Alpha Wrapping with an Offset

Cédric Portaneri, Mael Rouxel-Labbé, Michael Hemmer, David Cohen-Steiner, Pierre Alliez

Alpha Wrapping with an Offset

CÉDRIC PORTANERI, Inria, France
MAEL ROUXEL-LABBÉ, GeometryFactory, France
MICHAEL HEMMER, Independent, Germany
DAVID COHEN-STEINER, Inria, France
PIERRE ALLIEZ, Inria, France

Given an input 3D geometry such as a triangle soup or a point set, we address the problem of generating a watertight and orientable surface triangle mesh that strictly encloses the input. The output mesh is obtained by greedily refining and carving a 3D Delaunay triangulation on an offset surface of the input, while carving with empty balls of radius alpha. The proposed algorithm is controlled via two user-defined parameters: alpha and offset. Alpha controls the size of cavities or holes that cannot be traversed during carving, while offset controls the distance between the vertices of the output mesh and the input. Our algorithm is guaranteed to terminate and to yield a valid and strictly enclosing mesh, even for defect-laden inputs. Genericity is achieved using an abstract interface probing the input, enabling any geometry to be used, provided a few basic geometric queries can be answered. We benchmark the algorithm on large public datasets such as Thingi10k, and compare it to state-of-the-art approaches in terms of robustness, approximation, output complexity, speed, and peak memory consumption. Our implementation is available through the CGAL library.

CCS Concepts: • **Computing methodologies** → **Mesh models**; **Mesh geometry models**; **Volumetric models**.

Additional Key Words and Phrases: Watertight mesh generation, strictly enclosing, outer approximation, offset, wrapping, carving, Alpha-balls, Delaunay-based meshing, 3D Delaunay triangulation, Steiner points.

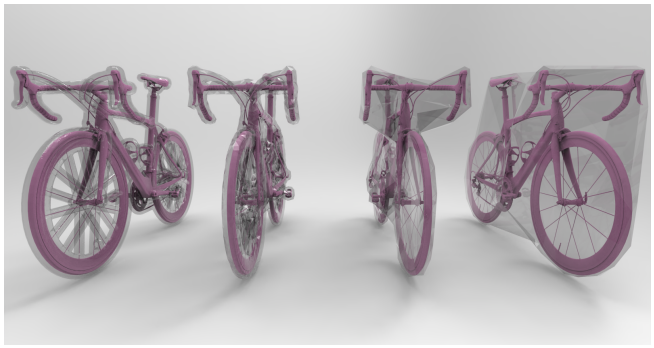


Fig. 1. Wrapping a bicycle. Input mesh (pink) from GrabCAD (530K facets). Varying values of the alpha parameter offer different levels of mesh feature preservation (resp. 1/500, 1/100, 1/20 and 1/5 of the bounding box diagonal length, from left to right). Offset is set to 1/100 for all wrap meshes. Output meshes (transparent rendering) are guaranteed to be watertight, orientable, and strictly enclosing the input with a guaranteed Hausdorff distance to the input. Wrap meshes: 285K facets (generated in 174s), 10.2K facets (3.2s), 564 facets (0.5s) and 142 facets (0.4s).

1 INTRODUCTION

Surface meshes are essential components in the majority of geometry processing and computer graphics applications such as segmentation, remeshing, or simulation. The feasibility of an operation and the quality of the results often depend on the validity and the quality of the mesh. For these reasons, most applications require — or prefer — input meshes that are *valid*, i.e., watertight, combinatorially 2-manifold and orientable, as these properties imply well-defined notions of interior/exterior and geodesic neighborhoods. Despite the crucial role played by meshes, many mesh generation processes (manual design by humans, automated generation from flawed CAD models, reconstruction from measurement data, ...) remain imperfect and can be responsible for a wide variety of possible defects in meshes: duplicates, degeneracies, holes and gaps, self-intersections, non-manifold features or inconsistent orientation. These defects generally compound into critical issues such as inconsistent boundary representations of the object they are supposed to model, significantly hampering further operations.

Repair. In most cases, one does not have access to the upstream process that has produced a mesh with defects, and a possible solution is then to repair the mesh. Unfortunately, reliably repairing all types of defects is difficult, especially as many repair problems are ill–posed, e.g. there may exist many ways to fill holes or to remove self–intersections [Attene et al. 2013]. There are two main approaches to mesh repair: *local repair*, which consists in modifying the input only in the vicinity of each defect, and *global repair*, which consists in generating a new, defect-free mesh that interpolates or approximates the input. On the one hand, local repair is relevant for highly detailed, feature-rich meshes with isolated defects or flaws and is in general faster than global repair since it does not modify the

defect-free parts of the mesh. The literature on local repair is vast, with contributions generally focusing on solving one class of defect at a time, such as degeneracy removal [Botsch and Kobbelt 2001], hole filling [Liepa 2003] or self-intersection removal [Campen and Kobbelt 2010]. Despite these advantages, local repair suffers from an intrinsic flaw: attempting to solve ill–posed problems implies that algorithms seldom achieve unconditional robustness. Moreover, different defects frequently coexist in real-world data, substantially complicating local repair and sometimes making it impossible for defect-laden meshes. On the other hand, global repair approaches often come with guarantees, but at the cost of extra, transient data structures and unnecessary repair in defect-free areas. Global repair approaches share common properties with so-called *approximation* approaches.

Approximation. For many real-world applications, input meshes might sometimes be overly complex, with excessively fine discretization, unnecessary geometric details, and spurious inner topological structures. In addition, the raw input geometry might not even be a mesh but instead a soup of primitives such as triangles, points or line segments. These cases advocate in favor of an approach that does not retain everything from the raw input, but instead constructs an approximation of the raw input geometry. Approximation herein refers to a method capable of filtering out defects, superfluous details, and inner topological structures, producing a valid, watertight mesh. We aim to generate such an approximation, with an emphasis on reliability and robustness.

Enclosing. In addition to validity, we further require that the output strictly encloses the input. The enclosing property is mandatory for applications related to conservative distance queries, collision avoidance, or motion planning [Bergen 1997; Von Dziegielewski et al. 2013; Sellán et al. 2021]. Although a simple geometric construction such as the axis-aligned bounding box, the optimal bounding box [Chang et al. 2011], or the convex hull would match all of our requirements, a larger free configuration space, i.e. a tighter approximation, is often required. Nevertheless, enlarging the free space must be balanced with satisfying safety margins required to ensure that moving solid objects never collide with the environment or surrounding objects. Taking into account these safety margins often translate into discarding unreachable cavities and inner structures from the input, so as to generate an outer approximation.

2 CONTRIBUTIONS

Given a raw input geometry (i.e., a soup of 3D geometric primitives), we address the problem of generating a watertight and orientable surface triangle mesh that strictly encloses the input (see Figure 1). Our method bridges the gap between a number of existing approaches by combining alpha shapes, mesh carving, and Delaunay-based meshing with the purpose of creating an enclosing mesh with adjustable margin. It is also designed to ignore inner structures and to favor well-shaped mesh elements.

Our algorithm offers the following properties:

- Guaranteed to terminate and strictly enclose the input (see Appendix B);
- Unconditional robustness to defects;
- User freedom: adjustable complexity-fidelity tradeoff via two meaningful parameters;
- Output-sensitive complexity: by greedily probing the input, we avoid dealing with superfluous inner structures;
- Compares favorably to state-of-the-art approaches in terms of reliability, robustness and approximation (Section 5);
- Mature software: a novel C++ component integrated into the CGAL library;
- Simple to reproduce;
- Implementation agnostic to the input representation through an abstract interface that probes the input. (Section 4.4).

3 RELATED WORK

The dire need for valid meshes in many applications is unfortunately met with a high frequency of defects in real-world meshes. For example, more than half of the models of the Thingi10K dataset [Zhou and Jacobson 2016] either do not describe manifold surfaces, or contain self-intersections or degeneracies. Consequently, repair and approximation of meshes are topics that have been receiving considerable research interest for decades. We restrict our overview of existing approaches to those that are relevant to our specific setting, which is to devise a global approach yielding a conservative approximation. We group these methods according to their common core underlying idea.

3.1 Alpha Shapes

The alpha shape of a set of points is a generalization of the notion of convex hull [Edelsbrunner and Mücke 1994]. A visual analogy of this structure is to think of the ambient space as ice cream and the input points as chocolate pieces. Carving the ice-cream with a spherical spoon of (squared) radius α without bumping into chocolate pieces, we end up with a (not necessarily convex) object bounded by caps, arcs and points. If we now straighten all "round" faces to triangles and line segments, we have an intuitive description of the so-called alpha shape. For well-chosen values of α, the boundary of the alpha shape is a linear approximation of the shape of the input. This property has made alpha shapes relevant for surface reconstruction [Guo et al. 1997; Teichmann and Capps 1998; Giesen et al. 2006] and mesh repair [Bernardini and Bajaj 1997; Bernardini et al. 1999]. However, the alpha shape structure does not fulfill all of our prerequisites. Firstly, a complete sampling of the input must be performed, followed by the construction of a 3D Delaunay triangulation, which may require heavy and unnecessary computational costs, especially when most of the input data might not even be reached. In addition, the output strongly depends on the sampling quality of the input: small and dense features will be captured, whereas flat but sparsely sampled regions will result in holes on the surface. Moreover, volumetric holes might get carved inside the reconstructed solid, which is both undesired and a superfluous computational cost in our context. Finally, a manifold output is not guaranteed for small α, producing scattered alpha simplicies.

3.2 Sculpting

Similarly to alpha shapes, sculpting techniques are based on the idea of carving the ambient space, but act more conservatively by only allowing carving inward, in a flood-filling manner. Sculpting usually starts from the Delaunay triangulation of the convex hull of an input point set, and iteratively removes cells on the boundary of the complex in a specific order until some termination criteria are met. The carving order can be evaluated locally using tetrahedron geometry with, e.g. the maximum distance between the boundary faces to their associated parts in the circumscribed sphere [Boissonnat 1984]. Other sculpting approaches choose to hinge over a more global ordering criterion such as using the flow complex [Edelsbrunner 2003; Giesen and John 2002]. Termination can be triggered once all remaining tetrahedra have their criterion reach a certain threshold, or if the volume increases upon removal [Boissonnat 1984]. In the convection approach pioneered by Allegre et al. [2005], the algorithm stops when all oriented facets "see" the positive half of their minimum enclosing sphere that is free of any input points. The resulting surface mesh is defined as the boundary of the final tetrahedral complex.

The notion of *alpha solid*, introduced by Bernardini et al. [1997; 1999] is a hybrid approach combining alpha shapes and sculpting to repair CAD models. After an initial sampling, a regularized version of the alpha shapes is first computed, free of isolated and dangling k-simplices. During sculpting, a cell is to be removed if one of its faces is both on the boundary and not a part of the alpha complex, i.e., if the radius of the Delaunay ball of this face is too large and we can pass through it using the α-sized spoon without bumping into its vertices. The result is a pure simplicial complex similar to that of the alpha shape, but in which unreachable k-simplices have been filtered out. Alpha solids improve over each technique: it avoids superfluous carving while benefiting from the same approximation qualities (for the outer hull) as alpha shapes. Unfortunately, the method is burdened with the heavy computational cost of a pre-processed sampling and the small-alpha scattered simplicies produced by the alpha shapes.

3.3 Constrained Polygonalizations

Previous methods assume no knowledge on the connectivity between the input points, but this is an important loss of information when handling triangle soups. A series of methods have been devised using an underlying constrained Delaunay triangulation: a triangulation of the input points is constructed, and the input faces are constrained, meaning that the input triangles are either part of the triangulation, or are represented by unions of triangles. Tetrahedra are then labeled inside or outside to extract a surface from the boundary. Also, such a labeling process is not trivial when the input consists of a soup of triangles that does not represent a closed manifold surface mesh. An elegant solution is to split the Delaunay cells with a binary space partitioning using the input faces planes, resulting in a constrained polygonalization [Hu et al. 2018]. Diazzi and Attene [2021] show that such a partitioning provides a means to represent not only the input triangles but also the input planes, in order to eventually interpolate exactly the input (and its flaws) and close holes. The partitioning approach can be further enhanced by triangulating the polygons improving the mesh quality so as to transform the interpolation into an approximation without error bounds [Hu et al. 2018]. Hu et al. [2020] combine generalized winding numbers [Jacobson et al. 2013] for inside / outside labeling, with a relaxed constrained triangulation to obtain a tetrahedral mesh generator with unprecedented robustness. The above approaches offer unconditional robustness but do not match our strict enclosing requirement. In addition, the transient data structure can be overly complex as it may contain all inner structures and excessively fine discretizations of the input.

3.4 Implicit Surfaces

Level-set (or contouring) meshing approaches have their roots in the field of surface reconstruction, which aims at transforming a point cloud into a mesh. Reconstruction was first achieved using Delaunay triangulations to interpolate the point cloud [Dey 2010], but this bounds the output to the quality of the input. Instead, implicit approaches construct a mesh by contouring an implicit surface which is defined as a level-set of a scalar function which approximates the inferred shape [Berger et al. 2017]. This implicit surface is defined at the vertices of a space-discretizing volumetric data structure around the input data, for example a regular grid [Nooruddin and Turk 2003], an adaptive octree [Kazhdan et al. 2006; Xu and Barbič 2014], a triangulation [Zhao et al. 2021], or a kd-tree [Shen et al. 2004]. The notions of interior and exterior can be defined through a smoothed indicator (from "zero" outside to "one" inside) [Nooruddin and Turk 2003; Ju 2004] or signed distance function [Calakli and Taubin 2011]. The level set of the implicit surface can be meshed via contouring methods (marching-cubes, dual contouring or variants) or by restricted Delaunay meshing [Cheng et al. 2012]. By generating completely new meshes, these approaches have some control over the validity of the result and offer theoretical guarantees. However, inner structures that we wish to ignore are also meshed, and the output mesh does not match our main requirement to enclose the input. Indeed, this is a dealbreaker. In addition, most of these approaches require constructing an underlying structure whose complexity depends on that of the input.

3.5 Shrink Wrapping

Shrink wrapping was spawned from the seminal work from Kobbelt et al. [1999]. It consists in building an initial watertight mesh around the input, before shrinking it through various means. A physical analogy of this process is to wrap a plastic membrane around an object and then vacuuming the air enclosed within the membrane. The initial mesh, referred to as wrapper surface, is often generated by labeling the cells of an adaptive octree inside or outside and then by extracting the mesh from the boundary. Cell labeling is performed via flood filling on the cells that intersect the input [Lee et al. 2010; Huang et al. 2018, 2020] or on the cells in which the dual edges (i.e. the line joining the centers of two adjacent cells) intersect the input [Martineau et al. 2016]. When the size of the hole is larger than the local element size, an adequate resolution to the octree may be enforced using a proximity size function used as a criterion to detect gaps [Lee et al. 2010]. Holes may be filled using morphological operations [Bischoff et al. 2005], or a transient coarser

stage of the octree refinement process can be utilized [Martineau et al. 2016]. The wrapper surface is then shrunk using successive operations such as projecting the vertices onto the input, relocating the vertices on the input via Laplacian smoothing, or editing local triangle facets to improve the mesh quality [Juretić and Putz 2011; Malyshev et al. 2018]. Most of the shrinking processes come with no guarantees, whether on the validity of the final mesh or on its enclosing property.

3.6 Bounding Volumes

From the aforementioned methods, only a few guarantee an upper bound on the approximation error made by their algorithm, and even fewer can guarantee that the output strictly encloses the input. Nonetheless, constructing an enclosing volume (also sometimes referred to as a *cage*) is a topic that has received much attention, especially in the context of mesh deformation. Existing methods similarly often rely on building coarse a mesh of an offset surface defined via a distance field [Shen et al. 2004]. Calderon and Boubekeur [2017] use morphological operations to create a bounding volume, but without guarantees and the output might be self-intersecting. Diazzi and Attene [2021] guarantee an enclosing algorithm as their constrained polyhedralization is exactly interpolating the input, but the algorithm might require many extra faces to produce a result. A number of approaches perform mesh decimation techniques while enforcing the enclosing property during each local decimation operation [Sander et al. 2000; Gumhold et al. 2003], with the obvious downside of starting from a potentially very dense mesh. Recent approaches [2020] explored the use of neural nets to generate such cages, but with little control over the quality of the result. Sacht et al. [2015] propose a way to construct a hierarchy of recursively enclosing volumes without using any distance field, but it is restricted to small offsets and is slow. The approach from Stuart et al. [2013] is closest to ours: starting from an outer shell carved from a regular grid, the vertices are projected on the offset with enclosure guarantees. Empirical comparisons with a number of methods are provided in Section 5.

4 METHOD

We first present an outline of our algorithm, a 2D illustration, as well as the strengths of our approach, before delving into the details.

4.1 Overview

We introduce a simple and generic algorithm which is guaranteed to generate a valid (watertight and orientable) mesh that strictly encloses the input geometry in an arbitrarily tight fashion. The core idea behind our approach is to detach the structure being carved from the input geometry: instead of carving a triangulation whose vertices are points of the input, as it is the case in sculpting methods, we start from an entirely new and coarse enclosing mesh, and interlace carving and refinement steps to create the final approximation. Similarly to previous methods, our carving operates on an underlying 3D Delaunay triangulation whose tetrahedral cells are tagged inside or outside, and the resulting surface mesh is defined as the set of Delaunay facets separating inside from outside Delaunay cells. The carving step trims the mesh inward by tagging outside a Delaunay cell that was formerly tagged inside. The refinement step is triggered when an upcoming carving step would expose the input geometry, i.e., when the tetrahedron cell that is to be tagged outside actually intersects the input. In this configuration, carving is not performed and a Steiner point is instead inserted into the Delaunay triangulation to refine the tetrahedron cell. The Steiner points added during refinement operations do not lie directly on the input but rather on the *offset surface*, which is defined as a level-set of the unsigned distance field to the input. More specifically, a Steiner point is computed either as the intersection of a dual Voronoi edge with the offset surface, or as the projection of the circumcenter of a Delaunay cell onto the offset surface. Intuitively, our method is devised to allocate on-demand new degrees of freedom for carving, in a manner that favors low output complexity and well-shaped elements.

Figure 2 illustrates our algorithm at work in 2D on a soup of line segments characterized by many defects such as intersections, gaps, and non-manifold features. The additional materials contain a screen-capture video (*japan*) with a more complex input.

Our combination of carving and on-demand refinement has several immediate advantages:

- The mesh is only refined when a carving operation would expose the input, thus avoiding superfluous constructions of potentially complex inner structures (constructions that would otherwise be performed in sculpting or level-set meshing approaches);
- By placing new vertices on the offset surface, the algorithm guarantees a watertight and orientable output surface mesh, even for degenerate or open inputs (see Section 5.2);
- Because of the implicit formulation of the offset surface as a level-set of a distance field, our algorithm is agnostic to the representation of the input: any raw geometry (points, segments, triangles, ...) can be used provided that three basic distance and intersection queries can be answered (see Section 4.4). A defect-free input is furthermore not required to answer these geometric queries, e.g. a triangle soup can have self-intersections;
- Since constructing an enclosing approximation on a large offset of the input requires fewer mesh elements (see Section 4.2), users have the freedom to trade tightness to the input for final mesh complexity;

The work of Stuart et al. [2013] is at first glance similar to ours: their algorithm is composed of three steps, (1) superimposing the input on a regular tetrahedral grid, (2) sculpting out grid cells that do not intersect the input, and (3) optimizing the position of the grid vertices to create an as-tight-as-possible enclosing mesh. There are however key differences, as well as strong downsides to their method. Firstly, the output complexity of their method is relatively independent of the chosen offset value: points eligible for movement towards the offset surface are the remaining outer grid vertices after carving. Consequently, if the grid is not sufficiently dense, the offset must be large enough such that a representation of the object is possible with the fixed number of grid vertices, otherwise the method does not converge. On the contrary, because we may introduce extra vertices as needed, our output vertices are always on

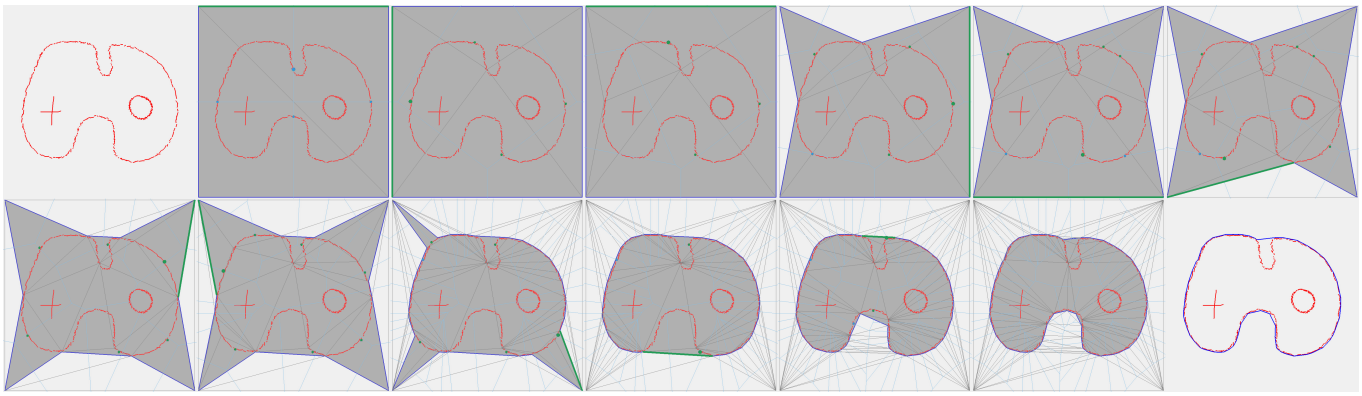


Fig. 2. Wrapping algorithm in 2D. Only a subset of the steps are shown for conciseness. **Top leftmost:** input line segments (red). **Middle:** the algorithm is initialized by inserting the corners of a loose bounding box into a Delaunay triangulation. Voronoi edges are depicted in light blue. All (finite) facets of the triangulation are tagged inside (grey). The green edge depicts the edge currently considered for carving (referred to as a *gate*). The green dots depict the points candidates for insertion into the triangulation, the largest one being the next one to be considered. Steiner points are inserted on the offset curve, either as intersections of Voronoi edges with the offset (blue), or as projections of Voronoi vertices onto the offset (green). One Steiner point has been inserted, and all incident facets are tagged inside. Each new Steiner point inserted adds new degrees of freedom for carving. Carving is performed when the inside facet adjacent to the current gate does not intersect the input and when the pencil of empty Delaunay discs circumscribing the gate's edge has a minimal radius larger than alpha. Note how the inaccessible inner structures are ignored by the refinement process. **Bottom rightmost:** The final output (blue) is the set of edges separating inside from outside Delaunay facets and is an immersed 1-manifold enclosing the input.

the offset, regardless of how close or distant the offset surface is from the input. Secondly, and as Stuart et al. note themselves, the topology and the quality of their output strongly depends on the position of the underlying grid in space, potentially forcing multiple runs to search for better results. Finally, the density of our underlying structure scales with the complexity of the output surface, whereas their underlying structure is a 3D grid whose density is that of the surface in the entirety bounding box, making their approach much slower and even unusable on large real-world meshes such as the complex crane model of Figure 26, as it would require an excessive number of grid vertices to recreate our dense result.

4.2 Algorithm

We provide next an in-depth description of our algorithm. A complete pseudo-code is also provided in Algorithm 1.

Parameters. Two parameters control the behavior of our algorithm: *alpha* (α) and *offset* (δ). Parameter α controls the minimum carving size, and thus the size of straits and holes that cannot be traversed during carving. Parameter δ is the value of the distance field level-set defining the offset surface. It controls the distance of the mesh vertices to the input, and thus the tightness of the approximation (see Figure 27). Both parameters can be chosen independently and arbitrarily large or small, but must be strictly positive.

Initialization. The algorithm begins with the construction of a coarse mesh that is valid and strictly encloses the input, two essential properties that are preserved throughout the course of the algorithm as the mesh is carved and refined. The simplest object fulfilling these requirements is a loose axis-aligned bounding box of the offset surface of the input, whose corner vertices are inserted into an empty 3D Delaunay triangulation. In order to deal only with tetrahedra, we consider that each facet located on the boundary of the triangulation (i.e., on the convex hull) is incident to an abstract *infinite* cell having as fourth vertex an auxiliary vertex called the *infinite vertex*. This way, each triangle facet of the triangulation is incident to exactly two tetrahedral cells. After the insertion of the bounding box corners, all the infinite cells are tagged as outside, and all the finite cells are tagged as inside. The initial mesh is composed of all triangle facets separating inside from outside cells.

Alpha wrapping. Past initialization, our algorithm traverses the cells of the triangulation outside-in, carving inside cells whenever possible (what is conveyed by *possible* will be detailed shortly), and refining them otherwise. We call *gate* a facet f that is incident to both an outside and an inside cells, and say that it is *α-traversable* if the minimum radius of the Delaunay balls of f — denoted $\Phi(f)$ — is larger than α (see Appendix B.1). Intuitively, cavities with a bottleneck smaller than α are not accessible as their gates are not

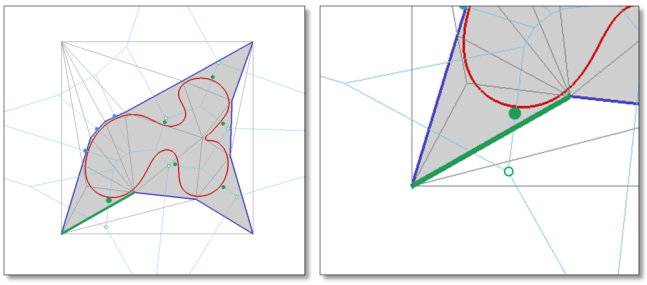


Fig. 3. Steiner points in 2D. **Left:** The current gate is depicted in green. Rule R1: Blue dots are the intersections between Voronoi edges and the offset. Rule R2: Green dots are the projections of Voronoi vertices (green circles) onto the offset. **Right:** closeup over the current gate.

ALGORITHM 1: Alpha Wrapping

Input: Input geometry $\mathcal{I}$, $Parameter\alpha$, Offset surface $\mathcal{I}_\delta$
Output: Output surface mesh $\mathcal{O}$
Initialize Delaunay triangulation $\mathcal{D}$ with the vertices of a loose bounding box of $\mathcal{I}_\delta$
Tag `outside` all infinite cells of $\mathcal{D}$
Tag `inside` all finite cells of $\mathcal{D}$
Initialize priority queue Q with convex hull facets of $\mathcal{D}$ that are α-traversable. Q is sorted in decreasing order of min Delaunay ball radius $\Phi(f)$
while Q *is not empty* **do**
  Gate $f \leftarrow Q.top()$
  $Q.pop()$
  if $\Phi(f) < \alpha$ **then**
    continue
  Let c_o be the cell incident to f tagged `outside`
  Let c_n be the other cell incident to f
  if c_n *is tagged* `outside` **then**
    continue
  Let k_c be the dual Voronoi vertex of c_o (circumsphere center), or the circumcircle of f if c_o is infinite
  Let k_{c_n} be the dual Voronoi vertex of c_n
  Let V_f be the oriented segment $[k_c; k_{c_n}]$
  Let P_s be a potential Steiner point
  if V_f *intersects* $\mathcal{I}_\delta$ **then**
    $P_s \leftarrow$ first intersection point between V_f and $\mathcal{I}_\delta$
  else if c_n *intersects* $\mathcal{I}$ **then**
    Let P_p be the point closest from k_{c_n} on $\mathcal{I}$
    Let E_p be the oriented segment $[k_{c_n}; P_p]$
    $P_s \leftarrow$ closest point from k_{c_n} on $\mathcal{I}_\delta$ and along E_p
  end
  if P_s *exists* **then**
    Remove from Q the gates that will be destroyed upon insertion of P_s into $\mathcal{D}$
    Insert P_s into $\mathcal{D}$
    Tag new finite cells `inside` and new infinite cells `outside`
    Push new α-traversable gates to Q
  else
    Tag c_n as `outside`
    Push all α-traversable gates incident to c_n to Q
  end
end
$\mathcal{O} \leftarrow$ surface triangle mesh represented facets separating `inside` from `outside` cells.
return $\mathcal{O}$

α-traversable. The order in which gates are traversed is governed by a priority queue containing α-traversable gates sorted by decreasing value of the gate's Φ, and initialized with the α-traversable gates on the convex hull of the loose bounding box. This particular choice of ordering mirrors the traditional ordering of elements — giving larger elements a higher priority — in Delaunay refinement algorithms, which our refinement operation has similarities with. Indeed, when traversing from an outside cell c_o to an inside cell c_i through an α-traversable gate f, two criteria (illustrated on Figure 3) are tested to prevent the carving process from exposing the input:

- **Rule R1:** We check for an intersection between the dual Voronoi edge of f, i.e. the segment connecting the circumcenters of its two incident cells, and the offset surface. If one or several intersections exist, the first intersection point along the dual Voronoi edge oriented from outside to inside is inserted into the triangulation as a Steiner point. See Figure 3. This insertion of the intersection of the dual Voronoi edge with the offset surface is similar to the popular restricted Delaunay refinement technique [Boissonnat and Oudot 2005].
- **Rule R2:** If Rule R1 does not apply, but the neighboring cell c_i intersects the input, c_i must not be carved, otherwise our mesh would no longer enclose the input. To prevent this, we construct the intersection of the offset surface with the segment connecting the circumcenter of c_i and its projection onto the input, and retain the intersection closest to c_i. This point is inserted into the triangulation as a Steiner point.

The intersection in Rule R1 is always tested, even if the neighboring cell does not intersect the input, in order to avoid inserting Steiner points on inner geometries.

After the insertion of a Steiner point because of either of these two rules, all newly created incident Delaunay cells are tagged as inside, and newly α-traversable gates are detected and pushed into the priority queue. If neither of the two rules apply, the gate f is traversed and its neighboring cell c_i is tagged as outside. Facets of c_i are tested and those that are α-traversable gates are added to the priority queue.

The construction phase terminates once the priority queue is empty, which is guaranteed to happen, as we will show next. The output triangle surface mesh is then extracted from the Delaunay triangulation as the set of facets separating inside from outside cells, which is guaranteed to fulfill our requirements, as we show next.

4.3 Guarantees

Thanks to its Delaunay-based refinement rules, our algorithm enjoys important theoretical guarantees. Complete proofs are available in Appendix B.

Termination. Because our algorithm interlaces carving and refinement steps, certainty of its termination is not immediate. We nevertheless demonstrate that termination is in fact guaranteed to happen, due to two complementary facts: the first (Lemma 1) is that the insertion rules construct Steiner points that are guaranteed to be strictly within the Delaunay spheres of the cells intended to be refined, causing the minimum Delaunay ball radius of facets to decrease with each insertion. The second is that the separation distance between any two vertices of the triangulation is bounded from below (Lemma 2). In that regard, the use of an offset surface is crucial as the parameter δ appears directly in the expression of the lower bound. We could not guarantee this separation distance if we inserted directly onto the input. Since the offset surface has finite area, termination is guaranteed under a packing argument (Theorem 1).

Watertight. Our output mesh is guaranteed to be watertight. This results from its definition as the set of facets of the Delaunay triangulation that are incident to both an inside cell and an outside cell. With the introduction of infinite cells, we have constructed a triangulation that is a partition of the Euclidean 3D space. If our final mesh were not closed, there would exist a path from one outside

tetrahedron cell to one inside without crossing a facet of the mesh, which is not possible.

Manifoldness. Each carving operation removes a whole tetrahedron and thus each edge of the output mesh is adjacent to an even number of faces, ensuring that it can be oriented.

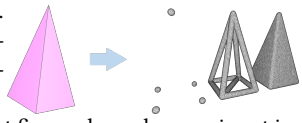
Enclosure. Our algorithm is guaranteed to generate a surface triangle mesh that strictly encloses the input. Indeed, we start from a mesh (the boundary of a loose bounding box of the input) which strictly contains the input and the carving process is not allowed to tag a cell outside if this cell intersects the input. Any time this is the case, Rule R2 is triggered beforehand and the cell is refined.

Quality Guarantees. We do not provide guarantees for the shape of the output mesh facets. Nevertheless, Rule R1 constructs Steiner points alike a common restricted Delaunay triangulation refinement, as the intersection of the dual of a facet with the surface being meshed. A Steiner point is thus equidistant from the facet vertices, which favors generating well-shaped facets. An empirical analysis of the quality of our meshes is part of our experiments (Section 5).

4.4 Implementation

We have implemented our algorithm in single-thread program in C++, using the components (kernels, triangulations, spatial searching data structures,...) of the CGAL library [The CGAL Project 2021].

Genericity. Genericity with respect to the input is achieved using the programming paradigm known as "Generic Programming". Through C++ templates, our implementation enables users to plug any input geometry in our algorithm, provided it is presented as a class answering three basic geometric queries: (a) intersection test between a Delaunay cell tetrahedron and the input, which is necessary to detect if carving must be disallowed; (b) construction of the intersection between a segment and the offset surface, which is necessary to compute the Steiner point in Rule R1; (c) projection of a query point onto the input, which is necessary to compute the Steiner point in Rule R2. In our algorithm, this abstract interface is referred to as the *oracle*. Our base implementation provides oracles for point sets, segment and triangle soups, triangle meshes, as well as mixed inputs (Section 7.1). The inset figure shows how an input is wrapped using three different oracles providing respectively access to vertices, edges, and facets of the pyramid.



Complexity Analysis. Thanks to its simplicity, the number of key computational steps of our algorithm is limited and its theoretical complexity is straightforward to analyze. At each iteration, we (a) compute circumcenters and circumradii of a small number of Delaunay triangulation cells, (b) check for an intersection between a tetrahedron and the input, possibly (c) compute a refinement point, and (d) insert it into the triangulation. Out of these four independent blocks, the first is a small amount of algebra with constant time complexity and is a negligible part of the total complexity. The last block consists in the insertion of a Steiner point into a 3D Delaunay triangulation for which we already know one cell of the so-called conflict zone, i.e., the set of cells that will be destroyed upon insertion of the new point. Indeed, the refinement algorithm specifically constructs Steiner points so as to "break" specific known cells. We therefore do not need to locate Steiner points in the triangulation — which would be a logarithmic-time complexity — and the insertion generally runs in constant time. An exception to this is the configuration of many vertices living on — or close to — the same Delaunay ball, causing major updates within the triangulation with each insertion. This degeneracy can be found in real-world data, such as a few half-domes that are part of the huge Thingi10k dataset [Zhou and Jacobson 2016]. It is however not a real issue as complexity can be brought down by inserting the center of the Delaunay ball onto which these many vertices live. The remaining operations, (b) and (c), both depend on the aforementioned oracle to detect an intersection between a cell and the input, to construct the intersection of a segment with the offset surface (Rule R1), and to construct projection of points onto the input (Rule R2). Whichever the representation of the input (point set, triangle soup, ...) or its associated oracle implementation, it is natural to expect that the complexity of any oracle query will be dependent on the complexity of the input. As such, oracle queries are by far the most expensive blocks of our algorithm. Among these oracle queries, the intersection between a segment and the offset surface (i.e., an implicit function) is the most expensive: the intersection point does not have a closed-form expression and consequently can only be approximated through dichotomy, which implies repeated point-input distance queries, each requiring traversal of the input data. On the other hand, a tetrahedron-input intersection test and the construction of a projection onto the input only require a single traversal of the input data. Efficient implementation of oracle queries are thus of considerable importance to achieve low runtime. In our oracles, traversal of the input data is considerably sped up using the Axis-Aligned Bounding Box (AABB) tree from the CGAL library [Alliez et al. 2021], a classic spatial searching data structure which provides both intersection and distance queries for the geometric objects of the CGAL kernel. We describe a number of advanced techniques that we have used to further speed up our implementation of oracle queries in Appendix E. We also provide extensive benchmarks of our implementation in our experiments (Section 5). Finally, we note that the complexity of our oracles depends (logarithmically) on that of the input, but it remains a fixed cost that does not depend on the number of vertices being constructed by our algorithm, which should exhibit roughly linear output complexity, i.e., a somewhat stable — or slightly decreasing due to the increasing small costs such as the priority queue — of vertex insertions per time unit.

Implementation Correctness. Although our method has multiple theoretical guarantees, we heavily use floating point number representations in our implementation for speed reasons. Despite the approximations caused by this inexact representation, our implementation is safe in the sense that we can ensure that the key computational steps do not fail or yield an invalid result, e.g. the construction of the refinement point of a Delaunay triangulation cell has to,and does break the cell it is meant to refine. This pratical correctness derives from the existence of a lower bound on the distance between a new point and the input, as well as between

any two vertices of the triangulation, which is large compared to the error we make while manipulating inexact number types (see Appendix C).

5 EXPERIMENTS

We now present the results of an in-depth empirical study of our algorithm. All experiments were performed on a desktop computer running Linux, with an Intel CPU clocked at 2.5 GHz, 128GB RAM and a solid-state drive. For consistency, the two parameters α and δ of the algorithm are always formulated in terms of ratio with respect to the length of the inner diagonal (l_b) of the bounding box of the model, that is $\alpha = 1/100$ in fact means that a threshold of l_b / 100 was used.

5.1 Validation

We begin by demonstrating that the theoretical guarantees of our algorithm translate well into the real-world by running our implementation on large datasets of complex and often defect-laden meshes: Thingi10 [Zhou and Jacobson 2016], GrabCAD, ABC [Koch et al. 2019] and Fusion 360 [Willis et al. 2021]. Our algorithm successfully handles all inputs and each output triangle mesh is verified — using exact predicates — to be a watertight and orientable mesh that strictly contains the input in all cases (see Figure 26).

A core feature of our algorithm is its unconditional robustness to input defects, which is a direct consequence of generating a mesh of the offset surface of the input. Figure 4 depicts the Tire, a model that has numerous gaps, holes, shifted surface parts and self-intersections created by imperfect CAD export. Despite being challenging for other methods (see also Figure 14), our algorithm produces a valid result.

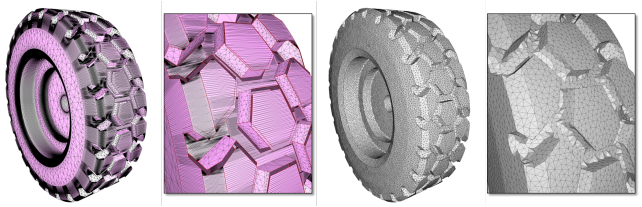


Fig. 4. Wrap of the defect-laden Tire model. Input mesh: 131K facets (pink). Output mesh: 60K facets (grey). α = 1/150, δ = 1/3000, 45s.

The use of an offset surface also provides stability with respect to noisy triangle soups, as demonstrated by Figure 5. Such cases are substantially more challenging for methods that can only locate vertices on the input geometry, or that rely upon facet orientations during inside/outside labeling.

5.2 Influence of the Parameters

The two parameters of the algorithm, α and δ, control the level of detail, the complexity, as well as the quality of the output mesh. Figures 6 and 27 illustrate a matrix of results for combinations of α and δ, on a free-form model and on a mechanical model with thin features. The input Bicycle model is non-manifold, and the input Hunter model has 1,200 pairs of intersecting triangles.

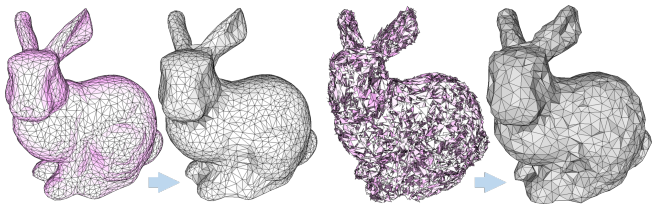


Fig. 5. Stability under random perturbation. **Left:** a clean input triangle mesh (5K facets) **Right:** the mesh facets are perturbed with a random rotation around their barycenter. Both wrapping meshes are generated using the same alpha and offset parameters (α = 1/40, δ = 1/500), in 4s.

Level of Detail. In our algorithm, the primary purpose of α is to control the size of the empty balls used during carving: a facet is α-traversable if the minimum radius of the Delaunay balls of the facet (Φ) is larger than α. As such, carving is only allowed through straits or holes with opening larger than α, and the value of the parameter determines which features will be present in the output (and the maximum size of triangle facets, see below). The second parameter, the offset distance δ, sets the distance between the input and the offset surface onto which the vertices of the output mesh are located, and thus controls the tightness of the output. The behavior of δ is similar to that of α when it comes to features: small values for δ ensure that the mesh is close to the input, and thus that it will capture well its geometry and features; on the other hand, large values for δ act as a defeaturing tool, for example closing holes and gaps such as on the wheels of the bike shown Figure 27 (bottom row, left to right).

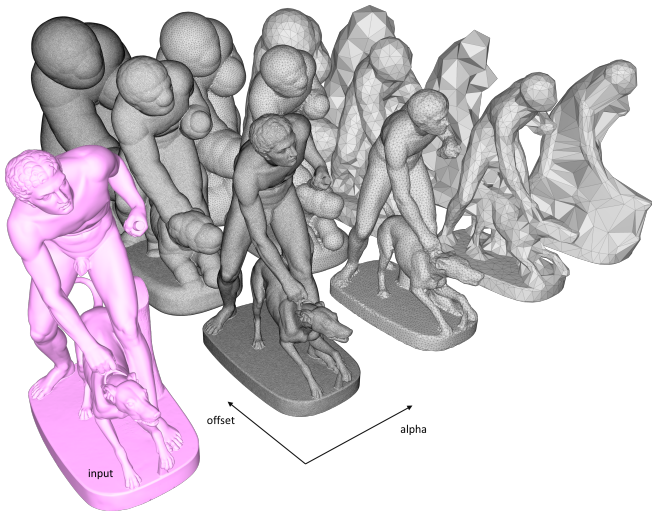


Fig. 6. Multiple wrappings of the Hunter model for various combinations of alpha and offset. Alpha controls the size of cavities that cannot be traversed during wrapping, and the size of triangles. Alpha values from left to right: 1/600, 1/200, 1/50 and 1/20. Offset controls the distance of the output mesh vertices to the input. Offset values from front to back: 1/5000, 1/50 and 1/20. These two parameters impact the complexity-fidelity tradeoff.

Complexity. An indirect influence of the parameter α is that it acts as a sizing criterion on the facets of the output mesh. Indeed, as long as a facet on the boundary between inside and outside Delaunay cells has a Φ larger than α, it will be traversed by the sculpting algorithm and its adjacent inside cell will be either carved or refined, depending on whether it intersects the input. Consequently, all boundary facets have a Φ smaller than α when the algorithm terminates. The offset value δ also plays a role in the density of the mesh: for a fixed value of α, locating vertices away from the input helps generate a sparser mesh — see Figure 6. A simple understanding of this behavior is to imagine a dense mesh of a sphere as input, and to enclose it with a mesh with four facets — i.e., a tetrahedron. The farther the offset is from the input, the lower the density of the mesh can be while still enclosing the input. Selecting a small value for α and a large value for δ reveals another influence of δ: in this configuration, α does not require the mesh to be dense, but since the offset surface is close to the input and because our mesh must be strictly enclosing, Rule R2 forces refinement in convex area to prevent cell-input intersections. In other words, a large flat area will satisfy a large requested α locally, whereas a high local curvature will trigger further refinement, possibly well beyond the requested α (which is hence only a sizing *criterion*, and not a strict sizing *field*). This can be observed for example on the wheels of the bike in the top left region of Figure 27. Appendix A provides more details on such a behavior.

Quality. We do not provide any theoretical bound on the quality of the output mesh elements such as minimal angles. Nevertheless, we observe that the two insertion rules of our algorithm are not similar when it comes to the quality of the elements created upon inserting their corresponding Steiner points. Indeed, Rule R1 constructs the intersection of a Voronoi edge with the offset surface and is the common restricted Delaunay refinement operation. Steiner points inserted through Rule R1 are equidistant from the vertices of the facet that the Steiner point refines. On the other hand, Rule R2 constructs the projection of a point onto the input, before lifting it back up onto the offset surface. As such, meshes tend to have better shaped elements when Rule R1 is responsible for most insertions. The two insertion rules of our algorithm (Section 4.2) are in fact not responsible for the same proportion of vertices for all values of α and δ: the farther the isosurface is from the input, the more new points are inserted through Rule R1 and the better the quality of the mesh elements (e.g., Figure 27, rightmost column). Conversely, selecting a small value for δ and a tight offset surface yields more intersections between tetrahedra and the input, and therefore more Rule R2 insertions.

Approximation Error. By placing mesh vertices on the offset surface, it is by definition that their distance to the input is δ. Let us now seek lower and upper bounds on the distance between an arbitrary point on the facets of the output mesh and the input. One can immediately see that the lower bound is zero — e.g., a segment of the output mesh can pass arbitrarily close to a convex corner, as long as it does not intersect it. The upper bound is given by $\alpha + \delta$ since the maximum distance for a point inside a facet f is reached at the center of the Delaunay sphere in the supporting plane of f (assuming it is in f), and all vertices are at distance δ from the input.

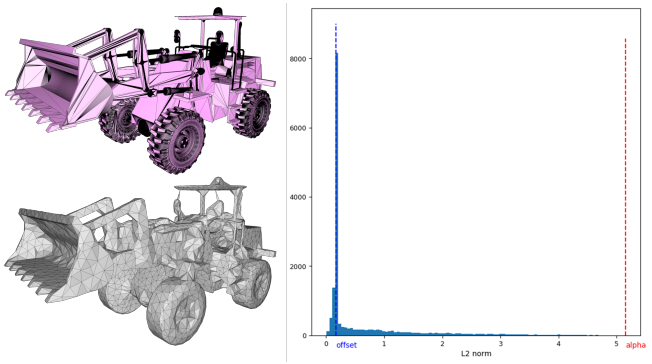


Fig. 7. Distribution of L2 errors between output and input. Input Loader model: 1.4M facets (pink). Output mesh: 9.6k facets (grey). α = 1/60, δ = 1/600, 13s.

Figure 7 is an empirical confirmation of these theoretical bounds, showing the distribution of one-sided L_2 errors between the output and the input meshes, computed by uniformly sampling 1M query points on the output mesh. The peak occurs around offset δ, and the maximum occurs slightly above α.

Sharp Features. All convex sharp edges of the input are rounded after offsetting. However, we observe empirically that when δ is small, the algorithm tends to preserve the convex sharp features, as the projections of circumcenters (Rule R2 insertion) in a region around an edge that is sharp enough to be called a feature, tend to be projected onto the said feature. See Figure 8. We discuss the handling of sharp concave edges in the limitations of our method.

Two-sided Wrap. The offset parameter δ is crucial to ensure that the output is an enclosing surface mesh, regardless of the quality and nature of the input. Indeed, when the input is a zero-volume structure (either intentionally or because of defects), methods that place vertices on the input cannot recover a volume, whereas our method yields a 2δ-thick wrap enclosing the zero-volume parts.

In addition to zero-volume structures, our algorithm can also handle open meshes, producing a 2δ-thick wrap everywhere, whereas

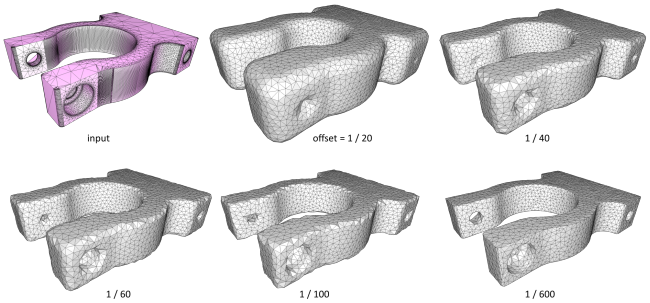


Fig. 8. The algorithm tends to better preserve convex sharp features when δ (offset) is small. Here, α is fixed (60). Respective face counts and runtimes from top middle to bottom right are 8,408 (10.2s), 6,856 (9.2s), 6,760 (9.3s), 6,484 (9.1s), and 7,118 (11.6s).

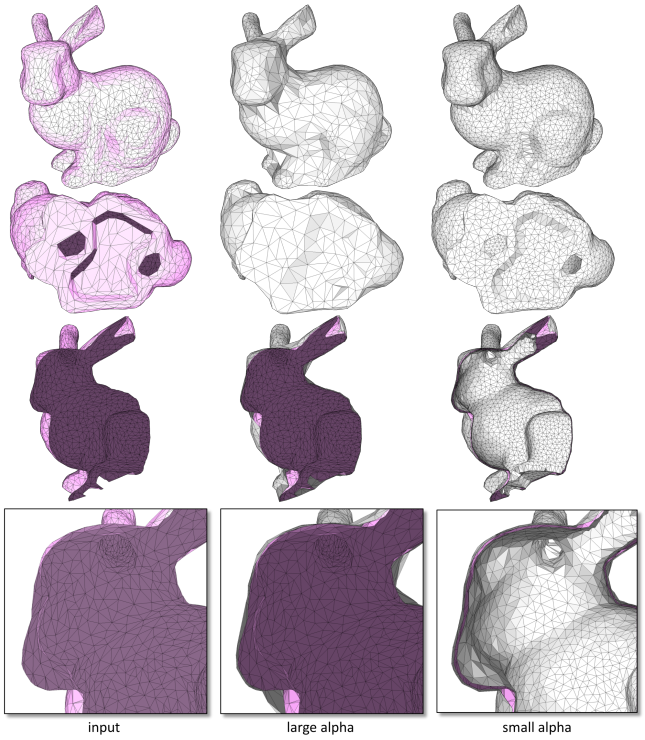


Fig. 9. Two-sided wrap. **Left column:** Input Bunny model: 5K facets, with two holes beneath the model. **Middle column:** Output with a large-enough α value (α = 1/30, δ = 1/500, 2306 faces, 0.7s): the holes are closed. **Right column:** Output with a small-enough α value (α = 1/60, δ = 1/500, 15132 faces, 5.9s): the holes are traversed during carving, resulting in a two-side wrap on most of the model (the interior of the ears are not fully carved in).

other methods would either fail or might need to add a lot of material to try and close holes. We call *two-sided wraps* partial or complete 2δ-thick wraps (Figure 9).

5.3 Performance

By using a simple underlying structure — a basic 3D Delaunay triangulation — and only refining it when required to capture the input, our approach does not suffer from the slow runtime of methods that are based on complex or memory-intensive underlying structures such as *constrained* Delaunay triangulations or dense regular grids. The evaluation of the distance field is by far the bottleneck of our algorithm. We use a number of advanced techniques to speed up evaluation — see Appendix E.

Figure 10 verifies that inner structures of the Volvo model that are inaccessible by empty balls of radius α are completely ignored by the algorithm: they do not even appear in the underlying 3D Delaunay triangulation. The input model, taken from GrabCAD, contains numerous self-intersections and non-manifold structures. It takes roughly 7 seconds for our approach to generate an output wrap of 30K facets, from an input model that contains 3M facets. The memory usage peaks at 2GB in our case, against 65GB for VolumeMesher [Diazzi and Attene 2021].

We confirm the predictions of our complexity analysis (Section 4.4) by selecting a single input mesh, the fan (GrabCAD), and recording the runtime (in ms) and the complexity (number of triangle facets of the output mesh) for two ranges of α and δ. In addition to being fast, the plots shown by Figure 11 confirm that the computational time of our algorithm highly correlates with the output mesh complexity.

In figure 12, we display the runtime for the entire Thingi10k dataset for four different α values.

5.4 Comparisons

To complete the evaluation of our algorithm, we have selected a number of recent approaches that are relevant to our problem of computing a watertight and strictly enclosing mesh, and for which implementations were available: PolyMender [Ju 2004], ManifoldPlus [Huang et al. 2020], TetWild [Hu et al. 2018], fTetWild [Hu et al. 2020], VolumeMesher [Diazzi and Attene 2021]. While some of these methods have goals close to ours, such as [Diazzi and Attene 2021], others have different purposes or requirements, e.g. Hu et al.[2020] whose method is a robust 3D mesh generator that does not intend to guarantee a strictly enclosing output. These differences should be kept in mind when comparing results, and especially runtimes. We have chosen not include any surface Delaunay mesher applied to the offset surface to our list because the guarantees and limitations of these algorithms are well known. They can guarantee a watertight and orientable mesh as output, but guaranteeing the strict enclosing property once refinement terminates — as for approximation bounds – would require knowledge on the offset surface such as local feature size, and connected components. This contradicts our will to only query the input via probing. Instead,

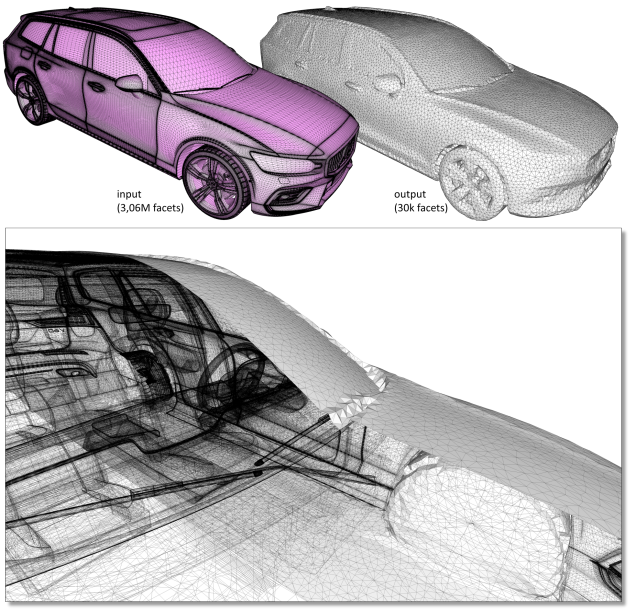


Fig. 10. Inner structures. The inner structures of this detailed detailed Volvo model (3M facets) are ignored by the wrapping process. The output mesh (61k facets) is generated in 17s (α = 1/200, δ = 1/400).

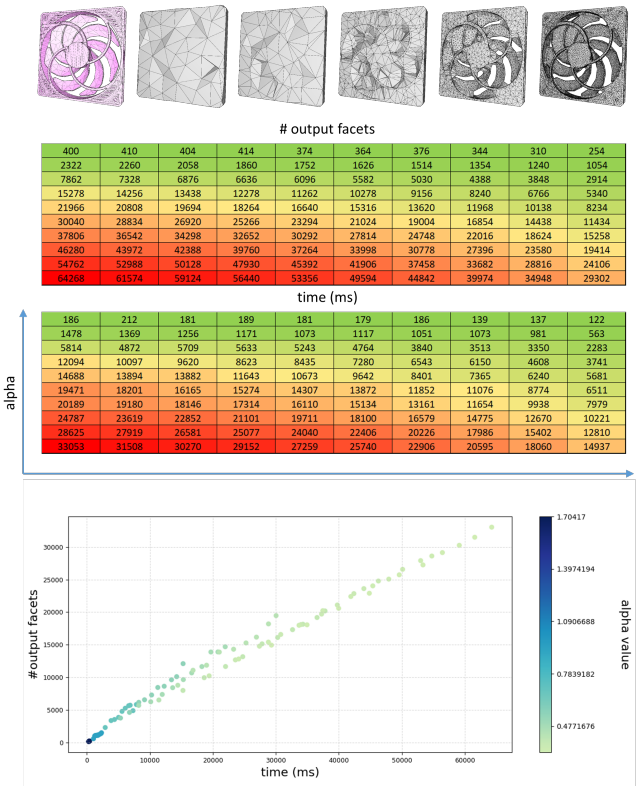


Fig. 11. Output complexity and time against alpha/offset parameters. **Top:** Input fan model, and few output meshes shown for increasing alpha parameter. **Middle top:** Complexity of the output mesh in number of triangle facets. **Middle bottom:** Computational time in ms. **Bottom:** Computational time against complexity of the output mesh. Each dot corresponds to one above combination of alpha/offset parameters.

our approach guarantees the enclosing property from the start and during the whole refinement process. In addition, our projection operator is effective at placing Steiner vertices on the tip of local convex bumps - a desirable feature to favor enclosing. A property that intersection operators with Voronoi edges does not offer.

More specifically, topological guarantees require both precise estimations of the local feature size of the offset, and a robust initialization of the Delaunay triangulation so as to not miss any connected component of the offset. Parameters of each algorithm have been selected to target the same output complexity, and with the intent of making comparisons as fair as possible. For our method, α is set to $l_b/100$ and δ to $l_b/3000$.

The comparison is performed over the entire Thingi10k dataset. For each method, we evaluate the robustness of each method, collecting the number and causes of failures (crash, invalid output) and timeouts (set to 1 hour), determine the properties of the output meshes sequentially (watertight, orientable, strictly enclosing), and record the average speed and memory. Results are shown in Table 1. Figure 13 and 14 illustrate two challenging models.

We observe that our algorithm performs reasonably well in terms of runtime and memory usage, and that other methods cannot be used in place of our algorithm as they almost-always fail to produce an enclosing output and even sometimes an orientable surface – a property which they do not guarantee but that is crucial in our setting.

Our initial benchmark revealed that six meshes reached the maximum allocated time (1 hour) and timeout for our algorithm. These meshes are characterized by the many vertices of the mesh living on or near the boundary of the same Delaunay ball, which makes the insertion into a 3D Delaunay triangulation much costlier as each new Steiner point induces a huge combinatorial modification of the triangulation. A solution was presented during the theoretical complexity analysis of our algorithm (4.4): when such large triangulation modifications are detected, we insert the center of the Delaunay ball sharing all these vertices. Such a modified refinement rule confirmed that this is an efficient solution: in the Thingi10k Earth Shot - Split model, which is a large Earth hemisphere, a single vertex is inserted for this rule. Such an insertion brings the runtime down from 3,015s to only 135s.

To complete our failure rate, speed, and validity analysis, we performed an additional benchmark to evaluate the quality of the results. Figure 15 records distributions computed over the complete Thingi10k dataset. We compare the runtime, memory consumption, output complexity, Hausdorff distance and several common facet quality criteria. Our approach performs well in terms of mesh quality and one-sided approximation error. Our peak memory and runtime are on par with TetWild and its fast variant, but are larger for other approaches.

We observe that in general, we compare favorably to other methods: although we are slower than the fastest methods, we use similar or less memory than other methods and our average facet quality (min angle, ratios) is higher than other methods. In addition, our one-sided Hausdorff distance is substantially lower, including for methods that only place vertices on the input.

6 APPLICATIONS

Although our algorithm can be used as a standalone to generate a valid mesh, its intended use is to be a key building block within an automated pipeline. We present two such possible pipelines. Both pipelines require a valid, watertight and enclosing mesh.

6.1 Alpha Wrapping for Flow Simulations

Predicting and optimizing flow around an object is central in industrial design. Modern computational power has made it standard to first iterate using only models and simulation before any real-world prototyping. Such a rapid conception process requires robust tools to bridge the gap between input representations that might contain defects, such as gaps between the panels of a CAD-designed car, and the expected input of a simulation, which is a valid and watertight model. Defeaturing possibilities may also be desired to speed up the process, and thus the number of iterations, during the simulation stage of the design. Our algorithm is a good fit for such tasks: it guarantees a valid, watertight mesh, and the alpha and offset parameters can be adjusted to select the appropriate level of defeaturing and accuracy to the input.

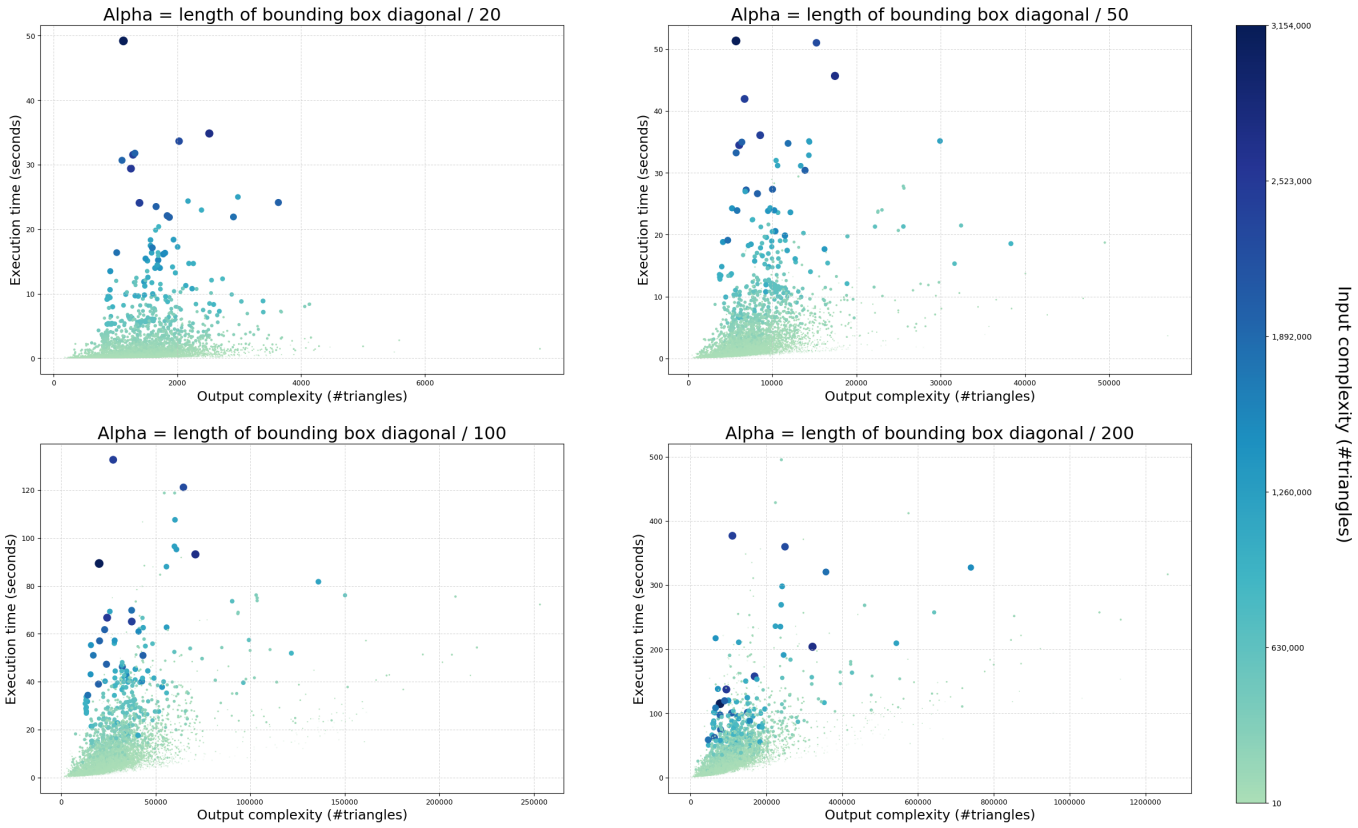


Fig. 12. For all models of Thingi10k, execution time against output complexity in number of triangles for four different alpha values. The color and size of each dot match the input complexity in number of triangles: larger, bluer dots correspond to inputs that are more complex than smaller, greener dots. The offset δ is set to $\alpha/30$ for all models.

Table 1. Comparison of robustness and performance of various state-of-the-art methods over the entire Thingi10k dataset. For each method, we document the percentage of inputs for which the method finishes before our timeout and produces a result. Among this percentage of results, we document when the result is orientable; orientable and watertight; orientable, watertight, and strictly enclosing. Note that methods other than ours were not designed to create a strictly enclosing output, and as such, fail to produce an output mesh that meets this property, hence the grayed out values.

Software	Failure	Finishes (<1h)	orientable	orientable + watertight	Average time (s)	Average memory peak (MB)
PolyMender [Ju 2004]	0.19%	99.81%	99.81%	99.81%	0.09	19.36
TetWild [Hu et al. 2018]	0%	99.59%	84.89%	84.89%	184.24	216.57
fTetWild [Hu et al. 2020]	0.01%	99.91%	99.55%	99.55%	93.11	146.79
ManifoldPlus [Huang et al. 2020]	0.06%	99.94%	99.94%	99.94%	2.27	47.90
VolumeMesher [Diazzi-Attene 2021]	0.67%	99.16%	71.3%	71.3%	10.55	279.13
Ours	0%	99.94%	99.94%	99.94% (+enclosing)	40.03	57.01

6.2 Alpha Wrapping for Efficient Swept Volumes

Being able to delineate the space swept by an object as it moves along a predetermined trajectory is crucial for motion planning and collision avoidance. For these problems, strict containment of the input at all times is required, and a tolerance around the input is required to accommodate safety constraints and real-world inaccuracies. Many methods have been proposed to compute swept volumes, but exact methods are in general very slow. Inexact methods rely on computing the union of a finite number of volumes from a discretization of the trajectory [Von Dziegielewski et al. 2013; Sellán et al. 2021]. Our approach here is orthogonal to the choice of the sweeping technique, and has many advantages: inner geometry is discarded, a valid input is guaranteed for the sweeping method, and the offset value can encode the desired real-world tolerance, as well

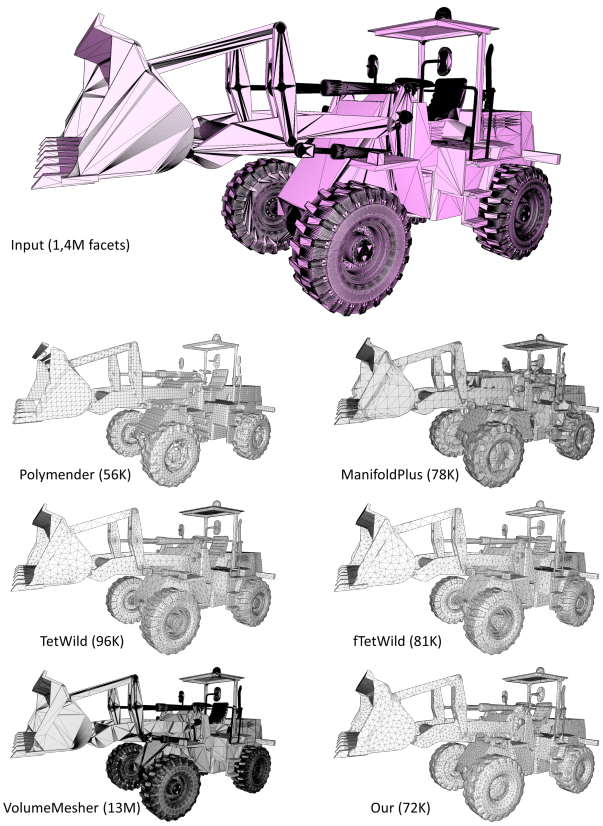


Fig. 13. Loader model meshed using various methods. Parts of the input mesh such as the roof top are missing from the output meshes for PolyMender, ManifoldPlus, and fTetWild. In addition, the output of ManifoldPlus contains many degenerate faces. The numbers indicate the number of facets of the output meshes. The runtimes and peak memory are as follows: PolyMender: 7.6s, 21.6MB. ManifoldPlus: 8.8s, 623MB. TetWild: 1,060s, 2GB. fTetWild: 551s, 2.1GB. VolumeMesher: 546s, 20GB. Ours: 17s (α=1/200, δ=1/600), 1.2GB.

as guarantee strict enclosing when using a conservative sweeping approach [Von Dziegielewski et al. 2013].

Figure 17 illustrates our method at work for generating low-complexity meshes provided as input to the high-precision conservative approach initially contributed by Von Dziegielewski et al. [2013], and improved by RMAGS GmbH (https://rmags.de/en/home-en/). The swept volume is generated from 191 poses that mingle rotations and translations. The computational times decrease from 914s (from the input model) to 242s, 85s and 74s depending on the resolution of the wrap meshes.

Figure 18 illustrates other swept volumes generated by the recent spacetime continuation approach contributed by Sellán et al. [2021]. The swept volumes are generated from 16 poses that mingle rotations and translations. The computational times decrease from 3,200s to 110s and lower.

7 EXTENSIONS

7.1 Mixed Inputs

Our generic approach enables easily switching input representations by simply replacing one oracle by another. In our implementation, we have added the possibility to combine oracles; consequently, different input representations can be mixed together in the same scene queried by the oracle. Figure 19 illustrates such a setting on the blade model, where we compute the alpha wrap on input data that combine points, polylines, and triangles. When alpha is selected to be smaller than the separating distance between two polylines (Figure 19, bottom), each polyline becomes wrapped within its own tubular mesh, and each point is wrapped by a tiny spherical mesh.

7.2 Inside-out Alpha Wraps

Our method has been devised to proceed by inward tetrahedron carving, motivated by the real-world use cases of collision detection and motion planning. It can also be desirable to perform the opposite, that is to carve from within to obtain an inner volume, for example in the context of indoor scene reconstruction. With the only knowledge of a seed point that is considered inside and at least δ away from the input, we extend our algorithm to perform *outward alpha wrapping with an offset*: all that is required is to change the initialization to create a few inside tetrahedra, and the carving process and insertion rules can then be performed identically. Figure 20 shows such an indoor scene. Despite the scene being riddled with many holes, degeneracies, and self-intersections, we generate a watertight, orientable, and *enclosed* mesh.

8 LIMITATIONS AND FUTURE WORK

Repair. Our approach is not intended to be a mesh repair solution, but rather a wrapping-through-remeshing approach for applications that require a conservative, enclosing approximation. For instance,

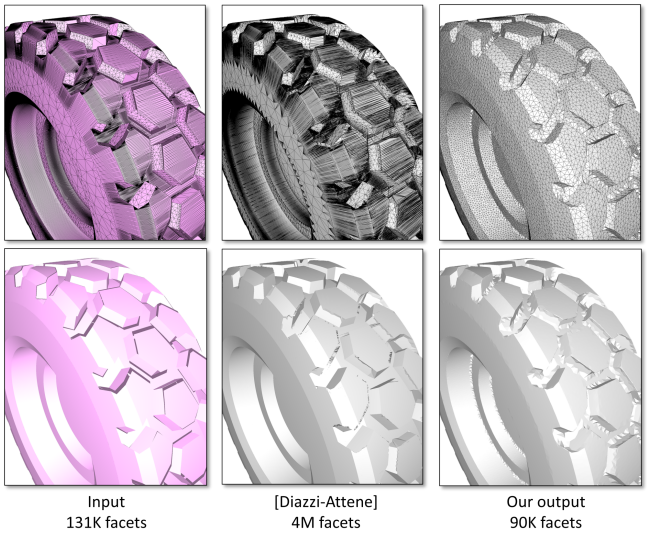


Fig. 14. Comparison with Diazzi-Attene. The Tire model from Thingi10K contains gaps, overlaps, and self-intersections. α = 1/200, δ = 1/3000, 57s.

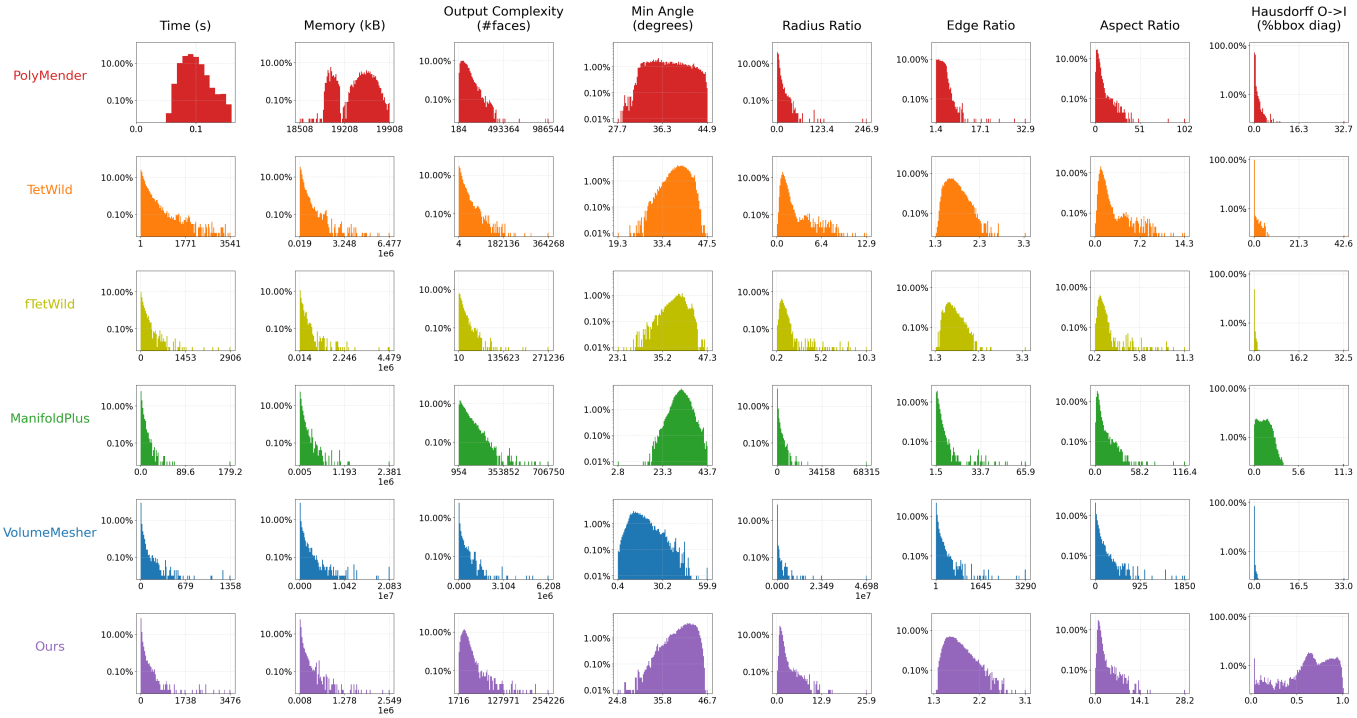


Fig. 15. Comparison of time, memory, and mesh quality distributions on the Thingi10k dataset. The vertical scales are logarithmic. The horizontal scales differ as the ranges are very different (notice the scaling factors bottom right).

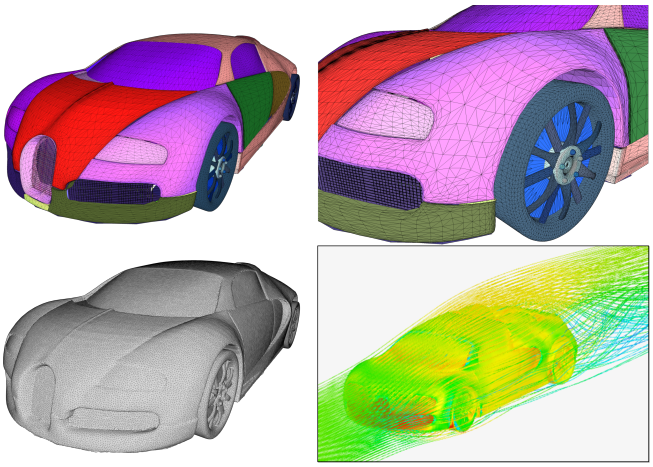


Fig. 16. Application to fluid simulation. Top: open input model (217k facets), with panel gaps, non-manifold and degenerate elements, self-intersections (non-conformal connected components). Bottom left: output mesh (476k facets) generated with α = 500, δ = 600, in 101 seconds. Bottom right: an incompressible flow simulation performed using Simscale's computational fluid dynamics.

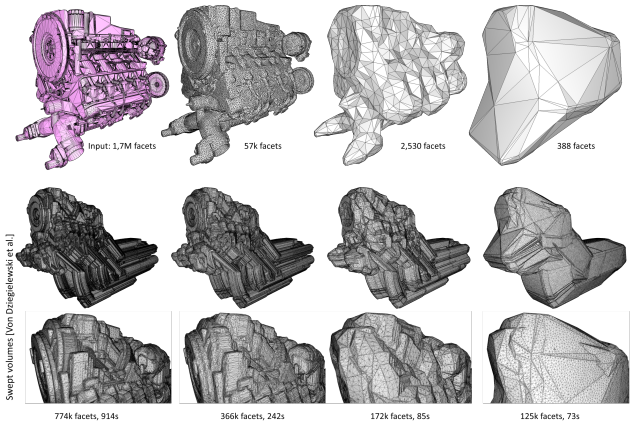


Fig. 17. High-precision swept volumes. Top: input mesh of an engine with many inner parts (*Chevy* model courtesy of GrabCAD), and three wraps generated using α = 200, 30 and 5, and a fixed offset parameter ($\delta = 1/400$). Bottom: High-precision conservative swept-volume meshes generated from 191 poses.

our method is not meant to fill large holes with planes as other global repair approaches do [Diazzi and Attene 2021]. Although it can close such holes with a sufficiently large alpha, this will likely cause undesirable collateral defeaturing. However, our approach enables users to choose the size of holes which must be filled based on a meaningful criterion.

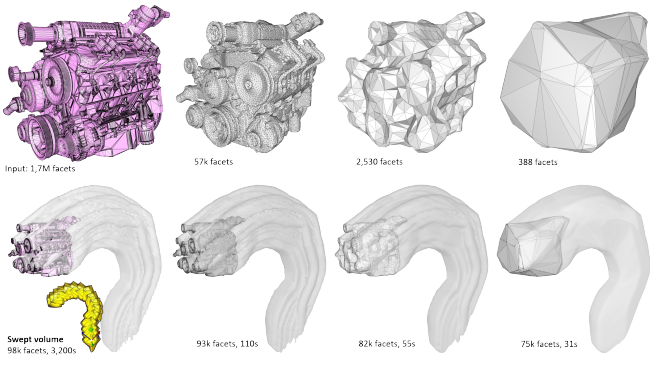


Fig. 18. Swept volumes generated by continuation. Top: input mesh of an engine with many inner parts (*Chevy* model courtesy of GrabCAD), and three wraps generated via continuation using α = 200, 30 and 5, and a fixed offset parameter ($\delta = 1/400$). Bottom: corresponding swept-volume meshes generated from 16 poses (yellow).

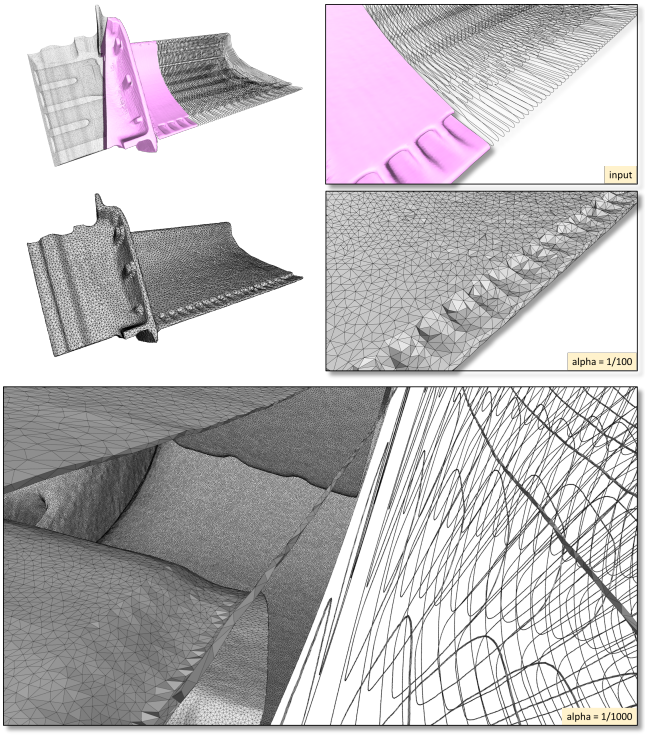


Fig. 19. Wrapping a mixed scene comprising points (leftmost slice of the input), triangles (middle slice), and line segments (rightmost slice). **Middle:** α = 1/100, 87K facets (11s). **Bottom:** α = 1/1000, 8M facets (982s).

Immersion versus embedding. Our algorithm generates an *immersed* combinatorial manifold (i.e., orientable), but it is not necessarily an *embedded* manifold. More specifically, elements of the output mesh always meet conformally since they are all extracted from the same 3D Delaunay triangulation; however, carving can create pinching at vertices or edges. For applications requiring strict

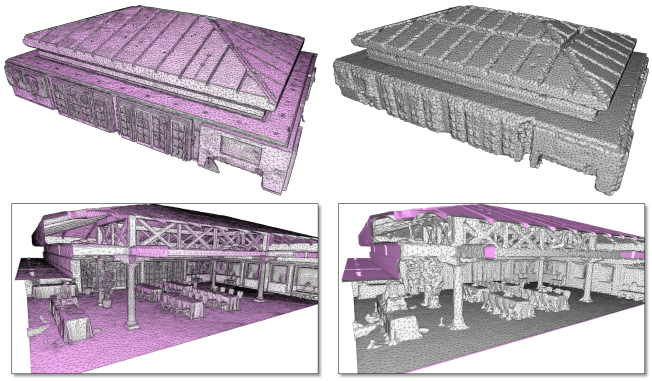


Fig. 20. Outer Alpha wrapping. Using a single seed placed in the middle of the mesh, we extract the indoor volume as a single volumetric component despite numerous holes (in the roof) and self-intersections (in the walls). Input mesh: 500K facets. Output mesh: 93K facets (α = 1/200). Computational time: 13s.

absence of self-intersections, forbidding such configurations during carving is straightforward, but yields larger volumes. A local ϵ-deflating of the inside elements seems like another valid option, but rigorously proving the correctness of the deflating paradigm is still out of reach.

Spatially-varying parameters. Alpha and offset parameters are global and uniform parameters, leading to meshes that are mostly uniform, especially for large offset values. An adaptive, non-uniform alpha parameter is desirable when seeking coarser meshes, but enlarging alpha may conflict with the objective of entering into small cavities. Such automatically-computed alpha fields would require understanding that the local geometry (1) is flat on a large area, and (2) has no cavities. The above understanding problem is ill-posed, especially when accessing defect-laden inputs via probing (as our oracle proceeds), and would require something akin to scale-space analysis. The problem is even more difficult when stating the problem of defining a notion of local curvature in a soup of disjoint 3D primitives embedded in 3D space. This ill-posed problem is thus left to future work.

Concave sharp features. The output wrap meshes are less tight near concave features. This is a well-known issue that arises in Delaunay meshes for which various solutions exist, such as the use of *protection balls* on sharp features [Cheng et al. 2008, 2010]. Our setting is more challenging, because we would need to (1) detect all concave sharp creases and corners, (2) offset the creases, (3) sample the offseted creases with α-sampling density, and (4) insert the sample points into the initial 3D Delaunay triangulation. However, this approach partially resolves the issue for three main reasons. First, detecting sharp creases and corners in soups of primitives is an ill-posed problem. Second, edges subtending small angles require resorting to a regular triangulation [Cheng et al. 2012]. Finally, sharp features of the offset surface may not even have corresponding sharp features in the input geometry. Heuristics to recover sharp features, such as those proposed for alpha-solids [Bernardini et al. 1999], is another direction, but they do not apply directly to our setting as

we need to sample an offset of the input. Inferring sharp features by probing a soup of 3D primitives is left to future work.

MLS. The Moving Least Squares (MLS) method is popular for reconstructing implicit surfaces from raw 3D geometries such as point sets or triangle soups. An exciting problem for future work is to extend our method with an MLS-based oracle in order to increase its capability to denoise, smooth, fill holes, defeature or detect sharp features. While the segment-offset surface intersection and input projection operators are within reach, devising reliable tetrahedron-intersection tests via MLS operators stands as a whole project on its own.

9 CONCLUSION

We propose a novel approach that is guaranteed to convert a raw input 3D geometry into a watertight, orientable surface triangle mesh that strictly encloses the input. Such a conservative standpoint is primarily motivated by simulation applications such as collision avoidance or trajectory optimization.

The algorithm interlaces greedy carving and refinement operations on a 3D Delaunay triangulation. Carving occurs outside-in with empty balls of user-defined radius α, while preserving at all times the enclosing property. Refinement inserts Steiner points on an offset surface of the input, either by intersecting Voronoi edges or by projecting Voronoi vertices. Once combined, theses principles result in a significant departure from creating an offset surface and using an off-the-shelf meshing algorithm. The level of detail, enclosing margin, and complexity of the output meshes are controllable through two meaningful user-defined parameters: α (size of unreachable cavities) and δ (offset value). A large value for α helps in getting rid of unreachable inner structures and cavities. A large value for both α and δ provides a means to drastically reduce the output mesh complexity and computational time. A large value for δ increases the quality of the output mesh elements, a good-to-have property for the robustness of many simulation algorithms. The algorithm is made generic via an abstract interface probing the input, making it possible to wrap a heterogeneous set of 3D primitives mingling points, line segments, and triangles.

Compared to other mesh-based approaches, our approach is reliable, simple and resilient to input defects. It performs well in terms of approximation and mesh quality, at the price of lower speed or higher memory consumption. We demonstrate its industry-grade robustness on large-scale datasets, for a wide range of defects. Our C++ implementation is already matured into a generic component for the open source CGAL project [2021]. The core algorithm and oracle fit into 2K lines of C++ code. They are completed by 8K lines of code for the tests, benchmarking, examples, user manual and reference manual.

ACKNOWLEDGMENTS

The work presented in this paper was funded by Google. The contributions by Michael Hemmer was done while the author was at Google. 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

Rémi Allegre, Raphaëlle Chaine, and Samir Akkouche. 2005. Convection-driven dynamic surface reconstruction. In *International Conference on Shape Modeling and Applications*. IEEE, IEEE, Cambridge, Massachusetts, USA., 33–42.

Pierre Alliez, Stéphane Tayeb, and Camille Wormser. 2021. 3D Fast Intersection and Distance Computation. In *CGAL User and Reference Manual* (5.3.1 ed.). CGAL Editorial Board, Sophia Antipolis, France. https://doc.cgal.org/5.3.1/Manual/packages.html#PkgAABBTree

Marco Attene, Marcel Campen, and Leif Kobbelt. 2013. Polygon mesh repairing: An application perspective. *ACM Computing Surveys (CSUR)* 45, 2 (2013), 1–33.

Gino van den Bergen. 1997. Efficient collision detection of complex deformable models using AABB trees. *Journal of graphics tools* 2, 4 (1997), 1–13.

Matthew Berger, Andrea Tagliasacchi, Lee M Seversky, Pierre Alliez, Gael Guennebaud, Joshua A Levine, Andrei Sharf, and Claudio T Silva. 2017. A survey of surface reconstruction from point clouds. *Computer Graphics Forum* 36 (2017), 301–329.

Fausto Bernardini and Chandrajit L Bajaj. 1997. Sampling and reconstructing manifolds using alpha-shapes. In *Proceedings of Canadian Conference on Computational Geometry*. Unknown publisher, Kingston, Ontario, Canada, 193—-198.

Fausto Bernardini, Chandrajit L Bajaj, Jindong Chen, and Daniel R Schikore. 1999. Automatic reconstruction of 3D CAD models from digital scans. *International Journal of Computational Geometry and Applications* 9, 04n05 (1999), 327–369.

Stefan Bischoff, Darko Pavic, and Leif Kobbelt. 2005. Automatic Restoration of Polygon Models. *ACM Transactions on Graphics* 24 (2005), 1332–1352.

Jean-Daniel Boissonnat. 1984. Geometric structures for three-dimensional shape representation. *ACM Transactions on Graphics (TOG)* 3, 4 (1984), 266–286.

Jean-Daniel Boissonnat and Steve Oudot. 2005. Provably good sampling and meshing of surfaces. *Graph. Models* 67, 5 (2005), 405–451.

Mario Botsch and Leif Kobbelt. 2001. A Robust Procedure to Eliminate Degenerate Faces from Triangle Meshes. In *Proceedings of the Vision Modeling and Visualization conference*. Aka GmbH, Stuttgart, Germany, 283–290.

Hervé Brönnimann, Christoph Burnikel, and Sylvain Pion. 2001. Interval arithmetic yields efficient dynamic filters for computational geometry. *Discrete Applied Mathematics* 109, 1-2 (2001), 25–47.

Fatih Calakli and Gabriel Taubin. 2011. SSD: Smooth signed distance surface reconstruction. *Computer Graphics Forum* 30, 7 (2011), 1993–2002.

Stéphane Calderon and Tamy Boubekeur. 2017. Bounding proxies for shape approximation. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–13.

Marcel Campen and Leif Kobbelt. 2010. Exact and robust (self-) intersections for polygonal meshes. *Computer Graphics Forum* 29, 2 (2010), 397–406.

Chia-Tche Chang, Bastien Gorissen, and Samuel Melchior. 2011. Fast oriented bounding box optimization on the rotation group SO (3, R). *ACM Transactions on Graphics (TOG)* 30, 5 (2011), 1–16.

Siu-Wing Cheng, Tamal K Dey, and Joshua A Levine. 2008. A practical Delaunay meshing algorithm for a large class of domains. In *Proceedings of the 16th International Meshing Roundtable*. IMR, Springer, USA, 477–494.

Siu-Wing Cheng, Tamal K Dey, and Edgar A Ramos. 2010. Delaunay refinement for piecewise smooth complexes. *Discrete & Computational Geometry* 43, 1 (2010), 121–166.

Siu-Wing Cheng, Tamal K. Dey, and Jonathan Shewchuk. 2012. *Delaunay Mesh Generation* (1st ed.). Chapman and Hall/CRC, Florida, USA.

Tamal Dey. 2010. Curve and Surface Reconstruction: Algorithms with Mathematical Analysis by Tamal K. Dey Cambridge University Press. *SIGACT News* 41, 1 (mar 2010), 24–27.

Lorenzo Diazzi and Marco Attene. 2021. Convex Polyhedral Meshing for Robust Solid Modeling. *ACM Transactions on Graphics* 40, 6, Article 259 (dec 2021), 16 pages.

Herbert Edelsbrunner. 2003. Surface reconstruction by wrapping finite sets in space. In *Discrete and computational geometry*. Springer, USA, 379–404.

Herbert Edelsbrunner and Ernst P Mücke. 1994. Three-dimensional alpha shapes. *ACM Transactions on Graphics (TOG)* 13, 1 (1994), 43–72.

Joachim Giesen, Frédéric Cazals, Mark Pauly, and Afra Zomorodian. 2006. The conformal alpha shape filtration. *The Visual Computer* 22, 8 (2006), 531–540.

Joachim Giesen and Matthias John. 2002. Surface reconstruction based on a dynamical system. *Computer graphics forum* 21, 3 (2002), 363–371.

Stefan Gumhold, Pavel Borodin, and Reinhard Klein. 2003. Intersection free simplification. *International Journal of Shape Modeling* 9, 02 (2003), 155–176.

Baining Guo, Jai Menon, and Brian Willette. 1997. Surface reconstruction using alpha shapes. *Computer graphics forum* 16, 4 (1997), 177–190.

Yixin Hu, Teseo Schneider, Bolun Wang, Denis Zorin, and Daniele Panozzo. 2020. Fast tetrahedral meshing in the wild. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 117–1.

Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. 2018. Tetrahedral Meshing in the Wild. *ACM Trans. Graph.* 37, 4, Article 60 (July 2018), 14 pages. https://doi.org/10.1145/3197517.3201353

Jingwei Huang, Hao Su, and Leonidas J. Guibas. 2018. Robust Watertight Manifold Surface Generation Method for ShapeNet Models. *CoRR* abs/1802.01698 (2018), 1–10. http://arxiv.org/abs/1802.01698

Jingwei Huang, Yichao Zhou, and Leonidas J. Guibas. 2020. ManifoldPlus: A Robust and Scalable Watertight Manifold Surface Generation Method for Triangle Soups. *CoRR* abs/2005.11621 (2020), 1–10. arXiv:2005.11621 https://arxiv.org/abs/2005.11621
Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. 2013. Robust inside-outside segmentation using generalized winding numbers. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–12.
Tao Ju. 2004. Robust Repair of Polygonal Models. *ACM Transactions on Graphics* 23, 3 (2004), 888–895.
Franjo Juretić and Norbert Putz. 2011. A Surface-Wrapping Algorithm with Hole Detection Based on the Heat Diffusion Equation. In *Proceedings of the 20th International Meshing Roundtable*. Springer, New York, USA, 405–418.
Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. 2006. Poisson Surface Reconstruction. In *Proceedings of EUROGRAPHICS Symposium on Geometry Processing*. Eurographics Association, Cagliari, Sardinia, Italy, 61–70.
Leif Kobbelt, Jens Vorsatz, Ulf Labsik, and Hans-Peter Seidel. 1999. A shrink wrapping approach to remeshing polygonal surfaces. *Computer Graphics Forum* 18, 3 (1999), 119–130.
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)*. IEEE, USA, 1–10.
YK Lee, Chin K Lim, Hamid Ghazialam, Harsh Vardhan, and Erling Eklund. 2010. Surface mesh generation for dirty geometries by the Cartesian shrink-wrapping technique. *Engineering with Computers* 26, 4 (2010), 377–390.
Peter Liepa. 2003. Filling holes in meshes. In *Proceedings of the ACM/Eurographics Symposium on Geometry Processing*. ACM, Aachen, 200–205.
Alexander Malyshev, Artem Zhidkov, Vadim Turlapov, and France Guyancourt. 2018. Adaptive mesh generation using shrink wrapping approach. In *Proceedings of GraphiCon'2018*. GraphiCon Scientific Society, Tomsk, Russia, 479–483.
D Martineau, J Gould, and J Papper. 2016. An integrated framework for wrapping and mesh generation of complex geometries. In *European Congress on Computational Methods in Applied Sciences and Engineering* (Greece). ECCOMAS, Europe, 6938–6954.
Fakir 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.
Leonardo Sacht, Etienne Vouga, and Alec Jacobson. 2015. Nested cages. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 1–14.
Pedro V Sander, Xianfeng Gu, Steven J Gortler, Hugues Hoppe, and John Snyder. 2000. Silhouette clipping. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press, USA, 327–334.
Silvia Sellán, Noam Aigerman, and Alec Jacobson. 2021. Swept Volumes via Spacetime Numerical Continuation. *ACM Transactions on Graphics* 40, 4 (2021), 11 pages.
Chen Shen, James F. O'Brien, and Jonathan R. Shewchuk. 2004. Interpolating and Approximating Implicit Surfaces from Polygon Soup. *ACM Transactions on Graphics* 23, 3 (2004), 896–904.
David A Stuart, Joshua A Levine, Ben Jones, and Adam W Bargteil. 2013. Automatic construction of coarse, high-quality tetrahedralizations that enclose and approximate surfaces for animation. In *Proceedings of Motion on Games*. ACM, USA, 213–222.
Marek Teichmann and Michael Capps. 1998. Surface reconstruction with anisotropic density-scaled alpha shapes. In *Proceedings of Visualization* (Research Triangle Park, North Carolina, USA). IEEE, IEEE, USA, 67–72.
The CGAL Project. 2021. *CGAL User and Reference Manual* (5.3 ed.). CGAL Editorial Board, Sophia Antipolis, France. https://doc.cgal.org/5.3/Manual/packages.html
Andreas Von Dziegielewski, Michael Hemmer, and Elmar Schömer. 2013. High precision conservative surface mesh generation for swept volumes. *IEEE Transactions on Automation Science and Engineering* 12, 1 (2013), 183–191.
Karl D. D. Willis, Yewen Pu, Jieliang Luo, Hang Chu, Tao Du, Joseph G. Lambourne, Armando Solar-Lezama, and Wojciech Matusik. 2021. Fusion 360 Gallery: A Dataset and Environment for Programmatic CAD Construction from Human Design Sequences. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–10.
Hongyi Xu and Jernej Barbič. 2014. Signed Distance Fields for Polygon Soup Meshes. In *Proceedings of Graphics Interface 2014* (Montreal, Quebec, Canada) *(GI '14)*. Canadian Information Processing Society, CAN, 35–41.
Wang Yifan, Noam Aigerman, Vladimir G Kim, Siddhartha Chaudhuri, and Olga Sorkine-Hornung. 2020. Neural cages for detail-preserving 3D deformations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, Nashville, 75–83.
Tong Zhao, Pierre Alliez, Tamy Boubekeur, Laurent Busé, and Jean-Marc Thiery. 2021. Progressive Discrete Domains for Implicit Surface Reconstruction. *Computer Graphics Forum* 40, 5 (2021), 143–156.
Qingnan Zhou and Alec Jacobson. 2016. Thingi10K: A Dataset of 10,000 3D-Printing Models. arXiv:1605.04797 [cs.GR]

A INTERPLAY BETWEEN ALPHA AND OFFSET

The interplay between α and δ requires considering the local (signed) curvature of the input geometry. Consider the convex case. The maximum size of the element depends on both δ and local curvature. Figure 21 depicts how a high local convex curvature triggers further refinement.

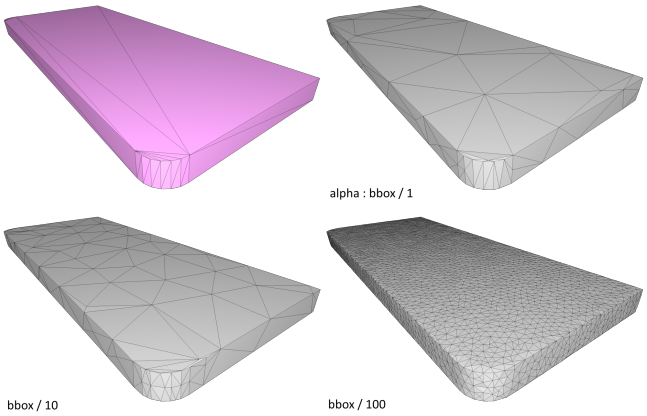


Fig. 21. Requested versus observed mesh sizing. δ is set to a very small value for all three models shown: 1/1000 of l_b. Even if α is selected very large, the high curvature area triggers further refinement at the rounded corners.

Figure 22 illustrates such a behavior step by step in 2D. Several skinny triangles (grey) keep intersecting the curved area, triggering refinement by projection of Voronoi vertices onto the offset. See also the *capsule* video in additional materials.

B PROOF OF TERMINATION

B.1 Definitions

We now prove that Algorithm 1 terminates under some assumptions on the algorithm parameters. The termination is proven by computing a lower bound on the insertion radius of vertices inserted into the mesh, and thus an upper bound on the number of vertices inserted by the above algorithm, under a packing argument.

Definition 1. *The 3D Delaunay triangulation of a set of points $\mathcal{P}$ of $\mathbb{R}^3$ is the pure simplicial complex $\mathcal{D}_\mathcal{P}$ whose cells satisfy the empty Delaunay ball property: the interior of the smallest circumscribing ball of any tetrahedron in $\mathcal{D}_\mathcal{P}$ does not contain any point of $\mathcal{P}$.*

Definition 2. *In 2D, there exists an infinite number of discs circumscribing the two vertices of a Delaunay edge (Figure 23). The geometrical locus of the centers of these discs is the (Voronoi)* dual edge *of the Delaunay edge. The notion of dual seamlessly extends to 3D Delaunay triangulations: the dual edge of a facet f of the triangulation is the geometrical locus of the center of the empty (Delaunay) balls circumscribing the three vertices of f. We denote by $\Phi(f)$ the minimum radius of the Delaunay balls of f.*

Definition 3. *Denote by $\mathcal{I} \subset \mathbb{R}^3$ a bounded input geometry. Denote by $\mathcal{V}_\delta = \{p \in \mathbb{R}^3 \mid d(p, \mathcal{I}) \leq \delta\}$ the offset volume of $\mathcal{I}$ and $\mathcal{I}_\delta$ its boundary.*

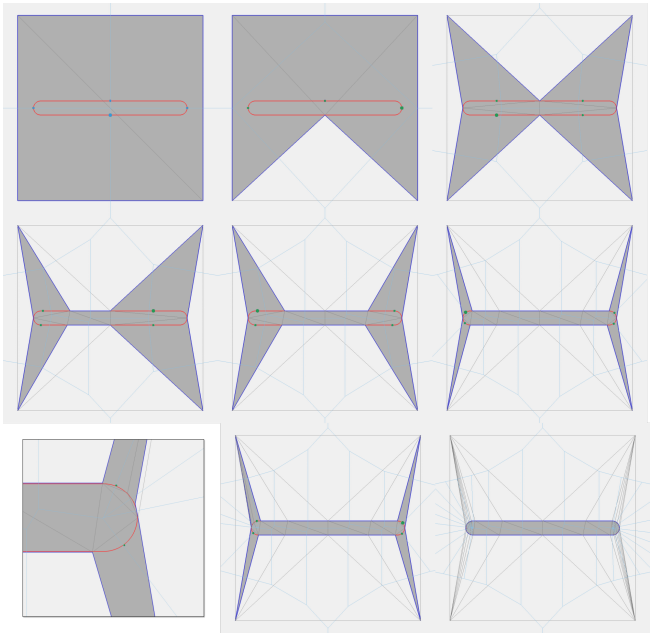


Fig. 22. Convex curvature in 2D. The 2D capsule is 1.2 wide. α is set to 1, and offset δ to 0.001. See also video *capsule* in additional materials.

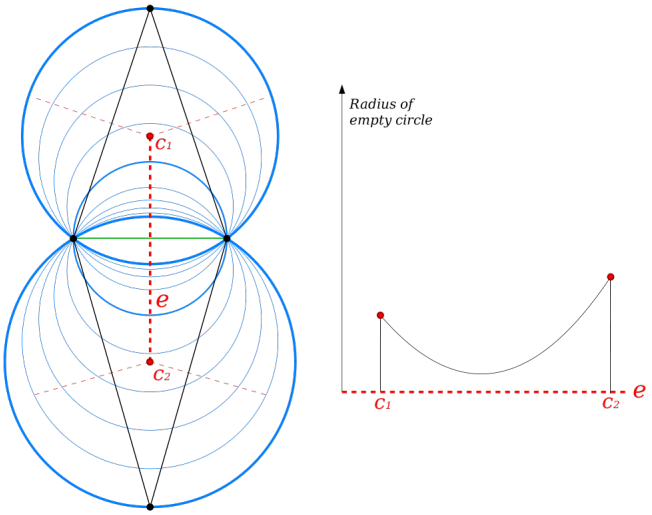


Fig. 23. Empty circles (blue) circumscribing a Delaunay edge (green) in a 2D Delaunay triangulation (black). From the top triangle circumcenter $c1$ to the bottom triangle circumcenter $c2$, the dual Voronoi edge denoted by e (doted red) is the trace of centers of the circles that are empty of Delaunay vertices. The graph plots the radius of empty circles centered on the Voronoi edge e, when progressing from $c1$ to $c2$. In this case the minimum circle coincides with the diameter circle of the green edge. When an angle facing the green edge is obtuse (not shown), the minimum circle coincides with either $c1$ or $c2$.

Definition 4. *Given a point p inserted within the set of vertices $\mathcal{P}$ by the algorithm, the* ***insertion radius*** *$r(p)$ is the Euclidean distance from p to $\mathcal{P}$ before the insertion (which is also the length of the smallest edge incident to the vertex after insertion into the 3D Delaunay triangulation):*

$$r(p) = \min\{d(p, p_i), p_i \in \mathcal{P}\}$$

Let f be a finite facet of a 3D Delaunay triangulation. Denote by c and c_n its two adjacent cells. Denote by k_c and k_{c_n} the circumcenters of the Delaunay ball of c (or of f, if c is an infinite cell) and c_n, respectively. Denote by $\Pi_\delta(p)$ the projection of a point $p \in \mathbb{R}^3$ onto $\mathcal{I}_\delta$, that is the point $q \in \mathcal{I}_\delta$ minimizing the distance between p and q.

In our algorithm, the cell c_n is eligible for refinement if (1) cell c is flagged outside and cell c_n is flagged inside, and (2) the facet f is α-traversable. If c_n is eligible for refinement, the following insertion rules can apply:

- Rule R1: Insert the intersection between the segment $[k_c; k_{c_n}]$ and $\mathcal{I}_\delta$ closest to k_c, if it exists.
- Rule R2: If c_n intersects $\mathcal{I}$, insert $\Pi_\delta(k_{c_n})$

Rule R1 overrides R2, i.e., Rule R2 applies only when Rule R1 does not apply.

The following lemma is a side observation, but elements of its proof will be used in the next (main) lemma.

Lemma 1. *The insertion of the new point using Rule* R2 *breaks cell c_n.*

Proof. Since c is outside and Rule R1 is not applied, k_c is not within $\mathcal{V}_\delta$ and the segment $[k_c, k_{c_n}]$ does not intersect $\mathcal{I}_\delta$. Hence k_{c_n} is not within $\mathcal{V}_\delta$. Since Rule R2 is being applied, c_n intersects $\mathcal{I}$ so its Delaunay ball also does, implying that the closest point projection π_n of k_{c_n} on $\mathcal{I}$ is in the Delaunay ball of c_n. Since $\Pi_\delta(k_{c_n})$ lies on the segment $[k_{c_n}; \pi_n]$, this implies that $\Pi_\delta(k_{c_n})$ is in the Delaunay ball of c_n, hence its insertion breaks c_n. □

Lemma 2. *The insertion radius of all points in the triangulation is bounded from below:*

$$\forall p \in \mathcal{P}, r(p) \geq \min(\alpha, \delta)$$

Proof. Let us consider the insertion of an arbitrary refinement point p in the triangulation. We distinguish depending on which rule governs the insertion of p.

Insertion due to Rule R1. Point p belongs to the dual Voronoi edge $[k_c, k_{c_n}]$, meaning it is the center of a Delaunay ball of f. Therefore, the distance between p and $\mathcal{P}$ is at least $\Phi(f)$, which is by assumption at least α.

Insertion due to Rule R2. Let π_n be the projection of k_{c_n} on $\mathcal{I}$. Let $\mathcal{B}_\pi$ be the ball centered at k_{c_n} and with radius $d(k_{c_n}, \pi_n)$. The Delaunay ball of c_n is empty and contains π_n (see the proof of Lemma 1), thus $\mathcal{B}_\pi$ is also empty of points of $\mathcal{P}$. Now, the point closest to $p = \Pi_\delta(k_{c_n})$ on the boundary of $\mathcal{B}_\pi$ is π_n, which is at distance δ from p. Hence p is a point in an empty sphere, at least δ away from the border, and thus:

$$\forall p_i \in \mathcal{P}, d(p, p_i) \geq \delta.$$

Finally, $\forall p \in \mathcal{P}, r(p) \geq \min(\alpha, \delta)$.

□

Theorem 1. *The algorithm described in Algorithm 1 terminates.*

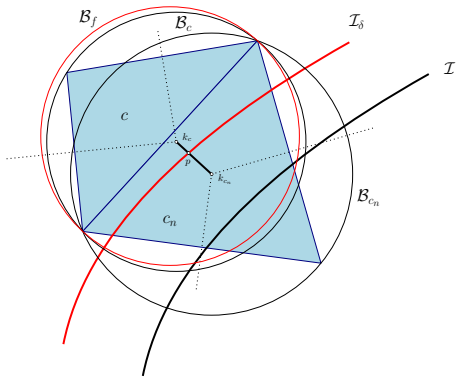


Fig. 24. Nomenclature for Rule R1.

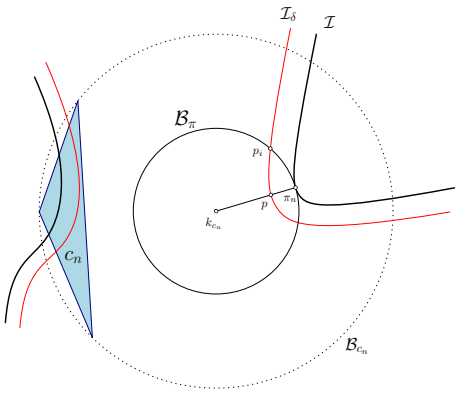


Fig. 25. Nomenclature for Rule R2.

Proof. From Lemma 2, we have that the insertion radius is bounded from below. Since $\mathcal{I}_\delta$ is bounded, the number of inserted vertices is bounded and the algorithm terminates. □

C PRACTICAL CORRECTNESS

Although our method has multiple theoretical guarantees, ensuring the in-practice correctness of our implementation is a different matter. An obvious departure from the theory is that to determine the intersection points between a Voronoi edge (a line segment) and the offset surface (an implicit function) used in Rule R2, we need to employ a dichotomy due to the lack of closed-form expression for these points. The resulting construction is thus in general an approximation of the exact position as there is no reason for the dichotomy, which recursively bisects the segment, to land exactly onto the implicit surface. Due to this approximation as well as for efficiency reasons, our implementation utilizes a floating point number representation (`double`), which systematically introduces additional numerical errors in our computations. A number of technical aspects must therefore be clarified to prove that our implementation, while not being the (impossible-to-achieve) exact result, is at least correct in the sense that it has the same guarantees as its ideal (theoretical) counterpart (termination, watertight and enclosing output mesh, etc).

Operations on geometric objects fall into two distinct categories: predicates and constructions. The former are operations that are akin to Boolean queries and provide a finite number of answers (e.g., yes-no) to questions such as "Is this point in the volume bounded by this sphere?". The latter construct and return new geometric objects such as the circumcenter of a Delaunay cell. Our geometric objects are double-based. We use the *filtered* predicates approach of CGAL [Brönnimann et al. 2001], which provides efficient and correct predicates by resorting to arbitrary precision if the evaluation of a predicate using double number types and interval arithmetic is insufficient.

Utilizing filtered predicates ensures that the steps of our algorithm requiring only predicates are correct: this includes the construction of the Delaunay triangulation (in-sphere tests), as well as the α-traversability and tetrahedron-input intersection tests. On the other hand, the refinement steps of our algorithm are constructions for which we — by default — cannot bound the error we make while using floating point representation, compared to the exact Steiner point position. We must thus ensure that the constructed position is *safe* albeit inevitably approximate: the point must satisfy its purpose, which is to break the Delaunay cell that it is meant to refine.

To ensure this, we perform new point constructions using an arbitrary precision (exact) number type, and convert the result to the base double representation. There are thus two moments where errors are performed: during the dichotomy in Rule R1 insertion, and during the final conversion. More specifically, even using an arbitrary precision would not guarantee that the dichotomic bisection will land exactly on the offset surface. A stopping threshold is therefore required. These errors must be controlled such that the final, post-conversion Steiner point is within the Delaunay ball. Fortunately, we can — for both refinement operations — give a lower bound on the (theoretical) distance between the Steiner point that is being inserted and the boundary of the Delaunay ball: δ. For example, in the case of the construction of a Steiner point of Rule R2, we know that the tetrahedron intersects the input. The closest point on the input is thus within the tetrahedron and within its Delaunay ball. Since the Steiner point is on the offset and thus δ away from this closest point, this gives a safety margin in the error we can make while converting to a floating point representation. The same reasoning stands for the R1-driven Steiner point construction, from which we can deduce the threshold used for terminating the dichotomy. Of course, these tolerances must be distinguishable from the δ parameter in our representation: if δ is chosen as the smallest representable value representable by a double, we cannot then set a small-enough bound on the dichotomy and conversion error. In this case, a different underlying number type should be used, but the reasoning for error control remains identical. Finally, we confirm empirically that the extra cost that comes with exact Steiner point constructions is negligible in practice, compared to the rest of the oracle query costs.

D COLOR PLATES

Our algorithm successfully handles all inputs shown by Figure 26: each output is a watertight and orientable surface triangle mesh that

strictly contains the input. Figure 27 illustrates a matrix of results for combinations of α and δ, on a bike model with thin features.

E IMPLEMENTATION DETAILS

In this section, we give insight into a few advanced techniques that we have used to speed up our CGAL-based implementation. As described in our complexity analysis (Section 4.4), the input distance query (i.e., the problem "Given a point p in R^3, what is the closest point q on the input, and what is the distance between p and q?") is by far the bottleneck of our algorithm. Indeed, the query is called repeatedly during the dichotomic search performed to hone in on the intersection between a segment and the offset surface (Rule R1). Consequently, most of our efforts have focused on accelerating this particular query, and minimizing the number of query calls.

Dichotomy. Because the construction of the intersection location between a segment and an implicit surface has in general no closed-form expression, we utilize a dichotomic search to approach the location, which requires an important number of point-input distance queries. The dichotomy stops once a certain precision bound has been met (see Section C, on enforcing in-practice correctness). Whether the input is a soup of points, segments, or triangles, or a polygon mesh, all oracles utilize the AABB tree data structure (see Section 4.4). An AABB tree is a hierarchy of bounding boxes, whose leaves contain the input primitives. Each parent node stores the bounding box of its children's bounding boxes.

Our two main improvements are based upon the observation that the distance function to the input is 1-Lipschitz. Indeed, a Lipschitz bound on the derivative of the distance function enables pruning entire intervals during the dichotomy search, considerably reducing the number of intervals required to be tested before convening to a solution. Our second improvement is specific to the AABB tree of CGAL: a distance query consists in finding the closest point on the input, which is performed by *traversing* the tree. The traversal consists in exploring the tree top-down from its root, swiftly pruning irrelevant branches of the tree by checking for intersection between a node's bounding box and the query's bounding box, and finding leaves (and thus input primitives) onto which the closest point may lie. In the traversal function of the AABB Tree, an upper bound on the distance can be passed as parameter to bypass some operations during traversal: if the traversal process is at a leaf and the closest point is farther than the bound, it is immediately ignored rather than having to compare it to the current best, and potentially updating this current best. At one step of the dichotomic search, a subsegment s knows only two distance values at its end. Since the distance function is in general non-linear in s, we utilize the Lipschitz property to bound the maximum distance value along s.

Input Intersection Tests. An important property required from our method is that it produces an output mesh which strictly encloses the input 3D geometry. To achieve this guarantee, our algorithm regularly checks for intersections between a tetrahedron (a Delaunay cell) and the input geometry. Similarly to point-input distance computations, this operation is performed with a traversal of the AABB Tree. We have made a number of improvements in the final intersection check — at the leaf level — between an input primitive and the tetrahedron, over a naive implementation. We detail below the case of an input soup of triangles, but similar improvements have been carried out for other geometric primitives.

The standard check at leaf level would be a triangle-tetrahedron intersection test, using a number of orientation predicates. Instead, we add extra filtering tests: similarly to the other nodes of the AABB Tree, we check for overlaps between the bounding boxes of these two primitives. In addition, we also perform bounding box overlapping tests for the four triangle-tetrahedron faces. If such an overlap exists, we use triangle-triangle intersection tests that are substantially faster. Finally, we enrich this intersection predicate with prior algorithmic knowledge. Knowing that the Delaunay facets traversed by the algorithm do not intersect the input, we can simply ignore some faces of the tetrahedron, further reducing the number of triangle-triangle tests. After the triangle-triangle tests, a simple point-tetrahedron bounding test suffices, as the triangle is either fully enclosed or fully outside. Finally, we heavily trade memory for speed by caching triangular faces, bounding boxes as well as already visited information.

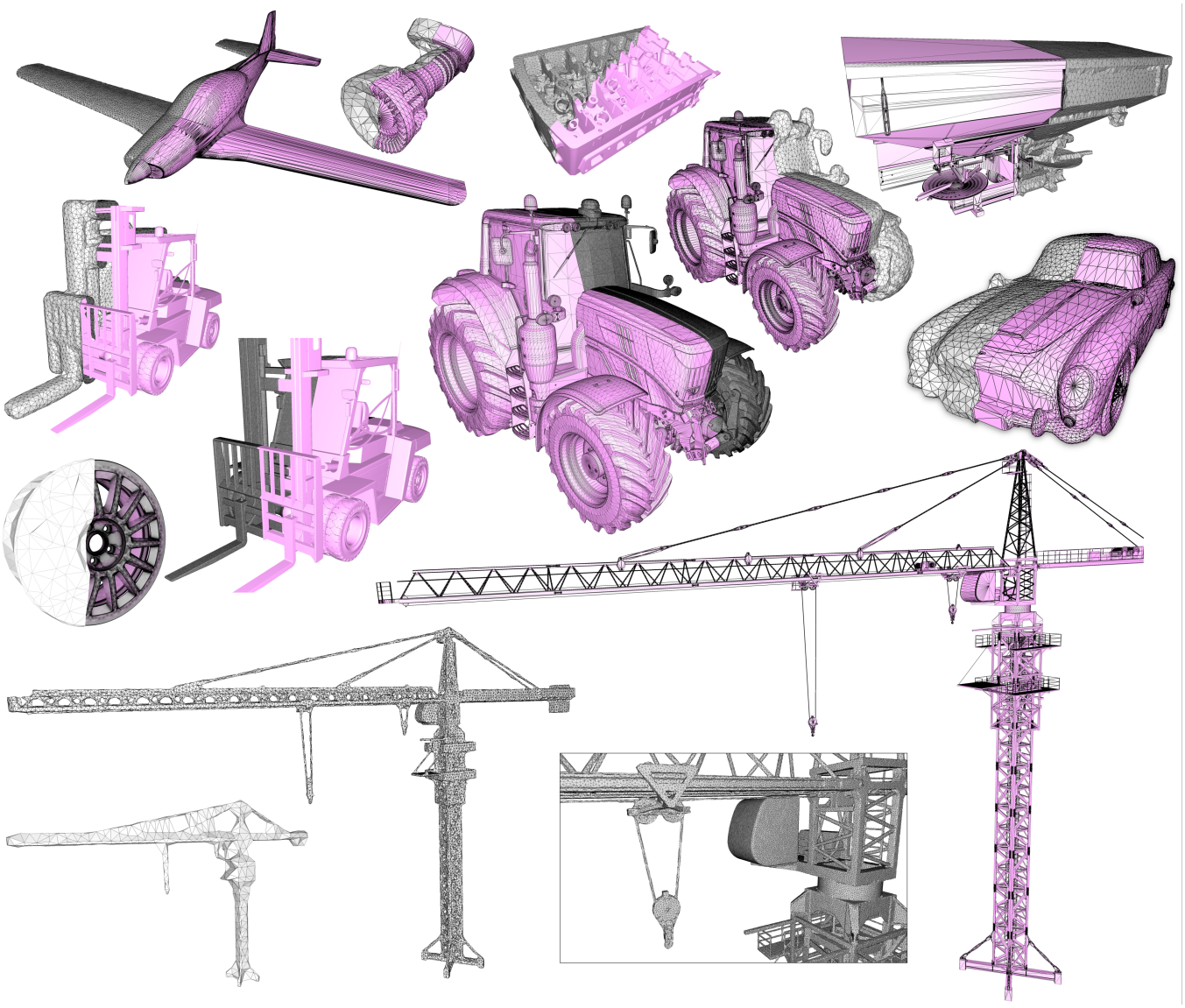


Fig. 26. Gallery of wrapped models. Input meshes in pink. Output meshes in grey. Computational times range from 2s for the nozzle to 8,000s for the finest crane model with tiny alpha and offset (output = 7,4M facets), through 25s for the coarse forklift with large offset, resp 2,350s for the small alpha.

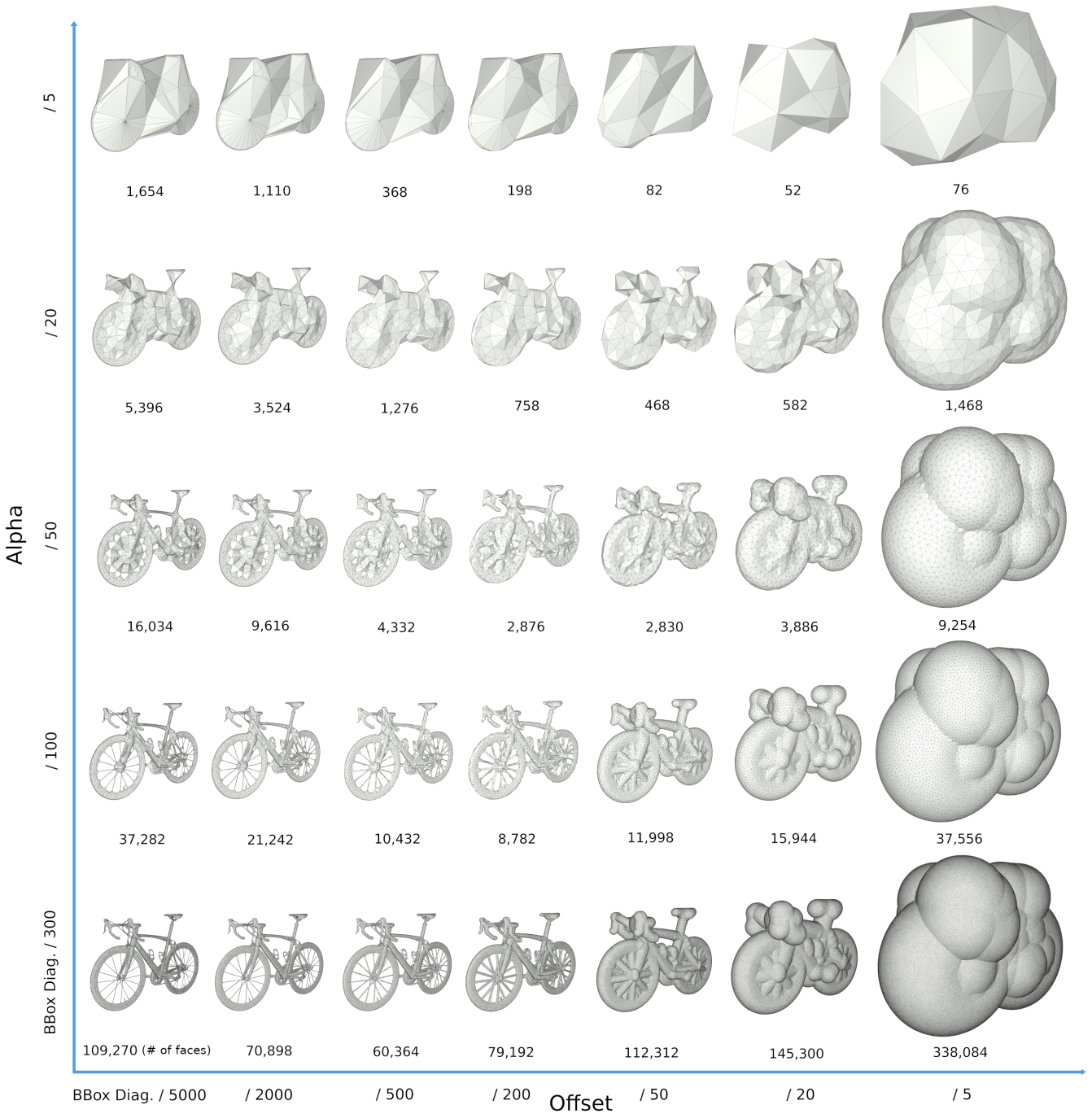


Fig. 27. Wrapping a bicycle with increasing alpha/offset parameters. Alpha controls the size of cavities that cannot be traversed during wrapping. Offset controls the distance of the output mesh vertices to the input. These two parameters provide a means to adjust the complexity-fidelity tradeoff. Numbers indicate the number of facets of the output mesh.