PCO: Precision-Controllable Offset Surfaces with Sharp Features

LEI WANG, Shandong University, China
XUDONG WANG, Shandong University, China
PENGFEI WANG, Shandong University, China
SHUANGMIN CHEN*, Qingdao University of Science and Technology, China
SHIQING XIN, Shandong University, China
JIONG GUO, Shandong University, China
WENPING WANG, Texas A&M University, USA
CHANGHE TU, Shandong University, China

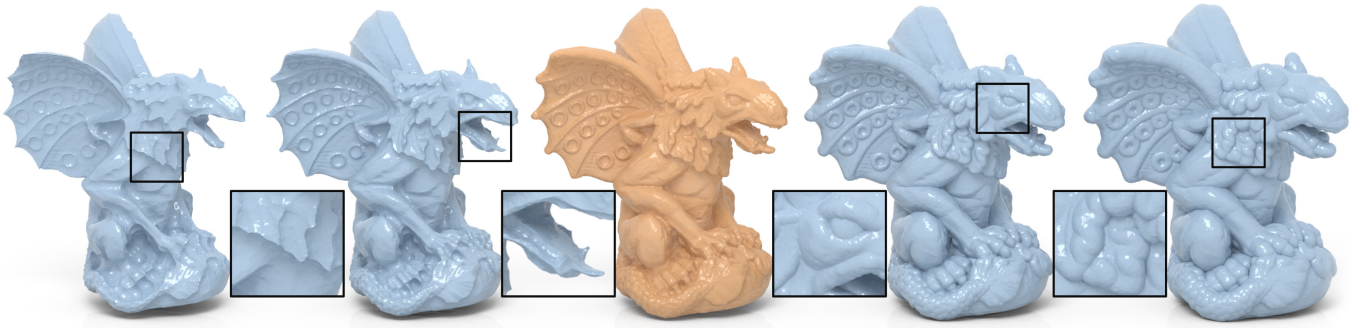


Fig. 1. This paper is centered around the generation of offset surfaces. The figure demonstrates how offset surfaces vary as the distance increases. A key advantage of our algorithm is its precise preservation of sharp features, eliminating the need for supplementary iso-surface extraction methods, which frequently struggle to accurately maintain these sharp features. See the highlighted windows.

Surface offsetting is a crucial operation in digital geometry processing and computer-aided design, where an offset is defined as an iso-value surface of the distance field. A challenge emerges as even smooth surfaces can exhibit sharp features in their offsets due to the non-differentiable characteristics of the underlying distance field. Prevailing approaches to the offsetting problem involve approximating the distance field and then extracting the iso-surface. However, even with dual contouring (DC), there is a risk of degrading sharp feature points/lines due to the inaccurate discretization of the distance field. This issue is exacerbated when the input is a piecewise-linear triangle mesh.

This study is inspired by the observation that a triangle-based distance field, unlike the complex distance field rooted at the entire surface, remains smooth across the entire 3D space except at the triangle itself. With a polygonal surface comprising n triangles, the final distance field for accommodating the offset surface is determined by minimizing these n triangle-based distance fields. In implementation, our approach starts by tetrahedralizing the space around the offset surface, enabling a tetrahedron-wise linear approximation for each triangle-based distance field. The final offset surface within a tetrahedral range can be traced by slicing the tetrahedron with planes. As illustrated in the teaser figure, a key advantage of our algorithm is its ability to precisely preserve sharp features. Furthermore, this paper addresses the problem of simplifying the offset surface's complexity while preserving sharp features, formulating it as a maximal-clique problem.

CCS Concepts: • **Computing methodologies → Shape modeling**.

Additional Key Words and Phrases: digital geometry processing, polygonal surface, offset, sharp feature, maximal clique

ACM Reference Format:
Lei Wang, Xudong Wang, Pengfei Wang, Shuangmin Chen, Shiqing Xin, Jiong Guo, Wenping Wang, and Changhe Tu. 2024. PCO: Precision-Controllable Offset Surfaces with Sharp Features. *ACM Trans. Graph.* 1, 1 (November 2024), 16 pages. https://doi.org/10.1145/3687920

1 INTRODUCTION

The creation of offset surfaces is a fundamental operation in geometry processing and computer-aided design, encompassing a wide range of applications such as shelling, collision detection, and path planning [Pham 1992; Teschner et al. 2005; Williams and Rossignac 2005]. In this paper, we concentrate on computing an accurate offset surface from a polygonal surface, with a specific focus on preserving sharp features.

*Corresponding author: Shuangmin Chen.

Authors' addresses: Lei Wang, Shandong University, Qingdao, Shandong, China, leiw1006@gmail.com; Xudong Wang, Shandong University, Qingdao, Shandong, China, sduwxd0319@gmail.com; Pengfei Wang, Shandong University, Qingdao, Shandong, China, pengfei1998@foxmail.com; Shuangmin Chen, Qingdao University of Science and Technology, Qingdao, Shandong, China, csmqq@163.com; Shiqing Xin, Shandong University, Qingdao, Shandong, China, xinshiqing@sdu.edu.cn; Jiong Guo, Shandong University, Qingdao, Shandong, China, xhjia@amss.ac.cn; Wenping Wang, Texas A&M University, Texas, USA, wenping@tamu.edu; Changhe Tu, Shandong University, Qingdao, Shandong, China, chtu@sdu.edu.cn.

ACM 0730-0301/2024/11-ART
https://doi.org/10.1145/3687920

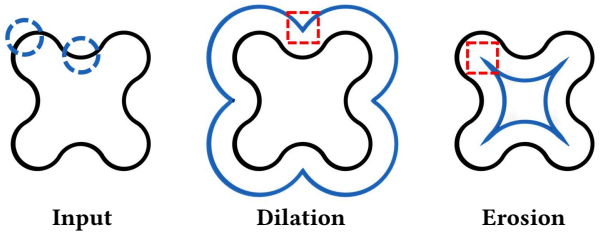


Fig. 2. The offsetting operation either expands or contracts the input shape by a specified distance. Even smooth surfaces can develop sharp features in their offsets due to the non-differentiable characteristics of the underlying distance field.

The offset at a distance δ can be considered a specific case of the Minkowski operation, widely explored in mathematical morphology. This process computes the offset surface by allowing a sphere to traverse with its center lying on the surface, as depicted in Fig. 2. It can be observed that even smooth surfaces may exhibit sharp features in their offsets due to the non-differential characteristics of the underlying distance field.

Methods for generating offset surfaces are primarily classified into two categories: explicit and implicit. Explicit methods [Farouki 1985] involve offsetting each geometric primitive separately and then amalgamating them to form a cohesive surface. These approaches require a complex post-processing phase to identify intersections and eliminate extraneous components. In contrast, implicit methods [Shen et al. 2004] focus on the accurate computation of a distance field, followed by the extraction of an iso-surface as the offset. The challenges associated with these methods are twofold. First, even if the distance field is represented by high-resolution volumetric grids, the discontinuities within the distance field may still be degraded to some extent. Second, even with dual contouring (DC) [Ju et al. 2002], the sharp feature points/lines may not be fully recovered due to the inaccurate discretization of the distance field.

The research presented in this paper is driven by an insightful observation: a distance field derived from individual triangles, as opposed to one given by the entire surface, remains smooth throughout the 3D space, with irregularities appearing only at the triangles themselves. This inherent smoothness of a triangle-based distance field enables us to divide the space into small units and apply a linear approximation within each unit. It is important to highlight that this piecewise linear approximation method is effective provided that the units are sufficiently small. Based on the fact that the actual distance field of the entire surface, necessary for creating the offset, can be constructed by minimizing across all the triangle-based distance fields, our approach begins with creating a piecewise approximation for the distance field of each triangle, followed by merging these approximations precisely to form the complete offset.

Our method, referred to as *PCO*, initiates by tetrahedralizing the space surrounding the offset surface, thus facilitating a tetrahedron-wise linear approximation for each triangle-based distance field. Suppose n represents the number of triangles. The actual distance field for approximating the offset surface is given by minimizing the n triangle-based distance fields. Within every tetrahedron, achieving

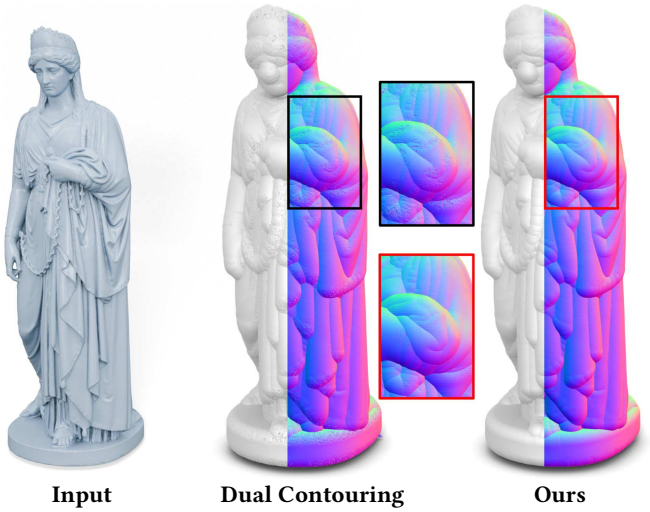


Fig. 3. Prevailing approaches to the offsetting problem involve approximating the distance field and then extracting the iso-surface. However, even with dual contouring (at a grid resolution of 1024), there is a risk of degrading sharp feature points and lines due to the inaccurate discretization of the distance field. In contrast, our method (with a grid resolution of 512) demonstrates advantages in both accuracy and feature preservation. Note that the offset distance is set to 2%, and the normals of the offset surfaces are visualized in a color-coded style.

the offset involves a series of half-plane cutting operations. This approach offers at least two significant benefits. Firstly, its precision is adjustable based on the size of the tetrahedron. Secondly, it effectively preserves sharp features because the only approximation present is within the linear approximation of each triangle-based distance field. In implementation, we utilize specific rules to filter out irrelevant triangle-tetrahedron pairs, enhancing computation speed. Moreover, we recommend merging the n fields, tetrahedron by tetrahedron, to simplify the complexity of the offset surface while maintaining the integrity of sharp features. This process can be formulated as a maximal-clique problem. Our extensive experimental findings demonstrate notable superiority in maintaining sharp features, as highlighted in the teaser figure and Fig. 3.

Our contributions are four-fold:

- We propose an accurate computation of the offset surface while preserving sharp features. Our approach, named *PCO*, leverages the smoothness of triangle-based distance fields except at the triangles themselves.
- We present a discrete implementation that begins with tetrahedralizing the space and slicing each tetrahedron with a set of planes. The size of the tetrahedra can be adjusted to control the precision of the approximation.
- We suggest combining distance fields, tetrahedron by tetrahedron, to simplify the complexity of the offset surface while retaining sharp features. Our method demonstrates advantages over state-of-the-art (SOTA) techniques in terms of accuracy and feature-preserving ability.

- Our approach can be naturally extended to support other morphological operations, such as opening and closing.

2 RELATED WORK

We begin with a concise overview of morphological operations (Section 2.1), before delving into the two principal approaches pertinent to our study: the Minkowski sum (Section 2.2) and volumetric methods (Section 2.3). Finally, we explore additional offsetting techniques in Section 2.4.

2.1 Morphological Operations

Morphology characterizes how a shape expands or contracts based on a specific operation and structuring element (or kernel). Offsetting, encompassing both dilation and erosion, serves as a fundamental component of morphological operations. These basic processes form the foundation for more advanced techniques like opening and closing operations [Sellán et al. 2020].

Mathematical morphology, originally traceable to the field of image processing [Haralick et al. 1987], primarily aims to streamline image data while preserving essential shape characteristics and eliminating irrelevancies. In recent years, this technique has garnered extensive attention within the realm of geometry processing [Calderon and Boubekeur 2014; Nooruddin and Turk 2003; Suriyababu et al. 2023].

2.2 Minkowski Sum

The Minkowski sum is a fundamental operation in computer graphics and is well-defined in mathematical morphology. Utilizing the Minkowski sum represents a direct approach for computing an offset, where the Minkowski sum of a surface and a sphere results in the two-sided offset of the surface. Computing Minkowski sums for convex polytopes is relatively straightforward compared to general polyhedral models. The complexity of computing the Minkowski sum for two convex polyhedra is $O(n^2)$. However, for non-convex polyhedra in 3D, the complexity can escalate to $O(n^6)$ in the worst case [Dobkin et al. 1993].

Given the ease of computing the Minkowski sum for convex polyhedra, most methods that address non-convex polyhedra [Chazelle et al. 1995; Ehmann and Lin 2001; Hachenberger 2009] start with a convex decomposition, followed by the computation and union of the Minkowski sums of the convex components. The Minkowski sum can also be conceptualized as calculating the volume by sweeping one solid with another [Campen and Kobbelt 2010b; Li and McMains 2014], dependent on precise and robust intersection calculations [Campen and Kobbelt 2010a]. However, these approaches are limited by numerical errors from various degenerate cases, requiring robust geometric predicates for accuracy [Milenkovic et al. 2013; Sacks et al. 2011]. Rather than directly computing the exact union of pairwise Minkowski sums, Varadhan and Manocha [2004] derive a signed distance field and extract its zero iso-surface. Peternell and Steiner [2007] demonstrate how to extract the Minkowski sum boundary from the convolution of two objects with piecewise smooth boundaries. Lien [2008] introduce a point-based method to approximate the result by generating a point set covering the boundary of a polyhedron. To mitigate high computational complexity, Leung et al. [2013] propose decomposing the polyhedra into convex pieces and merging the pairwise convex sums in groups on a GPU. Kyung et al. [2015] present a convex convolution algorithm for Minkowski sums with a GPU implementation, employing strategies to eliminate degeneracy and enhance robustness. Some algorithms leverage GPU-based approaches [Li and McMains 2010, 2011, 2014] to compute a voxelized representation, offering an alternative to traditional boundary representation (B-rep) based Minkowski sums.

To summarize, achieving high accuracy in the offset result, especially within concave regions, necessitates discretizing the sphere with a dense mesh. This significantly increases computational time and presents challenges in addressing self-intersection issues, making the Minkowski sum an impractical method for computing offsets.

2.3 Volumetric Approaches

The offset surface can be extracted from the distance field discretized on grid points, which is the basis of volumetric approaches. They concentrate on inferring the distance field [Park et al. 2019; Sharp and Jacobson 2022], followed by some contour extraction methods [Ju et al. 2002; Lorensen and Cline 1987; Sellán et al. 2023] to produce a polygonal representation of the offset surface.

In the context of computing offsets, a primary challenge for these methods is the extraction and preservation of sharp features. Liu and Wang [2009] attempt to simultaneously reconstruct the zero-level surface and an offset surface from an offset point set. However, this direct shifting along the normal direction leads to significant artifacts on the offset surface, particularly failing to reconstruct sharp features for larger offset distances. Various isosurfacing methods aimed at extracting sharp features [Ju et al. 2002; Kobbelt et al. 2001; Qu et al. 2004; Schaefer and Warren 2004] have been extensively employed. Notably, Dual Contouring is often chosen for its ability to incorporate surface normals and recover sharp features. Yet, it is important to acknowledge that the surfaces extracted often suffer from numerous self-intersections, posing problems for further applications. Pavić and Kobbelt [2008] introduced a hybrid method based on the union of primitives, reducing self-intersections through min/max operations on distance functions, though it does not automatically ensure sharp feature reconstruction. A modified Dual Contouring method [Liu and Wang 2010] aims to produce intersection-free offset surfaces while preserving sharp features from triangular mesh surfaces, albeit with potential topological discrepancies from the actual offset surface. Zint et al. [2023] utilized a topology-adapted octree [Varadhan et al. 2004] to maintain disk topology and reduce computational costs, yet cannot completely prevent self-intersections.

A common limitation of these methods is their dependency on high resolution to maintain accuracy. Even with dual contouring (DC), there is a risk of degrading sharp feature points and lines due to the inaccurate discretization of the distance field. Note that in our method, even at a low resolution of tetrahedra, the sharp feature points and lines can still be retained.

2.4 Other Approaches

Ray-based Methods. Ray-based data structures serve as an intermediate representation between traditional boundary representation and volumetric representations. A ray-based representation of a shape is achieved by calculating the intersections of a set of rays with the shape [Menon and Voelcker 1995].

Just a few offsetting methods leverage ray representations. Martinez et al. [2015] utilize a dexel structure [Van Hook 1986] for the solid, approximating morphological operations with a spherical kernel by a zonotope, effectively computing the Minkowski sum of the original shape with a sequence of segments in various directions. Chen et al. [2019] apply morphological operations on ray representations using a dexel structure, particularly for generating surface offsets, but face limitations due to uneven sampling in a single direction, which hampers output mesh reconstruction. Chen and Wang [2011] propose an offsetting method that creates a superset of primitives from an input polyhedron, constructs an LDNI, and filters the LDNI points belonging to the offset surface to remove self-intersections. However, the LDNI model's accuracy is constrained by pixel width, potentially omitting sharp features. Wang and Manocha [2013] approximate an object's boundary with point samples organized by LDI into structured points, computing the offset by merging balls centered at these points on the GPU. This method has limitations, including increased sphere intersections with larger offset distances and varying geometric error bounds, leading to potential topological discrepancies.

Shell Generation Methods. Shell generation relates to offsetting, but they are different. A shell, often seen as a proxy for the original surface, may not necessarily emphasize detail preservation. Wang et al. [2020] introduced a novel approach for constructing a convex polyhedral envelope containing the input model, starting with a prism containing a triangle and cutting it with additional planes to maintain proximity to the triangle. Jiang et al. [2020] built a shell by decomposing a prism into tetrahedra from a triangle mesh, determining extrusion directions to construct the shell, and optimizing to simplify it. Portaneri et al. [2022] developed an algorithm to create a shell enclosing a mesh within a specified offset distance and alpha value, beginning with a coarse mesh and employing carving and refinement steps to generate a watertight and orientable triangle mesh that encloses the input.

Optimization-based Methods. Meng et al. [2018] proposed an algorithm for generating a feature-aligned offset surface using a particle system, though its performance is deemed insufficient for interactive applications. The primary concept involves distributing a set of movable sites uniformly while maintaining a specified distance from the base surface throughout the optimization process, ultimately producing a triangle mesh through a restricted Delaunay triangulation of the sites.

3 OVERVIEW

3.1 Insight

Extracting the δ-offset surface can be understood as computing the Minkowski sum of the input polygonal surface and a sphere with radius δ; see Fig. 4 (a).

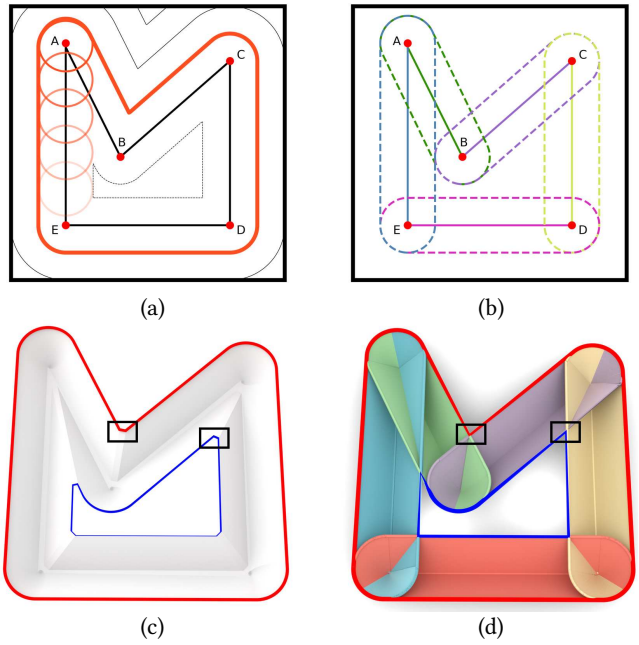


Fig. 4. A 2D example to demonstrate our insight. (a) The offsetting problem can be conceptualized as calculating the Minkowski sum, where a ball rolls along each point on the boundary. (b) The offset of an edge can be straightforwardly computed; by treating each edge as an individual primitive, the offset surface can be created by union operations. (c) Isosurface extraction techniques, which consider the boundary curve as a whole, often fail to capture sharp features effectively. (d) By treating each edge as a primitive and calculating the distance field for each edge separately (visualized in different colors), the offset surface can be obtained by minimizing across these edge-based distance fields.

This process has an alternative implementation. By considering each geometric primitive (e.g., lines in 2D, triangles in 3D) as a unit, one can individually determine the dilation region. The offset surface can then be obtained by computing the union of these individual dilation regions; see Fig. 4 (b).

Prevailing approaches to the offsetting problem typically involve approximating the distance field and then extracting the iso-surface at the distance δ. However, sharp feature points and lines are easily degraded due to the inaccurate discretization of the distance field, diminishing the feature-preserving capability of these methods; see Fig. 4 (c).

We observe that the overall distance field of the entire surface is determined by minimizing over the distance fields of each individual geometric primitive. It is important to note that the distance field of a geometric primitive is smooth throughout the 3D space except at the primitive itself, and the non-differential points of the overall distance field are a result of the minimization operation. Therefore, although our approach involves discretizing individual distance fields, it can still retain sharp feature points and lines on the offset surface even at low resolutions; see Fig. 4 (d).

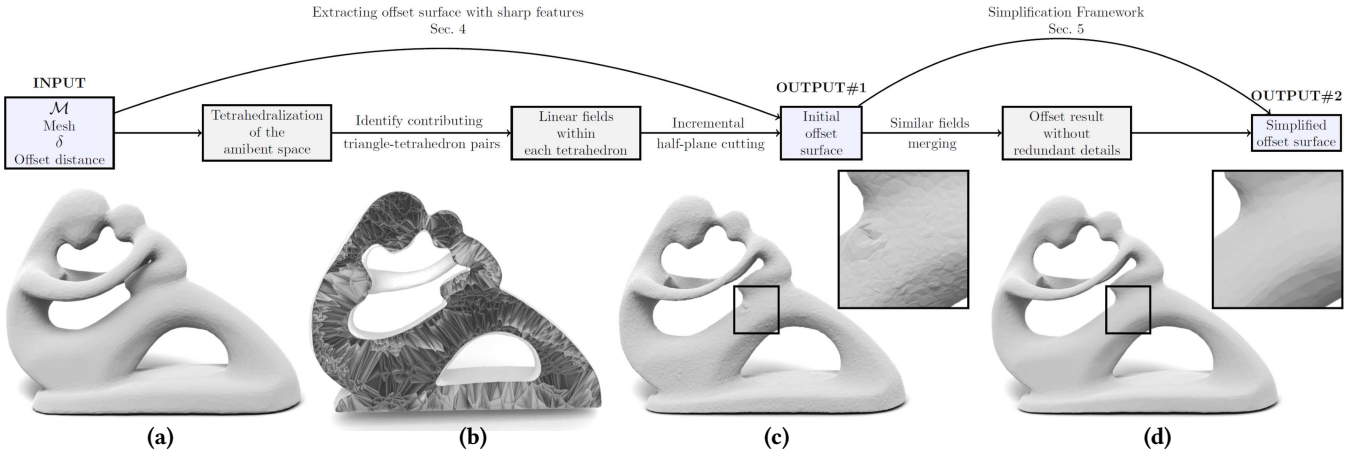


Fig. 5. The pipeline of our algorithm is demonstrated using an example on the fertility model (a), shrunk by 0.5%. The initial step involves applying tetrahedralization to the ambient offset space (b). Following that, each tetrahedron identifies triangles from the original surface that contribute offset within it. Each triangle contributes a piecewise linear distance field for the tetrahedron, leading to the extraction of the initial offset surface shown in (c). To reduce mesh complexity and freely control the precision of offsetting results, we designed a framework that involves merging similar distance fields. This approach significantly reduces mesh complexity while improving accuracy, as demonstrated in (d).

3.2 Pipeline

Given a triangle mesh $\mathcal{M}$ and an offset distance δ, the goal of this paper is to generate an accurate offset surface mesh $\mathcal{M}_\delta$. This offset surface must be a watertight, orientable, manifold surface and devoid of any self-intersections.

As illustrated in Fig. 5, our algorithm begins with tetrahedralization to discretize the ambient space surrounding the offset surface; see Section 4.1. The next step involves computing the linear distance field for each triangle-tetrahedron pair, with further details on filtering out unhelpful pairs provided later. Subsequently, we perform half-plane cutting operations on a tetrahedron-by-tetrahedron basis to approximate the offset surface, as detailed in Section 4.2 and Section 4.4. We then explore a method for combining tetrahedron-wide distance fields to simplify the complexity of the offset surface, which is formulated as a maximal clique problem; see Section 5.

4 OFFSET SURFACE EXTRACTION

4.1 Tetrahedralization of the Ambient Offset Space

We denote the offset distance by δ. In this subsection, we begin by identifying a set of cubic units within the ambient space of the δ-offset surface, and we conclude by dividing each candidate cubic unit into a set of tetrahedra.

Suppose the entire space is discretized into regular cubic units, each with a side length of τ. Let d_c denote the distance from the center of the cube to the base surface. Note that we utilize the P2M method [Zong et al. 2023] to rapidly report the distances from grid points to the base surface. If

$$d_c - \frac{\sqrt{3}}{2}\tau > \delta \quad \text{or} \quad d_c + \frac{\sqrt{3}}{2}\tau < \delta,$$

then it can be concluded that the cubic unit does not intersect the offset surface. In this case, the corresponding cubic unit will be prevented from further refinement.

In our implementation, we used a sufficiently large cube to enclose the target offset surface. The cube is then recursively subdivided into eight smaller cubes until the size of the cubes reaches a predetermined minimal threshold. The default maximum subdivision depth is set to eight. If a cubic unit is deemed to potentially contribute to the offset computation, it is further subdivided into five tetrahedra. This is achieved by cutting off every other vertex, ensuring the central tetrahedron remains regular.

4.2 Linear Field Approximation

Let T_i be a small tetrahedron and t_j be a triangle of the base surface $\mathcal{M}$. We use

$$\mathbf{D}(t_j; T_i) = \{d^{(1)}(t_j; T_i), d^{(2)}(t_j; T_i), d^{(3)}(t_j; T_i), d^{(4)}(t_j; T_i)\}$$

to denote the linear field within T_i, contributed by t_j. It must be noted that a linear field within T_i can be uniquely determined by the distances at T_i's four vertices: $(x_k, y_k, z_k), k = 1, 2, 3, 4$. Let the algebraic equation of the linear field be formed as

$$d = ax + by + cz + w.$$

Assume

$$A = \begin{pmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{pmatrix}. \tag{1}$$

A is invertible if and only if the tetrahedron is non-degenerate (with a non-zero volume). Therefore, as long as the four distances are known, the four variables can be solved immediately:

$$\begin{pmatrix} a \\ b \\ c \\ w \end{pmatrix} = A^{-1} \begin{pmatrix} d^{(1)}(t_j; T_i) \\ d^{(2)}(t_j; T_i) \\ d^{(3)}(t_j; T_i) \\ d^{(4)}(t_j; T_i) \end{pmatrix}. \tag{2}$$

In this manner, the distance field of a single triangle can be readily approximated in a piecewise linear fashion. This approximation can

closely approximate the actual distance field, provided that the linear approximation is confined within a sufficiently small tetrahedron.

4.3 Selection of Contributing Triangle-Tetrahedron Pairs

It is necessary to determine whether a triangle contributes to the tetrahedron T_i. The four vertices of T_i are denoted by v_1, v_2, v_3, and v_4, and $d^{(k)}(\mathcal{M}; T_i)$ represents the distance from the k-th vertex of T_i to the base surface $\mathcal{M}$, $k = 1, 2, 3, 4$. Suppose the circumscribed sphere of T_i has a radius R_i. We immediately have the following theorem:

Theorem 4.1. *The tetrahedron may contribute to the offset at δ only if*

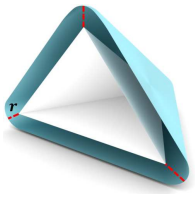
$$\max_{k=1}^{4}\{d^{(k)}(\mathcal{M}; T_i)\} \le \delta + 2R_i, \quad \min_{k=1}^{4}\{d^{(k)}(\mathcal{M}; T_i)\} \ge \delta - 2R_i.$$

The proof is given in the supplementary material A. Similar observation applies to a triangle-tetrahedron pair, as articulated in the following lemma:

Lemma 4.2. *The triangle-tetrahedron (t_j, T_i) may contribute to the offset at δ only if*

$$\max_{k}\{d^{(k)}(t_j; T_i)\} \le \delta + 2R_i, \quad \min_{k}\{d^{(k)}(t_j; T_i)\} \ge \delta - 2R_i.$$

In fact, we can understand the filtering rule from a different perspective. As shown in the inset figure, we imagine that the tetrahedron T_i (considered as a 3D volume) is enlarged by r ($r = |\delta|$), resulting in a larger enclosing surface $\widehat{T_i}$ (visualized in cyan). The triangle-tetrahedron pair (t_j, T_i) may contribute to the offset at δ if $\widehat{T_i}$ encloses or collides with t_j.



4.4 Tetrahedron-range Half-plane Cutting

Consider two triangles, t_1 and t_2, that contribute to the tetrahedron T_i. The corresponding distance fields are denoted as $\mathbf{D}(t_1; T_i)$ and $\mathbf{D}(t_2; T_i)$ respectively. Consequently, the condition $\mathbf{D}(t_1; T_i) = \delta$ defines a plane π_1, and similarly, $\mathbf{D}(t_2; T_i) = \delta$ defines a plane π_2. It is assumed that $\mathbf{D}(t_1; T_i) > \delta$ (resp. $\mathbf{D}(t_2; T_i) > \delta$) determines the positive side of each plane. Typically, π_1 and π_2 divide the entire space into four subspaces. A point $p \in T_i$ may have the following five situations:

(1) $p \in \pi_1^+$ and $p \in \pi_2^+$;
(2) $p \in \pi_1^-$ and $p \in \pi_2^+$;
(3) $p \in \pi_1^+$ and $p \in \pi_2^-$;
(4) $p \in \pi_1^-$ and $p \in \pi_2^-$;
(5) $p \in \pi_1^=$ or $p \in \pi_2^=$,

where $+, -, =$ respectively represent the positive side, the negative side and the plane itself.

Suppose that we are considering the outward offset surface. Obviously, p is in the exterior of the offset surface only if $p \in \pi_1^+$ and $p \in \pi_2^+$ hold at the same time. To align with the direction of gradient, we cut the tetrahedron T_i and retain the positive side of π_1 and π_2:

$$T_i \longrightarrow T_i \cap \pi_1^+ \longrightarrow T_i \cap \pi_1^+ \cap \pi_2^+,$$

which induces an incremental cutting process [Du et al. 2022; Xin et al. 2022] as Fig. 6 shows. During the cutting process, k-dimensional

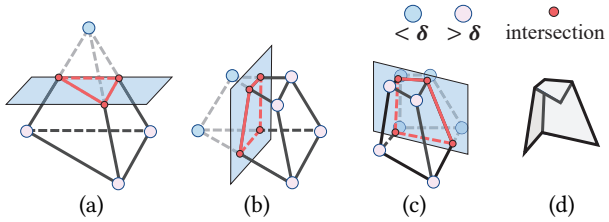


Fig. 6. An example of incremental half-plane cutting. From (a) to (c), the process of incremental cutting by half-planes is depicted, and (d) illustrates the extracted offset surface within the tetrahedron.

primitives ($k = 0, 1, 2$, corresponding to points, edges, and faces) are determined by $3-k$ planes, where the planes are defined based on either the tetrahedron's side faces or linear fields. However, when the number of intersecting planes exceeds $3 - k$, leading to coincident planes, the offsetting result may become non-manifold due to the overlap of multiple primitives at the same location. To address this issue, we retain only one of the overlapping primitives to ensure uniqueness and maintain manifoldness.

When cut by planes, the tetrahedron T_i is transformed into either a convex polytope or becomes empty. This ensures that the final extracted offset result is free of self-intersections. It is noted that when a tetrahedron is completely eliminated by these cuts, it indicates that the tetrahedron does not contribute to the offset surface.

Remark. The cutting process may encounter numerical issues when determining on which side of a plane a point lies. To address this, we utilize *barycentric predicates* proposed by Du et al. [2022] to ensure robustness. Consider a point p as a vertex of the up-to-date offset surface within the tetrahedron T_i. Since p can be considered as the intersection of three planes, each being the δ-offset level set of $\mathbf{D}_j$, $j = 1, 2, 3$. When a new linear distance field $\mathbf{D}_{\text{new}}$ is introduced, its iso-surface at δ also defines a plane π. The side of p with regard to π can be determined by the following sign:

$$\text{sign}(\det(A')) \times \text{sign}(\det(A)), \tag{3}$$

where $A = [\mathbf{D}_1 - \delta, \mathbf{D}_2 - \delta, \mathbf{D}_3 - \delta, 1]$ and $A' = [\mathbf{D}_1 - \delta, \mathbf{D}_2 - \delta, \mathbf{D}_3 - \delta, \mathbf{D}_{\text{new}} - \delta]$. This is implemented using an extension of the Predicate Construction Kit (PCK) provided by Attene [2020].

5 SIMPLIFICATION FRAMEWORK

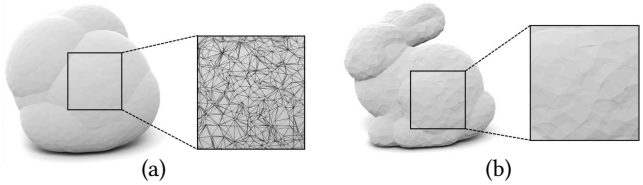
To this end, we obtain a polygonal offset surface by half-plane cutting operations. However, this surface may have a high level of complexity, with numerous unwanted details (weak features), as shown in Fig. 7. This is particularly evident when the input surface contains a large number of triangles compared to the octree resolution. To be detailed, in Fig. 7(a), the high complexity of the offset surface is due to the base surface while in Fig. 7(b), the bumpy features are introduced since linear approximations may produce direction-dependent errors. For both situations, the complexity of the offset surface needs to be reduced.

5.1 Merging Operator

Given a user-specified parameter $\alpha(< 1)$, two distance fields $\mathbf{D}_1 = \mathbf{D}(t_1; T_i)$ and $\mathbf{D}_2 = \mathbf{D}(t_2; T_i)$, are considered as *$\alpha$-similar* within a tetrahedron T_i if

$$\nabla \mathbf{D}_1 \cdot \nabla \mathbf{D}_2 \geq \alpha.$$

By merging these α-similar fields across all tetrahedra, weak features can be eliminated, resulting in a simplified mesh meanwhile.



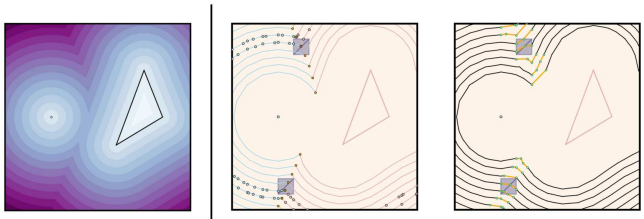
(a) (b)

Fig. 7. In (a), the resulting offset surface is geometrically accurate but the complexity is notably high due to the high resolution of the octree. In (b), when the octree depth is low, the unwanted bumpy features occur since linear approximations may produce direction-dependent errors. Therefore, we need to reduce the the complexity of the offset surface for both situations.

Consider merging the signed distance fields (SDFs) of two sources (the sources can be points, lines, triangles or any other geometric primitives). Merging these fields involves computing a unified SDF that encompasses both sources. In the context of tetrahedron-wise linear distance fields, the merging operator can be defined similarly. Let two linear distance fields, $\mathbf{D}_1 = \mathbf{D}(t_1; T_i)$ and $\mathbf{D}_2 = \mathbf{D}(t_2; T_i)$, contribute to tetrahedron T_i via t_1 and t_2. The merged distance field

$$\mathbf{D}_{12} = \mathbf{S} \odot \min(|\mathbf{D}_1|, |\mathbf{D}_2|),$$

where $\mathbf{S} = [\text{sign}(v_1), \text{sign}(v_2), \text{sign}(v_3), \text{sign}(v_4)]$ represents the in-out flags of the four vertices, and $\odot$ represents the element-wise product. The purpose of merging two similar distance fields is to eliminate weak features (see Fig. 8). The merged form is analyzed further in the supplementary material B.



(a) Ground truth (b) Ours (before) (c) Ours (after)

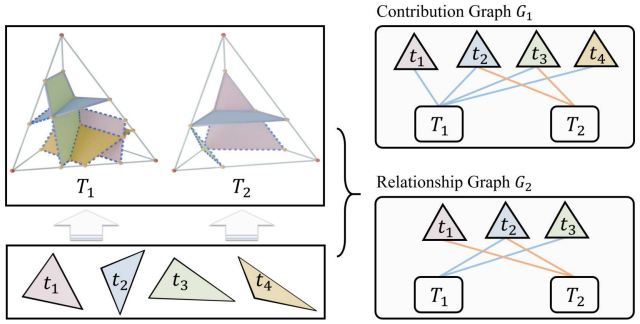
Fig. 8. In (a), we visualize the distance field originating from two sources: one is a point and the other is a triangle. (b) and (c) visualize the distance fields before and after the merging operation, respectively. By merging similar distance fields, weak features are eliminated while prominent features are retained.

5.2 Merging Fields Across Tetrahedra

Although the merging operation within a single tetrahedron appears straightforward, merging fields across multiple tetrahedra is non-trivial. A merging operation that is not carefully designed may lead to inconsistencies, such as gaps and outliers, across tetrahedra.

Identify α-similar fields. Before delving into the details of merging conditions, we first introduce two undirected graphs: G_1 (*Contribution Graph*) and G_2 (*Relationship Graph*). G_1 retains the contributing triangle-tetrahedron pairs and is constructed as follows: If a triangle t_j contributes to a tetrahedron T_i, an edge is established between t_j and T_i in G_1. G_2, a subgraph of G_1, includes the triangle-tetrahedron pairs whose distance fields actually contribute to the final extracted offset surface. G_1 and G_2 can be constructed directly after the offset surface has been computed, as shown in Fig. 9. To this end, G_1 and G_2 can be written as

$$G_1 = (V_1, E_1),\ G_2 = (V_2, E_2).$$



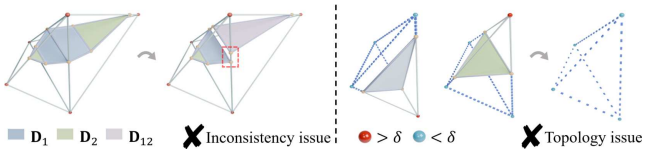
(a) (b)

Fig. 9. A diagram illustrates how the graph of fields is constructed. In (a), planes enclosed by solid lines in different colors represent the extracted offset within each tetrahedron, while the dotted lines represent the contributing tetrahedron-wise fields that do not pertain to the extracted offset. Two graphs shown in (b) are established from (a). It is noteworthy to highlight that t_4 does not appear in G_2 because it is entirely eliminated by two half-planes formed by t_1 and t_3 in T_1, and also dose not contribute to T_2.

Merging conditions and compatible graph. In G_2, each triangle is associated with a set of tetrahedra. Let $V_2(t_i)$ denote the tetrahedra related to the triangle t_i in G_2. Two triangles $t_i \in G_2$ and $t_j \in G_2$ can be merged only if specific conditions are met:

(1) $V_2(t_i) \cap V_2(t_j) \neq \emptyset$, and $\mathbf{D}_i$ and $\mathbf{D}_j$ are α-similar in each tetrahedron of $V_2(t_i) \cap V_2(t_j)$,
(2) The merged tetrahedron-wise distance fields $\mathbf{D}_{ij}$ still contributes to the offset, by referring to G_1.

The first condition is to enforce consistency, and the other is to address outlier issues (see Fig. 10).



(a) (b)

Fig. 10. In (a), merging the two distance fields $\mathbf{D}_1$ and $\mathbf{D}_2$ within only a single tetrahedron creates a gap between two adjacent tetrahedra (as shown in highlighted windows). In (b), the merged distance field $\mathbf{D}_{12}^{(k)}$ $(k = 1, 2, 3, 4)$ within the same tetrahedron are less than δ, they no longer contribute to offset, leading to its elimination and resulting in a non-watertight output.

ALGORITHM 1: Incremental Fields Merging

Input: Initial distance fields $\mathbf{D}(t;T)$, contribution graph $G_1 = (V_1, E_1)$ and relationship graph $G_2 = (V_2, E_2)$,
Output: Merged distance fields $\mathbf{D}_u(t;T)$
$\mathbf{D}_u(t;T) \leftarrow \mathbf{D}(t_j;T)$;
Construct compatibility graph $G_3 = (V_3, E_3)$ from G_2;
while $|E_2| \neq 0$ **do**
  Compute all maximal cliques $\{MC_i\}_{i=1}^m$ of G_2;
  $C_{\text{out}} \leftarrow$ **Algorithm 2** ($\{MC_i\}_{i=1}^m$);
  Calculate the merged distance field with regard to C_{out} and update G_1, $\mathbf{D}_u$.
  Reconstruct G_2 from G_1; // Here the half-plane cutting process will be employed.
  Reconstruct G_3 from G_2;
end
return $\mathbf{D}_u$.

ALGORITHM 2: Greedy Clique Selection

Input: A set of maximal cliques $\{MC_i\}_{i=1}^m$ of compatibility graph G_2
Output: The selected cliques C_{out}
$C_{\text{out}} = \emptyset$ // stores result
Initialize an empty priority queue $\mathbf{Q}$; // The priority depends on the size and the sum of edge weight of a clique
Initialize an empty set VIS to store old data in $\mathbf{Q}$;
for *each maximal clique* $mc \in \{MC_i\}_{i=1}^m$ **do**
  Push mc to $\mathbf{Q}$;
end
while $\mathbf{Q}$ *is not empty* **do**
  Take out the top-priority clique c;
  if $c \in VIS$ *or* $|c| < 2$ **then**
    continue;
  end
  Append c to the result C_{out};
  for *each triangle* t *in* c **do**
    if *an other clique* $c_r \in \{MC_i\}_{i=1}^m$ *contains* v **then**
      Append c_r to the set VIS;
      Remove t in other cliques c_r and update the sum of edge weight;
      Remove edges connected to t; // Update graph G_2
      Push the newly clique c_r to $\mathbf{Q}$; // Update priority queue
    end
  end
end
return C_{out}.

Based on the above discussion, we construct another undirected weighted *Compatibility Graph*

$$G_3 = (V_3, E_3)$$

to represent the compatibility between triangles, where $V_3 \subset V_2$. Two triangles $t_i, t_j \in V_3$ can be merged only if they are connected. The cost for merging t_i and t_j is thus defined by

$$W_{t_i,t_j} = \sum_{T \in V_2(t_i) \cup V_2(t_j)} [1 - \nabla\mathbf{D}(t_i;T) \cdot \nabla\mathbf{D}(t_j;T)]. \tag{4}$$

Therefore, the larger W_{t_i,t_j} is, the higher the priority with which we merge t_i and t_j.

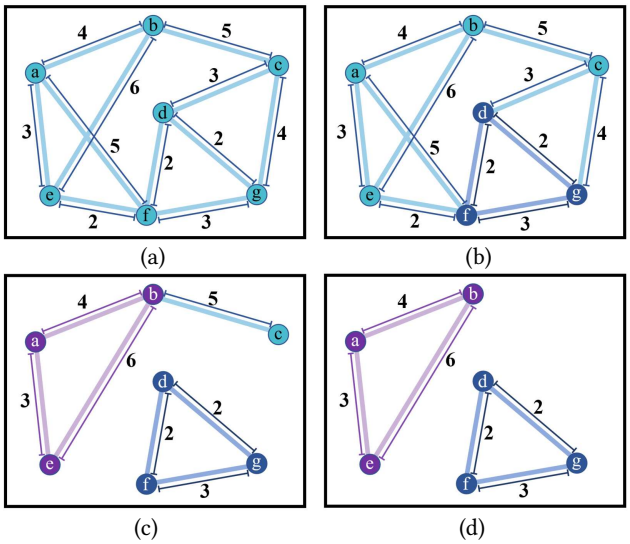


Fig. 11. An example of our greedy clique selection. In the original graph (a), we begin by selecting a maximal clique $\{d, f, g\}$, chosen for having the maximum sum of edge weights, as shown in (b). Subsequently, the edges $\{(a, f), (e, f), (d, c), (g, c)\}$ associated with the clique are removed, as depicted in (c). Finally, within the resulting sub-graph, the clique a, b, e is selected, as shown in (d).

Incremental merging. The set of compatible triangle nodes corresponds to the cliques in G_3, due to the non-transitivity of compatibility. It is likely that a merged distance field can be further merged with another one. Therefore, we have developed an incremental merging algorithm, as outlined in Algorithm 1.

In each iteration, we employ a greedy merging scheme that selects the clique with the largest number of nodes, aimed at reducing the number of iterations. When two cliques are of the same size, the one with the higher overall cost is prioritized.

Finding all maximal cliques is proven to be an **NP-complete** problem without any polynomial-time algorithm. In our implementation, we use the method proposed by Eppstein et al. [2013] to calculate all maximal cliques. The time complexity is $O((n - d_g)3^{\frac{d_g}{3}})$, where $n = |V_3|$ and d_g is the graph's degeneracy. It is evident that as α decreases, G_3 becomes sparser (i.e., with very low values of d_g), making the algorithm more efficient. Subsequently, the desired cliques are chosen from among the maximal cliques, as depicted in Fig. 11. The corresponding pseudocode is shown in Algorithm 2.

6 EXPERIMENTS

We implemented our algorithm in C++ using Eigen [Guennebaud et al. 2010] for linear algebra routines and CGAL [The CGAL Project 2024] and libigl [Jacobson et al. 2018] for basic geometry processing routines. All experiments were performed on a PC with a 32-core Intel CoreTM i9-13900K CPU clocked at 3.0 GHz and 64 GB of memory. The implementation details are provided in the supplemental material.

We use consistent parameter settings in our experiments. For consistency, the offset value δ is formulated as a ratio with respect to the diagonal length L_b of the model's bounding box, meaning that

Fig. 12. A gallery of our feature-preserving offsetting results from inputs of freeform and CAD models.

(a) CAD **(b) Self-intersection & thin-plate** **(c) Non-manifold & thin-plate**

Fig. 13. Our method efficiently recovers sharp features, as indicated by the red solid lines in (a). Additionally, our method can handle non-manifold, self-intersecting models and ensures valid output. It can preserve fine details even in models with thin plates, as shown in highlighted windows in (b) and (c).

$\delta = \pm 1\%$ corresponds to a threshold of $\delta = \pm L_b/100$. Additionally, the octree's depth is set to eight by default.

Evaluation metrics. We employ four metrics to assess the accuracy of the offsetting results. The first set of metrics includes the relative one-sided ($\mathcal{M}_\delta \rightarrow \mathcal{M}$) *Chamfer Distance* ($d_{\mathrm{C}}$) and *Hausdorff Distance* (d_{H}) [Wang and Manocha 2013; Zint et al. 2023]. To further access the consistency of the normals with the original surface, we compute the *Mean Absolute Error* of the normal angles ($\mathrm{N}_{\mathrm{MAE}}$) between $\mathcal{M}_\delta$ and $\mathcal{M}$. Additionally, we introduce a new metric, the *N-Score*, which measures the percentage of normal deviations using a threshold of 5°. More details can be found in our supplementary material D.1.

6.1 Extensive Validation

A primary feature of our algorithm is its robustness to defects. We validate the effectiveness on extensive datasets with complex, often defect-laden meshes: Thingi10K [Zhou and Jacobson 2016] and ABC [Koch et al. 2019]. When the input model is closed and watertight, we generate a single-layer offset by computing signed distances. Instead, when the input model has non-manifold or self-intersecting artifacts, we generate a double-layer offset by computing unsigned distances. A gallery of offsetting results is shown in Fig. 12. Error statistics for Thingi10K (10K models) and a subset of ABC (7,482 models) are presented in Fig. 14.

Fig.13 (a) illustrates inward and outward offsetting results of a CAD model, highlighting our algorithm's ability to preserve sharp features. Fig.13 (b) and (c) present more challenging inputs, including non-manifold elements, self-intersections, and models with thin

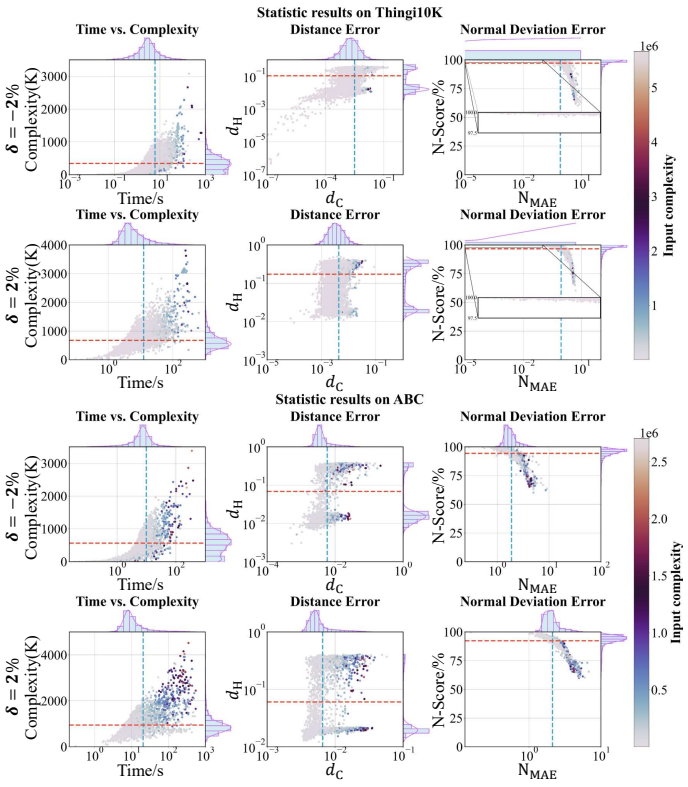


Fig. 14. Statistical results of our method on Thingi10K and a subset of ABC are presented. The offset distances are set at -2% and 2%. The scatter plot shows the metric values for different models, with the blue and red dashed lines representing the mean values of the dataset for the x-axis and y-axis metrics, respectively. The edge histograms provide the frequency distribution corresponding to the scatter plot. The average values of d_C and d_H are generally below $\mathbf{10^{-1}}$, N_{MAE} values are predominantly within $[\mathbf{0}°, \mathbf{5}°]$, and most N-Scores approach 100%, with all exceeding **60%**. In fact, for Thingi10K, the proportion of N-Score **exceeding 90.0%** is **93.67%** for $\delta = -2\%$ and **92.48%** for $\delta = 2\%$. For ABC, the proportions are **89.70%** and **84.95%** for $\delta = -2\%$ and $\delta = 2\%$ respectively. These results demonstrate the high accuracy of our algorithm. The runtime performance depends on the complexity of the input mesh, but the total time remains under $\mathbf{10^3}$ seconds even for meshes with millions of triangles.

plates. Despite these complexities, our algorithm consistently produces desired outputs while maintaining its effectiveness and precision.

6.2 Ablation Studies

There are two primary parameters influencing our extracted offset result: the octree depth, d, and the offset distance, δ.

Octree depth vs. offset distance. We now investigate the influence of the tetrahedron resolution and the offset distance. With a small δ, if the octree depth d is too low, the errors in the linear approximation will increase, potentially resulting in bumpy features. However, with a larger δ, the output mesh becomes smoother and more accurate, as illustrated in Fig. 15 and Fig. 16.

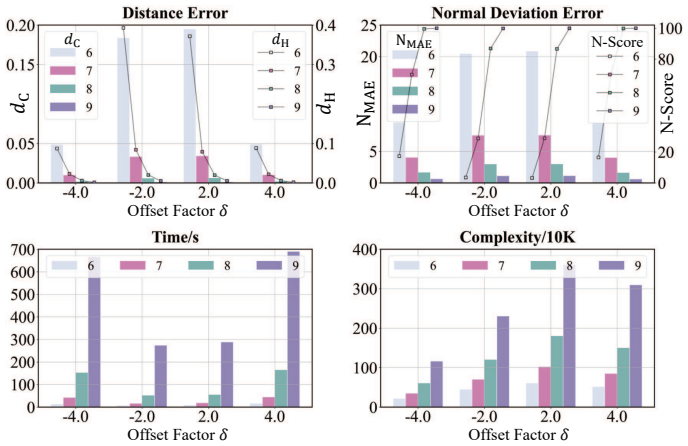


Fig. 15. Quantitative comparison of different offset distances and octree depths.

It is important to note that while the accuracy of the offset surface with a small δ depends on the tetrahedral resolution, the ability to preserve sharp features remains consistent regardless of resolution, which is a significant advantage of our algorithm.

Varying offset distance. Regardless of the offset distance (δ), our algorithm consistently produces the desired results while preserving sharp features, even when δ increases significantly, as depicted in Fig. 17. Table 1 shows that as δ increases, the results become more precise. This is attributed to our linear approximation being highly accurate in regions far from the source triangle.

Fig. 16. Visual comparison of different offset distances and octree depths. The feature lines are highlighted in red.

6.3 Comparisons

To comprehensively evaluate our algorithm, we selected several state-of-the-art approaches capable of generating offset surfaces with available implementations. We conducted comparison experiments on three types of models:

Table 1. Quantitative comparison results for different models under varying distances. The octree depth is set to 8. As the offset distance increases, our method achieves more accurate results.

		δ_1	δ_2	δ_3	δ_4	δ_5	δ_6	δ_7	δ_8	δ_9
Time/s	Fandisk	6.87	3.60	3.00	3.23	4.30	7.72	15.98	45.90	113.31
	34784	29.72	15.92	8.42	7.60	8.79	8.80	15.17	40.04	98.81
	54725	6.43	6.29	6.75	12.80	25.31	67.52	112.16	267.71	414.36
	Dancing Children	12.67	6.88	9.72	21.73	29.60	51.33	75.38	113.24	153.64
d_C ($\times 10^{-3}$) ↓	Fandisk	0.737	1.939	7.683	6.413	2.596	1.027	0.459	0.243	0.147
	34784	0.410	0.982	3.015	10.093	10.701	3.854	1.303	0.512	0.245
	54725	11.701	66.866	78.887	16.981	5.229	1.652	0.672	0.293	0.235
	Dancing Children	17.213	65.962	69.515	18.830	6.190	3.425	2.291	1.681	1.311
d_H ($\times 10^{-3}$) ↓	Fandisk	51.618	42.655	76.682	65.625	75.832	7.540	21.714	37.146	2.761
	34784	26.170	16.966	58.679	101.173	183.779	192.103	85.695	33.719	6.366
	54725	41.865	241.676	320.430	57.696	21.348	4.607	1.549	0.623	0.489
	Dancing Children	331.589	331.348	71.749	73.517	19.842	9.397	28.266	34.537	8.668
N_{MAE} ↓	Fandisk	0.670	1.215	2.165	2.322	1.181	0.689	0.421	0.309	0.261
	34784	0.417	0.677	1.370	2.528	2.482	1.219	0.620	0.356	0.247
	54725	3.192	10.810	11.017	4.277	2.220	1.233	0.762	0.524	0.479
	Dancing Children	12.671	10.276	10.312	4.712	2.354	1.526	1.083	0.831	0.656
N-Score (%) ↑	Fandisk	99.98	98.21	86.70	85.35	98.10	99.96	99.98	100.00	100.00
	34784	99.98	99.99	97.84	81.76	82.07	97.85	99.99	100.00	100.00
	54725	82.07	25.62	24.51	66.30	95.24	99.94	100.00	100.00	100.00
	Dancing Children	27.27	27.52	26.53	58.83	91.23	98.69	99.69	99.86	99.94

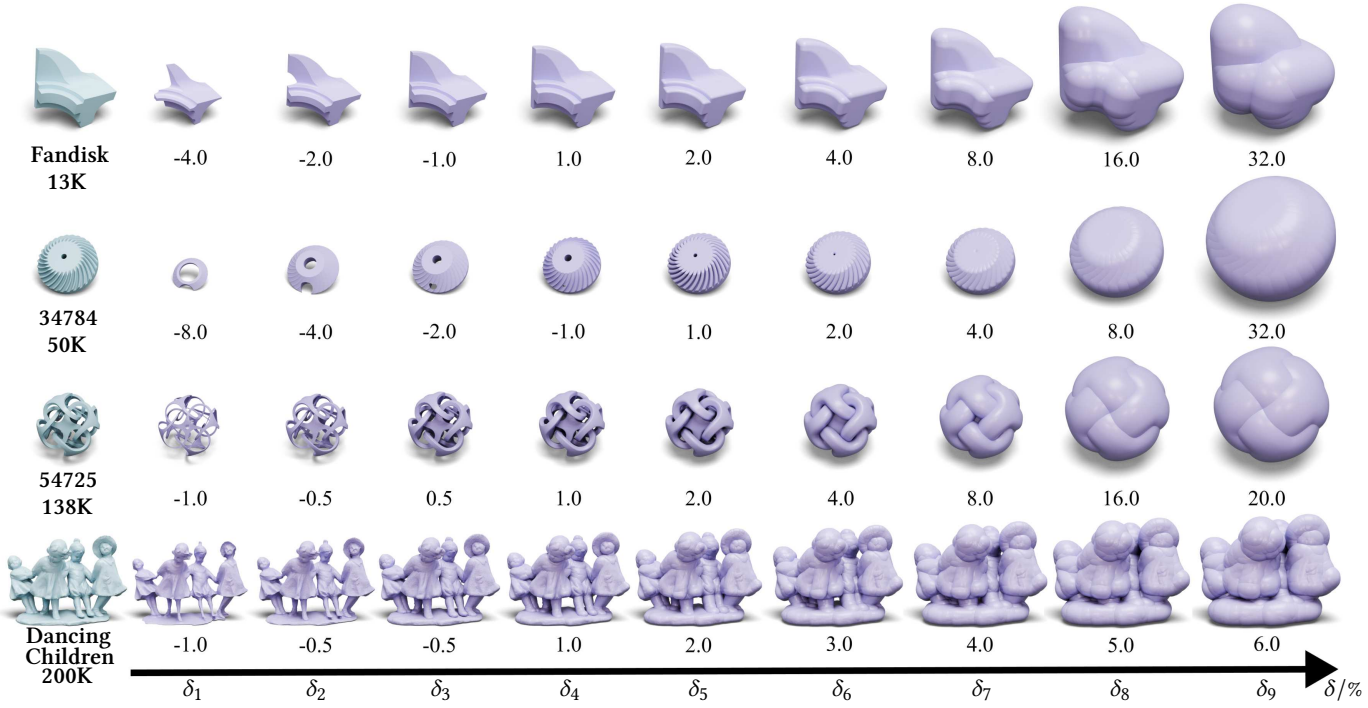


Fig. 17. Our method can produce accurate offsets with varying distance, where the octree depth is set to 8.

- CAD models from the ABC dataset [Koch et al. 2019]. CAD models often exhibit prominent feature lines, thus used to test the capability of preserving sharp features. The results are visualized in the first row of Fig. 18.

Table 2. Quantitative comparison results with state-of-the-art works on three models under two offset distances are presented (corresponding to Fig. 19). Full statistical data is provided in the supplementary material D. The **best** scores are highlighted in bold with underlining, while the **second best** scores are highlighted in bold.

		Time/s			d_C ($\times 10^{-3}$) ↓			d_H ($\times 10^{-3}$) ↓			N_{MAE} ↓			N-Score/% ↑		
		Dragon	Angel	Puck	Dragon	Angel	Puck	Dragon	Angel	Puck	Dragon	Angel	Puck	Dragon	Angel	Puck
δ_1	DC	1203.00	360.98	1064.32	**2.228**	3.168	**0.837**	26.193	55.081	**22.468**	22.977	7.767	**6.112**	5.55	65.10	**67.73**
	FPO	7573.34	8231.20	-	**1.107**	**1.635**	-	**17.287**	**28.884**	-	**18.799**	8.982	-	62.20	51.46	-
	HSP	**9.35**	**26.88**	**27.03**	4.048	5.7762	4.171	82.456	264.311	99.041	18.871	**5.670**	8.296	**65.73**	**71.10**	59.26
	Ours	**462.73**	**58.51**	**315.06**	2.883	**2.358**	**0.132**	**20.828**	**23.370**	**12.216**	**4.153**	**3.495**	**3.465**	**69.16**	**75.84**	**80.07**
δ_2	AW	**159.88**	222.89	**187.23**	**1.098**	**3.008**	0.599	**22.782**	77.075	18.534	**4.381**	4.393	4.048	**77.63**	73.50	78.66
	DC	1230.44	369.19	1118.53	2.150	**2.641**	**0.593**	24.576	**66.593**	**13.772**	7.431	6.239	**3.486**	62.00	71.33	**83.93**
	FPO	5350.55	1678.08	4422.47	12.742	6.954	8.290	350.560	321.060	284.792	8.581	7.095	58.696	43.31	57.91	38.93
	HSP	**17.54**	**36.65**	**35.63**	6.206	5.385	4.149	66.947	86.129	190.982	4.541	**3.997**	**3.509**	**72.46**	**74.62**	80.13
	Ours	620.47	**67.71**	327.32	**1.114**	8.815	**0.420**	**13.688**	**63.401**	**8.367**	**3.902**	**3.304**	3.703	70.65	**80.00**	**84.91**

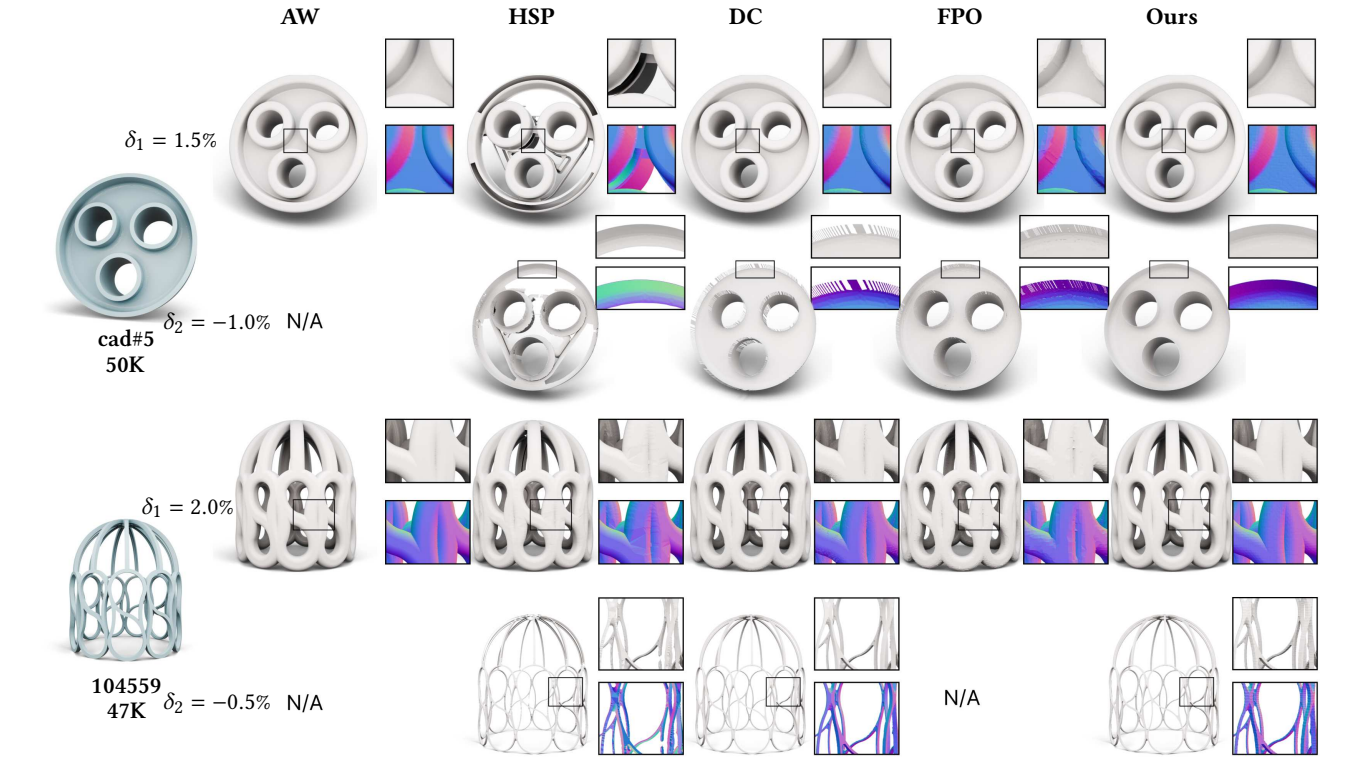


Fig. 18. A visual comparison on CAD and thin-plate models demonstrates that our approach ensures correct topology while effectively recovering sharp features and preserving thin-plate details. Additional results are included in the supplementary material D.4 and D.5.

- Twisted or thin-plate models from the Thingi10K dataset [Zhou and Jacobson 2016]. It presents challenges in preserving sharp features and correctly handling intersections when perform the offsetting process on these models. The results are visualized in the second row of Fig. 18.
- Models reconstructed from real raw large scan data [Huang et al. 2022; Levoy et al. 2000; Zhang et al. 2021]. These models always exhibit rich geometric details and have millions of faces, which presents challenges in preserving sharp features while maintaining high performance. The results are visualized in Fig. 19.

The complete visual and quantitative results are provided in the supplementary material.

Comparison with DC [Ju et al. 2002]. Dual contouring is a well-known algorithm designed to extract isosurfaces that maintain sharp features. It is necessary to compare DC with ours in preserving sharp features. Note that we utilize an implementation from libigl and set the uniform grid resolution to 1024. Despite the high resolution, it

is hard for DC to capture fine details. Its results tends to include degraded features, as illustrated in Fig. 19.

Comparison with FPO [Zint et al. 2023]. This volumetric method employs dual contouring [Ju et al. 2002] to extract the offset surface using an adaptive octree that refines in areas with complex offset topologies. To maintain consistency with its default configurations, the maximum subdivision depth of the octree is set to 10, with two additional levels added to resolve non-manifold elements.

Although dual contouring is utilized, recovering sharp features remains challenging, especially in regions with dense features. Moreover, the method occasionally fails to compute inward offsets for some models, as shown in Fig. 18 and Fig. 19.

Comparison with HSP [Chen et al. 2019]. This approach computes offsets from a ray-rep representation. It utilizes a dexel structure to discretize the input shape, followed by the construction of an offset surface through the computation of a half-space power diagram. In our experiments, the resolution was set to 1024. Since the output retains the dexel structure, we first convert it into a point cloud and then reconstruct it into a mesh [Boltcheva and Lévy 2017] for a clearer and more meaningful comparison.

A significant drawback of this method is its poor precision, as evidenced in Table 2. Additionally, it is clear that the method struggles to recover sharp features.

Comparison with AW [Portaneri et al. 2022]. The method, named *Alpha Wrapping*, is designed for approximately computing outward but not inward offset surfaces. It controls the quality and precision of the generated mesh through a parameter α_m. A smaller value of α_m results in improved accuracy of the generated offset surface. In our experiments, we set α_m to $\frac{L_b}{1024}\%$. However, even with a minimized α_m, the method struggles to preserve sharp features.

Overall, compared to these methods, our algorithm consistently demonstrates better accuracy and superior feature-preserving ability. However, our method does not outperform *HSP* in terms of runtime performance.

6.4 Offset Simplification

Table 3. Quantitative results for the bunny model under various θ values are presented. **The octree depth is set to 6.**

		w/o Merging	5°	10°	15°	20°	25°	30°
δ=-2%	Time/s	-	**0.97**	4.03	5.67	7.04	8.05	9.00
	Complexity	75986	75425	71909	66045	62170	58030	**54299**
	d_C ($\times 10^{-3}$) ↓	47.601	47.406	46.559	42.932	39.144	36.088	**33.618**
	d_H($\times 10^{-3}$) ↓	189.144	172.651	169.579	179.071	**151.260**	164.880	162.260
	N_{MAE} ↓	9.348	9.502	9.668	9.415	**8.868**	8.876	9.015
	N-Score ↑	28.15	28.36	27.62	30.19	33.23	**34.70**	31.34
δ=2%	Time/s	-	**0.61**	2.56	4.77	5.48	6.05	7.12
	Complexity	92536	91552	85785	81165	76703	72737	**69896**
	d_C ($\times 10^{-3}$) ↓	63.741	62.111	60.784	59.253	55.823	51.417	**49.044**
	d_H ($\times 10^{-3}$) ↓	198.356	198.841	198.831	192.456	198.080	199.440	**176.440**
	N_{MAE} ↓	10.685	10.568	10.537	9.507	9.592	**9.476**	9.868
	N-Score ↑	21.53	21.25	21.57	26.76	25.33	**29.17**	26.29

The parameter α in our algorithm is used to control the target complexity of the offset surface. For better clarity in the following discussion, we define an angle θ depending on α:

$$\theta = \arccos(\alpha).$$

To investigate the impact of θ (or α) on controlling precision and the degree of simplification in offset generation, we conducted experiments with different values of θ (or α).

Table 4. Quantitative results for the Sappho's Head model under different θ values are presented. **The octree depth is set to 8.**

		w/o Merging	2°	5°	8°	10°	12°	15°
$\delta = -2\%$	Time/s	-	**15.51**	36.81	67.23	98.35	105.31	120.78
	Complexity	540678	538266	532843	522024	514184	509809	**506728**
	d_C ($\times 10^{-3}$) ↓	0.118	**0.102**	0.111	0.115	0.106	0.127	0.132
	d_H ($\times 10^{-3}$) ↓	0.386	**0.382**	0.384	0.384	0.385	0.385	0.385
	N_{MAE} ↓	1.430	1.321	1.351	**1.269**	1.495	1.572	1.416
	N-Score ↑	95.67	96.48	96.17	**97.62**	94.36	94.90	95.10
$\delta = 2\%$	Time/s	-	**41.38**	102.42	124.99	139.02	160.03	173.56
	Complexity	712335	707375	693302	689995	684514	676882	**678139**
	d_C ($\times 10^{-3}$) ↓	0.124	0.135	0.117	0.129	0.147	0.124	**0.114**
	d_H ($\times 10^{-3}$) ↓	0.384	0.384	**0.379**	0.383	0.385	0.384	0.382
	N_{MAE} ↓	1.575	1.680	1.548	**1.539**	1.652	1.628	1.641
	N-Score ↑	95.06	92.67	94.88	**95.44**	94.86	94.90	93.18

Fig. 20 illustrates the local details of the generated offset surface under various values of α. It is evident that as θ increases (corresponding to a decrease in α), weaker features are removed, resulting in a smoother surface and simultaneously enhancing precision. It is worth noting that when θ increases, more merging operations need to be conducted, thereby increasing the total computation time.

As shown in Fig. 21, with the increase in θ (corresponding to a decrease in α), weak features are eliminated (as seen in the highlighted windows), while sharp features are retained in the offsetting results. In other words, our offset simplification algorithm prioritizes the removal of weak features over sharp features.

7 EXTENSION TO OPENING AND CLOSING

Offsetting (erosion and dilation) operations are foundational building blocks in morphology. They are distinct yet related to opening and closing operations, which can be defined as combinations of them (assuming in L_2 space):

$$\text{opening}(\mathcal{M}, \mathcal{S}) = \mathcal{M}_\delta^o = ((\mathcal{M} \ominus \mathcal{S}) \oplus \mathcal{S}) \tag{5}$$

$$\text{closing}(\mathcal{M}, \mathcal{S}) = \mathcal{M}_\delta^c = ((\mathcal{M} \oplus \mathcal{S}) \ominus \mathcal{S}) \tag{6}$$

In other words, the opening and closing operations can be accomplished by utilizing a two-stage offsetting computation. Unlike the offsetting operation, the opening and closing operations preserve the original shape while filling overly convex or concave parts, using a user-specified parameter. This feature is particularly useful in various applications, such as a cleanup and analysis tool for preprocessing 3D printing shapes and computational fabrication. It is also beneficial for mesh repairing and simplification in geometric modeling. Refer to Fig. 22 for a visual representation.

8 CONCLUSION AND LIMITATIONS

In this paper, we propose a method for accurately computing offset surfaces while maintaining sharp features, termed *PCO*. This research is based on the smoothness of triangle-based distance fields,

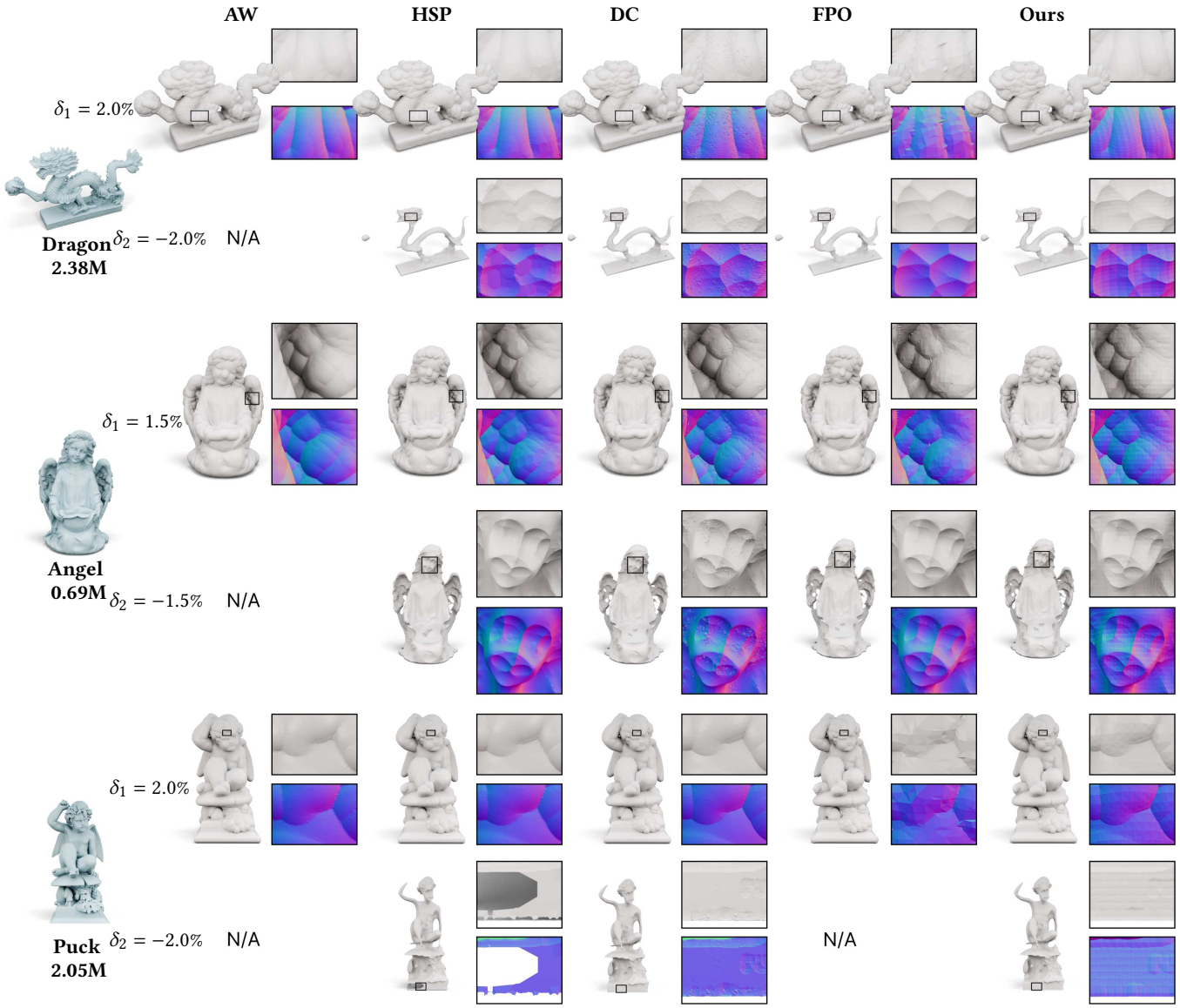


Fig. 19. A visual comparison of models reconstructed from actual raw scans is presented. The highlighted differences demonstrate that our algorithm can produce surfaces with high-fidelity geometric details and ensure normal consistency. Additional results are included in supplementary material D.6.

except at the triangles themselves. After the ambient space of the offset surface is discretized into tetrahedral units, the offset surface is traced through plane cutting. Therefore, although our approach involves discretizing individual distance fields, it can still retain sharp feature points and lines on the offset surface even at low resolutions, unlike existing approaches whose discretized distance fields may degrade the sharp features. We further discuss the operation of merging distance fields, formulated as a maximal-clique problem, to simplify the complexity of the offset surface. Our algorithm includes a pair of parameters to tune the precision: one is the size of the tetrahedra, and the other is the angle parameter for merging two distance fields. Additionally, it can efficiently handle additional morphological operations such as opening and closing.

Extensive experimental results demonstrate our method's advantages over state-of-the-art (SOTA) techniques in terms of accuracy and feature-preserving capabilities.

It is important to acknowledge that although our current method excels in experimental comparisons, it does not yet achieve the desired runtime performance. This shortfall is primarily due to the extensive use of linear fields in computations when the offset distance is significantly large or the discretization units are very small. Given its parallel nature, we aim to develop a GPU-based version in future work to expedite the computation process.

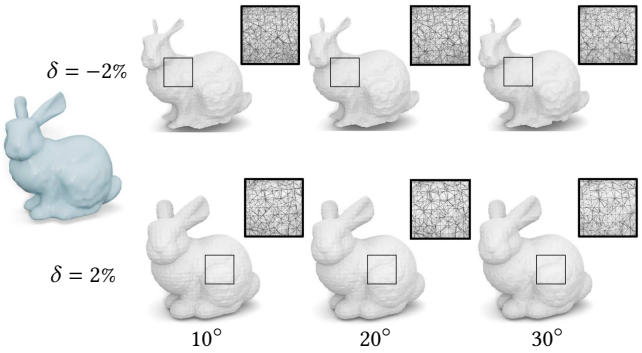


Fig. 20. Visual comparison of offsetting results under different alpha values. Highlighted differences visualize the smoothing and simplifying effects.

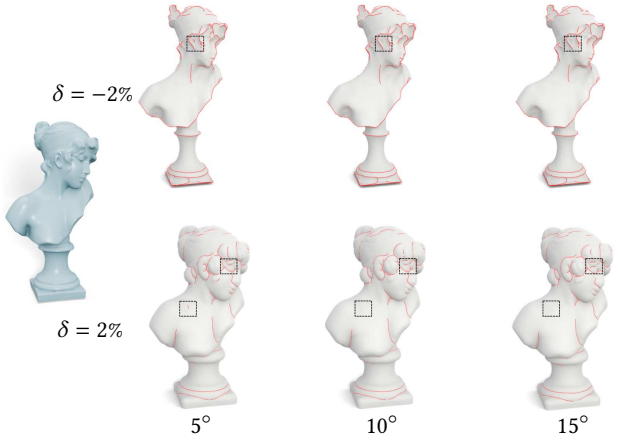


Fig. 21. Visual comparison of offset surfaces under different alpha values. More weak features (see highlighted windows) are eliminated when θ becomes large.

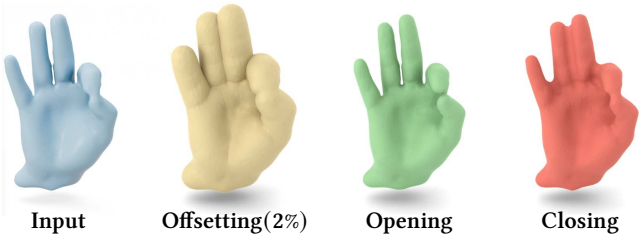


Fig. 22. Unlike the offsetting operation, the opening and closing operations preserve the original shape while filling the overly convex or concave parts with a user-specified parameter. This is particularly useful in consolidating mechanical structures. The example highlights their use in various geometry processing applications.

Another drawback of our method is that the outcomes often lack high triangulation quality. Currently, one must employ third-party remeshing techniques, such as Hu et al. [2018], to enhance triangulation quality, as illustrated in Fig. 23.

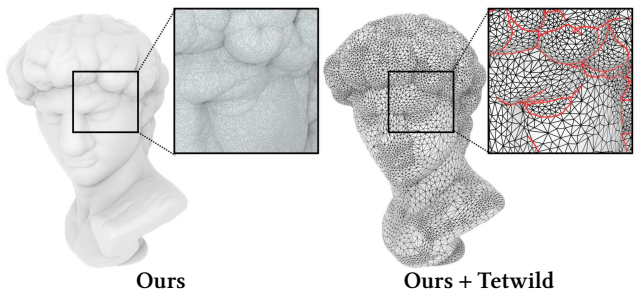


Fig. 23. By using Tetwild [Hu et al. 2018] as a post-processing technique, we can remesh our results without compromising the feature lines.

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 (2021YFB1715900), and National Natural Science Foundation of China (62272277, U23A20312, 62072284, 61772314, 61761136017, and 62072275).

REFERENCES

Marco Attene. 2020. Indirect Predicates for Geometric Constructions. *Computer-Aided Design* 126 (2020), 102856. https://doi.org/10.1016/j.cad.2020.102856

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

Stéphane Calderon and Tamy Boubekeur. 2014. Point morphology. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 1–13.

Marcel Campen and Leif Kobbelt. 2010a. Exact and robust (self-) intersections for polygonal meshes. In *Computer Graphics Forum*, Vol. 29. Wiley Online Library, 397–406.

Marcel Campen and Leif Kobbelt. 2010b. Polygonal boundary evaluation of minkowski sums and swept volumes. In *Computer Graphics Forum*, Vol. 29. Wiley Online Library, 1613–1622.

Bernard Chazelle, David P Dobkin, Nadia Shouraboura, and Ayellet Tal. 1995. Strategies for polyhedral surface decomposition: An experimental study. In *Proceedings of the eleventh annual symposium on Computational geometry*. 297–305.

Yong Chen and Charlie CL Wang. 2011. Uniform offsetting of polygonal model based on layered depth-normal images. *Computer-aided design* 43, 1 (2011), 31–46.

Zhen Chen, Daniele Panozzo, and Jeremie Dumas. 2019. Half-space power diagrams and discrete surface offsets. *IEEE Transactions on Visualization and Computer Graphics* 26, 10 (2019), 2970–2981.

David Dobkin, John Hershberger, David Kirkpatrick, and Subhash Suri. 1993. Computing the intersection-depth of polyhedra. *Algorithmica* 9, 6 (1993), 518–533.

Xingyi Du, Qingnan Zhou, Nathan Carr, and Tao Ju. 2022. Robust computation of implicit surface networks for piecewise linear functions. *ACM Trans. Graph.* 41, 4, Article 41 (jul 2022), 16 pages. https://doi.org/10.1145/3528223.3530176

Stephen A Ehmann and Ming C Lin. 2001. Accurate and fast proximity queries between polyhedra using convex surface decomposition. In *Computer Graphics Forum*, Vol. 20. Wiley Online Library, 500–511.

David Eppstein, Maarten Löffler, and Darren Strash. 2013. Listing all maximal cliques in large sparse real-world graphs. *Journal of Experimental Algorithmics (JEA)* 18 (2013), 3–1.

Rida T Farouki. 1985. Exact offset procedures for simple solids. *Computer Aided Geometric Design* 2, 4 (1985), 257–279.

Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. http://eigen.tuxfamily.org.

Peter Hachenberger. 2009. Exact Minkowksi sums of polyhedra and exact and efficient decomposition of polyhedra into convex pieces. *Algorithmica* 55, 2 (2009), 329–345.

Robert M Haralick, Stanley R Sternberg, and Xinhua Zhuang. 1987. Image analysis using mathematical morphology. *IEEE transactions on pattern analysis and machine intelligence* 4 (1987), 532–550.

Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. 2018. Tetrahedral meshing in the wild. *ACM Trans. Graph.* 37, 4 (2018), 60.

Zhangjin Huang, Yuxin Wen, Zihao Wang, Jinjuan Ren, and Kui Jia. 2022. Surface reconstruction from point clouds: A survey and a benchmark. *arXiv preprint arXiv:2205.02413* (2022).

Alec Jacobson, Daniele Panozzo, et al. 2018. libigl: A simple C++ geometry processing library. https://libigl.github.io/.

Zhongshi Jiang, Teseo Schneider, Denis Zorin, and Daniele Panozzo. 2020. Bijective projection in a shell. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–18.

Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. 2002. Dual contouring of hermite data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. 339–346.

Leif P Kobbelt, Mario Botsch, Ulrich Schwanecke, and Hans-Peter Seidel. 2001. Feature sensitive surface extraction from volume data. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 57–66.

Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. 2019. ABC: A Big CAD Model Dataset For Geometric Deep Learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Min-Ho Kyung, Elisha Sacks, and Victor Milenkovic. 2015. Robust polyhedral Minkowski sums with GPU implementation. *Computer-Aided Design* 67 (2015), 48–57.

Yuen-Shan Leung, Charlie CL Wang, and Yong Chen. 2013. GPU-Based Super-union for Minkowski Sum. *Computer-Aided Design & Applications* 10, 3 (2013).

Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, et al. 2000. The digital Michelangelo project: 3D scanning of large statues. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. 131–144.

Wei Li and Sara McMains. 2010. A GPU-based voxelization approach to 3D Minkowski sum computation. In *Proceedings of the 14th ACM symposium on solid and physical modeling*. 31–40.

Wei Li and Sara McMains. 2011. Voxelized Minkowski sum computation on the GPU with robust culling. *Computer-Aided Design* 43, 10 (2011), 1270–1283.

Wei Li and Sara McMains. 2014. A sweep and translate algorithm for computing voxelized 3D Minkowski sums on the GPU. *Computer-Aided Design* 46 (2014), 90–100.

Jyh-Ming Lien. 2008. Covering Minkowski sum boundary using points with applications. *Computer Aided Geometric Design* 25, 8 (2008), 652–666.

Shengjun Liu and Charlie CL Wang. 2009. Duplex fitting of zero-level and offset surfaces. *Computer-Aided Design* 41, 4 (2009), 268–281.

Shengjun Liu and Charlie CL Wang. 2010. Fast intersection-free offset surface generation from freeform models with triangular meshes. *IEEE Transactions on Automation Science and Engineering* 8, 2 (2010), 347–360.

William E Lorensen and Harvey E Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. *ACM SIGGRAPH Computer Graphics* 21, 4 (1987), 163–169.

Jonas Martinez, Samuel Hornus, Frédéric Claux, and Sylvain Lefebvre. 2015. Chained segment offsetting for ray-based solid representations. *Computers & Graphics* 46 (2015), 36–47.

Wenlong Meng, Shuangmin Chen, Zhenyu Shu, Shi-Qing Xin, Hongbo Fu, and Changhe Tu. 2018. Efficiently computing feature-aligned and high-quality polygonal offset surfaces. *Computers & Graphics* 70 (2018), 62–70.

Jai P Menon and Herbert B Voelcker. 1995. On the completeness and conversion of ray representations of arbitrary solids. In *Proceedings of the third ACM symposium on Solid modeling and applications*. 175–286.

Victor Milenkovic, Elisha Sacks, and Steven Trac. 2013. Robust free space computation for curved planar bodies. *IEEE transactions on automation science and engineering* 10, 4 (2013), 875–883.

Fakir S. Nooruddin and Greg Turk. 2003. Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics* 9, 2 (2003), 191–205.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 165–174.

Darko Pavić and Leif Kobbelt. 2008. High-resolution volumetric computation of offset surfaces with feature preservation. In *Computer Graphics Forum*, Vol. 27. Wiley Online Library, 165–174.

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

Binh Pham. 1992. Offset curves and surfaces: a brief survey. *Computer-Aided Design* 24, 4 (1992), 223–229.

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

Huamin Qu, Nan Zhang, Ran Shao, Arie Kaufman, and Klaus Mueller. 2004. Feature preserving distance fields. In *2004 IEEE Symposium on Volume Visualization and Graphics*. IEEE, 39–46.

Elisha Sacks, Victor Milenkovic, and Min-Ho Kyung. 2011. Controlled linear perturbation. *Computer-Aided Design* 43, 10 (2011), 1250–1257.

Scott Schaefer and Joe Warren. 2004. Dual marching cubes: Primal contouring of dual grids. In *12th Pacific Conference on Computer Graphics and Applications, 2004. PG 2004. Proceedings.* IEEE, 70–76.

Silvia Sellán, Christopher Batty, and Oded Stein. 2023. Reach For the Spheres: Tangency-aware surface reconstruction of SDFs. In *SIGGRAPH Asia 2023 Conference Papers*. 1–11.

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

Nicholas Sharp and Alec Jacobson. 2022. Spelunking the deep: Guaranteed queries on general neural implicit surfaces via range analysis. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–16.

Chen Shen, James F O'Brien, and Jonathan R Shewchuk. 2004. Interpolating and approximating implicit surfaces from polygon soup. In *ACM SIGGRAPH 2004 Papers*. 896–904.

Vijai Kumar Suriyababu, Cornelis Vuik, and Matthias Möller. 2023. Towards a High Quality Shrink Wrap Mesh Generation Algorithm Using Mathematical Morphology. *Computer-Aided Design* 164 (2023), 103608.

Matthias Teschner, Stefan Kimmerle, Bruno Heidelberger, Gabriel Zachmann, Laks Raghupathi, Arnulph Fuhrmann, M-P Cani, François Faure, Nadia Magnenat-Thalmann, Wolfgang Strasser, et al. 2005. Collision detection for deformable objects. In *Computer graphics forum*, Vol. 24. Wiley Online Library, 61–81.

The CGAL Project. 2024. *CGAL User and Reference Manual* (5.6.1 ed.). CGAL Editorial Board. https://doc.cgal.org/5.6.1/Manual/packages.html

Tim Van Hook. 1986. Real-time shaded NC milling display. *ACM SIGGRAPH Computer Graphics* 20, 4 (1986), 15–20.

Gokul Varadhan, Shankar Krishnan, TVN Sriram, and Dinesh Manocha. 2004. Topology preserving surface extraction using adaptive subdivision. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. 235–244.

Gokul Varadhan and Dinesh Manocha. 2004. Accurate Minkowski sum approximation of polyhedral models. In *12th Pacific Conference on Computer Graphics and Applications, 2004. PG 2004. Proceedings.* IEEE, 392–401.

Bolun Wang, Teseo Schneider, Yixin Hu, Marco Attene, and Daniele Panozzo. 2020. Exact and efficient polyhedral envelope containment check. *ACM Trans. Graph.* 39, 4 (2020), 114.

Charlie CL Wang and Dinesh Manocha. 2013. GPU-based offset surface computation using point samples. *Computer-Aided Design* 45, 2 (2013), 321–330.

Jason Williams and Jarek Rossignac. 2005. Mason: morphological simplification. *Graphical Models* 67, 4 (2005), 285–303.

Shiqing Xin, Pengfei Wang, Rui Xu, Dongming Yan, Shuangmin Chen, Wenping Wang, Caiming Zhang, and Changhe Tu. 2022. SurfaceVoronoi: Efficiently Computing Voronoi Diagrams Over Mesh Surfaces with Arbitrary Distance Solvers. *ACM Transactions on Graphics (TOG)* 41, 6 (2022), 1–12.

Lulu Zhang, Tao Luo, Yaoyu Zhang, Zhi-Qin John Xu, Zheng Ma, et al. 2021. MOD-Net: A machine learning approach via model-operator-data network for solving PDEs. *arXiv preprint arXiv:2107.03673* (2021).

Qingnan Zhou and Alec Jacobson. 2016. Thingi10k: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797* (2016).

Daniel Zint, Nissim Maruani, Mael Rouxel-Labbé, and Pierre Alliez. 2023. Feature-Preserving Offset Mesh Generation from Topology-Adapted Octrees. In *Computer Graphics Forum*. 12.

Chen Zong, Jiacheng Xu, Jiantao Song, Shuangmin Chen, Shiqing Xin, Wenping Wang, and Changhe Tu. 2023. P2M: a fast solver for querying distance from point to mesh surface. *ACM Transactions on Graphics (TOG)* 42, 4 (2023), 1–13.