Feature-Preserving Offset Mesh Generation from Topology-Adapted Octrees

Daniel Zint, Nissim Maruani, Mael Rouxel-Labbé, Pierre Alliez

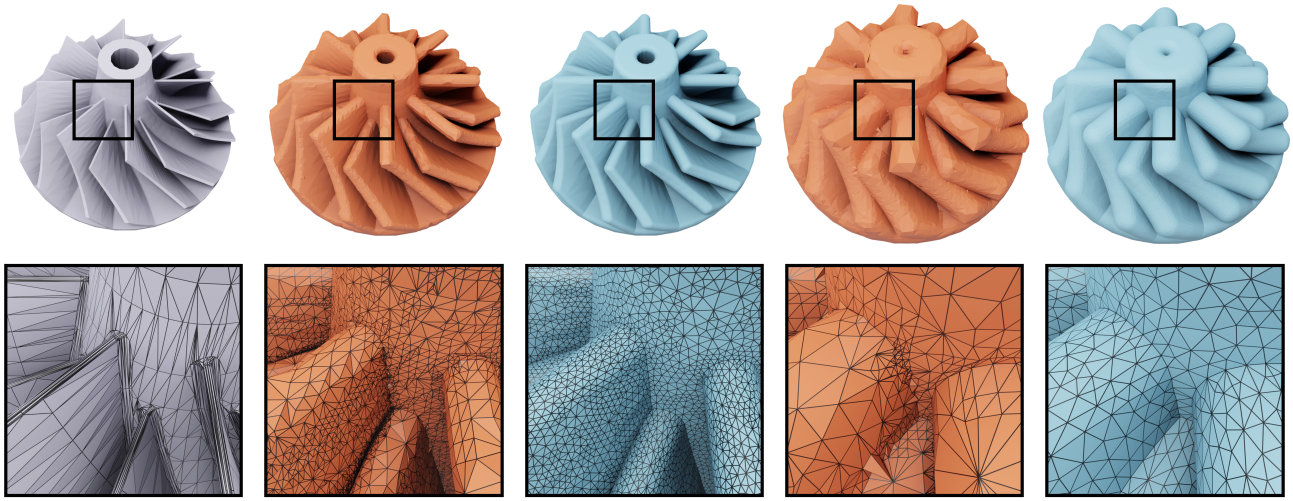
Eurographics Symposium on Geometry Processing 2023
P. Memari and J. Solomon
(Guest Editors)

Volume 42 (2023), Number 5

Feature-Preserving Offset Mesh Generation from Topology-Adapted Octrees

D. Zint[1,3], N. Maruani[1], P. Alliez[1], and M. Rouxel-Labbé[2]

[1]Inria center at Université Côte d'Azur, France
[2]GeometryFactory, France
[3]New York University, USA



(a) Input shape (b) Dual Contouring (2%) (c) Offset (2%) (d) Dual Contouring (5%) (e) Offset (5%)

Figure 1: Method overview: (a) The input consists of a mesh and two main user-defined parameters: offset radius and maximum normal deviation; (b,d) The topology of the initial offset mesh is computed with Dual Contouring on a topology-adapted octree; (c,e) A high-quality offset mesh is obtained through remeshing.

Abstract

We introduce a reliable method to generate offset meshes from input triangle meshes or triangle soups. Our method proceeds in two steps. The first step performs a Dual Contouring method on the offset surface, operating on an adaptive octree that is refined in areas where the offset topology is complex. Our approach substantially reduces memory consumption and runtime compared to isosurfacing methods operating on uniform grids. The second step improves the output Dual Contouring mesh with an offset-aware remeshing algorithm to reduce the normal deviation between the mesh facets and the exact offset. This remeshing process reconstructs concave sharp features and approximates smooth shapes in convex areas up to a user-defined precision. We show the effectiveness and versatility of our method by applying it to a wide range of input meshes. We also benchmark our method on the Thingi10k dataset: watertight and topologically 2-manifold offset meshes are obtained for 100% of the cases.

CCS Concepts

• ***Computing methodologies*** → ***Shape modeling;***

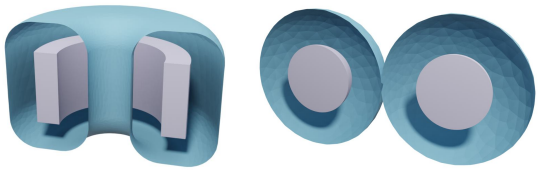


Figure 2: Cut-view of simple input meshes (gray) with complex offset surfaces (blue). The inside tube of the pipe (left) can be arbitrarily small as the offset radius grows. The offset surface of two spheres (right) is the union of two larger spheres that can share a non-manifold point.

1. Introduction

Rolling a ball of fixed radius over a surface describes a volume whose boundary is referred to as the offset surface. More formally, this operation is known as the Minkowski sum of the surface and the ball. The offset surface can also be defined as the level-set of the unsigned distance field to the input. The construction of offset surfaces is a fundamental tool in many applications such as computer-aided design (CAD), collision detection, path planning, boundary layer mesh generation, architectural design, design exploration, etc.

Despite being a simple, well-defined mathematical operation, the reliable generation of discrete offset surfaces is a notoriously difficult scientific challenge, as even simple inputs can lead to complex offset surfaces (see Figure 2). This complexity has caused offset surface computation to be studied mostly in the context of discrete inputs and outputs. Using discrete inputs and outputs is nevertheless reasonable, given the discrete nature of most real-world data. In addition, a discrete approximation of the offset that lies within a user-defined error bound is often sufficient, or even required.

1.1. Focus and Problem Statement

This paper addresses the problem of generating a discrete offset in the form of a surface triangle mesh, given a surface mesh as input, with two main user-defined parameters: an offset radius δ and a maximum normal deviation in degrees $\sigma_{\max}$. Minimum edge length and maximum octree depth are optional parameters that provide additional control to the user. The maximum normal deviation $\sigma_{\max}$ provides a means to control the approximation error. The minimum edge length and the maximum octree depth limit the mesh complexity. Our main focus is to guarantee a closed, combinatorially 2-manifold output surface triangle mesh through a process that is robust to input defects such as self-intersections or holes. We also aim to generate offset meshes with well-shaped (isotropic) triangles everywhere, except near sharp features subtending small angles.

1.2. Related Work

The literature abounds with the generation of offset meshes as it is a complex, multifaceted problem. We observe no consensus on the best methodological approach due to multiple dilemmas: exact vs. inexact, over-refinement followed by filtering vs. coarse-to-fine, etc. As such, some approaches aim at constructing data structures in relation to the structure of the distance function in space (generalized 3D Voronoi diagrams), while others aim at understanding the structure of the exact local offset. Some approaches reformulate the problem as the approximation of Minkowski sums, and many approaches utilize volumetric discretizations.

Generalized 3D Voronoi Diagram. The distance field $d(\mathbf{x})$ of a triangle mesh is C_∞ continuous almost everywhere but is only C_0 in some areas. More specifically, the distance field is only C_0 whenever the nearest primitive (vertex, edge, face) changes. For example, when we move in the proximity of a concave edge, the gradient of the distance field jumps at some point. The nearest primitive to each point in space is encoded by the generalized 3D Voronoi diagram. The distance field is C_∞ inside Voronoi cells, while it is only C_0 on its bisectors. Offset generation can thus be seen as a sub-problem of generating the generalized 3D Voronoi diagram. If we had the exact 3D diagram, we could simply compute the offset for each cell, a simple task due to the C_∞ continuity, and then cut the offsets at the bisectors. However, contrary to 2D where generalized Voronoi diagrams are well studied, with available implementations [Kar22], the reliable generation of such a diagram in 3D is still open and only partial or approximate methods exist. In 3D, Boada et al. approximate the generalized Voronoi diagram using a voxel grid [BCMA08]. Hemmer et al. compute the exact 3D Voronoi diagram but only for lines [HSH10]. The complexity of computing the generalized 3D Voronoi diagram in Euclidean space is well described by Yap et al. [YSL12]. To our knowledge, there is no method that can compute either the exact diagram or an approximation with the precision required by our problem.

Exact local offsets. Instead of using the complete generalized 3D Voronoi diagram, Aubry et al. use its local subset, namely the generalized spherical Voronoi diagrams around vertices, to compute the offset [ADMK17]. This elegant approach makes it possible to compute the exact topology of the offset around a vertex (with the crucial requirement of computing the exact generalized Voronoi diagram on the sphere) and therefore to create an offset mesh with arbitrary precision. However, this method does not yet extend to global intersections, i.e. when two non-adjacent elements are closer than twice the offset distance.

Minkowski sum. Varadhan and Manocha approximate the Minkowski sum of polyhedral models by computing distance fields and reconstructing the surface with a variant of Marching Cubes [VM04]. Campen and Kobbelt proposed a powerful method to compute intersections of polygonal meshes and leverage it to generate approximate Minkowski sums between the input mesh and a sphere [CK10a; CK10b]. A polygonal surface mesh approximates the sphere swept over the input mesh. For the context of manufacturing, approximating this sphere by a zonotope is often sufficient. Martinez et al. use this property to provide an efficient method for approximating offset surfaces [MHCL15]. However, when aiming for high accuracy, especially in concave areas, the sphere must be discretized with a dense mesh.

Volumetric discretizations. To avoid the difficulty of constructing exact offset surfaces, most methods shift their objective towards

approximating the offset. Marching Cubes [LC87] and Dual Contouring [JLSW02] are both widely applicable isosurfacing methods that have been constantly improved over the past decades. Qu et al. use an approach similar to Schaefer and Warren's Dual Marching Cubes that is based on a structured irregular grid [QZS*04]. While this approach can represent sharp features, it is based on a dense voxel grid, which limits its practicality. Liu and Wang successfully use a modified version of Dual Contouring to generate intersection-free offset surfaces [LW10]. This method is also based on a dense voxel grid. Pavić and Kobbelt use an octree to avoid the overhead caused by voxel grids [PK08]. However, the octree is subdivided to the lowest level whenever cells are non-empty, causing milder but similar types of over-refinement. Features are reconstructed in a post-processing step by setting a threshold on the normals and adding features by subdividing faces and edges. Outliers and low-quality elements are removed by further post-processing. Although the aforementioned methods are reliable and robust, the uniform grid resolution (or maximal octree depth) aspect hampers scalability and generates transient over-refined meshes.

GPU-based and sampling methods. Whenever a method relies on structured data, GPUs can reduce runtime significantly by performing several thousands of operations in parallel. Wang and Manocha use Layered Depth Images on GPUs to sample the surface of the input mesh [WM13]. This method can deliver more than 100 times speedup in comparison to related CPU methods. Chen et al. compute offset surfaces with an efficient dexel data structure that stores an array of cells containing a list of intersections with rays [CPD19]. While these methods solve the runtime issues that many volumetric approaches have, they still generate meshes with large complexity. Meng et al. distribute sample points uniformly across the offset [MCS*18]. This method requires dense sampling to ensure the correct representation of concave creases.

Meshing and Remeshing. In many cases, the raw output of the methods described above requires post-processing to treat issues like self-intersections or low-quality elements. Additionally, remeshing can be used to recover features and reduce the discretization error. One such technique is the isotropic remeshing introduced by Botsch and Kobbelt, which focuses on element quality [BK04]. While this method succeeds in its objective, it is not sufficient in our context as the method assumes that the underlying surface is continuously differentiable, i.e. it does not contain sharp features, which is not true for offsets. Shen et al. approximate implicit surfaces in [SOS04] but their method also smoothes away small features. The remeshing approach contributed by Pavić and Kobbelt in [PK08] first detects feature lines in the previously generated offset mesh by setting a threshold on the normal deviation within a triangle and reconstructs them by adding vertices along the feature line and flipping edges to reconstruct the features. Smoothing is applied to improve element quality and reduce the discretization error.

2. Method Overview

Given an input triangle surface mesh $\mathcal{T}$ and an offset radius δ, the offset surface $\mathcal{S}(\mathcal{T},\delta)$ is defined as the isosurface $d(\mathbf{x},\mathcal{T}) = \delta$ where $d(\mathbf{x},\mathcal{T})$ is the distance field of $\mathcal{T}$. If the input is a closed mesh, the distance field of $\mathcal{T}$ can be signed, and the offset is composed of multiple closed surfaces. We denote by $\mathcal{V}(\mathcal{T},\delta)$ the volume bounded by $\mathcal{S}(\mathcal{T},\delta)$.

Our approach consists of two distinct yet complementary steps, depicted in Figure 1:

Octree construction and Dual Contouring: This step is designed to generate an initial offset mesh with the desired topology: closed, combinatorially 2-manifold, and approximately (up to a user-defined tolerance) equivalent to the offset surface. The octree is constructed using several topological criteria to drive refinement. The geometry of the generated mesh is still crude with, e.g., no particular care given to sharp features. See Section 4.

Remeshing: This step applies a novel remeshing algorithm tailored to offset surfaces to improve the geometric precision while keeping the topology unchanged. See Section 5.

3. Positioning and Contributions

Our approach builds upon several existing methods related to offset mesh generation and sharp feature recovery.

Adapted Octrees. At first glance, the method of Pavić and Kobbelt [PK08] is closest to ours: it is based on a similar two-phase approach. Its first phase is also based on an octree, but the octree is refined to its maximum depth everywhere around the offset surface. This has a number of disadvantages. First, it causes a significant computational overhead because one subdivides cells even in topologically and geometrically simple regions, see Section 6.2.1. It also means that the maximum depth is limited as the octree becomes exceedingly large. Furthermore, the user must select a maximum octree depth. While the relationship to the approximation error is clear, the one to the topological errors is not intuitive. Finally, such a dense refinement translates into a complex intermediate mesh, which slows down further processing steps such as remeshing.

Varadhan et al. make use of a topology-adapted octree for computing Minkowski sums [VM04; VKSM04]. The offset needs to be approximated by sweeping a discretized sphere over the input mesh. The offset quality depends on the sphere discretization and the computation of pairwise-convex Minkowski sums is computationally expensive.

Our first main contribution is an octree generation that estimates the offset conservatively at all times and is therefore guaranteed to never miss the exact offset, up to floating point precision. Additionally, our conservative cell-to-triangle distance approximation is computationally lighter than the methods presented in [VM04] and [PK08], and does not require the input to be closed or manifold, e.g. see Figure 3. Our topology-adapted octree reduces computational overhead and output complexity by performing fewer subdivisions in regions with disk topology. In our framework, the maximum depth parameter is not as much a limiting factor as in other methods because the maximum depth can be set to a large value without hampering runtime when such a maximum depth is not utilized.

(a) Bunny triangle soup (b) 0.5% offset (c) 5% offset

Figure 3: Our method handles triangles soups well and guarantees that the output is closed and manifold, even in complex scenarios.

The intermediate offset mesh is reconstructed by Dual Contouring, which generates closed meshes by design. By post-processing this output, we also guarantee that the final offset mesh is not just closed but also topologically 2-manifold everywhere.

Remeshing. Meshes generated by Dual Contouring may contain low-quality or tangled elements. Our second main contribution reduces this limitation and improves on existing remeshing algorithms. More specifically, we contribute a novel remeshing algorithm tailored to offset surfaces, which generates elements with overall good quality while ensuring that the offset mesh still approximates the offset well. The remeshing algorithm departs from earlier approaches in two ways: (1) It is driven by normal deviation instead of by the common edge length or approximation error, and (2) Its vertex relocation operator utilizes the quadric error metric (QEM) initially devised for mesh decimation [GH97]. The resulting remeshing algorithm is feature-aware and recovers a faithful discretization of concave creases and corners. It also yields high-quality reconstructions of challenging geometric features such as thin tunnels. Finally, it is independent of the octree and can also be applied as a post-processing step to other offset mesh generation methods such as [PRH*22] or [CK10b].

QEM-based remeshing has been explored before, for example by Valette et al. [VCP08]. This is however a fundamentally different remeshing approach: surfaces are sampled and sampling points are regrouped into clusters. Starting from those clusters, a Centroidal Voronoi Diagram is generated by minimizing an energy term. Error quadrics are used to relocate centers of Voronoi cells, which has the effect of snapping vertices to feature lines or corners. Although effective, this method cannot be applied to offset surfaces due to its implicit nature. If we were to approximate the offset with a mesh, we could not guarantee that this mesh does not contain any tangling that cannot be resolved by remeshing. In addition, the resulting mesh might contain non-manifold edges or vertices. Finally, in the case of Valette et al., the method generates a user-defined number of vertices. While this is an important feature for surface decimation, when generating offset meshes the complexity of the output is often unknown a priori.

4. Octree Construction and Dual Contouring

The octree construction begins with its initialization by a single cell made large enough to contain the input mesh and its offset. We utilize the offset surface $\mathcal{S}(t_i, \delta)$ and the enclosed volume $\mathcal{V}(t_i, \delta)$ of individual triangles of $\mathcal{T} = \{t_i\}$. Each octree cell is subdivided recursively until one of the following criteria is met:

1. No triangle offset $\mathcal{V}(t_i, \delta)$ intersects the cell;
2. The cell is fully enclosed inside the offset surface of a triangle $\mathcal{V}(t_i, \delta)$;
3. The offset surface $\mathcal{S}(\mathcal{T}, \delta)$ is topologically equivalent to a disk within the cell;
4. A user-defined maximum octree depth is reached.

The first two criteria are devised to stop the refinement of cells that do not intersect the offset. They are regrouped in the so-called *Intersection criterion*, described in Section 4.1. The third criterion aims to stop the refinement of cells that are intersected by the offset, but where subdivision does not add any relevant topological information. We call it the *Disk criterion*, Section 4.2. Finally, we might have to subdivide cells to circumvent the known issue that Dual Contouring may generate non-manifold vertices and edges. This step is detailed in Section 4.3. All criteria are evaluated conservatively, meaning that we might over-refine but never terminate too early.

Checking these criteria with all triangles at all cells would be prohibitive. Instead, we store for each octree cell all triangles such that $\mathcal{V}(t_i, \delta)$ intersects the circumscribing sphere of the cell, similarly to Pavić and Kobbelt [PK08]. Our initial cell thus contains all input triangles and each new subdividing cell needs only find those that are relevant within the triangles of its parent cell.

4.1. Intersection Criterion

Octree cells are only of interest if the offset surface intersects them. However, checking for the exact intersection of a cell with all triangle offsets is costly. We accelerate this test by approximating the cell with its enclosing sphere. To check whether the offset of a triangle t_i intersects this sphere, we find the nearest point $\tilde{p}_i$

(a) Input (b) 5% offset (c) -5% offset

Figure 4: Closed meshes like *caesar* can have positive or negative offsets.

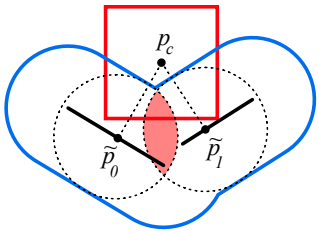


Figure 5: We project the center of the cell (red) on each input triangle (black). If the spheres with offset radius centered at the projection points have a non-empty intersection (light red), the union of triangle offsets is guaranteed to be a star domain (blue).

of the cell center p_c on the triangle. Next, we compute the distance $d_i = \|\tilde{p}_i - p_c\|$ to that point. The sphere intersects the triangle offset if the difference between the absolute offset radius $|\delta|$ and the distance is at most the radius r of the sphere,

$$d_i - r < |\delta| < d_i + r. \tag{1}$$

Triangles whose offset do not intersect the sphere are culled from the cell's triangle list.

As cells are divided, some cells become fully enclosed within the offset of a triangle,

$$d_i + r < |\delta|. \tag{2}$$

In this case, the whole triangle list is discarded and the cell is removed from the subdivision queue.

So far, the intersection criterion only considers the unsigned offset radius. When the input mesh is closed, we use the signed offset by checking whether the cell contains the correct offset sign for any of its primitives,

$$d_{s,i} - r < \delta < d_{s,i} + r. \tag{3}$$

We utilize the signed distance $d_{s,i}$ with a negative sign when the cell center p_c is on the bounded side of a closed mesh. The sign is determined with *Side_of_triangle_mesh* from the CGAL component Polygon Mesh Processing [LRTY23]. A negative offset radius corresponds to offsetting towards the inside, see Figure 4. A cell in which no triangle t_i satisfies Equation (3) is not subdivided any further, as it does not contain the offset of interest.

4.2. Disk Criterion

An offset tends to have simple topology in most areas but it can be very complex in areas where multiple offset sheets touch. Capturing the topology then requires a high level of octree refinement. As we wish to preserve the topology of the exact offset surface, we apply a criterion to capture topological features: A cell in which the offset surface is topologically equivalent to a disk is no longer subdivided. To evaluate this predicate, we construct a sphere with radius δ for every projection point $\tilde{p}_i$ from Section 4.1. If the set of spheres has a non-empty intersection, the union of triangle offset surfaces $\bigcup_{t_i \in \mathcal{T}} \mathcal{V}(t_i, \delta)$ form a star domain, as the offset of a triangle is always convex, see Figure 5. Any star domain is simply connected and therefore of genus 0. It follows that any closed and manifold subset has disk topology.

A non-empty intersection exists if there is a point with a distance to all primitives smaller than the offset. Such a point is found by constructing the minimal sphere that contains all projection points. If this sphere has a radius smaller than the offset, then an intersection exists. We also perform a cheap test by checking if all triangles share a vertex, which trivially guarantees the existence of a non-empty intersection.

We need to ensure that the above genus 0 object can also be reconstructed with Dual Contouring. Therefore, if all cell corners are located on the same side of the offset, the cell must be further subdivided.

While we can guarantee that the union of triangle offset surfaces forms a star domain, the second part, showing that the intersection of the cell with the star domain is a closed manifold, is more complex. A proof that the intersection of all triangle offset surfaces and also the cell is non-empty would be required. In the case of creases subtending very small angles, Dual Contouring tends to yield the wrong topology, as the second part of the disk criterion is no longer met. We alleviate this issue by subdividing a cell if the offset normals within the cell vary by more than 120 degrees. Nevertheless, when the offset contains very sharp creases, we cannot guarantee the topological correctness of our offset mesh as it may contain small holes or disconnected islands.

Our disk criterion relates to the subdivision criteria introduced by Varadhan and Manocha [VKSM04], but its use and evaluation differ significantly.

4.3. Subdivision for Manifoldness

A known pitfall of the Dual Contouring method is the possible creation of non-manifold edges and vertices. If an occurrence of such a configuration is detected, we apply a subdivision step to the cells involved. If such a subdivision is forbidden because we reach the user-defined maximum octree depth, non-manifoldness is resolved by duplicating vertices that are either non-manifold or incident to an edge that is non-manifold. Alternatively, replacing Dual Contouring by the Manifold Dual Contouring presented in [SJW07] would also be feasible.

5. Offset-Aware Remeshing

We use the common mesh operators used for surface remeshing: edge splits, edge collapses, edge flips, and vertex relocation. Instead of driving the remeshing process by the edge length as common in previous work, we utilize the *normal deviation* between the offset mesh triangles and the exact offset.

Definition 5.1 The *normal deviation* $\sigma(t)$ of a triangle t is the maximal angle between the triangle normal n_t and the offset normal n_o in the offset area this triangle is mapped to:

$$\sigma(t) = \max \angle(n_t, n_o(\mathbf{x})),$$

where $\mathbf{x}$ denotes any location in the triangle offset area.

Normal deviation can be understood as a measure of offset curvature, but with the advantage that normal deviation is still well defined at concave creases, where the offset curvature is discontinuous. Therefore, it can be used to isotropically refine smooth regions, implicitly adapting elements to the curvature, and detect concave creases.

Each operation fulfills a different task in the remeshing. Edge splits increase the number of vertices in regions where the normal deviation is too large. More vertices lead to more flexibility in adapting the mesh to the desired geometry. Edge collapses do the exact opposite, they reduce the mesh complexity in regions with low normal deviation, e.g. in planar regions. Edge flips improve element quality and snap edge to concave creases. Vertex relocation minimizes the normal deviation using QEM. Operations are performed subsequently, in the order they are mentioned above.

5.1. Computing Normal Deviation

Computing the exact normal deviation is time-consuming as it requires the use of optimization for searching the local mapping. As the distance field is only C_0-continuous, the optimization problem is challenging. Instead, we sample the triangles of the offset mesh uniformly and project the sample points onto the input mesh. For each sample point, the offset normal n_t is then constructed by the normalized vector pointing from the projection point $\tilde{p}_s$ to the sample point p_s. The maximum normal deviation of a triangle is computed by taking the maximum angle between the triangle normal and all offset normals at the sample points within the triangle. We use CGAL's AABB tree data structure [ATW23] to project sample points onto the input mesh.

5.2. Edge Split

We perform a batch of edge split operations. The list of candidate edges for splitting is found by searching for triangles with $\sigma(t) > \sigma_{\max}$, where $\sigma_{\max}$ is a user defined maximum normal deviation. The longest edge of such a triangle is considered as a candidate. The new vertex inserted by the edge split operator is located as described in the vertex relocation step, see Section 5.5.

Note that we perform only a single batch of edge split operators, without considering the newly created edges as candidates for recursive splitting. The edge split operator does not guarantee a reduction of normal deviation, but adding more vertices to the mesh adds new degrees of freedom that are later used in the vertex relocation.

Our method contains an optional parameter so that the user can set a minimum length. If an edge is already smaller than the minimal length, the edge will not be refined. This option avoids resolving smaller features that are not of interest to the user.

5.3. Edge Collapse

In areas that contain more vertices than necessary, we perform a series of halfedge collapse operations to reduce the mesh complexity. A vertex is collapsed into another one when the normal deviation in all its incident faces is smaller than the user-defined maximum deviation.

Our remeshing operations were designed under the assumption that the mesh is isotropic. Therefore, we must avoid rapid changes in element size. If we collapse all edges only according to the normal deviation, flat areas would be fully collapsed, and the assumption of isotropy would not hold any longer. We avoid this by only collapsing edges that are smaller than twice the maximum edge length in convex offset regions, $l_{\max} = 2\delta \sin(\sigma_{\max})$.

5.4. Edge Flip

Edge flip operators are used to favor well-shaped (isotropic) triangles. As they should not interfere with the optimization for normal deviation, we introduce the following metric for a triangle:

$$m_q = 2\sqrt{3}A/(l_1^2 + l_2^2 + l_3^2) \tag{4}$$

$$m_\angle = 1 - \angle(n_t, n_o)/90^\circ \tag{5}$$

$$m_{flip} = m_q m_\angle^3, \tag{6}$$

where A denotes the triangle area, $l_i, i \in [1,2,3]$ is the length of each edge incident to the triangle, n_t is the triangle normal, and n_o is the offset normal at the triangle center. Equation (4), also known as the mean-ratio metric, was already used for mesh optimization [Ban98; RL17]. Equation (5) denotes the normalized angle between the triangle and the offset surface normal at the triangle center. Equation (6) combines these two metrics, with a greater preference given to the angle metric. Finally, an edge is flipped if the minimum m_{flip} quality of its two incident triangles is improved. Flips on triangles with $\sigma(t) < \sigma_{\max}$ that causes the normal deviation to exceed the user-defined maximum are prohibited.

5.5. Vertex Relocation

Vertices are relocated using a combination of Laplace smoothing and minimization of error quadrics. This approach is similar to the one presented in [VCP08]. Intuitively, our vertex relocation operator improves the shape of triangles in smooth areas and snaps vertices onto creases or corners if there are any within the umbrella of a vertex.

Given a vertex v at position x, we first compute the center of mass of its neighboring vertices,

$$\mathbf{x}' = \frac{1}{|N(v)|} \sum_{x_n \in N(v)} x_n, \tag{7}$$

where $N(v)$ denotes the set of neighboring vertex locations and $|N(v)|$ denotes the number of neighbors. This step is referred to as Laplace smoothing [Fie88].

We then compute the weighted error quadrics of tangential planes at all sampling point projections onto the offset $\hat{s}$. We use the same sampling as for computing the normal deviation, see Section 5.1, on each triangle incident to v. The tangential planes are weighted by the area of the triangle they belong to. For each point $\hat{s}_i$, the offset normal $\hat{n}_i$ and triangle area a_i are required. The sum of weighted error quadrics is minimized by solving the following linear system of equations:

$$\mathbf{A}\mathbf{x}'' = \mathbf{b} \tag{8}$$

$$\mathbf{A} = \sum_{i=1}^{n_{samples}} a_i \cdot \hat{n}_i \hat{n}_i^\top \tag{9}$$

$$\mathbf{b} = \sum_{i=1}^{n_{samples}} a_i \cdot \hat{n}_i^\top \hat{s}_i. \tag{10}$$

Solving Equation (8) is often not possible because $\mathbf{A}$ does not have full rank, e.g. in planar regions all normals $\hat{n}_i$ are similar. We compute the pseudo-inverse with singular value decomposition, as proposed by Lindstrom [Lin00]:

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^\top \tag{11}$$

$$\mathbf{A}^+ = \mathbf{V}\Sigma^+\mathbf{U}^\top \tag{12}$$

$$\mathbf{x}'' = \mathbf{x}' + \mathbf{A}^+(\mathbf{b} - \mathbf{A}\mathbf{x}'). \tag{13}$$

Using the pseudo-inverse requires a starting position $\mathbf{x}'$, chosen as the previously computed center of mass, Equation (7). This way, the vertices are relocated by the error quadrics minimization only when there is a distinct minimum. If the quadrics minimization yields a line or plane of optimal points, then we choose the one that is closest to $\mathbf{x}'$.

6. Results

We evaluate our method on a wide range of input meshes with varying offset radii. Section 6.1 demonstrates robustness by running our method on the Thingi10k dataset. Section 6.2 offers a more in-depth analysis, discussing the impact of parameters on quality and runtime.

Our method was implemented in C++ and depends on the libraries Eigen [GJ*10] and CGAL [The23]. For measuring runtime, the code was serially executed on a cluster with Intel Xeon Platinum 8268 processors that have a base frequency of 2933 MHz.

6.1. Validation

We generated offsets for all meshes from the Thingi10k dataset [ZJ16]. Unless stated differently, all results presented in this paper use the following default values. We use a maximum octree depth of 10 plus two additional levels for resolving non-manifoldness. The remeshing step performs 10 iterations and uses a maximum normal deviation of $\sigma_{\max} = 7^\circ$. Figure 6 showcases results generated from the Thingi10k dataset.

Our implementation expects non-degenerate triangles as input,

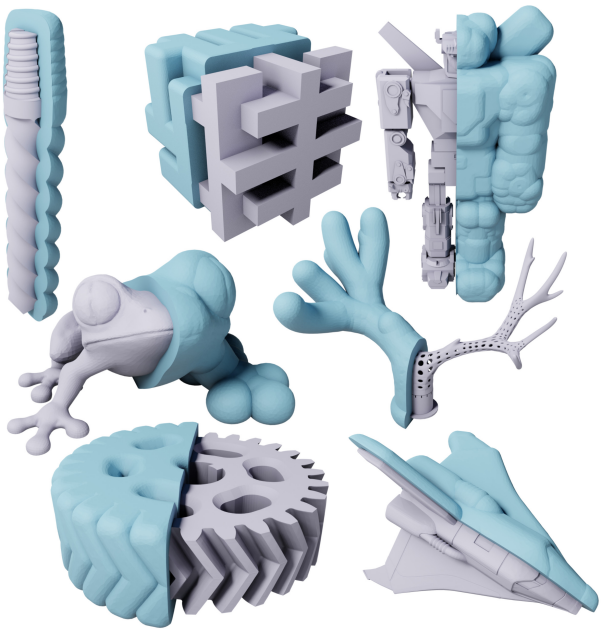
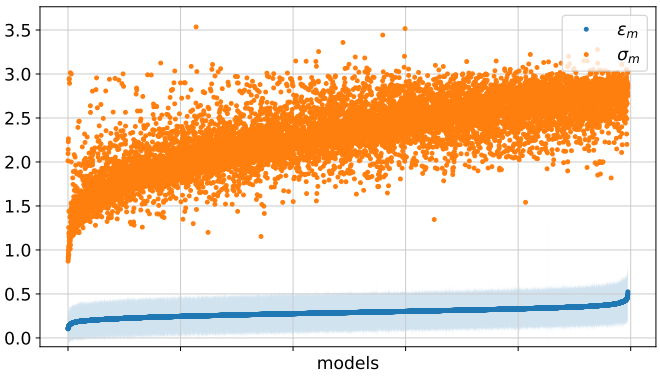
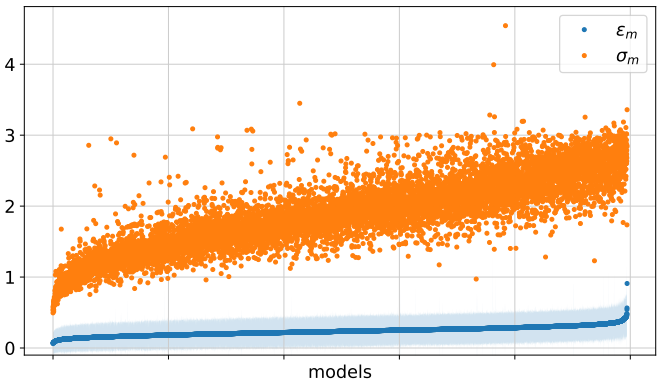


Figure 6: Showcase of our results on Thingi10k.

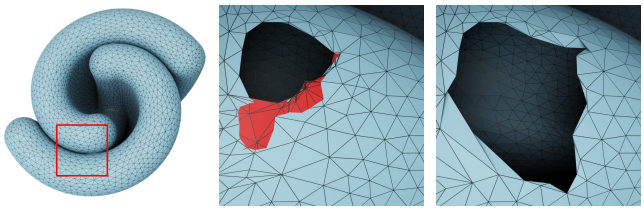


(a) 10% offset



(b) 5% offset

Figure 7: ε_m and σ_m for the models of Thingi10k sorted by ε_m.



(a) Offset mesh (b) self-intersections (c) post-processed

Figure 8: 10% offset on mesh *87687* from the Thingi10k dataset with self-intersections. Back-faces are colored in red. Self-intersections are resolved completely by post-processing.

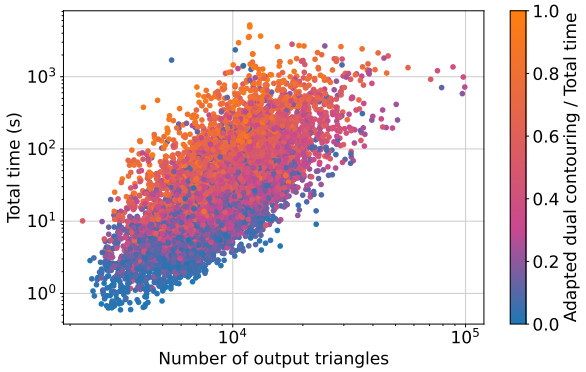


Figure 9: Runtime of our method measured on Thingi10k for $\delta = 10\%$.

but the underlying principles make it feasible to operate on point clouds or meshes with dangling edges. Eventually, we ran our method on 9,952 meshes. All generated offset meshes are closed manifolds. Besides the aforementioned default values, we set a relative offset radius of 10% and 5% with respect to the largest edge of the bounding box, see Figure 7. For evaluation, we densely sample the offset mesh and compute the distance to the offset and the normal deviation using the Libigl geometry processing library [JP*18].

We denote by ε_m the discretization error, measured as the mean distance of a mesh to the offset surface relative to the given offset radius, and denote by σ_m the mean normal deviation. For almost all offset meshes, ε_m is below 0.5%. This means that for a mesh with an offset radius of 0.05, the exact offset is missed by an average of 0.00025. The average maximum distance of all offset meshes relative to the given offset radius is 2.12% and 2.49% for $\delta = 10\%$ and 5% respectively.

Dual Contouring has no guarantees for reconstructing the correct topology of a given implicit surface. This is a fundamental issue of all reconstruction methods based on structured volumetric discretization and is subject to many studies but has not been resolved entirely yet [ZGG22; RSA16; Gro16]. In our case, this means that the topology might be incorrect at concave creases with an angle that is too small to be resolved by the octree. As remeshing expects that the given offset mesh has the correct topology, it may create self-intersections in such areas. For $\delta = 10\%$, self-intersections appear in 2.9%, and for $\delta = 5\%$ in 6.9% of all meshes. The discretization error remains small even in those regions because self-intersecting regions are pushed towards the concave creases. The appearance of self-intersections can be effectively reduced using the CGAL function *remove_self_intersections* from the CGAL component Polygon Mesh Processing [LRTY23] to 0.19% and 0.65% for $\delta = 10\%$ and 5% respectively, without impact on the first three significant digits of σ_m and ε_m. An example is depicted in Figure 8.

We also measure runtimes on Thingi10k, see Figure 9. For $\delta = 10\%$, we observe a correlation between input complexity and runtime for the topology-adapted Dual Contouring. In contrast, remeshing does not show any correlation, not even to the output complexity. The same behavior is observed for $\delta = 5\%$. The mean full runtime of our method is 91 and 148 seconds for $\delta = 10\%$ and 5% respectively. While these values are insufficient to show correlation, they show that runtime tends to increase for smaller offset radii. The increased runtime for $\delta = 5\%$ can be explained by the increasing offset complexity that needs to be captured. A larger offset radius is more likely to seal features like tunnels. The mean runtime is dominated by a few offsets that are compute intensive. Over 70% of all offset meshes are computed in less than 1 and 2 minutes respectively.

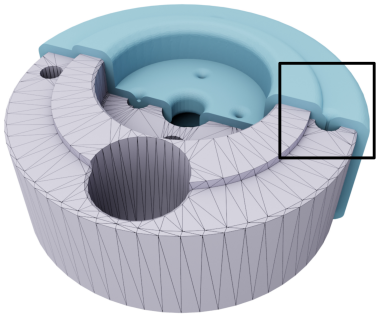
6.2. Analysis

In this section, we give an in-depth analysis of the two main parts, Dual Contouring on the topology-adapted octree, Section 6.2.1, and offset remeshing, Section 6.2.2.

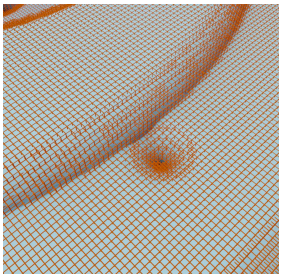
6.2.1. Topology-Adapted Dual Contouring

The major added value of our octree in comparison to those that always refine to the maximum level, e.g. [PK08], is that we avoid over-refinement in topologically simple regions. This is especially important when the offset has small features like narrow tunnels. This fact is exemplified by the *couplingdown* model that contains several drill holes. We generate the offset surface twice, once with the regular octree as described in Section 4 and once with the octree where the maximum level of subdivision is enforced by ignoring the disk criterion defined in Section 4.2. For a relative offset of 3%, an octree depth of 8 is required to capture all the tunnels within the drill holes, see Figure 10. In both scenarios, the topology was reconstructed correctly. However, generating the Dual Contouring mesh requires 34 seconds without our disk criterion but only 11 seconds with it. Additionally, the Dual Contouring mesh is substantially more complex and contains 523,784 triangles, while the disk criterion reduces the complexity to 45,392 triangles. After remeshing, which also takes longer without the criterion, 155 seconds vs. 110 seconds, both meshes have a low ε_m of 0.11% and 0.18%. The meshes then contain 104,724 and 67,840 triangles. Thus, remeshing can reduce the over-refinement caused by the octree very well but the mesh generated using the disk criterion remains less complex.

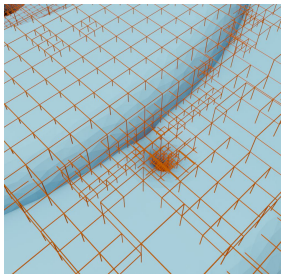
The advantage of the adaptive octree depends on the complexity of the offset topology and may not be that severe all the time. However, by using the adaptive octree, the user does not require a



(a) Input (purple) and offset (blue)



(b) Octree on maximum depth
runtime: 190 seconds
F = 105k
$\varepsilon_m = 0.11\%$
$\sigma_m = 1.07°$



(c) Topology-adapted octree
runtime: 121 seconds
F = 68k
$\varepsilon_m = 0.18\%$
$\sigma_m = 1.31°$

Figure 10: A hole in the input creates an arbitrarily thin tunnel in the offset surface. Our octree is automatically refined to fit small details on the offset.

priori knowledge of the offset topology and does not need to run the method multiple times until the octree depth is large enough to cover all topological features. Thus, this not only improves performance but also simplifies the method for the user.

6.2.2. Remeshing

The remeshing step is designed to reduce the distance and normal deviation from the offset meshes to the exact offset. Figure 11 illustrates remeshing at work on the *anchor* mesh with $\delta = 2\%$ and $\sigma_{max} = 3°$. The discretization error ε_m is effectively reduced and converges after only 6 iterations.

We perform offsetting on the *anchor* model with offset radii δ = 15%, 10%, and 5%. For each radius, we set the maximum normal deviation to $\sigma_{max} = 20°, 10°$, and $5°$. Both, σ_m and ε_m correlate to σ_{max}, independent of the offset radius. The output meshes are depicted in Figure 12.

6.3. Applying Remeshing to Other Offset Methods

Mesh sharpening. Remeshing is completely independent of the octree constructed in Section 4 and can be applied to other methods that generate offset meshes. Alpha wrapping [PRH*22] is such

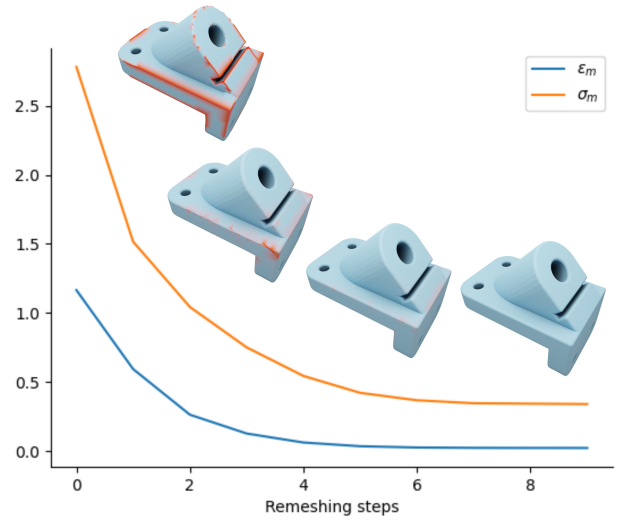
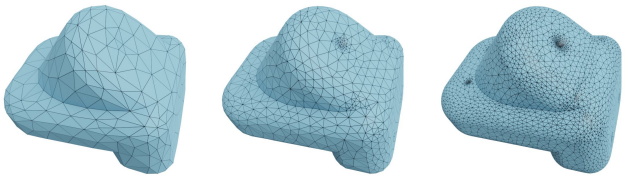


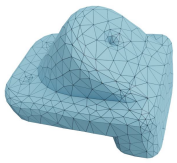
Figure 11: Remeshing reduces both, σ_m and ε_m. Renderings visualize ε_m after 0, 1, 2, and 9 remeshing steps.



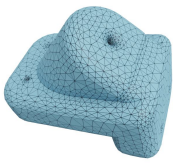
$\varepsilon_m = 1.59\%$
$\sigma_m = 4.86°$

$\varepsilon_m = 0.64\%$
$\sigma_m = 3.24°$

$\varepsilon_m = 0.14\%$
$\sigma_m = 1.59°$

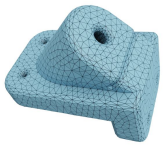


$\varepsilon_m = 1.35\%$
$\sigma_m = 4.18°$



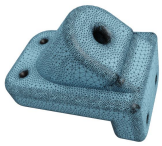
$\varepsilon_m = 0.53\%$
$\sigma_m = 2.75°$

$\varepsilon_m = 0.12\%$
$\sigma_m = 1.34°$



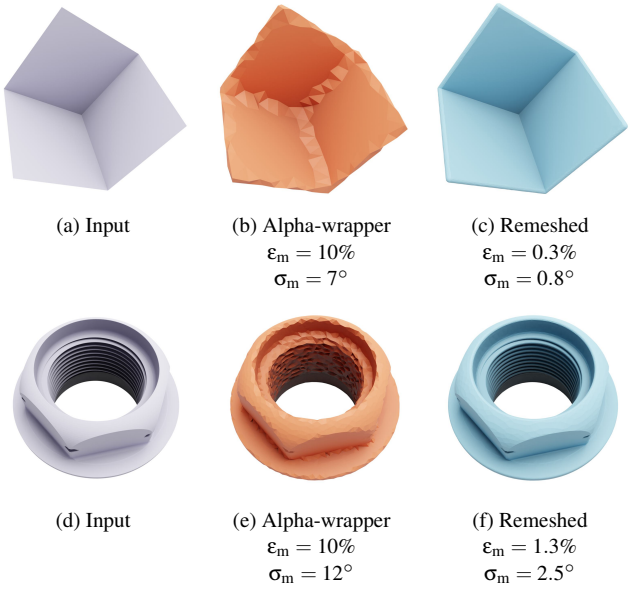
$\varepsilon_m = 0.90\%$
$\sigma_m = 3.11°$

$\varepsilon_m = 0.36\%$
$\sigma_m = 2.05°$



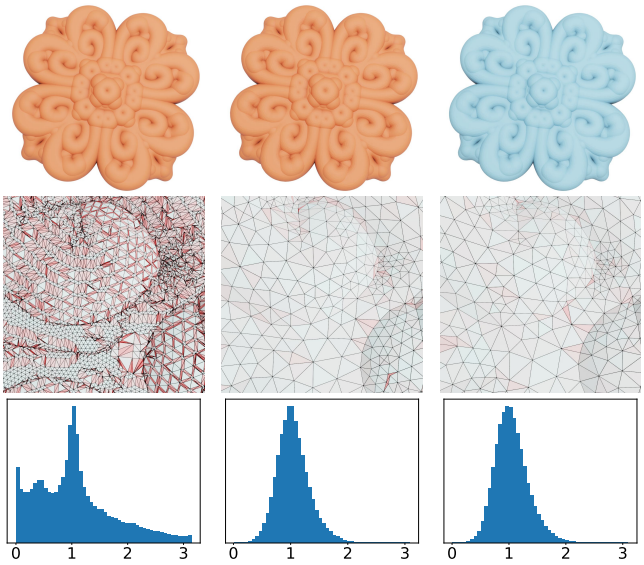
$\varepsilon_m = 0.09\%$
$\sigma_m = 1.02°$

Figure 12: Anchor: $\delta = 15\%, 10\%, 5\%$ (top to bottom), $\sigma_{max} = 20°, 10°, 5°$ (left to right)



(a) Input
(b) Alpha-wrapper $\varepsilon_m = 10\%$ $\sigma_m = 7°$
(c) Remeshed $\varepsilon_m = 0.3\%$ $\sigma_m = 0.8°$
(d) Input
(e) Alpha-wrapper $\varepsilon_m = 10\%$ $\sigma_m = 12°$
(f) Remeshed $\varepsilon_m = 1.3\%$ $\sigma_m = 2.5°$

Figure 13: Remeshing successfully reconstructs concave creases and reduces the discretization error ε_m on outputs of the alpha-wrapper.



(a) MS F = 1051k $\varepsilon_m = 0.34\%$ $\sigma_m = 1.6°$
(b) MS remeshed F = 73k $\varepsilon_m = 0.20\%$ $\sigma_m = 2.1°$
(c) Ours F = 66k $\varepsilon_m = 0.23\%$ $\sigma_m = 2.3°$

Figure 14: Comparison of our method to the approximated Minkowski sum (MS) from [CK10b]. Applying remeshing greatly reduces mesh complexity while keeping ε_m and σ_m low. It also produces better shaped triangles with more consistent angles.

a method: it aims to construct a manifold englobing volume around an input and for this generates an offset mesh with a user-defined precision. Although the method guarantees the absence of intersections with the input mesh, it does not recover sharp features in the output offset surface. Our remeshing step is relevant for post-processing such an output. To avoid over-refinement in convex regions, we set a maximum normal deviation of 15 degrees during remeshing. For the degree-three corner model (Figure 13a) σ_m reduces from 6.5° to 2.6° and ε_m from 14% to 0.7%. A similar result is achieved on the screw nut, Figure 13d, with σ_m reducing from 10° to 2.8° and ε_m from 13% to 0.9%. In this process, the concave creases in the offset are reconstructed, as depicted in Figures 13c and 13f.

Minkowski sums. Offsets can be also computed via Minkowski sums. Such a Minkowski-based approach was used by Campen and Kobbelt [CK10b]. We compare with this approach and show the effectiveness of our remeshing step by applying it to its output meshes, see Figure 14. On the *filigree* model, the Minkowski sum approximation consists of 732,230 triangles for an absolute offset of approximately 0.07. For an absolute offset of 0.035 it consists of 1,051,354 triangles (Figure 14a). In comparison, our method yields 20,090 and 65,528 triangles, respectively (Figure 14c). When applying our remeshing approach to the Minkowski sum approximations, the complexity reduces to 22,892 and 73,772 triangles (Figure 14b). Additionally, triangle quality is significantly improved. While the Minkowski sum approximations contain self-intersections, ours contain none. Our remeshing successfully removes them also from the Minkowski sum approximations.

6.4. Limitations

The proposed approach guarantees that the output meshes are closed and combinatorially 2-manifold, but cannot guarantee the absence of self-intersections that are caused by offset meshes that are topologically different from the exact offset. This is an issue that all methods based on Dual Contouring or Marching Cubes share. In most cases, self-intersections could be resolved with a simple post processing. Self-intersections can be an issue for downstream applications and need to be resolved in the future. The user parameters provide a means to control the complexity of the output meshes, but a strict upper bound on the Hausdorff error is still an open problem. The remeshing step is effective at reconstructing sharp features, but its running time is hard to predict as the number and size of features present in the offset surface are unknown a priori.

7. Conclusion and Future Work

This paper introduced an approach for generating isotropic offset meshes from input triangle meshes. A first step applies the Dual Contouring method on an adaptive octree refined only where the offset topology is complex. A second step operates a novel remeshing method tailored to the offset that reconstructs sharp features and reduces the deviation between the normals of the output mesh and the normals of the offset while improving the shape of the mesh elements. The result is a reliable algorithm that operates even on defect-laden inputs, guarantees closed and combinatorially 2-manifold output meshes, and generates faithful offset meshes

with low approximation errors and well-shaped triangles. The novel offset-aware remeshing approach is also applied with success to offset meshes generated by other approaches.

In future work, we plan to explore a variant that generates anisotropic meshes with improved complexity-distortion trade-offs. We also wish to explore alternatives to the axis-aligned octree data structure, such as unstructured tetrahedral meshes or binary space partitions.

Acknowledgments

This work has been co-funded by GeometryFactory and the national France Relance recovery plan. The work of Pierre Alliez is supported by the French government, through the 3IA Côte d'Azur Investments in the Future project managed by the National Research Agency (ANR) with the reference number ANR-19-P3IA-0002.

References

[ADMK17] AUBRY, R, DEY, S, MESTREAU, EL, and KARAMETE, BK. "Boundary layer mesh generation on arbitrary geometries". *International Journal for Numerical Methods in Engineering* 112.2 (2017), 157–173.

[ATW23] ALLIEZ, PIERRE, TAYEB, STÉPHANE, and WORMSER, CAMILLE. "3D Fast Intersection and Distance Computation". *CGAL User and Reference Manual*. 5.5.2. CGAL Editorial Board, 2023. URL: https://doc.cgal.org/5.5.2/Manual/packages.html#PkgAABBTree.

[Ban98] BANK, RANDOLPH E. *Pltmg: A software package for solving elliptic partial differential Equations: Users' Guide 8.0*. SIAM, 1998.

[BCMA08] BOADA, IMMA, COLL, NARCÍS, MADERN, NARCÍS, and ANTONI SELLARES, J. "Approximations of 2d and 3d generalized voronoi diagrams". *International Journal of Computer Mathematics* 85.7 (2008), 1003–1022.

[BK04] BOTSCH, MARIO and KOBBELT, LEIF. "A remeshing approach to multiresolution modeling". *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. 2004, 185–192.

[CK10a] CAMPEN, MARCEL and KOBBELT, LEIF. "Exact and robust (self-) intersections for polygonal meshes". *Computer Graphics Forum*. Vol. 29. 2. Wiley Online Library. 2010, 397–406.

[CK10b] CAMPEN, MARCEL and KOBBELT, LEIF. "Polygonal boundary evaluation of Minkowski sums and swept volumes". *Computer Graphics Forum*. Vol. 29. 5. Wiley Online Library. 2010, 1613–1622.

[CPD19] CHEN, ZHEN, PANOZZO, DANIELE, and DUMAS, JEREMIE. "Half-space power diagrams and discrete surface offsets". *IEEE Transactions on Visualization and Computer Graphics* 26.10 (2019), 2970–2981.

[Fie88] FIELD, DAVID A. "Laplacian smoothing and Delaunay triangulations". *Communications in applied numerical methods* 4.6 (1988), 709–712.

[GH97] GARLAND, MICHAEL and HECKBERT, PAUL S. "Surface simplification using quadric error metrics". *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. 1997, 209–216.

[GJ*10] GUENNEBAUD, GAËL, JACOB, BENOÎT, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010.

[Gro16] GROSSO, ROBERTO. "Construction of Topologically Correct and Manifold Isosurfaces". *Proceedings of the Symposium on Geometry Processing*. SGP '16. Berlin, Germany: Eurographics Association, 2016, 187–196. URL: https://doi.org/10.1111/cgf.12975.

[HSH10] HEMMER, MICHAEL, SETTER, OPHIR, and HALPERIN, DAN. "Constructing the Exact Voronoi Diagram of Arbitrary Lines in Three-Dimensional Space". *Algorithms – ESA 2010*. Ed. by de BERG, MARK and MEYER, ULRICH. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, 398–409.

[JLSW02] JU, TAO, LOSASSO, FRANK, SCHAEFER, SCOTT, and WARREN, JOE. "Dual contouring of hermite data". *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. 2002, 339–346.

[JP*18] JACOBSON, ALEC, PANOZZO, DANIELE, et al. *libigl: A simple C++ geometry processing library*. https://libigl.github.io/. 2018.

[Kar22] KARAVELAS, MENELAOS. "2D Segment Delaunay Graphs". *CGAL User and Reference Manual*. 5.5.1. CGAL Editorial Board, 2022. URL: https://doc.cgal.org/5.5.1/Manual/packages.html#PkgSegmentDelaunayGraph2.

[LC87] LORENSEN, WILLIAM E and CLINE, HARVEY E. "Marching cubes: A high resolution 3D surface construction algorithm". *ACM siggraph computer graphics* 21.4 (1987), 163–169.

[Lin00] LINDSTROM, PETER. "Out-of-core simplification of large polygonal models". *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. 2000, 259–262.

[LRTY23] LORIOT, SÉBASTIEN, ROUXEL-LABBÉ, MAEL, TOURNOIS, JANE, and YAZ, ILKER O. "Polygon Mesh Processing". *CGAL User and Reference Manual*. 5.5.2. CGAL Editorial Board, 2023. URL: https://doc.cgal.org/5.5.2/Manual/packages.html#PkgPolygonMeshProcessing.

[LW10] LIU, SHENGJUN and WANG, CHARLIE CL. "Fast intersection-free offset surface generation from freeform models with triangular meshes". *IEEE Transactions on Automation Science and Engineering* 8.2 (2010), 347–360.

[MCS*18] MENG, WENLONG, CHEN, SHUANGMIN, SHU, ZHENYU, et al. "Efficiently computing feature-aligned and high-quality polygonal offset surfaces". *Computers & Graphics* 70 (2018), 62–70.

[MHCL15] MARTINEZ, JONAS, HORNUS, SAMUEL, CLAUX, FRÉDÉRIC, and LEFEBVRE, SYLVAIN. "Chained segment offsetting for ray-based solid representations". *Computers & Graphics* 46 (2015), 36–47.

[PK08] PAVIĆ, DARKO and KOBBELT, LEIF. "High-resolution volumetric computation of offset surfaces with feature preservation". *Computer Graphics Forum*. Vol. 27. 2. Wiley Online Library. 2008, 165–174.

[PRH*22] PORTANERI, CÉDRIC, ROUXEL-LABBÉ, MAEL, HEMMER, MICHAEL, et al. "Alpha Wrapping with an Offset". *ACM Trans. Graph.* 41.4 (July 2022). URL: https://doi.org/10.1145/3528223.3530152.

[QZS*04] QU, HUAMIN, ZHANG, NAN, SHAO, RAN, et al. "Feature preserving distance fields". *2004 IEEE Symposium on Volume Visualization and Graphics*. IEEE. 2004, 39–46.

[RL17] RANGARAJAN, RAMSHARAN and LEW, ADRIAN J. "Provably robust directional vertex relaxation for geometric mesh optimization". *SIAM Journal on Scientific Computing* 39.6 (2017), A2438–A2471.

[RSA16] RASHID, TANWEER, SULTANA, SHARMIN, and AUDETTE, MICHEL A. "Watertight and 2-manifold surface meshes using dual contouring with tetrahedral decomposition of grid cubes". *Procedia engineering* 163 (2016), 136–148.

[SJW07] SCHAEFER, SCOTT, JU, TAO, and WARREN, JOE. "Manifold dual contouring". *IEEE Transactions on Visualization and Computer Graphics* 13.3 (2007), 610–619.

[SOS04] SHEN, CHEN, O'BRIEN, JAMES F, and SHEWCHUK, JONATHAN R. "Interpolating and approximating implicit surfaces from polygon soup". *ACM SIGGRAPH 2004 Papers*. 2004, 896–904.

[The23] THE CGAL PROJECT. *CGAL User and Reference Manual*. 5.5.2. CGAL Editorial Board, 2023. URL: https://doc.cgal.org/5.5.2/Manual/packages.html.

[VCP08] VALETTE, SÉBASTIEN, CHASSERY, JEAN MARC, and PROST, RÉMY. "Generic remeshing of 3D triangular meshes with metric-dependent discrete Voronoi diagrams". *IEEE Transactions on Visualization and Computer Graphics* 14.2 (2008), 369–381.

[VKSM04] VARADHAN, GOKUL, KRISHNAN, SHANKAR, SRIRAM, TVN, and MANOCHA, DINESH. "Topology preserving surface extraction using adaptive subdivision". *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. 2004, 235–244.

[VM04] VARADHAN, GOKUL and MANOCHA, DINESH. "Accurate Minkowski sum approximation of polyhedral models". *12th Pacific Conference on Computer Graphics and Applications, 2004. PG 2004. Proceedings.* IEEE. 2004, 392–401.

[WM13] WANG, CHARLIE CL and MANOCHA, DINESH. "GPU-based offset surface computation using point samples". *Computer-Aided Design* 45.2 (2013), 321–330.

[YSL12] YAP, CHEE K, SHARMA, VIKRAM, and LIEN, JYH-MING. "Towards exact numerical Voronoi diagrams". *2012 Ninth International Symposium on Voronoi Diagrams in Science and Engineering*. IEEE. 2012, 2–16.

[ZGG22] ZINT, DANIEL, GROSSO, ROBERTO, and GÜRTLER, PHILIPP. "Resolving Non-Manifoldness on Meshes from Dual Marching Cubes". *Eurographics 2022 - Short Papers*. Ed. by PELECHANO, NURIA and VANDERHAEGHE, DAVID. The Eurographics Association, 2022. ISBN: 978-3-03868-169-4.

[ZJ16] ZHOU, QINGNAN and JACOBSON, ALEC. "Thingi10K: A Dataset of 10,000 3D-Printing Models". *arXiv preprint arXiv:1605.04797* (2016).