Multi-Material Mesh-Based Surface Tracking with Implicit Topology Changes

PETER HEISS-SYNAK*, Institute of Science and Technology Austria (ISTA), Austria
ALEKSEI KALINOV*, Institute of Science and Technology Austria (ISTA), Austria
MALINA STRUGARU, Institute of Science and Technology Austria (ISTA), Austria
ARIAN ETEMADI, Institute of Science and Technology Austria (ISTA), Austria
HUIDONG YANG, University of Vienna, Austria
CHRIS WOJTAN, Institute of Science and Technology Austria (ISTA), Austria

We introduce a multi-material non-manifold mesh-based surface tracking algorithm that converts self-intersections into topological changes. Our algorithm generalizes prior work on manifold surface tracking with topological changes: it preserves surface features like mesh-based methods, and it robustly handles topological changes like level set methods. Our method also offers improved efficiency and robustness over the state of the art. We demonstrate the effectiveness of the approach on a range of examples, including complex soap film simulations with thousands of interacting bubbles, and boolean unions of non-manifold meshes consisting of millions of triangles.

CCS Concepts: • **Computing methodologies** → **Physical simulation**; *Shape modeling*; • **Applied computing** → *Physical sciences and engineering*.

Additional Key Words and Phrases: surface tracking, topology change, non-manifold meshes, multi-material flows, solid modeling

ACM Reference Format:
Peter Heiss-Synak, Aleksei Kalinov, Malina Strugaru, Arian Etemadi, Huidong Yang, and Chris Wojtan. 2024. Multi-Material Mesh-Based Surface Tracking with Implicit Topology Changes. *ACM Trans. Graph.* 43, 4, Article 54 (July 2024), 14 pages. https://doi.org/10.1145/3658223

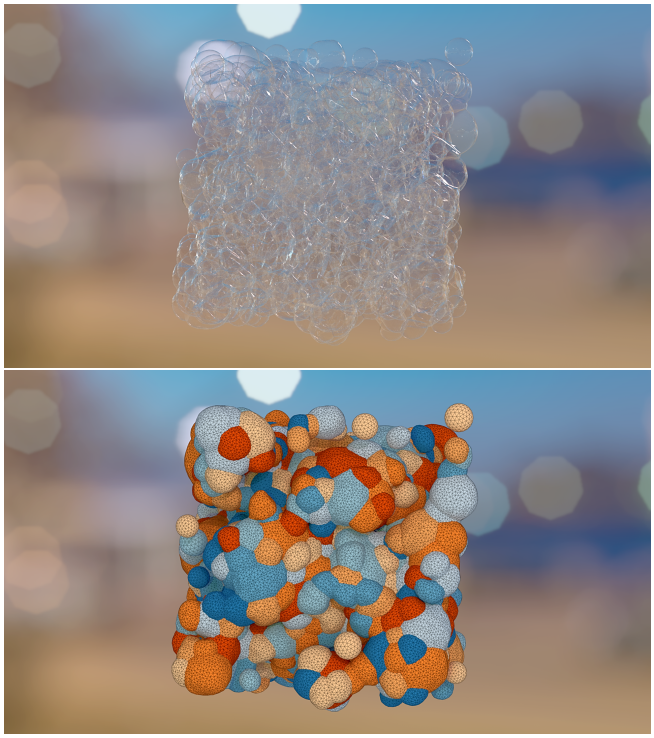


Fig. 1. Our mesh-based surface tracker efficiently computes topological changes at huge scales, like this soap film simulation of 1000 bubbles.

1 Introduction

Non-manifold surfaces and volumes made of multiple materials are commonplace in the fields of biology (multi-cellular organization), material science (foams), digital fabrication (multi-material 3D printing), and physics animation (multi-phase fluids). This paper focuses on the problem of non-manifold surface tracking with *topological changes*, with demonstrated applications in physics simulation and the solid modeling of shapes consisting of multiple materials. Existing methods for solving this problem either rely exclusively on implicit surfaces [Losasso et al. 2006], or on combinations of collision resolution and mesh surgery [Da et al. 2014]. Implicit methods robustly handle topological changes, but they degrade geometric features over time with constant re-sampling. The existing mesh-based method preserves features but has difficulty with scaling to large problem sizes robustly and efficiently. We present a hybrid algorithm which combines the benefits of both approaches: simultaneously preserving geometric features while quickly and robustly producing output meshes even in challenging cases.

Our method is inspired by work on manifold topology changes [Wojtan et al. 2009]: it preserves surface details by representing geometry as an explicit mesh, and it computes topological changes using a local implicit surface. This approach enjoys increased reliability compared to explicit mesh surgery, because it only requires robust geometric operations on the boundary of a small well-defined and

*Both authors contributed equally, and are listed in alphabetical order

Authors' Contact Information: Peter Heiss-Synak, Institute of Science and Technology Austria (ISTA), Austria, psynak@ista.ac.at; Aleksei Kalinov, Institute of Science and Technology Austria (ISTA), Austria, akalinov@ista.ac.at; Malina Strugaru, Institute of Science and Technology Austria (ISTA), Austria, istrugar@ist.ac.at; Arian Etemadi, Institute of Science and Technology Austria (ISTA), Austria, aetemadi@ista.ac.at; Huidong Yang, University of Vienna, Austria, huidong.yang@univie.ac.at; Chris Wojtan, Institute of Science and Technology Austria (ISTA), Austria, wojtan@ist.ac.at.

ACM 1557-7368/2024/7-ART54
https://doi.org/10.1145/3658223

predictable region (instead of requiring robustness for all possible mesh self-intersections). We offer the following contributions:

- The generalization of hybrid mesh-grid topology changes [Wojtan et al. 2009] to non-manifold surface meshes.
- A novel algorithm for constructing an intersection-free implicit surface from a region of self-intersecting meshes.
- Robustness improvements over prior work: our algorithm behaves reliably even with degenerate inputs, and it is guaranteed to terminate in finite time. We also introduce a statistical robustness benchmark for surface tracking algorithms.
- Our algorithm's optimized data structures and increased stability translate into efficient performance on large problems.

2 Related work

2.1 Surface Tracking

Implicit methods like the level set method [Osher and Fedkiw 2001] evolve an implicit surface, usually on a background grid, and handle topological changes automatically when the feature size drops below the grid resolution. The idea has been extended to represent multiple materials for the purposes of simulating liquids [Kim 2010; Losasso et al. 2006] and clusters of bubbles [Kim et al. 2007; Zheng et al. 2009]. Level set methods can also model the topology-changing evolution of co-dimensional structures like curves [Burchard et al. 2001] and vortex filaments [Ishida et al. 2022]. Semi-Lagrangian Contouring methods [Bargteil et al. 2006; Li et al. 2016] represent surfaces as a signed distance field (SDF) and reconstruct an explicit mesh at each step to improve the accuracy of SDF advection. While handling topological changes with ease, all these methods frequently re-sample the interface and cause it to degrade over time.

Explicit triangle meshes are a good alternative, because they have specific control over the re-sampling of the interface. Mesh deformation tools like *The Surface Evolver* [Brakke 1992] and *Front Tracking* methods [Glimm et al. 2000; She et al. 2016] are especially useful for preserving detailed surface features, but handling self-intersections and topological changes is comparatively more challenging. Brochu and Bridson [2009] explicitly resolve mesh collisions and impose topological changes with local mesh surgery. Others [Misztal and Bærentzen 2012; Pons and Boissonnat 2007] evolve entire volumetric meshes in order to handle changes at the surface. A hybrid approach [Du et al. 2006; Müller 2009; Wojtan et al. 2009; Yang et al. 2019] combines manifold explicit surface meshes with implicit level-set-style topology changes. These ideas were extended to eliminate the use of a background grid [Bo et al. 2011; Chentanez et al. 2016] and to preserve thin structures [Wojtan et al. 2010].

Despite their utility in evolving manifold surfaces, few mesh-based surface tracking algorithms are able to cope with multiple materials. The *Deformable Simplicial Complex* method [Misztal and Bærentzen 2012] tracks multiple materials at the expense of evolving an entire volume. *Los Topos* [Da et al. 2014] extends mesh tracking with collision-based topological changes to handle multiple materials, and they also provide guarantees that a mesh will always be intersection-free. However, they do not guarantee that the algorithm will terminate, and indeed it fails to terminate for large and complicated problems.

A hybrid approach by [Yang et al. 2019] represents surfaces as a set of closed manifold meshes. The algorithm handles topology changes implicitly by selective conversion of complicated mesh geometry to regional level sets. This method requires both maintenance of triangle meshes *and* accurate advection of regional level sets, leading it to differ from our method in a number of ways. The duplicated mesh and level set data set operations lead to substantially higher memory cost and implementation complexity. The method also requires a global velocity field defined throughout the volume and cannot handle generic mesh displacements as input. This technique also avoids using non-manifold triangle meshes and instead approximates them with manifolds that lie exactly on top of each other, requiring redundant computations and potentially causing numerical errors when surfaces inevitably self-intersect due to numerical noise. It is inapplicable to settings where truly non-manifold meshes play a crucial role, such as surface-tension-dominated scenarios.

As noted by Da et al. [2014], a multi-material extension of explicit mesh tracking with implicit topology changes [Wojtan et al. 2009] is non-trivial; ours is the first to do so for generic non-manifold meshes.

2.2 Non-Manifold Mesh Processing

Outside of the surface-tracking application, several researchers studied the problem of generating [Shimada and Gossard 1995], manipulating [Hubeli and Gross 2000; Ying and Zorin 2001], and repairing [Wagner et al. 2003] non-manifold geometry. Surprisingly, non-manifold discrete Laplacian operators did not emerge until recently, for the simulation of surface tension [Da et al. 2015] and soap film evolution [Ishida et al. 2020, 2017], followed by the development of a non-manifold Laplacian for meshes and point clouds based on intrinsic Delaunay triangulation [Sharp and Crane 2020]. Geometry sculpting tools with topological changes [Bernstein and Wojtan 2013] are also rare, partly because of the difficulty in defining the correct behavior when meshes intersect.

2.3 Embedding and Immersing Meshes in $\mathbb{R}^3$

To decide whether a mesh exhibits topological problems, our work uses the idea of a volume being embedded in $\mathbb{R}^3$. Winding numbers [Barill et al. 2018; Jacobson et al. 2013] can be used to classify inside-out and overlapping surfaces in the two-material (manifold surface) case, though we are unaware of an extension to multiple materials. Other techniques generate an immersion in $\mathbb{R}^3$ from self-intersecting surfaces [Gagniere et al. 2022; Li and Barbič 2018], which is related to our problem of finding and fixing overlapping geometry. Again, we are unaware of any multi-material (non-manifold) surface variants of these works.

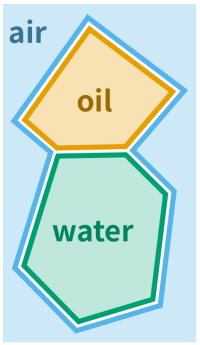
2.4 Mesh Repair, Cut Cells, and Solid Modeling

The "implicit topology change" idea replaces a portion of a mesh by clipping it to a grid, discarding part of the mesh, and replacing it with new triangles. This process shares a great deal of algorithmic overlap with existing techniques for generating cut-cell meshes [Fang et al. 2022; Tao et al. 2019] and repairing flawed meshes [Bischoff and Kobbelt 2005]. Algorithms for robust solid modeling [Sellán et al.

2019; Trettner et al. 2022; Zhou et al. 2016] also solve many of the same problems we do, especially the problem of computing the approximate Boolean intersection and union of meshes [Pavić et al. 2010; Wang 2010]. Recent algorithms even perform solid geometry processing with non-manifold meshes [Diazzi and Attene 2021].

3 Algorithm Overview

Our algorithm takes as input a non-manifold triangle mesh without boundary, that is, a mesh in which every triangle edge is adjacent to at least two triangles. The mesh represents a deformed multi-material configuration with defects such as self-intersections, overlaps, inversions, and their combinations (see Figure 3 for examples). Such defects frequently occur in simulation and correspond to physically impossible configurations.



As an output, our algorithm returns a fixed, non-manifold triangle mesh, that is a union of closed volumes, each associated with a unique material. Additionally, our algorithm performs topological operations that are necessary in many simulation scenarios, such as merging together portions of the mesh that come in very close proximity, forming interfaces between volumes corresponding to different materials, and splitting portions of the mesh apart at very thin junctions. Furthermore, we *only* change the mesh in the regions with defects, and in regions where topological changes occur, leaving the rest of the mesh untouched. This minimal mesh surgery is generally useful for preserving surface features and avoiding information loss over repeated operations.

Our notion of a *material* is based on the idea that when real-world objects made of different materials collide, they form an interface. These materials can be physically distinct, such as air, oil, and water, as illustrated in the inset, or volumes that do not mix, such as individual soap bubbles or biological cells.

Each triangle in the mesh forms a part of an interface between two volumes of different materials. For example, a surface might represent the interface between a solid object and the air around it or between different non-mixing liquids. We therefore assume that each triangle is associated with two different material labels, one for each side, i.e. each normal of the triangle. Furthermore, we assume that triangle labeling is consistent—labels on neighboring triangles whose normals point into the same closed volume are the same. The input surface thus represents a segmentation of space into a number of volumes, whose spatial arrangement might include the aforementioned defects.

Meshes with clean divisions between inside/outside (two-material case) or between multiple different materials, such as in the inset figure, allow us to associate closed volumes directly with materials, and are therefore useful for applications like physics simulation, constructive solid geometry, and 3D printing, but *only if they are properly embedded in space*. If the mesh overlaps itself or twists inside-out, then the materials of volumetric regions inside the mesh are poorly defined, and the applications will fail. Our algorithm thus provides a mesh-fixing tool that can be executed in between simulation steps, in order to guarantee mesh quality.

This problem has already been addressed by two approaches:

- *Explicit* approaches (exemplified by the *Los Topos* algorithm of Da et al. [2014]) operate directly on the triangle mesh and detect improper embeddings when they first appear through collision detection. They remove these self-intersections with collision avoidance and mesh surgery. This custom mesh surgery is difficult to make computationally efficient and robust, but the mesh is flawlessly preserved away from surgery locations.
- *Implicit* approaches (exemplified by regional level set methods like [Zheng et al. 2009]) work directly within the embedded space, classifying which portions of the volume belong to which material and then reconstructing an implicit surface from these volumetric material classifications. Topological changes happen implicitly by construction, because each point in the volume can only belong to a single material. This strategy is efficient and robust, but it degrades the input surface due to constant re-sampling.

Our strategy is to use the implicit approach locally where we know topological changes must occur, and to preserve the explicit surface everywhere else. Our algorithm thus inherits the efficiency and robustness of level set methods without degrading the surface mesh. We use a sparse background grid data structure to compute implicit topology changes, so our method guarantees topological correctness only up to the resolving ability of this background grid (unlike explicit approaches, which guarantee topological correctness on the level of precise mesh intersections). However, we find this decision to be a remarkably useful practical trade-off due to the robustness and efficiency benefits.

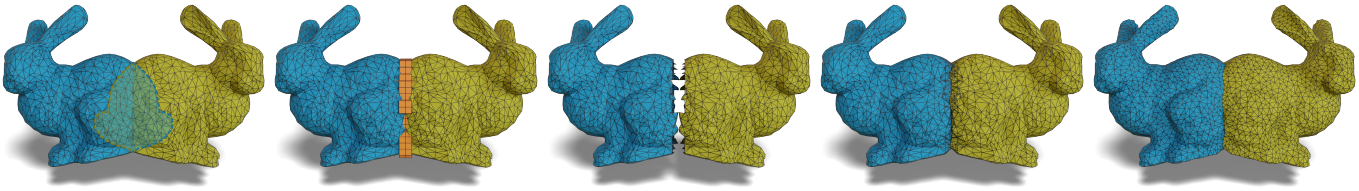
The rest of this paper explains our approach in detail. First, we explain the input non-manifold mesh and background grid data structures in Section 4. Then, we detail our algorithm as follows (also illustrated in Figure 2):

- **Material Assignment** (Section 5) Classify material properties of each vertex on the background grid.
- **Topological Flaw Detection** (Section 6) Using material properties and mesh/grid intersections, decide which regions of the mesh require remeshing.
- **Mesh Cutting** (Section 7) Remove flawed portions of the surface by clipping mesh triangles to grid cell boundaries.
- **Mesh Replacement** (Section 8) Replace removed surfaces with triangles from a topologically clean implicit surface.
- **Mesh Improvement** (Section 8.5) Locally re-mesh the newly changed surface to improve triangle quality.

After explaining the algorithm, we discuss our results and limitations in Sections 9 and 10.

4 Data Structures

As discussed in Section 3, the input mesh represents interfaces between different materials. We assign each material a unique integer label, and each triangle stores exactly two of these material labels.



(a) Overlapping input (b) Flaw detection (c) Mesh cutting (d) Mesh replacement (e) Mesh improvement

Fig. 2. Our algorithm takes an input mesh with topological flaws (e.g. self-intersections, overlaps, and inside-out regions) and outputs a new mesh that is topologically clean based on its sampling on a background grid.

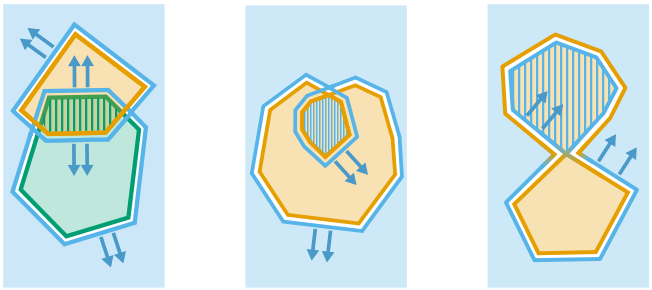


Fig. 3. **Input mesh defects.** Shaded regions highlight typical defects found in the input mesh. Arrows point in the outward normal direction.

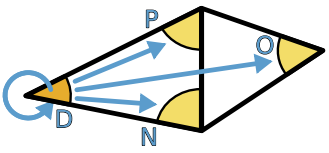
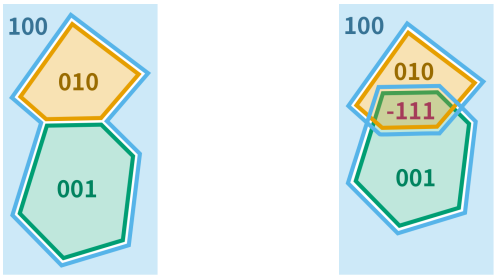


Fig. 4. **Mesh corner connectivity.** Each mesh corner has pointers to four other corners: next (N), previous (P), dual (D), and opposite (O).



(a) Embedded (b) Overlapping

Fig. 5. **Material vector assignment.** Our algorithm introduces *material vectors* to distinguish between properly embedded meshes and meshes with topological problems. Properly embedded regions have one-hot encoded vectors; material vectors in violating regions have negative components.

The first label of the pair represents the material in the triangle's normal direction with respect to the given consistent orientation, and the second label represents the material in the opposite direction. This way a triangle with a material label pair (m_1, m_2) represents an element of the interface between materials m_1 and m_2.

We store the non-manifold triangle mesh using a custom data structure similar to the corner table [Rossignac 2001]: Each triangle has two sides, and each side has three corners, giving us 6 corners per triangle. Each corner has pointers to four other corners in the same mesh. Two of those corners are neighbors within the same side. Another one, a "dual" corner, lies on the other side of the same triangle at the same vertex. The final "opposite" corner is located on the adjacent triangle across the edge opposite the original corner (see Figure 4).

Each vertex also stores pointers to all corners at this vertex in the adjacent triangles. Our data structure allows us to store meshes with non-manifold edges and vertices, and allows us to perform local neighborhood operations (like access, insertion, and deletion of elements) in constant time based purely on mesh connectivity, rather than geometric queries.

We also make use of a background grid for performing topological tests and mesh surgery. The grid is created using a bounding box of the input mesh with a random small perturbation and filled with cubical grid cells with edge lengths equal to a user-provided topological length scale ℓ (our experiments set ℓ equal to the average edge length in the triangle mesh). Because we only perform grid operations in regions that overlap the mesh, we use a sparse grid data structure to only store information for grid cells that intersect the triangle mesh. Specifically, we allocate a grid cell only if it overlaps a mesh triangle's bounding box, producing a narrow band of allocated grid cells around the mesh surface. For a general surface with n mesh triangles of similar sizes, the number of allocated cells is $O(n)$. Each grid cell stores pointers to the overlapped triangles. Each grid vertex stores a sparsely encoded *material vector* (discussed in the next section) and bookkeeping information for local topological tests (Section 6) and for mesh surgery (Sections 7 and 8).

5 Assigning Grid Materials

The first step of our algorithm is to test whether the input triangle mesh (and the volumes enclosed by it) is properly embedded into space. For a correctly embedded mesh, every point in space will belong to exactly one material; it will lie inside of exactly one closed volume of the triangle mesh exactly once, and it will lie outside of all others. This corresponds to a physically valid configuration. However, a mesh that is not properly embedded can exhibit strange features like a region of space belonging to multiple materials, or

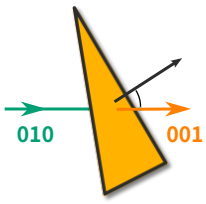


Fig. 6. **Material switch.** A ray travels from left to right and intersects a triangle. At the intersection, a material vector initially equal to $(0\,1\,0)^T$ is updated to $(0\,0\,1)^T$, based on the triangle's material labels and the ray's relative orientation with the triangle normal.

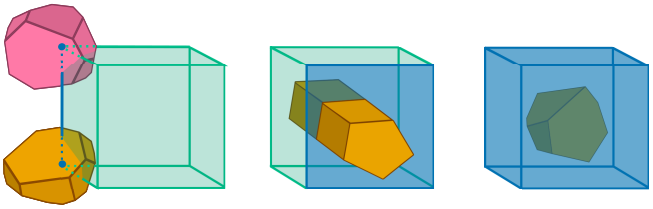


Fig. 7. **Complex grid primitives.** From left to right: a complex edge, a complex face, a complex cell.

a region of space that lies inside the same material multiple times. Figure 3 shows an example of such configurations.

To identify the material for a given point in space, and to allow for all possible bizarre combinations of nested materials at this phase in the algorithm, we introduce a *material vector* V: a sparse vector of length m (for m different materials). If a point in space lies once inside material i and zero times inside other materials, then V should have a 1 in its i^{th} entry and 0 everywhere else. Assume we start drawing a line at a point in space that has material vector V. Each triangle in the mesh represents part of an interface between two different materials. Therefore, when our line intersects a triangle, we adjust V, based on the material labels of the intersected triangle. Let i be the label associated with the triangle normal pointing into the volume our line is leaving, and j be the label associated with the triangle normal pointing into the volume our line is entering. We adjust V by subtracting 1 from the i^{th} entry and adding 1 to the j^{th} entry (see Figure 6).

We use the idea that regions correctly embedded in space correspond to material vectors in a one-hot configuration—we call such material vectors **physical**. Note that a physical material vector directly corresponds to a single material label. Similarly, erroneously embedded regions correspond to material vectors with values greater than 1 or less than 0 (but all entries will have integer values, and all entries sum to 1)—we call such material vectors **non-physical**. See Figure 5b for an example of such a configuration.

In order to find these erroneously embedded regions of space, we assign material labels to each vertex in our sparse grid data structure. We start far outside the mesh with a physical material vector indicating the "outside" material, *e.g.*, with a value of 1 in the first entry, and zeros everywhere else. We then cast axis-aligned rays along grid edges, updating V with each triangle intersection in order. The rays are cast in three principal directions, thus covering all grid edges, and allowing us to additionally compute and store all intersection points between grid edges and mesh triangles, which we will use in the subsequent steps.

We reduce the amount of necessary computation by acknowledging that material vectors cannot change on grid edges with no triangle intersections. It is therefore sufficient to compute material vectors only on edges that are adjacent to allocated grid cells, fitting with our sparse data structure. Additionally, we only store a material vector on a grid vertex if the previous grid vertex along a ray has a different material vector. If we then query a grid vertex for its material vector, it either has a material vector stored, or it returns the material vector of the nearest earlier grid vertex along a ray. We store material vectors sparsely by only saving the non-zero entries.

As mentioned in Section 4, we allocate a narrow band of grid cells around the mesh surface. As long as most grid vertices lie inside a small number of materials, sparse material vectors will use constant memory even in the presence of an overall large number of materials. This shows that our algorithm scales linearly with the number of input triangles. The stored material vector data is sufficient for the implicit surface reconstruction in Section 8.

As the density of the ray sampling is determined by the grid edge length, it is possible that rays will miss improperly embedded thin features of size smaller than the specified resolution ℓ. Although this prevents our algorithm from guaranteeing an output which is properly embedded at *all* scales, we find it a good trade-off for gaining robustness and efficiency in practical scenarios (Section 9).

This algorithm is similar to the strategies for determining inside/outside labels for manifold triangle meshes [Müller 2009; Wojtan et al. 2009, 2010], but extended to the multi-material case. Robustness in this step is crucial for precise topological changes, so we take additional steps to ensure that the triangle-ray intersection operations do not produce nonsense in the case of degenerate configurations (like rays exactly hitting triangle edges or vertices). First, we follow Müller [2009] and trace rays along all three axes, marking any inconsistent findings as topological flaws (Section 6). Most importantly, we employ symbolic perturbations (Simulation of Simplicity [Edelsbrunner and Mücke 1990]) to consistently resolve degenerate orientation tests, reducing the need for special-case code and enhancing robustness.

6 Detecting Topological Flaws

Our next step is to locate regions of space where the mesh needs to be re-sampled. It is self-explanatory that we should eliminate non-physical regions such as overlaps and inversions, and it is also important in many simulation applications to merge surfaces together when they come within a certain minimum distance, and to split them, when they become very thin. Thus, unlike algorithms dedicated to resolving mesh CSG operations, surface tracking algorithms [Brochu and Bridson 2009; Da et al. 2014; Du et al. 2006; Losasso et al. 2006; Wojtan et al. 2009] tend to force topological changes when surfaces come within a user-defined topological resolution ℓ. At the same time, we employ techniques to avoid unnecessary resampling whenever possible, in order to maintain as much of the input surface as we can.

In the rest of this section we describe how we partition all grid cells into two groups: **flawed cells**, in which we will remove all of the existing mesh and replace it with a new mesh; and **intact cells**, in which we will preserve the input mesh. We call the collection of flawed cells the **flawed region**, the collection of intact cells the **intact region**, and the collection of grid faces separating the flawed region from the intact region the **flawed boundary**.

6.1 Non-Physical Cells

First we find all grid cells with defects such as overlaps and inversions. We recall that such defects correspond to non-physical material vectors. As such, for each grid vertex with a non-physical material vector, we mark its eight adjacent cells as **non-physical**. Additionally, we investigate how the material vector changes as we traverse along grid edges. If the material vector becomes non-physical after crossing an intersection on a grid edge, we mark the four adjacent grid cells as non-physical. We skip grid edges with no intersections, since the material vector cannot change along them. In order to obtain a mesh with well-defined materials everywhere, we must remesh all non-physical cells, and so we add them into the flawed region.

6.2 Complex Cells

In this step, we determine all grid cells in which we will merge and split surfaces. We generalize the manifold/two-material ideas of Wojtan et al. [2009] and seek out cells where the topology of the *non-manifold* surface mesh disagrees with the topology implied by a *multi*-material implicit surface sampled on the background grid. We do this by examining the intersections of grid primitives with the triangle mesh. We call grid primitives that indicate a disagreement between the implicit and explicit surfaces topologically **complex**; we list the cases in order of increasing dimension, with illustrations shown in Figure 7.

- A **complex edge** is a grid edge that intersects more than one mesh triangle.
- A **complex face** is a grid face whose intersection with the mesh forms a cycle, or a face adjacent to a complex edge.
- A **complex cell** is a grid cell whose intersection with the mesh contains a closed volume, or a grid cell that is adjacent to a complex face.

6.3 Deep Cells

As explained in [Wojtan et al. 2009], remeshing all complex cells would lead to an overzealous re-sampling of the surface. For example, the tiniest convex surface bump can intersect a grid edge twice (Figure 9). The "Deep Cell" test, employed by [Wojtan et al. 2009], avoids remeshing small surface details, but it only works for two materials. It uses the idea that a surface should only be remeshed in areas where the explicit and the implicit surfaces differ *significantly*.

We generalize the deep cell test to the multi-material setting. Our goal is to allow complex features to persist if they have the same material labels as nearby grid vertices. To achieve this, we inspect all complex edges that are not adjacent to non-physical cells. For each such edge, we check if all the material labels found along the edge can be found on grid vertices that have physical material vectors and lie within Manhattan distance 2ℓ of the edge. If true, it means that the explicit surface (represented by grid edge-mesh triangle intersections) and the implicit surface (represented by labels on grid vertices) are similar to each other in the neighborhood of the complex edge and we call the edge **shallow**. Complex cells for which all adjacent complex edges are shallow contain a mesh that likely represents surface features, therefore we opt to avoid remeshing them. Conversely, if a material label on a complex edge *cannot* be found on a nearby grid vertex as described above, it means the implicit and explicit surfaces differ significantly in the neighborhood of the edge. We call such an edge **deep**, as it is located deeply inside a volume, away from material interfaces. We refer to a complex cell with at least one deep edge as a **deep cell**, and we add all deep cells to the flawed region for remeshing. Our strategy allows small surface textures to persist while preventing the emergence of long thin structures.

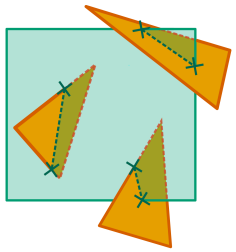


Fig. 8. In a geometrically non-degenerate configuration, a triangle intersects a grid face in one of the three possible ways. In each case, the number of triangle edge-grid face intersections is equal to two. If we find any deviation, indicating a numerical instability or a geometrically degenerate configuration, we mark the grid face as complex.

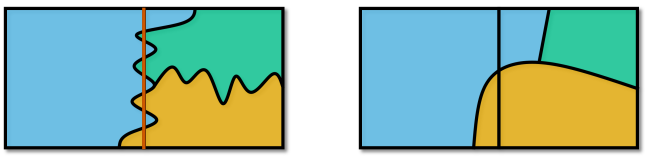
6.4 Flawed Region

So far we added non-physical and deep cells to the flawed region. Next, we add grid cells in which numerical inconsistencies are detected in the grid *vertices* and grid *edges*. Flawed vertices occur when the ray casting along the three different coordinate axes produce different material vectors (Section 5), and flawed edges occur during geometric intersection with triangles (see Figure 8). These events are rare and lead to very little additional re-meshing in practice.

For the mesh reconstruction step, it is important that there are no complex edges or faces in the flawed boundary. We iterate over all grid edges and faces in the flawed boundary, and if any of them is complex, we add its adjacent cells to the flawed region and repeat until the flawed boundary is free of complex edges and faces.

7 Local Mesh Cutting

Next, we surgically cut away the mesh in the regions with topological flaws by deleting the geometric intersection of the triangle mesh and the flawed region. Prior authors [Bischoff and Kobbelt 2005; Du et al. 2006; Wojtan et al. 2009, 2010] do this by first subdividing the mesh where it intersects the grid edges, then subdividing again where mesh edges intersect grid faces. Unfortunately, this sequential subdivision strategy can create zero-area triangles and degrade overall robustness (Figure 11), so we follow Pavić et al. [2010] by



(a) (b)

Fig. 9. If we are not careful, high frequency details on the surface are (a) detected as complex edges and (b) subsequently resampled.

computing a constrained Delaunay triangulation for each triangle, using Triangle [Shewchuk 1996] to force the resulting mesh to have edges wherever a triangle intersects a grid face. We also add additional checks to guarantee numerical robustness as described in Appendix A.

After this step, the mesh triangles are separated into two groups: those that lie completely inside the intact region, and those that lie completely inside the flawed region. We delete all triangles inside the flawed region in preparation for the next step.

8 Local Mesh Replacement

Our next step is to fill the now-empty flawed region with a clean mesh extracted from the background grid. However, we know that the non-physical vertices (Section 6.1) will have non-physical material vectors, and the geometry of the discarded mesh in the flawed region is arbitrary and untrustworthy. We create new surface geometry using the following strategy:

(1) Determine a material label for each grid vertex.
(2) Generate new triangles based on multi-material marching cubes and connect them to the original mesh.
(3) Optimize the positions of the new mesh vertices.
(4) Transfer properties to the new mesh.
(5) Improve mesh quality.

Note that we first determine the *topology* of the new surface in steps 1 and 2, and then pin down its *geometry* in step 3. We also note how this strategy differs from the two-material approach of Wojtan et al. [2009], which determines grid labels by projecting the winding number (a process that only works for two-materials), and determines mesh vertex locations based on distances to the original (untrustworthy and discarded) triangle mesh.

8.1 Correcting Material Vectors

In this step we want to assign unique material labels to all grid vertices. *Physical* material vectors map one-to-one to unique material labels, so we only have to focus on grid vertices with *non-physical* material vectors here. In order to estimate a suitable material label for each of these vertices, we look for the nearest point on the grid where the material vector is physical. We reuse grid-mesh intersections from Section 5 to identify the exact points along grid edges, where material vectors first transition from physical to non-physical, which we call **breaking points**. A breaking point is therefore the last point with a well-defined material label, before entering a region with an ill-defined material label. For each grid vertex with a non-physical material vector, we find its closest breaking point using a fast marching method [Sethian 1999]. We then overwrite the non-physical material vector with the unique material label of the closest breaking point. Figure 10 illustrates the process in 2D.

8.2 Creating and Connecting New Triangles

In this step, we describe how we generate new triangles to cleanly replace the deleted mesh. We generate each new triangle such that exactly one of its edges lies fully within a single grid face. We therefore think of each new triangle as belonging to a specific grid face. Note that there are two types of grid faces in the flawed region. A **boundary face** lies in the flawed boundary, and is therefore adjacent to one flawed cell, and one intact cell. Conversely, an **internal face** does not lie in the flawed boundary, and is therefore adjacent to two flawed cells.

We first focus on generating triangles that belong to an internal face. Given that we have a unique material label assigned to each grid vertex, we adjust the multi-material iso-surface extractor method of [Reitinger et al. 2005]. Their approach splits the grid cell into eight octants, each corresponding to one grid vertex, and therefore to one material label. Then it generates triangles that separate octants corresponding to differing material labels. Each triangle is defined by three special vertices: the point at the center of the grid cell, a point at the center of a grid face, and a point at the center of a grid edge; thus exactly one of this triangle's edges is fully contained within a grid face (Figure 12a). Given an internal face, we generate the subset of triangles generated by [Reitinger et al. 2005] that belong to this face. By design, the triangles belonging to an internal face connect perfectly to triangles belonging to adjacent internal faces.

Next we address how to generate triangles belonging to a boundary face. The generated triangles must connect to the clipped triangles in the intact region across the boundary face. Each of the clipped triangles has exactly one mesh edge lying exactly in the boundary face. We generate a new triangle for each such mesh edge by connecting the endpoints of the mesh edge with the center of the flawed cell, and then copying material labels to the new triangle from the clipped triangle (See Figure 12b). By construction, the newly generated triangles connect perfectly to the clipped mesh in the intact region.

Lastly, we have to ensure that whenever a boundary face is adjacent to an internal face, the triangles we generate inside them connect perfectly. Consider a grid edge e that forms the intersection of a boundary face b and an internal face i. In Section 6.4, we made sure that there are no complex edges in the flawed boundary. Consequently, there are only two possibilities for the triangles with a vertex on edge e: In the first case, there is exactly one triangle belonging to b and one triangle belonging to i, both of which have an edge connecting e to the cell center. These triangles have matching labels by construction. In order to seamlessly connect them, we slide the vertex of the triangle belonging to i along e, and match it with the vertex of the triangle belonging to b that lies on e. In the alternative case, there are no triangles belonging to b and i that need to be connected, and so there is nothing to be done.

With these steps completed, we conclude that the newly generated triangles seamlessly connect to each other and to the clipped mesh everywhere, while maintaining label consistency.

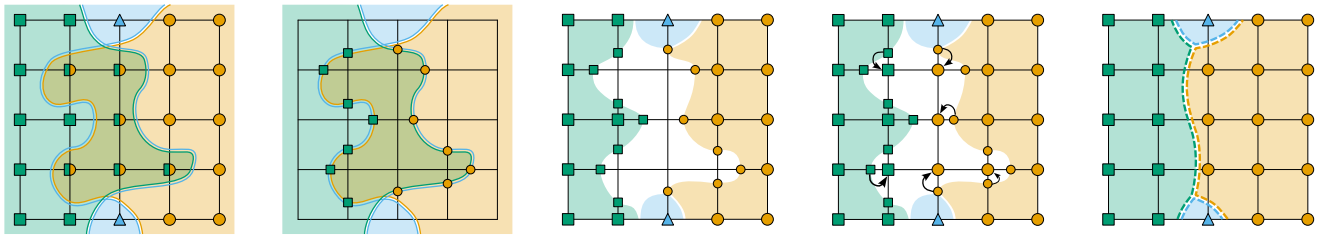


Fig. 10. **Correcting material vectors.** Starting in the complex region with invalid material vectors, our algorithm samples *breaking points* on the grid edges, finds the closest one to each complex vertex (indicated by black arrows), and reassigns valid material vectors to complex vertices from the found samples. The new material vector assignment allows us to reconstruct a properly embedded surface.

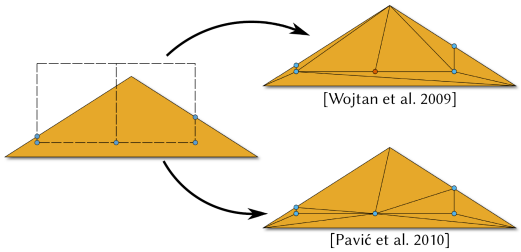
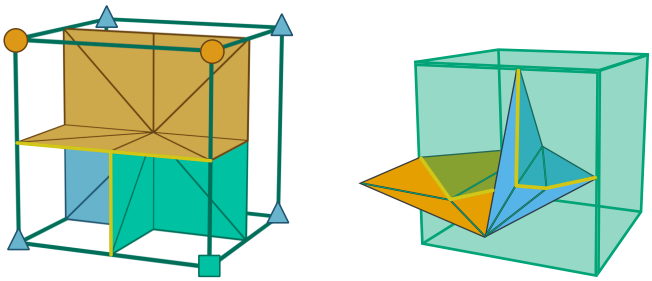


Fig. 11. Comparison of subdivision strategies on the complex boundary. [Wojtan et al. 2009] produces a T-junction with a 0-area triangle; constrained triangulation [Pavić et al. 2010] avoids these degeneracies.



(a) Internal face (b) Boundary face

Fig. 12. Reconstruction of geometry in a grid cell. Material labels on new triangles are inferred from either the material labels on grid vertices (a) or from the clipped triangles in the intact region (b). Highlighted edges lie within the front facing grid face.

8.3 Determining Vertex Positions

Having ensured the desired *connectivity* of the newly generated mesh, we now adjust its *geometry*. We focus on regions that are remeshed on account of being non-physical, as it is in these regions where we might need to reconstruct complicated geometry from scratch. We recall that all newly generated mesh vertices lie either on grid edges, in grid faces, or at grid cell centers, and note that we are free to shift these vertices along their grid edges, within their grid faces, and within their grid cells respectively, without changing the mesh connectivity.

To determine the location of mesh vertices along grid edges, we use a familiar isosurface extraction strategy of finding the optimal transition point from one material to another. We already computed *breaking points* (the closest physically valid points to each non-physical grid vertex) in Section 8.1. For a grid edge with a newly generated mesh vertex, if it is adjacent to one physical and one non-physical grid vertex, then it has exactly one breaking point, and we slide the mesh vertex to this point. For a grid edge with two adjacent non-physical grid vertices, we find the point along the edge that is equidistant to the breaking points stored at its two adjacent grid vertices. The two grid vertices by construction have different material labels, so the new position of the mesh vertex is our best guess of the transition point from one material to another.

We experimented with computing similar optimizations for newly generated vertices on grid faces and cell centers, but in the end we found it sufficient to heuristically set these vertex positions to the geometric average of mesh vertices on adjacent grid edges, as this approach results in a smooth surface within each grid cell.

We note that our strategy for placing new vertices is preliminary; the generic problem may even be ill-posed, since the notion of optimal placement and which data is trustworthy will strongly depend on application-specific information. For our visual applications, we found it useful to smooth non-manifold curves in the mesh: for each new vertex adjacent to a non-manifold triangle edge, we execute 10 steps of curve smoothing by iteratively setting its position as the average of its non-manifold neighbors.

8.4 Transfer Mesh Properties to New Elements

Common applications store properties (e.g. texture coordinates or simulation variables like velocity) on mesh vertices, edges, or triangles, so we transfer these properties from the input mesh to the recomputed one in this step. Before proceeding, we note that an "optimal" transfer of properties is application-specific, and some conservative or higher-order reconstruction may be more appropriate in some cases. Furthermore, it is unclear whether it is even possible to define a generic application-independent optimization problem for the transfer of data between surfaces of differing topology.

We follow a simple particle-in-cell (PIC) [Harlow 1964] strategy for passing information from the old mesh to the new one via an intermediate grid data structure. We discuss how to do this for data

stored at mesh vertices; data stored at triangles and edges can be treated similarly by storing data at their barycenter, or they might need special treatment. The PIC algorithm proceeds as follows:

(1) Transfer data from the input mesh to the grid: each grid vertex in the flawed region stores the average data from mesh vertices located in its adjacent flawed cells.
(2) Propagate this data to the grid vertices with no associated data in the complex region: flood fill via breadth-first search (BFS), starting from the grid vertices that received data in step (1). If a grid vertex receives data from more than one source in the same BFS step, store the averaged data. This process ensures that triangles generated in grid cells that contained no input triangles will receive data from the nearest input mesh.
(3) Transfer data to the newly generated mesh: new mesh vertices receive data via tri-linear interpolation from the grid.

8.5 Local Mesh Improvement

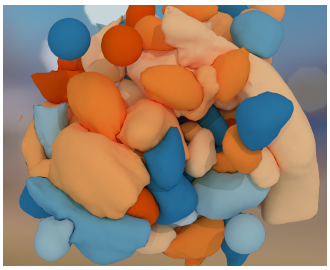
The last step of our algorithm is to perform local mesh improvement. Given that the main focus of explicit surface tracking algorithms is to achieve the correct mesh topology of evolving surfaces, the reconstructed mesh might not be directly suitable for simulation. Mesh improvement is a practical tool commonly used to remedy this by improving mesh quality. We follow [Da et al. 2014] by performing vertex separation, edge splits, edge flips, edge collapses, and vertex smoothing. The resulting triangles are acceptable for efficient physics simulation, though we believe our implementation of mesh improvement has substantial room for further optimization, both theoretically (in terms of the range of allowed non-manifold local operations) and practically (with a more efficient implementation).

9 Results, comparisons, and discussion

We performed a number of simulations and stress tests by alternating one step of mesh deformation for one step of our topological change algorithm.

9.1 Parameters

Our topology-changing algorithm has a single parameter: the topological resolution ℓ which determines the size of the background grid cells. Our mesh improvement step (Section 8.5) features the expected suite of parameters for minimum/maximum edge length and criteria for flipping edges. For all examples, we set the topological resolution ℓ equal to the average triangle edge length, and keep the local mesh improvement parameters such that the final triangle edge lengths are within 0.5–1.5 times the topological resolution ℓ. For comparisons with Los Topos [Da et al. 2014], we keep their "merge ratio" equal to 0.02 times the average triangle edge length—a comparatively more conservative choice of merging distance used in the Los Topos repository. In our limited experience, increasing the merge ratio in Los Topos seems to sew triangles together more often but increases the chance of the algorithm being stuck in an infinite loop. On the other hand, decreasing the Los Topos merge ratio makes surfaces repel each other instead of merging, leading to long, snake-like bubbles like the inset figure.



9.2 Computational Efficiency

Our method's computational cost is weakly dependent on the complexity of the input problem. The cost is mostly proportional to the number of intersections between the triangle mesh and the background grid. A spacious mesh and denser background grid (smaller ℓ) results in more ray-triangle intersection tests and more tests for non-physical vertices and edges, which make up the majority of our algorithm's cost. In contrast, although mesh surgery is more computationally arduous, it is performed relatively rarely.

The computational cost of our method does not depend on the topological complexity of the mesh (number of self-intersections or overlaps) in any obvious way. Our method uses a sparse grid to gain efficiency over previous grid-based solvers; we can expect the solver to gain efficiency as we increase the ratio of volume to surface area in the input mesh. We note that this sparse grid speedup is *not* showcased by our examples and stress tests, which overwhelmingly consist of a space-filling volumetric mess of triangles. Our naïve implementation of local mesh improvement (Section 8.5) is the current bottleneck in our code, taking up roughly 80% of the total run-time.

Like Wojtan et al. [2009; 2010], our method has a built-in safety net; if it ever encounters an internally inconsistent state (perhaps due to catastrophic numerical degeneracy), it marks the affected region for remeshing, aborts the current computation, and re-starts the algorithm with a guarantee that the offending geometry will be replaced with a clean mesh. Our implementation never actually executed this safety-net algorithm in any of our tests, so we suspect that our other robustness improvements made this procedure obsolete.

Directly comparing computational cost between our method and Los Topos is not straightforward. We can confidently conclude that our method can take larger time steps, which is reflected both in our performance statistics and the strongly differing algorithmic designs (Los Topos takes incremental adaptive steps while ours jumps right to the final deformed state). However, the computational cost per time step is strongly dependent on the scenario: our method performs a comparatively larger number of operations regardless of input complexity, while Los Topos can range from almost no computation per step (if no collisions are detected) to a significant loop of computation (for complex intersections requiring adaptive time steps).

9.3 Verification and Stress Tests

The rotating notched disk was designed by Zalesak [1979] to highlight spurious surface deformations in a surface tracker. We use a multi-material version of the test introduced by Da et al. [2014] in Figure 13 and our supplementary video. Like most other Lagrangian surface tracking techniques, our method completes the test without any re-sampling or spurious topology changes.

Fig. 13. **Zalesak disk.** A multi-material test disk rotating around an exterior point without resampling. The disk consists of 108 triangles and fits a 9×9×3 grid region. The whole test runs on a 25^3 grid.

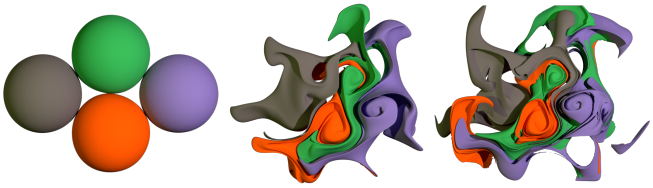


Fig. 14. **Curl noise.** Four spheres of different materials stretching out in a procedural divergence-free flow.

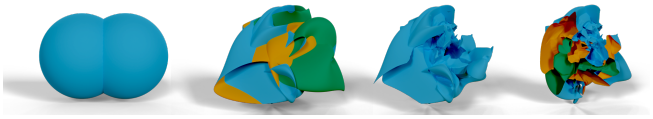


Fig. 15. **Inversion test.** *(Left)* Initial mesh of two spheres with a blue outside material and orange and green internal ones. *(Center left)* Surface mesh deformed with strong Perlin noise. Mesh inversions are shown as orange and green colors. *(Center right)* Fixed mesh with our algorithm. *(Right)* Cut-away view of the result.

Figure 15 tests our method's ability to identify and resolve topological problems in extremely deformed and inverted shapes. The test applies strong vector-valued Perlin noise to the surface of a double bubble consisting of two different materials. The noise then instantly deforms the original shape into a highly contorted and self-intersecting blob. Our method resolves these topological problems in a single step, returning a new shape just as visually unpleasant as the input, but without any self-intersections, and with the interior cleanly separated into different volumes.

Figure 14 repeats a test by Da et al. [2014] featuring four spheres of different materials passively advected through a divergence-free flow. The initially bulbous spheres quickly stretch out into thin wisps, stressing a surface tracker's ability to resolve thin features. It also brings surfaces close together, causing topological merges if surfaces come within the distance threshold ℓ. Our method dutifully merges these surfaces when they come close, though it does erode some small features when they become thinner than the distance threshold ℓ. Increasing the grid resolution (lowering ℓ) delays this effect, similar to other implicit surface techniques. In the original experiment, Los Topos sets the merge distance threshold 1000-times lower than the average triangle edge length in the simulation. This 1:1000 ratio highlights the differences in the way our algorithms implement this parameter: in Los Topos, it essentially tells the algorithm to wait until the last possible moment before colliding,

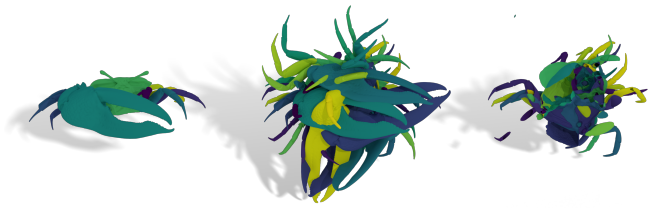
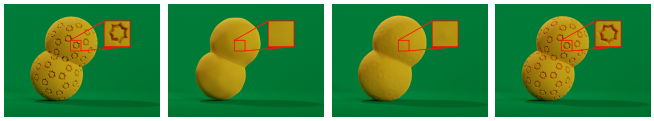


Fig. 16. **Dr. Krabunkle.** A detailed crab mesh (left) is cloned and rotated around its center of mass 5 times to generate a shape with 5.3 million triangles and 72 materials. Our algorithm resolves all overlaps on a 570^3 grid in 8 minutes producing a mesh with 3.6 million triangles (center). Cut-away view (right) shows that internal materials are correctly separated.



(a) Ours (b) GR (c) PR (d) PR ($7^3\times$ grid)

Fig. 17. **Rolling stones.** Two decorated spheres roll into each other, merge and roll away, tracked by (a) our algorithm, (b) a modified algorithm with global resampling (GR), and (c)–(d) particle-based methods (PR). (a)–(c) use the same grid resolution, while (d) requires a $7^3\times$ denser voxel grid.

and then snaps vertices together without creating small triangles. In contrast, such a large ratio causes our method to allocate a large grid, and, in the event of a topological change, it subdivides triangles into excessively tiny bits before collapsing them all together again during a particularly busy follow-up re-meshing step. To reflect the difference in our methods while still retaining the spirit of the original idea, we set the grid resolution ℓ in this experiment to be 10 times lower than the mesh resolution.

9.4 Comparison with Particle-Based Methods

To highlight the importance of only re-sampling surfaces where absolutely necessary, we track the motion of two ornamental spheres (each consisting of 37937 vertices and 75870 triangles) rolling and merging together. We compare the performance of four different surface tracking algorithms in Figure 17 and our video. Figure 17a uses our algorithm and perfectly preserves the original surface details. Figure 17b uses a modified version of our algorithm which naïvely reconstructs the surface everywhere on the grid at every step (instead of limiting the flawed region to a minimial size), leading to degradation of surface details after repeated re-sampling.

We also consider an alternative meshless method for tracking the surface which densely samples the original surfaces with 200k particles once at the start, then moves the particles at every time step and extracts a new surface each frame using VDB [Museth 2013] implemented in the Houdini software package. This idealized alternative *never* re-samples the surface particles and experiences no degradation over time, but it requires a large number of particle samples and a dense grid to reconstruct a detailed and temporally coherent surface. Figure 17c uses the same grid resolution as our

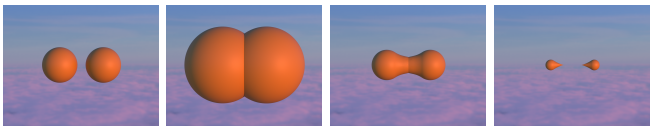


Fig. 18. **Double bubble.** Two spheres expand in the normal direction at a constant speed, then merge, reverse direction and separate.

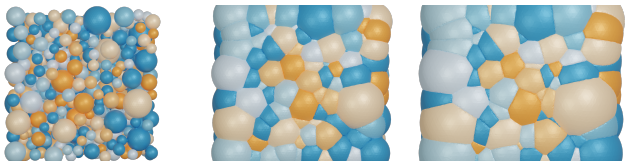


Fig. 19. **Normal flow of 1000 spheres.** One thousand spheres, each bounding its own unique material region, are expanding in the normal flow direction at a constant speed, creating many topological events.

method, while Figure 17d uses a grid with voxels that are 7× smaller in each dimension. The particle-based method is unable to reconstruct the fine surface details without this extraordinarily dense grid, leading to higher computational overheads in computational time, memory, and triangle count (1.4 million).

9.5 Multi-Material Solid Geometry

Instead of slowly deforming a surface from one shape into another, our method allows arbitrarily large deformations/time steps. This allows us to essentially teleport shapes into non-embedded spatial configurations and resolve all topological problems at once. We show an extreme example of this in Figure 16, where we instantly overlap six detailed crab meshes from the Three D Scans repository [Laric 2012], each consisting of 12 overlapping materials and 878617 triangles. Our method resolves the overlapping materials by approximating a multi-material boolean union of these shapes. The method works robustly despite the high resolution and degenerate input triangles.

9.6 Normal Flow and Robustness

We demonstrate another stress test consisting of numerous multi-material overlaps and non-manifold surface geometry in Figures 18 and 19 by evolving a set of spheres outward in their normal direction at a constant speed. Figure 18 recreates the basic scenario of Da et al. [2014], in which two spheres expand until they merge into a double bubble™, then reverse the flow, causing the structure to separate and dwindle to deletion.

Using many more spheres creates a significantly more stressful surface tracking scenario. Figure 19 illustrates how our method copes with 100 and 1000 expanding spheres to create a complex foam with myriads of materials. Furthermore, we can randomize the initial placement of spheres (ensuring that they start from a non-intersecting initial state) and re-run the same experiment multiple times to create a statistical benchmark for surface tracker robustness. We find these statistical tests useful for comparing the robustness of different surface tracking algorithms, because theoretical failure analysis of all competing approaches is difficult and incomplete.

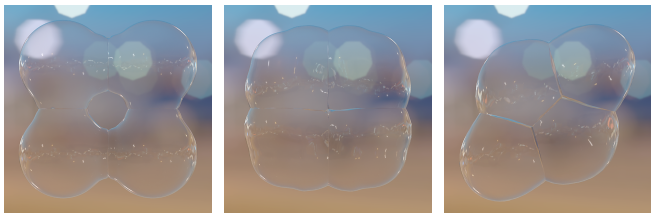


Fig. 20. **Symmetry breaking.** Four soap bubbles colliding with each other and choosing one preferred direction for area minimization.

We define success in this scenario if our algorithm completes the task without crashing, running out of memory, or failing to finish within ten hours. In this scenario with 100 spheres, our method succeeded in 100 out of 100 tests (a 100% success rate) with an average run-time of 6.1 minutes, while using naïve numerical evaluation of vertex normals to compute the flow direction. In comparison, the state of the art surface tracker Los Topos [Da et al. 2014], succeeded in 85 out of 100 tests (85%) with an average run-time of 84.5 minutes. Using *Face Offsetting* [Jiao 2007] for a more stable normal flow keeps Los Topos's success rate at 83 out of 100 (83%, 81.1 minutes run-time). Increasing the complexity by flowing 1000 spheres leads to a success rate of 50 out of 50 (100%, 44.6 minutes run-time) for our method, 13 out of 50 (26%, 30.6 minutes run-time) for Los Topos with Face Offsetting and 0 out of 50 (0%) for Los Topos with naïve normal flow. Unexpectedly, Los Topos performs faster on the larger test. This discrepancy is explained by survivor bias related to the algorithm's intersection avoidance procedure: Los Topos repeatedly halves the time step and retries all computations until a state with no intersections is reached. This strategy allows the 100 sphere simulations to make progress during frames with many intersection events but at a cost of 3 time-step reductions (and restarts) per frame on average. For the 1000 sphere test, this recovery strategy always fails, but the successful runs avoided restarts altogether and finished more quickly.

We note a number of caveats with our proposed robustness benchmark which make it difficult to directly draw conclusions: first, although we found it useful for probing a large number of unexpected configurations, it is not exhaustive and may not fairly sample the space of mesh configurations. Furthermore, normal flow may not be representative for other applications, especially those with substantial tangential motion. We also did not optimize parameter settings for either algorithm, so it is likely that changing grid size or the Los Topos merge ratio will influence these results. Finally, the two tested methods have different constraints on the solution; Los Topos guarantees that the final mesh will be free of self-intersections, while ours only guarantees this for intersections larger than some constant proportional to ℓ.

9.7 Soap Bubble Simulation

We couple our method to the soap film evolution method of Ishida et al. [2017]. We took authors' original code and replaced the Los Topos surface tracker with our method. Figure 20 shows four bubbles spontaneously breaking symmetry to minimize surface area.

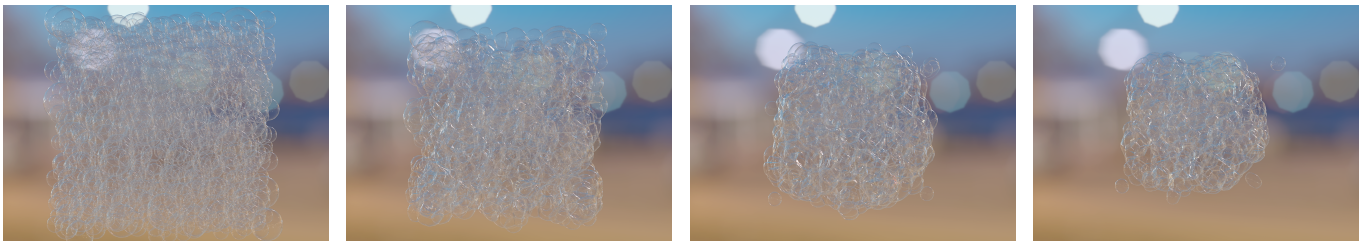


Fig. 21. **Soap foam.** Our mesher robustly handles the complexity of soap films, allowing us to simulate them in such detail at a never-before-seen scale.

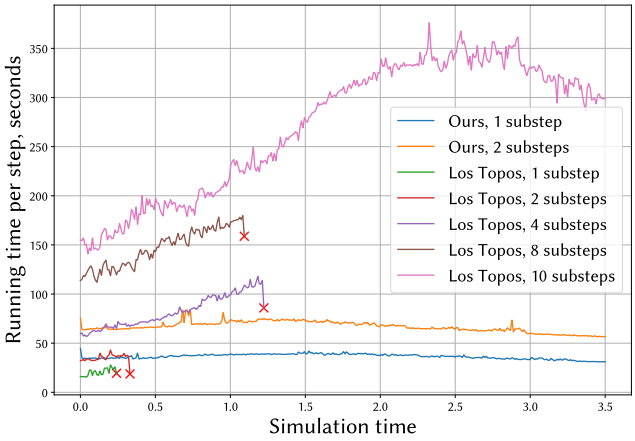


Fig. 22. **Running time for 1000 soap bubbles test.** A comparison of per-frame running times for two different surface trackers on the same 1000-soap-bubble animation. Timings exclude the cost of physics calculations (i.e., surface tracker only). × indicates failure of the surface tracker.

Figure 21 simulates 1000 interacting bubbles undergoing volume-preserving surface tension flow, merging together, bursting, rippling, and rearranging. To the best of our knowledge, this is the largest and most complex foam simulation of its kind ever computed. Figure 22 lists performance statistics for this example: with roughly 625000 triangles, our method demonstrated an average run-time per time step of 36.5 seconds on an Intel® Core™ i7-7820X with 64 GB of RAM. Compared to Los Topos, our method allowed a 10 times larger time step size and computed 7.5 times faster overall.

9.8 Further Discussion

To meet expectations for peer review, we found it necessary to directly compare our algorithm with the state of the art multi-material surface tracking algorithm, Los Topos. However, we believe that each algorithm excels in different scenarios. Our implementation robustly handles larger deformations more efficiently, but Los Topos runs faster in less complicated scenarios and provides guarantees that our method does not. Regardless, we do not believe our direct comparisons are definitive, as they are based on two research implementations that may have inefficiencies and bugs.

10 Limitations and Future Work

Our method has room for improvement on the implementation and algorithmic levels. As mentioned in Section 9, our implementation will benefit from a more sophisticated strategy for mesh improvement, which would make our method more efficient by removing the computational bottleneck, and even more widely applicable by improving element quality for PDE solvers. Our strategy for creating a smooth non-manifold surface from large overlaps relies on grid-related heuristics and post-hoc mesh fairing; it should be possible to phrase this problem more elegantly and find a more direct solution.

Our algorithm for computing topological changes depends on how the mesh intersects the grid, and our implementation generally avoids testing for complex faces and cells everywhere, so it could miss very small topological defects. In the future we would like to reinforce our approach with a hard guarantee that we fix topological flaws at *all* scales, in addition to the ones larger than ℓ. Our algorithm does not attempt to preserve volume exactly, because it is a general-purpose surface tracker meant to handle all possible deformations. For small time steps, our local re-meshing strategy will change volumes proportional to the grid edge length ℓ and the surface area of the mesh regions which merge together or split apart. This is consistent with Los Topos and other mesh-based surface trackers that change the volume by merging surfaces which lie closer than the user-defined merge threshold.

In the future, we would also like to consider extending our multi-material approach with techniques that proved useful in the two-material case, like support for thin sheets [Wojtan et al. 2010], grid-free triangulation [Bo et al. 2011; Chentanez et al. 2015], and GPU acceleration [Chentanez et al. 2016].

In conclusion, we introduced a surface tracking algorithm for non-manifold surface meshes that generalizes the manifold surface tracker of Wojtan et al. [2009]. Our method works efficiently and reliably, and is capable of state-of-the-art performance on large practical problems in physics animation and solid geometry processing.

Acknowledgments

Peter Heiss-Synak helped conceive the project, helped formulate the algorithm structure, contributed ideas and code to Sections 6 & 8, the mesh data structure, algorithm robustness and benchmarks, helped write the paper, and provided supervision and conceptual solutions throughout the project. Aleksei Kalinov contributed ideas and code to Sections 7, 8.5, and 5, the sparse grid data structure, algorithm robustness and benchmarks, optimized the performance, produced

all results, most figures, and the supplementary video, helped write the text, and provided conceptual solutions throughout the project. Malina Strugaru helped implement the mesh data structure and designed re-meshing operations for non-manifold triangle meshes. Arian Etemadi developed early prototypes for ideas in Sections 8.1 and 8.3 and helped write the paper. Huidong Yang developed early prototypes for isosurface extraction and visualization. Chris Wojtan helped conceive the project, helped write the paper, and provided supervision, prototype grid data structure code, and conceptual solutions throughout the project.

We thank the anonymous reviewers for their helpful comments, the members of the Visual Computing Group at ISTA for their feedback, Christopher Batty for discussions about LosTopos, and SideFX for the Houdini Education software licenses. This research was funded in part by the European Union (ERC-2021-COG 101045083 CoDiNA).

References

Adam W Bargteil, Tolga G Goktekin, James F O'Brien, and John A Strain. 2006. A Semi-Lagrangian Contouring Method for Fluid Simulation. *ACM Trans. Graph.* 25, 1 (2006).

Gavin Barill, Neil G. Dickson, Ryan Schmidt, David I. W. Levin, and Alec Jacobson. 2018. Fast Winding Numbers for Soups and Clouds. *ACM Trans. Graph.* 37, 4 (2018), 43:1–43:12. doi:10.1145/3197517.3201337

Gilbert Louis Bernstein and Chris Wojtan. 2013. Putting holes in holey geometry: Topology change for arbitrary surfaces. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–12.

Stephan Bischoff and Leif Kobbelt. 2005. Structure preserving CAD model repair. In *Computer Graphics Forum*, Vol. 24. Amsterdam: North Holland, 1982-, 527–536.

Wurigen Bo, Xingtao Liu, James Glimm, and Xiaolin Li. 2011. A Robust Front Tracking Method: Verification and Application to Simulation of the Primary Breakup of a Liquid Jet. *SIAM J. Sci. Comput.* 33, 4 (2011), 1505–1524. doi:10.1137/10079135X

Kenneth A Brakke. 1992. The surface evolver. *Experimental mathematics* 1, 2 (1992), 141–165.

Tyson Brochu and Robert Bridson. 2009. Robust Topological Operations for Dynamic Explicit Surfaces. *SIAM J. Sci. Comput.* 31, 4 (2009), 2472–2493. doi:10.1137/080737617

Paul Burchard, Li-Tien Cheng, Barry Merriman, and Stanley Osher. 2001. Motion of curves in three spatial dimensions using a level set approach. *J. Comput. Phys.* 170, 2 (2001), 720–741.

Nuttapong Chentanez, Matthias Müller, and Miles Macklin. 2016. GPU Accelerated Grid-Free Surface Tracking. *Computers & Graphics* 57, C (2016), 1–11. doi:10.1016/j.cag.2016.03.002

Nuttapong Chentanez, Matthias Müller, Miles Macklin, and Tae-Yong Kim. 2015. Fast Grid-Free Surface Tracking. *ACM Trans. Graph.* 34, 4 (2015), 148:1–148:11. doi:10.1145/2766991

Fang Da, Christopher Batty, and Eitan Grinspun. 2014. Multimaterial Mesh-Based Surface Tracking. *ACM Trans. Graph.* 33, 4 (2014), 112:1–112:11. doi:10.1145/2601097.2601146

Fang Da, Christopher Batty, Chris Wojtan, and Eitan Grinspun. 2015. Double bubbles sans toil and trouble: Discrete circulation-preserving vortex sheets for soap films and foams. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 1–9.

Lorenzo Diazzi and Marco Attene. 2021. Convex Polyhedral Meshing for Robust Solid Modeling. *ACM Trans. Graph.* 40, 6 (2021), 259:1–259:16. doi:10.1145/3478513.3480564

Jian Du, Brian Fix, James Glimm, Xicheng Jia, Xiaolin Li, Yuanhua Li, and Lingling Wu. 2006. A simple package for front tracking. *J. Comput. Phys.* 213, 2 (2006), 613–628.

Herbert Edelsbrunner and Ernst Peter Mücke. 1990. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics (tog)* 9, 1 (1990), 66–104.

Xianzhong Fang, Mathieu Desbrun, Hujun Bao, and Jin Huang. 2022. TopoCut: Fast and Robust Planar Cutting of Arbitrary Domains. *ACM Trans. Graph.* 41, 4 (2022), 1–15. doi:10.1145/3528223.3530149

Steven W Gagniere, Yushan Han, Yizhou Chen, David AB Hyde, Alan Marquez-Razon, Joseph Teran, and Ronald Fedkiw. 2022. A Robust Grid-Based Meshing Algorithm for Embedding Self-Intersecting Surfaces. *arXiv preprint arXiv:2201.06256* (2022).

James Glimm, John W. Grove, X. L. Li, and D. C. Tan. 2000. Robust Computational Algorithms for Dynamic Interface Tracking in Three Dimensions. *SIAM Journal on Scientific Computing* 21, 6 (Jan. 2000), 2240–2256. doi:10.1137/S1064827598340500

Francis H Harlow. 1964. The particle-in-cell computing method for fluid dynamics. *Methods Comput. Phys.* 3 (1964), 319–343.

Andreas Hubeli and Markus Gross. 2000. Fairing of non-manifolds for visualization. In *Proceedings Visualization 2000. VIS 2000 (Cat. No. 00CH37145)*. IEEE, 407–414.

Sadashige Ishida, Peter Synak, Fumiya Narita, Toshiya Hachisuka, and Chris Wojtan. 2020. A Model for Soap Film Dynamics with Evolving Thickness. *ACM Trans. Graph.* 39, 4, Article 31 (aug 2020), 11 pages. doi:10.1145/3386569.3392405

Sadashige Ishida, Chris Wojtan, and Albert Chern. 2022. Hidden degrees of freedom in implicit vortex filaments. *ACM Transactions on Graphics (TOG)* 41, 6 (2022), 1–14.

Sadashige Ishida, Masafumi Yamamoto, Ryoichi Ando, and Toshiya Hachisuka. 2017. A hyperbolic geometric flow for evolving films and foams. *ACM Transactions on Graphics (TOG)* 36, 6 (2017), 1–11.

Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. 2013. Robust Inside-Outside Segmentation Using Generalized Winding Numbers. *ACM Trans. Graph.* 32, 4 (2013), 33:1–33:12. doi:10.1145/2461912.2461916

Xiangmin Jiao. 2007. Face offsetting: A unified approach for explicit moving interfaces. *Journal of computational physics* 220, 2 (2007), 612–625.

Byungmoon Kim. 2010. Multi-phase fluid simulations using regional level sets. *ACM Transactions on Graphics (TOG)* 29, 6 (2010), 1–8.

Byungmoon Kim, Yingjie Liu, Ignacio Llamas, Xiangmin Jiao, and Jarek Rossignac. 2007. Simulation of bubbles in foam with the volume control method. *ACM Transactions on Graphics (TOG)* 26, 3 (2007), 98–es.

Oliver Laric. 2012. *Three D Scans.* Retrieved 21 Januarry 2024 from https://threedscans.com

Xiaosheng Li, Xiaowei He, Xuehui Liu, Jian J. Zhang, Baoquan Liu, and Enhua Wu. 2016. Multiphase Interface Tracking with Fast Semi-Lagrangian Contouring. *IEEE Transactions on Visualization and Computer Graphics* 22, 8 (2016), 1973–1986. doi:10.1109/TVCG.2015.2476788

Yijing Li and Jernej Barbič. 2018. Immersion of Self-Intersecting Solids and Surfaces. *ACM Trans. Graph.* 37, 4 (2018), 45:1–45:14. doi:10.1145/3197517.3201327

Frank Losasso, Tamar Shinar, Andrew Selle, and Ronald Fedkiw. 2006. Multiple interacting liquids. *ACM Transactions on Graphics (TOG)* 25, 3 (2006), 812–819.

Marek Krzysztof Misztal and Jakob Andreas Bærentzen. 2012. Topology-Adaptive Interface Tracking Using the Deformable Simplicial Complex. *ACM Trans. Graph.* 31, 3 (2012), 24:1–24:12. doi:10.1145/2167076.2167082

Matthias Müller. 2009. Fast and Robust Tracking of Fluid Surfaces. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New Orleans, Louisiana) *(SCA '09)*. Association for Computing Machinery, New York, NY, USA, 237–245. doi:10.1145/1599470.1599501

Ken Museth. 2013. VDB: High-resolution Sparse Volumes with Dynamic Topology. 32, 3 (2013), 27:1–27:22. doi:10.1145/2487228.2487235

Stanley Osher and Ronald P Fedkiw. 2001. Level set methods: an overview and some recent results. *Journal of Computational physics* 169, 2 (2001), 463–502.

Darko Pavić, Marcel Campen, and Leif Kobbelt. 2010. Hybrid booleans. In *Computer Graphics Forum*, Vol. 29. Wiley Online Library, 75–87.

J-P Pons and J-D Boissonnat. 2007. A lagrangian approach to dynamic interfaces through kinetic triangulation of the ambient space. In *Computer Graphics Forum*, Vol. 26. Wiley Online Library, 227–239.

Bernhard Reitinger, Alexander Bornik, and Reinhard Beichel. 2005. Constructing Smooth Non-Manifold Meshes of Multi-Labeled Volumetric Datasets. *The 13-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision* (2005).

Jarek Rossignac. 2001. 3D compression made simple: Edgebreaker with zipandwrap on a corner-table. In *Proceedings International Conference on Shape Modeling and Applications*. IEEE, 278–283.

Silvia Sellán, Herng Yi Cheng, Yuming Ma, Mitchell Dembowski, and Alec Jacobson. 2019. Solid geometry processing on deconstructed domains. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 564–579.

James A Sethian. 1999. Fast marching methods. *SIAM review* 41, 2 (1999), 199–235.

Nicholas Sharp and Keenan Crane. 2020. A laplacian for nonmanifold triangle meshes. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 69–80.

D. She, R. Kaufman, H. Lim, J. Melvin, A. Hsu, and J. Glimm. 2016. Chapter 15 - Front-Tracking Methods. In *Handbook of Numerical Analysis*, Rémi Abgrall and Chi-Wang Shu (Eds.). Handbook of Numerical Methods for Hyperbolic Problems, Vol. 17. Elsevier, 383–402. doi:10.1016/bs.hna.2016.07.004

Jonathan Richard Shewchuk. 1996. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Workshop on applied computational geometry*. Springer, 203–222.

Kenji Shimada and David C Gossard. 1995. Bubble mesh: automated triangular meshing of non-manifold geometry by sphere packing. In *Proceedings of the third ACM symposium on Solid modeling and applications*. 409–419.

Michael Tao, Christopher Batty, Eugene Fiume, and David I. W. Levin. 2019. Mandoline: Robust Cut-Cell Generation for Arbitrary Triangle Meshes. *ACM Trans. Graph.* 38, 6 (2019), 1–17. doi:10.1145/3355089.3356543

Philip Trettner, Julius Nehring-Wirxel, and Leif Kobbelt. 2022. EMBER: Exact Mesh Booleans via Efficient & Robust Local Arrangements. *ACM Trans. Graph.* 41, 4 (2022), 1–15. doi:10.1145/3528223.3530181

Marc Wagner, Ulf Labsik, and Günther Greiner. 2003. Repairing non-manifold triangle meshes using simulated annealing. *International Journal of Shape Modeling* 9, 02 (2003), 137–153.

Charlie CL Wang. 2010. Approximate boolean operations on large polyhedral solids with partial mesh reconstruction. *IEEE transactions on visualization and computer graphics* 17, 6 (2010), 836–849.

Chris Wojtan, Nils Thürey, Markus Gross, and Greg Turk. 2009. Deforming Meshes That Split and Merge. *ACM Trans. Graph.* 28, 3 (2009), 76:1–76:10. doi:10.1145/1531326.1531382

Chris Wojtan, Nils Thürey, Markus Gross, and Greg Turk. 2010. Physics-Inspired Topology Changes for Thin Fluid Features. *ACM Trans. Graph.* 29, 4 (2010), 50:1–50:8. doi:10.1145/1778765.1778787

Meng Yang, Juntao Ye, Frank Ding, Yubo Zhang, and Dong-Ming Yan. 2019. A Semi-Explicit Surface Tracking Mechanism for Multi-Phase Immiscible Liquids. *IEEE Transactions on Visualization and Computer Graphics* 25, 10 (2019), 2873–2885. doi:10.1109/TVCG.2018.2864283

Lexing Ying and Denis Zorin. 2001. Nonmanifold subdivision. In *Proceedings Visualization, 2001. VIS'01.* IEEE, 325–569.

Steven T Zalesak. 1979. Fully multidimensional flux-corrected transport algorithms for fluids. *Journal of computational physics* 31, 3 (1979), 335–362.

Wen Zheng, Jun-Hai Yong, and Jean-Claude Paul. 2009. Simulation of bubbles. *Graphical Models* 71, 6 (2009), 229–239.

Qingnan Zhou, Eitan Grinspun, Denis Zorin, and Alec Jacobson. 2016. Mesh Arrangements for Solid Geometry. *ACM Trans. Graph.* 35, 4 (2016), 39:1–39:15. doi:10.1145/2897824.2925901

A Robust mesh clipping

To clip a triangle mesh to the flawed boundary, we find which mesh edges need to be added for each triangle, similar to the work of [Pavić et al. 2010], and call the Triangle library to perform constrained Delaunay triangulation. For the new triangulation, we need to determine which triangles lie inside the flawed region, and should therefore be deleted. In order to avoid additional numerical tests, we reuse the grid face-mesh edge intersections, and mark triangles for deletion by a flood-filling algorithm.

For similar reasons, we use integer grid coordinates for storing the positions of newly generated triangulation vertices, instead of relying on the floating point representation. Vertices on triangle edges also lie in grid faces, we sort them on each triangle edge using integer grid face coordinates, similar to Mandoline [Tao et al. 2019]. Vertices inside triangle faces also lie on grid edges, we use integer grid edge coordinates to move them in a way that maximizes mutual distance, and minimizes degeneracy. We speculate that the robustness of this algorithm can be increased even further in the future by reformulating the problem as an integer-based Planar Straight-Line Drawing.