Filling the Holes of Non-Manifold Self-Intersecting Meshes

for Implicit Topology Changes in Surface Tracking

by

Arian Etemadi

September, 2024

A thesis submitted to the
Graduate School
of the
Institute of Science and Technology Austria
in partial fulfillment of the requirements
for the degree of
Master of Science

Committee in charge:

Chris Wojtan, Supervisor
Krishnendu Chatterjee

The thesis of Arian Etemadi, titled *Filling the Holes of Non-Manifold Self-Intersecting Meshes*, is approved by:

Supervisor: Chris Wojtan, ISTA, Klosterneuburg, Austria

Signature: _____

Committee Member: Krishnendu Chatterjee, ISTA, Klosterneuburg, Austria

Signature: _____

Signed page is on file

ISTA Master's Thesis, ISSN: 2791-4585

I hereby declare that this thesis is my own work and that it does not contain other people's work without this being so stated; this thesis does not contain my previous work without this being stated, and the bibliography contains all the literature that I used in writing the dissertation.

I accept full responsibility for the content and factual accuracy of this work, including the data and their analysis and presentation, and the text and citation of other work.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee, and that this thesis has not been submitted for a higher degree to any other university or institution.

I certify that any republication of materials presented in this thesis has been approved by the relevant publishers and co-authors.

Signature: _____

Arian Etemadi
September, 2024

Signed page is on file

Abstract

Physics simulation in computer graphics can bring triangle meshes into topologically invalid states. The method in this thesis contributed to Heiss-Synak* and Kalinov* et al. [2024] who devised a non-manifold hybrid surface tracker—a surface tracker that repairs explicit non-manifold triangle meshes with the help of the implicit domain. Specifically, this thesis provides an algorithm for filling the holes that are left after removing problematic parts of the mesh.

Acknowledgements

I would like to thank Chris, Peter, and Aleksei, for their support.

To Farnaz, Reza, and Amir, for all their sacrifices.

List of Collaborators and Publications

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)

Table of Contents

List of Figures

CHAPTER 1

Introduction

Surface tracking is an essential part of animating scenes in computer graphics—scenes that are composed of various objects and materials with distinct physical properties. Examples include water flowing downstream in the river, showing all sorts of intricate waves and ripples in different scales on the surface; honey dribbling down the spoon onto the plate, viscously folding onto itself, forming interesting creases that soon vanish into smoothness; and bubbles approaching each other, merging into a cluster, sliding across each other in the attempt to minimize surface area.

In all these scenarios, the scene configuration in a single frame is sufficiently captured by the geometry and the topology of two-dimensional surfaces. Surfaces delineate interfaces between distinct materials; they mark where one material ends and the other starts. Some surfaces bound solid objects. Some separate non-mixing fluids from each other (e.g. oil, water, and air). As surfaces are moved by the invisible laws of physics over thousands of discrete time steps, each frame is rendered slightly different than the previous one and an animation is generated.

As we will see, implementing physics alone does not necessarily lead to robust methods. It must be accompanied by algorithms from geometry processing that specialize in maintaining not just the quality, but also, the semantics of these surfaces—*surface tracking algorithms*.

1.1 Implicit and Explicit Surfaces

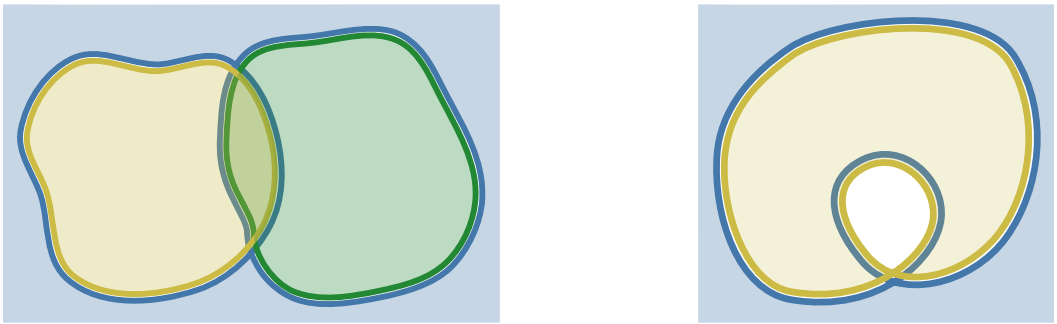
There are two general approaches to representing surfaces in computer graphics: *explicit* and *implicit*—also known as *Lagrangian* and *Eulerian* respectively. The former

is the usual and the more intuitive way: surfaces are explicitly constructed by a discretization of the continuous counterpart into linked individual elements, e.g. most commonly a triangle mesh. In the latter, a function is defined on a grid over the whole three-dimensional domain. The surface, then, is implicitly marked by a certain level-set (isocontour) of the function—usually the zero level-set—and must be reconstructed by methods like Marching Cubes [Lorensen and Cline 1987] into an explicit mesh when needed. Both approaches are the discretization of their continuous counterparts as computers are finite in their memory and calculation. Both enjoy a vast literature of research and neither has a clear-cut advantage over the other. One must choose between the two by evaluating the trade-off given each specific application.

1.1.1 Handling Topology Changes

As explicit and implicit surfaces are animated, the connectivity of surfaces undergo change: e.g. separate components of oil merge as they bump into each other floating on the water; and forces like surface tension and gravity cause thin spindles of water to split into droplets. We say these merge and split events are *topology changes* to the tracked surface since (some) points on the surface experience a sudden discrete change to their local neighborhood. When the physics part of an animation advects the mesh into invalid states, with no regards for overlapping regions or regions that are too thin, we say the mesh exhibits *topological defects*.

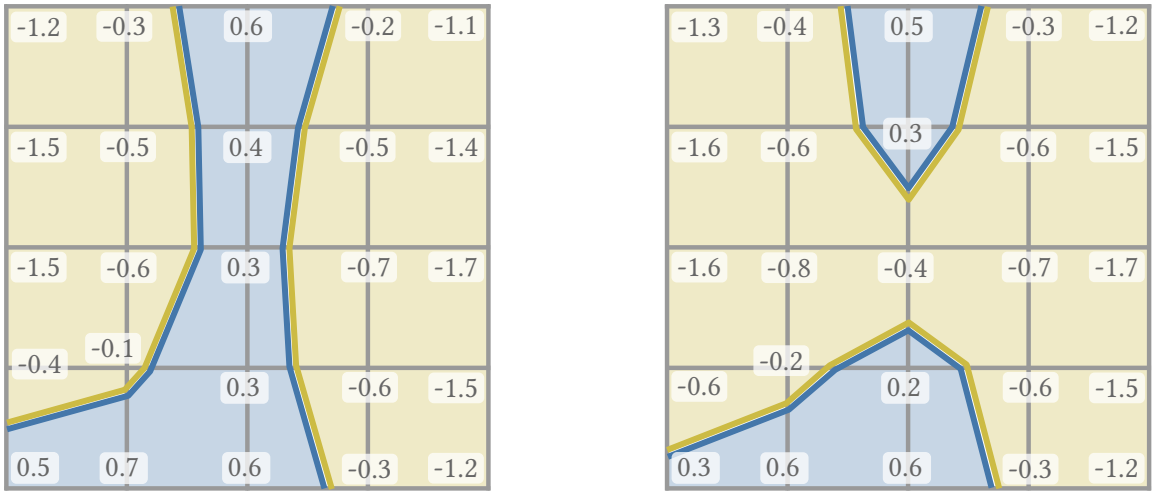
In addition to the more prominent merge and split events involving multiple mesh components, smaller defects can also occur. Neighborhoods can exhibit small-scale self-intersections. Furthermore, portions of the mesh can get inverted (inside-out). In a perhaps confusing way of using the mathematical terminology, inversions are also considered as topological defects. See Figure 1.1.



(a) Overlap (merge)

(b) Inversion

Figure 1.1: Examples of topological defects

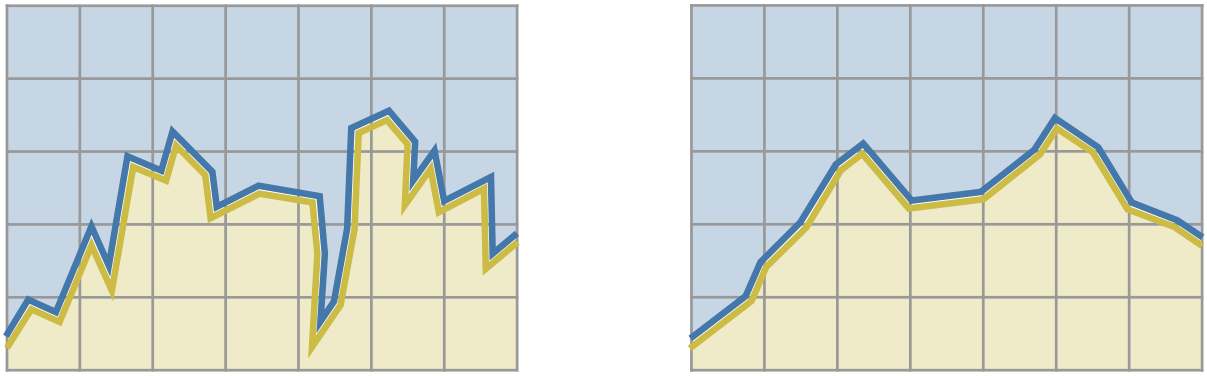


(a) Before (b) After

Figure 1.2: **Implicit methods handle topology changes intrinsically.** Implicit methods only work with function values on the grid. Negative values indicate inside the surface and positive values indicate outside. In this example, the sign of the grid vertex in the middle is flipped after some time step. Consequently, the explicit mesh that is later extracted shows modified topology, that the yellow material has merged with itself.

For explicit surfaces, these interruptions in the animation require careful surgery of the mesh—a notoriously difficult task. Triangles must be torn apart in the local neighborhoods where topological defects occur. The holes that are created, then, must be sewn back into a valid mesh in a way that reflects the desired topological change. Since triangle meshes can manifest arbitrarily peculiar holes, dents, ridges, and spikes, such mesh surgery is often complicated and prone to mistakes; the space of interactions between local neighborhoods of triangles is huge. Numerical errors in floating-point computation and degenerate configurations of input triangle meshes further hinder robustness.

Implicit methods, however, have the advantage that they can intrinsically handle topological changes in the background with no additional effort; the simulation only pays attention to the values everywhere on its grid and is ignorant of any changes to the topology of the desired level-set. Indeed, if topology changes take place, the explicit mesh that is later extracted for rendering will show modified topology (Figure 1.2). As a result, different components that collide with each other get merged, parts that get too thin—compared to the grid resolution—split apart, and the surface will not have any other topological defects like inversions *by construction*. Hence, implicit surfaces offer a comparatively effortless implementation in terms of robustness.



(a) Original (b) Reconstructed

Figure 1.3: **Loss of surface detail.** The original explicit mesh is reconstructed from samples on the grid which leads to loss of surface detail. Higher-resolution grids can only help to a certain degree and at the expense of significantly higher computational cost.

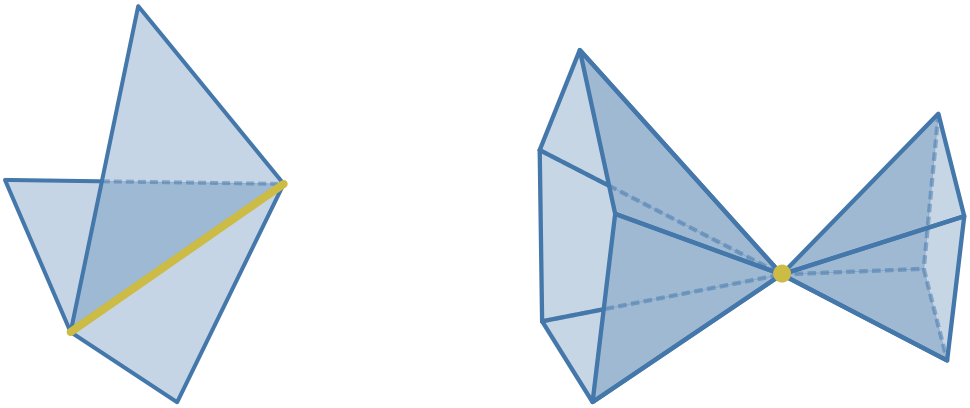
1.1.2 Preserving Surface Detail

Unlike handling topology changes, preserving surface detail is a task that explicit surfaces are naturally good at, since mesh surgery only alters the mesh in locations of topological defect.

Implicit surfaces, on the other hand, are the exact opposite. One single trip to the implicit domain and back causes an explicit mesh to lose all its features below the grid resolution (Figure 1.3). Perhaps even worse, the immunity of sharp features in larger scales is only short-lived. As the implicit surface is advected by modifying the values on the grid, larger features get smoothed out over time too. Therefore with implicit meshes surface detail is lost over time, creating visually distracting artifacts and damaging the perceived realism of the animation; e.g. loss of high-frequency waves and ripples on the surface of the water in a fluid simulation; or blurring of the mesh texture when animating viscoelastic material.

1.2 Manifold and Non-Manifold Surfaces

A surface is *manifold* if it is locally Euclidean everywhere—in other words, it is topologically the same as the open disk at every point. Likewise, a surface is *non-manifold* if there are points whose neighborhoods—no matter how small—do not resemble the open disk topologically. In the context of explicit triangle meshes, manifoldness implies that each edge is incident to exactly two triangles and the neighbors of each vertex form one single loop—unlike the edge and the vertex in



(a) Non-manifold edge (b) Non-manifold vertex

Figure 1.4: Non-manifold elements in explicit triangle meshes

Figure 1.4. For implicit meshes, it means the number of distinct materials present in every grid cell is at most two.

Most algorithms in geometry processing assume the input is manifold. Even when the additional complexity of non-manifoldness does not seriously hinder the generalization to non-manifold meshes, algorithms tend to assume manifoldness anyway for the sake of simplicity and convenience. However, non-manifold surfaces are essential to many real scenarios of physics that we wish to animate. For instance, in simulating multi-phase fluids that do not mix, e.g. oil, water, and air (also considered fluid in this context), the interface formed between distinct materials is non-manifold; there are regions where more than two distinct materials meet. Methods primarily designed with manifold surfaces in mind cannot proceed with such inputs—even if there are no theoretical obstacles which is often not the case.

Perhaps, soap bubbles are a more interesting case study. Soap bubbles that approach each other, form interfaces on contact rather than join volume. The interfaces in the whole cluster then constitute a non-manifold surface with numerous junctions where three or four materials meet (more is also possible, though physically unstable). Figure 1.5 shows a 2D analogue with 16 non-manifold vertices (edges cannot be non-manifold in 2D). Air bubbles submerged in water, on the other hand, can behave differently and have the tendency to merge into larger bubbles on contact.

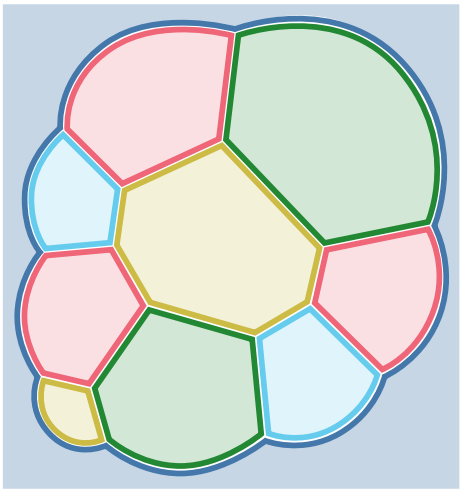


Figure 1.5: 2D cluster of bubbles

Hence, it is necessary to develop methods for surface tracking of non-manifold surfaces as well. There are both implicit and explicit approaches to representing and animating non-manifold surfaces. Regarding topology changes and preserving surface detail, it is the same story of opposing advantages and disadvantages: handling topology changes is natural to implicit surfaces, and difficult for explicit non-manifold meshes as the space of all possible ways in which advection can lead to topological defects is made even larger by non-manifoldness; and preserving surface detail is natural to explicit non-manifold meshes while implicit representations undesirably smooth out surface features.

1.3 Method Overview

As demonstrated, the two main approaches to surface tracking each leave something to be desired—either they suffer from robustness issues of explicit mesh surgery or degrade surface features over time. Is it possible to break out of this polarizing trade-off and combine the best of both worlds?

Indeed, implicit methods unnecessarily sacrifice surface detail everywhere in order to accommodate robust topological changes. Wojtan et al. [2009] devised a hybrid method to overcome this for surface tracking of manifolds:

1. Surfaces are represented explicitly to preserve detail.
2. Regions of topological defect are detected with the help of a corresponding implicit surface constructed on a background grid.
3. The explicit mesh is then clipped to the boundary of this region and the inside is completely removed.
4. The implicit mesh is reconstructed locally to fill this region while realizing the desired topological change.
5. The old and new parts are robustly stitched back together to form a valid mesh ready to be input to the next step of advection in the animation.

Contribution. The method we present in this thesis contributed to the generalization of this hybrid method to non-manifold surfaces which was published at ACM Siggraph 2024 under the title *"Multi-Material Mesh-Based Surface Tracking with Implicit Topology Changes"* [Heiss-Synak* and Kalinov* et al. 2024]. It follows in the

same general footsteps as its manifold inspiration, and solves additional challenges unique to non-manifold meshes. Specifically, the method we present in this thesis contributed to the part analogous to Step 4 above.

Outline. Chapter 2 briefly goes through the related work. Chapter 3 is dedicated to summarizing certain core parts of Wojtan et al. [2009] and Heiss-Synak* and Kalinov* et al. [2024], to lay the foundation for our method presented in Chapter 4. Chapter 5 provides results and qualitative analysis of our method. Finally, Chapter 6 concludes the thesis with discussions of limitation and future work.

CHAPTER 2

Related Work

2.1 Implicit Surface Tracking

Level-set methods as introduced by Osher and Sethian [1988] are the main way of tracking surfaces implicitly in computer graphics. They were originally designed for manifold surfaces separating the domain into inside/outside regions, wherein by convention, the implicit function takes negative/positive values, respectively [Osher and Fedkiw 2001]. As the function values on the Eulerian grid are advected through physically-based or geometric (e.g. normal and mean-curvature) flows, the zero level-set marking the target interface is updated implicitly.

Level-set methods were later extended to tracking the interface between multiple materials and advecting non-manifold surfaces for simulating multi-phase liquids [Losasso et al. 2006; Kim 2010] and clusters of bubbles [Kim et al. 2007; Zheng et al. 2009]. Although they offer robustness and are capable of handling topological changes with ease, implicit methods tend to smooth out important surface detail aggressively.

2.2 Converting between Domains

In practice, implicit methods still have to work a lot with explicit meshes; inputs to methods are more often than not explicit, and at the end of each time step, an explicit mesh must be extracted for visualization/rendering purposes. Therefore, algorithms are needed for converting to and from the implicit domain.

Signed Distance Functions (SDF) are usually the first choice for converting to the implicit domain [Osher et al. 2004]—the value of the implicit function at each grid

vertex is set to the signed distance to the explicit mesh with points inside being negative. It is necessary, then, to assume that input meshes are closed and orientable to begin with.

For converting back to the explicit domain, *Marching Cubes* [Lorensen and Cline 1987] is the most common and convenient method. It uses the sign of grid vertices to detect where the target level-set must be passing through and fills in those grid cells with a piecewise linear surface taken from predefined templates. In particular, given a grid edge having opposite signs on its vertices, linear interpolation is used to guess the location where the zero level-set crosses. Since local reconstructions inside individual cells agree on the cell boundaries, they seamlessly connect to each other with no additional processing needed to form the explicit mesh. Reitinger et al. [2005] extended the original Marching Cubes to work with multiple materials and reconstruct non-manifold surfaces. We will take a closer look at the non-manifold structure later on in Chapter 3.

2.3 Explicit Surface Tracking

Careful mesh surgery must be performed to handle topological changes when explicit meshes get entangled after advection. *El Topo* [Brochu and Bridson 2009b], however, adopted a different approach to this problem of entanglement: to never allow meshes to self-intersect in the first place and perform merging when elements come in sufficient proximity of each other. Starting from a topologically valid mesh, they ensure the advection proposed by the physics simulator causes no self-intersections: potential self-intersections are detected via Continuous Collision Detection (CCD) [Brochu and Bridson 2009a] and repulsion forces are applied accordingly to avoid collision. This results in a slightly perturbed velocity field compared to the proposed advection which ensures meshes remain in a topologically valid state. A *zippering* operation is then performed to merge portions of the mesh that are sufficiently close to each other: the pair of edges closest to each other are found; the four triangles incident to them are deleted; the two quadrilateral holes that are left are zippered up using a closed-form triangulation that realizes the desired merging event. The splitting operation, on the other hand, takes place less directly by duplicating and separating non-manifold vertices created by some mesh improvement operations that lead to degenerate cases–operations such as edge flips, edge splits, and edge collapses.

The same approach was extended to non-manifold meshes for tracking multi-material interfaces by *Los Topos* [Da et al. 2014]. Non-manifold meshes make the task more

complicated, requiring an extended set of operations for handling topology changes. Nonetheless, all operations—including those that lead to larger effects of merges and splits, in addition to operations for mesh quality improvement—are fine-grained local edits. Furthermore, all operations are atomic, meaning they are either fully executed or canceled in case they are not guaranteed to lead to valid non-self-intersecting states. This breaking down of global changes into a series of local atomic edits is a core advantage that this approach makes possible; it would be more challenging to fill large holes left after removing self-intersecting regions *at once*. Moreover, Los Topos improved upon the zippering operation with a more delicate local surgery called *snapping*, resulting in more accurate merges for colliding triangle meshes. All of this results in a mesh surgery that preserves surface detail perfectly, and is relatively efficient and robust in many scenarios including the soap film and bubble cluster animations of Ishida et al. [2020]—*though not in all.*

Another fully-explicit approach is *Fast Grid-Free Surface Tracking* by Chentanez et al. [2015]. First, intersecting triangles or triangles inside intersecting regions are identified and are consequently deleted. Then, for each matching pair of holes that remain, one pair of edges are connected across the hole by two triangles forming a quadrilateral. This replaces the pair of holes with one large hole. The core idea, then, is to fill in these holes by hole-filling algorithms. Specifically, dynamic programming is used to compute the optimal triangulation of the polygon bounding the hole with respect to objectives like minimizing surface area or sum of squared edge-lengths. This approach, though relatively fast, does not guarantee the resulting mesh is free of self-intersections and can have robustness issues in certain scenarios. Furthermore, it is limited to manifold surfaces and it is unclear how a generalization to non-manifolds would look like.

2.4 Hybrid Surface Tracking

Hybrid methods attempt to combine the advantages of both approaches: they represent the surface explicitly to preserve surface detail, and aim to repair entangled regions with meshes extracted from implicit representations.

Wojtan et al. [2009] and Müller [2009] both had the idea to impose a grid on the mesh and use Marching Cubes to repair the mesh. Müller [2009] perform the reconstruction everywhere, including regions far from topological changes. Some measures are implemented for preserving thin sheets narrower than the grid resolution and restoring some details of the original surface. Although, these measures are far from exhaustive

and surfaces do get noticeably smoothed out over time as the animation advances. Wojtan et al. [2009], on the other hand, carefully detect regions of topological defect and repair the surface only where needed, leaving original surface detail untouched elsewhere. This is confirmed by the Zalesak test [Zalesak 1979]: a disk with a notch in the middle is rotated around a point in the scene. In contrast to Müller [2009] who smooth out the sharp notch over time, Wojtan et al. [2009] manage to preserve it perfectly. It is worth mentioning that their approach gives rise to an additional challenge that must be solved robustly: newly reconstructed parts of the mesh need to be stitched back to the old parts. We will discuss their algorithm in more detail in Chapter 3.

Wojtan et al. [2010] then further built upon their previous method with support for thin features like sheets and strands which is crucial for realistic animations of fluids—especially water. Indeed, reconstruction by Marching Cubes deletes thin sheets created by, e.g. water splashing down onto a surface, leading to noticeable loss of volume in addition to the lack of realism in animation. Instead of marking a cell defective where the topology of the mesh is different than that of Marching Cubes templates', they mark it defective if it is not homeomorphic to a sphere. Furthermore, reconstruction with Marching Cubes is replaced by convex hulls, allowing for even more surface detail to be preserved inside grid cells.

Bo et al. [2011] developed a method similar to Wojtan et al. [2009]. They repair regions of topological defect with templates of Marching Cubes. However, for stitching new and old parts of the mesh, they take a different approach than subdivision at the boundary of the defected region: they delete all triangles even slightly overlapping the defected region, and then use hole-filling algorithms to bridge the gap in between. Though convenient, this leads to smoothing a larger-than-necessary portion of the mesh.

Heiss-Synak* and Kalinov* et al. [2024] were the first to devise a hybrid method for handling topological changes for non-manifold surfaces. Together with Los Topos, they are state-of-the-art in non-manifold surface tracking, offering different levels of efficiency and robustness in different scenarios. Generalizing the ideas in the manifold counterpart [Wojtan et al. 2009], their method provides the same general benefits: handling topological changes robustly while preserving surface detail as much as possible. They can handle arbitrarily large self-intersections which allows for a wider range of applications such as solid modeling of multi-material shapes compared to Los Topos who are constrained to starting from and maintaining a valid mesh—only

meaningful in the context of simulation. We will briefly go through the core parts of this method in Chapter 3 as preliminary to our method in Chapter 4.

CHAPTER 3

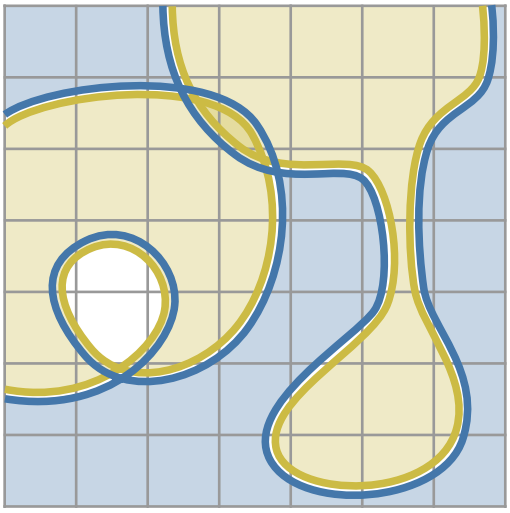
Explicit Surface Tracking with Implicit Topology Changes

This chapter provides an overview of the underlying context of our method: *explicit surface tracking with implicit topology changes*. First, we visit the work of Wojtan et al. [2009] who pioneered the idea for manifold meshes. Then, we summarize the generalization to non-manifold meshes [Heiss-Synak* and Kalinov* et al. 2024] without delving deep into the part that is the contribution of this thesis. That will be done in Chapter 4.

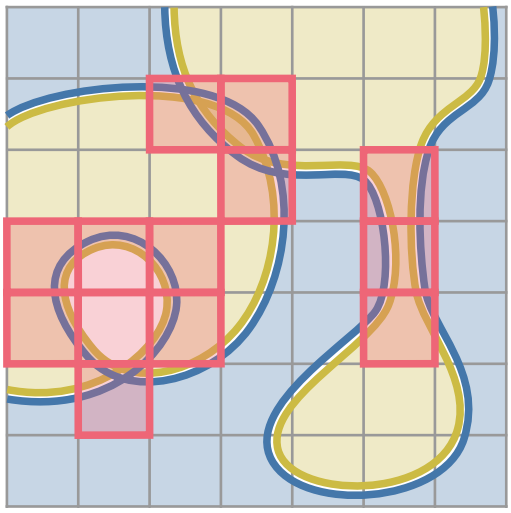
3.1 Manifold (Two-Material)

Wojtan et al. [2009] designed a hybrid method for tracking a closed *manifold* mesh M given as input. Their method can be coupled with any physics simulation—even implicit methods—as long as the final product of the simulation in each time step is an advection for the manifold Lagrangian mesh M. Accepting the proposed advection while keeping the same connectivity for M inherently preserves surface detail, but it can also introduce topological defects such as self-intersections and inversions (M getting inside-out in some parts) which must be repaired before the next time step. See method overview in Figure 3.1.

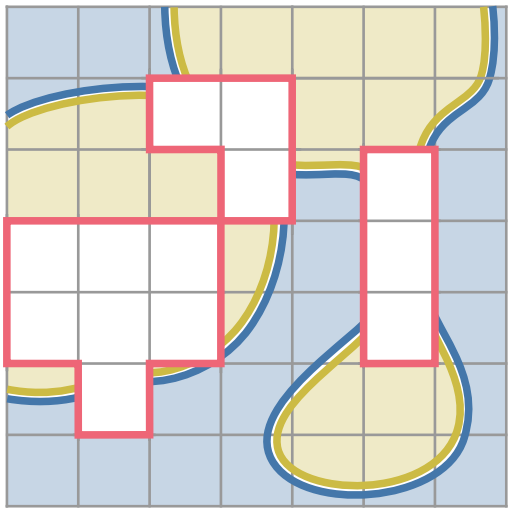
They take M to the implicit domain by sampling a Signed Distance Field D on a regular 3D grid of sufficient resolution: first, the (unsigned) distance to the closest triangle is computed at each grid vertex; then, distance values at grid vertices determined to be inside M are made negative. Converting D back to the explicit domain by Marching Cubes would result in a mesh that is, by construction, free of topological defects.



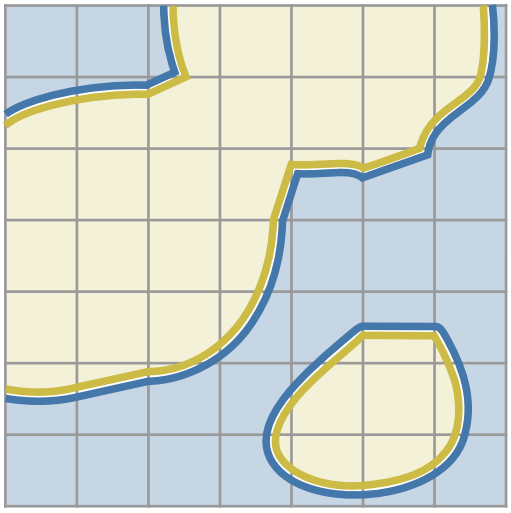
(a) Mesh with topological defects



(b) Detection of flawed cells



(c) Subdivision at the boundary and removal of the inside



(d) Repaired mesh

Figure 3.1: **Overview of Wojtan et al. [2009].** The input mesh has three separate topological problems that need fixing: merging, splitting, and inversion. Note the duality between merges and splits: the split of the yellow material to the right can be seen as merging of the blue as it comes within distance, smaller than the grid resolution, of itself.

Figure 3.2: Subtle bumps on the surface are also marked for reconstruction by the complex cell test: the edge in the middle is intersected three times. The deep cell test reverts the marking of such shallow cells to prevent smoothing out the detail.

However, in order to preserve surface detail on M, only entangled regions of M must be replaced by extractions from D. The first challenge that Wojtan et al. [2009] solve therefore is to carefully detect regions of topological defect.

They mark a grid cell as *complex* if the topology of M restricted to that cell is different than that suggested by Marching Cubes templates on D. For instance, Marching Cubes would never produce a mesh that crosses a grid edge more than once; therefore, grid cells where M crosses an edge more than once are marked as complex. Similarly, it would never produce a mesh that intersects a grid face in the shape of a closed loop, or a mesh that has a whole component entirely within one grid cell. Thus, cells are checked one by one and are marked as complex where fulfilling such criteria. Furthermore, cells inside or adjacent to regions of overlap are also marked as complex. This scheme without any further modification results in marking many important surface details for reconstruction, as subtle bumps on the surface far from regions of topological defect can trigger the complex cell test as well (Figure 3.2).

In order to mitigate this, an additional test is introduced as the *deep cell* test: a cell is *deep* if it is at least one cell away from the zero level-set of D. Of all cells marked as complex, those that are *shallow*—adjacent to the zero level-set—are unmarked. Then, those remaining are flood-filled outwards until the boundary of marked cells consists entirely of cube faces that are topologically simple. This flood-fill in effect restores the marking of shallow cells around important topologically flawed regions.

Triangles inside marked cells must be deleted and triangles outside must be preserved. However, some triangles overlap both inside and outside. In order to preserve as much surface detail as possible, those triangles overlapping both sides are carefully subdivided at the boundary of clusters of marked cells in a way that no triangle lies both inside and outside afterwards. Subsequently, the triangles inside are removed.

Next, Marching Cubes is used to reconstruct an explicit mesh from D in flawed regions.

The new triangles are then easily stitched back to M since the flood-fill of deep cells had ensured the grid faces on the boundary of flawed regions are topologically simple. By construction, the new mesh cannot have topological flaws; portions of the mesh that had collided are merged, portions of the mesh that had got too thin are split, and remaining artifacts like inversions are resolved as well. Furthermore, the mesh is closed and manifold again.

The portions of the mesh generated by Marching Cubes can be of low quality since the method draws from a predefined set of templates. The geometry is massively improved by using linear interpolation on distance values of D, as a more accurate guess for the placement of new mesh elements. However, one round of mesh improvement operations such as edge-splits, edge-collapses, edge-flips, and triangle-subdivisions are nevertheless needed to ensure good triangle quality. Finally, the mesh is ready to be input to the physics simulator as the simulation time is incremented once more.

3.2 Non-Manifold (Multi-Material)

The first challenge in the generalization to non-manifolds is to work out the specifics of the implicit representation. Indeed, manifold surfaces are relatively straightforward to work with, in that they partition the 3D domain into inside and outside regions, making it possible to assign to each point a *label* that takes on *binary* values. This allows the use of Signed Distance Functions to encode labels with signs; negative and positive signs indicate inside and outside the surface respectively; and the target surface can be conveniently extracted as the locus of transition from negatives to positives.

Non-manifold surfaces, however, allow the domain to be partitioned into multiple "insides" that are in contact with each other as illustrated in Figure 1.5. Heiss-Synak* and Kalinov* et al. [2024] assign an integer *label* to each distinct *material*, including the one global outside. When two closed *volumes* approach each other during the simulation, they form an interface on contact if they have different labels/materials, and otherwise merge into one as in the manifold work.

Heiss-Synak* and Kalinov* et al. [2024] use material vectors to indicate the label(s) present in each region. In *properly embedded* states of the mesh, each point in the domain (except those exactly on the mesh) lies in exactly one material and has a unique label, and therefore has a one-hot encoded material vector. One time step of the simulation, however, can violate this condition as self-intersections and inversions

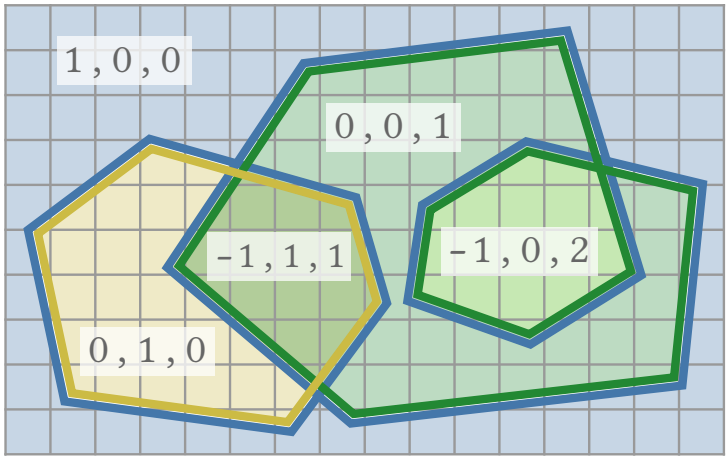


Figure 3.3: **Invalid material vectors.** Material vectors can be computed by starting from the outside and tracing a ray through the mesh. As the ray intersects with triangles (or rather edges in this 2D analogue), one material is exited from and another is entered in. The sum is invariant and always equal to one. Vectors that are not one-hot encoded indicate regions of topological defect.

occur, causing a point to be e.g. "twice" in one material and "minus one times" in another, which are implausible (Figure 3.3).

Grid vertices with invalid material vectors indicate regions that need to be repaired. Complex cell and deep cell tests analogous to those in the manifold work are run to detect the flawed region by comparing the topology of the mesh to those of the multi-material Marching Cubes templates. Then, mesh triangles are subdivided on the boundary of the flawed region so that no triangle lies both inside and outside the flawed region. Subsequently, the triangles inside are removed.

The holes that are created then must be filled with newly reconstructed triangles. This is the main task where the non-manifold generalization faces a remarkably more difficult challenge. The manifold counterpart could simply use inside/outside queries to check if a grid vertex is inside any of the overlapping meshes, since there is only one material, excluding the one global outside. The multi-material generalization, though, has to decide which of many materials present in an overlapping region, a grid vertex is *"more inside"* of, in an obscure open-ended way. The contribution of this thesis as fully detailed in Chapter 4 is to find a reasonable and generic solution for all applications concerned.

After determining correct labels for grid vertices in the flawed region, a multi-material generalization of Marching Cubes [Reitinger et al. 2005] is used to fill in individual grid cells. This multi-material version is more of a recipe for construction rather than templates covering all possible arrangements of labels; as up to eight distinct

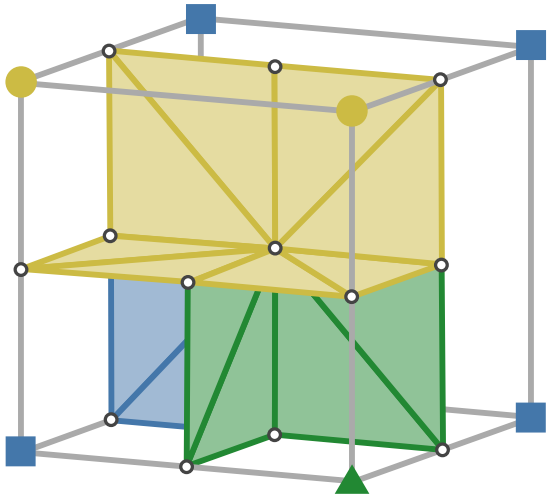


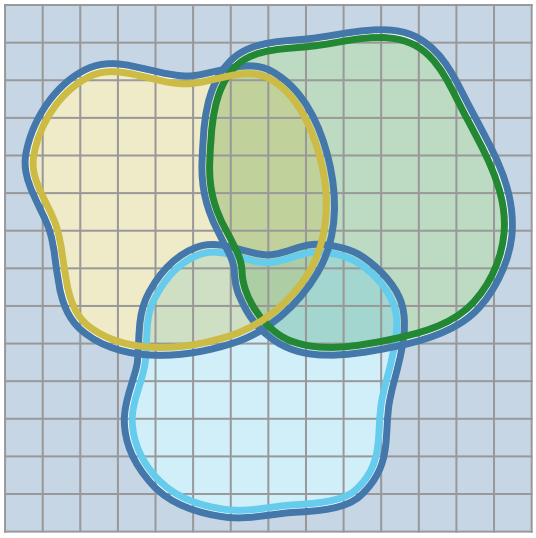
Figure 3.4: **Multi-material Marching Cubes.** Each pair of adjacent octants that have different labels are separated by a square made of two triangles. Triangle vertices are simply put at the center of grid edges, faces, and cells.

materials can coexist in a grid cell, the configuration space is too large for templates in lookup tables to be practical, even after eliminating equivalent cases by rotations and symmetries.

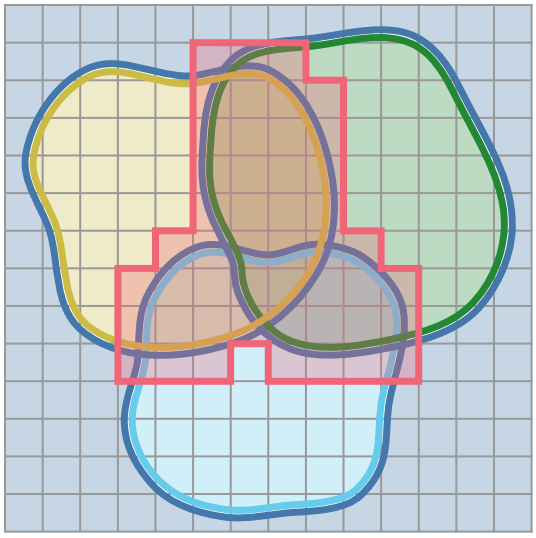
The first step is to ensure the correct topology. Each grid cell is cut to eight equal octants. Assuming each octant inherits the label of its corresponding grid vertex, each pair of adjacent octants that have different labels must be separated by the new mesh. As such, the algorithm is to check all twelve pairs of adjacent octants, and to simply create two triangles making the square interface in between, if they have different labels (see Figure 3.4). Adding all such squares ensures the correct topology and triangles in individual cells can be seamlessly connected to each other, since neighboring grid cells agree on the labels of their shared face.

So far the newly reconstructed mesh looks very jagged as the triangle vertices are placed exactly at the center of grid edges, faces, and cells. The next step then is to optimize the placement of these vertices which will be discussed in Chapter 4 as well. Subsequently, the newly reconstructed mesh is stitched back to the old one, and mesh improvement operations are performed to ensure good triangle quality.

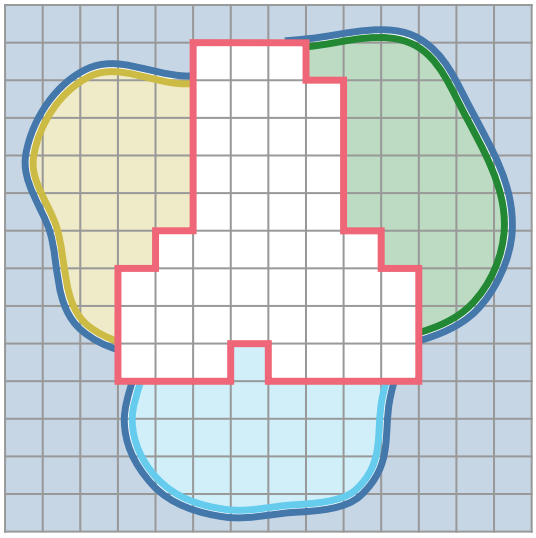
See Figure 3.5 for an overview of Heiss-Synak* and Kalinov* et al. [2024].



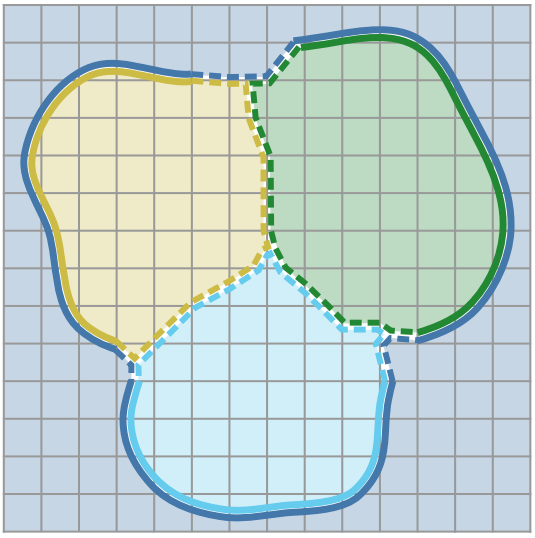
(a) Mesh with topological defects



(b) Detection of flawed cells



(c) Subdivision at the boundary and removal of the inside



(d) Repaired mesh

Figure 3.5: **Overview of Heiss-Synak* and Kalinov* et al. [2024].** Intermediate steps between (c) and (d) are detailed in Chapter 4.

CHAPTER 4

Method

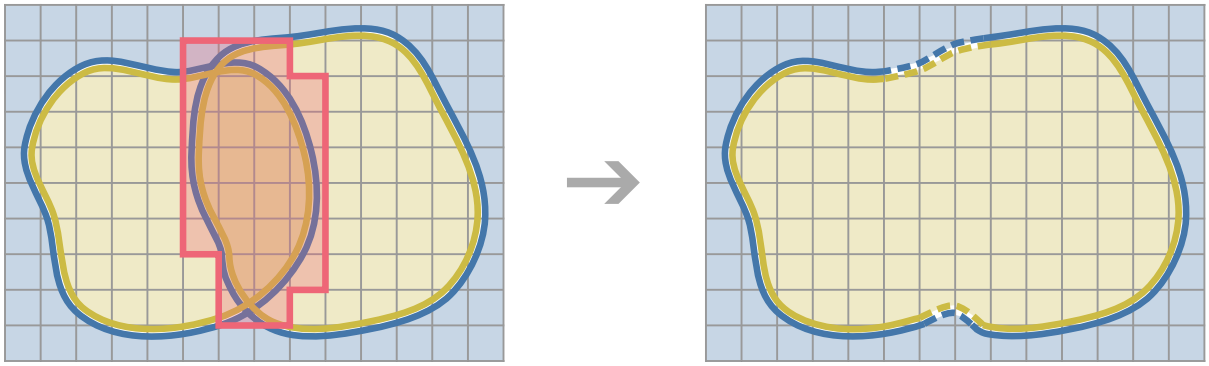
This chapter is dedicated to the contribution of this thesis to the method of Heiss-Synak* and Kalinov* et al. [2024]. We will describe the algorithm after going over the setup once more.

4.1 Problem Setup

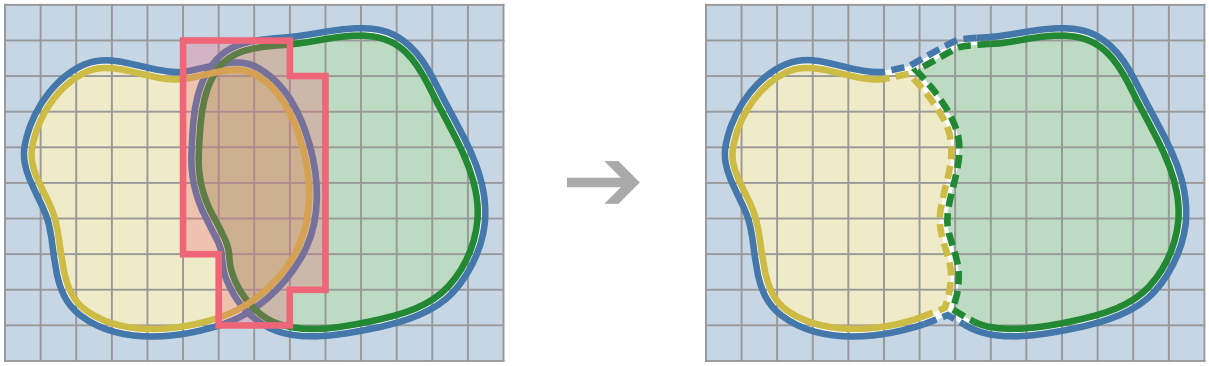
A non-manifold triangle mesh M and an overlaid grid are given. Certain clusters of grid cells are detected as regions that need to be reconstructed. The triangle mesh is subdivided at the boundary of those clusters and the inside is removed. The task is to fill in those holes that are left.

Let us first examine the simpler case of two volumes overlapping (Figure 4.1). The flawed region is the same for both manifold and non-manifold reconstructions. The manifold task is relatively straightforward: to reconstruct the *shallow* triangles that were inside marked cells. In fact, those triangles were only removed because of the limitations of the discrete grid; as the grid resolution approaches infinity, there is less and less surface to reconstruct. At the extreme, assuming the grid is continuous, there is no reconstruction to be done—*the manifold reconstruction is merely an attempt to restore a subset of the original mesh.*

Our non-manifold generalization, however, is burdened with additional non-trivial reconstruction: where exactly should the interface between distinct "inside" volumes pass through? Since the surface tracker has to be able to accommodate many applications, *this problem is ill-posed*: the solution has to be generic yet reasonable in an obscure open-ended way. Such circumstances promote solutions that are justifiable by



(a) Manifold



(b) Non-manifold

Figure 4.1: **Comparison of manifold and non-manifold reconstructions.** The flawed region is marked in red. Dashed lines are the reconstructed portions. In the non-manifold case, a new interface is drawn between the yellow and the green that is not hinted at or suggested by the original mesh.

some default yet precise measure of optimality: one could choose to fill the hole with the surface of minimal area or that of minimal bending energy. We choose a solution that draws interfaces that partition the flawed region based on closest points.

4.2 Algorithm

The interface separating two volumes can be thought of as the set of points that belong to both—points that are in some interpretation of the word, *equidistant* to both. Taking this quite literally, our idea is to locate the points that are of equal *Euclidean* distance to two or more materials in flawed regions. Unlike points inside flawed regions, points on the boundary have correct (unique) labels and are therefore trustworthy. Consequently, each point inside can be assigned to the same label as the boundary point that is closest to it.

4.2.1 Flood-Fill

As such, the first part of the algorithm is to find the closest point on the boundary, for each grid vertex inside a flawed region. This can be done efficiently by the Fast Marching Method (FMM) [Sethian 1999]. For each label present on the boundary, distance values are initialized to zero at the corresponding boundary vertices (Figure 4.2a), and are propagated—or *flood-filled*—inwards. Grid vertices are then assigned corrected labels depending on which flood-fill reaches them first (Figure 4.2b).

FMM in this case works similar to the well-known algorithm of Dijkstra for computing shortest paths on a graph with positive weights. Only, it considers the Euclidean geometry of nodes when updating distances. At any moment, there is a frontier of nodes for which the distances have been fully determined. These nodes suggest distances for their immediate neighbors, kept in a min-heap. Each round, the smallest of all suggestions is declared true and the corresponding node joins the frontier, adding/updating suggestions for its undetermined neighbors. The flood-fill is over once there is no undetermined neighbor, once the min-heap is empty.

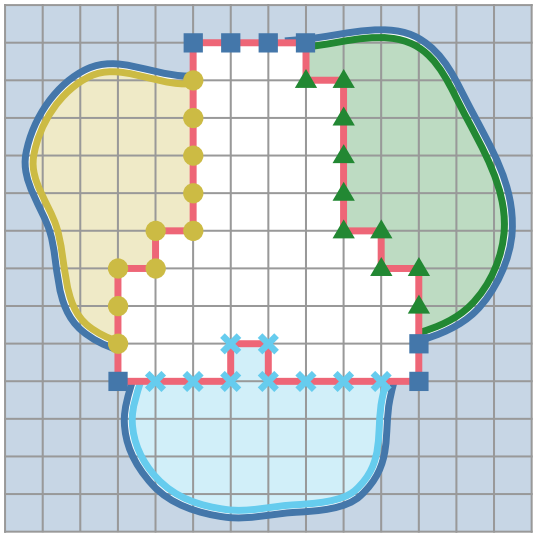
FMM is effectively computing an unsigned distance function u for grid vertices in the flawed region. The gradient of this function has a magnitude of one (almost) everywhere; and when a node is updated by its neighbors, the distance always grows larger. On this basis, the magnitude of the gradient at any grid vertex g_{ijk} (i, j, and k are indices along each axis in 3D) is discretized approximately as

$$\|(\nabla u)_{ijk}\|_2 = \begin{bmatrix} \max(D^{-x}_{ijk}u, -D^{+x}_{ijk}u, 0)^2 \\ + \max(D^{-y}_{ijk}u, -D^{+y}_{ijk}u, 0)^2 \\ + \max(D^{-z}_{ijk}u, -D^{+z}_{ijk}u, 0)^2 \end{bmatrix}^{1/2} \tag{4.1}$$

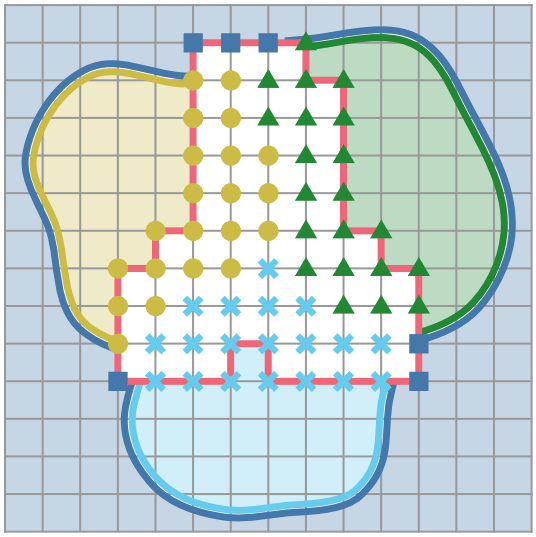
where D^{+}_{ijk} and D^{-}_{ijk} are forward and backward finite differences of partial derivatives respectively [Sethian 1998]. For instance:

$$D^{+x}_{ijk}u = \frac{u_{(i+1)jk} - u_{ijk}}{h} \quad \text{and} \quad D^{-x}_{ijk}u = \frac{u_{ijk} - u_{(i-1)jk}}{h} \tag{4.2}$$

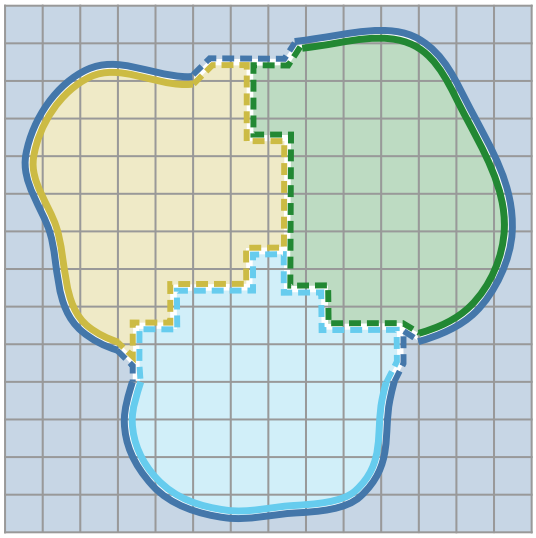
where h is the length of one grid cell. When the distance of an undetermined vertex is to be updated by its neighbor(s), Equation 4.1 is solved for the u_{ijk} that results in $\|(\nabla u)_{ijk}\|_2 = 1$. Terms in Equation 4.1 are activated/deactivated depending on how many neighbors of u have been determined previously. In particular, the first time a vertex g_{ijk} is updated, say from the neighbor $g_{(i-1)jk}$ to its left, its value will be crudely suggested as $u_{ijk} = u_{(i-1)jk} + h$, since only one of six finite differences is



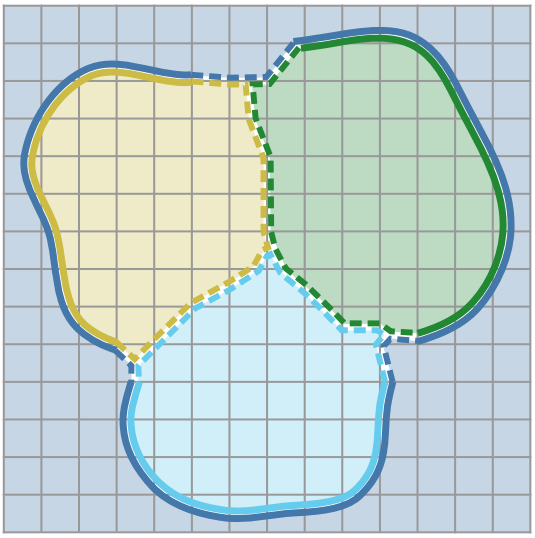
(a) Initial frontiers from trustworthy labels on the boundary



(b) Labels corrected after the flood-fills



(c) Naïve Marching Cubes reconstruction (see Chapter 3)



(d) Optimizing vertex placement

Figure 4.2: **Algorithm overview.** The flood-fills and vertex optimization reconstruct a fairly accurate Voronoi diagram for labels on the boundary, which is then used to fill the hole.

yet active. Suggestions for a vertex become smaller—and more accurate—as more neighbors become determined and therefore participate in the equation.

Note that there is a separate flood-fill for each material present on the boundary of the flawed region. For the sake of efficiency, they are not run past each other as they meet and their flows do not overlap; it is unnecessary to compute the distance of every grid vertex to every material in the flawed region. Furthermore, they share the same min-heap data structure.

4.2.2 Optimizing Vertex Placement

Once the flood-fills cover the whole flawed region and grid vertices have correct labels, the multi-material Marching Cubes (Chapter 3) is used to reconstruct a non-manifold mesh that fills the hole. New triangles are stitched back to the old mesh at the boundary of the flawed region and all topological flaws are resolved. However, the new portion of the mesh exhibits an undesirable staircase effect, since the vertices of new triangles are put exactly at the center of grid edges, faces, and cubes (Figure 4.2c). Afterthought mesh improvements can help smooth the jagged appearance, though only to a certain degree.

For Marching Cubes vertices on *grid edges*, a simple interpolation technique can be used to dramatically improve mesh quality. Assume p_1p_2 is a grid edge where the new mesh crosses. Assume $l_1 \neq l_2$ are the corrected labels for grid vertices p_1 and p_2, respectively. Let $d_{i,j}$ note the distance of p_i to l_j ($1 \leq i, j \leq 2$). Allowing the separate flood-fills to run past each other for a narrow band of one grid length, we can have all four pair-wise distances computed. Let $p_* = p_1 + x \cdot (p_2 - p_1)$ interpolate edge p_1p_2 where $0 \leq x \leq 1$. Its distance to l_1 can be linearly interpolated as $d_{*,1} = d_{1,1} + x \cdot (d_{2,1} - d_{1,1})$. Similarly, $d_{*,2} = d_{1,2} + x \cdot (d_{2,2} - d_{1,2})$. Hence, a more accurate guess as the placement of the new triangle vertex p_* on grid edge p_1p_2 can be computed as

$$d_{*,1} = d_{*,2} \tag{4.3}$$

$$\implies \quad d_{1,1} + x \cdot (d_{2,1} - d_{1,1}) = d_{1,2} + x \cdot (d_{2,2} - d_{1,2}) \tag{4.4}$$

$$\implies \quad x = \frac{(d_{1,2} - d_{1,1})}{(d_{1,2} - d_{1,1}) + (d_{2,1} - d_{2,2})} \tag{4.5}$$

Regarding Marching Cubes vertices on *grid faces* and in *grid cubes*, the distance of a point to a material can be similarly computed with the help of bilinear and trilinear interpolations, respectively. However, finding the exact point that is equidistant to *three or more* materials is difficult in practice. And in case there are only two materials present in a grid face or cube, the problem is undetermined and the answer is not unique (unlike the previous case for grid edges). Consequently, we adopt a different scheme. Let

$$d_{*,j} = (1-x)(1-y) \cdot d_{1,j} + x(1-y) \cdot d_{2,j} + (1-x)y \cdot d_{3,j} + xy \cdot d_{4,j} \tag{4.6}$$

be the bilinearly interpolated distance of a point p_* on a grid face $p_1p_2p_3p_4$ to label l_j, where $d_{i,j}$ is the distance of vertex p_i to label l_j as computed by the flood-fill. Instead

of finding the p_* where all $d_{*,j}$'s are equal for corresponding l_j's present on the face, one can find the p_* that minimizes $\sum_j (d_{*,j})^2$ where the sum is over all corresponding labels l_j that are present on the face. This can be done, for example, by gradient descent as the objective is quadratic in x and in y, and calculating partial derivatives in closed-form is straightforward. Marching Cubes vertices inside grid cubes can be optimized in a similar fashion; only, $d_{*,j}$ is calculated by trilinear interpolation, of values from all eight vertices of the grid cube.

However, we preferred to take averages instead of the optimization by gradient descent, as it was simple to implement and the results were satisfactory: Marching Cubes vertices on grid faces can be placed at the average of (up to four) vertices on edges of their face; similarly, vertices in grid cubes can be set to the average of (up to six) vertices on faces of their cube. Figure 4.2d shows the resulting reconstruction which is still of high accuracy since the linear interpolation for vertices on grid edges is the main contributing factor; and taking averages ensures better triangle quality [Taubin 1995].

4.2.3 Computational Complexity

Assume there are n grid vertices in flawed regions. The graph of these vertices is sparse as each vertex has at most four neighbors. Thus, $\mathcal{O}(n)$ many vertices are pushed into and popped from the min-heap during the flood-fills. Since each operation of the min-heap takes $\mathcal{O}(\log n)$ long, the flood-fills together take $\mathcal{O}(n \log n)$ long. Since separate flood-fills are stopped when they meet each other, the complexity is not multiplied by the number of distinct labels present in a flawed region, and is therefore independent of it. Furthermore, the optimization of vertices takes $\mathcal{O}(n)$ long.

CHAPTER 5

Results

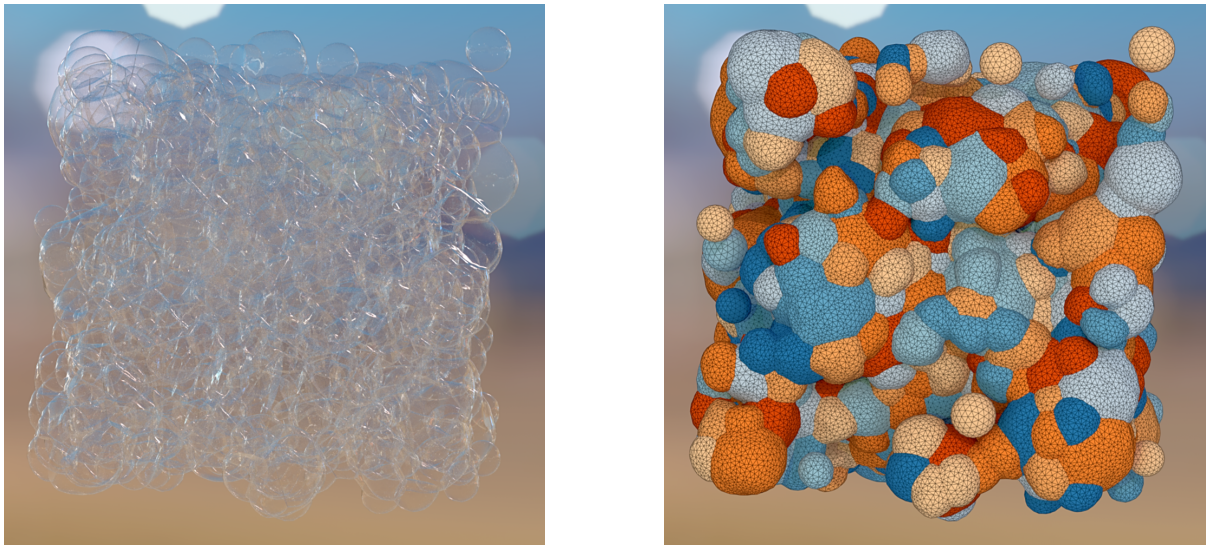
This chapter presents some results, analysis, and comparison of our method.

Acknowledgement. All results are produced with the official code[1] of Heiss-Synak* and Kalinov* et al. [2024]. The paper improves upon what we discussed in Chapter 4 in two ways. First, they introduce *breaking points*: the exact points of crossing to the flawed regions on grid edges. Breaking points provide a slightly more accurate initial frontier for the flood-fill. Second, a different Fast Marching Method is incorporated that keeps track of closest (breaking) points themselves, rather than propagating distance values. This is more elegant from an implementation point of view but has the same computational complexity: only one flood-fill is needed instead of multiple non-overlapping flood-fills.

5.1 Applications

Remember that the method of Heiss-Synak* and Kalinov* et al. [2024] can be coupled with any physics simulation that advects an explicit (Lagrangian) mesh in each time step. Many physics simulations involve non-manifold elements in their meshes, though perhaps none more prominently than the simulation of soap bubbles. Figure 5.1 shows an animation of Heiss-Synak* and Kalinov* et al. [2024] where the surface tracker is coupled with the soap film evolution method of Ishida et al. [2017]; for every step of physics advection, there is one step of mesh repair. As bubbles merge with each other and form larger clusters, they create non-manifold junctions. Over time, some bubbles pop and others slide over to take their place. The surface evolves

[1] https://git.ista.ac.at/psynak/superdupertopofixer



(a) Photorealistic rendering

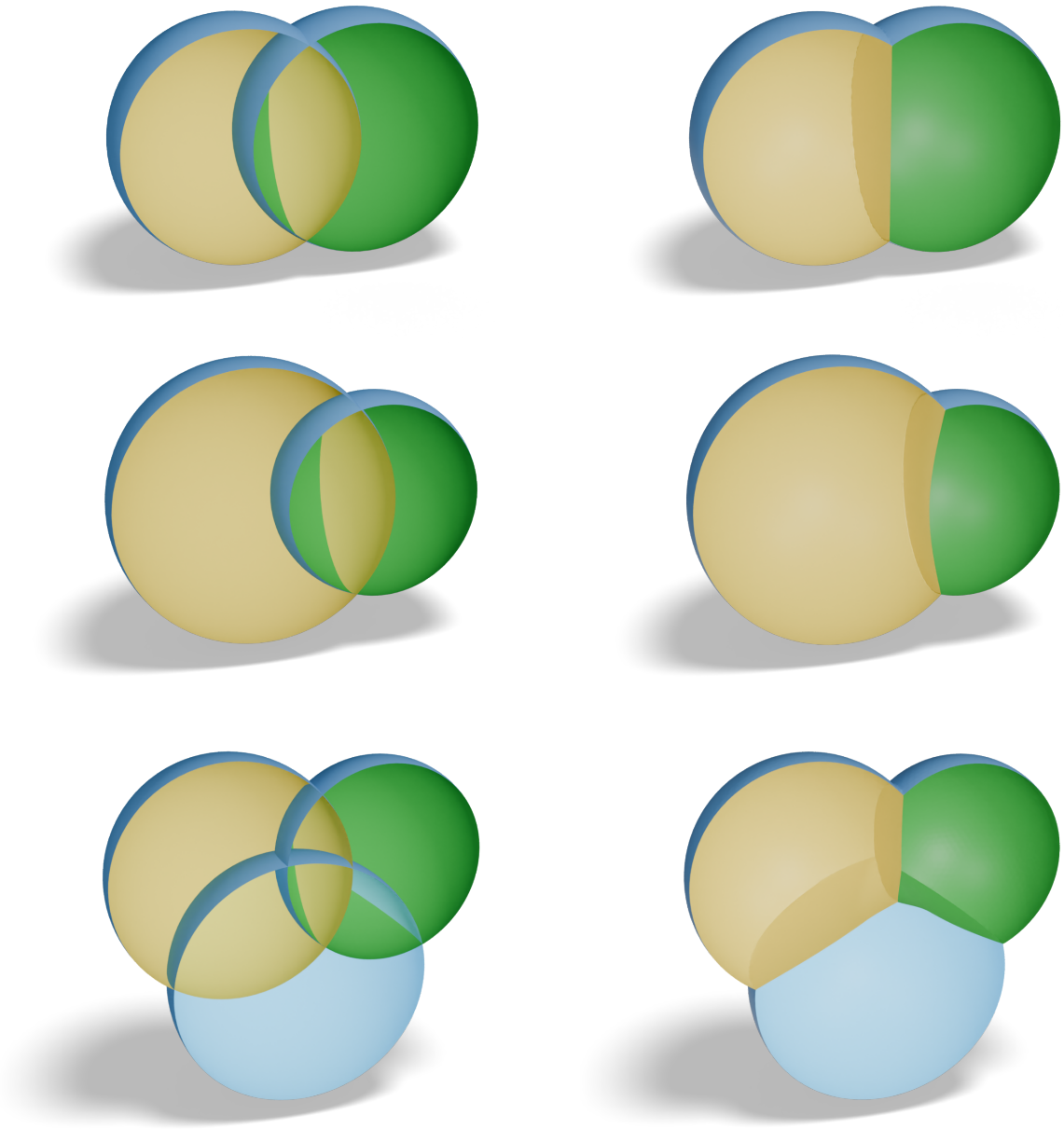
(b) Wireframe rendering

Figure 5.1: **Soap film simulation.** Robustness and efficiency of the surface tracker allow for a complicated simulation with 1000 initial bubbles. The wireframe rendering shows mesh triangles, with each manifold patch colored separately. This simulation is the largest and most complex foam simulation of its kind ever computed. Images are taken from Heiss-Synak* and Kalinov* et al. [2024].

from one complicated non-manifold mesh to another, all the while trying to minimize surface area.

One advantage of their approach is that the simulation can get more efficient by taking larger time steps than usual—unlike Los Topos who restrict the time step as much as needed since their approach needs to prevent the mesh from entering self-intersecting states at all times. Nonetheless, time steps for most physics simulations are still relatively minuscule. Otherwise, the physics computations will be too inaccurate; or worse, the simulation will blow up. This implies, for instance, that in this simulation of bubbles, regions of overlap between two merging bubbles are very thin no matter how wide—thin enough that most heuristics that limit the new interface to the flawed region, would reconstruct nearly the same interface that cuts through the middle of the region; and therefore all heuristics would introduce comparable amounts of error.

Even so, it is important to make sure our method reconstructs something reasonable in the case of larger overlaps as well. Figure 5.2 shows the result of our method for spheres sharing large overlaps. The first two rows show two spheres. When they are of the same size, the interface in between is completely symmetric and planar as one would expect. In case one is smaller, the interface takes a slight bend towards the larger sphere. This happens since our algorithm aims to reconstruct the exact bisector of the arched flawed region. The last row shows three overlapping spheres.

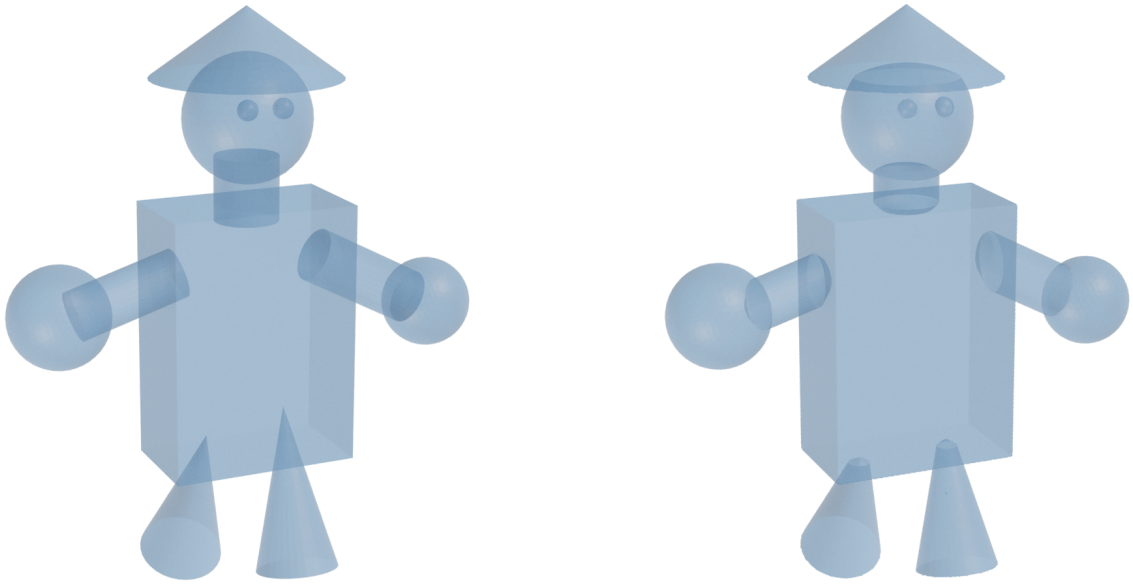


(a) Flawed (b) Repaired

Figure 5.2: **Cutaway view of overlapping spheres.** Our method (as part of Heiss-Synak* and Kalinov* et al. [2024]) reconstructs the interfaces in between. The grids used for all three examples were of resolution 200^3.

The reconstructed interfaces are smooth and connect to each other at a non-manifold triple junction in the middle.

Since the method allows for arbitrarily large overlaps, solid modeling of non-manifold shapes is another important application. Artists who design 3D models can use our method: first, they would create a rough outline by arranging manifold primitive shapes—like spheres, cylinders, cones, and cubes—that have overlaps; next, they assign different materials to primitives that should be separated by an interface; afterwards,



(a) Disjoint overlapping primitives

(b) Resolved non-manifold mesh

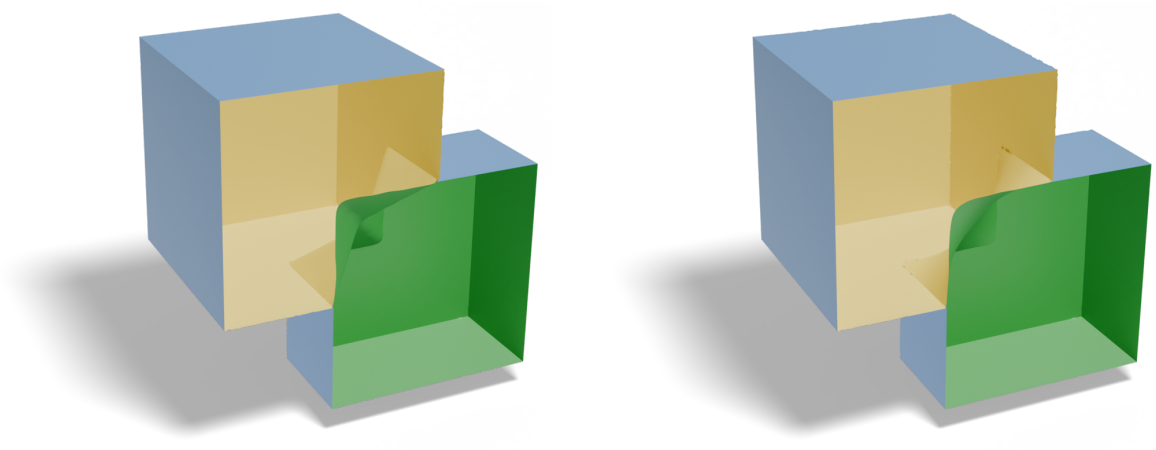
Figure 5.3: **Solid modeling of non-manifold shapes.** Note the reconstructed interfaces. The grid used for this example was of resolution 300^3.

our method approximates a multi-material boolean union of the primitives; after obtaining the intended topology of the model this way, artists can continue to sculpt the model further with the usual tools of 3D modeling software. See Figure 5.3 for an example.

5.2 Comparison

An alternative to our reconstruction algorithm is the minimal surface: the surface that takes the smallest surface area possible among all surfaces that separate distinct labels on the boundary of the flawed region. In many cases there is no significant difference between the minimal surface and ours. In some cases, our reconstruction is noticeably less smooth and piecewise flat. Figure 5.4 shows one such example. However, ours can be superior overall since we guarantee the new surface lies within the flawed region and does not intersect any remaining part of the original mesh. This is not true for the minimal surface, at least in instances where the flawed region might be concave.

Remember that the method of Heiss-Synak* and Kalinov* et al. [2024] is a generalization to non-manifold meshes. Yet, it still can and should be used for resolving topological problems of manifold meshes as well (when there are only two materials available as in the setting of Wojtan et al. [2009]). In that case, our method behaves



(a) Ours (b) The minimal surface

Figure 5.4: **Comparison with the minimal surface.** This is the cutaway view of an example where the minimal surface might be preferred. Two cubes of equal size share an overlap at their corners. The overlapping region is also a cube, but half their length. The interface our method reconstructs is piecewise flat and it looks to consist of six large triangles. The minimal surface, on the other hand, is a completely smooth saddle shape. The grids used were of resolution 100^3.

slightly differently, in that we correct labels by propagating *trustworthy* labels on the boundary of the flawed region to the grid vertices inside. The two-material setting, however, allows for a much simpler label correction scheme that Wojtan et al. [2009] implemented: using old distance values to the discarded mesh in flawed regions to reconstruct the new interface. This can lead to artifacts in the reconstructed surface. Ours is conservative and therefore avoids those artifacts.

CHAPTER 6

Conclusion

The method in this thesis is a contribution to the hybrid non-manifold surface tracker of Heiss-Synak* and Kalinov* et al. [2024]. Our reconstruction method fills in those holes that are created by removing the mesh in topologically flawed regions.

While our method is reasonable in this general-purpose setting, it is ignorant of the specific requirements of applications. For instance, some physics applications might prefer a solution that respects constraints such as volume preservation when repairing the mesh. Perhaps, a more elegant heuristic can intrinsically satisfy such requirements—at least partially—and therefore introduce less error into the simulation.

Furthermore, the surface tracker of Heiss-Synak* and Kalinov* et al. [2024] does not support thin sheets and strands. Future improvement in this regard must make changes to our part of the algorithm as well.

Finally, note that our reconstruction method lies in this context of fixing topological problems for surface trackers. However, the general problem of hole-filling for non-manifold meshes is indeed interesting on its own. For instance, what are some grid-free approaches to hole-filling for non-manifold meshes?

Bibliography

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 Journal on Scientific Computing*, 33(4):1505–1524.

Tyson Brochu and Robert Bridson. 2009a. Numerically robust continuous collision detection for dynamic explicit surfaces.

Tyson Brochu and Robert Bridson. 2009b. Robust topological operations for dynamic explicit surfaces. *SIAM Journal on Scientific Computing*, 31(4):2472–2493.

Nuttapong Chentanez, Matthias Müller, Miles Macklin, and Tae-Yong Kim. 2015. Fast grid-free surface tracking. *ACM Trans. Graph.*, 34(4).

Fang Da, Christopher Batty, and Eitan Grinspun. 2014. Multimaterial mesh-based surface tracking. *ACM Trans. Graph.*, 33(4).

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).

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).

Sadashige Ishida, Masafumi Yamamoto, Ryoichi Ando, and Toshiya Hachisuka. 2017. A hyperbolic geometric flow for evolving films and foams. *ACM Trans. Graph.*, 36(6).

Byungmoon Kim. 2010. Multi-phase fluid simulations using regional level sets. *ACM Trans. Graph.*, 29(6).

Byungmoon Kim, Yingjie Liu, Ignacio Llamas, Xiangmin Jiao, and Jarek Rossignac. 2007. Simulation of bubbles in foam with the volume control method. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, page 98–es, New York, NY, USA. Association for Computing Machinery.

William E. Lorensen and Harvey E. Cline. 1987. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169.

Frank Losasso, Tamar Shinar, Andrew Selle, and Ronald Fedkiw. 2006. Multiple interacting liquids. *ACM Trans. Graph.*, 25(3):812–819.

Matthias Müller. 2009. Fast and robust tracking of fluid surfaces. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, page 237–245, New York, NY, USA. Association for Computing Machinery.

Stanley Osher, Ronald Fedkiw, and K Piechor. 2004. Level set methods and dynamic implicit surfaces. *Appl. Mech. Rev.*, 57(3):B15–B15.

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

Stanley Osher and James A Sethian. 1988. Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations. *Journal of Computational Physics*, 79(1):12–49.

Bernhard Reitinger, Alexander Bornik, and Reinhard Beichel. 2005. Constructing smooth non-manifold meshes of multi-labeled volumetric datasets. In *Proc. of WSCG 2005*, pages 227–234.

J. A. Sethian. 1998. von karman institute lecture series. *Computational Fluid Mechanics*.

J. A. Sethian. 1999. Fast marching methods. *SIAM Review*, 41(2):199–235.

Gabriel Taubin. 1995. A signal processing approach to fair surface design. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 351–358.

Chris Wojtan, Nils Thürey, Markus Gross, and Greg Turk. 2009. Deforming meshes that split and merge. *ACM Trans. Graph.*, 28(3).

Chris Wojtan, Nils Thürey, Markus Gross, and Greg Turk. 2010. Physics-inspired topology changes for thin fluid features. In *ACM SIGGRAPH 2010 Papers*, SIGGRAPH '10, New York, NY, USA. Association for Computing Machinery.

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

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