Solid-Shell Labeling for Discrete Surfaces

SIQI WANG, New York University, USA
JANOS MENY, Roblox, USA
IZAK GRGURIC, Roblox, Canada
MEHDI RAHIMZADEH, Roblox, USA
DENIS ZORIN, New York University, USA
DANIELE PANOZZO, New York University, USA
HSUEH-TI DEREK LIU, Roblox, Canada

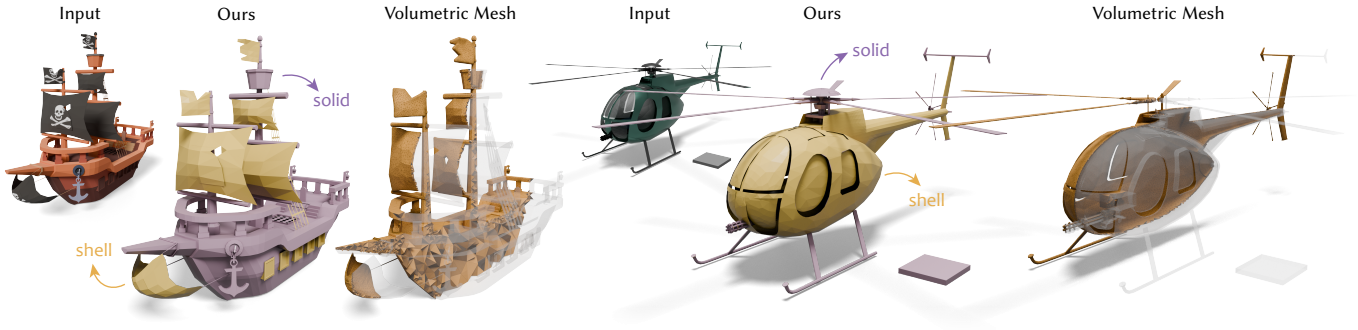


Fig. 1. Given an input shape, our method labels each triangle as either the boundary of a *solid* (purple) or the medial surface of a *shell* (yellow), driven by sparse user guidance. These facet labels can be passed to off-the-shelf tetrahedral (for solid faces) and offset (for shell faces) meshers to produce volumetric meshes (orange) that are ready for downstream applications. We obtain the ship model from kenny.nl licensed under CC0, and the helicopter model ©Roblox.

Artist-created meshes in-the-wild often do not have a well defined interior. We observe that they typically consist of a mix of *solid* elements, faces that bound a volume, and *shell* elements that represent the medial surface of a thin shell. The lack of a well-defined interior prevents downstream applications, such as solid-modeling, simulation, and manufacturing. We present a method that takes as input a surface mesh and assigns to each face a label determining whether it belongs to a solid or shell. These labels reduce ambiguity by defining the interior for solid faces through thresholding the generalized winding number field, and for shell faces as the volume within an offset. We cast the labeling problem as an optimization that outputs a solid/shell label for each face, guided by a sparse set of user inputs. Once labeling is complete, we show how the shape can be volume meshed by passing the shell faces through an offset mesher and the solid faces to an off-the-shelf tetrahedral mesher, producing a final volumetric mesh by taking their union. Experiments on diverse meshes with defects and multiple solid and shell components demonstrate that our approach delivers the desired labels, enabling modeling and simulation on wild meshes in a way that respects the user intent.

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

Additional Key Words and Phrases: shape modeling, volumetric models

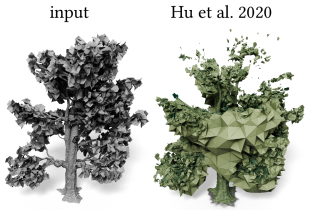
ACM Reference Format:
Siqi Wang, Janos Meny, Izak Grguric, Mehdi Rahimzadeh, Denis Zorin, Daniele Panozzo, and Hsueh-Ti Derek Liu. 2025. Solid-Shell Labeling for Discrete Surfaces. In *SIGGRAPH Asia 2025 Conference Papers (SA Conference Papers '25), December 15–18, 2025, Hong Kong, Hong Kong*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3757377.3763847

Authors' addresses: Siqi Wang, sw4429@nyu.edu, New York University, New York, USA; Janos Meny, jmeny@roblox.com, Roblox, San Mateo, USA; Izak Grguric, igrguric@roblox.com, Roblox, Vancouver, Canada; Mehdi Rahimzadeh, mrahimzadeh@roblox.com, Roblox, San Mateo, USA; Denis Zorin, dzorin@cs.nyu.edu, New York University, New York, USA; Daniele Panozzo, panozzo@nyu.edu, New York University, New York, USA; Hsueh-Ti Derek Liu, hsuehtil@gmail.com, Roblox, Vancouver, Canada.

SA Conference Papers '25, December 15–18, 2025, Hong Kong, Hong Kong

ACM ISBN 979-8-4007-2137-3/2025/12
https://doi.org/10.1145/3757377.3763847

1 INTRODUCTION



Generating volumetric meshes from a given shape is a fundamental subroutine within geometric computation and modeling. Accordingly, extensive research has addressed key subproblems, such as tetrahedralizing a bounded volume and determining inside/outside relative to a triangle mesh. Decades of research have yielded robust methods that reliably mesh volumetric domains.

Despite these advancements, existing volumetric meshing algorithms can sometimes produce unexpected results (see the inset), especially for artist-created assets in online repositories. These failures stem from the assumption that the input triangle mesh "approximately" describes the boundary of a well-defined solid volume. In

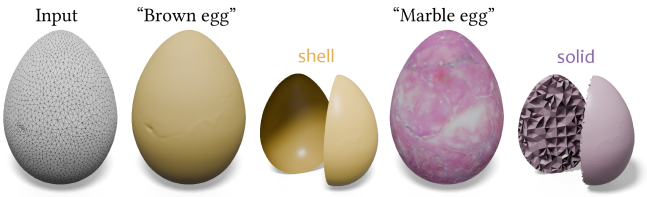


Fig. 2. Given a triangle mesh (left), whether a shape is a shell or solid may require understanding the semantics, such as an eggshell (middle) or a marble (right). Our method incorporates user guidance and is able to produce user-desired face labels (yellow shell and purple solid) from the same input geometry.

Fig. 3. Existing heuristics, such as using triangle visibility [Zheng et al. 2024], to classify solid/shell faces are prone to artifacts, such as deleting interior components. ©Roblox

practice, however, this assumption does not always hold. Whether a shape is a volumetric thin *shell* or a *solid* depends on the semantics (see Fig. 2), instead of purely relying on geometric information. Moreover, many shapes in the wild consist of a mixture of solid and shell components (see Fig. 1), posing further challenges to meshing.

This ambiguity between shell and solid structures results in some heuristic strategies in practice, such as assuming everything is solid (or shell) or attempting to infer solids/shells based on visibility. However, these heuristics frequently prove inadequate (see Fig. 3), leading to meticulous, time-consuming manual intervention to correct inaccuracies at the individual polygon level. Obtaining user-desired face labels is critical to volumetrically meshing the object: tetrahedral meshing for solid faces and offset meshing for shell faces. Incorrect labeling can cause artifacts in the resulting volume mesh such as distorted appearance (Fig. 12), deletion of invisible structures (Fig. 3), and unexpected physical behavior (Fig. 10).

In lieu of this, we present a method for classifying each face of a triangle mesh as belonging to either a *solid* or a *shell* component, requiring only sparse user guidance. We formulate this labeling problem as an energy optimization, wherein the label of each face serves as the optimization variable, and the user-provided guidance is incorporated into the energy functional. After minimization, we take the resulting face labels to construct a volumetric mesh. Faces labeled as "solid" are processed through tetrahedral meshing tools to obtain a volumetric mesh. Faces that are identified as "shell" are passed through offset surfacing techniques, yielding a volumetric thin shell (see Fig. 4). After these two streams of volumetric meshing, we union the results and obtain a single mesh which is amenable to

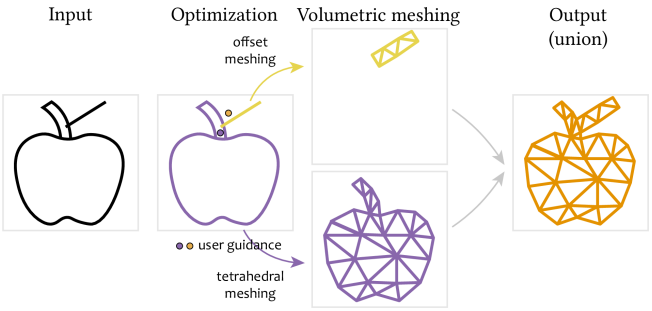


Fig. 4. Given an input mesh (left), our method optimizes the label for each face to be either a solid (purple) or a shell (yellow) given sparse user guidance (purple/yellow dots). After the optimization, shell faces are passed through an offset mesher (top), solid faces are passed through a tetrahedral mesher (bottom). Then these two volumetric meshes are unioned to produce the final volumetric mesh (orange).

downstream applications, such as the Constructive Solid Geometry (CSG) modeling or physically-based simulation.

2 RELATED WORK

Our method classifies each face in a mesh as either solid or shell to support volumetric meshing. We provide a brief overview of volumetric meshing and related topics that require determining the interior of an object.

2.1 Volumetric Meshing and Offset Construction

One can compute a volumetric mesh from a surface mesh with either *volumetric* (e.g., tetrahedral meshing) or *offsetting* meshing algorithms, which operate under opposing implicit assumptions. Solid-oriented volume meshers (e.g., [Alliez et al. 2024; Diazzi and Attene 2021; Diazzi et al. 2023; Hu et al. 2020, 2018; Si 2015]) assume the input triangle mesh is an approximation of a *closed solid* and produce a volume mesh whose boundary faces approximate the input. This "closed-solid" assumption leads these methods to automatically close gaps and remove interior details, but this behavior can conflict with artistic intent (see Fig. 13). In contrast, shell-oriented meshers, such as [Cao et al. 2023, 2024; Zint et al. 2024], treat each triangle of a mesh as a thin shell element. These methods construct a volumetric mesh by offsetting the surface in both directions and tetrahedralizing the thin volume enclosed within the resulting shell. While offset surfacing methods preserve the visual appearance and maintain all interior components, they tend to produce unrealistic physical behaviors for objects that should behave as solid bodies. Due to conflicting assumptions between solid- and shell-oriented meshers, existing techniques are only applicable to *homogeneous* inputs where all faces in the mesh are consistently either solid or shell. In practice, however, artist-created meshes frequently contain mixed elements that violate these assumptions (see Fig. 1), causing both approaches to produce undesirable results. To address this limitation, our method automatically generates per-face labels that reconcile these two perspectives. Faces classified as solid are processed by a solid-oriented tetrahedralizer, while faces labeled as

shell are handled by an offsetter that generates a thin volumetric mesh with appropriate physical thickness.

2.2 Solid/Shell Heuristics in Mesh Repairing

Mesh repairing aims at converting defective triangle soups into watertight manifolds by, for instance, filling gaps [Ju 2004; Liepa 2003] and removing self-intersections [Attene 2014; Campen and Kobbelt 2010; Guo and Fu 2024; Zhou et al. 2016]. Comprehensive surveys are provided by Attene et al. [2013] and Ju [2009]. The question of whether a triangle belongs to a shell or solid often comes up as a subproblem in a repairing pipeline. For instance, wrapping-based methods [Chen et al. 2023; Portaneri et al. 2022] often impose the assumption that all faces should be labeled as solid, similar to tetrahedral meshing techniques, but similarly suffer from the problem of removing internal structures. The method by Huang et al. [2020] indirectly determines face labels by checking whether each element connects to infinity through a voxelized representation of the shape. The closest method to ours is the method proposed by Zheng et al. [2024]. Their pipeline uses ray casting to estimate the visibility of each face and converts this visibility information into an openness score. Faces judged open are offset, all geometry is inserted into a BSP tree, and a global graph cut is used to extract a watertight surface. Because the test hinges on visibility, shell structures hidden behind other geometry can be misclassified (see Fig. 3 and Fig. 13).

Our approach differs in two key ways. First, we rely on the visibility-independent generalized winding number (GWN), eliminating occlusion-related errors. Second, solid–shell segmentation can depend on semantics or be genuinely ambiguous, so a fully automatic method may choose an unintended interpretation with no easy recourse for the user. We let users add "inside" hints, treating them as soft constraints when optimizing the face labels and providing an intuitive mechanism for correcting ambiguous regions.

2.3 Winding Numbers for Inside/Outside Reasoning

The generalized winding number (GWN) introduced by Jacobson et al. [2013] determines whether a point lies inside or outside an arbitrary triangle soup. Most applications assume the entire mesh encloses a uniform solid region, using the GWN for isosurface extraction via marching cubes [Lorensen and Cline 1987] or dual contouring [Ju et al. 2002], or as the basis for "fuzzy" CSG operations [Barill et al. 2018]. We discard this uniform-solid assumption: our method removes faces not intended to enclose volume, yielding a cleaner set of solid faces whose induced GWN field is better suited to downstream contouring, CSG, and simulation. Fast-evaluation schemes for the GWN, such as that of Barill et al. [2018], can be incorporated into our method for improved speed.

The winding number has also been employed in applications outside of meshing, such as coloring vector sketches [Scrivener et al. 2024], reconstruction [Chen et al. 2024] and has inspired work like computing generalized signed distances [Feng et al. 2023]. Another line of work [Lin et al. 2024; Metzer et al. 2021; Xu et al. 2023] has used the GWN to orient point clouds. Our method is similar in spirit, but has a different use case. We optimize a scalar variable attached to each face of a triangle mesh, determining how that face contributes to the GWN field. This optimization process results in a set of solid faces for which the GWN field more closely approximates that of an ideal closed solid object, so downstream volumetric meshing with homogeneous solid assumptions is more well-behaved.

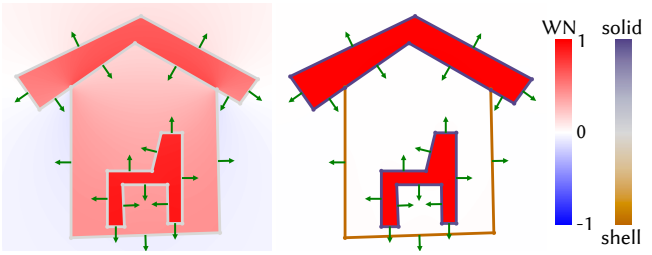


Fig. 5. Generalized winding number field of a collection of segments before (left) and after (right) the optimization. Segments are colored by the tendency of being classified as solid or shell.

3 METHOD

The input to our method is a triangle mesh $\mathcal{M} = (\mathbf{V}, \mathbf{F})$, comprising a set of vertex positions $\mathbf{V}$ and a set of faces $\mathbf{F}$. Our method imposes no constraints on the input topology and is capable of handling non-manifold elements or even a soup of triangles. We do, however, assume that each face has the orientation intended by the artist. The output of our method is a set of (soft) labels $\boldsymbol{\ell} = \{\ell_f\}$ where each label $\ell_f \in [0, 1]$ indicates whether the face f is a *shell* (0) or a *solid* (1) component. These soft labels can then be thresholded to achieve the final binary labels.

We cast the labeling problem as an energy optimization, where the labels $\boldsymbol{\ell}$ are the optimization variables, and the energy can optionally incorporate user-controllable terms. Without user guidance, our method can serve as an automatic tool to provide solid/shell labels. But, in general, the label for a face can be ambiguous and can depend on the semantics of the object (see Fig. 2). We thus provide additional user-controllable terms to manipulate the outcome of the optimization.

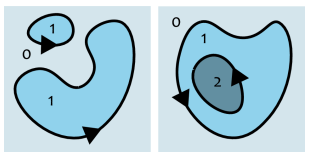
3.1 Energy

We obtain face labels by solving an unconstrained optimization problem

$$\boldsymbol{\ell} = \arg\min_{\boldsymbol{\ell}} \; E_b(\boldsymbol{\ell}) + \alpha E_u(\boldsymbol{\ell}) + \beta E_s(\boldsymbol{\ell}), \tag{1}$$

where $\alpha, \beta \in \mathbb{R}^+$ are positive scalars to control the balance between these terms. E_b encourages the resulting face labels to induce well-separated interior/exterior regions, E_u incorporates user controls into the optimization, and E_s encourages smoothness across the labels $\boldsymbol{\ell}$ within the same surface patch.

Binary Winding Number Term. We observe that artist-created shapes often do not contain too many "irrelevant" components, meaning that most triangles in a mesh are created for a purpose.



With this observation in mind, we design an energy which, by default, preserves nested interior structures (e.g., furniture inside a

house, see Fig. 5). We draw inspiration from the globally injective parameterization [Du et al. 2021] which detects nested structures using the *winding number*. Specifically, the winding number field of a set of closed, non-self-intersecting curves will be piecewise constant 0/1 if the curves neither overlap nor have inverted orientations. This motivates our design of a binary winding number energy E_b which encourages the winding number field induced by the labeling to be either 0 or 1

$$E_b(\boldsymbol{\ell}) = \frac{1}{|\mathcal{P}|} \sum_{\mathbf{p}\in\mathcal{P}} \big(w(\mathbf{p},\boldsymbol{\ell})(1 - w(\mathbf{p},\boldsymbol{\ell})\big)^2 \tag{2}$$

with

$$w(\mathbf{p},\boldsymbol{\ell}) = \frac{1}{4\pi} \sum_{f} \Omega_f(\mathbf{p})\ell_f, \tag{3}$$

where $\mathcal{P}$ denotes the set of sampled points we use to evaluate the energy, $|\mathcal{P}|$ denotes the number of samples, Ω_f is the solid angle of triangle f with respect to point $\mathbf{p}$, and ℓ_f is the face label for face f bounded between $0 \le \ell_f \le 1$. Intuitively, one can think of $w(\mathbf{p})$ as a *weighted* generalized winding number [Jacobson et al. 2013] with options to ignore shell faces ($\ell_f = 0$). The binary winding number energy E_b has two global minima when $w(\mathbf{p}) = 0$ or 1, promoting a binary winding number field as the result of minimization.

User-Control Term. However, this energy formulation is motivated by the idealized scenario in [Du et al. 2021] designed to promote global injectivity. When applied to 3D triangle soups that may contain various defects and non-manifold elements, minimizing E_b alone does not always produce the results users expect. Moreover, geometric cues by themselves (i.e., E_b alone) are insufficient to achieve the desired solid/shell classifications (see Fig. 2). To incorporate user preference in the labeling process, we introduce another energy term which allows users to specify the target winding number $t_{\mathbf{q}} \in \{0, 1\}$ for a sparse set of spatial locations $\mathbf{q} \in C$, indicating whether the point $\mathbf{q}$ should lie in the interior or exterior of the model. Specifically, we define the user control energy E_u as

$$E_u(\boldsymbol{\ell}) = \frac{1}{|C|} \sum_{\mathbf{q}\in C} \|w(\mathbf{q},\boldsymbol{\ell}) - t_{\mathbf{q}}\|^{\gamma}, \tag{4}$$

where $|C|$ denotes the number of user-specified points and γ is a positive even integer, controlling how sensitive this term is to small deviations from the target winding number $t_{\mathbf{q}}$. The higher the power, the more tolerant this term becomes to small errors. Empirically, we set the default $\gamma = 4$ which gives us the easiest way to control the output labels (see Sec. 4.3).

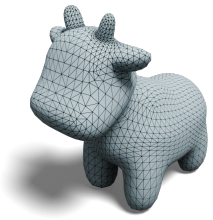
Smoothness Term. As the control of E_u is local, using only E_u often requires more computation time to propagate label adjustments or specifying more constrained locations C. We therefore add a smoothness energy E_s on face labels $\boldsymbol{\ell}$ to encourage connected pieces to share the same label.

$$E_s(\boldsymbol{\ell}) = \frac{1}{A} \sum_{i,j\in\mathcal{N}_F} a_{ij}\|\ell_i - \ell_j\|^2, \tag{5}$$

where we use $\mathcal{N}_F$ to indicate neighboring faces which share an edge in-between, a_{ij} denote the barycentric edge area, and $A = \sum_{ij} a_{ij}$ is the sum of the areas. The combination and three terms forms our energy (see Eq. 1). We set $\alpha = 10^3$, $\beta = 10$ as the default parameter throughout the experiments.

3.2 Implementation

We implement our method in Python, utilize PyTorch library [Paszke et al. 2017] for auto-differentiation, and optimize our energy with the Adam optimizer [Kingma and Ba 2015]. We evaluate the binary winding number loss E_b on points $\mathcal{P}$ sampled near the surface. Specifically, we uniformly sample points (by default 64^3) that have distance less than ϵ away from the mesh, where ϵ is set to be 10% of the longest edge of the mesh bounding box.



As the winding number computation $\Omega_f(\mathbf{p})$ for $\mathbf{p} \in \mathcal{P}$ are constant throughout the optimization, we cache the result of $\{\Omega_f(\mathbf{p})\}$ as a single matrix to accelerate the optimization, leading to about 10 milliseconds runtime per optimization step on a mesh with ~10,000 faces, using GPU acceleration on an Apple M4 Pro chip. The models presented in this paper typically take 500 to 5000 iterations until convergence, with convergence determined by the gradient norm. We report the runtime for a "Spot" mesh (see the inset) at different resolutions in Table 1.

Table 1. Summary of runtime for initialization, per-iteration and total convergence duration on a mesh at different resolutions.

#Faces	*Initialization Time*	*Per-iteration Time*	*Convergence Time*
5K	1.7 s	8.3 ms	14.7 s
10K	3.0 s	12.7 ms	21.8 s
20K	8.5 s	32.7 ms	48.2 s
40K	32.7 s	91.6 ms	122.1 s

We notice that some meshes have big triangles slicing through multiple solid/shell regions (like a ground plane of a house scene). This issue prevents users from getting desired volumetric meshes because neither assigning solid nor shell to the problematic face yields a user-desired result. We thus apply a mesh arrangement [Campen and Kobbelt 2010; Guo and Fu 2024] algorithm as a preprocessing step to partition intersecting triangles into smaller fragments before running our algorithm.

We visualize the result of our method with Polyscope [Sharp et al. 2019]. We allow users to add constraint points in Eq. 4 by clicking points in our preliminary interface and then specify whether this point belongs to an interior region or not. Due to the efficiency of our method, one can interactively see the result updated during optimization (please see the supplementary video for a demo usage of our implementation).

After obtaining the face labels $\boldsymbol{\ell}$, for the solid part, we extracted a closed surface mesh from the tetrahedral mesh computed with TetWild [Hu et al. 2018]. For the shell part, we extrude shells with a user-specified thickness τ using a robust topological offset algorithm [Zint et al. 2024], where τ is by default 1% of the longest edge of the mesh bounding box. Next, we take the union of the two closed surfaces with a robust mesh boolean code [Cherchi et al.

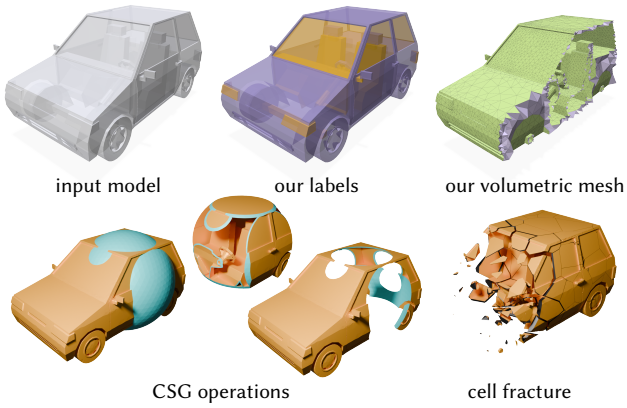


Fig. 6. The face labels optimized by our method can assist volumetric meshing (green). This mesh is ready for boolean operations (e.g., intersection, union, subtraction) with the method by [Cherchi et al. 2022], and volumetric simulation such as the fractures. See the video at 02:20. ©Roblox

Fig. 7. Left: A single constraint guides the optimization to classify a corrupted bunny (10 percent of faces removed) as fully solid. Right: A pipe modeled as a union of cylinders is classified as pure shell after placing some constraints. ©Roblox

Fig. 8. Incrementally adding point constraints can alter whether faces are labeled as solid or shell. Purple spheres denote interior constraints—points designated to lie within the solid—whereas yellow spheres denote exterior constraints—points designated to lie outside it. By default, we convert the soft labels into binary labels by applying a threshold of 0.5. ©Roblox

Fig. 9. Our method allows one to control face labeling results by setting different configurations of the guidance point (colored spheres). We convert the soft labels into binary labels by applying a threshold of 0.8. ©Roblox

2022]. Finally, we use TetWild [Hu et al. 2018] to obtain our output tetrahedral mesh, that is ready for solid modeling and simulation applications as shown in Sec. 4. Moreover, one could apply the method by Attene et al. [2009] to extract a closed manifold triangle mesh from our output to support surface computations.

4 RESULTS

Our method outputs a set of face labels indicating solid and shell faces. We show that these labels can benefit off-the-shelf algorithms for constructing volumetric meshes, with unambiguous definitions of inside and outside based on users' specifications. With such user-defined interior/exterior definitions, one can ensure that the simulation behavior aligns with the semantic properties of an object (see Fig. 10) and can perform boolean operations without the need of relying on extra tools for "guessing" where the interior region is located (see Fig. 6).

Our method enables user control through point-based constraints that specify whether locations should be inside or outside the object. This approach can obtain a wide variety of user-desired segmentations by placing constraints appropriately. When all specified points are constrained to be outside, our energy function has a unique global minimum that labels every face as shell (provided there is at least one outside constraint). For instance, in Fig. 7, our optimization produces an all-shell result for a watertight mesh representing pipes. When a constraint specifies that a point lies inside the object, the optimization encourage face labels to be solid when doing so helps achieve a winding number of 1 at that point. Fig. 7 shows a watertight mesh with a randomly deleted subset of faces, where a single inside constraint produces an all-solid segmentation. While classification is straightforward when meshes contain only solid or shell faces, in the wild meshes often combine faces that bound solids with others that represent thin shell surfaces. Our method provides flexibility to achieve diverse segmentations on wild meshes. Fig. 8 shows how adding constraints incrementally modifies face labeling until the desired result is achieved (bracket and light source as solid, light bulb as shell). These interactions allows one to place different constraint configurations to achieve different solid/shell segmentations (Fig. 9). Fig. 10 demonstrates the importance of user interaction in the labeling process. Given a closed manifold representing a coke can—which most heuristics would classify as solid (e.g., [Zheng et al. 2024])—our method allows users to specify a thin shell result, producing simulation behavior that better matches the expected physical properties of an actual coke can.

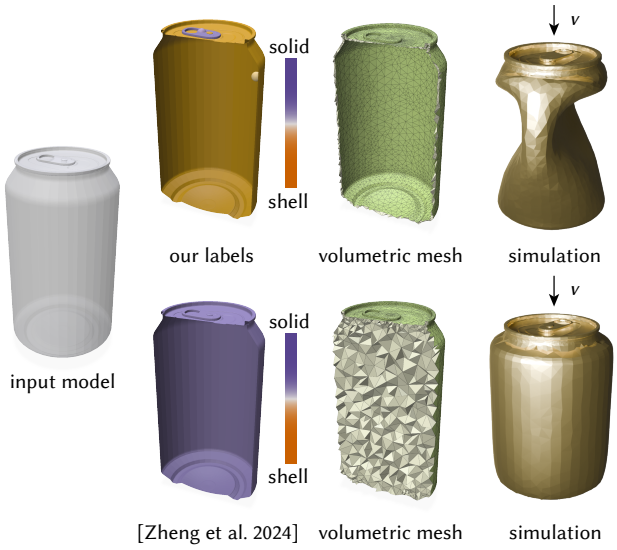


Fig. 10. We present solid/shell labels (second column) and the corresponding volumetric mesh (third column) produced by our method (top) with two user-added constraints and the (all-solid) labeling from [Zheng et al. 2024] (bottom). The last column shows a non-inverting, neo-Hookean elasticity simulation with contact [Li et al. 2020] imposed on the two volumetric meshes, where we fix the bottom 5% and move the top 5% of the can downwards to compress the can. Making the can filled with solid yields unexpected simulation behavior (bottom right), contrasting our method which successfully produce the compression bending behavior (top right). See the video at 03:40. We obtain the model from [Chern et al. 2018].

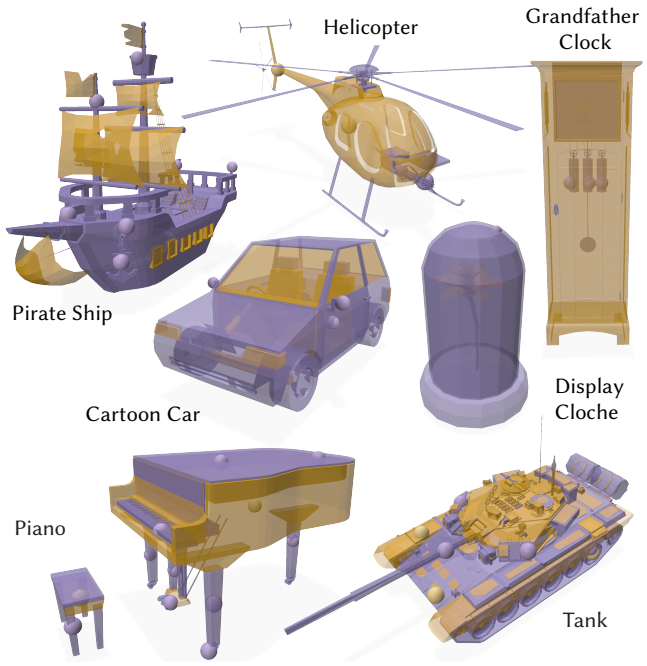


Fig. 11. Seven models benchmark for measuring the labeling effort. Face color represents the target labels: solid faces are colored purple, shell faces are colored yellow. The purple and yellow spheres in the scene represent the interior and exterior constraints in Eq. 4 one needs to obtain the target labels, respectively. ©Roblox

Table 2. Summary of labeling effort for each benchmark mesh in Fig. 11. For each model, we first obtain target face labels through manual face selection by a user. We report: (i) the number of constraints added to our optimization to achieve the target segmentation, (ii) the number of faces labeled as shell/solid, and (iii) the number of face components containing at least one shell/solid label. This table demonstrates that using our tool requires less effort compared to manual selection, by comparing the number of constraints needed to achieve target labels.

Model	*#Constraints*	*#Faces*	*#Components*
Cartoon Car	3	480/2764	15/14
Display Cloche	1	1610/496	32/6
Grandfather Clock	4	1245/2331	15/36
Helicopter	4	5110/8347	63/98
Pirate Ship	8	883/4086	29/52
Piano	12	669/2355	61/122
Tank	7	4105/4903	326/217

4.1 Comparison to Explicit Labeling

To evaluate our method, we build a preliminary interface that enables users to provide hints by specifying whether selected points should be classified as *inside* or *outside* the object. Table 2 quantifies the user effort required to obtain manually-created target labels for a set of seven benchmark meshes (see Fig. 11) and compares our method against two baselines. The first baseline represents individual face labeling, where each face is selected separately. The second baseline approximates practical selection mechanisms such as face component selection or flood filling by counting the number of face components that contain at least one shell/solid face. For the second baseline, we pre-process the mesh by merging duplicate vertices to reduce the number of face components, because the meshes have too many face components to make selection mechanisms that rely on mesh connectivity practical. Note that this number represents a lower bound since connected face components do not necessarily correspond to actual shell/solid regions. These comparisons consistently show that our tools requires orders of magnitude less effort, in terms of the number of constraints needed, to achieve manually created target labels.

We further note that our labeling paradigm depends only on whether a point lies inside or outside the object and is independent of the object's discretization, requiring only that the discretization supports the definition of a GWN field. Users therefore need to understand only the inside or outside concept, not details about the geometry representation. By contrast, both baseline methods require awareness of the triangle mesh, and the component-selection approach is especially fragile: it often requires an additional preprocessing step to merge vertices before it is practical.

4.2 Comparison to Prior Arts

Labeling all the faces as solid or shell does not always create a physically meaningful volumetric mesh. In Fig. 12 (left), we demonstrate

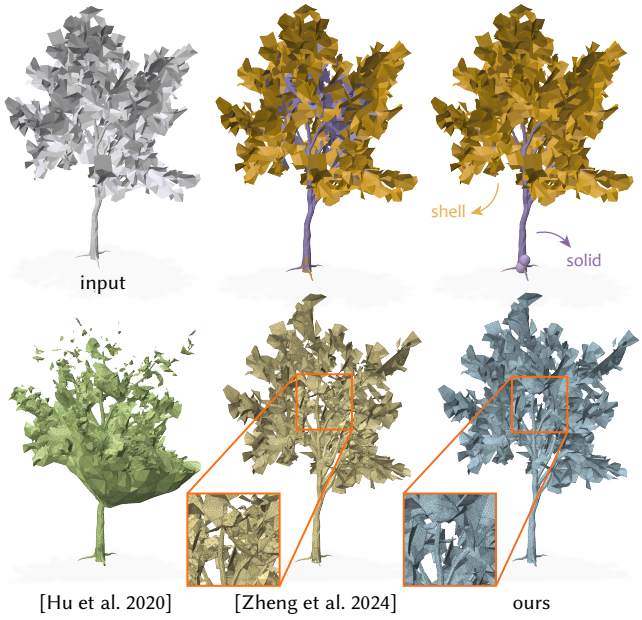


Fig. 12. Given a tree model with a mixture of solid (e.g., trunk) and shell (e.g., leaves) elements (top left), using a visibility-driven classification by [Zheng et al. 2024] is prone to label invisible leaves as solid elements (top middle), yielding unwanted volumes after tetrahedralization (bottom middle). In contrast, our method is able to label trunk faces to be solid, and leaf triangles to be shell (top right), yielding a user-desired volumetric mesh (bottom right). We also include the reconstruction from [Hu et al. 2020] if one assign every face to be solid. ©Roblox

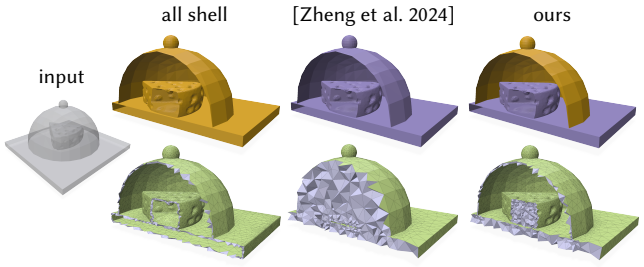


Fig. 13. We show the volumetric mesh (bottom) created by our solid/shell face labels (no user constraint is needed in this case), which successfully labels the lid as shell and the cheese as solid (right), in contrast to all-shell labeling (left) and the visibility-driven labeling by [Zheng et al. 2024] (middle).

a naive uniform solid labeling of the tree leads to artifacts when reconstructing a volumetric mesh based on the method by [Hu et al. 2020]. Labeling based on heuristics, such as visibility [Zheng et al. 2024], tends to label closed object as solid, which is prone to unexpected physical behavior or deleting invisible structures. In Fig. 10, solid/shell labeling method based on visibility [Zheng et al. 2024] tends to label the can as all-solid, while with easy user-added constraints, our method gives the capability to label the body as shell and the zip as solid. These two different labels lead to different

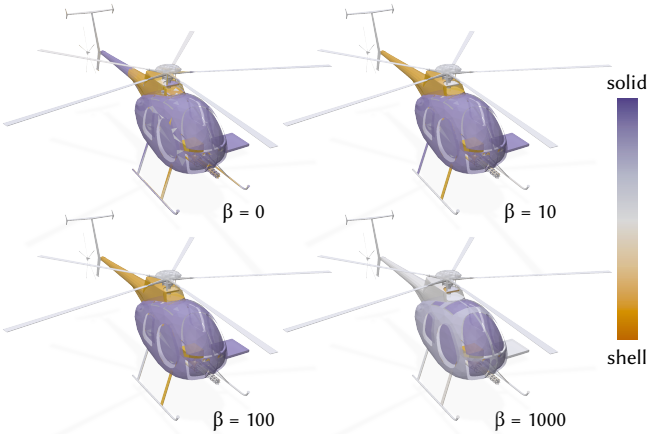


Fig. 14. Comparison on different β values in E_s on a helicopter model without user control E_u. Each optimization is allotted a maximum of 2 minutes; runs that do not converge within this time are terminated. ©Roblox

physical behavior when performing a compression simulation. For models with self-occlusion, the visibility-based labeling tends to remove important, but invisible, structures (e.g. Fig. 13) or assigning solid labels to invisible shell structures (see Fig. 12). In contrast, our method addresses the above issues by using a non-visibility-based winding number field as a prior, augmented with user guidance, to allow control over different inside/outside labelings (e.g., in Fig. 9) to be aligned with user's intent.

4.3 Ablations

Choice of α and β. The larger α is, the more optimization will prioritize the user constraints E_u. The larger β is, the smoother $\boldsymbol{\ell}$ will be on the connected components. However, β cannot be too large, otherwise the total energy will be dominated by E_s rather than E_b. Consequently, the optimization exhibits slow convergence while still maintaining a relatively high E_b value. Fig. 14 compares the labeling results with different choice of β on a helicopter model. In our paper, we set $\alpha = 10^3$, $\beta = 10$ for all the experiments.

Choice of γ. We perform an ablation study on how γ in Eq. 4 influences the convergence behavior of labels $\boldsymbol{\ell}$ in optimization. In Fig. 15, we demonstrate that for a given mesh with the same user-specified constraint, $\gamma = 4$ leads to an immediate convergence on $\boldsymbol{\ell}$ towards either 0 or 1, while $\gamma = 2$ fails to converge during the optimization with non-decreasing gradient. As even higher exponent can lead to numerical instability, we choose $\gamma = 4$ in Eq. 4 for better optimization behavior and more effective solid/shell label classification.

Influence of each energy term E_b, E_s and E_u. The necessity of E_b is to encourage a binary winding number field, so if we keep E_b alone, the optimization will still give us a labeling that induces a binary field, which is similar to the situation when the input mesh is a triangle soup, and E_s is zero ($\beta = 0$ in Fig. 14). In that case, the resulting labels may not match the artist intent. Incorporating user controls with E_b and E_s in the energy will take artist intent into consideration – connected faces are more likely to share similar

Fig. 15. Comparison on the difference of convergence behavior between $\gamma = 2$ (left) and $\gamma = 4$ (right) in Eq. 4. ©Roblox

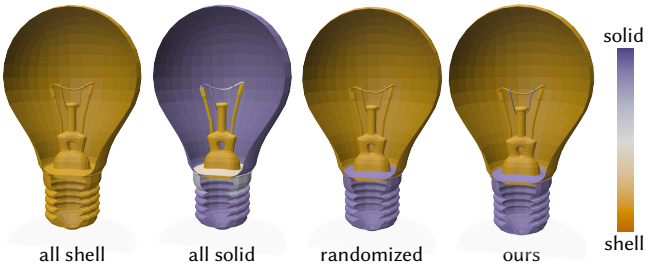


Fig. 16. Comparison on different initialization on labels ℓ. From left to right, initializing all the labels as shell ($\ell = 0$), as solid ($\ell = 1$), randomized initialization and as the midpoint between solid and shell ($\ell = 0.5$). We obtain the model from polyhaven.com by Josh Dean licensed under CC0.

labels. Finally, E_u is more specific for user-desired semantics. For instance, the top middle of Fig. 8 shows the result without E_u, and as the user controls are added increasingly, the labels can be more specialized to the user intent.

Initialization on labels ℓ. Initializing all the labels as shell ($\ell = 0$) will force ℓ to stay at zero, as the GWN field is already binary and ℓ is also smooth. The effect of initializing all the labels as solid ($\ell = 1$) relies on the initial GWN field: if the field is already binary, ℓ will stay at 1, otherwise it will continue to be optimized until a binary field appears. Random initialization almost always provides a non-binary initial GWN field, thus, ℓ will go wherever the optimization converges. Therefore, all solid and random initialization are both reasonable alternatives, but may converge to different results without user control, and often take longer time to converge. Fig. 16 shows different labeling results with different ℓ initializations on a light bulb model. In our paper, the soft labels ℓ are initialized as 0.5 (in the middle between solid and shell).

5 LIMITATIONS & FUTURE WORK

Our approach lays the groundwork towards an accessible tool that could help turning "in-the-wild" 3D assets into simulation-, solid-modeling-, or 3D-print-ready form. Our strategy requires far less manual efforts (often less than 10 user provided points C) to divide meshes into solid and shell components, in comparison to explicit face selection, which typically requires selecting thousands of faces and familiarity with 3D modeling details such as triangulations, connectivity, selection, and flood-filling. Future work could involve developing a comprehensive and intuitive user interface that enables users to place spheres to designate interior/exterior regions, along with common utilities like mesh selection, undo, and slicing. Collecting a wide range of examples from such an interface could motivate future data-driven methods to learn how to automatically place those guiding spheres. Removing the assumption of requiring the input faces to be correctly oriented, possibly by incorporating normal orientation tools [Metzer et al. 2021] or extending the face label searching field to $\ell_f \in [-1, 1]$, could extend the reach of our method to more types of mesh defects. Allowing face orientations to change during optimization could achieve a broader range of user intents, such as turning a sphere inside out to create a volumetric mesh that extends towards infinity but is hollow inside. Building on top of the generalization of winding number to other representations [Barill et al. 2018], extending our methods to different representations (e.g., point clouds) could further impact other fields, such as robotics and 3D sensing/reconstruction.

ACKNOWLEDGMENTS

We would like to extend our thanks to the Alpha Strike team of Roblox for the use of their models in our examples. We would like to thank members of the Geometric Computing Lab at New York University; Michael Tao for method discussion; Daniel Zint for instructions in running the topological offsets code. This work was also partially supported by the NSF grant OAC-2411349.

REFERENCES

Pierre Alliez, Clément Jamin, Laurent Rineau, Stéphane Tayeb, Jane Tournois, and Mariette Yvinec. 2024. 3D Mesh Generation. In *CGAL User and Reference Manual* (6.0.1 ed.). CGAL Editorial Board. https://doc.cgal.org/6.0.1/Manual/packages.html#PkgMesh3

Marco Attene. 2014. Direct repair of self-intersecting meshes. *Graph. Model.* 76, 6 (2014), 658–668.

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

Marco Attene, Daniela Giorgi, Massimo Ferri, and Bianca Falcidieno. 2009. On converting sets of tetrahedra to combinatorial and PL manifolds. *Computer Aided Geometric Design* 26, 8 (2009), 850–864.

Gavin Barill, Neil G Dickson, Ryan Schmidt, David IW Levin, and Alec Jacobson. 2018. Fast winding numbers for soups and clouds. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–12.

Marcel Campen and Leif Kobbelt. 2010. Exact and Robust (Self-)Intersections for Polygonal Meshes. *Comput. Graph. Forum* 29, 2 (2010), 397–406.

Hongyi Cao, Gang Xu, Renshu Gu, Jinlan Xu, Xiaoyu Zhang, and Timon Rabczuk. 2023. A Parallel Feature-preserving Mesh Variable Offsetting Method with Dynamic Programming. *CoRR* abs/2310.08997 (2023). https://doi.org/10.48550/ARXIV.2310.08997 arXiv:2310.08997

Hongyi Cao, Gang Xu, Renshu Gu, Jinlan Xu, Xiaoyu Zhang, Timon Rabczuk, Yuzhe Luo, and Xifeng Gao. 2024. Robust and Feature-Preserving Offset Meshing. *CoRR* abs/2412.15564 (2024). https://doi.org/10.48550/ARXIV.2412.15564 arXiv:2412.15564

Hanyu Chen, Bailey Miller, and Ioannis Gkioulekas. 2024. 3D Reconstruction with Fast Dipole Sums. *ACM Trans. Graph.* 43, 6 (2024), 192:1–192:19. https://doi.org/10.1145/3687914

Zhen Chen, Zherong Pan, Kui Wu, Etienne Vouga, and Xifeng Gao. 2023. Robust Low-Poly Meshing for General 3D Models. *ACM Trans. Graph.* 42, 4 (2023), 119:1–119:20.

Gianmarco Cherchi, Fabio Pellacini, Marco Attene, and Marco Livesu. 2022. Interactive and Robust Mesh Booleans. *ACM Trans. Graph.* 41, 6 (2022), 248:1–248:14.

Albert Chern, Felix Knöppel, Ulrich Pinkall, and Peter Schröder. 2018. Shape from metric. *ACM Trans. Graph.* 37, 4 (2018), 63.

Lorenzo Diazzi and Marco Attene. 2021. Convex polyhedral meshing for robust solid modeling. *ACM Transactions on Graphics (TOG)* 40, 6 (2021), 1–16.

Lorenzo Diazzi, Daniele Panozzo, Amir Vaxman, and Marco Attene. 2023. Constrained Delaunay Tetrahedrization: A Robust and Practical Approach. *ACM Transactions on Graphics (TOG)* 42, 6 (2023), 1–15.

Xingyi Du, Danny M. Kaufman, Qingnan Zhou, Shahar Z. Kovalsky, Yajie Yan, Noam Aigerman, and Tao Ju. 2021. Optimizing global injectivity for constrained parameterization. *ACM Trans. Graph.* 40, 6 (2021), 260:1–260:18.

Nicole Feng, Mark Gillespie, and Keenan Crane. 2023. Winding Numbers on Discrete Surfaces. *ACM Trans. Graph.* 42, 4 (2023), 36:1–36:17. https://doi.org/10.1145/3592401

Jia-Peng Guo and Xiao-Ming Fu. 2024. Exact and Efficient Intersection Resolution for Mesh Arrangements. *ACM Trans. Graph.* 43, 6 (2024), 165:1–165:14.

Yixin Hu, Teseo Schneider, Bolun Wang, Denis Zorin, and Daniele Panozzo. 2020. Fast tetrahedral meshing in the wild. (2020).

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

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). 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 Trans. Graph.* 23, 3 (2004), 888–895.

Tao Ju. 2009. Fixing Geometric Errors on Polygonal Models: A Survey. *J. Comput. Sci. Technol.* 24, 1 (2009), 19–29.

Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. 2002. Dual contouring of hermite data. *ACM Trans. Graph.* 21, 3 (July 2002), 339–346. https://doi.org/10.1145/566654.566586

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.).

Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy R Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M Kaufman. 2020. Incremental potential contact: intersection-and inversion-free, large-deformation dynamics. *ACM Trans. Graph.* 39, 4 (2020), 49.

Peter Liepa. 2003. Filling Holes in Meshes. In *Eurographics Symposium on Geometry Processing*, Leif Kobbelt, Peter Schroeder, and Hugues Hoppe (Eds.). The Eurographics Association. https://doi.org//10.2312/SGP/SGP03/200-206

Siyou Lin, Zuoqiang Shi, and Yebin Liu. 2024. Fast and Globally Consistent Normal Orientation based on the Winding Number Normal Consistency. *ACM Trans. Graph.* 43, 6, Article 189 (Nov. 2024), 19 pages. https://doi.org/10.1145/3687895

William E. Lorensen and Harvey E. Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Comput. Graph.* 21, 4 (Aug. 1987), 163–169. https://doi.org/10.1145/37402.37422

Gal Metzer, Rana Hanocka, Denis Zorin, Raja Giryes, Daniele Panozzo, and Daniel Cohen-Or. 2021. Orienting point clouds with dipole propagation. *ACM Trans. Graph.* 40, 4 (2021), 165:1–165:14. https://doi.org/10.1145/3450626.3459835

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. (2017).

Cédric Portaneri, Mael Rouxel-Labbé, Michael Hemmer, David Cohen-Steiner, and Pierre Alliez. 2022. Alpha wrapping with an offset. *ACM Trans. Graph.* 41, 4 (2022), 127:1–127:22.

Daniel Scrivener, Ellis Coldren, and Edward Chien. 2024. Winding Number Features for Vector Sketch Colorization. *Comput. Graph. Forum* 43, 5 (2024), i–x. https://doi.org/10.1111/CGF.15141

Nicholas Sharp et al. 2019. Polyscope. www.polyscope.run.

Hang Si. 2015. TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator. *ACM Trans. Math. Softw.* 41, 2 (2015), 11:1–11:36.

Rui Xu, Zhiyang Dou, Ningna Wang, Shiqing Xin, Shuangmin Chen, Mingyan Jiang, Xiaohu Guo, Wenping Wang, and Changhe Tu. 2023. Globally Consistent Normal Orientation for Point Clouds by Regularizing the Winding-Number Field. *ACM Trans. Graph.* 42, 4, Article 111 (July 2023), 15 pages. https://doi.org/10.1145/3592129

Zhongtian Zheng, Xifeng Gao, Zherong Pan, Wei Li, Peng-Shuai Wang, Guoping Wang, and Kui Wu. 2024. Visual-Preserving Mesh Repair. *IEEE Transactions on Visualization & Computer Graphics* 30, 09 (2024), 6586–6597.

Qingnan Zhou, Eitan Grinspun, Denis Zorin, and Alec Jacobson. 2016. Mesh arrangements for solid geometry. *ACM Trans. Graph.* 35, 4 (2016), 39:1–39:15. https://doi.org/10.1145/2897824.2925901

Daniel Zint, Zhouyuan Chen, Yifei Zhu, Denis Zorin, Teseo Schneider, and Daniele Panozzo. 2024. Topological Offsets. *arXiv preprint arXiv:2407.07725* (2024).