Mesh repairing using topology graphs

Article *in* Journal of Computational Design and Engineering · December 2020

DOI: 10.1093/jcde/qwaa076

CITATIONS
10

READS
462

3 authors:

Jérôme Charton
Korea Institute of Science and Technology
9 PUBLICATIONS **113** CITATIONS
SEE PROFILE

Stephen Baek
University of Virginia
85 PUBLICATIONS **1,229** CITATIONS
SEE PROFILE

Youngjun Kim
Imagoworks Inc.
83 PUBLICATIONS **1,359** CITATIONS
SEE PROFILE

OXFORD CDE

Journal of Computational Design and Engineering, 2020, 00(0), 1–17

doi: 10.1093/jcde/qwaa076
Journal homepage: www.jcde.org

RESEARCH ARTICLE

Mesh repairing using topology graphs

Jerome Charton [1], Stephen Baek[1] and Youngjun Kim[2,*]

[1]Department of Industrial and Systems Engineering, University of Iowa, Iowa City, IA 52242, USA and [2]Center for Bionics, Korea Institute of Science and Technology, Seongbuk-gu, Seoul 02792, Republic of Korea

*Corresponding author. E-mail: junekim@kist.re.kr

Abstract

Geometrical and topological inconsistencies, such as self-intersections and non-manifold elements, are common in triangular meshes, causing various problems across all stages of geometry processing. In this paper, we propose a method to resolve these inconsistencies using a graph-based approach. We first convert geometrical inconsistencies into topological inconsistencies and construct a topology graph. We then define local pairing operations on the topology graph, which is guaranteed not to introduce new inconsistencies. The final output of our method is an oriented manifold with all geometrical and topological inconsistencies fixed. Validated against a large data set, our method overcomes chronic problems in the relevant literature. First, our method preserves the original geometry and it does not introduce a negative volume or false new data, as we do not impose any heuristic assumption (e.g. watertight mesh). Moreover, our method does not introduce new geometric inconsistencies, guaranteeing inconsistency-free outcome.

Keywords: mesh repairing; geometrical/topological inconsistencies; self-intersections; non-manifold meshes; topology graphs

1 Introduction

Triangular meshes are a widely adopted standard for representing surfaces in many applications, including computer graphics, computer-aided (geometric) design (CAD/CAGD), games and entertainment, virtual reality simulations, computer-assisted surgery, digital dentistry, etc. In triangular meshes, the geometry of an object is defined by the vertex coordinates of triangles, while the edges and triangular faces encode the topology, providing a light and concise representation of graphical data.

Despite the benefit of being light weight and easy to implement, triangular meshes are, however, prone to *geometrical* and *topological inconsistencies*, such as those listed in Fig. 1. For example, geometrical inconsistencies occur when vertices or faces are spatially coincident (e.g. duplicate vertices, overlapping surfaces, self-intersections). On the other hand, topological inconsistencies occur when the connectivity among mesh elements (i.e. vertices, edges, and faces) is invalid (e.g. non-manifold). More formally, geometrical inconsistencies are defined as cases in which the immersion of a surface is not injective, such that a point in the ambient space, in which the surface is immersed, is occupied by the surface more than one time (Fig. 2a–c). On the other hand, topological inconsistencies are defined as when not all points on the triangular mesh are locally homeomorphic to the Euclidean 2-space (Fig. 2d–f). Sources of such inconsistencies are ubiquitous across all stages of geometry processing, ranging from, for instance, smoothing and mesh refining, to feature extrusion and isosurface extraction.

There is already a body of literature as will be discussed shortly in the next section, alongside commercially and publicly available software programs (3D Systems, 1997–2016; Attene, Campen, & Kobbelt, 2012; Cignoni et al., 2011; Aguerre, Charton, Desbarats, & Recur, 2013) to resolve such inconsistencies. However, a majority of these methods begin with an assumption that a triangular mesh is a boundary representation of some solid object occupying a volume. Hence, the methods attempt to estimate the solid volume first, by performing hole

Received: 16 June 2020; **Revised:** 23 October 2020; **Accepted:** 31 October 2020

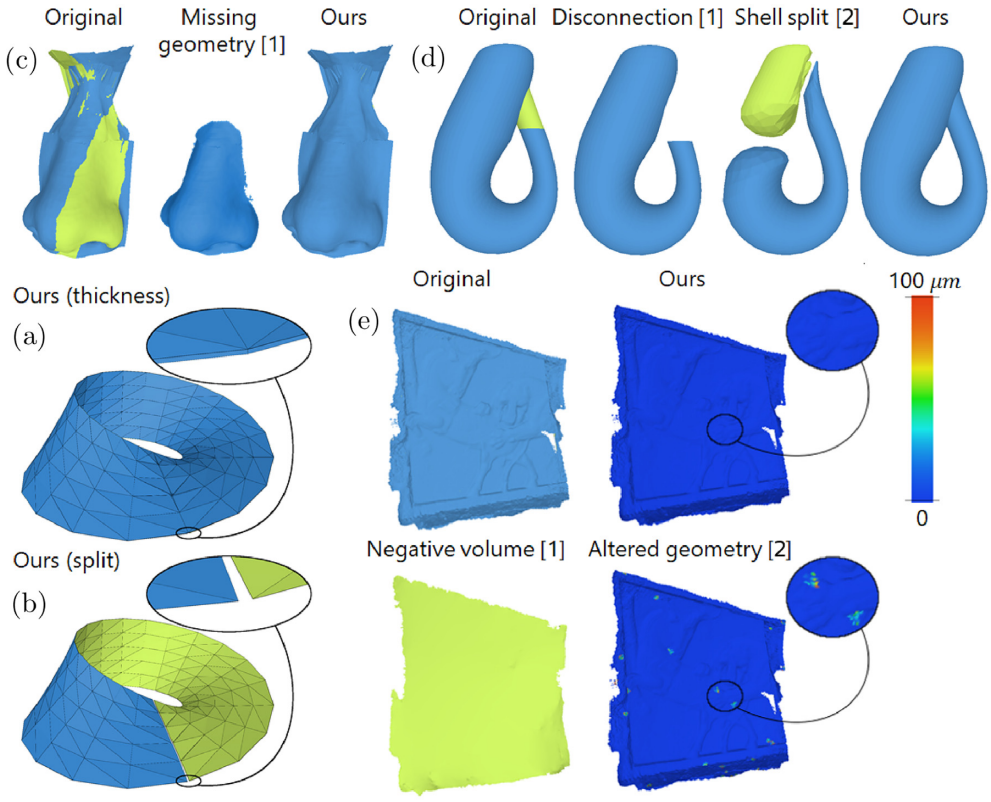


Figure 1: Our method can repair a variety of geometrical and topological inconsistencies in triangular meshes. Displayed here are a few examples of virtual and real-world mesh data with self-intersections, non-manifold features, etc. Blue faces are triangles with outward normals and green are the ones with inward normals. Our method demonstrates a reliable performance in contrast to other benchmark methods (Attene, 2010–2016; 3D Systems, 1997–2016), which tend to cause problems such as falsely estimated volumes, an incorrect split of shells, unwanted elimination of geometry, etc. Note also that, as demonstrated in the Möbius strip example, our method can provide the user options to resolve topological conflicts. (a) The *Möbius strip* solved by offsetting. (b) The *Möbius strip* solved by disconnections. (c) The nose model. (d) The Klein bottle. (e) Persepolis—Relief Lion Bull model (https://sketchfab.com/3d-models/persepolis-relief-lion-bull-7e37e222c31b4b5eb424b703ec306d58).

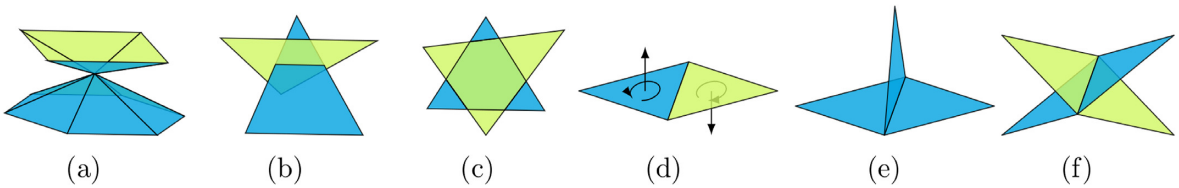


Figure 2: Examples of geometrical (a–c) and topological (d–f) inconsistencies: (a) two vertices with the same coordinate values; (b) non-coplanar self-intersection between two faces; (c) coplanar intersection; (d) two opposing faces with conflicting orientations; (e) non-manifold edge with more than two (odd-numbered) adjacent faces; and (f) non-manifold edge with even number of adjacent faces.

filling, gap closing, etc., before resolving inconsistencies. Unfortunately, the assumption of solid can lead to plenty of problems in practice. For example, the bas relief model in Fig. 1(e) captures only one facet of a presumably larger solid. When a large portion of the solid is missing as such, the heuristics to estimate the solid may produce a false estimation of volume, resulting in negative volume with falsely defined interior and exterior, as shown in the figure. Moreover, like the Klein bottle example in the same figure, estimation of volume may not even be possible in some models. Many existing algorithms fail to repair self-intersections and other inconsistencies in such cases.

Furthermore, the existing mesh repairing tools and methods remove substantial regions around the problem locations against the intent of the user. For example, Fig. 1c displays a scan of a nose, obtained for a surgical purpose, containing many flipped normals. During the process of resolving such inconsistencies, the existing tools often erase significant portions of the geometry while trying to eliminate the source of conflicts. A significant amount of useful geometric information could be lost as a result, rendering practical issues in many applications.

In this paper, we propose a more generic and rigorous method to resolve the geometrical and topological inconsistencies, with neither of those problems. The proposed method takes a triangular mesh or a triangle soup (e.g. stereolithography) as input and returns a set of oriented manifold meshes with all the inconsistencies resolved. The method first performs an initial clean up by converting geometrical inconsistencies to topological inconsistencies. During this operation, self-intersections and other geometrical inconsistencies are detected and remeshed. The result of this first step is a mesh free of geometrical inconsistencies but topological inconsistencies still remaining. The method then constructs a topology graph by partitioning the

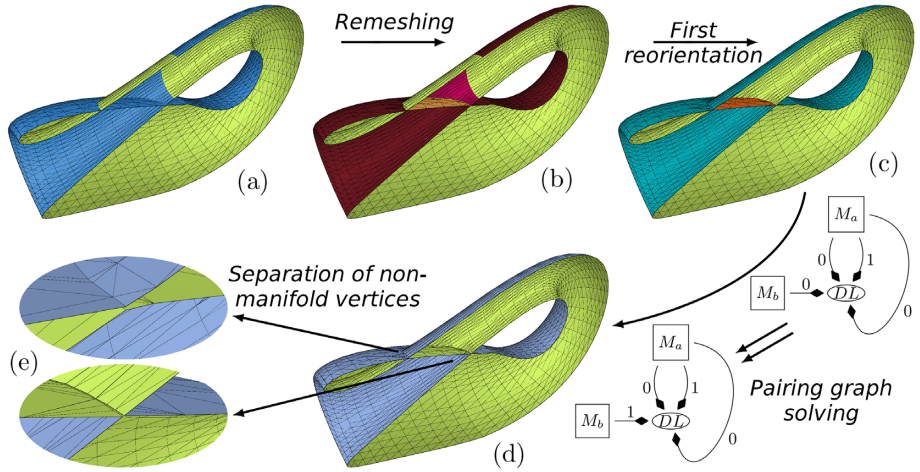


Figure 3: The global algorithm presented on the example of a half-cut of the Klein bottle surface.

mesh along topologically conflicting boundaries. These boundaries, called *dividing lines*, as well as surface partitions divided thereby, become the nodes of the topology graph, interconnected by the graph edges based on their adjacency (see Figs 3, 6, and 7). Finally, building upon this novel topology graph, we define pairing operations and conflict handling schemes to resolve topological inconsistencies locally around the dividing lines. The final mesh produced in this last step not only is guaranteed to be free of geometrical and topological inconsistencies but also preserves the original geometry within the user-defined alteration tolerance $\sigma \geq 0$. While capable of being fully automated, the method also provides manual control on how topological conflicts are managed via a scoring system. This allows the user to cope with various topological ambiguities that might occur in real-world problems.

1.1 Related work

Many geometry processing operations on triangular meshes, for instance, curvature/normal estimation, decimation, refining, and smoothing, demand the input mesh to be free of geometric and topological inconsistencies. Hence, a substantial body of geometry processing literature has been dedicated to computational methods to resolve such inconsistencies. According to an extensive review of Attene, Campen, and Kobbelt (2013), these methods can be summarized into two categories, namely *global* and *local* approaches.

Global approaches impose an assumption that a mesh is essentially the boundary representation of a solid. These approaches estimate in/out information of a solid and construct a watertight mesh accordingly to resolve inconsistencies. The methods in this category can further be divided into several subcategories. Certain methods intend to fill the volume of an object using a volume representation, such as a 3D image with the aim of closing holes and fixing inconsistencies (Andújar, Brunet, & Ayala, 2002; Bischoff, Pavic, & Kobbelt, 2005; Jacobson, Kavan, & Sorkine-Hornung, 2013; Lu, Quadros, & Shimada, 2017; Imai, Hiraoka, & Kawaharada, 2014), while other methods propose a surface-based approach. They propagate surface normals continuously over the surface (Sagawa & Ikeuchi, 2008; Schertler, Savchynskyy, & Gumhold, 2016; Voutchkov, Keane, Shahpar, & Bates, 2017). Still in a pure surface-based approach, considering closed mesh, parity-counting methods (Nooruddin & Turk, 2003; Ju, 2004; Spillmann, Wagner, & Teschner, 2006; Zhou, Grinspun, Zorin, & Jacobson, 2016) count all the transitions between the exterior and the interior of the shell and reorient the mesh, piece by piece. Besides these methods that mainly aim to resolve inconsistencies, other methods aim to fill large holes. Within these methods, Furukawa, Itano, Morisaka, and Kawasaki (2007) and Immonen (2018) reconstruct the outer hull of the object using lines of sight to detect missing acquisition angles and fill missing data. Centin and Signoroni (2018) fill large holes between disconnected shells. In addition, Hornung and Kobbelt (2006) and Hétroy, Rey, Andújar, Brunet, and Vinacua (2011) propose multi-resolution representations in order to determine an optimal topology of the reconstructed mesh. In an alternative context, where a sequence of meshes is used, Feng, Zhang, Huang, Wang, and Bao (2010) present a solution that rectifies the topology of a model via skeleton matching between successive elements in the sequence. The assumption of solid, however, renders a problem when the mesh surface does not cover large enough area of the solid boundary, such as the volume can be falsely filled or in/out can be misdeemed (e.g. like the bas relief models in Fig. 1e).

On the other hand, local approaches isolate sources of inconsistencies and attempt to fix them locally around each of the sources. These approaches readjust vertex locations through operations such as edge hammering, face lifting, and edge swapping (Yamakawa and Shimada 2009); removing self-intersecting elements and filling the holes generated thereby (Attene 2010); growing manifold elements from a seed face (Jung, Shin, & Choi, 2004; Zaharescu, Boyer, & Horaud, 2007); and locally remeshing regions of self-intersections based on local voxel occupancy (Bischoff & Kobbelt, 2005; Ju & Udeshi, 2006; Campen & Kobbelt, 2010). Afterward, topological inconsistencies are resolved by a 'cut and restitch' approach, where the non-manifold elements are duplicated and decomposed first, and then reconnected to build manifold shells. The common strategy to this end is to minimize the number of resulting manifold shells (Rossignac and Cardoze 1999); to fill the inside of the mesh with tetrahedra and create a 3-manifold (Attene, Giorgi, Ferri, & Falcidieno, 2009); to convert the whole mesh into a watertight shell using surface offsetting or removal; or to leave the user to decide from among different topological combinations at each problem location (Guéziec, Taubin, Lazarus, & Horn, 2001).

In terms of the completeness of the output, the global approaches tend to be more preferable over the local approaches, as they typically return an output mesh that is globally repaired,

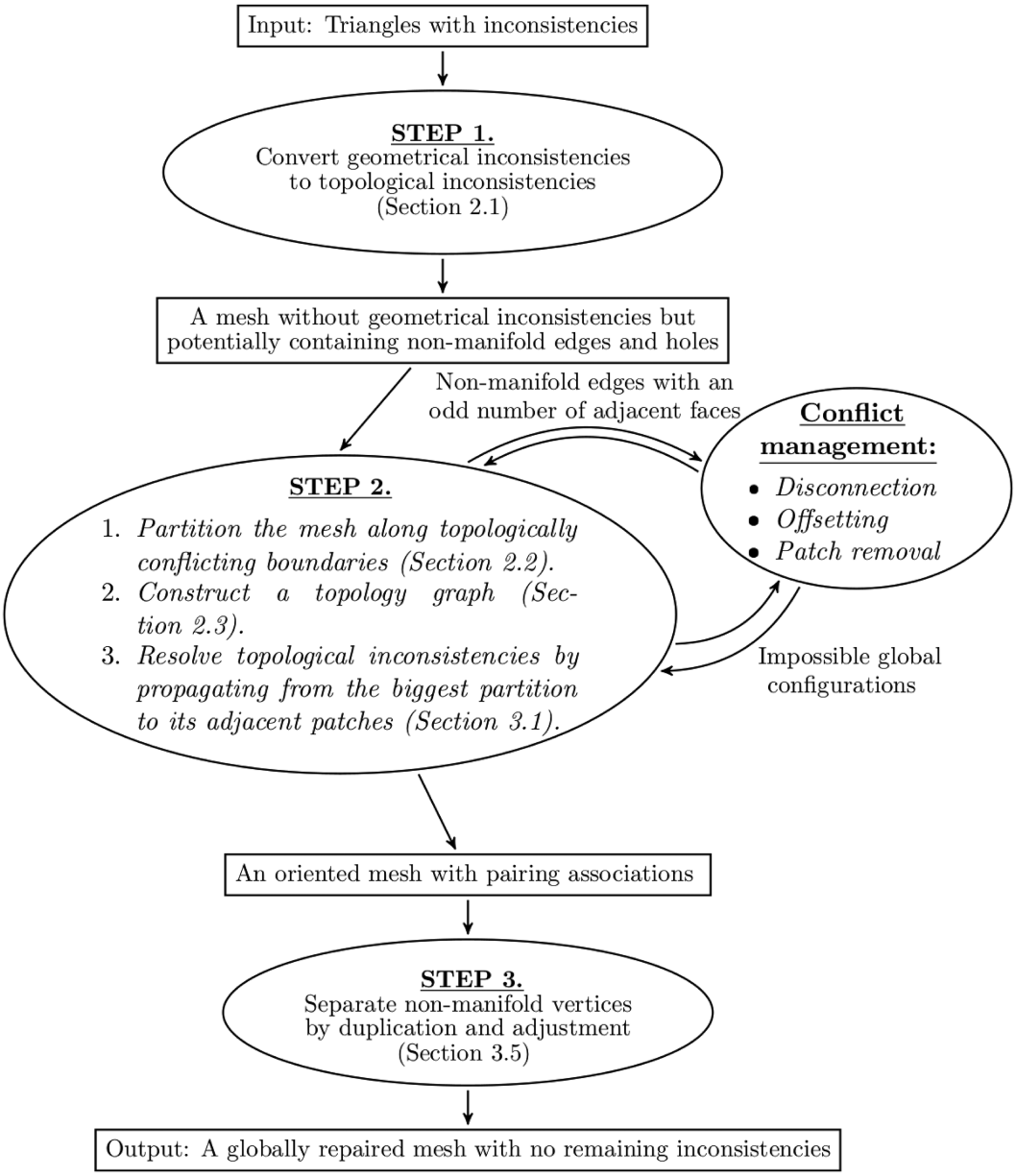


Figure 4: Overview of the method. The rectangular blocks represent the states of the data and oval blocks represent the treatments. The conflict management is a side processing used to resolve impossible graph configuration.

while the local approaches return partially repaired objects with potentially remaining inconsistencies. However, at the same time, global approaches alter or damage the original geometry quite substantially during the volume estimation. Hence, global approaches are more limited to applications where the mesh covers large enough area of the solid boundary but contains a large number of defects all over, whereas local approaches are mainly for sparsely scattered defects.

1.2 Overview of the method

The method proposed in this paper is a hybrid of the global and the local method, taking advantage of both the global methods, which guarantee the completeness of repairing throughout the mesh, and the local methods, which do not introduce the false assumption of solid. Our method does not add or remove any geometric information to or from the original data. Besides, the amount of modification can be limited via the user-controllable parameter $\sigma \geq 0$.

As illustrated in Fig. 4, the method consists of three steps. First, triangles are processed into a mesh free of geometrical inconsistencies. At this stage, only geometrical inconsistencies such as self-intersections and duplicated elements are focused. Hence, existing topological inconsistencies remain and new topological inconsistencies can be introduced as a result of geometrical repairing. Those topological inconsistencies are handled by the second and third steps. In the second step, the method constructs a topology graph and repairs topological inconsistencies by reorientation and pairing of the triangles. During this stage, surface patches are created by partitioning the mesh along topologically conflicting areas. The patches are then reoriented and paired to resolve the topological inconsistencies.

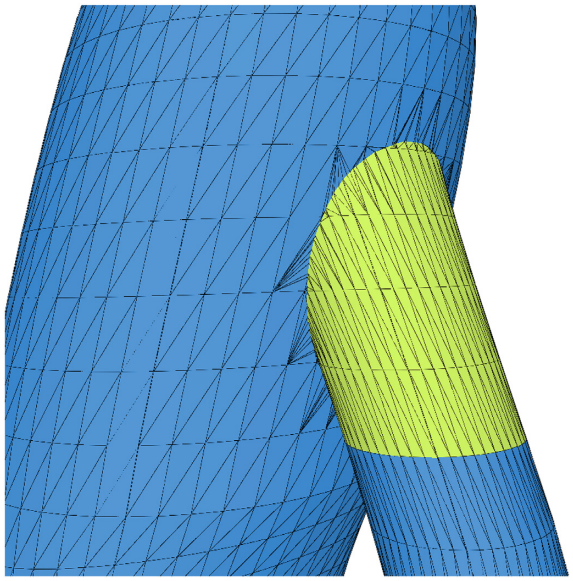


Figure 5: The Klein bottle model after remeshing self-intersections. The geometric inconsistencies (self-intersecting faces) have been resolved in this way, while there are topological inconsistencies still remaining.

Where the pairing is not trivial, patches are either disconnected, offset, or removed depending on the user-defined preference. Finally, in the third step, non-manifold elements are separated based on the pairing result. During this step, vertices can be readjusted to avoid duplication, within the user-controlled tolerance $\sigma \geq 0$. We define a self-collision-free operation for readjusting vertex locations such that no additional geometrical inconsistencies are introduced.

2 Topology Graph

The topology graph is a crucial component of the method proposed in this paper. To construct a topology graph, we first aim to resolve geometrical inconsistencies, such as duplicated elements, isolated vertices, degenerate elements, and self-intersections. This produces a mesh free from geometrical inconsistencies but still contains topological inconsistencies, which is then partitioned into separate manifold patches along the dividing lines where the topology is inconsistent. Finally, based on the adjacency among such manifold patches, as well as the dividing lines, a topology graph is constructed. Below are more technical details.

2.1 Converting geometrical inconsistencies to topological inconsistencies

For a given input, our method merges duplicated elements, discards isolated vertices, and collapses degenerate elements by using the method presented in Attene (2010). The stray elements after the initial cleaning (e.g. one of the faces preserved from a duplicated pair of faces) are tagged '*leftover*' so that they can be referenced and treated in the later step (Section 3.2.2).

After the initial cleaning, we then detect self-intersections and remesh them (Fig. 5). Here, two types of self-intersections must be distinguished, namely *coplanar* and *non-coplanar* intersections. Detecting these self-intersections is a non-trivial task. Resolving a coplanar intersection is a 2D problem, in which the union of colliding faces forms a 2D polygon. Hence, coplanar collisions can be resolved simply by triangulating the polygon formed by the union of colliding faces using, for instance, Delaunay triangulation (Shewchuk 2008). On the other hand, a non-coplanar intersection results in a line segment at the intersection between the intersecting triangles. To this end, we employ a method presented in Attene (2014). During the remeshing, Attene uses an exact arithmetic representation of vertices. This representation is preserved all along with the algorithm and is used to compute all geometrical computations. The key idea is to split a self-intersecting triangle into polygonal faces by adding edges along the intersection lines. The polygons formed by added edges are then triangulated.

Detecting self-intersections on a mesh is a non-trivial task. A naive method for checking intersections between all potential pairs of faces will result in $\mathcal{O}(n^2)$ complexity, where n is the number of faces. To reduce the number of intersection tests, an axis-aligned bounding box bounding volume hierarchy (AABB BVH) is widely adopted. This allows limiting the search space into a small subset where AABBs intersect, and thus, can speed up the intersection test substantially (Ericson 2004).

2.2 Surface reorientation

The result of the first step presented in Section 2.1 is a mesh composed of one or more shells free of geometrical inconsistencies. However, the mesh might still inherit topological inconsistencies, or, during the resolution of geometric inconsistencies, new topological inconsistencies might have been introduced.

For that, we first segment the triangles based on their topological orientations determined by the ordering of vertices, to construct *oriented manifold patches*. Adjacent triangles with the same ordering of vertices (i.e. the same topological orientation) are grouped to form a manifold patch. To construct such patches, we grow a patch from one randomly selected triangle to its adjacent neighbours across edges that have the same topological orientation. The growth of the patch stops at the boundary of the surface or where the growing front comes across a non-manifold edge that has more than two adjacent triangles due to some topological error. When growth stops at all fronts, the current patch is registered as a manifold patch and the process is repeated until there is no remaining face that does not belong to a patch.

After the partitioning is done, the patch segments are further merged to form a bigger patch whenever there exists a pair of patches where one of them can flip its orientation to match the orientation of the other. To facilitate this, we find the largest patch first, based on the total surface area of the patch. The orientation of the largest patch is propagated to its adjacent patches by flipping the orientation of the neighbouring patches accordingly. We propagate towards other neighbouring patches until the flipping of orientation is no longer trivial due to topological inconsistencies, e.g. at a patch boundary with more than two adjacent patches, or when there is a conflict between two (or more) propagating fronts, e.g. at the joining of the Möbius strip.

2.3 A topology graph

Finally, we construct a topology graph based on the manifold patches we have developed so far. Here, we first define the following terms for the convenience of discussion.

2.3.1 *Edge types*

A *full edge* is an edge that has two adjacent faces with consistent orientations. A *boundary edge* is an edge adjacent to a single

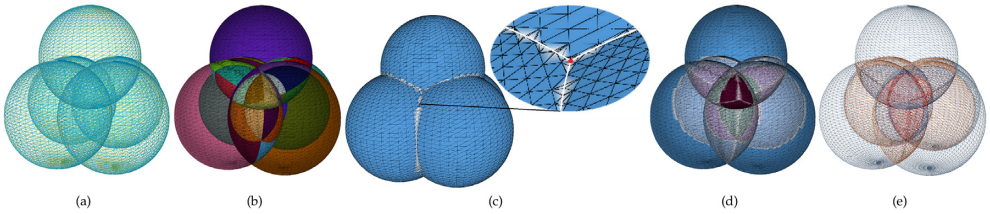


Figure 6: Example of the five colliding spheres model. The different phases of the proposed algorithm are shown. (a) Original model. (b) Decomposition in oriented manifold components (rendered in random colours). (c) After remeshing and initialization, vertices are classified into three types: normal vertices are rendered in blue; *proper dividing vertices* in white; and *limit dividing vertices* in red. (d) After the pairing of oriented manifold components (rendered in random colours). (e) Output (each oriented manifold component is rendered in random colours).

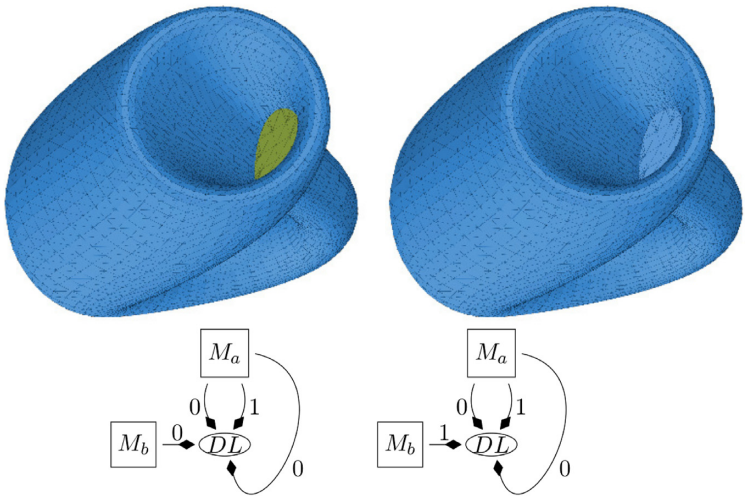


Figure 7: *Pairing graph* of the *Klein bottle* model. Left shows the initial configuration and right shows the final configuration, where M_a models the *oriented manifold component* of the body of the bottle and M_b models the *oriented manifold component* of the bottom.

face. A *non-manifold edge* is an edge with more than two adjacent faces. A *turning edge* is an edge with two adjacent faces with inconsistent orientation. Both full and boundary edges are topologically consistent, while non-manifold and turning edges are inconsistent and need to be remedied.

2.3.2 *Umbrellas and sub-umbrellas*

A set of neighbouring faces fanning out of a vertex v is called an *umbrella*. An umbrella is *complete* if none of its spokes, i.e. edges adjacent to v, is a boundary edge; otherwise, it is called *partial*. Meanwhile, an umbrella, either complete or partial, is *manifold* if all of the spokes are full edges. A *sub-umbrella* is a continuous subset of an umbrella. A sub-umbrella is called manifold if all edges between its faces are full edges.

2.3.3 *Dividing lines*

A dividing line is a chain of inconsistent edges. A dividing line comprised only of turning edges is called *turning dividing line*. Similarly, a dividing line comprised only of non-manifold edges is called *non-manifold dividing line* (white lines, Fig. 6c). It is noteworthy that a dividing line may pass through a vertex with a partial umbrella (i.e. touches the surface boundary). In such cases, we split the dividing line into two dividing lines at the boundary vertex and treat them separately, for convenience.

2.3.4 *Vertex types*

A vertex without inconsistent edges in the spokes of its umbrella is called *consistent*. A vertex on a dividing line is called a *dividing vertex*. Among dividing vertices, a vertex at the junction of dividing lines or at the end of a dividing line is called a *limit dividing vertex*. To make a distinction between *limit dividing vertex* and other dividing vertices, we call the latter *proper dividing vertices* (see Fig. 6c).

Based on such definitions, we now construct a topology graph. A topology graph $\mathcal{G}$ is comprised of nodes $\mathcal{N}$, connected via connections $\mathcal{C}$. Two types of nodes comprise $\mathcal{N}$ of a topology graph, namely dividing lines $\mathcal{D}$ and manifold patches $\mathcal{M}$. The two different types of nodes in $\mathcal{N} = \{\mathcal{D}, \mathcal{M}\}$ are connected by a connection in $\mathcal{C}$ if and only if they are adjacent. Note that two connected nodes must be of different types. In other words, a connection always connects a dividing line to a manifold patch, or vice versa. No connection between two manifold patches or between two dividing lines is allowed. This property will facilitate the pairing and manipulation of topological configurations in the later stage. Also, a manifold patch can be connected to a dividing line more than one time.

For example, consider the Klein bottle example in Fig. 7. The larger patch of the Klein bottle is connected three times to its unique dividing line. Consider the Möbius strip (see Fig. 10a), whose related topology graph is composed of two nodes, one for its unique patch and one for the reversing of the surface, and two edges between these two nodes representing the two connections around the *dividing line*.

In a graph $\mathcal{G}$, a connection is labelled either 0 or 1, based on the orientations of the nodes it connects. Initially, an arbitrary direction is assigned to each of dividing lines. Then, a

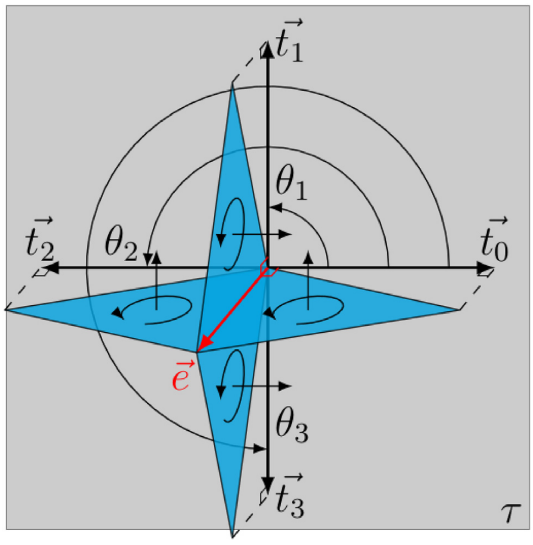


Figure 8: Computation of the θ_i angles. $\theta_i = \measuredangle(\hat{t}_0, \hat{t}_i)$, where $\hat{t}_i$ are directions orthogonal to e in the plane of adjacent faces, and $\hat{t}_0$ is arbitrarily chosen.

connection is labelled 0 if the corresponding manifold patch and the dividing line are in the same orientation, otherwise 1. The orientations of a connection and a dividing line are considered to be the same if the arbitrary direction assigned to the dividing line is the same as that of the adjacent faces that compose the connection (see Fig. 8).

3 Pairing on a Topology Graph

3.1 Orientation pairing

It is worth to emphasize again that inconsistencies remaining at this stage are purely topological, as the geometric inconsistencies have been fixed in the first stage. Hence, inconsistent features we would like to address include, adjacent faces with conflicting orientations, edges neighbouring with more than two faces, and vertices adjacent to several umbrellas.

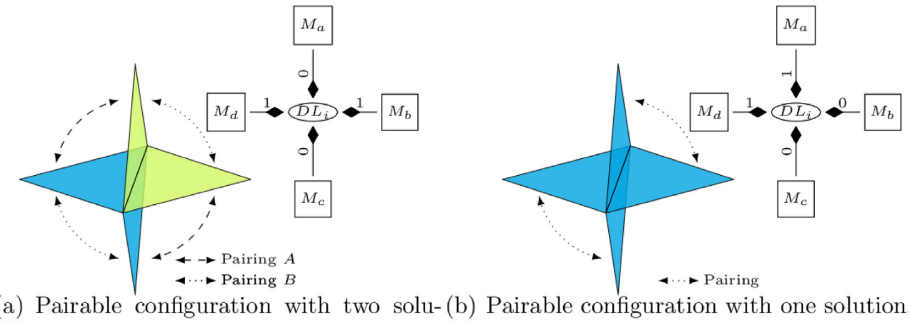
Given a topology graph $\mathcal{G}$, topological inconsistencies are resolved locally at each dividing line. Our approach is to propagate from a seed node in $\mathcal{G}$ and pair the orientation of the manifold patches to enforce consistent topology. We define local operations here in such a way that a local change of topology does not induce a new topological inconsistency on the nodes that have been already visited.

We first begin with the biggest manifold patch in $\mathcal{G}$, which we denote as $\mathcal{M}_0$, and flag it as *visited*, where the size of a patch is determined by the sum of the areas of its triangles. We then push all the dividing lines $D \in \mathcal{D}$ adjacent to $\mathcal{M}_0$ into a queue Q. While Q is not empty, we visit a dividing line D and try to pair the manifold patches $\mathcal{M}$ that are adjacent to D. When there are only two such patches, pairing is straightforward as the two can be paired together. When there are more than two patches, however, the patches are paired based on a criterion defined by the user. For each of these pairs, un-visited manifold patches may be flipped to enforce the consistent surface orientation. All patches $\mathcal{M}$ as well as the current dividing line D are marked *visited* and the un-visited dividing lines that are adjacent to $\mathcal{M}$ are pushed into Q.

Here, note that there can be local configurations that cannot be paired, such as odd connections (Fig. 9d). For such cases, we use one of the conflict management strategies defined in Sections 3.2 and 3.3.

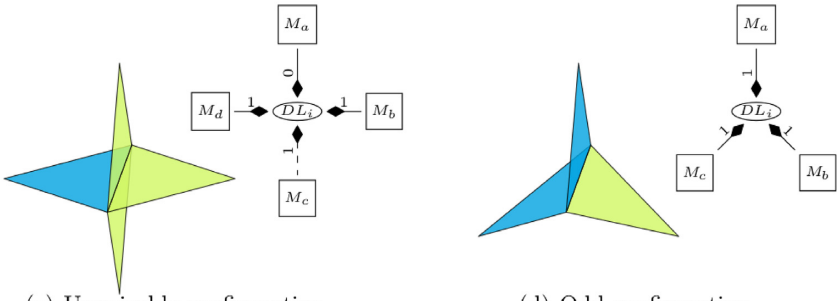
3.2 Handling conflicts via disconnection

As described above, there are cases where topological inconsistencies cannot be resolved locally through the pairing-based operations, e.g. the Möbius strip (see Fig. 10). For such cases, we let



(a) Pairable configuration with two solutions.

(b) Pairable configuration with one solution.



(c) Unpairable configuration.

(d) Odd configuration.

Figure 9: Examples of configurations of inconsistent edges on the left and their graph representations on the right.

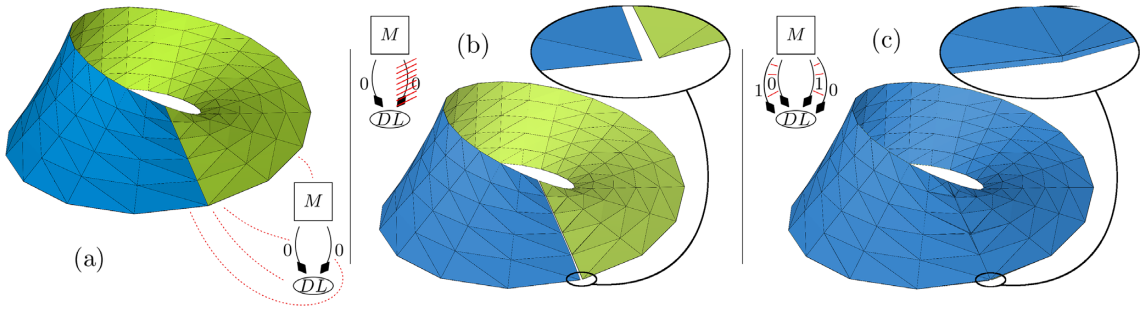


Figure 10: Example of topology graph construction and conflict management with the Möbius strip model. (a) Original model with its related topology graph. (b) Handling conflicts via disconnection. One of the two connections is cut. (c) Handling conflicts via offsetting. The whole patch is duplicated and offsetted creating a solvable configuration.

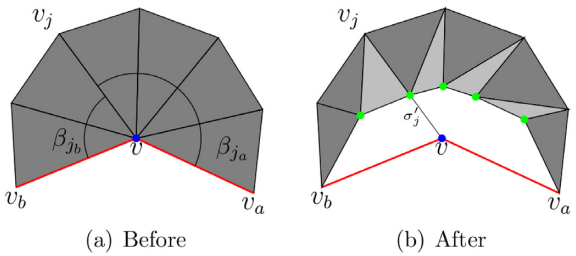


Figure 11: Disconnection of a vertex. Blue point: the inner vertex to disconnect. Red edge: non-manifold edges. Green points: duplicate vertex of v added to shrink edges. Light grey face: added faces.

the user decide from one of the following strategies to manage the issue.

Disconnection lets the user drop the connection between a dividing line in conflict and selected manifold patches. A disconnection will split a topologically inconsistent patch away from a dividing line to resolve the inconsistencies (see Fig. 10b). During this operation, a patch could be moved apart from the dividing line geometrically, i.e. with displacements in vertices, or just topologically without a geometric displacement. This can be controlled by a user-defined real number parameter $\sigma \geq 0$. The parameter σ can be understood as the distance of disconnection. When σ equals zero, no geometric modification is introduced during the disconnection. When σ is non-zero, the gap between the disconnected manifold patch and the dividing line is going to be σ or smaller.

3.2.1 *Collision-free disconnection*

It is worthwhile to note that a naïve disconnection may introduce a new self-intersection, as the displacement of a vertex may cause, albeit small, a rotation of a face which may cause a collision with another face. To this end, we define a collision-free disconnection operation as follows. The idea is to confine the geometric displacement of the patch boundary within existing triangles. To achieve this, let us focus on a proper dividing vertex v of a dividing line D in conflict. Let $e_j = (v, v_j)$ be an edge adjacent to v in the manifold to be moved apart, and v_j the opposite vertex in e_j. Then, as illustrated in Fig. 11, we duplicate a vertex at v, and contract the edge e_j by moving the duplicated vertex along the edge.

The amount of contraction is determined by the user-defined parameter σ. When v_a and v_b are other dividing vertices adjacent to v, and β_j is the minimum angle between $\angle(\overrightarrow{vv_j}, \overrightarrow{vv_a})$ and $\angle(\overrightarrow{vv_j}, \overrightarrow{vv_b})$, the actual amount of contraction (i.e. displacement of the duplicated vertex) σ'_j is determined from the following equation.

$$\sigma'_j = \begin{cases} \dfrac{\sigma}{\sin(\beta_j)} & \text{if } \beta_j < \pi/2, \\ \sigma & \text{otherwise} \end{cases} \tag{1}$$

Here, note that Equation (1) projects the user-defined amplitude σ onto the edge e_j such that the orthogonal distance after the displacement from the dividing line becomes σ. For the actual implementation, the first condition in Equation (1) can be re-written as $\frac{\sigma}{\sin(\beta_j)} = \frac{\sigma}{\sqrt{1-\rho_j^2}}$, where ρ_j is the dot product corresponding to β_j, since the computation of the dot product and the square root is faster than the computation of the sinusoidal function. Note when σ'_j is greater than or equal to the length of the edge e_j, we simply collapse the edge by deleting the faces attached to it.

3.2.2 *Disconnection scoring criteria*

Finally, building upon such definition of the collision-free disconnection operation, we propose the following strategy to resolve topological conflicts. First, for a given dividing line, we set the cardinality p of disconnection as 1 or 2, respectively, for odd or even number of connections sharing the dividing line. We then evaluate a *hole impact* score, a *duplication* score, and a *connectivity* score, which are defined in the following paragraph. The lexicographical order of three scores, in that order, defines the priority of disconnection. If there is no p in such connections, increase p by 2 and repeat from the first step.

The lexicographical scoring system we use for determining the priority of disconnection is given below.

(i) If a limit dividing vertex has an umbrella corresponding to the connection that contains a boundary edge, we give +1 to the connection. As such, if both of the limit dividing vertices contain a boundary edge in their umbrellas, the connection receives +2. This is to promote the minimization of the number of holes into the mesh.
(ii) If any of the faces directly adjacent to the connection were previously tagged as a duplicate face in Section 2.1, we give 0 to the connection, otherwise 1. This gives a priority to duplicated patches since they may be flat volumes.
(iii) Disconnect the one that has the smallest manifold patch. The size of a manifold patch is determined by the sum of the areas of faces contained in the manifold. This is to avoid the separation between large components of the mesh.

3.3 Handling conflicts via removal and offsetting

For the choices of removal/offsetting of conflicting *oriented manifold components* (see Fig. 10c), a greedy algorithm is proposed that

creates an ordered set of *oriented manifold components* $\{M_i\}$. It is initialized with the first *oriented manifold component* M connected to conflicting dividing line dl. To determine M, a priority order on the *oriented manifold component* related to dl is defined according to the following score.

(i) The negative value of the number of *end edges*. Note: If an edge is at both extremities, then it counts for two. This value minimizes the risk of introducing new holes.
(ii) If the representative face of the *connection* is tagged as a duplicated face in Section 2.1, then 0; otherwise, 1. Note: This gives the priority to duplicated patches since they may be empty volumes.
(iii) The size of M. Note: This minimizes the loss/addition of data.
(iv) The number of generated conflicting DLs if M is removed/duplicated.
(v) The number of resolved conflicting DLs if M is removed/duplicated.

The two last values minimize the addition of conflict on the *pairing graph* and maximize the number of conflicts solved by modifying M, respectively. $\{M_i\}$ grows until all DLs are resolvable. Finally, the modification is applied by removing/offsetting all elements of $\{M_i\}$. Offsetting an *oriented manifold component* consists in duplicating the inner data and reversing the duplicated surface of the hollow component, except for the vertices in inconsistent edges, which are not duplicated. To separate these two components, the duplicated vertices are offset with $\sigma/2$. This offsetting can be performed following Kim, Lee, and Yang (2004); Chen, Wang, Rosen, and Rossignac (2005); Liu and Wang (2011); and Li and Kim (2015). If the method used for offsetting introduces new self-intersections, the algorithm should restart from the beginning, but during remeshing both duplicated faces are removed and a constraint is added. A patch issued from a duplication cannot be duplicated. If no patch can be offset and conflicts remain, the conflictual strategy is used.

Algorithm 1 *Pairing graph* solving.

```
Solving DL with odd connections using disconnection, offsetting, or patch removal operations according to user parametrisation.
Let s be an empty score parametrised by the user.
while all oriented manifold components have not been visited do
    Let m be the biggest unvisited oriented manifold components.
    Fixing the orientation of m.
    for each unvisited dividing line dl adjacent to m ordered by size of the DLs do
        Extend s with the best score including dl.
        if s is conflictual then
            Solving the dl by conflict handler defined by the user.
        else
            Fixing the orientation of orientation manifold components adjacent to dl according to s.
        end if
    end for
end while
```

3.4 Pairing

3.4.1 *Pairing graph solving*

For solving the *pairing graph*, several solutions can be obtained using the scoring rule system. The best pairing configuration of the graph is determined by the following penalty score at each node independently.

(i) If a node does not perform a complete pairing of all its connections, then 0, otherwise 1.
(ii) *#R*: The sum of the areas of reversed faces. This forces the reversing of a patch only if it blocks the pairing.
(iii) *Or*: Number of pairs constructed oriented towards an inner interval, i.e. pairing B (Fig. 9a). This score adds a penalty when a material is removed. When a pair is constructed towards the inner, the later separation will result in a local erosion of the surface. Otherwise, it will result in a local offset.
(iv) *#Mg*: Number of pairs realized between two connections related to the same *oriented manifold component*. This score evaluates the merging/segmentation of the result.

Except for the first entry, the others are interchangeable or can be reversed to obtain the expected result. As in the previous scoring, the best score is the lowest in the lexicographical order.

In addition, two constraints can be applied. The first is preserving patch orientations (PO) and the second is creating in/out shells (*In/Out*). An in/out shell is a shell, potentially with inconsistent edges and holes, that can model a surface of an object, in any inner point of its mesh, i.e. provide a consistency definition of the interior/exterior of the shell. This configuration is the minimum requirement to perform Boolean operations by Charton, Kim, and Kim (2017) and Barki, Guennebaud, and Foufou (2015).

The *In/Out* constraint is applied by the propagation of the orientation from the largest patch to its adjacent patches using the priority of the number of edges of adjacent nodes.

Remark: These two constraints can also be combined. However, this can result in several conflicts, which will be handled according to the scoring rule chosen by the user (as seen above in Section 3.1).

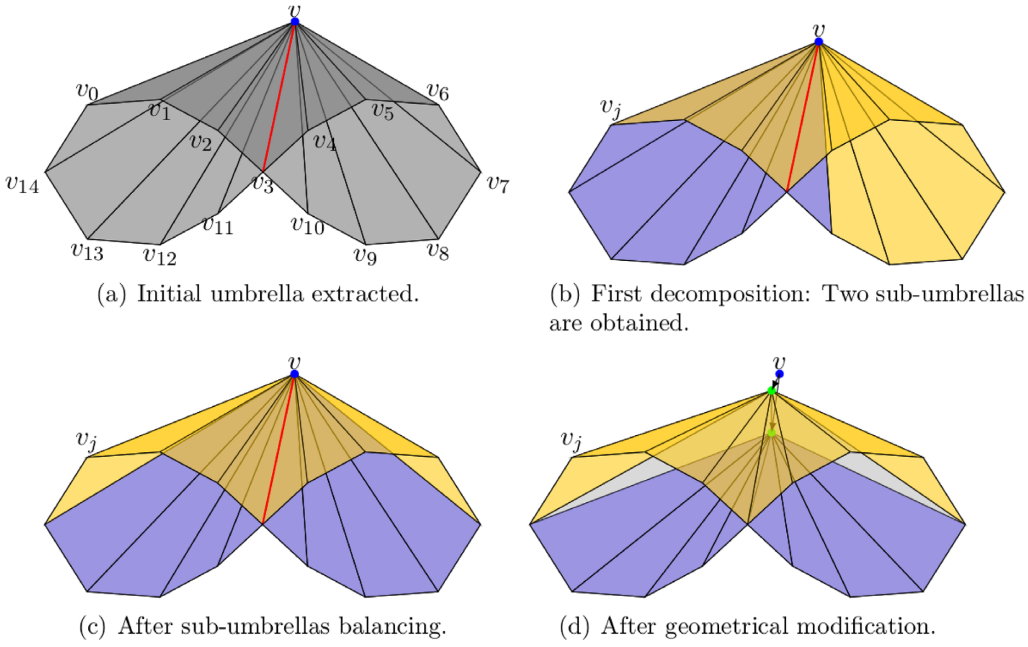
In the following, the pairing score will be written as

$$(Field_1, Field_2, ...)\{Constraints\}, \tag{2}$$

e.g. $(Or, \#Mg)\{In/Out\}$. The field zero is ignored because it is invariable. For each field, the reverse value can be used by adding "–" in front of it.

3.5 Separation of non-manifold edges and vertices

For this final stage, non-manifold elements are duplicated and optionally geometrically separated while preserving the self-intersection free condition of the mesh. This geometrical displacement is bordered by the alteration tolerance parameter σ. To do this, each vertex of the mesh is visited. If a non-manifold vertex is found, its neighbourhood is decomposed into umbrellas and then each umbrella again decomposed into sub-umbrellas to define to the final topology of the mesh after the separation. The first decomposition into umbrellas is made in such manner that, in the resulting umbrellas, two faces are successive if and only if they are adjacent through a regular edge or they are paired into the pairing relation defined in Section 3.1. By this construction, all faces have a predecessor and a successor (see Fig. 12a). The second decomposition into sub-umbrellas



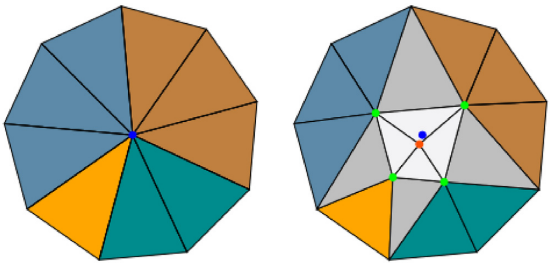
(a) Initial umbrella extracted.

(b) First decomposition: Two sub-umbrellas are obtained.

(c) After sub-umbrellas balancing.

(d) After geometrical modification.

Figure 12: Example of sub-umbrella decomposition with one non-manifold edge. v is the target vertex, $\{v_j\}$ are the neighbours of v, and the red edge is a inconsistent edge. (b) Orange and purple faces: sub-umbrellas obtained. (d) Green points: vertices added for each sub-umbrella to perform the contraction. Light grey faces: faces are added to avoid creation of holes.



(a) Umbrella after decomposition in sub-umbrellas

(b) After opening.

Figure 13: 2D representation of the separation of the duplicate vertices. Blue point: target vertex. (a) Faces are coloured randomly according to the sub-umbrella decomposition. (b) Green points: vertices added for the contraction of edge. Orange point: vertex added to the barycentre of the green points to radially fill the middle hole. Light grey faces: faces added between existing faces. White faces: faces added in the middle to fill the hole.

is initially built to avoid repetition over a neighbouring vertex. Then, it is re-balanced to optimize the continuity of the normals of the faces of each sub-umbrella. That avoids the creation of spikes after the geometrical displacement (see Fig. 12b–c).

For each sub-umbrella obtained, a duplicate vertex is created. These duplicates are moved at a distance that does not exceed a σ orthogonal distance with inconsistent edges adjacent to the original vertex. Moreover, the orthogonal projection of this move vector onto the altitude on each of the adjacent faces does not exceed the face itself. The direction of this move is the average direction of all the regular adjacent edges except the last one in the sequence. If the move vector is null, the sub-umbrella is split into two sub-umbrellas. This separation of the neighbouring faces between the duplicate vertices generates a hole (see Figs 12d and 13). This hole is filled by adding a vertex in the barycentre of the duplicate vertex and faces between successive sub-umbrellas and this next vertex (see Fig. 13b).

The displacement of the duplicate vertices can create new self-intersections, whether by moving the existing faces or by adding new faces (white and light grey faces, Fig. 13). With the intention to avoid these new self-intersections, we distinguish three types of faces. The first is a modified face (grey face, Fig. 14), the second is a face added between two successive duplicate vertices and a pre-existing vertex of the mesh (light grey face, Fig. 14) and the third is face added between two successive duplicate vertices and the added vertex. By this distinction, we obtain three kinds of tetrahedra with one, two and three moves, respectively. With these constructions, when a collision is detected with one of these tetrahedra, this collision is avoided by reducing the amplitude of the move(s) such as that presented in Fig. 14. Finally, if one of the displacement fails, that is, the amplitude of displacement $\approx$0, each sub-umbrella is subdivided into two if possible; otherwise another umbrella is treated first.

4 Results

The proposed method was tested on standard computer graphics benchmark models as well as over 35 000 meshes chosen from the Princeton data set (Shilane, Min, Kazhdan, & Funkhouser 2004), the Thingi10K data set (Zhou and Jacobson 2016), the ShapeNet (Chang et al. 2015) data set, as well as a proprietary data set collected by the authors. The test set covers broadly different types of meshes, ranging from synthetic meshes, such as *Klein bottle*, *Möbius strip*, *Elephant head*, *Bikini*, *Beast*, and *Rhino*, to preprocessed 3D scans, such as the *Happy Buddha* model from the Stanford 3D Scan Repository, as well as noisy real-world data, including *Mandible & Maxilla* mesh obtained from a magnetic resonance (MR) image, *Heart* mesh obtained from a computed tomography (CT) data, *Nose Skin & Cartilage* model, and dental implant models, *Implants A & B* (A: 3D laser scan reconstruction and B: Marching Cubes reconstruction from CT scan), collected by the authors.

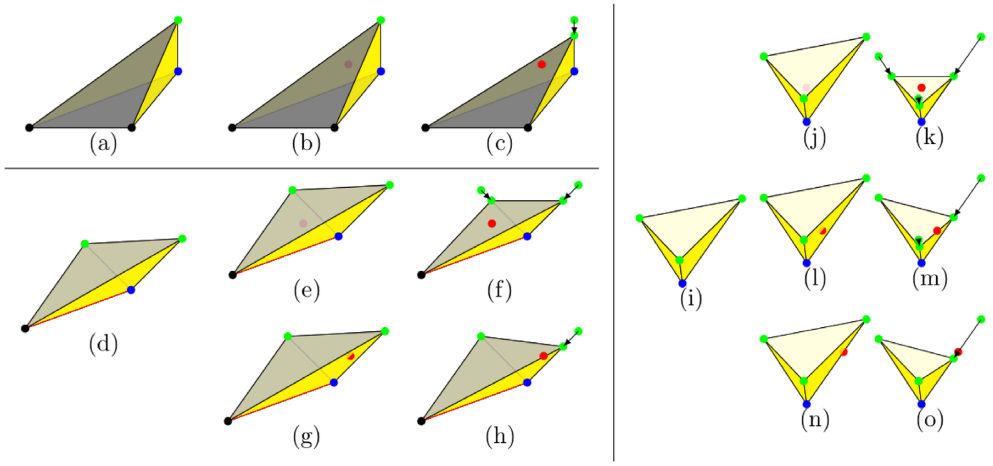


Figure 14: Corrections of detected collisions during the geometrical separation of inconsistent vertices. Three types of faces are distinguished: the moved faces (grey), the faces added between two successive sub-umbrellas and their common vertex (light grey), and the faces added between two successive sub-umbrellas and the newly added barycentre vertex (white), which are shown in tabular form. The first column shows the initial configuration; in the middle, a collision case is encountered; and in the third, the correction is carried out. Blue point: original vertex. Green points: duplicate vertex after displacement. Red point: collision point or vertex of colliding face. Black point: neighbour of the targeted vertex.

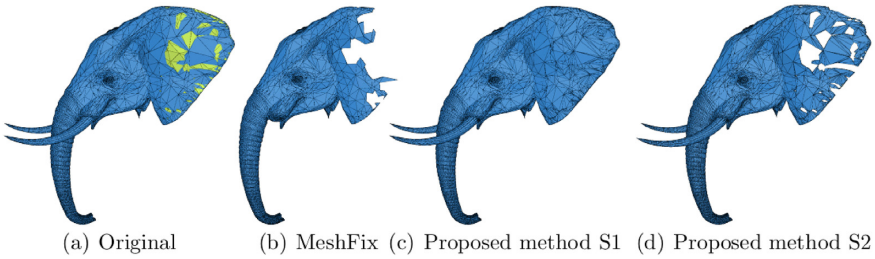


Figure 15: Comparison of MeshFix and the proposed method on the *Elephant head* model. S1: Solution with patch reversing. S2: solution without patch reversing and after removing reversed shells.

Upon such data set, the proposed method was compared against two state-of-the-art methods, namely Geomagic studio® 3D Systems (1997–2016) and MeshFix (Attene 2010–2016, 2010). During these tests, all outputs generated by the proposed method were confirmed to be free of geometrical and topological inconsistencies, while other methods exhibited persisting inconsistencies and/or newly introduced inconsistencies.

4.1 Synthetic data

The *Klein bottle* model (the example in Section 1.2) and the *Möbius strip* are typical surfaces known to have only one side, with the difference that the *Möbius strip* has boundary edges, whereas *Klein bottle* does not. Moreover, the *Klein bottle* contains self-intersections. These two models present certain problems concerning orientation, namely unresolvable *turning edges* (for the *Möbius strip*) and unpairable connections of *dividing lines* without reversing (for the *Klein bottle*). For the *Möbius strip*, *turning edges* would be considered conflict regardless of the score chosen. If the user chose to resolve it by disconnection, the strip would be cut (Fig. 1b). This result is similar to that in MeshFix without hole filling. If the user chose to resolve the conflict by offsetting, the strip would obtain volume (Fig. 1a). For the *Klein bottle*, if the user allowed the reversing of patches, the inner component would be reversed to allow pairing with the other component and would be associated with the connection of the inside of the bottle if the field *Or* was positive (favourable for the pairing *A*). This result can be obtained by volume, winding, or parity-counting approaches. Using MeshFix or Geomagic studio®, the reversed part of the handle and the inner patch was removed and holes were filled.

The *Elephant head* is a mesh model without inconsistent orientation edge but multiple overlaps in the ear area (Fig. 15a). MeshFix removes self-intersecting faces and fills holes, whereas the proposed method provides alternatives. For instance, reversed shells can be separated from the main shell. The user can thus force the preservation of orientations of faces using the score, as presented in Equation (2), $(Or, \#Mg)\{P\,O\}$ (Fig. 15d). Alternatively, faces can be reversed, creating an in/out structure, as the number of shells would be minimized using the score $(Or, \#Mg)\{In/\,Out\}$ (Fig. 15c).

Beast and *Rhino* are two synthetic models with different structures. The *Beast* model is composed of one shell with overlaps, whereas the *Rhino* model is composed of 26 overlapping shells (Fig. 16b). The shells of the *Rhino* model contain a small number of self-intersections (27 intersecting faces). Even though these models have different construction, only the external shell is relevant. Also, the in/out structure is not required if separation is used. Consequently, they can be solved using the score $(Or)\{\}$. Using this simple scoring, the outer hull would be preserved and inner components would be isolated to allow their removal (Fig. 16a and c).

The *Bikini* model is a concrete example where offsetting can be used to resolve conflicting components. When the user

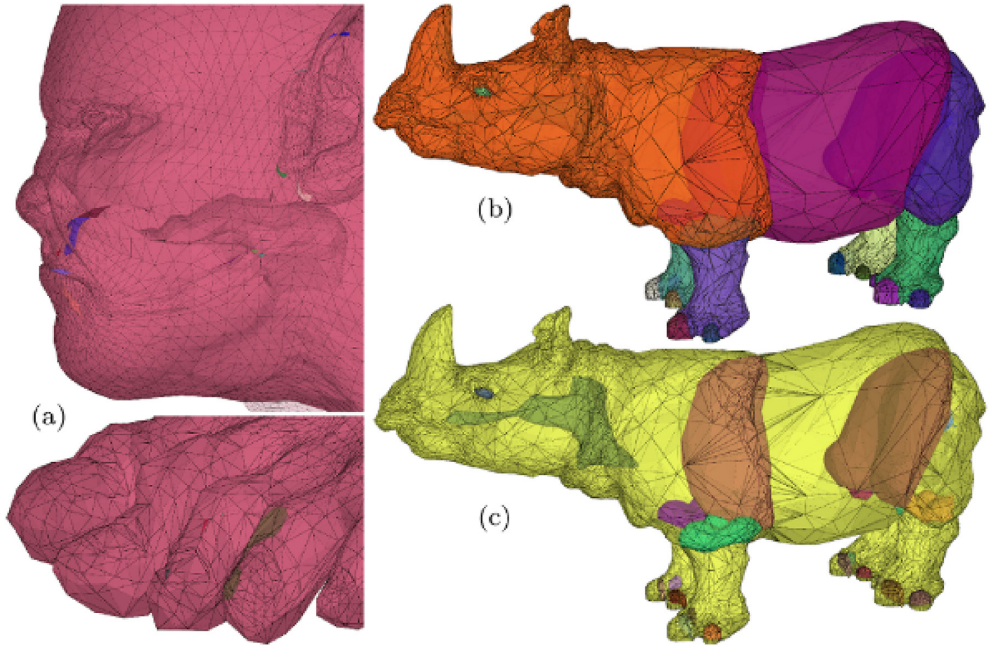


Figure 16: Fixing of *Beast* and *Rhino* models, where shells are rendered in random colours. (a) *Beast* model after repairing. (b) *Rhino* before repairing. (c) *Rhino* after repairing.

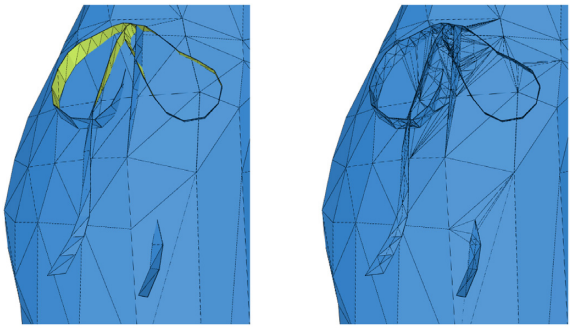


Figure 17: *Bikini* model. (a) Original model and (b) output obtained using the handling of conflicts by offsetting.

requires an in/out structure on this model, all components of the ribbon are conflicting. Choosing offsetting to resolve these conflicts, the patches of the ribbon are duplicated, creating volume from open surfaces. Then, if the field *Or* is positive, the outer components are associated with the body component, whereas the inner components are detached towards the inside (Fig. 17). If the user chose disconnection for resolving conflicts, elements of the ribbon would be dissociated from the body component.

4.2 Acquired data

The *Happy Buddha* model is a classic model obtained from the Stanford 3D scanning repository. It contains 153 self-intersections, 5581 non-manifold edges, but only 2 connected components. The goal is to clean this data beyond the main hull, that is, to remove all wrong facing. To perform this cleaning, the score $(-\#Mg, Or)\{In/Out\}$, with the disconnection scoring rule for conflicts, is most suitable. The constraint *In/Out* will remove dangling faces, and scoring will isolate small components touching the surface of the main shell and offset inconsistencies inside the main shell. For this test, a result similar to that of Geomagic and MeshFix was obtained. However, as can be seen in Fig. 18, the proposed method alters the data less than Geomagic or MeshFix. The maximal error (distance) between matching facing of input and output was 0.001 for Geomagic, 0.0038 for MeshFix, and 0.0002 for the proposed method.

The *Mandible & Maxilla* model is data used in virtual surgical planning, where the outside and the inside of the mesh are required. Common defects that may be found in the data are self-intersections owing to decimation or smoothing processing and non-manifold edges owing to reconstruction. To resolve this, the adopted scoring should be $(\#Mg, Or)\{In/Out, NoRev\}$.

The *Heart* model is extracted from volumetric data. The thin part of the septum has a large number of self-intersections and degenerate triangles (Fig. 19). Here, the user certainly should discard these components, which are in fact noise. To accomplish that, the score $(-\#Mg, -Or)\{In/Out\}$ should be chosen, which would fragment small particles so that they can be easier to remove.

The *Nose* model is an open surface resulting from the combination of two different acquisitions. This model is non-manifold, highly perturbed, and contains several stretched triangles at the base of the nose and multiple self-intersections. Resolving these defects using MeshFix requires filling holes, which would result in loss of the base of the nose (Fig. 1c). Geomagic and the proposed method yielded similar results close to the input, with a maximum difference of 0.3 for Geomagic and 0.05 for the proposed method.

Implant A is reconstructed from 3D laser scan data. It presents similar problems to those of the *Elephant head*. It exhibits surface overlapping in sharp edges. This model can be repaired as the *Elephant head* above. *Implant B* was obtained from the CT scan and contains a small number of self-intersections but several non-manifold edges (Fig. 20b). It contains 1197 edges with 3 adjacent faces, 1445 edges with 4 adjacent faces, 10 edges with

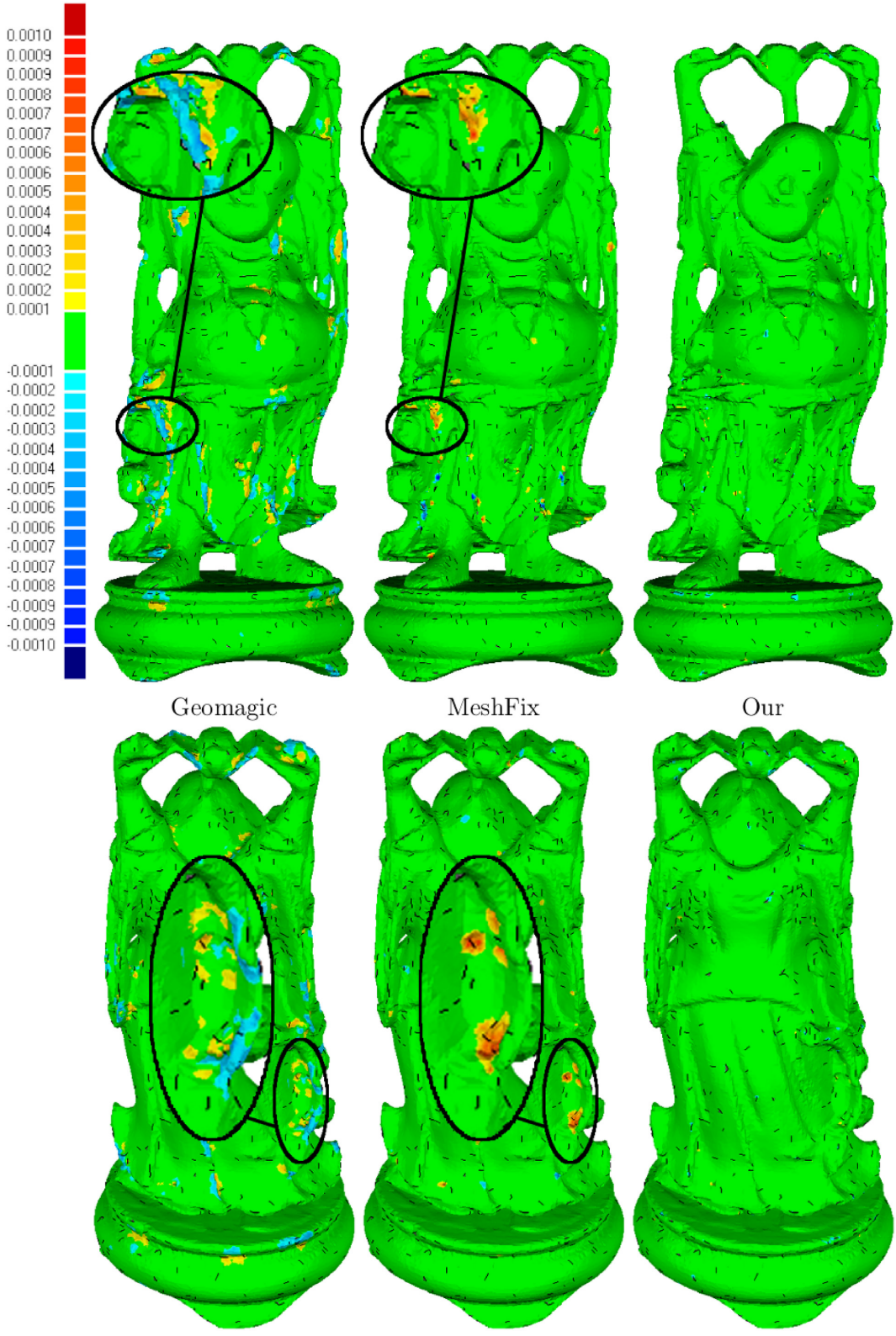


Figure 18: *Happy Buddha* model: comparison of the distance between the outputs of Geomagic, MeshFix, and the proposed method. The comparison was performed using Geomagic studio, where black segments are bordering edges of the original model.

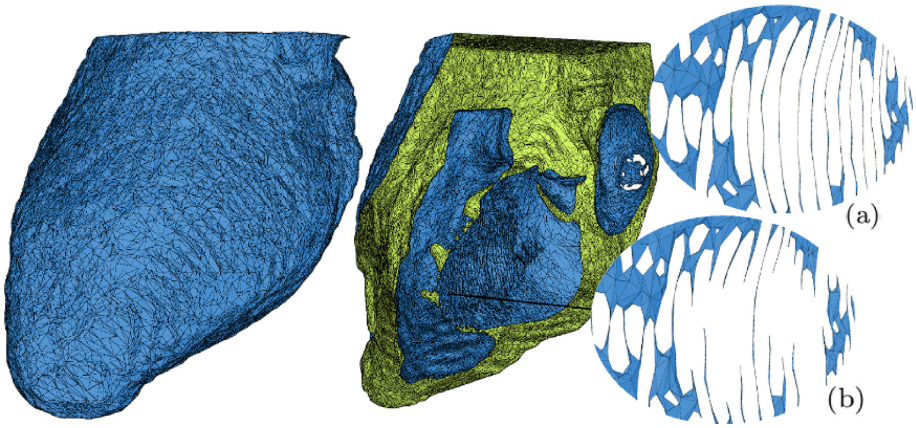


Figure 19: *Heart* model: complete original model (left) and half-cut model showing the thin web of the chopped septum (right). (a) Original data. (b) After correction and removal of small particles.

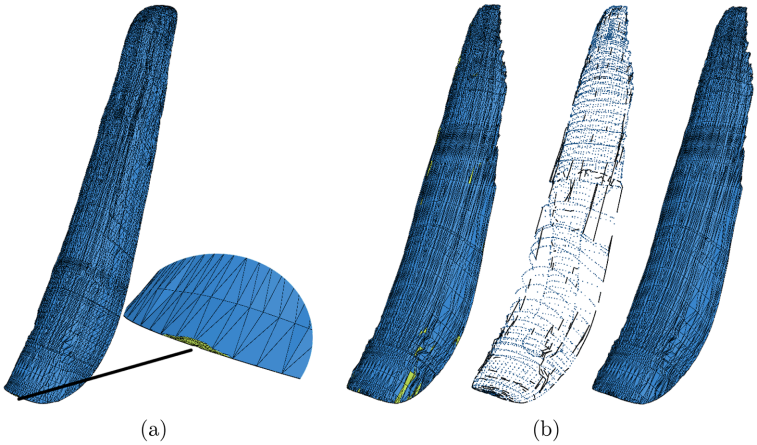


Figure 20: Nasal implants: two examples of acquisition by 3D optical scanning and CT scanning. (a) Implant A: original mesh with zoom in one of reversing surfaces. (b) Implant B: from left to right—the mesh rendered with edges, point cloud in blue with non-manifold edges in black, the output of the proposed method without constraint and a scoring $(\#R, Or, \#Mg)$.

5 adjacent faces, and 14 edges with 6 adjacent faces. The proposed algorithm handled these problems by disconnecting all small connections in odd or not pairable. It created several small shells but preserved the main component as much as possible.

The benchmark tests have been processed with Thingi10K, Princeton, and ShapeNet data sets using MeshFix and our method. Our method preserves more the original data with around 85% of the data altered to almost 0% and the data alteration does not exceed 20% for the worst cases (see Fig. 21c) when preserving or reducing the number of the faces (see Fig. 21a). For its part, MeshFix erases the whole data from over 87% of the input models (see Fig. 21b and d).

4.3 Discussion

The advantages of the proposed method are that it preserves as much as possible the original data instead of rebuilding it. The method is efficient, and the processing time of the resolution of the pairing and the separation of inconsistencies takes only 20% of the global processing time. That processing time is minor compared with the 80% used by remeshing. In addition, it is, on average, linearly proportional to the number of input faces. By minimizing the addition of faces during the separation step, the quantity of faces is significantly reduced by the cleaning step and increased by the remeshing. At the end of the process, we observed a slight reduction in the number of faces. This reduction of faces is mainly due to removing duplicate and degenerate faces. The method enables the user to choose the scoring rule of scoring. The latter is both an advantage and a disadvantage. Obviously, flexibility in choosing the most suitable scoring rule facilitates the process. However, it requires user expertise. Regarding the examples considered, two main strategies can be distinguished. The first is to match noisy acquisitions. It requires an in/out structure and isolate small components, as in the *Happy Buddha* model. The second scoring rule is to match synthetic models constructed by boundary representation using the score $(\#R, Or, \#Mg)\{\}$. It results in an outer hull of the input model preserving the inner shells. In the examples, the Or field is given priority over the $\#Mg$ field. This is true for most models. However, in certain cases, such as lung with bronchus meshes, where there should be only one inner component, the $\#Mg$ field should precede the Or field. Otherwise, for complex models, a semi-automatic scoring rule would be more suitable.

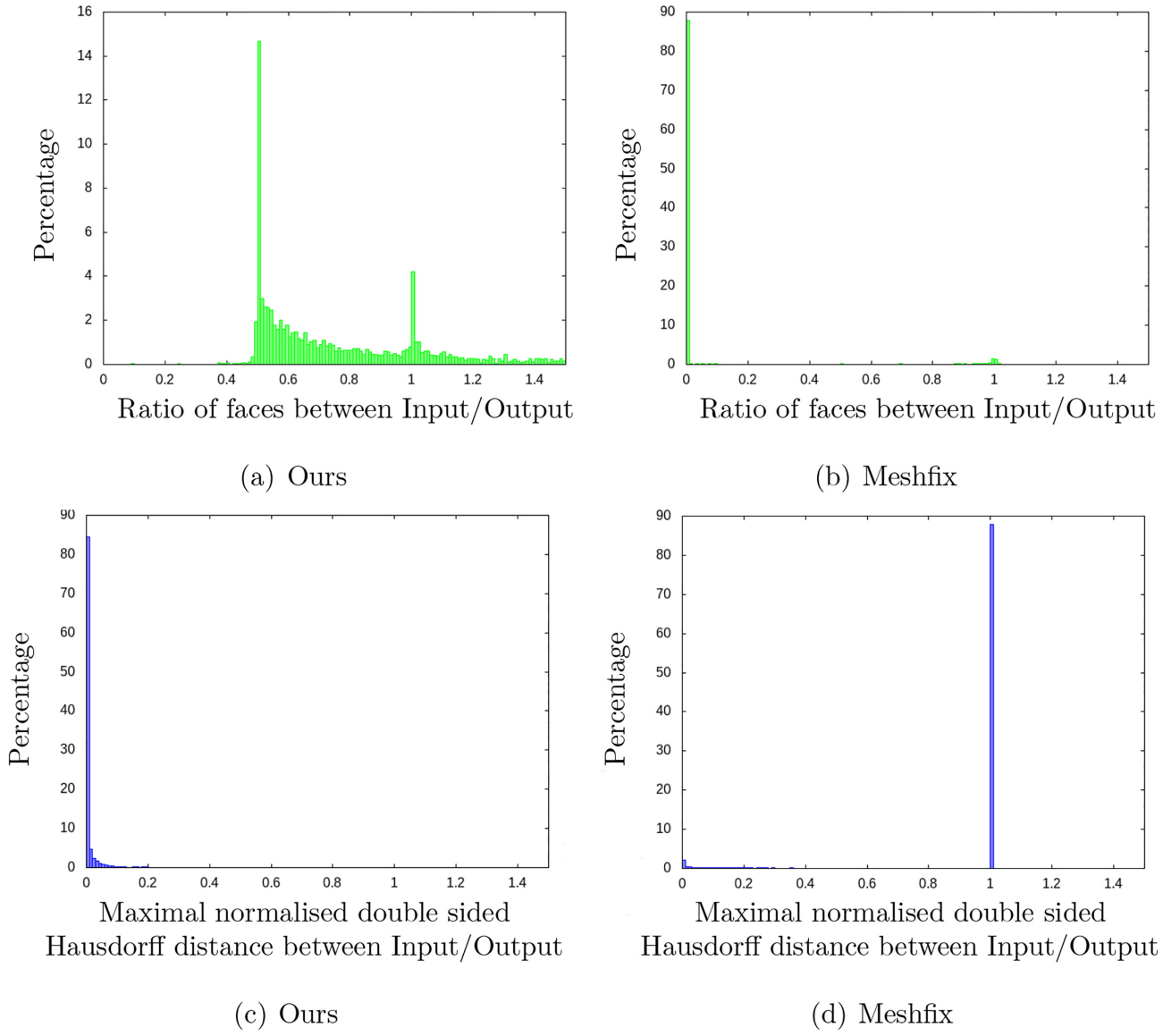


Figure 21: Ratio of faces and maximal normalized double-sided Hausdorff distance between input/output for random sampling of 10 000 models from Thingi10K, Princeton, and ShapeNet data sets. Our method uses a scoring $(Or, \#R, \#Mg)\{In/Out\}$, the disconnection for solving conflicts, and $\sigma = 0.001$. The double-sided Hausdorff is normalized using the radius of the bounding box of the input data. When the output data are empty, we consider the normalized double-sided Hausdorff distance as 1.

5 Conclusion

A method for solving geometrical and topological inconsistencies using graph representations and a scoring system was proposed. The scoring was established to fit with the scoring rule expected by the user in order that a 2-manifold surface may be generated with a minimum of modifications. The capability of the proposed method to resolve inconsistencies was demonstrated on various models composed of synthetic and acquired meshes. Compared with other existing methods/software, e.g. MeshFix and Geomagic, the proposed method proved to preserve the geometry of the input data while handling odd topological inconsistencies.

Furthermore, the proposed method depends on scoring rules that should be defined by the user according to the expected result. However, this score is not intuitive and requires experience. To overcome that, future work would be focused on the simplification of this scoring using an auto-parametrization by analysing the input data.

Acknowledgements

The authors thank Marco Attene for sharing the mesh repair open source library MeshFix. They also thank Alec Jacobson, Kavan Ladislav, and Olga Sorkine-Hornung for sharing their synthetic mesh models and Mathieu Laurentjoye for the *Mandible* & *Maxilla* model. This work was supported by the Korea Research Fellowship Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT, South Korea (Grant No. 2015H1D3A1065744, PM: Dr. Youngjun Kim). This research was supported in part by the Korea Institute of Science and Technology (KIST) institutional program (Grant No. 2E30342, PM: Dr. Youngjun Kim).

Conflict of Interest Statement

The authors declare that they have no conflict of interest.

REFERENCES

3D Systems(1997–2016). Geomagic studio. http://www.geomagic.com, last visit 16 June 2020.

Aguerre, C., Charton, J., Desbarats, P., & Recur, B. (2013). SmithDR (scientific multi imaging tool handled by a DAG layeR). http://www.lab327.net/projects/SmithDR/, last visit 16 June 2020.

Andújar, C., Brunet, P., & Ayala, D. (2002). Topology-reducing surface simplification using a discrete solid representation. *ACM Transactions on Graphics*, *21*(2), 88–105.

Attene, M. (2010). A lightweight approach to repairing digitized polygon meshes. *The Visual Computer*, *26*(11), 1393–1406.

Attene, M. (2010–2016). Meshfix. https://sourceforge.net/projects/meshfix/, last visit 16 June 2020.

Attene, M. (2014). Direct repair of self-intersecting meshes. *Graphical Models*, *76*(6), 658–668.

Attene, M., Campen, M., & Kobbelt, L. (2012). Meshrepair. http://meshrepair.org/, last visit 16 June 2020.

Attene, M., Campen, M., & Kobbelt, L. (2013). Polygon mesh repairing: An application perspective. *ACM Computing Surveys*, *45*(2), 15:1–15:33.

Attene, M., Giorgi, D., Ferri, M., & Falcidieno, B. (2009). On converting sets of tetrahedra to combinatorial and PL manifolds. *Computer Aided Geometric Design*, *26*(8), 850–864.

Barki, H., Guennebaud, G., & Foufou, S. (2015). Exact, robust, and efficient regularized booleans on general 3D meshes. *Computers & Mathematics With Applications*, *70*(6), 1235–1254.

Bischoff, S., & Kobbelt, L. (2005). Structure preserving cad model repair. *Computer Graphics Forum*, *24*(3), 527–536.

Bischoff, S., Pavic, D., & Kobbelt, L. (2005). Automatic restoration of polygon models. *ACM Transactions on Graphics*, *24*(4), 1332–1352.

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

Centin, M., & Signoroni, A. (2018). Advancing mesh completion for digital modeling and manufacturing. *Computer Aided Geometric Design*, *62*, 73–90.

Chang, A. X., Funkhouser, T. A., Guibas, L. J., Hanrahan, P., Huang, Q.-X., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., & Yu, F. (2015). ShapeNet: An information-rich 3D model repository. *CoRR*, **abs/1512.03012**.

Charton, J., Kim, L., & Kim, Y. (2017). Boolean operations by a robust, exact, and simple method between two colliding shells. *Journal of Advanced Mechanical Design, Systems, and Manufacturing*, *11*(4), JAMDSM0041.

Chen, Y., Wang, H., Rosen, D. W., & Rossignac, J. (2005). A point-based offsetting method of polygonal meshes. GVU Tech Report GIT-GVU-05.

Cignoni, P., Callieri, M., Corsini, M., Dellepiane , M., Ganovelli, F., & Ranzuglia, G. (2008). MeshLab: an Open-Source Mesh Processing Tool, Eurographics Italian Chapter Conference. 129-136

Ericson, C. (2004). *Real-time collision detection*. Boca Raton, FL: CRC Press, Inc.

Feng, W., Zhang, H., Huang, J., Wang, C., & Bao, H. (2010). Repairing topological inconsistency of mesh sequences. *Computer Animation and Virtual Worlds*, *21*(34), 355–364.

Furukawa, R., Itano, T., Morisaka, A., & Kawasaki, H. (2007). Improved space carving method for merging and interpolating multiple range images using information of light sources of active stereo. In *Proceedings of the Computer Vision – ACCV 2007, 8th Asian Conference on Computer Vision, Tokyo, Japan, November 18-22, 2007, Proceedings, Part II*, pp. 206–216.

Guéziec, A., Taubin, G., Lazarus, F., & Horn, W. (2001). Cutting and stitching: Converting sets of polygons to manifold surfaces. *IEEE Transactions on Visualization and Computer Graphics*, *7*(2), 136–151.

Hétroy, F., Rey, S., Andújar, C., Brunet, P., & Vinacua, A. (2011). Mesh repair with user-friendly topology control. *Computer-Aided Design*, *43*(1), 101–113.

Hornung, A., & Kobbelt, L. (2006). 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*, SGP '06, pp. 41–50, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland.

Imai, Y., Hiraoka, H., & Kawaharada, H. (2014). Quadrilateral mesh fitting that preserves sharp features based on multi-normals for Laplacian energy. *Journal of Computational Design and Engineering*, *1*(2), 88–95.

Immonen, E. (2018). A parametric morphing method for generating structured meshes for marine free surface flow applications with plane symmetry. *Journal of Computational Design and Engineering*, *6*(3), 348–353.

Jacobson, A., Kavan, L., & Sorkine-Hornung, O. (2013). Robust inside–outside segmentation using generalized winding numbers. *ACM Transactions on Graphics*, *32*(4), 33:1–33:12.

Ju, T. (2004). Robust repair of polygonal models. In *Proceedings of the ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pp. 888–895, ACM, New York, NY, USA.

Ju, T., & Udeshi, T. (2006). Intersection-free contouring on an octree grid. In *Proceedings of the 14th Pacific Conference on Computer Graphics and Applications (PG 06)*.

Jung, W., Shin, H., & Choi, B. (2004). Self-intersection removal in triangular mesh offsetting. In *Proceedings of the CAD'04 Conference*.

Kim, S.-J., Lee, D.-Y., & Yang, M.-Y. (2004). Offset triangular mesh using the multiple normal vectors of a vertex. *Computer-Aided Design and Applications*, *1*(1–4), 285–292.

Li, Y., & Kim, J. (2015). Fast and efficient narrow volume reconstruction from scattered data. *Pattern Recognition*, *48*(12), 4057–4069.

Liu, S., & Wang, C. C. (2011). Fast intersection-free offset surface generation from freeform models with triangular meshes. *IEEE Transactions on Automation Science and Engineering*, *8*(2), 347–360.

Lu, J. H.-C., Quadros, W. R., & Shimada, K. (2017). Evaluation of user-guided semi-automatic decomposition tool for hexahedral mesh generation. *Journal of Computational Design and Engineering*, *4*(4), 330–338.

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

Rossignac, J., & Cardoze, D. (1999). Matchmaker: Manifold BReps for non-manifold r-sets. In *Proceedings of the Fifth ACM Symposium on Solid Modeling and Applications*, SMA '99, pp. 31–41, ACM, New York, NY, USA.

Sagawa, R., & Ikeuchi, K. (2008). Hole filling of a 3D model by flipping signs of a signed distance field in adaptive resolution. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *30*(4), 686–699.

Schertler, N., Savchynskyy, B., & Gumhold, S. (2016). Towards globally optimal normal orientations for large point clouds. *Computer Graphics Forum*. *36*(1), 197–208.

Shewchuk, J. (2008). General-dimensional constrained delaunay and constrained regular triangulations, I: Combinatorial properties. *Discrete & Computational Geometry*, *39*(1–3), 580–637.

Shilane, P., Min, P., Kazhdan, M., & Funkhouser, T. (2004). The Princeton shape benchmark. In *Proceedings of the Shape Modeling International*, pp. 167–178.

Spillmann, J., Wagner, M., & Teschner, M. (2006). Robust tetrahedral meshing of triangle soups. In *Proceedings of the Vision, Modeling, Visualization VMV'06*, pp. 9–16.

Voutchkov, I., Keane, A., Shahpar, S., & Bates, R. (2017). (Re-)meshing using interpolative mapping and control point optimization. *Journal of Computational Design and Engineering*, 5(3), 305–318.

Yamakawa, S., & Shimada, K. (2009). Removing self-intersections of a triangular mesh by edge swapping, edge hammering, and face lifting. In *Proceedings of the 18th International Meshing Roundtable*, pp. 13–29, ed. B. Clark , Springer Berlin Heidelberg.

Zaharescu, A., Boyer, E., & Horaud, R. (2007). TransforMesh: A topology-adaptive mesh-based approach to surface evolution. In *Proceedings of the ACCV 2007 8th Asian Conference on Computer Vision*, Vol. *4844*, pp. 166–175, Springer-Verlag, Tokyo, Japan.

Zhou, Q., Grinspun, E., Zorin, D., & Jacobson, A. (2016). Mesh arrangements for solid geometry. *ACM Transactions on Graphics (TOG)*, *35*(4), 1–15.

Zhou, Q., & Jacobson, A. (2016). Thingi10k: A dataset of 10,000 3D-printing models. *CoRR*, **abs/1605.04797**.