Visual-Preserving Mesh Repair

Zhongtian Zheng, Xifeng Gao, Zherong Pan, Wei Li, Peng-Shuai Wang, Guoping Wang*, Kui Wu*

Abstract—Mesh repair is a long-standing challenge in computer graphics and related fields. Converting defective meshes into watertight manifold meshes can greatly benefit downstream applications such as geometric processing, simulation, fabrication, learning, and synthesis. In this work, by assuming the model is visually correct, we first introduce three visual measures for visibility, orientation, and openness, based on ray-tracing. We then present a novel mesh repair framework incorporating visual measures with several critical steps, i.e., open surface closing, face reorientation, and global optimization, to effectively repair meshes with defects (e.g., gaps, holes, self-intersections, degenerate elements, and inconsistent orientations) and preserve visual appearances. Our method reduces unnecessary mesh complexity without compromising geometric accuracy or visual quality while preserving input attributes such as UV coordinates for rendering. We evaluate our approach on hundreds of models randomly selected from ShapeNet and Thingi10K, demonstrating its effectiveness and robustness compared to existing approaches.

Index Terms—Mesh repairing, geometry processing.

◆

1 INTRODUCTION

Meshes in games created by modelers often prioritize visual appearance over geometric and topological correctness, leading to various defects like gaps, holes, self-intersections, singular elements, and inconsistent orientations [1]. Additionally, raw data from online repositories, like ShapeNet [2], may contain quality issues like duplicated faces, self-intersections, and non-manifold elements. Due to these issues, meshes with such defects become invalid for downstream applications. Therefore, developing a robust mesh repair pipeline is essential. Three key properties should be pertained during mesh repair, so it can facilitate the downstream applications. First, the mesh should be *manifold* to calculate differential quantities such as normals and curvatures. Second, the mesh should be *watertight* to have a well-defined interior and exterior volume. Finally, the mesh should be repaired with *minimal modification*, preserving the sharp features and UVs of the input mesh as much as possible. These properties are crucial for simulation, 3D printing, geometric Boolean operators, and learning-related applications such as shape analysis and synthesis [3], [4]. Besides the three features, *robustness* and *efficiency* are desirable features that must be considered in mesh repair algorithms.

Despite the considerable research efforts invested in mesh repair, the challenge of reliably converting a problematic mesh into a watertight manifold mesh, with the input details and UVs preserved as much as possible, remains unsolved. Broadly speaking, existing mesh repair approaches can be categorized into two groups. Local approaches aim at addressing individual defects by analyzing the geometry and topology of a local sub-mesh, as done in [5], [6], [7], [8]. Local approaches can repair sparse defects while preserving large portions of visual features, but they

- This work was done when Z. Zheng was an intern at LightSpeed Studios.
- Z. Zheng, P. Wang, and G. Wang are with Peking University.
- X. Gao, Z. Pan, W. Li, and K. Wu are with LightSpeed Studios.
- Corresponding Authors: Kui Wu, E-mail: kwwu@global.tencent.com, Guoping Wang, E-mail: wgp@pku.edu.cn

TABLE 1: *Summary of related work. In each column, green indicates the preferred property; red means the method lacks the preferred property; yellow means the method has some, but not all aspects of the preferred property. (Pres. for preserving)*

Method	Watertight	Face #	Pres. shape	Pres. UV	Memory	Speed
PolyMender (PM) [12]	Yes	Low	Medium	No	Low	Fast
TetWild (TW) [10]	Yes	Low	Low	No	Medium	Slow
fTetWild (fTW) [15]	Yes	Low	Low	No	Medium	Medium
AlphaWrapping (AW) [13]	Yes	Medium	Medium	No	Low	Fast
ManifoldPlus (MP) [14]	Yes	High	Low	No	Medium	Fast
Takayama et al. (T14) [16]	No	Low	High	Yes	Low	Fast
VisualRepair (VR) [11]	No	Low	High	Yes	Low	Slow
VolumeMesher (VM) [9]	Yes	High	Medium	No	Medium	Fast
Ours	Yes	Low	High	Yes	Medium	Medium

often lack guarantees and may inadvertently introduce new issues, e.g., self-intersections, during the repair process. In recent years, researchers have turned their focus to global techniques for superior robustness. On the downside, however, global methods can violate the minimal modification requirement and oftentimes impair the defectless mesh parts due to global conversions and remeshing, as noted in survey [1]. For example, recent works, VolumeMesher [9] and TetWild [10] utilize a BSP tree to partition the ambient space and close gaps and holes by solving a graph cut problem or using winding numbers to filter out interior and exterior volumes. Although these methods can guarantee watertight meshes, their results are sensitive to mis-oriented input meshes. Indeed, a small mis-orientation or nested structures can lead to drastically different output with undesirable modifications to the visual appearance (Fig. 1). In addition, the BSP tree can introduce a massive number of unnecessary faces that are inherited in the output mesh. On a parallel front, Chu et al. [11] proposed a surface-based method in which patch orientation and connectivity are globally optimized based on visual guidance, leading to output with minimal modification. Unfortunately, their method splits the inconsistent patches, resulting in gaps and non-watertight open surfaces.

This paper proposes a mesh repair pipeline that combines the merits of all prior local and global algorithms. Our work is based on two observations. First, while artists may inevitably create models with defects during the modeling
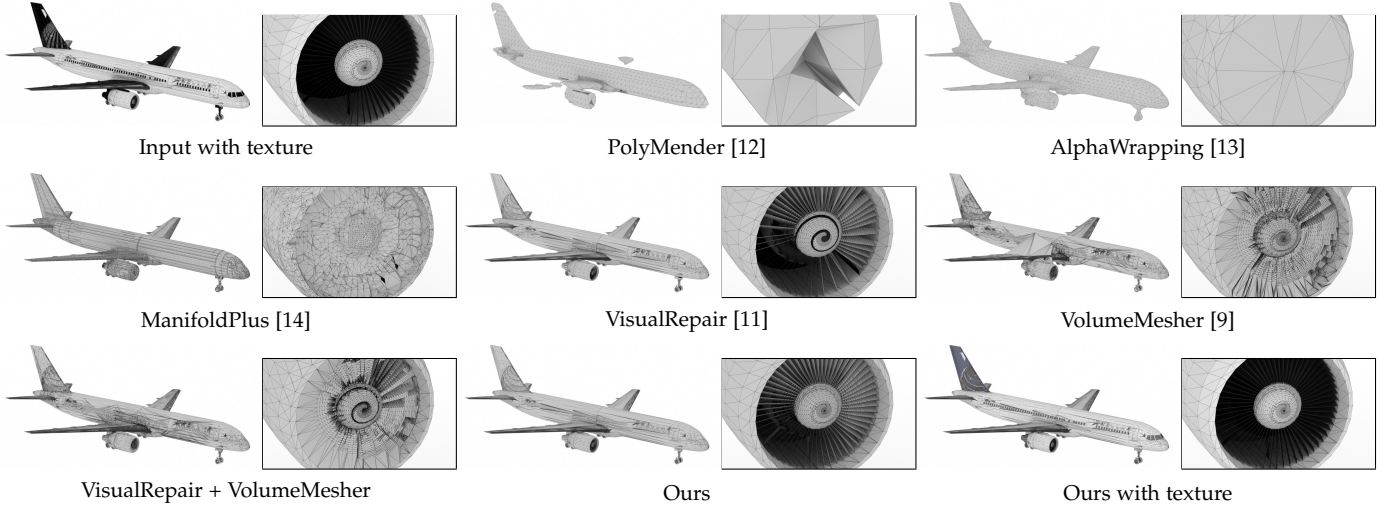
Fig. 1: *Comparison with existing methods: We compare mesh repair algorithms on the textured and mis-oriented input model from ShapeNet [2]. None of the existing works can convert the input mesh into watertight manifold mesh while preserving the textures and input details (see the zoom-in view of the engine).*

process, the output model usually looks visually correct, which can be a valuable measure to guide mesh repair. Second, completely invisible components are not always needed after repair, particularly within the context of graphical applications, including rendering, level of detail (LOD) generation, collision proxy generation, etc. Our key idea is to guide the global graph cut algorithm using local visual cues. Our method consists of three major steps. In our first (local) step, we propose novel visual measures to quantify the visibility, orientation, and openness of each face. We show that these visual measures can be computed efficiently using GPU ray tracing. Next, we rely on the orientation measure to reorient the faces and use the openness measure to identify and close open surfaces. These local adjustments bootstrap the graph cut, providing well-conditioned initial guess and solution space. Finally, inspired by VolumeMesher [9], our global step divides the ambient space into polyhedral cells and graph-cuts the interior cells from the exterior, guided by our visual measures. Compared with existing repair algorithms, our key innovations involve:

- A set of ray-tracing-based visual measures to fix misorientations, detect open surfaces, and guide the graph cut algorithm to produce watertight manifold mesh while preserving visual cues.
- A constrained simplification post-process to remove unnecessary split faces.
- A mesh repair algorithm preserves arbitrary attributes defined on the input mesh, such as UV coordinates.

We highlight the effectiveness and robustness of our method on randomly chosen 1000 models from ShapeNet [2] and 400 models from Thingi10K [17], respectively. Our method outperforms the state-of-the-art in terms of measurements from various aspects such as Hausdorff distance, light field distance (LFD), and PSNR.

2 RELATED WORK

We summarize the differences between our work and previous papers in Table 1. The problem of robustly repairing digital 3D models has been a topic of research for over two decades. For a more in-depth discussion of mesh repair problems and solutions, we refer readers to comprehensive surveys [18], [1]. In general, mesh repair involves remedying the geometric and topological defects from the input mesh, so the output mesh can be used in downstream applications such as mesh processing and simulation. Recently, applications of mesh repair have expanded to 3D printing and learning-related tasks, such as shape analysis and synthesis.

Mesh repairing methods can be broadly classified into two categories: local and global approaches. Local approaches are suitable for input meshes with only sparse defects and remedy the defects by only modifying the mesh structures in a small vicinity. These methods have been widely used to resolve manifold connectivity [19], [20], close gaps [21], [5], fill holes [22], [6], remove degeneracy [7], [23], and remove self-intersection [24], [8], [25]. Since local approaches fix defects locally, they can preserve as many details as possible, but lack guarantees and may introduce new flaws, e.g., self-intersection.

To address the limitations of local mesh repair approaches, researchers have developed several global methods that leverage volumetric representations to distinguish between interior and exterior volumes, using flood-filling [26], [27], line-of-sight information [28], [29], distance diffusion [30], [31], [5], ray-stabbing [32], parity counting [33], [12], morphology [5], [34], etc. These global methods offer greater robustness in resolving complex defects such as gaps and holes while also ensuring high-quality output. As an example, [12] proposed PolyMender, a volumetric method based on the octree structure to patch holes for arbitrary input meshes. However, the sharp feature could be aliased due to the voxel-based representation.

Instead of using regular voxels, recent works [10], [9] partition the space using a binary space partitioning (BSP)
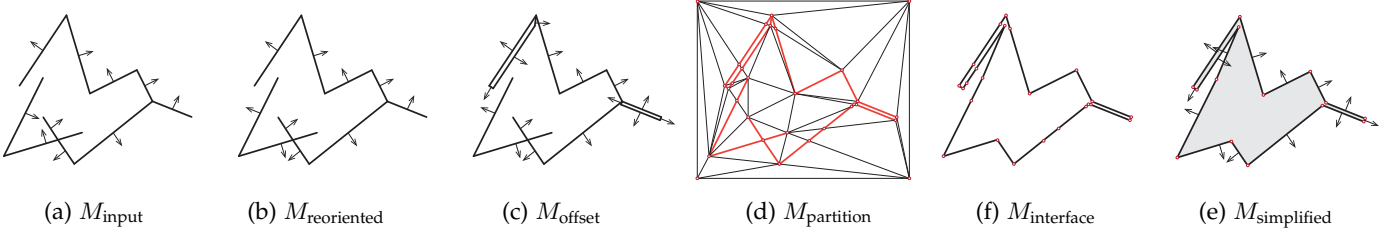
Fig. 2: *Our mesh repair pipeline: Given the input mesh M_{input} (a), we first reorient the faces based on the orientation measure to yield $M_{reoriented}$ (b). Next, we identify the open faces based on the openness measure and add offsets to close the open faces and arrive at M_{offset} (c). Then, we use BSP to partition within the ambient space, getting $M_{partition}$ (d), and solve a graph cut problem based on visibility measure to obtain the interface surface $M_{interface}$ (f), which is further simplified to remove the redundant faces and vertices (red dot), yielding $M_{simplified}$ (e).*

tree [35] to align the polyhedral cells with the input polygons, so that input-aligned outer shell can be obtained by solving a global segmentation problem. For the input mesh containing open surfaces, Tetwild [10] uses winding number [36] to determine the interior and exterior space, while volume mesher [9] solves a minimum graph cut problem to minimize the total area of the output mesh, so as to close the holes. The same graph cut strategy has been used by [37], [38] as well. Unfortunately, these works are based on the assumption that the input faces already have the correct orientations or visibility. But consistent orientations are unavailable in casually collected 3D mesh datasets such as ShapeNet [2]. The wrapping technique presented by [13], [14] provides an alternative approach to repair meshes without relying on face orientation. These methods can produce a strictly enclosing mesh by shifting the faces both inward and outward, but these approaches can corrupt visually sharp features. In contrast, [11] proposes a method that utilizes the rasterization pipeline to evaluate the visual significance of each patch and performs a global optimization to ensure consistent orientation and connectivity of the patches. In cases of inconsistent faces, their method splits them to guarantee a manifold output, creating many gaps and holes. We notice that visual measures based on the rasterization pipeline have been adopted in various prior works [11], [39], [40], [16]. However, such visual measures cannot account for indirect visibility. This issue is resolved in our novel measures via multi-bounce ray-tracing, which is further integrated into graph cut for visual-preserving segmentation. Using GPU ray-tracing, our measures are also faster to compute than rasterization, leading to improved overall efficacy.

3 METHOD

We define a triangle mesh as $M_{\bullet} \triangleq <\mathcal{V}_{\bullet}, \mathcal{F}_{\bullet}>$, where $\mathcal{V}_{\bullet}$ is a set of vertices and $\mathcal{F}_{\bullet}$ is a set of triangles connecting vertices. Our mesh repair algorithm requires the input to be a triangle mesh M_{input}, which is assumed to be visually satisfactory but comes with various geometric and topological defects, including gaps, holes, self-intersection, non-manifold elements, duplicated faces, and inconsistent orientations. Our method generates an output mesh M_{output} that is guaranteed to be manifold and watertight, with as few as possible modifications to the visual appearance of the input. Our mesh repair pipeline is demonstrated in Fig. 2, and we detail each step below.

3.1 Visual Measures

Since our input mesh is visually satisfactory, the visual cues provide strong guidance to our repair algorithm. Therefore, we propose three visual measures to quantify the visibility, orientability, and openness of each face, as illustrated in Fig. 3 (a).
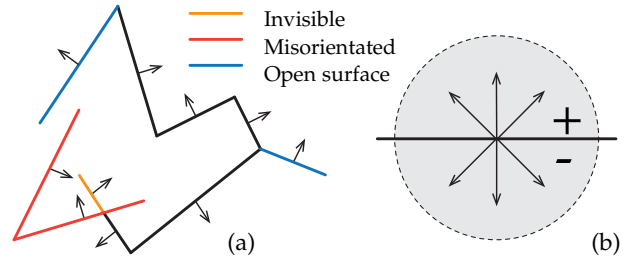

Fig. 3: *Given an input mesh (a) with misoriented faces, invisible faces, and open faces, we sample directions (b) on the unit hemisphere for the positive $(+)$ and negative $(-)$ side of the face, and the arrow indicates the face normal.*

We first uniformly sample each face f_{input}^{i}, where the number of sample points, N_{s}, depends on the face area $A(f_{input}^{i})$:

$$N_{s}(f_{input}^{i}) = \max\left\{ \left\lceil \frac{A(f_{input}^{i})}{A(M_{input})} N_{total} \right\rceil, N_{min} \right\}, \quad (1)$$

where $A(\bullet)$ indicates the surface area of input geometric entity and N_{total} is the total sample number over the surface. We set a minimum sample number N_{min} to avoid under-sampling and we use $\mathcal{S}^{i}$ to indicate the set of all samples on the face f_{input}^{i}, so $|\mathcal{S}^{i}| = N_{s}(f_{input}^{i})$.

At each sample location, we then use uniformly sampling [41] to generate N_{d} directions on the unit hemisphere for positive side $(+)$ and negative side $(-)$ of the face, as shown in Fig. 3 (b). Unlike conventional ray tracing, which shoots rays from the camera, we shoot rays from the sampled location along the sampled direction. When a ray hits the surface, a random direction over the normal hemisphere is picked as the reflection direction. We consider a ray valid if the ray can hit the bounding box of M_{input} within a given number of bounces N_{b}. At location $\mathbf{p} \in \mathcal{S}^{i}$, the total number of valid rays for positive and negative sides of the face are $N_{\mathbf{p}}^{+}$ and $N_{\mathbf{p}}^{-}$, respectively. For example, $\sum_{\mathbf{p} \in \mathcal{S}_{i}}(N_{\mathbf{p}}^{+} + N_{\mathbf{p}}^{-}) > 0$ means face f_{input}^{i} is visible by at least one of the rays.

Utilizing $N_{\mathbf{p}}^+$ and $N_{\mathbf{p}}^-$ computed by ray-tracing, we quantify the probability of f_{input}^i being visible from outside the mesh using the visibility measure defined as:

$$\Phi_{\text{visibility}}(f) = \frac{\max_{\mathbf{p} \in \mathcal{S}^i} \max\{N_{\mathbf{p}}^+, N_{\mathbf{p}}^-\}}{N_{\text{d}}}, \qquad (2)$$

and we further classify f_{input}^i as being visible if $\Phi_{\text{visible}}(f_{\text{input}}^i) > 0.5$. We further quantify the probability that f_{input}^i is consistently oriented via the orientation measure defined as:

$$\Phi_{\text{orientation}}(f) = \begin{cases} \frac{\sum_{\mathbf{p} \in \mathcal{S}^i} (N_{\mathbf{p}}^+ - N_{\mathbf{p}}^-)}{\sum_{\mathbf{p} \in \mathcal{S}^i} (N_{\mathbf{p}}^+ + N_{\mathbf{p}}^-)}, & \text{if } \sum_{\mathbf{p} \in \mathcal{S}^i} (N_{\mathbf{p}}^+ + N_{\mathbf{p}}^-) > 0 \\ 0, & \text{otherwise.} \end{cases} \qquad (3)$$

$\Phi_{\text{orientation}} = -1$ means there is a large chance that the back side face f_{input}^i is much more visible than the front side, and the face should be flipped. Finally, we consider the face an open surface if it has high visibility from both sides, whose probability is quantified by our openness measure:

$$\Phi_{\text{openness}}(f) = \begin{cases} \max_{\mathbf{p} \in \tilde{\mathcal{S}}^i} \frac{\min\{N_{\mathbf{p}}^+, N_{\mathbf{p}}^-\}}{\max\{N_{\mathbf{p}}^+, N_{\mathbf{p}}^-\}} \frac{N_{\mathbf{p}}^+ + N_{\mathbf{p}}^-}{2N_{\text{d}}}, & \text{if } \tilde{\mathcal{S}}^i \neq \varnothing \\ 0, & \text{otherwise.} \end{cases} \qquad (4)$$

where $\tilde{\mathcal{S}}^i \subset \mathcal{S}^i$ indicates the visible samples of the face. We have $\Phi_{\text{openness}}(f_{\text{input}}^i) \in [0,1]$, and a higher measure indicates f_{input}^i is more likely to be an open thin shell.

3.2 Orientation Adjustment

Guided by $\Phi_{\text{orientation}}$, our algorithm locally adjusts the misoriented faces as much as possible, which leads to the better performance of the follow-up global graph cut step. To this end, we first remove the duplicated faces sharing the same vertices. We then group input faces into patches $\mathcal{P}_{\text{input}}^j = \{f_{\text{input}}^{ji}\}$ using a flood fill algorithm, such that faces sharing an edge and having consistent orientations will be grouped into the same patch. Next, we calculate a weighted average orientation measure over each patch $\mathcal{P}^j$, which is defined as:

$$\Phi_{\text{orientation}}(\mathcal{P}_{\text{input}}^j) = \begin{cases} \frac{\sum_{f \in \tilde{\mathcal{P}}^j} A(f) \Phi_{\text{orientation}}(f)}{\sum_{f \in \tilde{\mathcal{P}}^j} A(f)}, & \text{if } \tilde{\mathcal{P}}_{\text{input}}^j \neq \varnothing \\ 0, & \text{otherwise,} \end{cases} \qquad (5)$$

where $\tilde{\mathcal{P}}_{\text{input}}^j \subset \mathcal{P}_{\text{input}}^j$ indicates the visible set of faces in the patch $\mathcal{P}_{\text{input}}^j$. Finally, we flip the patch $\mathcal{P}_{\text{input}}^j$ if $\Phi_{\text{orientation}}(\mathcal{P}_{\text{input}}^j) < 0$. Note that for nearly open patches with high visibility measures from both sides, its orientation measure is close to zero, meaning there is no preference for its orientation. The output of this step is denoted as $M_{\text{reoriented}}$.

3.3 Offsetting Open Surface

For the graph cut algorithm to close open surfaces with minimal visual modification, our method offsets the open faces by a small distance. Although graph cut algorithms can find watertight meshes even without such offset, as done in VolumeMesher [9], it can introduce large unnecessary volumes as demonstrated in Fig. 4 (ab), impairing the visual appearance. Again, our first step is identifying the open surfaces as guided by Φ_{openness}. Unlike the orientation adjustment step, we cannot identify open surfaces in a patchwise manner because a patch may contain inner structures and self-intersections. Instead, we classify each face $f_{\text{reoriented}}^i$ as an open surface if $\Phi_{\text{openess}}(f_{\text{reoriented}}^i) > \epsilon_{\text{openess}}$, where $\epsilon_{\text{openess}}$

is an openness threshold to control the preservation of thin shells. However, offsetting each open face would create too many volumetric cells for the graph cut algorithm, slowing down the overall algorithm. Therefore, after open faces are classified, we group connected, consistent-oriented, open faces into open patches. We offset the vertices on the patch along the negative normal direction with a user-defined distance d_{offset} to create thin volumetric shells, as shown in Fig. 4 (c). The vertex normal is the average normal of adjacent face normals weighted by the face area. Note that, in the case of non-orientable meshes, such as the Mobius strip, grouping neighboring open faces can end up with non-manifold edges with a zero normal vector. In this case, we offset vertices on the non-manifold edge along each adjacent face normal (Fig. 4 (d)). The output of this step is denoted as M_{offset}.
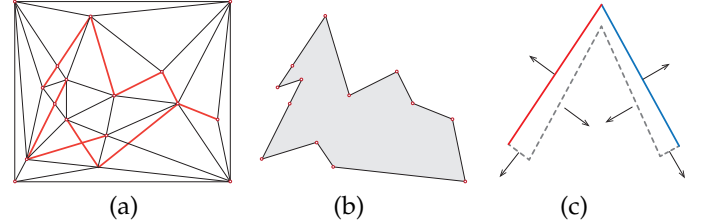

Fig. 4: *If offset is not applied to the open surface for the example in Fig. 2, we end up with $M_{\text{partition}}$ in (a) and $M_{\text{interface}}$ in (b), where a large piece of unnecessary volume is introduced, impairing visual appearance. Instead, we offset vertices of the open surface to form a closed thin shell (c), with offset faces denoted as dashed lines.*

3.4 Space Partition

After offsetting open faces, M_{offset} may still contain gaps between patches that are not identified as open surfaces. Following the previous work [9], we adopt the global step by partitioning the ambient space and solving the graph cut problem to find the interface mesh that closes all gaps.

We initialize the partitioned mesh via a Delaunay tetrahedrization of the vertex set $\mathcal{V}_{\text{offset}}$. However, such tetrahedrization cannot ensure all the input faces are included in $\mathcal{F}_{\text{partition}}$. Therefore, we iteratively split the initial partition mesh using two sets of splitting faces in the same way as constructing the BSP tree. The first set is all the faces in $\mathcal{F}_{\text{offset}}$. Including all of $\mathcal{F}_{\text{offset}}$ ensures geometric fidelity, but this is not enough to preserve user-defined surface attributes such as UV coordinates and material IDs. This is because certain edges are shared by two co-planar faces, which are recognized as a single large face by the BSP data structure. The default BSP construction algorithm will erroneously remove such edges from the data structure. Unlike [9], We additionally use an arbitrary face passing through the edge to split the partition mesh. By that, if the two neighboring faces have discontinuous surface attributions, we require their shared edge to be included in the BSP data structure.

To make the iterative partition process unconditionally robust, we use exact arithmetic during splitting [9] via LPI (Line-Plane Intersection), and TPI (Three-Planes Intersection) [42], [43] for fast exact constructions. This step yields the partitioned mesh $M_{\text{partition}}$.
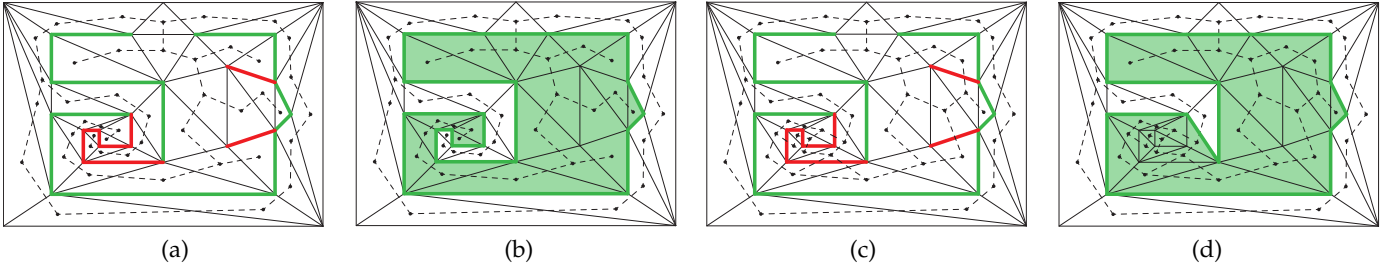
Fig. 5: *Example of graph cut: Black dots represent cell nodes, and dashed lines represent the edges between cells. Given the input mesh (a), green and red segments represent visible and invisible faces, respectively, and there is no edge over these visible and invisible faces. After solving the graph cut, the extracted mesh is in green (b). If we keep the edges across invisible faces (c), the graph cut can return an erroneously extracted mesh that closes the entire mesh (d).*

In order to preserve the surface attributions, e.g., UV coordinates, material IDs, etc., we need to maintain a mapping $\mathcal{M} : \mathcal{F}_{\text{partition}} \rightarrow \mathcal{F}_{\text{input}}$. Since we use exact arithmetic, $\mathcal{M}(f^i_{\text{partition}})$ can be determined by checking whether the barycenter of $f^i_{\text{partition}}$ lies exactly on some f^j_{input}, following the strategy used in VolumeMesher [9]. Due to the choice of our splitting surfaces, $\mathcal{M}$ is well-defined, i.e., each $f^i_{\text{partition}}$ is either contained in some f^j_{input} or does not belong to any face of $\mathcal{F}_{\text{partition}}$ (in which case we let $\mathcal{M}(f^i_{\text{partition}}) \triangleq \varnothing$).

3.5 Interface Mesh Extraction

To perform the global graph cut, we first refine the face orientation (Section 3.5.1) based on $M_{\text{partition}}$. Next, we utilize the visibility measure to classify each face in $\mathcal{F}_{\text{partition}}$ (Section 3.5.2). These measures will be used to formulate the objective function in the graph cut (Section 3.5.3) to determine the interior/exterior cells, whose interface surface will be the watertight mesh $M_{\text{interface}}$.

3.5.1 Face Reorientation

We use a similar procedure as Section 3.2 to reorient $\mathcal{F}_{\text{partition}}$. Specifically, we use a flood fill strategy to group faces in $\mathcal{F}_{\text{partition}}$ into patches, such that no patch contains non-manifold edges, meaning each edge always has lower than two adjacent faces from the same patch (illustrated in Fig. 6). Two patches can be merged if they are co-planar, have consistent orientation, and have no non-manifold edge after merging. Then, we reorient each patch based on orientation measure $\Phi_{\text{orientation}}(\mathcal{P}^j_{\text{partition}})$ as defined in Section 3.2.
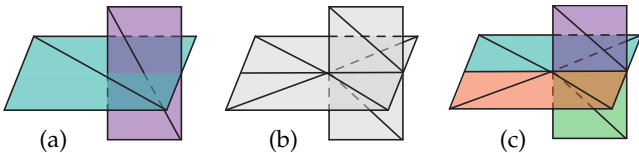

Fig. 6: *Given an input mesh (a) with two intersected patches, our extracted $M_{partition}$ (b) contains a non-manifold edge shared by the intersected patches. Starting from isolated faces, we group patches with non-manifold edges labeled by different colors in (c).*

3.5.2 Face Classification

We use the procedure in Section 3.1 to compute the visibility measure for each face in $\mathcal{F}_{\text{partition}}$. As a result, $\mathcal{F}_{\text{partition}}$ can

be classified into three groups: visible faces, invisible faces, and extra faces:

$$\mathcal{F}^{\text{visible}}_{\text{partition}} \triangleq \{f \in \mathcal{F}_{\text{parition}} \text{ is visible} \wedge \mathcal{M}(f) \neq \varnothing\}$$
$$\mathcal{F}^{\text{invisible}}_{\text{partition}} \triangleq \{f \in \mathcal{F}_{\text{parition}} \text{ is invisible} \wedge \mathcal{M}(f) \neq \varnothing\}$$
$$\mathcal{F}^{\text{extra}}_{\text{partition}} \triangleq \mathcal{F}_{\text{partition}} - \mathcal{F}^{\text{visible}}_{\text{partition}} - \mathcal{F}^{\text{invisible}}_{\text{partition}}.$$

To form the watertight interface surface, our goal is to use as many visible faces and as few extra faces as possible.

3.5.3 Cell Classification

We treat each cell in the BSP tree as a node in graph G that can be labeled as either interior or exterior. Each facet of a cell corresponds to an edge in G, but no edge is created for mappable faces ($\mathcal{M}(f^i_{\text{partition}}) \neq \emptyset$), no matter whether the face is visible or invisible as shown in Fig. 5.

The faces bordering the interior and exterior cells are guaranteed to form a watertight mesh. We solve for a set of cell labels l^i to produce the interface mesh that maximizes the use of visible faces while minimizing extra faces, which can be formulated as the following minimal cut problem:

$$E(\mathcal{L}) = \sum_{l^i} D(l^i) + \sum_{e^{ij}} S(l^i, l^j), \tag{6}$$

where $l^i \in \{I, E\}$ indicates the ith cell c^i to be either Interior or Exterior. In Eq. 6, the first data cost is formulated as:

$$D(l^i) = \begin{cases} \sum_{f \subset c^i \wedge f \in \mathcal{F}^{\text{visible}}_{\text{partition}} \wedge f \text{ has inward normal}} A(f) & \text{if } l^i = I \\ \sum_{f \subset c^i \wedge f \in \mathcal{F}^{\text{visible}}_{\text{partition}} \wedge f \text{ has outward normal}} A(f) & \text{if } l^i = E \end{cases}. \tag{7}$$

In other words, $D(l^i)$ penalizes incorrectly oriented faces. If a cell c^i is chosen to be interior, then its visible faces should have normals facing outward. Similarly, an exterior cell c^i should have visible faces facing inward. Intentionally, we exclude invisible faces' orientation from $D(l^i)$ to prevent potential bias introduced by those invisible faces within the model.

The second edge cost simply penalizes the use of any extra faces, defined as:

$$S(l^i, l^j) = \begin{cases} A(f), & \text{if } f \subset c^i \cap c^j \wedge f \in \mathcal{F}^{\text{extra}}_{\text{partition}} \wedge l^i \neq l^j \\ 0, & \text{otherwise} \end{cases}. \tag{8}$$

Notice that we remove dual edges across invisible faces so that the partition result could follow the input geometry in case of inadequate measuring samples.

It is well-known that, as long as the regular condition [44] $S(I, I) + S(E, E) \leq S(I, E) + S(E, I)$ holds, the problem of binary graph cut has a polynomial complexity

algorithm. It is trivial to see the regular condition holds in our case as $S(I, I) = S(E, E) = 0$ and $S(I, E) = S(E, I) \geq 0$. The output of this step is denoted as $M_{\text{interface}}$.

3.6 Constrained Simplification

Although our global step guarantees a watertight output, it could also incur many redundant, small facets. We could remove them using conventional mesh simplifiers, e.g., QEM-based mesh reduction [45]. However, these methods would lead to inverted faces or incur expensive computation to check for inverted faces at each simplification step. Instead, we introduce a constrained mesh simplification to reduce $M_{\text{interface}}$. We first detect the geometric and UV patch boundaries (Section 3.6.1) and then re-triangulate each patch (Section 3.6.2) to significantly reduce the face number while complying with detected boundaries.

3.6.1 Boundary Detection

We use a flood fill strategy to group adjacent co-planar faces with consistent orientation from $\mathcal{F}_{\text{interface}}$ into manifold patches $\{\mathcal{P}^j_{\text{interface}}\}$. There are two types of boundaries we want to preserve during the simplification. The first type is geometric boundary edges $\mathcal{E}_{\text{geometric}}$ that are shared by at least two patches. Preserving geometric boundaries edges can ensure that the simplified mesh is identical to $\mathcal{M}_{\text{interface}}$ geometrically.

Additionally, we preserve UV boundary edges $\mathcal{E}_{\text{UV}}$, whose UV coordinates lie on the border of the UV map. $\mathcal{E}_{\text{UV}}$ are represented as a tuple of two vertices from $\mathcal{V}_{\text{interface}}$. During partition, these UV boundaries could be split into segments, and invisible ones are discarded to make $\mathcal{M}_{\text{interface}}$ manifold and watertight, as shown in Fig. 7. We use the intersection between the origin UV and geometric boundaries to find these segments, which uses the rational number to ensure accuracy.
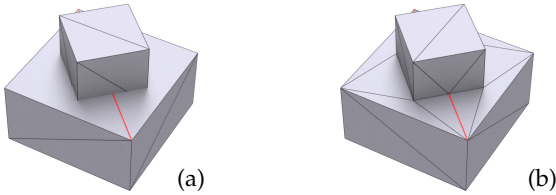

Fig. 7: *Given two intersecting boxes as the input mesh (a), there is a UV boundary edge in red. Our method still preserves that edge in the extracted mesh $M_{\text{interface}}$ (b). As the interior part is removed to from the watertight manifold output mesh, that UV boundary edge is split into segments and some segments are discarded.*

3.6.2 Constrained Triangulation

Our constrained triangulation complies with the geometry and texture boundaries found in the previous step. We first use edge-collapse to remove any vertex, whose degree is two and adjacent to co-linear edges in $\mathcal{E}_{\text{geometric}} \cup \mathcal{E}_{\text{UV}}$. Then, we add a sanity check to prevent any self-intersection before performing the collapse operation. In particular, we check if the resulting faces intersect with all other faces within the extended bounding box of P_i with extended length l_{extended} before each ear-cut operation. Then, we use the constrained ear-cut triangulation [46] that obeys geometry and texture

boundaries for each patch $\mathcal{P}^j_{\text{interface}}$. After triangulation, we get the simplified mesh $M_{\text{simplified}}$ with much fewer faces and vertices. It should be noted that both edge collapse and ear-cut triangulation techniques introduce no new vertices, operating solely within the interior of edges or faces. Consequently, these processes do not introduce any self-intersection.

3.7 Topological Correction

It is worth noting that the extracted mesh from Section 3.5.2 is a watertight combinatorial 3-manifold with boundary [9], meaning that it may contain non-manifold edges and vertices. In this case, we split these non-manifold edges and vertices to recover manifoldness. As our edge collapse and triangulation always produce edges with an even number of adjacent triangles, our method is guaranteed to output watertight and manifold meshes [47], [20]. A proof is provided in the supplemental document. The output mesh is denoted as $M_{\text{simplified}}$.

3.8 Recovering Surface Attributes

There are three types of faces in $\mathcal{F}_{\text{simplified}}$: inherited faces from F_{output}, offset faces due to Section 3.3, and extra faces defined in Section 3.5. We recover inherited faces' attributes from M_{input} using barycentric interpolation. The offset faces' attributes are copied from their original faces. In our experiment, most faces can be traced back to their original faces. However, for those extra faces created for closing holes and gaps, we do not have prior knowledge. To assign surface attributes, we perform a flood fill and iteratively set the attributes of extra faces by averaging from their one-ring neighboring vertices.

4 RESULTS

We implement our framework in C++ with CGAL, libigl, and Eigen. We use Optix [48] to compute visual metrics via ray tracing on GPU and a fast approximate energy minimization solver [49] to solve the graph cut. We did all experiments on a computer with an AMD Ryzen Threadripper 3970X 32-core Processor at 3.69 GHz and 256 GB RAM. We use $N_{\text{total}} = 2e7$, which is sufficiently large for oversampling all the models in our input dataset, and $N_{\text{min}} = 5$ for good coverage of mesh surface, and sample $N_d = 5$ directions on the unit hemisphere for both sides of f^i_{input}. Each ray has a maximum bounce number $N_{\text{b}} = 10$. We use openness threshold $\epsilon_{\text{openness}} = 0.5$ to achieve a balance between preserving thin shells and filling holes. The offset distance $d_{\text{offset}} = D/20000$ is used to control the thickness of the thin shells, and D is the diagonal length of the model's bounding box. For the ear-cut triangulation used in constrained simplification, an extension distance is set as $l_{\text{extended}} = D/1000$. We also provide a summary of notations in Table 2.

We compare our method with several state-of-the-art mesh repair methods, including PolyMender (PM) [12], TetWild (TW) [10], fTetWild (fTW) [15], VisualRepair (VR) [11], VolumeMesher (VM) [9], combining of VisualRepair and VolumeMesher (VR+VM), combining of Takayama
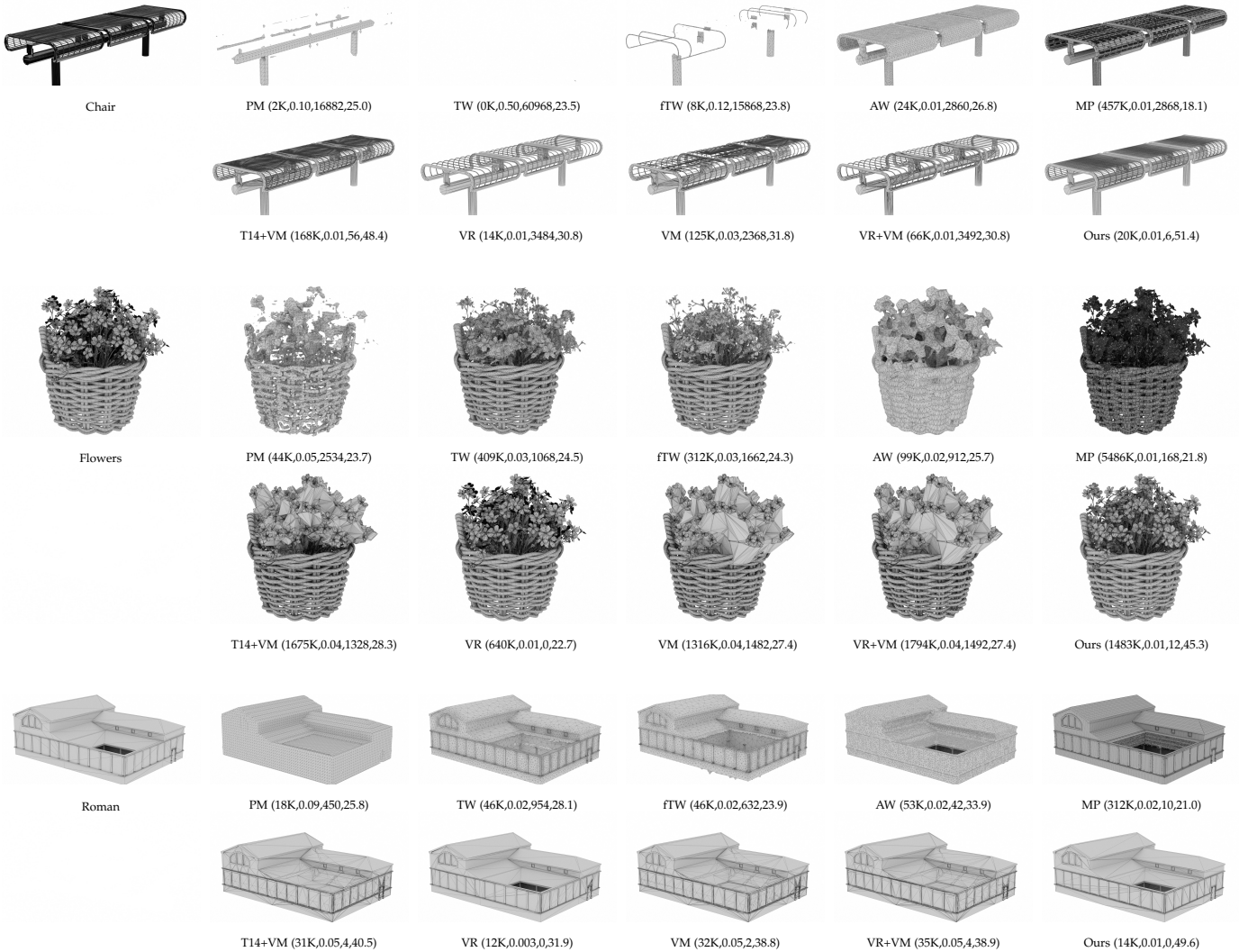
Fig. 8: *Results: We choose "Chair", "Flowers", and "Roman" to demonstrate the issues of existing methods and our superiority over them.* $(\bullet, \bullet, \bullet, \bullet)$ *indicates face number, Hausdorff distance, LFD, and PSNR.*

TABLE 2: *Summary of notations*

Notations	Descriptions
$M_\bullet = <\mathcal{V}_\bullet, \mathcal{F}_\bullet>$	Mesh $\bullet$ with vertices $\mathcal{V}_\bullet$ and face $\mathcal{F}_\bullet$
$v_\bullet^i \in \mathcal{V}_\bullet$	Mesh vertex
$f_\bullet^i \in \mathcal{F}_\bullet$	Mesh face
$\mathcal{P}_\bullet^j = \{f_\bullet^{ji}\}$	Patch of faces
$\mathcal{E}_\bullet^j = \{(v_\bullet^{j0}, v_\bullet^{j1})\}$	Edge set
$A(\cdot)$	Face area
$Adj(\cdot)$	Adjacent faces
$|\cdot|$	Cardinality of a discrete set
$\Phi_{\text{visibility}}(f_\bullet^i)$	Visibility score of face
$\Phi_{\text{orientation}}(f_\bullet^i)$	Orientation score of face
$\Phi_{\text{openness}}(f_\bullet^i)$	Openness score of face

et al. [16] and VolumeMesher (T14+VM), and AlphaWrapping (AW) [13] (we use hyper-parameters $\alpha = D/100$, $\delta = D/3000$ for AW as suggested in their work) on randomly chosen 1000 models from ShapeNet [2] and 400 models from Thingi10K [17]. All results can be found in https://github.com/VisualGuidedMeshRepair/dataset. We evaluate the result quality with three qualitative metrics, Hausdorff distance (HD), light-field distance (LFD), and

peak signal-to-noise ratio (PSNR). Notably, we render the input mesh with double face rendering as the reference and render others' and our results with the back face colored in black. The resulting statistics are collected in Table 3, and we pick four examples ("Chair" and "Table" are from ShapeNet) to demonstrate the issues of the existing methods in Fig. 8 and Fig. 9. Note that for the "Chair" model, TW's result loses a large portion of the shape due to the mis-oriented input faces.

Watertightness, Manifoldness, and Number of Faces. Unlike VR [11], our proposed method guarantees a watertight and manifold output, which only subtracts the input mesh without filling gaps and holes. Other approaches, such as TW and VM, only ensure a combinatorial 3-manifold with boundary output that may contain non-manifold edges. Moreover, VM's output is unnecessarily complex due to BSP partitioning, resulting in tripling the number of faces in M_{input}. In contrast, our constrained simplification step significantly reduces the face count to approximately the same level as M_{input}.

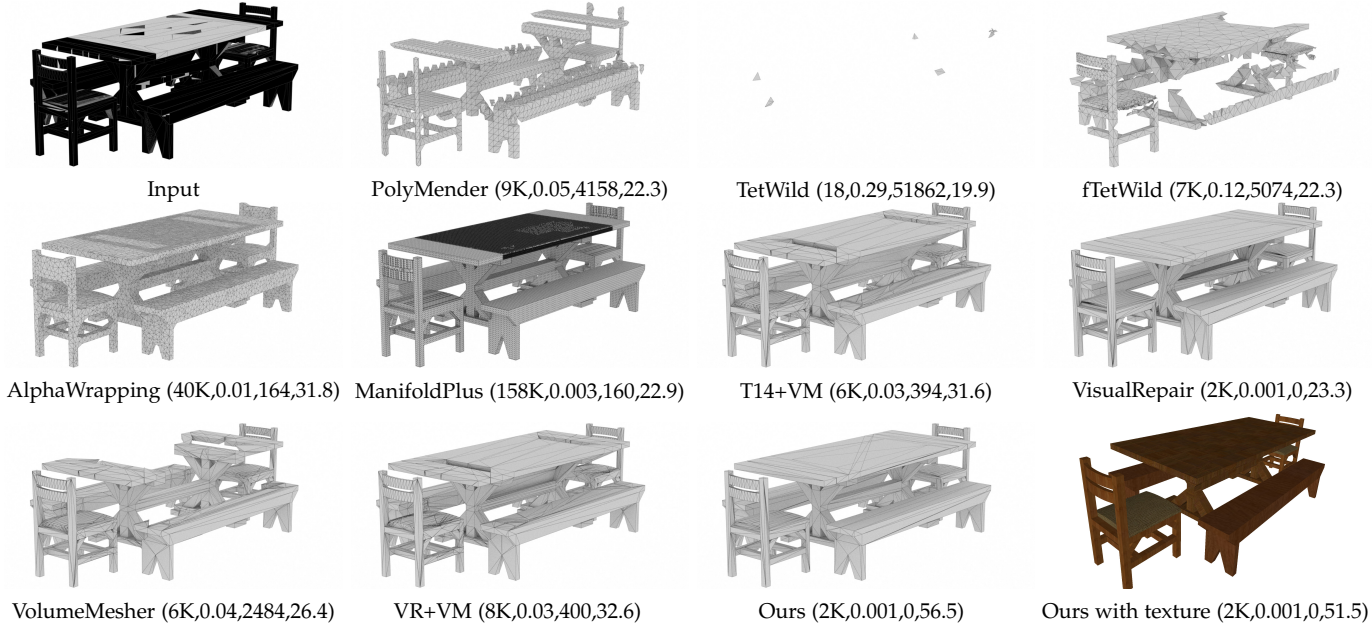**HD, LFD, and PSNR.** Due to misorientation of the input

Fig. 9: *Table:* We choose "Table" to demonstrate the issues of existing methods and our superiority over them. (●, ●, ●, ●) indicates face number, Hausdorff distance, LFD, and PSNR.

TABLE 3: *Statistics of 1000 models from ShapeNet and 400 models of Thingi10K, including the percentage of results that are finished within 1 hour, watertight and manifold, respectively, and average face number, HD, LFD, PSNR, peak memory, and time usage. Smaller values are more desired for all listed numerical metrics except PSNR. * indicates PNSR of our result with UVs, while all other methods do not have UVs. Note that Models in Thingi10K do not have UVs.*

	Finishes (< 1h)	Watertight	Manifold	Face #	HD	LFD	PSNR	Mem.	Time (s)
ShapeNet									
Input	–	82%	2%	32K	–	–	–	–	–
PM	100%	100%	100%	14K	0.06	$2.2e3$	29.0	7 MB	0.4
TW	99.5%	100%	23%	4K	0.18	$3.5e4$	22.2	943 MB	259.4
fTW	99.7%	100%	100%	10K	0.18	$1.3e4$	26.3	714 MB	114.8
AW	100%	100%	100%	34K	0.03	$2.3e2$	35.3	96 MB	6.0
VR	97.9%	18%	100%	22K	0.02	$1.0e1$	48.1	111 MB	176.7
VM	100%	100%	16%	95K	0.05	$1.4e3$	32.8	313 MB	3.6
VR+VM	97.9%	100%	37%	180K	0.03	$2.0e2$	49.4	381 MB	186.8
MP	100%	100%	100%	325K	0.02	$8.0e1$	36.9	217 MB	3.8
T14+VM	100%	100%	31%	117K	0.03	$2.4e2$	46.6	299 MB	11.8
Ours	100%	100%	100%	29K	0.02	$1.0e0$	58.9 (58.8*)	722 MB	16.9
Thingi10K									
Input	–	5%	97%	40K	–	–	–	–	–
PM	100%	100%	100%	17K	0.02	$9.5e2$	31.7	8 MB	0.9
TW	100%	100%	87%	13K	0.006	$3.3e2$	47.9	213 MB	77.7
fTW	100%	100%	100%	10K	0.006	$3.3e2$	46.3	143 MB	24.0
AW	100%	100%	100%	27K	0.014	$1.1e2$	37.2	88 MB	4.4
VR	90%	66%	92%	34K	0.0001	$5.0e\text{-}2$	67.9	149 MB	406.2
VM	100%	100%	81%	128K	0.006	$1.9e2$	64.1	258 MB	4.4
VR+VM	90%	100%	84%	119K	0.003	$7.7e1$	65.2	259 MB	411.4
MP	100%	100%	100%	416K	0.006	$3.8e1$	50.8	274 MB	7.8
T14+VM	98%	100%	75%	110K	0.004	$9.0e0$	64.1	237 MB	24.9
Ours	100%	100%	100%	36K	0.001	$5.5e0$	66.7	755 MB	26.7

from ShapeNet, methods that rely on input orientations to determine the interior and exterior, such as PM, TW, fTW, and VM, cannot produce a result even close to the input. Additionally, "Flowers" in Fig. 8 demonstrate that VM might close the surface with lots of unnecessary volumes for the open faces. VM also fails to close the court due to the open surface under the eave in "Roman", and VM always solves for the smallest surface area, which might close the concave structure. On the other hand, AlphaWrapping, which does not require consistent orientation, can suffer from the blurring of input geometric details and sharp

features if the surface is offset by a large distance. PM fails to preserve sharp features due to limited octree resolution (see "Chair"). For ShapeNet, although VR scores well in HD and LFD, its optimization cannot ensure correct patch orientation across the entire mesh, resulting in a low PSNR score, for which one reason is that visual measures in PM are based on rasterization, which may not capture small or occluded faces accurately, such as the frames on the chair (Fig. 8) and engine on the plane (Fig. 1).

For the models in Thingi10K, which does not have complex interior structure, only VR can get better

HD/LFD/PSNR than ours; there are 40 out of 400 models that VR cannot produce the results within 1 hour due to the high input face number. For a more fair comparison, we manually tune the parameters of fTW, PM, and AW to match the same face number with ours for the "Flower" example. Unfortunately, fTW is out of 64G memory. More importantly, fTW cannot handle open faces or complex nested structures as shown in Fig. 10, as it uses the winding number to determine the interior/exterior..
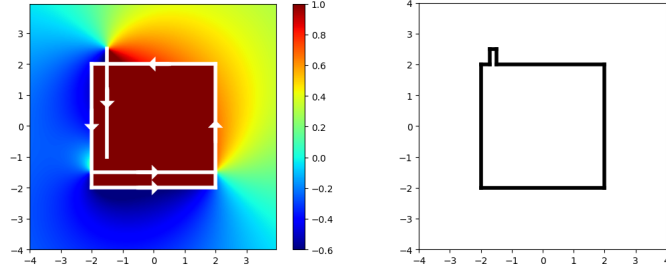

Fig. 10: *Example of winding number: left, the winding number (visualized by color) of a nested input disrupted by inner structures; right, our target output.*

Given the same face number, HD/LFD/PSNR of PM (0.025/1492/20.1) and AW (0.01/200/25.8) are still worse than ours (0.01/12/45.3). We further tune AW and manage to produce a similar HD and LFD as ours. However, even with 10x more faces, AW (0.01/56/30.4) is still unable to achieve a similar quality. In summary, our method outperforms all listed state-of-the-art techniques. In addition, unlike other methods, our method can close the hole and propagate UV for the newly added faces from neighboring faces, as shown in Fig. 11.
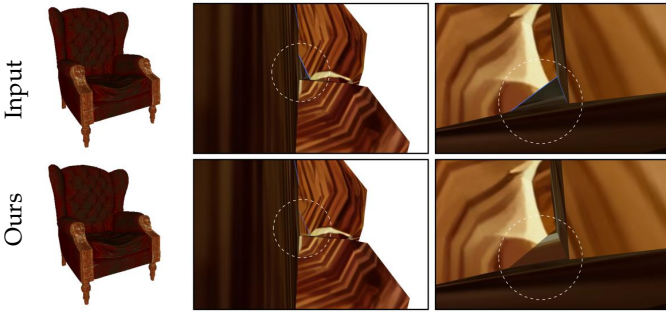

Fig. 11: *Example of UV recovering: Top, Input textured model with holes; bottom, Our repaired model, with holes filled and UV recovered.*

Memory and Time Usage. Due to offset faces and extra cuts for boundary edges, ours need more memory and computational time than VM. Fortunately, our visual evaluation step is based on ray tracing where multi-bounce offers a much more efficient space exploration than VR [11]. We also plot the time breakdown of our pipeline in Fig. 12, where our ray-tracing step only takes 7% of the computation time. Boundary detection occupies almost one-third of the computation time due to using rational numbers. Our processing time does not solely depend on mesh size. The complexity of the input, the number of intersection faces, and the size of the output all have impacts.
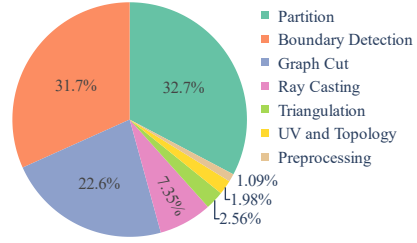

Fig. 12: *Time breakdown*

Compared to VR+VM and T14+VM. As previously stated, VM is a volumetric approach that can generate a mesh without holes, yet it relies on correct input orientation. In contrast, VR and T14 can adjust orientation but are unable to fill gaps. Therefore, we use VR and T14 to orientate the faces and use VM to mend the topology, denoted as VR+VM and T14+VM. As anticipated, these approaches perform better in PSNR than any other existing technique that guarantees watertightness. Nevertheless, the resulting mesh contains numerous faces due to the additional face division caused by VR. Furthermore, our method achieves around 10 PSNR advantages over the VR+VM and T14+VM approaches. One reason is that our method incorporates visual guidance throughout the entire repair process and optimization procedure. On the other hand, VR+VM and T14+VM only apply the visual metric to the surface optimization step, i.e., the graph cut stage; thus, it has not fully exploited the critical visual cues. For instance, in "Chair" example, since the rasterization cannot capture tiny faces, VR and T14 cannot correct all face orientations, which leads to incorrect output from VM. The same issue can be observed in Fig. 1 as well. For "Table" in Fig. 8, VM, VR+VM, and T14+VM all discard part of the table because the graph cut in VM solves for the minimal surface area, which is not always the case.

Maximum Bounce Number N_b. The maximum bounce number N_b governs the ray-tracing process's ability to navigate complex structures. When $N_b = 0$, no bounces are allowed, potentially misguiding the representation of hard-to-see structures. Conversely, when $N_b = 10$, the exploration is thorough, facilitated by multiple reflections. We conducted tests with varying max bounce numbers using the 500 models from ShapeNet [2]. For $N_b = 0, 1, 2, 3, 5,$ and 10, we obtained HD/LFD/PSNR values of 0.03/3.3/57.1, 0.02/2.0/58.3, 0.02/0.9/59.1, 0.02/0.7/59.5, and 0.02/0.7/60.2, respectively. Subsequently, we employed $N_b = 10$ for all subsequent experiments in this paper.

Hard-to-see Structure. The combination of ray casting and graph cut makes our method robustly handle complex structures. It is worth noting that, for the case that is hard to see, we design our graph cut to respect the input surface geometry. As shown in Fig. 13, we tested our method on a Hilbert Cube from Thingi10K using default parameters $N_b = 10$ and $N_{\text{total}} = 2 \times 10^7$ to obtain the complete visual guidance and $N_b = 0$ and $N_{\text{total}} = 2 \times 10^3$ to obtain only partial visual guidance. Under default parameters, multiple reflections enable a comprehensive exploration of hard-to-see structures. On the other hand, no bounce can only provide partial visual guidance, while breaking edges for
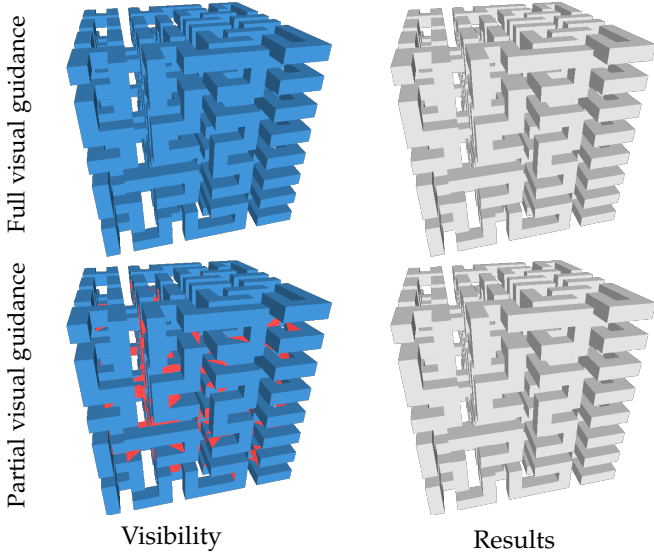
Fig. 13: *Example of dealing with hard-to-see structures:* The Hilbert Cube model from Thingi10K exhibits intricate structures. The visibility (left) and the outcomes (right) of our method under complete visual guidance (top) and partial visual guidance (bottom) are shown. **Blue** indicates visible faces, while **Red** represents faces not visible by the ray with zero bounces. Remarkably, in both scenarios, our approach achieves zero HD.

invisible faces in the graph cut process allows our method to respect input geometry. In both cases, the output meshes have zero HD, meaning our results are geometrically identical to the input mesh.

Orientation and Offset. Our method utilizes an orientation measure to guide the orientation and offset direction. In the case of non-orientable surfaces, such as the Möbius strip (Fig. 14), our method offsets the surface in two directions for non-orientable edges, creating a valid solid shell. We used $d_{\text{offset}} = D/200$ to make the results more pronounced.
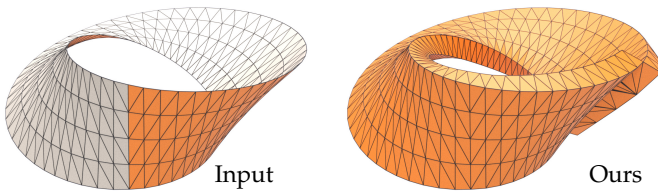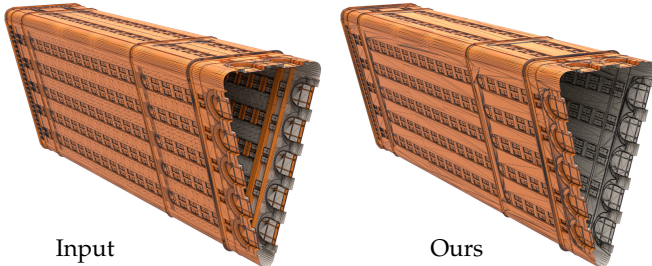

Fig. 14: *Example of non-orientable surface*

Fig. 15: *Example of preserving hole*

Preserving Holes. Although our method closes the hole and converts the large open surface into a thin shell by default, our system allows users to specify the open boundary
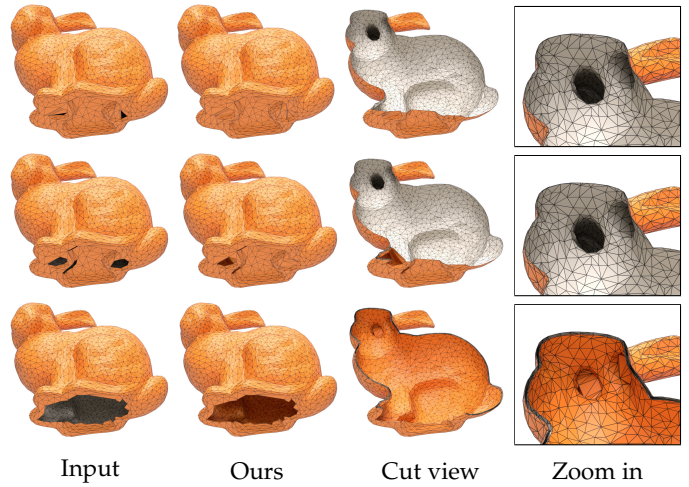

Fig. 16: *Study on hole size:* With $\epsilon_{openness} = 0.5$, from left to right, as the hole size in the "Bunny" model increases from small to large, more faces are progressively identified as open, leading to their filling by our method, ultimately forming a thin shell. The front side is orange, while the back side is white.

to be preserved during the repair (Fig. 15). Particularly, our system first closes the hole and marks all corresponding faces. After repair, all marked faces are removed to recover the boundary.

5 CONCLUSION

In this work, we presented three crucial visual measures to assess visibility, orientation, and openness, and we proposed a novel framework for mesh repair that incorporates these measures into critical steps such as local open surface closing, face reorientation, and global graph cut using a visual-guided objective function. Our method was evaluated on a set of 500 models randomly selected from ShapeNet, showcasing its effectiveness and robustness compared to existing techniques.

Limitations. Nevertheless, it is important to acknowledge that our method does possess certain limitations.

Firstly, since our output targets visual-driven applications, such as rendering-related mesh processing tasks, our method would entirely discard invisible inner structures, which may be an issue when preserving these structures is vital, such as internal seats and engines in cars. Additionally, assessing the openness of structures exhibiting channel-like characteristics presents sample efficiency challenges for raycasting. For instance, even with a substantial opening at the base, the ears of the "Bunny" model do not manifest a shell-like structure Fig. 16. In the future, we would like to utilize differentiable rendering techniques to fill textures for newly added faces. It would also be interesting to remesh our output to improve the mesh quality, e.g., mesh aspect ratio.

Besides, although our method can robustly convert any input mesh into a watertight manifold mesh, using default parameters tends to preserve the input topological features rather than closing holes and gaps to produce a single closed surface, as done in PolyMender [12]. , extreme parameters can be used to ensure hole closure as illustrated in Fig. 17.
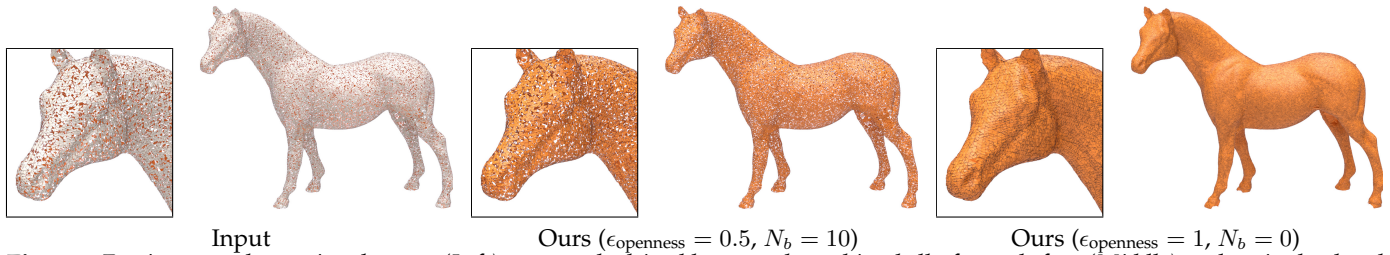
| Input | Ours ($\epsilon_{openness} = 0.5$, $N_b = 10$) | Ours ($\epsilon_{openness} = 1$, $N_b = 0$) |

Fig. 17: *For input such as triangle soup (Left), our method is able to produce thin shells for each face (Middle) and a single closed manifold (right) by tuning $\epsilon_{openness}$ and N_b.*

Although the ray casting is not the bottleneck of our current pipeline, utilizing advanced rendering techniques, such as an adaptive adjustment for the maximum bounce number, could improve the sample efficiency. Also, we'd like to adjust the distance for offsetting open surfaces adaptively to avoid potential issues when the open is small and the model is large.

Finally, our method guarantees topological manifoldness and watertightness, and ensures geometrically self-intersection-free and degenerate-face-free under exact arithmetic, but geometric guarantees are lost when exporting our output to standard file format for downstream applications under finite floating point precision. This problem can be potentially solved by using TetWild to process our output mesh or saving and transferring our output using exact arithmetic data format.

ACKNOWLEDGEMENT

REFERENCES

[1] M. Attene, M. Campen, and L. Kobbelt, "Polygon mesh repairing: An application perspective," *ACM Comput. Surv.*, vol. 45, no. 2, mar 2013.

[2] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "ShapeNet: An Information-Rich 3D Model Repository," Stanford University — Princeton University — Toyota Technological Institute at Chicago, Tech. Rep. arXiv:1512.03012 [cs.GR], 2015.

[3] R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, and D. Cohen-Or, "Meshcnn: a network with an edge," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, pp. 1–12, 2019.

[4] S.-M. Hu, Z.-N. Liu, M.-H. Guo, J.-X. Cai, J. Huang, T.-J. Mu, and R. R. Martin, "Subdivision-based mesh convolution networks," *ACM Transactions on Graphics (TOG)*, vol. 41, no. 3, pp. 1–16, 2022.

[5] A. Hornung and L. Kobbelt, "Robust reconstruction of watertight 3d models from non-uniformly sampled point clouds without normal information," in *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, ser. SGP '06. Goslar, DEU: Eurographics Association, 2006, p. 41–50.

[6] W. Zhao, S. Gao, and H. Lin, "A robust hole-filling algorithm for triangular mesh," in *2007 10th IEEE International Conference on Computer-Aided Design and Computer Graphics*. New York, NY, USA: IEEE, 2007, pp. 22–22.

[7] M. Attene, "A lightweight approach to repairing digitized polygon meshes," *The Visual Computer*, vol. 26, no. 11, pp. 1393–1406, 2010.

[8] ——, "Direct repair of self-intersecting meshes," *Graph. Models*, vol. 76, no. 6, p. 658–668, nov 2014.

[9] L. Diazzi and M. Attene, "Convex polyhedral meshing for robust solid modeling," *ACM Trans. Graph.*, vol. 40, no. 6, dec 2021.

[10] Y. Hu, Q. Zhou, X. Gao, A. Jacobson, D. Zorin, and D. Panozzo, "Tetrahedral meshing in the wild," *ACM Trans. Graph.*, vol. 37, no. 4, jul 2018.

[11] L. Chu, H. Pan, Y. Liu, and W. Wang, "Repairing man-made meshes via visual driven global optimization with minimum intrusion," *ACM Trans. Graph.*, vol. 38, no. 6, nov 2019.

[12] T. Ju, "Robust repair of polygonal models," *ACM Trans. Graph.*, vol. 23, no. 3, p. 888–895, aug 2004.

[13] C. Portaneri, M. Rouxel-Labbé, M. Hemmer, D. Cohen-Steiner, and P. Alliez, "Alpha wrapping with an offset," *ACM Trans. Graph.*, vol. 41, no. 4, jul 2022.

[14] J. Huang, Y. Zhou, and L. Guibas, "Manifoldplus: A robust and scalable watertight manifold surface generation method for triangle soups," *arXiv preprint arXiv:2005.11621*, 2020.

[15] Y. Hu, T. Schneider, B. Wang, D. Zorin, and D. Panozzo, "Fast tetrahedral meshing in the wild," *ACM Trans. Graph.*, vol. 39, no. 4, Jul. 2020.

[16] K. Takayama, A. Jacobson, L. Kavan, and O. Sorkine-Hornung, "A simple method for correcting facet orientations in polygon meshes based on ray casting," *Journal of Computer Graphics Techniques*, vol. 3, no. 4, p. 53, 2014.

[17] Q. Zhou and A. Jacobson, "Thingi10k: A dataset of 10,000 3d-printing models," *arXiv preprint arXiv:1605.04797*, vol. 0, no. 0, 2016.

[18] T. Ju, "Fixing geometric errors on polygonal models: A survey," *Journal of Computer Science and Technology*, vol. 24, pp. 19–29, 2009.

[19] A. Guéziec, G. Taubin, F. Lazarus, and B. Horn, "Cutting and stitching: Converting sets of polygons to manifold surfaces," *IEEE Transactions on Visualization and Computer Graphics*, vol. 7, no. 2, pp. 136–151, 2001.

[20] M. Attene, D. Giorgi, M. Ferri, and B. Falcidieno, "On converting sets of tetrahedra to combinatorial and pl manifolds," *Computer Aided Geometric Design*, vol. 26, no. 8, pp. 850–864, 2009.

[21] G. Turk and M. Levoy, "Zippered polygon meshes from range images," in *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '94. New York, NY, USA: Association for Computing Machinery, 1994, p. 311–318.

[22] J. Podolak and S. Rusinkiewicz, "Atomic volumes for mesh completion," in *Proceedings of the Third Eurographics Symposium on Geometry Processing*, ser. SGP '05. Goslar, DEU: Eurographics Association, 2005, p. 33–es.

[23] M. Campen and L. Kobbelt, "Exact and robust (self-)intersections for polygonal meshes," *Computer Graphics Forum*, vol. 29, no. 2, pp. 397–406, 2010.

[24] S. Bischoff, D. Pavic, and L. Kobbelt, "Automatic restoration of polygon models," *ACM Trans. Graph.*, vol. 24, no. 4, p. 1332–1352, oct 2005.

[25] M. Attene, "As-exact-as-possible repair of unprintable stl files," *Rapid Prototyping Journal*, vol. 24, no. 5, pp. 855–864, 2018.

[26] S. Oomes, P. Snoeren, and T. Dijkstra, "3d shape representation: Transforming polygons into voxels," in *Proceedings of the First International Conference on Scale-Space Theory in Computer Vision*, ser. SCALE-SPACE '97. Berlin, Heidelberg: Springer-Verlag, 1997, p. 349–352.

[27] C. Andújar, P. Brunet, and D. Ayala, "Topology-reducing surface simplification using a discrete solid representation," *ACM Trans. Graph.*, vol. 21, no. 2, p. 88–105, apr 2002.

[28] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '96. New York, NY, USA: Association for Computing Machinery, 1996, p. 303–312.

[29] R. Furukawa, T. Itano, A. Morisaka, and H. Kawasaki, "Improved space carving method for merging and interpolating multiple range images using information of light sources of active stereo,"

in *Computer Vision – ACCV 2007: 8th Asian Conference on Computer Vision, Tokyo, Japan, November 18-22, 2007, Proceedings, Part II.* Berlin, Heidelberg: Springer-Verlag, 2023, p. 206–216.

[30] T.-q. Guo, J.-j. Li, J.-g. Weng, and Y.-t. Zhuang, "Filling holes in complex surfaces using oriented voxel diffusion," in *2006 International Conference on Machine Learning and Cybernetics*. USA: IEEE, 2006, pp. 4370–4375.

[31] T. Masuda, "Filling the signed distance field by fitting local quadrics," in *Proceedings. 2nd International Symposium on 3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004.* USA: IEEE, 2004, pp. 1003–1010.

[32] F. S. Nooruddin and G. Turk, "Simplification and repair of polygonal models using volumetric techniques," *IEEE Transactions on Visualization and Computer Graphics*, vol. 9, no. 2, p. 191–205, apr 2003.

[33] J. Spillmann, M. Wagner, and M. Teschner, "Robust tetrahedral meshing of triangle soups," in *Proc. Vision, Modeling, Visualization (VMV)*, Citeseer. Netherlands: Ios Pr Inc, 2006, pp. 9–16.

[34] F. Hétroy, S. Rey, C. Andújar, P. Brunet, and ı. Vinacua, "Mesh repair with user-friendly topology control," *Comput. Aided Des.*, vol. 43, no. 1, p. 101–113, jan 2011.

[35] T. M. Murali and T. A. Funkhouser, "Consistent solid and boundary representations from arbitrary polygonal data," in *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, ser. I3D '97. New York, NY, USA: Association for Computing Machinery, 1997, p. 155–ff.

[36] A. Jacobson, L. Kavan, and O. Sorkine-Hornung, "Robust inside-outside segmentation using generalized winding numbers," *ACM Trans. Graph.*, vol. 32, no. 4, jul 2013.

[37] J.-P. Bauchet and F. Lafarge, "Kinetic shape reconstruction," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 5, pp. 1–14, 2020.

[38] P. Labatut, J.-P. Pons, and R. Keriven, "Robust and efficient surface reconstruction from range data," in *Computer graphics forum*, vol. 28, no. 8. Wiley Online Library, 2009, pp. 2275–2290.

[39] K. Wu, X. He, Z. Pan, and X. Gao, "Occluder Generation for Buildings in Digital Games," *Computer Graphics Forum*, vol. 41, no. 7, 2022.

[40] X. Gao, K. Wu, and Z. Pan, "Low-poly mesh generation for building models," in *ACM SIGGRAPH 2022 Conference Proceedings*, ser. SIGGRAPH '22. New York, NY, USA: Association for Computing Machinery, 2022.

[41] P. Dutré, "Global illumination compendium," *Computer Graphics, Department of Computer Science, Katholieke Universiteit Leuven*, vol. 0, no. 0, 2003.

[42] G. Cherchi, M. Livesu, R. Scateni, and M. Attene, "Fast and robust mesh arrangements using floating-point arithmetic," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 6, pp. 1–16, 2020.

[43] B. Wang, T. Schneider, Y. Hu, M. Attene, and D. Panozzo, "Exact and efficient polyhedral envelope containment check." *ACM Trans. Graph.*, vol. 39, no. 4, p. 114, 2020.

[44] V. Kolmogorov and R. Zabin, "What energy functions can be minimized via graph cuts?" *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 2, pp. 147–159, 2004.

[45] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '97. USA: ACM Press/Addison-Wesley Publishing Co., 1997, p. 209–216.

[46] M. Held, "Fist: Fast industrial-strength triangulation of polygons," *Algorithmica*, vol. 30, pp. 563–596, 2001.

[47] J. Rossignac and D. Cardoze, "Matchmaker: Manifold breps for non-manifold r-sets," in *Proceedings of the Fifth ACM Symposium on Solid Modeling and Applications*, ser. SMA '99. New York, NY, USA: Association for Computing Machinery, 1999, p. 31–41.

[48] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, and M. Stich, "Optix: A general purpose ray tracing engine," *ACM Trans. Graph.*, vol. 29, no. 4, jul 2010.

[49] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 23, no. 11, pp. 1222–1239, 2001.

Zhongtian Zheng received the B.Sci. degree from Peking University in 2021. He is currently a master student at Graphics and Interaction Lab (GIL) of Peking University. His research interests include 3D vision and geometry processing.
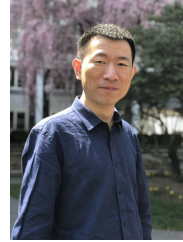
Xifeng Gao is a principal researcher at Light-Speed Studios of Tencent America. He has more than 10 years of academic research experience, with a focus on geometry computing. The target applications are computer graphics, computer-aided-design and analysis, and robotics. He has more than 40 peer-reviewed papers published in top-ranked journals and conferences. More information can be found on https://gaoxifeng.github.io/

Zherong Pan is a senior researcher at Tencent America. He was a postdoctoral researcher at the Intelligent Motion Lab of the University of Illinois Urbana-Champaign. He obtained Ph.D. degree from the University of North Carolina at Chapel Hill. His research is focused on numerical analysis, motion planning, and physics-based modeling for high-dimensional deformable objects.


Wei Li is a senior research scientist in Tencent Shanghai, LightSpeed Studios. Previously, he was a Postdoc at Inria Saclay, worked with Prof. Mathieu Desbrun. He obtained my doctorate from ShanghaiTech University in China, supervised by Prof. Xiaopei Liu, and majoring in computer graphics. Differing from traditional fluid simulation, my research is multi-disciplinary, covering many aspects in computer science and computational physics, as well as applying artificial intelligence in physical simulation.

Peng-Shuai Wang got his Ph.D. degree from the Institute for Advanced Study at Tsinghua University in 2018. He is currently a tenure-track Assistant Professor at Peking University. Before that, he was a senior researcher at Microsoft Research Asia. His research interest includes deep learning, geometry processing, and computer graphics. His paper on Octree-based Sparse Convolutional Networks is one of the Top-5 Most Cited Papers in ACM Transactions on Graphics from 2016 to 2021.

Guoping Wang received the bachelor and master degree from Dept. of Mathematics, Harbin Institute of Technology in 1987 and 1990 respectively and Ph.D degree from Institute of Mathematics, Fudan University in 1997. He was engaged in postdoctoral research in Tsinghua University from 1997 to 1999, and got full professor position in Peking University in 2002. He achieved the National Science Fund for Distinguished Young Scholars in 2009. His research interests include computer graphics, human computer interaction and virtual reality.
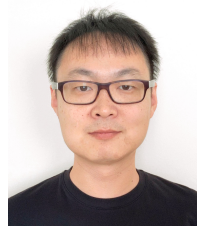
Kui Wu is a principal research scientist in Light-Speed Studios, Tencent America. Previously, he was a postdoctoral associate in the Computational Design & Fabrication Group under the guidance of Prof. Wojciech Matusik at MIT CSAIL. He received my Ph.D. degree in Computer Science from University of Utah in 2019, advised by Prof. Cem Yuksel. My research interests are computer graphics, especially on mesh processing, knitting, real-time rendering, and physical-based simulation.

Supplemental Document: Visual-Preserving Mesh Repair

Zhongtian Zheng, Xifeng Gao, Zherong Pan, Wei Li, Peng-Shuai Wang, Guoping Wang*, Kui Wu*

✦

1 ADDITIONAL RESULTS

Ablation Study. We scrutinize and analyze the efficacy of pivotal components of our method, including visual guidance, graph cut, and constrained simplification, using the "Chair" model. As shown in Table 1, without visual guidance, all faces are considered visible and are not reoriented. This leads to subpar performance in terms of HD, LFD, and PSNR due to the presence of misoriented or open faces. Without a graph cut, it entails retaining steps prior to the graph cut, as the subsequent steps rely on its outcomes. Consequently, the outcome lacks watertightness and manifold properties, resulting in the model being unrepaired. Lastly, after simplification, the face counts are reduced from 169K to 20K.

TABLE 1: *Ablation study on each component of our method*

	Watertight	Manifold	Face #	HD	LFD	PSNR
Input	Yes	No	27K	–	–	–
No Visual guidance	Yes	Yes	17K	0.12	7956	27.3
No graph cut	No	No	13K	0	0	56.7
No simplification	Yes	Yes	169K	0.01	6	51.4
Ours	Yes	Yes	20K	0.01	6	51.4

"Skyscraper" (Fig. 2) shows VM, TW, and fTW have difficulty handling the model with inner structures, which misdirect the cell classification when using graph cut or winding numbers.

Offset Distance d_{offset}. The offset distance d_{offset} controls the thickness of the thin shells created for open faces. We illustrate the impact of the offset distance d_{offset} using a flower model with different d_{offset}. Fig. 1 demonstrates our method can robustly convert the input model with hundreds of open faces to a watertight mesh, where the Hausdorff distance between output and input meshes is controlled by d_{offset}. We use $d_{\text{offset}} = D/20000$ as default. On average, the introduction of intersected faces by offset faces amounts to approximately 280 pairs across the 1400 testing models (with an average of 33,829 faces). In the majority of cases, due to the presence of intersecting open faces in the original model, our offset faces will inevitably intersect with these existing intersections.

- This work was done when Z. Zheng was an intern at LightSpeed Studios.
- Z. Zheng, P. Wang, and G. Wang are with Peking University.
- X. Gao, Z. Pan, W. Li, and K. Wu are with LightSpeed Studios.
- Corresponding Authors: Kui Wu, E-mail: kwwu@global.tencent.com, Guoping Wang, E-mail: wgp@pku.edu.cn

Openness and Holes Filling. Our method utilizes the concept of an "openness measure" to maintain thin shell structures. When the threshold is increased, or the holes are made smaller, our method prioritizes hole filling. Conversely, when the threshold is decreased, or the holes are enlarged, our method leans toward creating thin shells. Due to the stochastic nature of ray tracing, our method can be sensitive in visual measure computation. Hence, we conducted a study to investigate the impact of the opening ratio of a sphere and openness threshold by opening portions of the sphere with various $\epsilon_{\text{openness}}$ in Fig. 3, where we use $d_{\text{offset}} = D/200$ to make the volumetric shell more pronounced. With $\epsilon_{\text{openness}} = 1.0$, all faces are identified as not open no matter how many rays can be shot from faces to outside, and all small holes are closed to form a sphere with a single layer. When $\epsilon_{\text{openness}} = 0.5$, a smaller opening ratio leads to the closure of the hole, as there are insufficient rays shot from the interior to identify any face as an open surface. As the opening ratio increases, more faces are progressively
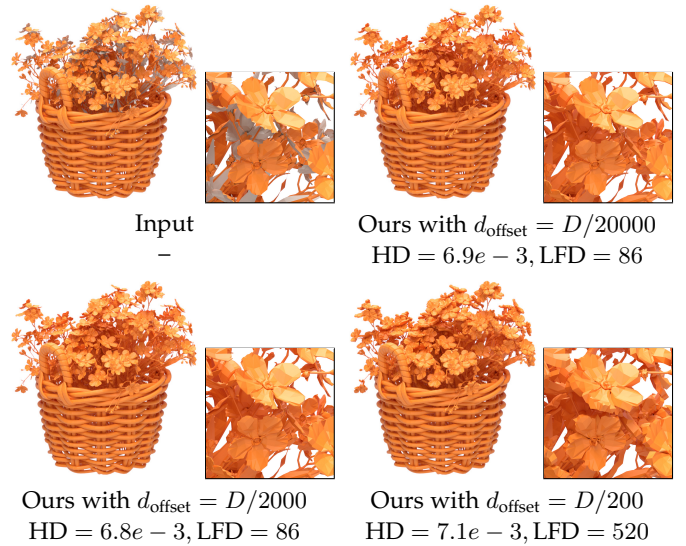

Input
–

Ours with $d_{\text{offset}} = D/20000$
HD $= 6.9e-3$, LFD $= 86$

Ours with $d_{\text{offset}} = D/2000$
HD $= 6.8e-3$, LFD $= 86$

Ours with $d_{\text{offset}} = D/200$
HD $= 7.1e-3$, LFD $= 520$

Fig. 1: *Study on d_{offset}:* *Given an input plant model containing numerous open faces, our method can reliably convert it into watertight manifold meshes, enabling control over Hausdorff distance (HD) and light field distance (LFD) by adjusting d_{offset}. D refers to the diagonal length of the model's bounding box. The front side is orange, while the back side is white.*

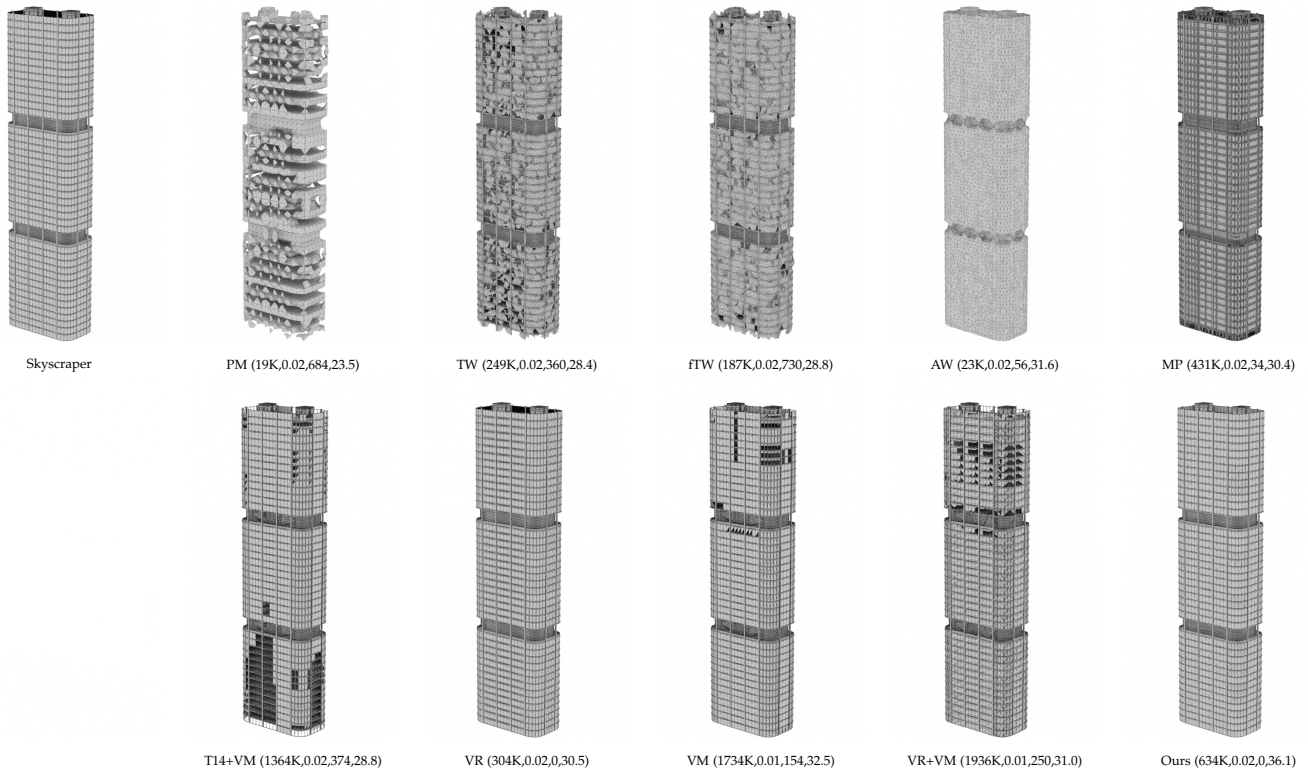**Fig. 2: *Skyscraper:*** *We choose "Skyscraper" to demonstrate the issues of existing methods and our superiority over them. (●, ●, ●, ●) indicates face number, Hausdorff distance, LFD, and PSNR.*
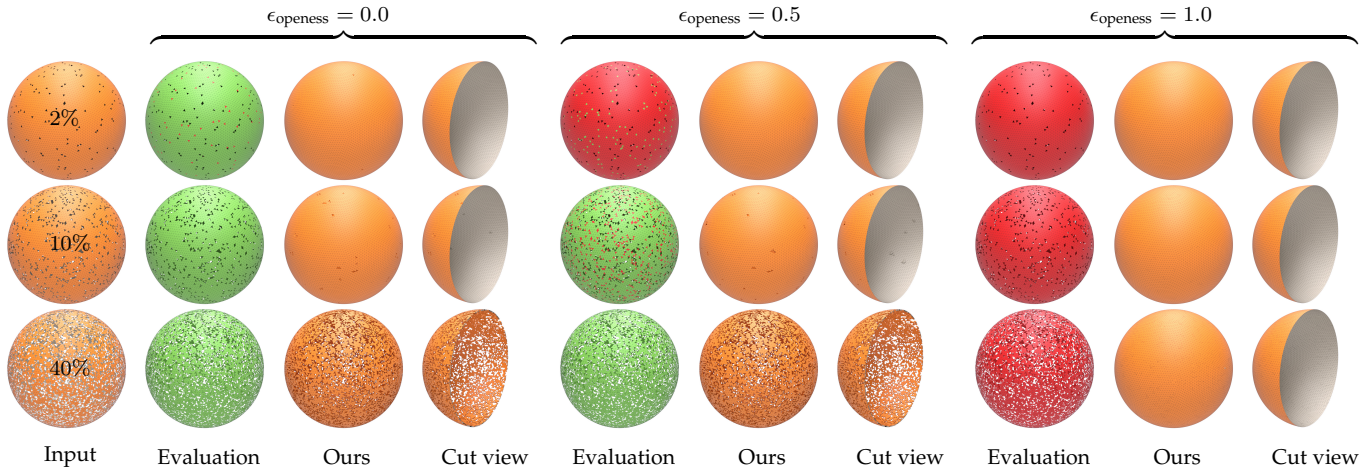

Fig. 3: *Study on opening ratio and $\epsilon_{openness}$:* *From top to bottom, we removed different portions of a sphere as input (left) to make the opening ratio of the sphere as 2%, 10%, and 40%, respectively. Then, we set the openness threshold $\epsilon_{openness}$ as 0.0, 0.5, and 1.0, respectively, from left to right. Faces that are identified as open faces are highlighted in green, and non-open faces are highlighted in red. The front side of the face is orange, while the back side is white.*

identified as open, leading to their offsetting by our method, ultimately forming a volumetric shell. When $\epsilon_{openness} = 0.0$, most faces are categorized as open, and our method leans toward thin shell formation. In short, our method allows switch preference from filling holes to forming volumetric shells based on different openness thresholds.

Statistical Analysis on ShapeNet. Table 2 demonstrates our method is significantly effective across a diverse range of shapes. Individual details are provided for the eight most frequent categories, while all other categories are collectively summarized as "others". Notably, for the "car" category, which has large holes and intricate inner structures, our method excels in hole filling and removing inner invisible parts, creating an outer shell. This can lead to larger Hausdorff Distance (HD) and lower Peak Signal-to-Noise Ratio (PSNR). Overall, our method showcases prowess in handling complex shapes effectively over previous methods in all categories.

Disconnected Components. For the example containing disconnected components, our method merges the inter-

TABLE 2: *The statistical analysis of 1000 models in ShapeNet, categorized into various classes.*

	Samples	InputFace	Face	HD	LFD	PSNR	Mem.	Time(s)
airplane	86	51K	32K	0.02	1.5e0	61.3 (58.3*)	955 MB	23.4
car	143	65K	94K	0.06	6e-1	50.1 (40.4*)	1574 MB	47.9
chair	135	19K	20K	0.01	8e-1	61.6 (61.6*)	604 MB	10.8
lamp	39	16K	13K	0.01	2.4e0	61.6 (60.7*)	418 MB	5.7
rifle	54	26K	14K	0.02	1.1e0	65.5 (66.9*)	425 MB	7.9
sofa	60	17K	10K	0.02	6.7e-2	65.3 (65.3*)	333 MB	4.8
table	136	11K	6K	0.01	3.1e-1	62.3 (66.7*)	266 MB	3.1
watercraft	38	55K	34K	0.02	5.3e0	56.9 (55.2*)	854 MB	20.5
others	309	28K	20K	0.02	9.2e-1	60.1 (60.3*)	665 MB	15.3

sected components through inner removal while preserving all other disconnected components, as shown in Fig. 4.
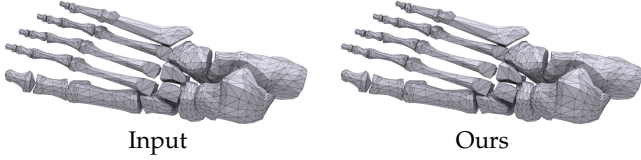

Input   Ours

Fig. 4: *Example of processing disconnected components: Given the input mesh with disconnected components, our method merges intersected components while preserving other components.*

2 APPLICATIONS

We present four downstream applications of our method, mesh simplifications (Fig. 5), Boolean operations on meshes (Fig. 6), geodesic distance computation (Fig. 7), and fluid simulation (Fig. 8). It is evident that meshes repaired by our method facilitate these applications, while the input meshes cannot be used due to their geometric and topological errors. On the other hand, combining the latest mesh repairing techniques, VR+VM results in the loss of original geometric structures, rendering the simulation results useless (see Fig. 8).
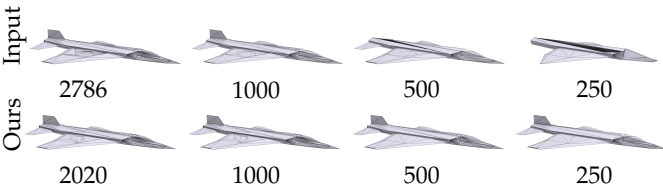

Fig. 5: *Example application of mesh simplification: The input airplane model is shown without (top) and with (bottom) using our mesh repair before making quadratic error metric (QEM) simplification [1]. The model repaired by our method can better preserve its original shape after simplification.*

2.1 Proof of Watertight Manifold

Theorem 2.1. *The number of adjacent triangles of any edge in the extracted mesh $M_{interface}$ is even:*

$$\forall e_i \in \mathcal{E}_{interface} \quad |Adj(e_i)| \equiv 0 \pmod 2 \tag{1}$$

Proof. Given an edge $e_i \in \mathcal{E}_{\text{partition}}$, the adjacent cells form a loop, effectively segmenting the 3D domain into N cells

M_0 M_1 M_0 not M_1 M_0 M_1 M_0 not M_1
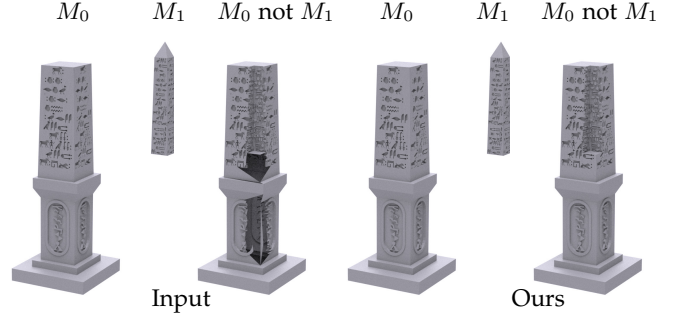
Input   Ours

Fig. 6: *Example application of boolean operation: After applying the boolean operation [2] directly to the input meshes, the resulting mesh is broken due to wrong orientation at M_1's bottom and tiny gaps, which may not be visible in the image. However, our method can successfully repair the mesh, enabling the boolean operation to output the correct result.*
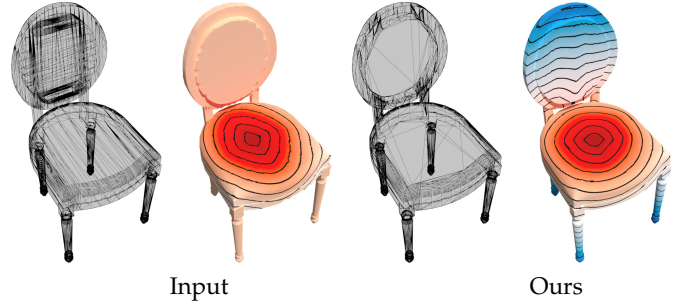

Input   Ours

Fig. 7: *Example application of geodesic distance computation: Computing geodesic distances on the input mesh through the Heat method [3] resulted in incorrect results due to disconnectivity and inner structure. However, our method produced a more desired distance map.*

surrounding the edge, where $N = |\text{Adj}(e_i)|$. If $|\text{Adj}(e_i)| \equiv 1 \pmod 2$, each face in $\text{Adj}(e_i)$ that's adjacent to the edge (e_i) separates the 3D domain into an odd number of cells.

It is important to note that face $f^i_{\text{interface}}$ is extracted in $M_{\text{partition}}$ if and only if its two adjacent cells are labeled internal and external, respectively. This condition contradicts the assumption of an odd number of cells forming a loop, where any neighboring cells are designated as one internal and one external.

Therefore, $\forall e_i \in \mathcal{E}_{interface} \quad |\text{Adj}(e_i)| \equiv 0 \pmod 2$. □

Theorem 2.2. *If the number of adjacent triangles of any edge $e \in \mathcal{E}$ in M is even, after any edge collapse operation, the number of adjacent triangles of any edge $\hat{e} \in \hat{\mathcal{E}}$ in the output mesh $\hat{M}$ is*
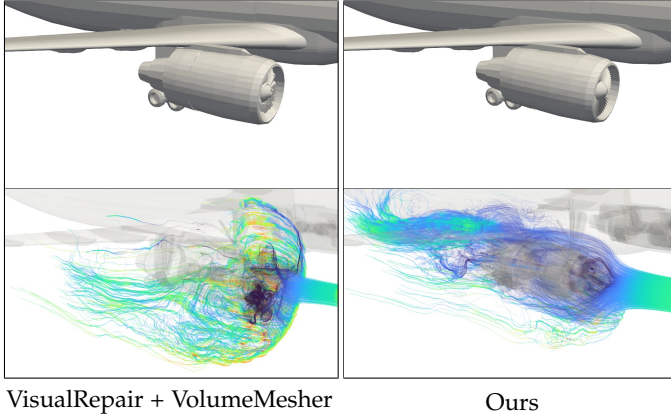
VisualRepair + VolumeMesher   Ours

Fig. 8: *Example application of flow simulation: The airplane mesh repaired using VisualRepair + VolumeMesher results in a closed engine model that does not allow airflow passage. In contrast, our method preserves the original structure of the engine and allows for the correct flow of fluid through the model.*

also even.

Proof. The edge collapse process for any edge in origin mesh $e_i = (v^{i0}, v^{i1})$ could be viewed as firstly move v^{i1} to v^{i0}, then remove all triangles whose corners has two v^{i0}. The edges in collapsed mesh $\hat{M}$ could be split into two categories: those that do not share the vertex v^{i0} and those that do.

For edges in collapsed mesh $\hat{M}$ that do not share the vertex v^{i0}: $\forall \hat{e}_a \in A = \{\hat{e} = (v_x, v_y) | \hat{e} \in \hat{\mathcal{E}}, v_x, v_y \neq v^{i0}\}$, their adjacent faces are not changed during the collapse. This means, for each $\hat{e}_a \in \hat{\mathcal{E}}$, we have $|\hat{\text{Adj}}(\hat{e}_a)| \equiv 0 \pmod 2$ as given in precondition.

For edges in collapsed mesh $\hat{M}$ that share the vertex v^{i0}: $\forall \hat{e}_b \in B = \{\hat{e} = (v^{i0}, v_y) | \hat{e} \in \hat{\mathcal{E}}, v_y \neq v^{i0}\}$, the set of adjacent faces in collapsed mesh $\hat{M}$ of edge $\hat{e}_b$ is the union of the set of faces in origin mesh adjacent to edge (v^{i0}, v_y) or (v^{i1}, v_y) and remove whose faces with edge (v^{i0}, v^{i1}), thus $\hat{\text{Adj}}(\hat{e}_b) = \text{Adj}((v^{i0}, v_y)) \cup \text{Adj}((v^{i1}, v_y)) - \{(v^{i0}, v^{i1}, v_y)\}$, now that the faces to be removed are shared in both sets: $\{(v^{i0}, v^{i1}, v_y)\} \subset \text{Adj}((v^{i0}, v_y))$, $\{(v^{i0}, v^{i1}, v_y)\} \subset \text{Adj}((v^{i1}, v_y))$.

We have $|\hat{\text{Adj}}(\hat{e}_b)| = |\text{Adj}((v^{i0}, v_y))| + |\text{Adj}((v^{i1}, v_y))| - 2|\{(v^{i0}, v^{i1}, v_y)\}| \equiv 0 \pmod 2$

Notice that v^{i1} has been removed from $\hat{M}$, we have $\hat{\mathcal{E}} = A \cup B$. As a result, the number of adjacent triangles of any edge $\hat{e} \in \hat{\mathcal{E}}$ in $\hat{M}$ is also even.

$\square$

Theorem 2.3. *Given the input mesh, in which the number of adjacent triangles of any edge is even, we could create a watertight manifold output mesh by repeatedly splitting the edge.*

Proof. If the input mesh has non-manifold edges, select one of them named (v_a, v_b) and a pair of adjacent faces (v_a, v_b, v_1), (v_a, v_b, v_2). Then, we replace these two faces with $(v_1, v_a, v_{\text{new}})$, $(v_1, v_{\text{new}}, v_b)$, $(v_2, v_a, v_{\text{new}})$, and $(v_2, v_{\text{new}}, v_b)$, where v_{new} is on the midpoint of (v_a, v_b) as shown in Fig. 9. Notice that the number of adjacent faces of newly added edges (v_a, v_{new}), (v_b, v_{new}), (v_1, v_{new}), and
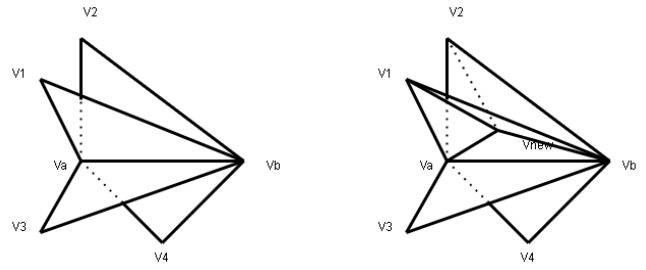

Fig. 9: *Split a non-manifold edge by adding a new vertex for a pair of faces.*

(v_2, v_{new}) is exactly 2. The number of adjacent faces of edge (v_a, v_b) is reduced by 2.

As a result, in the newly created mesh, the number of adjacent triangles of any edge is even; thus, we can repeatedly split the edge until there are only manifold edges. Given that the number of adjacent triangles of any edge is even, the output mesh is also watertight; otherwise, there would be an edge with only one adjacent face, which leads to a contradiction. Hence, we could split all non-manifold vertices to produce a watertight manifold output mesh. $\square$

Theorem 2.4. *The output mesh of our method is a watertight manifold.*

Proof. According to Theorem 2.1, the number of adjacent triangles for any edge in the extracted mesh $M_{\text{interface}}$ is even, meeting the precondition of Theorem 2.2. Consequently, the triangulation, preserving boundaries, ensures that the simplified mesh also satisfies the precondition of Theorem 2.3—that the number of adjacent triangles for any edge in the mesh is even. Hence, by Theorem 2.3, the output mesh is guaranteed to be watertight and manifold. $\square$

REFERENCES

[1] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '97. USA: ACM Press/Addison-Wesley Publishing Co., 1997, p. 209–216.

[2] Q. Zhou, E. Grinspun, D. Zorin, and A. Jacobson, "Mesh arrangements for solid geometry," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, pp. 1–15, 2016.

[3] K. Crane, C. Weischedel, and M. Wardetzky, "The heat method for distance computation," *Communications of the ACM*, vol. 60, no. 11, pp. 90–99, 2017.