant number of long, thin triangles. In contrast, calculating the medial axis as the Voronoi diagram of patches cleverly avoids this issue. Additionally, the results calculated by MATFP inevitably contain self-intersections, whereas our results do not exhibit this problem.

5.2.0.3 Run-time performance.: Before conducting comparisons, we first analyze the complexity of the algorithm. Consider the scenario of computing the medial axis for a model with $n$ points within a tetrahedral mesh containing $n_t$ elements. The time complexity for querying the distance from any arbitrary point in space to the surface of the model is $O(\log n)$, which is achieved by using a spatial data structure to accelerate distance queries. Since the volume of each tetrahedron is relatively small, the number of hyperplanes required for incremental cutting within a single tetrahedron remains constant. Thus, the incremental cutting within a single tetrahedron can be completed in $O(1)$ time. Therefore, the overall complexity of the incremental cutting process is approximately $O(n_t \log n)$. For specific models, the runtime of the propagation and incremental cutting of the algorithm increases linearly with the resolution of the tetrahedral mesh. This means that as the tetrahedral mesh resolution increases, the computational time also increases proportionally, but at a linear rate.

In our methodology, we employ fTetwild as the tetrahedralization solver, setting the target edge length parameter to $0.015$. Simultaneously, for MATFP, we set the downsampling percentage to $0.1$, and for the Voxel Core method, we adjust the voxel sizes to $128^3$. These settings basically ensure a fair comparison by maintaining similar resolutions across methods. It's important to highlight that, within our algorithm, operations such as distance field propagation and incremental hyperplane cutting can be performed independently for different tetrahedra, allowing for a high degree of

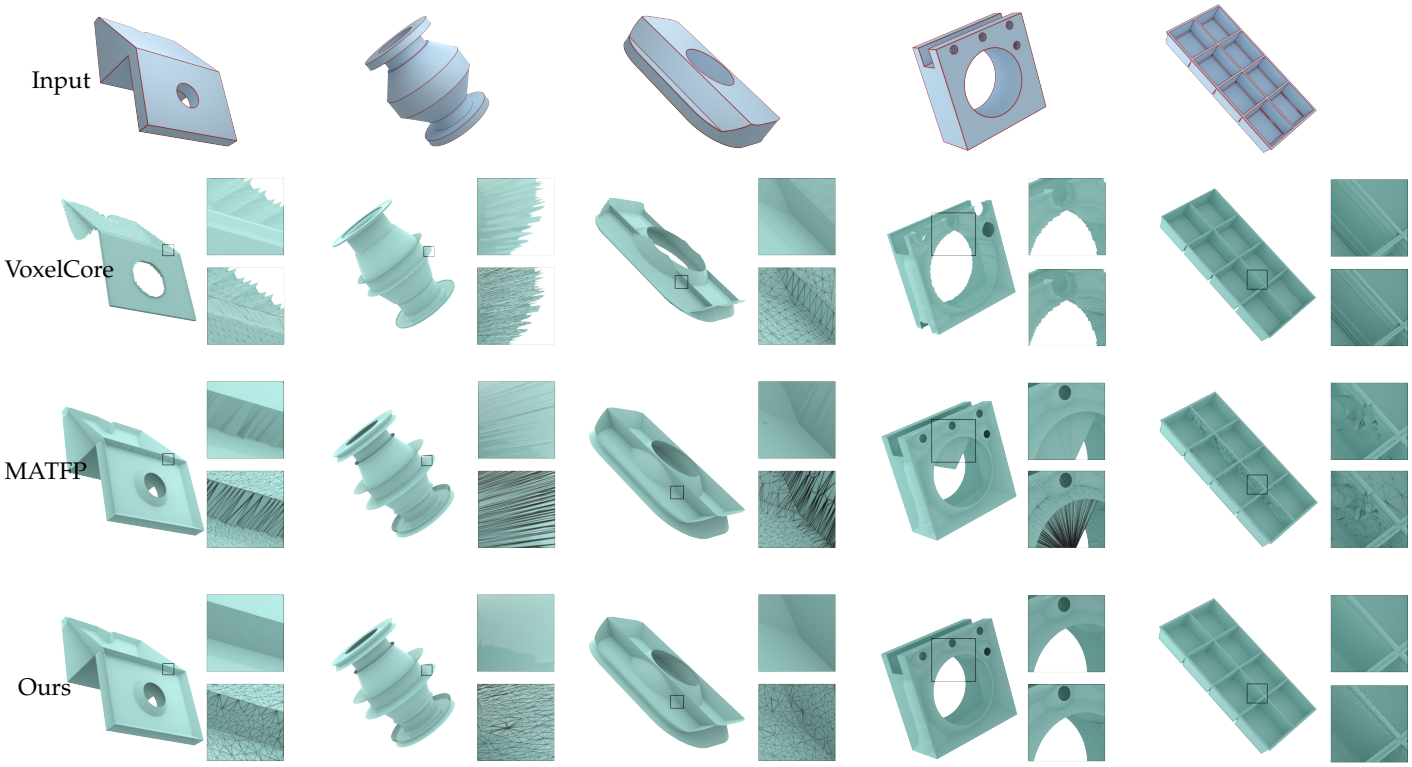



Fig. 6. Visual comparison among Voxel Core [40], MATFP [41], and ours shows that our approach significantly outperforms existing medial axis computation methods on CAD models.

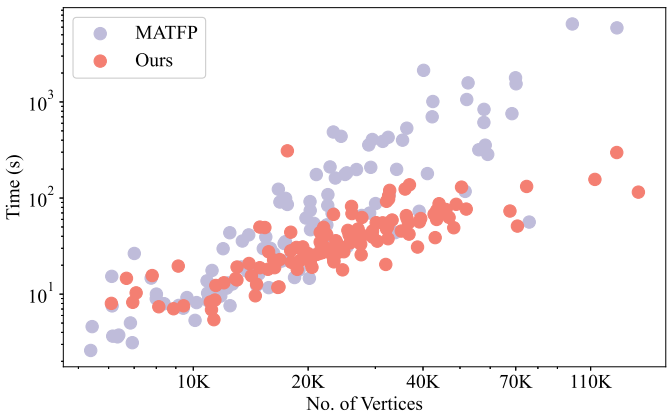



Fig. 7. Statistics about the computational time on 120 models are provided. MATFP and our approach are marked with grey and red dots, respectively.

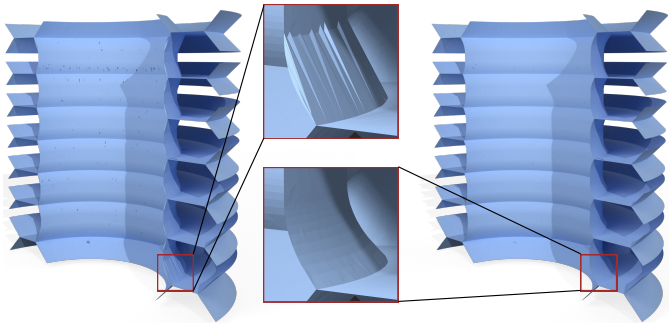


Fig. 8. The sectional view of the medial axis before and after MATFP's thinning process. For this model, the thinning step takes 6,388 seconds, whereas our method takes only 72 seconds to achieve comparable results, making it faster than MATFP by two orders of magnitude.

parallelization. Timing statistics (in seconds) for VoxelCore, MATFP, and our method are comprehensively presented in Table 1. It is noteworthy that the thinning process in MATFP can be extremely time-consuming, in some cases taking almost half an hour, which occurs for the fifth model shown in Fig. 6. Additionally, Figure 7 illustrates the computation time statistics for the medial axis across 120 randomly selected models of varying scales, comparing the performance between MATFP and our method, with the x-axis representing the number of vertices in the medial axis results. In some cases, MATFP's thinning process also requires excessively long overhead. This is because the preliminary medial axis computed using the MATFP method may not consist of a collection of two-dimensional sheets, meaning it does not satisfy the condition of being thin without any solids. Therefore, a thinning step was employed for post-processing to address this. Depending on the model, the thinning step can sometimes be time-consuming. As shown in Figure 8, the medial axis cross-section results before and after the thinning process are presented. For this particular model, the post-processing step took 6388s. Furthermore, by randomly selecting 10 models and adjusting parameters to achieve different resolutions for the computed medial axis, Figure 9 illustrates the relationship between the time cost and the number of vertices in the medial axis, comparing our method with MATFP. It is evident that our method demonstrates increasing runtime performance advantages as the specified accuracy increases.

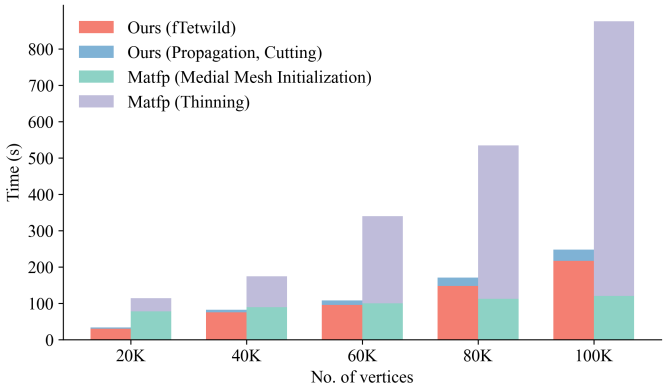



Fig. 9. The runtime performance statistics of our algorithm indicate that the timing cost incurred in the core steps scales linearly with the resolution of tetrahedralization.

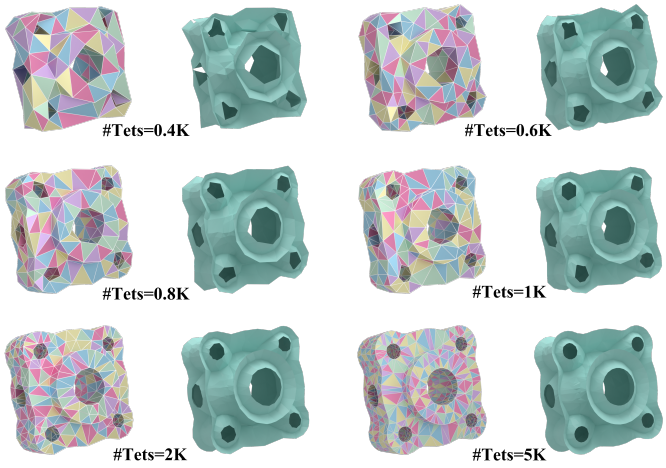



Fig. 10. We discretize the interior of the input model into different numbers of tetrahedral elements to observe the influence of tetrahedral size. We annotated the number of tetrahedra (Tets) in the figure.

5.2.0.4 Results under different tetrahedral resolutions: Given our assumption that the distance field within each tetrahedron changes linearly, and considering our practice of performing incremental cutting for each tetrahedron during computation, the size of the tetrahedra plays a crucial role in influencing the outcomes. In Fig. 10, we employed the fTetwild method on the model at varying tetrahedralization resolutions and showcased the computational findings. It is evident that with an increase in the density of tetrahedralization, the accuracy of the computed results also improves. As demonstrated in Fig. 10, our methodology is capable of achieving precise results with as few as 2K tetrahedral elements.

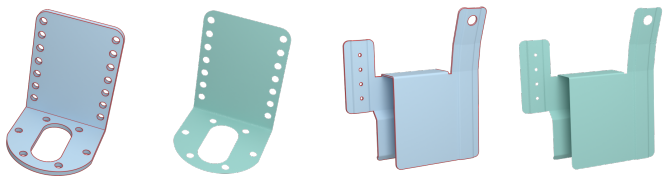


Fig. 11. Even if the input models are as thin as sheet metal, our algorithm can still generate a faithful medial-axis surface. It's worth noting that, for sheet metals, the side faces are excluded before computing the medial axis.

5.2.0.5 Sheet metal models: Our method can be easily extended to compute the medial-axis surface of sheet metal models. The main challenge with sheet metal models is their thinness, which poses a significant hurdle for sampling-based approaches that require a large number of points. However, our approach considers each surface patch as a whole, eliminating the need for a sampling step. As the side faces typically have minimal impact on the structural behavior during the simulation, they are excluded from the generator list before computing the medial axis. Fig. 11 illustrates two typical examples, showcasing the effectiveness of our method.

5.2.0.6 Organic meshes.: To validate the effectiveness of the proposed algorithm, we conducted tests on organic meshes. The key difference between organic meshes and CAD models lies in the lack of clear segmentation criteria. We observed that one step of the Variational Shape Approximation method [51] involves partitioning the model's triangles into several categories. Therefore, we set the total number of categories to 200 and used these categories as surface patches for the algorithm's input. For organic models, neighboring surface patches typically exhibit smooth transitions, meaning that the Voronoi diagram between them is generally not part of the true medial axis. Therefore, we removed these unnecessary tiny structures during the medial axis computation. As Figure 13 shows, our algorithm still produces fair medial-axis results, demonstrating its great potential.

5.2.0.7 Comparison with generalized Voronoi diagram algorithms.: There are several methods for computing generalized Voronoi diagrams that take an open or closed surface as a generator. A seminal work in this field was proposed by Edwards et al. [52]. It involves initially constructing an octree, storing essential information at nodes, and subsequently obtaining approximate Voronoi diagram results through reconstruction. However, these algorithms face two primary issues when applied to the medial axis computation of CAD models. On one side, its approximation strategy is not globally accurate, resulting in a lack of smooth transitions at the intersections of different cubes. On the other side, the algorithm's configuration of the solution space is not flexible, making it challenging to confine the results within the model's interior. Figure 14 demonstrates that the approach by Edwards et al. [52] results in severely bumpy surfaces due to the imprecision of its approximation strategy.

5.2.0.8 Robustness.: We randomly selected 500 models from the ABC dataset (all models free of self-intersections) for testing in two modes. This collection encompasses a variety of models including thin plates, slender tubes, and high-genus structures, among others. When computing the medial axis using the tolerance technique, we encountered 8 instances of numerical errors, resulting in an overall error rate of $1.6\%$. However, by adopting a robust implementation (with exact numerical types discussed in Section 4.2), we observed no numerical errors. We provide additional results in Fig. 12, showcasing a gallery of medial axis surfaces.

5.2.0.9 Impact of numeric types and threads on runtime: In the incremental hyperplane cutting process, the number of threads and the choice of exact/approximate

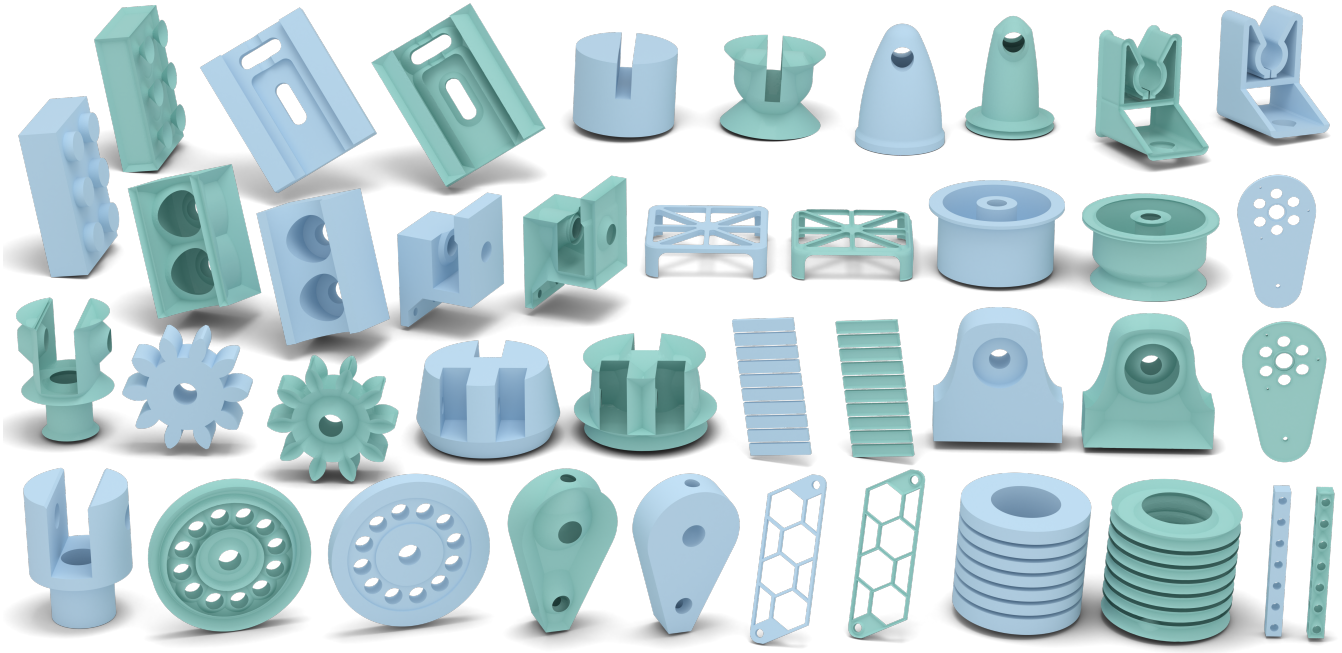


Fig. 12. More medial-axis results computed by our algorithm.

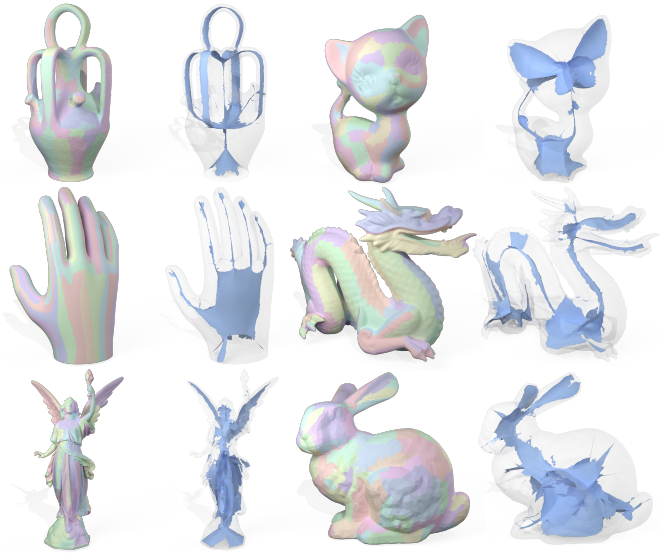


Fig. 13. Medial axis computation results on organic meshes. Different surface patches are represented using distinct colors.

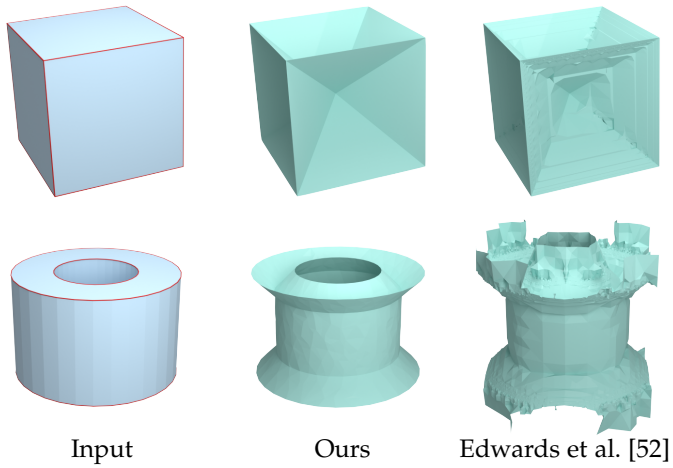



Fig. 14. Comparison with Edwards et al. [52].

incremental cutting strategies are closely related to the algorithm's runtime. Therefore, we use the model in Figure 8 as an example to study the algorithm's running speed under different thread counts and incremental cutting strategy settings. The results are illustrated in Figure 15. It is observed that when the number of threads reaches 32, the speed tends to stabilize, which is also the number of threads chosen in the experiments of this paper. Overall, the runtime of the precise cutting strategy is approximately twice that of the approximate incremental cutting strategy. In addition, we made statistics about the time cost when merely replacing the double data type with an exact type for precise computation, while keeping the number of threads set to 32. The execution time cost in this case is 290.156 seconds. This demonstrates the speed advantage of the precise cutting strategy proposed in this paper.

### 5.3 Offset Surfaces

5.3.0.1 Problem statement.: Offsetting is a fundamental operation in computational geometry and computer graphics. It approximates the shape of a 2D curve or 3D surface by generating a parallel curve or surface at a fixed distance from the original shape. This operation finds various applications, such as creating smooth boundaries around a shape [53], performing collision detection [54], and generating tool paths for CNC machining [55].

5.3.0.2 Primary challenge.: For a given triangle mesh, achieving accurate computation of offsets involves dilating each triangle and explicitly resolving the introduced self-intersections [56, 57, 58]. However, these approaches

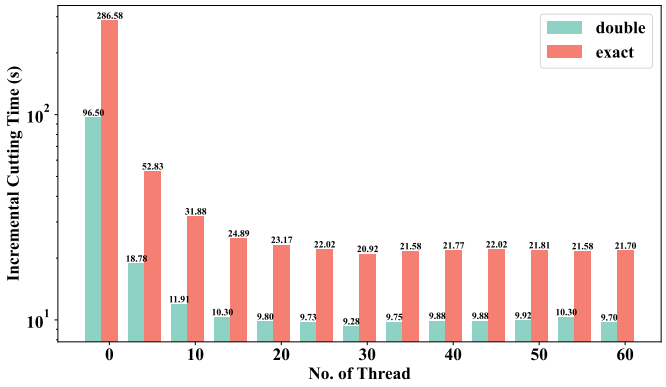



Fig. 15. Runtime statistics of the exact and approximate incremental cutting strategies under different thread counts for the model in Figure 8.

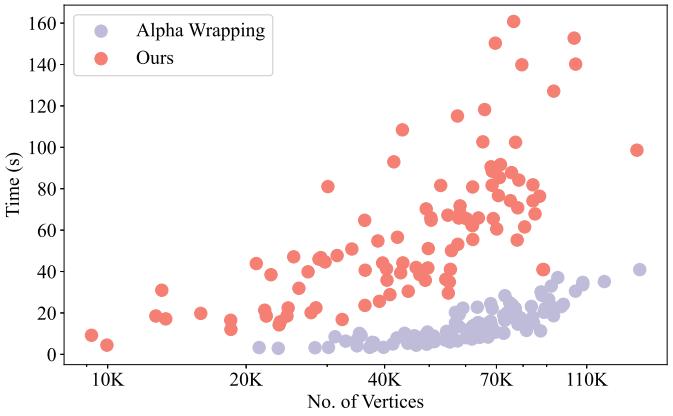



Fig. 17. Computation time of the offset surface on 100 models of Alpha Wrapping and Ours.

necessitate a tedious post-processing step for handling self-intersections. Alternatively, some methods entail resampling the offset surface by computing the isosurface of the signed-distance function derived from the original surface and then enforcing a deviation [59]. CGAL includes an offsetting function called Alpha Wrapping [53]. The resulting output is achieved by greedily refining and carving a 3D Delaunay triangulation on an offset surface of the input, while carving with empty balls of radius alpha. Nevertheless, these methods may lead to inaccurate offset surfaces with missing sharp features. The approach proposed by [60] preserves feature information, but its computational process is extremely time-consuming, making it impractical for some applications.

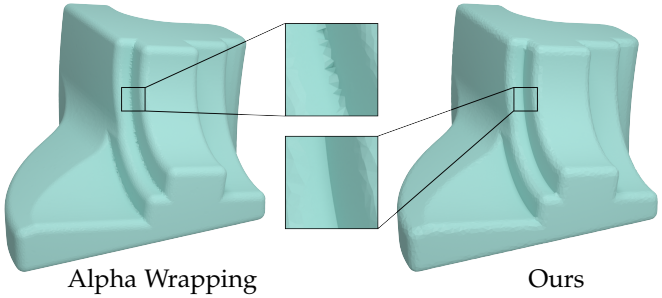



Fig. 16. A visual comparison between Alpha Wrapping [53] (left) and our method (right) is shown, with close-up windows highlighting the differences. It is evident that our algorithm can preserve sharp feature lines, whereas Alpha Wrapping cannot.

5.3.0.3 Our solution.: Suppose we are computing an offset surface at a user-specified distance of $d$. Mathematically, we aim to find the the surface where $|\mathbf{D}| = d$. Considering that a CAD model can be easily decomposed into a collection of simple surface patches, we assume that each surface-patch generator $s_i$ contributes a linear distance field within a tetrahedron $t = v_1v_2v_3v_4$. Therefore, unlike medial-axis extraction, $s_i$ can survive in the tetrahedron $t = v_1v_2v_3v_4$ if:

1) $s_i$ is not defeated by other generators in $t$;
2) $\min_j\{\mathbf{D}(s_i, v_j)\} \leq d \leq \max_j\{\mathbf{D}(s_i, v_j)\}$.

At the end of distance over-propagation, if a tetrahedron contains an empty generator list, it does not contribute to the offsetting surface.

TABLE 2
Time statistics (in seconds) of offset calculation for the models shown in Fig. 18.

| | | model 1 | model 2 | model 3 | model 4 | model 5 | model 6 |
|---|---|---|---|---|---|---|---|
| Alpha Wrapping | | 24.662 | 12.855 | 7.980 | 11.213 | 6.190 | 11.096 |
| Ours | fTetwild | 67.613 | 49.344 | 25.375 | 66.829 | 42.516 | 36.547 |
| | Propagation, Cutting | 16.852 | 7.797 | 5.594 | 12.469 | 7.125 | 9.203 |

Our fundamental insight is that the offset surface at distance $d$ can be derived by identifying where the distance field $\mathbf{D}$ matches $2d - \mathbf{D}$. It is much like the situation of medial axis computations. For a given tetrahedron $t$, consider its associated surface patch set as $S = \{s_i\}_{i=1}^m$. We then define a corresponding virtual patch set $S^{'} = \{s_i\}_{i=1}^m$. The distances from $s_i^{'}$ to the vertices $v_i$ are set as $2d - \mathbf{D}(s_i, v_i)$. The distance fields within the set $S$ are used to incrementally cut and preserve the lower envelope within a four-dimensional framework. Concurrently, the distance fields within set $S^{'}$ perform incremental cuts to maintain the upper envelope of the same four-dimensional structure. The intersection of these two envelopes, resulting from the iterative process, indicating the competition between $S$ and $S^{'}$. Through a procedure akin to computing a medial axis, we are thus able to delineate the offset surface, comprising both an inward and an outward layer. Separating these layers based on their connectivity subsequently becomes a straightforward task.

5.3.0.4 Visual comparison.: In Fig. 16, a visual comparison between the Alpha Wrapping algorithm and our method is presented, with differences highlighted in close-up views. The comparison demonstrates that our offsetting algorithm can generate distinctive feature lines, while Alpha Wrapping cannot. Additionally, it is important to note that Alpha Wrapping does not support inward offsetting. More examples for comparison are provided in Fig. 18.

5.3.0.5 Run-time performance.: In our experiments, we chose an offset distance equivalent to 2% of the diagonal length of the model's bounding box. It is evident that the total time required is significantly influenced by the resolution of tetrahedralization; a higher resolution leads to improved accuracy. For the purpose of tetrahedralization, we leverage the space created by offsetting the model's bounding box

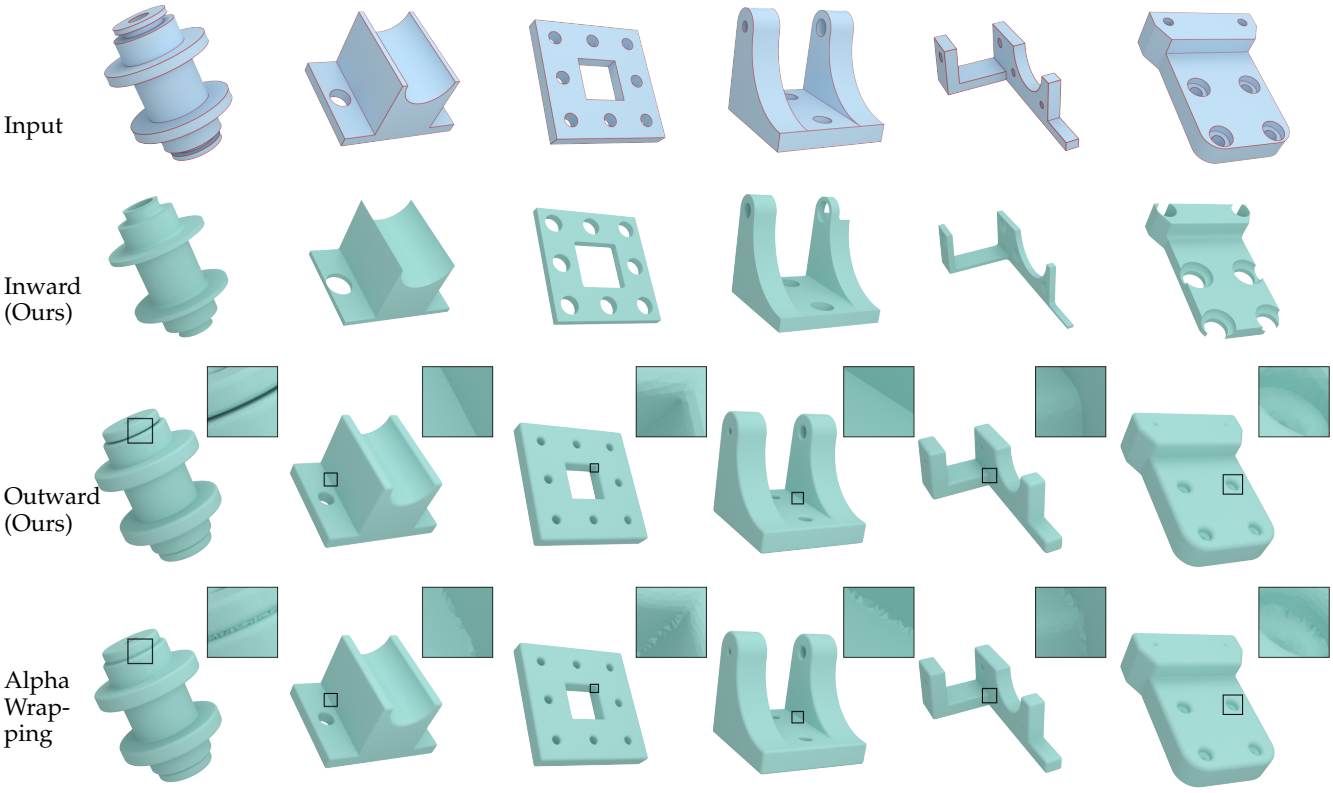



Fig. 18. Visual comparison between Alpha Wrapping [53] and ours for computing offset surfaces.

and embed the input model within the tetrahedralization outcome. Specifically, we set the target edge length for fTetwild to 0.015 and adjust the alpha parameter for the Alpha Wrapping algorithm to 256. The details regarding runtime performance are provided in Table 2. Additionally, we made statistics about the runtime of the Alpha Wrapping method and ours on 100 randomly selected models, as shown in Figure 17.

5.3.0.6 Results.: In Fig. 5, variant Voronoi diagrams are presented, utilizing four identical Koala models as generators. It should be noted that each variant provides a tailored solution for various applications. This example highlights the high flexibility and adaptability offered by our algorithm.

## 6 LIMITATIONS AND FUTURE WORK

Our algorithm, in its current form, exhibits at least three disadvantages. Firstly, it assumes that the tetrahedron-range distance field for a single generator undergoes linear change, which may not hold if the tetrahedral size is large. Secondly, the run-time performance diminishes if the tetrahedralization is too dense. Finally, while our algorithm can produce faithful medial axis and offset results for CAD models, extending it to organic shapes is non-trivial due to the equally challenging problem of decomposing the surface of a free-form shape.

In the future, we plan to address these issues from two perspectives. Firstly, we will introduce adaptive tetrahedralization as an initialization step to balance computational accuracy and run-time performance. Secondly, we will develop improved strategies for decomposing the surface of organic shapes.

## 7 CONCLUSION

In this paper, we extend SurfaceVoronoi to 3D and introduce an accurate algorithm for computing Voronoi diagrams of surface patches. The key observation is that the lower envelope of the 4D roof-like structure encodes the Voronoi diagram structure restricted to a tetrahedron. In implementation, we develop a numerically stable lifting technique for 4D hyperplane cutting operations. The new algorithm operates independently of pre-existing Voronoi numerical packages. We demonstrate its effectiveness in extracting high-quality medial axis transforms (MAT).

Due to its flexibility and scalability, our algorithm can also be adapted to compute the offset surface and various variants of the Voronoi diagram. We provide extensive experimental results to validate its effectiveness and usefulness.

### ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions. This work is supported by National Key R&D Program of China (2022YFB3303200), National Natural Science Foundation of China (62272277, U23A20312, 62072284, 62172415), the NSF of Shandong Province (ZR2020MF036), the Strategic Priority Research Program of the Chinese Academy of Sciences (XDB0640000 and XDB0640200), and the Beijing Natural Science Foundation (Z240002).

## REFERENCES


[1] Brian Mirtich. V-clip: Fast and robust polyhedral collision detection. *ACM Transactions On Graphics (TOG)*, 17(3):177–208, 1998.

[2] Leduin José Cuenca Macas and Israel Pineda. Collision avoidance simulation using voronoi diagrams in a centralized system of holonomic multi-agents. In *Information and Communication Technologies: 10th Ecuadorian Conference, TICEC 2022*, pages 18–31, 2022.

[3] Dobrina Boltcheva and Bruno Lévy. Surface reconstruction by computing restricted voronoi cells in parallel. *Computer-Aided Design*, 90:123–134, 2017.

[4] S Garrido and L Moreno. Mobile robot path planning using voronoi diagram and fast marching. In *Robotics, automation, and control in industrial and service settings*, pages 92–108. IGI Global, 2015.

[5] Thomas Lindemeier, Jens Metzner, Lena Pollak, and Oliver Deussen. Hardware-based non-photorealistic rendering using a painting robot. In *Computer Graphics Forum(CGF)*, volume 34, pages 311–323, 2015.

[6] Shi-Qing Xin, Shuang-Min Chen, Ying He, Guo-Jin Wang, Xianfeng Gu, and Hong Qin. Isotropic mesh simplification by evolving the geodesic delaunay triangulation. In *Voronoi Diagrams in Science and Engineering (ISVD ), 2011 Eighth International Symposium on*, pages 39–47, 2011.

[7] Dong Ming Yan, Wenping Wang, Bruno Lévy, and Yang Liu. Efficient computation of clipped voronoi diagram for mesh generation. *Computer-Aided Design*, 45:843–852, 2013.

[8] Shen Ying, Guang Xu, Chengpeng Li, and Zhengyuan Mao. Point cluster analysis using a 3d voronoi diagram with applications in point cloud segmentation. *ISPRS International Journal of Geo-Information*, 4(3):1480–1499, 2015.

[9] Shiqing Xin, Pengfei Wang, Rui Xu, Dongming Yan, Shuangmin Chen, Wenping Wang, Caiming Zhang, and Changhe Tu. Surfacevoronoi: Efficiently computing voronoi diagrams over mesh surfaces with arbitrary distance solvers. *ACM Transactions on Graphics (TOG)*, 41(6):1–12, 2022.

[10] Xi Zhao, He Wang, and Taku Komura. Indexing 3d scenes using the interaction bisector surface. *ACM Transactions on Graphics (TOG)*, 33(3):1–14, 2014.

[11] Sheng-Kai Huang, Wen-June Wang, and Chung-Hsun Sun. A path planning strategy for multi-robot moving with path-priority order based on a generalized voronoi diagram. *Applied Sciences*, 11(20):9650, 2021.

[12] Chen Zong, Jiacheng Xu, Jiantao Song, Shiqing Xin, Shuangmin Chen, Wenping Wang, and Changhe Tu. P2m: A fast solver for querying distance from point to mesh surface. *ACM Transactions on Graphics (TOG)*, pages 1–11, 2023.

[13] Rui Xu, Zhiyang Dou, Ningna Wang, Shiqing Xin, Shuangmin Chen, Mingyan Jiang, Xiaohu Guo, Wenping Wang, and Changhe Tu. Globally consistent normal orientation for point clouds by regularizing the winding-number field. *ACM Transactions on Graphics (TOG)*, pages 1–14, 2023.

[14] Rui Xu, Zixiong Wang, Zhiyang Dou, Chen Zong, Shiqing Xin, Mingyan Jiang, Tao Ju, and Changhe Tu. Rfeps: Reconstructing feature-line equipped polygonal surface. *ACM Transactions on Graphics (TOG)*, (6):1–15, 2022.

[15] Pengfei Wang, Zixiong Wang, Shiqing Xin, Xifeng Gao, Wenping Wang, and Changhe Tu. Restricted delaunay triangulation for explicit surface reconstruction. *ACM Transactions on Graphics*, 41(5):1–20, 2022.

[16] Mohammed Laraqui, Abderrahim Saaidi, Ali Mouhib, and Mustapha Abarkan. Images matching using voronoï regions propagation. *3D Research*, 6:1–16, 2015.

[17] Laura Bianca Bilius and Ştefan Gheorghe Pentiuc. Efficient unsupervised classification of hyperspectral images using voronoi diagrams and strong patterns. *Sensors*, 20(19):56–84, 2020.

[18] Michael Ian Shamos and Dan Hoey. Closest-point problems. In *16th Annual Symposium on Foundations of Computer Science*, pages 151–162, 1975.

[19] Peter J Green and Robin Sibson. Computing Dirichlet tessellations in the plane. *The computer journal*, 21(2):168–173, 1978.

[20] Steven Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2(1-4):153, 1987.

[21] Steven Fortune. Voronoi diagrams and Delaunay triangulations. In *Computing in Euclidean geometry*, pages 225–265. World Scientific, 1995.

[22] Guoqing Wu, Hongyun Tian, Guo Lu, and Wei Wang. Parvoro++: A scalable parallel algorithm for constructing 3d voronoi tessellations based on kd-tree decomposition. *Parallel Computing*, 115:102–995, 2023.

[23] Jiechen Wang, Can Cui, Yikang Rui, Liang Cheng, Yingxia Pu, Wenzhou Wu, and Zhenyu Yuan. A parallel algorithm for constructing voronoi diagrams based on point-set adaptive grouping. *Concurrency and Computation: Practice and Experience*, 26(2):434–446, 2014.

[24] Daniel Reem. The projector algorithm: a simple parallel algorithm for computing voronoi diagrams and delaunay graphs. *arXiv preprint arXiv:1212.1095*, 2012.

[25] Seunghwan Choi, Joonghyun Ryu, Mokwon Lee, Jehyun Cha, Hyunwoo Kim, Chanyoung Song, and Deok-Soo Kim. Support-free hollowing with spheroids and efficient 3d printing utilizing circular printing motions based on voronoi diagrams. *Additive Manufacturing*, 35:101254, 2020.

[26] Nissim Maruani, Roman Klokov, Maks Ovsjanikov, Pierre Alliez, and Mathieu Desbrun. Voromesh: Learning watertight surface meshes with voronoi diagrams. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14565–14574, 2023.

[27] Menelaos I Karavelas. A robust and efficient implementation for the segment voronoi diagram. In *International Symposium on Voronoi diagrams in science and engineering*, volume 2004, pages 51–62, 2004.

[28] Martin Held and Stefan Huber. Topology-oriented incremental computation of voronoi diagrams of circular arcs and straight-line segments. *Computer-Aided Design*, 41:327–338, 2009.

[29] Lin Lu, Bruno Lévy, and Wenping Wang. Centroidal voronoi tessellation of line segments and graphs. *Computer Graphics Forum*, 31(2pt4):775–784, 2012.

[30] Chunxu Xu, Yong-Jin Liu, Qian Sun, Jinyan Li, and

Ying He. Polyline-sourced geodesic voronoi diagrams on triangle meshes. In *Computer Graphics Forum(CGF)*, volume 33, pages 161–170, 2014.

[31] Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri. The power crust. In *Proceedings of the sixth ACM symposium on Solid modeling and applications*, pages 249–266, 2001.

[32] Wenlong Meng, Pengbo Bo, Xiaodong Zhang, Jixiang Hong, Shiqing Xin, and Changhe Tu. An efficient algorithm for approximate voronoi diagram construction on triangulated surfaces. *Computational Visual Media*, 9(3):443–459, 2023.

[33] Chen Zong, Pengfei Wang, Dong-Ming Yan, Shuangmin Chen, Shiqing Xin, Changhe Tu, and Qiang Hu. Parallel post-processing of restricted voronoi diagram on thin sheet models. *Computer-Aided Design*, 159:103–511, 2023.

[34] Dong-Ming Yan, Bruno Lévy, Yang Liu, Feng Sun, and Wenping Wang. Isotropic remeshing with fast and exact computation of restricted voronoi diagram. In *Computer Graphics Forum(CGF)*, volume 28, pages 1445–1454, 2009.

[35] Yipeng Qin, Xiaoguang Han, Hongchuan Yu, Yizhou Yu, and Jianjun Zhang. Fast and exact discrete geodesic computation based on triangle-oriented wavefront propagation. *ACM Trans. Graph.*, 35(4), July 2016.

[36] Philipp Herholz, Felix Haase, and Marc Alexa. Diffusion diagrams: Voronoi cells and centroids from diffusion. *Computer Graphics Forum*, 36(2):163–175, 2017.

[37] Alexander G. Belyaev and Pierre-Alain Fayolle. On variational and pde-based distance function approximations. *Computer Graphics Forum*, 34(8):104–118, 2015.

[38] Gareth Bradshaw and Carol O'Sullivan. Adaptive medial-axis approximation for sphere-tree construction. *ACM Transactions on Graphics (TOG)*, 23(1):1–26, 2004.

[39] Feng Sun, Yi-King Choi, Yizhou Yu, and Wenping Wang. Medial meshes: A compact and accurate medial shape representation. *IEEE Transactions on Visualization and Computer Graphics*, 22:1278–1290, 2014.

[40] Yajie Yan, David Letscher, and Tao Ju. Voxel cores: Efficient, robust, and provably good approximation of 3d medial axes. *ACM Transactions on Graphics (TOG)*, 37(4):1–13, 2018.

[41] Ningna Wang, Bin Wang, Wenping Wang, and Xiaohu Guo. Computing medial axis transform with feature preservation via restricted power diagram. *ACM Transactions on Graphics (TOG)*, 41(6):1–18, 2022.

[42] Yixin Hu, Teseo Schneider, Bolun Wang, Denis Zorin, and Daniele Panozzo. Fast tetrahedral meshing in the wild. *ACM Trans. Graph.*, 39(4), July 2020.

[43] Michael Hemmer, Susan Hert, Sylvain Pion, and Stefan Schirra. Number types. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.5.2 edition, 2023.

[44] Xingyi Du, Qingnan Zhou, Nathan Carr, and Tao Ju. Robust computation of implicit surface networks for piecewise linear functions. *ACM Transactions on Graphics (TOG)*, 41(4):1–16, 2022.

[45] Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. Abc: A big cad model dataset for geometric deep learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[46] Hiroshi Imai, Masao Iri, and Kazuo Murota. Voronoi diagram in the laguerre geometry and its applications. *SIAM Journal on Computing*, 14:93–105, 1985.

[47] D. T. Lee and Robert L. Scot Drysdale. Generalization of voronoi diagrams in the plane. *SIAM Journal on Computing*, 10:73–87, 1981.

[48] Peter F Ash and Ethan D Bolker. Generalized dirichlet tessellations. *Geometriae dedicata*, 20(2):209–243, 1986.

[49] Pan Li, Bin Wang, Feng Sun, Xiaohu Guo, Caiming Zhang, and Wenping Wang. Q-mat: Computing medial axis transform by quadratic error minimization. *ACM Transactions on Graphics (TOG)*, 35(1):1–16, 2015.

[50] Truc Le and Ye Duan. A primitive-based 3d segmentation algorithm for mechanical cad models. *Computer Aided Geometric Design*, 52:231–246, 2017.

[51] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. Variational shape approximation. *ACM Trans. Graph.*, 23(3):905–914, August 2004.

[52] John Edwards, Eric Daniel, Valerio Pascucci, and Chandrajit Bajaj. Approximating the generalized voronoi diagram of closely spaced objects. *Computer Graphics Forum*, 34(2):299–309, 2015.

[53] Cédric Portaneri, Mael Rouxel-Labbé, Michael Hemmer, David Cohen-Steiner, and Pierre Alliez. Alpha wrapping with an offset. *ACM Transactions on Graphics (TOG)*, 41(4):1–22, 2022.

[54] Matthias Teschner, Stefan Kimmerle, Bruno Heidelberger, Gabriel Zachmann, Laks Raghupathi, Arnulph Fuhrmann, M-P Cani, François Faure, Nadia Magnenat-Thalmann, Wolfgang Strasser, et al. Collision detection for deformable objects. In *Computer Graphics Forum(CGF)*, volume 24, pages 61–81, 2005.

[55] Su-jin Kim and Min-Yang Yang. Incomplete mesh offset for nc machining. *Journal of Materials Processing Technology*, 194:110–120, 2007.

[56] Wonhyung Jung, Hayong Shin, and Byoung K Choi. Self-intersection removal in triangular mesh offsetting. *Computer-Aided Design and Applications*, 1(1-4):477–484, 2004.

[57] Marcel Campen and Leif Kobbelt. Exact and robust (self-) intersections for polygonal meshes. In *Computer Graphics Forum(CGF)*, volume 29, pages 397–406, 2010.

[58] Silvia Sellan, Jacob Kesten, Ang Yan Sheng, and Alec Jacobson. Opening and closing surfaces. *ACM Transactions on Graphics (TOG)*, 39(6):1–13, 2020.

[59] Martin Peternell and Tibor Steiner. Minkowski sum boundary surfaces of 3d-objects. *Graphical Models*, 69(3-4):180–190, 2007.

[60] D. Zint, N. Maruani, M. Rouxel-Labb, and P. Alliez. Feature-preserving offset mesh generation from topology-adapted octrees. *Computer Graphics Forum*, 42(5):e14906, 2023.

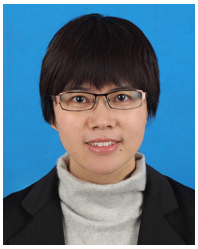
**Pengfei Wang** is currently working toward the PhD degree in computer science with Shandong University. His research interests include computational geometry, computer graphics, and computer-aided design.

**Jiantao Song** is currently pursuing a Ph.D. in computer science at Shandong University His research interests are computer graphics, computational geometry, point cloud reconstruction and geometric modeling.

**Lei Wang** is a master's candidate in Computer Science at Shandong University. His research interests include computer graphics and geometric processing, with a focus on shape analysis, representation, and geometric solutions for practical applications.

**Shiqing Xin** is currently a professor in the School of Computer Science at Shandong University. He obtained his Ph.D. from Zhejiang University (China) in 2009. His research interests encompass a range of geometry processing algorithms. Over the past decade, he has authored and co-authored over 100 papers published in renowned journals and conferences, including IEEE TVCG and ACM TOG, among others. He won the Best Paper Award at SIGGRAPH 2023.

**Dongming Yan** (Member, IEEE) is a professor at the State Key Laboratory of Multimodal Artificial Intelligence Systems of the Institute of Automation, Chinese Academy of Sciences. He received his Ph.D. from Hong Kong University in 2010 and his Master's and Bachelor's degrees from Tsinghua University in 2005 and 2002, respectively. His research interests include computer graphics, computer vision, geometric processing and pattern recognition.



**Shuangmin Chen** received the PhD degree from Ningbo University in 2018. She is currently an associate professor with the School of Information and Technology, Qingdao University of Science and Technology, China. She has authored or coauthored more than 40 research papers in famous journals and conferences. Her research interests include computer graphics and computational geometry

**Changhe Tu** received the BSc, MEng, and PhD degrees from Shandong University, China, in 1990, 1993, and 2003, respectively. He is currently a professor with the School of Computer Science and Technology, Shandong University, China. He currently leads the CG-VIS Group, Shandong University. He has authored or coauthored more than 100 papers in international journals and conferences. His research interests include computer graphics, 3D vision, and computer-aided geometric design.

**Wenping Wang** (Fellow, IEEE) received the PhD degree in computer science from the University of Alberta, in 1992. He is currently a chair professor of Computer Science with the University of Hong Kong. His research interests include computer graphics, computer visualization, computer vision, robotics, medical image processing, and geometric computing. He is associate editor of several premium journals, including the Computer Aided Geometric Design (CAGD), Computer Graphics Forum (CGF), IEEE Transactions on Computers, and IEEE Computer Graphics and Applications, and has chaired more than 20 international conferences, including Pacific Graphics 2012, ACM Symposium on Physical and Solid Modeling (SPM) 2013, SIGGRAPH Asia 2013, and Geometry Submit 2019. He received the John Gregory Memorial Award for his contributions in geometric modeling.