DOI: 10.1111/cgf.14986

COMPUTER GRAPHICS *forum*
Volume 43 (2024), number 1, e14986

A Robust Grid-Based Meshing Algorithm for Embedding Self-Intersecting Surfaces

S. Gagniere,[1] Y. Han,[1] Y. Chen,[1] D. Hyde,[2] A. Marquez-Razon,[2] J. Teran[3] and R. Fedkiw[4]

[1]University of California Los Angeles, Los Angeles, California, USA
sgagniere@alumni.ucla.edu, yushanh1@math.ucla.edu, chenyizhou@ucla.edu
[2]Vanderbilt University, Nashville, Tennessee, USA
{david.hyde.1, alan.marquez-razon}@vanderbilt.edu
[3]University of California Davis, Davis, California, USA
jteran@math.ucdavis.edu
[4]Stanford University, Stanford, California, USA
fedkiw@cs.stanford.edu

Abstract
The creation of a volumetric mesh representing the interior of an input polygonal mesh is a common requirement in graphics and computational mechanics applications. Most mesh creation techniques assume that the input surface is not self-intersecting. However, due to numerical and/or user error, input surfaces are commonly self-intersecting to some degree. The removal of self-intersection is a burdensome task that complicates workflow and generally slows down the process of creating simulation-ready digital assets. We present a method for the creation of a volumetric embedding hexahedron mesh from a self-intersecting input triangle mesh. Our method is designed for efficiency by minimizing use of computationally expensive exact/adaptive precision arithmetic. Although our approach allows for nearly no limit on the degree of self-intersection in the input surface, our focus is on efficiency in the most common case: many minimal self-intersections. The embedding hexahedron mesh is created from a uniform background grid and consists of hexahedron elements that are geometrical copies of grid cells. Multiple copies of a single grid cell are used to resolve regions of self-intersection/overlap. Lastly, we develop a novel topology-aware embedding mesh coarsening technique to allow for user-specified mesh resolution as well as a topology-aware tetrahedralization of the hexahedron mesh.

Keywords: mesh generation, modelling

CCS Concepts: • Computing methodologies → Computer graphics; • Mathematics of computing → Mesh generation

1. Introduction

In many computer graphics and computational mechanics applications, it is necessary to create a volumetric mesh associated with the interior of an input polygonal surface mesh. Most commonly, a volumetric tetrahedron mesh is created whose boundary coincides topologically and/or geometrically with an input triangle mesh [MBTF03, LS07, HZG*18, Si15]. A volumetric embedding mesh that contains the input surface but whose boundary is different than an input triangle mesh is also commonly used [SDF07, TBFL19, KBT17, TSB*05]. It is generally required that the surface mesh be closed and orientable. It is also generally required that the surface mesh is free of self-intersection or overlap. While the closed and orientable requirements are relatively easy to satisfy in practice, the self-intersection constraint is more challenging, particularly near regions of high-curvature. In many computer graphics applications, this constraint can be violated without any artifacts since the overlap regions are not visible, however most volumetric mesh creation techniques either break down or give numerically 'glued' meshes if the constraint is violated. Even intersection free, but nearly intersecting meshes can cause problems for many volumetric mesh creation techniques.

While many surface geometry creation techniques address the importance of its prevention [HPSZ11, FTS06, Att10, ACWK06, GD01], as noted in, *e.g.* Refs. [SJP*13, LB18], self-intersecting surface meshes are common in practice. Often those involved in the surface geometry creation process are not involved in volumetric

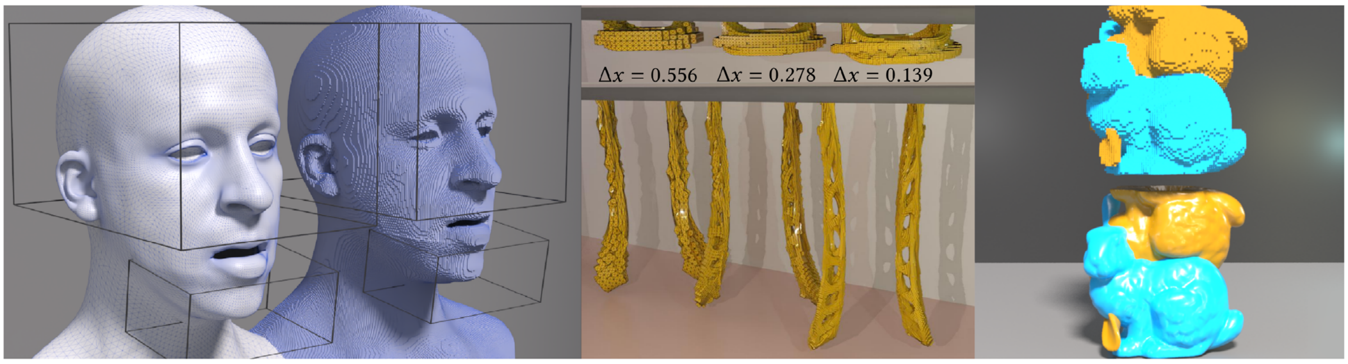


Figure 1: *(Left) Our method can generate a consistent volumetric mesh for a facial geometry that contains self-intersections,* e.g. *around the lips. (Middle) Two interlocking Möbius-strip-like bands separate freely at various spatial resolutions of the background grid, despite many near self-intersections in the surface geometry. (Right) Two bunny geometries can naturally separate despite significant initial overlaps.*

simulation or similar down-stream portions of the production pipeline and introduction of self-intersecting regions arises from a lack of communication. Furthermore, completely removing all regions of self-intersection is often deemed not worthy of the effort since it can significantly increase modelling time. In some cases, it is even desirable to have an overlapping input surface. *E.g.* it is desirable to have overlapping lips in the neutral pose of a deformable volumetric face mesh since lips resting in non-overlapping contact are not in a stress free state [CBE*15, CBF16]. It should be noted that although in practice, a non-negligible number of slightly overlapping or nearly overlapping regions are common (see Figure 1 left), generally the intersection-free constraint is not violated to an extreme degree with overlap regions typically having minimal volume.

Various approaches have developed volumetric mesh creation techniques specifically designed to be robust to self-intersecting [SJP*13, LB18] or nearly self-intersecting [TSB*05, LB18] input surfaces. Sacht *et al.* [SJP*13] use conformalized mean-curvature flow (cMCF) to first evolve the surface to a self-intersection-free state from which the flow is reversed, attracting the surface to its original, self-intersecting state but with a collision prevention safeguard. This defines an intersection free counterpart to the original input surface which can be meshed with standard techniques. Li and Barbič [LB18] create embedding tetrahedron meshes from unmodified surface meshes with self-intersection by computing locally injective immersions that can be used to unambiguously duplicate embedded mesh regions near overlaps. They sew these duplicated regions together using a technique inspired by the Constructive Solid Geometry (CSG) approaches in Refs. [TSB*05, SDF07] but with reduced use of expensive exact precision arithmetic. Teran *et al.* [TSB*05] use an element duplication/sewing technique to create embedding tetrahedron meshes for nearly intersecting input surfaces meshes.

We design an approach for the construction of a uniform-grid-based embedding hexahedron mesh counterpart $\mathcal{V}$ to an input triangulated surface mesh $\mathcal{S}$ that is well-defined (*i.e.* free from numerical mesh 'glueing' artifacts) when the surface is self-intersecting. As in Sacht *et al.* [SJP*13], we assume that there exists a nearby non-self-intersecting mesh $\tilde{\mathcal{S}}$ and a mapping $\boldsymbol{\phi}_{\tilde{\mathcal{S}}}^{\mathcal{S}} : \tilde{\mathcal{S}}^V \rightarrow \mathbb{R}^3$ with

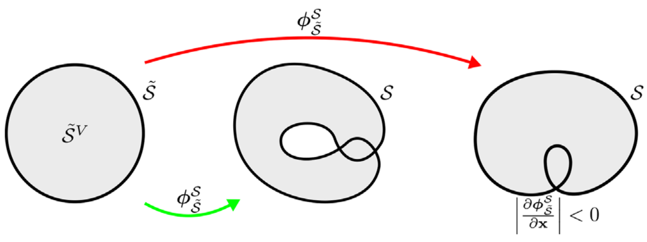

$\phi_{\tilde{\mathcal{S}}}^{\mathcal{S}}$
$\tilde{\mathcal{S}}^V$
$\tilde{\mathcal{S}}$
$\mathcal{S}$
$\phi_{\tilde{\mathcal{S}}}^{\mathcal{S}}$
$\mathcal{S}$
$\left|\frac{\partial \phi_{\tilde{\mathcal{S}}}^{\mathcal{S}}}{\partial \mathbf{x}}\right| < 0$

Figure 2: *Intersection-free mapping. Two mappings from a non-self-intersecting region $\tilde{\mathcal{S}}^V$ to self-intersecting boundary $\mathcal{S}$ are shown. The second mapping (right) requires the existence of a negative Jacobian determinant.*

non-singular Jacobian determinant (see Figure 2). Here $\tilde{\mathcal{S}}^V$ is the unambiguously defined interior of the non-self-intersecting $\tilde{\mathcal{S}}$. Intuitively, if we can find a mapping $\boldsymbol{\phi}_{\tilde{\mathcal{S}}}^{\mathcal{S}}$, then we can define a volumetric embedding mesh for $\tilde{\mathcal{S}}$ unambiguously with standard techniques and then push it forward under the mapping. However, unlike Sacht *et al.* [SJP*13], we do not explicitly create $\boldsymbol{\phi}_{\tilde{\mathcal{S}}}^{\mathcal{S}}$ or $\tilde{\mathcal{S}}$ but rather use their existence to guide our mesh creation strategy.

We build our embedding hexahedron mesh $\mathcal{V}$ from the intersection of the input surface $\mathcal{S}$ with a uniform background grid where cells in contiguous regions are copied to form sub-meshes that are sewn together using techniques inspired by Teran *et al.* [TSB*05] and Sifakis *et al.* [SDF07] but in a manner designed to mimic the image of $\boldsymbol{\phi}_{\tilde{\mathcal{S}}}^{\mathcal{S}}$. Our approach is ultimately similar to that of Li and Barbič [LB18] in that we create the volumetric embedding mesh without modifying the self-intersecting surface and our region duplication/sewing is equivalent to discovering immersions. Unlike Li and Barbič [LB18], our approach uses nearly no exact and/or adaptive precision arithmetic as we do not resolve the geometry of intersection from triangles in $\mathcal{S}$ with themselves or with cells in the background grid and we do not use CSG operations as in Sifakis *et al.* [SDF07]. We simply require accurate determination of which triangles intersect which grid cells. This limits the accuracy of our method for large grid spacing (low-resolution) and we run with smaller grid spacing (high-resolution) when necessary. To prevent this from causing excessive element counts, we provide a topology-preserving mesh coarsening strategy similar to that of Wang *et al.* [WJST14]. Lastly, we provide a technique for efficiently converting the uniform-grid-based embedding hexahedron mesh to a tetrahedron mesh that robustly handles duplicated regions of the hexahedron mesh near self-intersecting features. As in Li and Barbič [LB18], we use a body-centred cubic (BCC) structure [MBTF03] for this conversion.

We summarize our novel contributions as

- An efficient technique with reduced use of exact/adaptive precision arithmetic for building an embedding hexahedron mesh for an input self-intersecting triangle mesh from a uniform grid that is equivalent to pushing forward one unambiguously defined from a self-intersection-free state.
- A topology aware embedding mesh coarsening strategy to provide for flexible resolution/element count.
- A topology aware BCC approach for converting the embedding hexahedron mesh into an embedding tetrahedron mesh.

2. Related Work

We discuss methods in the existing literature that are related to our approach. We first provide detailed discussion of Li and Barbič [LB18] and Sacht *et al.* [SJP*13] since these works are most relevant to ours. In addition to techniques that compute a volumetric mesh from an input triangle mesh, we discuss relevant works in the fracture and virtual surgery literature since our approach makes use of grid cutting operations to intersect the input surface mesh with a uniform background grid. Lastly, we discuss relevant surface modelling techniques that address prevention of self-intersection and overlap.

2.1. Volumetric mesh creation from a self-intersecting triangle mesh

Sacht *et al.* [SJP*13] were the first to design an approach that creates an appropriately overlapping tetrahedron mesh from a self-intersecting triangle mesh. As with our approach, they assume the existence of a mapping $\boldsymbol{\phi}^{S}_{\tilde{S}}$ from a non-self intersecting counterpart $\tilde{\mathcal{S}}$ to the input mesh $\mathcal{S}$. Unlike our approach, they explicitly form $\tilde{\mathcal{S}}$ and the mapping $\boldsymbol{\phi}^{S}_{\tilde{S}}$. $\tilde{\mathcal{S}}$ is created by a backward process using cMCF followed by a forward process that minimizes distortion-energy and deviation from $\mathcal{S}$ subject to collision constraints. The cMCF is known to remove self-intersections for sphere-topology surfaces [KSBC12] and accordingly, their method is limited to input surfaces with genus zero. They create a tetrahedron mesh using the self-intersection-free $\tilde{\mathcal{S}}$ and then push it forward under $\boldsymbol{\phi}^{S}_{\tilde{S}}$, which is created by mapping the boundary of the tetrahedron mesh to $\mathcal{S}$ and propagating deformation to the interior. Our approach is similar in spirit, but we do not explicitly create $\tilde{\mathcal{S}}$ or $\boldsymbol{\phi}^{S}_{\tilde{S}}$; furthermore, we can support input surfaces with genus larger than zero. In addition, since they do not directly generate tetrahedra in world space, they must take care to maintain tetrahedron mesh quality under deformation in $\boldsymbol{\phi}^{S}_{\tilde{S}}$.

Like Li and Barbič [LB18], we create a volumetric embedding mesh in world space. Li and Barbič [LB18] observed that the creation of a volumetric mesh from a self-intersecting surface is related to the geometric and algebraic topological determination of immersions (locally injective mappings) from a compact 3-manifold to a portion of the world space domain. As in our approach, they start by dividing world space into contiguous regions using the input surface mesh $\mathcal{S}$. However, they use exact/adaptive precision arithmetic to intersect $\mathcal{S}$ with itself to achieve this. We use simplified/less costly intersections of triangles in $\mathcal{S}$ with uniform background grid cells and edges. We only need to know whether an intersection occurs or not; we do not need to resolve the intersection geometry. Immersions do not always exist, and Li and Barbič [LB18] developed a graph-based algorithm to determine if one exists. Their method for computing these is NP-complete; however, as they note, this is not a bottleneck for most computer graphics applications. When such an immersion exists, they compute it by duplicating the contiguous regions, intersecting each duplicate with a uniform background tetrahedron lattice to create local tetrahedron meshes that are then sewn together appropriately using their graph structure. We also duplicate and then sew together contiguous regions, but we use simplified criteria that, while more efficient, can only give accurate results for simple immersions. Although, as Li and Barbič [LB18] note, the vast majority of applications in computer graphics only require simple immersions. As with our approach, they also prevent artificial glueing for embedded meshes with nearly intersecting features. While Li and Barbič [LB18] can accurately compute non-simple immersions, they cannot handle exactly coincident portions $\mathcal{S}$ with non-zero measure, which we can handle. Broadly speaking, the Li and Barbič [LB18] approach is more general than our method, but more costly, primarily due to the comparably large use of exact/adaptive precision arithmetic.

2.2. Mesh creation and mesh cutting

The virtual node algorithm (VNA) of [MBF04] allows cutting a tetrahedron mesh along piecewise-linear paths through the mesh. As in our approach, duplicates of cut elements are used to resolve necessary topological features. Teran *et al.* [TSB*05] built a generalization of this approach to create embedding meshes for nearly overlapping input triangle meshes. Sifakis *et al.* [SDF07] further extended the VNA to allow for arbitrary cut geometry. A downside to the geometric flexibility provided by these generalizations is their need for adaptive precision arithmetic and CSG. Motivated by this, Wang *et al.* [WJST14] developed a technique that allows for geometric flexibility without the need for adaptive precision arithmetic. Their approach allows for arbitrary cut surfaces by generalizing the original VNA [MBF04] to allow cuts to pass through vertices, edges or faces of the embedding mesh. This alone does not provide sufficient geometric flexibility since cuts cannot pass through facets multiple times. To resolve such cuts, the algorithm is run at high-resolution where facets are only intersected once and then coarsened in a topologically aware manner.

The extended finite element method (XFEM) [BB99] is very similar to VNA. An XFEM-based but re-meshing-free approach for cutting of deformable bodies is presented in Koschier *et al.* [KBT17]. In a similar spirit, Zhang *et al.* [ZDZ*18] utilized the cracking node method [SB09], which is similar to XFEM but uses discontinuous cracks centred at nodes in order to approximate crack paths. This yields an efficiency advantage over XFEM which in turn allows for simulating materials with many evolving, branching cracks. The reader is also referred to the survey of Wu *et al.* [WWD15] for more discussion of mesh cutting techniques in computer graphics.

More generally, tetrahedron mesh creation has been robustly addressed by a number of works [Si15, HZG*18, LS07, MBF03, DCB13, JAYB15]. For example, Si [Si15] pursued a Delaunay refinement strategy in order to provide certain guarantees on tetrahedron quality. However, sliver tetrahedra are still possible [HZG*18]. The method presented in Hu *et al.* [HZG*18] can handle arbitrary triangle soup as input and returns a high-quality approximated constrained tetrahedron mesh, though performance is hindered to an extent due to prominent usage of exact rational arithmetic. However, recently, those performance bottlenecks were alleviated and replaced with floating-point computations [HSW*20]. Notably, researchers have recently presented a successful method for learning high-quality tetrahedron meshes from noisy point clouds or a single image [GCX*20].

2.3. Self-intersecting curves and surfaces

Self-intersecting curves and surface meshes have been considered for many years in both the mathematics and computer science literature. In two dimensions, algorithms and theorems related to identifying self-intersecting curves date back to Titus [Tit61], with many more recent contributions [Bla67, Mar74, SV92, HL95, GC11, EFW20]. Notably, many problems related to identifying self-intersections are NP-complete [EM09]. Despite this, efficient algorithms frequently exist; for example, Mukherjee [Muk14] gave a quadratic algorithm (in the number of points on the discrete curve) to determine the mapping from a disk to an arbitrarily stretched, potentially self-overlapping curve, also known as computing an immersion of the disk. In another vein, Li [Li11] used Gauss diagrams from knot theory to characterize self-intersecting two-dimensional projections of three-dimensional polygons, in order to understand whether there are one or multiple ways to perform mesh repair algorithms like Brunton *et al.* [BWS*09].

In the context of three-dimensional mesh generation and animation, self-intersections are typically treated as degeneracies to be avoided or removed. For example, Von Funck *et al.* [FTS06] provided a method for deforming surfaces that prevents new self-intersections from occurring, due to the smoothness requirements they place on the vector fields governing the deformation. The tool devised in Angelidis *et al.* [ACWK06] allows for local prevention of self-intersections when deforming a mesh. A method for avoiding introducing self-intersections within the free-form deformation (FFD) modelling scheme [Béz70, SP86] was presented in Gain and Dodgson [GD01]. The space-time interference volumes introduced in Harmon *et al.* [HPSZ11] can be used to eliminate self-intersections in meshes, although this method is not always guaranteed to work (the method is primarily intended for interacting with non-self-intersecting input geometry). Shen *et al.* [SOS04] built an implicit surface from polygon soup, resulting in a watertight mesh that approximates the input surface data. Attene [Att10] deleted overlapping triangles and subsequently performed a gap-filling procedure in the resulting holes. Similarly, Jacobson *et al.* [JKSH13] presented a method based on the generalized winding number (which, notably, is still applicable to triangle soups and point clouds [BDS*18], unlike the standard winding number). Their method results in fusing together self-intersecting parts of the mesh. Recently, Tao *et al.* [TBFL19] demonstrated a method for accurately and efficiently generating cut cell meshes for arbitrary triangulated surfaces, including those with degeneracies. However, again, they treat self-intersections as flaws to be removed, unlike in our method where self-intersections are valid features of our inputs and outputs. Nonetheless, an attractive aspect of their algorithm is robust resolution of mesh degeneracies and singularities, unlike methods like Refs. [EB14, KT10] which require random numerical perturbations of the background cut cell grid. Finally, we also highlight Mitchell *et al.* [MASS15], which describes a method for representing self-intersecting surfaces using implicit functions sampled on a specialized hexahedron mesh.

3. Algorithm Overview

The input to our algorithm is a triangulated surface mesh $\mathcal{S}$. The output is a uniform-grid-based embedding hexahedron mesh counterpart $\mathcal{V}$ to $\mathcal{S}$ that is well-defined (*i.e.* free from numerical mesh 'glueing' artifacts) even when $\mathcal{S}$ is self-intersecting (see Section 10 for examples).

We briefly summarize the three main stages of our algorithm, as detailed in Figure 3. In the first stage, volumetric extension (Section 5), we create a hexahedron mesh $\mathcal{U}$ from the background grid that only covers the input surface $\mathcal{S}$ with connectivity designed to mimic it. We sign its vertices depending on inside/outside information derived from the hypothetical self-intersection-free counterpart $\tilde{\mathcal{S}}$. We emphasize that this volumetric extension mesh only surrounds $\mathcal{S}$. Accordingly, the second stage of the algorithm is interior extension region creation (Section 6). Nodes of the background grid are partitioned using the edges cut by $\mathcal{S}$, and then we decide which regions are interior. Interior regions will be copied a certain number of times corresponding to the number of times which interior portions of the hypothetical self-intersection-free counterpart $\tilde{\mathcal{S}}^V$ will overlap under the hypothetical push forward mapping $\boldsymbol{\phi}^S_{\tilde{S}}$; the number of copies is approximate at this stage. For each interior region j^I with at least one copy, we create a hexahedron mesh $\mathcal{V}^{j^I,c}$ for each copy c. In the third stage of the algorithm (Section 7), interior extension regions meshes $\mathcal{V}^{j^I,c}$ are sewn together and into the volumetric extension $\mathcal{U}$ to produce the final output mesh. We additionally provide a coarsening approach in Section 8 to provide user control over the embedding mesh resolution as well as a topologically aware technique for converting the hexahedron mesh $\mathcal{V}$ into a tetrahedron mesh $\mathcal{T}$.

4. Definitions and Notation

We take a triangle mesh $\mathcal{S} = (\mathbf{x}^S, \mathbf{m}^S)$ as input where $\mathbf{x}^S = [\mathbf{x}^S_0, \mathbf{x}^S_1, \ldots]$ denotes the array of vertex positions and $\mathbf{m}^S = [\mathbf{m}^S_0, \mathbf{m}^S_1, \ldots]$ denotes the array of triangle vertices. More precisely, $\mathbf{x}^S_i \in \mathbb{R}^3$ is the position of vertex i and $\mathbf{m}^S_t \in \mathbb{N}^3$ stores the vertices of triangle t. We also define $\mathcal{I}^S_i$ to be the collection of triangles incident to vertex i. We show an example of these definitions for the mesh $\mathcal{S}$ in Figure 4 (left) using a flattened array representation of $\mathbf{m}^S$ where entry m^S_j is a vertex of triangle $\lfloor j/3 \rfloor$ (*e.g.* m^S_3, m^S_4, m^S_5 are vertices for triangle 1). We assume that $\mathcal{S}$

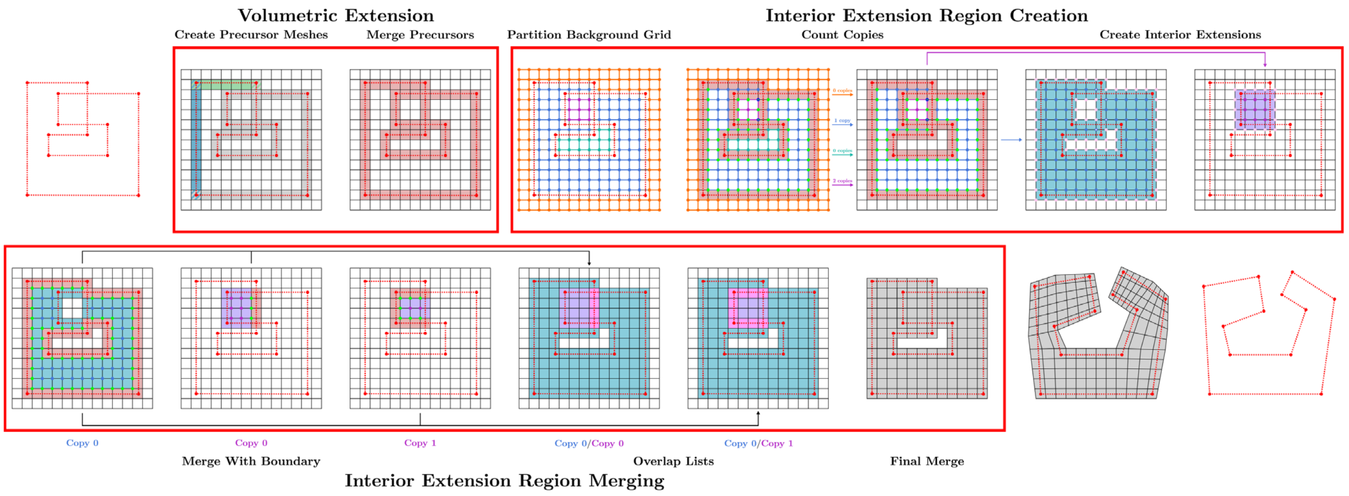
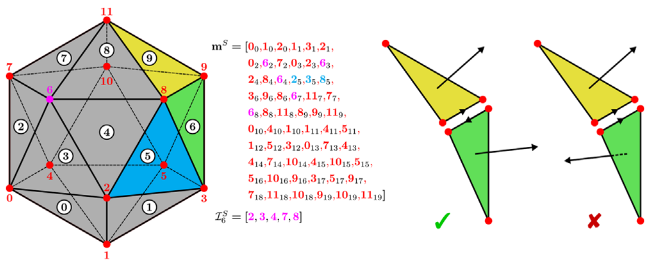


Figure 3: *Algorithm overview. Given an initial input surface mesh $\mathcal{S}$, there are three major steps in the computation of the final volumetric extension mesh $\mathcal{V}$: volumetric extension, interior extension region creation, and interior extension region merging. (Volumetric extension) In this step, we create a precursor mesh for each element in $\mathcal{S}$, and compute preliminary signing information for the vertices. We then merge the precursor meshes to create the volumetric extension $\mathcal{U}$ and correct the signing information where necessary. (Interior extension region creation) In preparation for growing the volumetric extension into the interior, we first partition the nodes of the background grid using the edges cut by $\mathcal{S}$. We decide which regions are interior and count the copies of each region using the vertices of $\mathcal{U}$ which have negative sign. For each interior region I with at least one copy, we then create a hexahedron mesh $\mathcal{V}^{I,c}$ for each copy c. (Interior extension region merging) The merging process begins with copying relevant hexahedra from $\mathcal{U}$ into $\mathcal{V}^{I,c}$. First, certain vertices of $\mathcal{V}^{I,c}$ are replaced by corresponding vertices from $\mathcal{U}$. Hexahedra to be replaced are then removed from $\mathcal{V}^{I,c}$ before the boundary hexahedra are copied in. We then merge the various meshes $\mathcal{V}^{I,c}$ by first determining where different meshes overlap, and then using these hexahedra overlap lists to perform the final merge.*



$$\mathbf{m}^S = [0_0,1_0,2_0,1_1,3_1,2_1,0_2,6_2,7_2,0_3,2_3,6_3,2_4,8_4,6_4,2_5,3_5,8_5,3_6,9_6,8_6,6_7,11_7,7_7,6_8,8_8,11_8,8_9,9_9,11_9,0_{10},4_{10},1_{10},1_{11},4_{11},5_{11},1_{12},5_{12},3_{12},0_{13},7_{13},4_{13},4_{14},7_{14},10_{14},4_{15},10_{15},5_{15},5_{16},10_{16},9_{16},3_{17},5_{17},9_{17},7_{18},11_{18},10_{18},9_{19},10_{19},11_{19}]$$

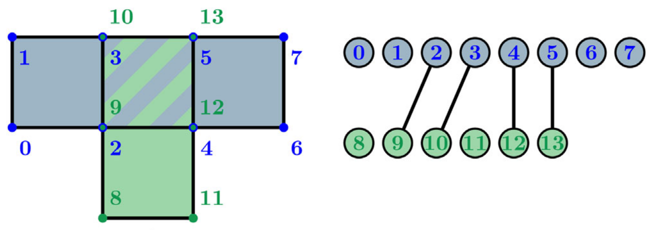
$$\mathcal{I}_6^S = [2,3,4,7,8]$$

Figure 4: *Mesh conventions. (Left) A sample triangle mesh is shown, along with the vector $\mathbf{m}^S$. The subscripts on the entries of $\mathbf{m}^S$ denote the triangle t which the entries correspond to. The incident triangles $\mathcal{I}_6^S$ for vertex 6 are also shown. For example, triangle 4 has vertices 2, 8, 6. Hence, 4 is in $\mathcal{I}_6^S$. The first 10 faces, visible from the front, have been labelled on the mesh. (Right) The left pair of triangles is consistently oriented; the orientations of the edge induced by the normals point in opposite directions. For the right pair, the orientations on the common edge point in the same direction; this is not consistent.*

is closed and consistently oriented. The orientation condition is demonstrated in Figure 4 (right). We output a hexahedron mesh $\mathcal{V} = (\mathbf{x}^V, \mathbf{m}^V)$ with $\mathbf{x}^V = [\mathbf{x}_0^V, \mathbf{x}_1^V, \dots]$ and $\mathbf{m}^V = [\mathbf{m}_0^V, \mathbf{m}_1^V, \dots]$ defined in an analogous manner, with $\mathbf{m}_h^V \in \mathbb{N}^8$ storing the vertices of hexahedron h. Each hexahedron in the mesh is geometrically coincident with a grid cell in a background uniform grid $\mathcal{G}_{\Delta x}$ with spacing Δx. We will occasionally refer to vertices whose positions are stored in $\mathbf{x}^V$ as vertices in $\mathbf{x}^V$. For ease of visualization, we use 2D counterparts to $\mathcal{S}$ and $\mathcal{V}$ in illustrative figures. In this case, $\mathcal{S}$ is a segment mesh and $\mathcal{V}$ is a quadrilateral mesh. As in Figure 4, we shall continue to flatten arrays in figures for a simplified presentation.

4.1. Merging

We construct the final hexahedron mesh $\mathcal{V}$ by merging portions of various precursor hexahedron meshes in a manner similar to techniques used in Refs. [TSB*05, WDG*19, WJST14, LB18]. As with $\mathcal{V}$, each hexahedron in a precursor mesh is geometrically coincident with background grid cells. All precursor meshes share the same vertex array $\mathbf{x}^V$, although its size will change as we converge to the final $\mathcal{V}$. At various stages of the algorithm, we will merge certain geometrically coincident precursor hexahedra. To perform a merge, we view the set of all vertices in $\mathbf{x}^V$ as nodes in a single undirected graph and introduce graph edges between them. In subsequent sections, we refer to such edges in the undirected graph as adjacencies to distinguish them from edges in the various meshes; vertices connected by an adjacency will be called adjacent. Each adjacency will correspond to a pair of geometrically coincident vertices, but generally not all geometrically coincident vertices will be adjacent. Once all adjacencies are defined, we compute the connected components of the graph using depth-first search. All vertices in a connected component are considered to be the same and we choose one representative for all mesh entries. We note that this operation may be carried out on more than two meshes at once and that it can lead to duplicate hexahedra and in this case, we remove all but one.



$$\mathbf{m}^0 = [0,2,3,1,2,4,5,3,4,6,7,5]$$
$$\mathbf{m}^1 = [8,11,12,9,9,12,13,10]$$
$$\mathbf{m}^2 = [0,2,3,1,2,4,5,3,4,6,7,5,8,11,4,2,2,4,5,3]$$
$$\mathbf{m}^2 = [0,2,3,1,2,4,5,3,4,6,7,5,8,11,4,2]$$

Figure 5: *Mesh merge. An example of two meshes merging together. Vertices 2, 3, 4 and 5 merge with vertices 9, 10, 12 and 13, respectively. A new vector* $\mathbf{m}^2$ *is created to hold all of the hexahedron vertices post-merge, and the extra hexahedron (in red) is then removed.*

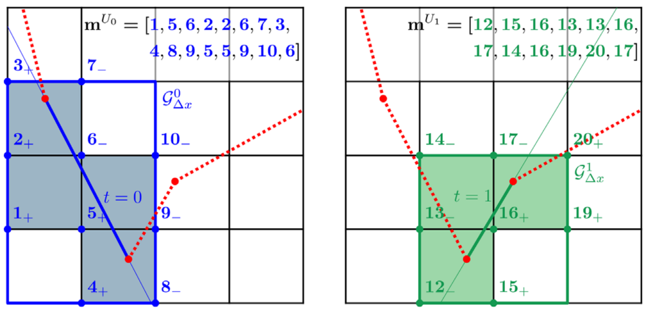

$\mathbf{m}^{U_0} = [1,5,6,2,2,6,7,3,4,8,9,5,5,9,10,6]$
3_+
7_-
$\mathcal{G}^0_{\Delta x}$
2_+
6_-
10_-
$t=0$
1_+
5_+
9_-
4_+
8_-
$\mathbf{m}^{U_1} = [12,15,16,13,13,16,17,14,16,19,20,17]$
14_-
17_-
20_+
$\mathcal{G}^1_{\Delta x}$
$t=1$
13_-
16_+
19_+
12_-
15_+

Figure 6: *Precursor meshes. (Left) Surface element* $t = 0$ *creates quadrilateral mesh* $\mathcal{U}_0$*. (Right) Surface element* $t = 1$ *creates quadrilateral mesh* $\mathcal{U}_1$*. Each element creates copies of the grid cells it intersects by introducing new vertices which are geometrically coincident to grid nodes. The dashed red segments in both are other surface elements in* $\mathcal{S}$*.*

Furthermore, replacing all vertices in a connected component with one representative results in unused vertices in $\mathbf{x}^V$. We remove all unused vertices in a final pass, changing indexing in $\mathbf{m}^V$ accordingly. We illustrate the connected component calculation, vertex replacement and unused vertex removal in Figure 5.

5. Volumetric Extension

We first create a volumetric extension $\mathcal{U}$ of the surface $\mathcal{S}$. It is a hexahedron mesh that covers the input surface $\mathcal{S}$ and is designed to have topological properties analogous to $\mathcal{S}$. Since it is an extension of $\mathcal{S}$, we can sign the vertices of $\mathcal{U}$ depending on which side of the surface they lie on. Overlapping regions in $\mathcal{S}$ complicate this process, but it can be disambiguated by considering the pre-image of the surface to its overlap-free counterpart $\tilde{\mathcal{S}}$ under the mapping $\boldsymbol{\phi}^{\mathcal{S}}_{\tilde{\mathcal{S}}}$. We sign vertices using a well-defined local criterion, in contrast to global methods such as ray casting which are not applicable to self-intersecting polygons.

5.1. Surface element precursor meshes

In order to mimic the topology of $\mathcal{S}$, we create its volumetric extension $\mathcal{U}$ from precursor hexahedron meshes $\mathcal{U}_t = (\mathbf{x}^V, \mathbf{m}^{U_t})$ associated with each triangle t in $\mathcal{S}$. All precursor meshes share the common vertex array $\mathbf{x}^V$ and this process begins its evolution to the final vertex array for $\mathcal{V}$. We define the hexahedron mesh $\mathcal{U}_t$ for triangle t from the sub-grid $\mathcal{G}^t_{\Delta x}$ of $\mathcal{G}_{\Delta x}$ defined by the grid-cell-aligned bounding box of t. Since $\mathcal{G}^t_{\Delta x}$ can itself be viewed as a hexahedron mesh, we define $\mathcal{U}_t$ as the sub-mesh consisting of hexahedra in $\mathcal{G}^t_{\Delta x}$ intersected by t. The vertices incident to these hexahedra are added to $\mathbf{x}^V$ as new vertices. Note that each triangle creates its own hexahedron mesh with distinct vertices. We sign the vertices in each $\mathcal{U}_t$ depending on which side of the plane containing the triangle t that they lie on. We illustrate this process in Figure 6. Lastly, we note that these signs are low-cost preliminary approximations to the signs in the final volumetric extension $\mathcal{U}$. In some cases, the signs computed in this phase will not be accurate in the volumetric extension, and we provide a more accurate but costly signing method when this occurs (discussed in Section 5.2); however, in many cases, they are equal to the final signs, and their comparably low computational cost improves overall algorithm performance.

We perform the aforementioned intersection of triangle t with background grid cells using the intersection function from CGAL's 2D/3D Linear Geometry Kernel [The20, BFG*20]. Every call to the CGAL library here and in subsequent sections uses the exact arithmetic kernel; conversely, all of our exact/adaptive precision arithmetic is limited to CGAL.

5.2. Merge surface element meshes

We merge portions of the precursor meshes $\mathcal{U}_t$ to form the volumetric extension hexahedron mesh $\mathcal{U}$ by defining adjacency between vertices in $\mathbf{x}^V$ as described in Section 4.1. We define this adjacency from the mesh connectivity of $\mathcal{S}$ as follows. Two vertices i_1, i_2 are adjacent if

- they are geometrically coincident ($\mathbf{x}^V_{i_1} = \mathbf{x}^V_{i_2}$), and
- there is a vertex j of $\mathcal{S}$ and triangles $t_1, t_2 \in \mathcal{I}^S_j$ such that i_1 is incident to a hexahedron h_1 of $\mathcal{U}_{t_1}$, i_2 is incident to a hexahedron h_2 of $\mathcal{U}_{t_2}$ and h_1 is geometrically coincident to h_2.

Note that this is different from defining geometrically coincident vertices in $\mathcal{U}_t$ for $t \in \mathcal{I}^S_j$ to be adjacent (see the geometry of Figure 14). In other words, if t_1 and t_2 share a common vertex, then geometrically coincident hexahedra from $\mathcal{U}_{t_1}$ and $\mathcal{U}_{t_2}$ are merged (see Figure 7). If all preliminary signs for merged vertices agree, the merged vertex is given that sign. Otherwise (see Figure 8), we re-compute the sign from their geometric relation to $\mathcal{S}$. This disagreement occurs in regions of high curvature, and we use an eikonal strategy [OF03] to propagate positive signs from $\mathcal{S}$ in the direction of the surface normal and minus signs in the opposite direction.

This is well-defined in light of the assumed existence of the pre-image $\tilde{\mathcal{S}}$ of $\mathcal{S}$ under $\boldsymbol{\phi}^{\mathcal{S}}_{\tilde{\mathcal{S}}}$. Here, each vertex $\mathbf{x}^V_i$ in $\mathcal{U}$ is associated with some collection of precursor meshes $\mathcal{U}_t$ where $\mathbf{x}^V_i$ was created from merging vertices in $\mathcal{U}_t$. This defines a local patch S^V_i of triangles t in $\mathcal{S}$ associated with $\mathbf{x}^V_i$ where $t \in S^V_i$ if a vertex of $\mathcal{U}_t$ merged to create $\mathbf{x}^V_i$.

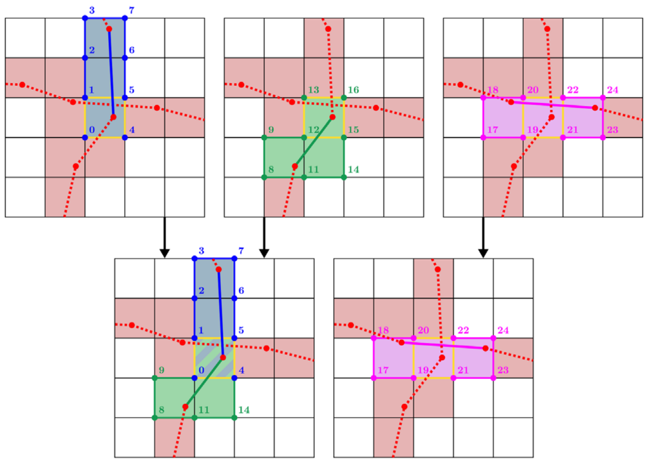


Figure 7: *Precursor merge. The 12 vertices bordering the cell marked in yellow are merged into eight resulting vertices. Blue vertices 0, 1, 4, 5 and green vertices 12, 13, 15, 16 are merged, respectively. However, magenta vertices 19, 20, 21, 22 do not merge with the blue or green vertices since their associated surface element is topologically distant.*

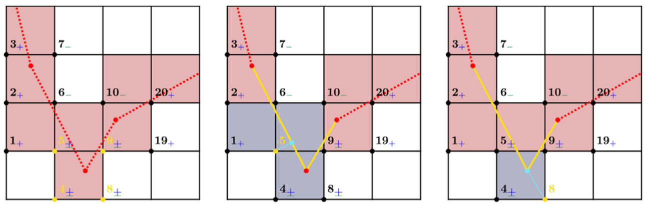


Figure 8: *Closest facet. (Left) The four vertices in yellow all have ambiguous signs. (Middle) To sign vertex 5, we generate the local patch S_5^V, which are the segments shown in yellow. The closest facet (indicated in cyan) lies on a face. (Right) A similar process is illustrated for vertex 8, but here the closest facet is a vertex.*

When propagating signs from $\mathcal{S}$ to $\mathbf{x}_i^V$, only these local triangles are considered in order to exclude geometrically close but topologically distant triangles. We adopt the local point-in-polygon method of Horn and Taylor [HT89]. First, we compute the closest mesh facet (including faces) in S_i^V to $\mathbf{x}_i^V$ by converting S_i^V to a CGAL surface mesh and then using the locate function from the Polygon Mesh Processing package [BSMF20, LRLTY20]. If the closest facet is an edge or vertex, we add its incident triangles from $\mathcal{S}$ to S_i^V if they were not already included. If more triangles are added, we re-compute the closest mesh facet. We illustrate this process in Figures 8 and 9. If the closest facet is a triangle, we compute the sign depending on the side of the plane containing the triangle that the point lies on. If the closest facet is an edge or point, we use the conditions from Horn and Taylor [HT89], which we summarize below:

- If the closest facet is an edge, then the sign is -1 if the edge is concave (as determined by the normals of the incident faces) and $+1$ if it is convex.
- If the closest facet is a vertex, then there exists a discrimination plane with an empty half-space. Choosing any such plane, the sign is -1 if the edges defining the plane are concave and $+1$ if they are convex.

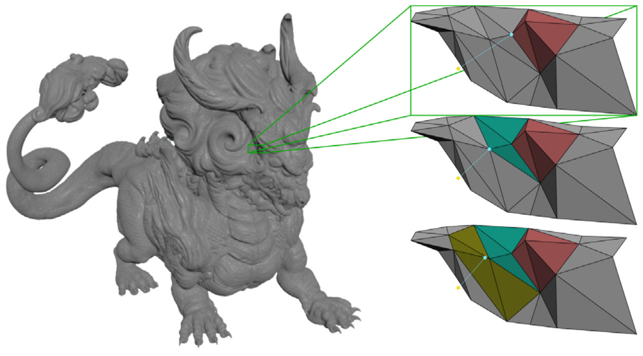


Figure 9: *Patch expansion. The local patch S_i^V corresponding to the yellow vertex is shown. The initial patch is indicated in red, and the closest facet is a vertex. We add the missing incident triangles (turquoise) and re-compute the closest facet, which is again a vertex with incident triangles not in the patch. Repeating the process (with new triangles in dark yellow), the closest feature is now on an edge and we proceed to the signing criteria.*

A discrimination plane is defined by two non-collinear incident edges and it has an empty half-space if all incident faces and edges lie on one side of the plane or on the plane itself. Note that geometries can be constructed at any resolution for which the addition of more triangles in the closest facet detection can result in an incorrect sign evaluated from the above conditions. However, we did not observe this failure in any practical mesh. Even the addition of more triangles is only rarely needed; among the meshes used in the present work, only the mesh of Figure 27 requires additional triangles at few resolutions.

6. Interior Extension Region Creation

We grow the volumetric extension $\mathcal{U}$ on its interior boundary (defined by vertices with negative sign) to create the remainder of the volumetric mesh $\mathcal{V}$. To determine where to grow the extension, we first partition the background grid nodes into connected components defined by its intersection with $\mathcal{S}$. Viewing $\mathcal{G}_{\Delta x}$ as a graph, we define two grid nodes to be adjacent if they are incident to a common grid edge that does not intersect $\mathcal{S}$ (again using CGAL to determine intersection). We then compute the connected components using depth-first search; we refer to connected components as regions. This is a simplistic criterion which can lead to an over-count in the number of regions, as demonstrated in Figure 10. We opt not to use a more accurate criteria using material connectivity determined by the intersection of $\mathcal{S}$ with $\mathcal{G}_{\Delta x}$ as in the CSG operations of Sifakis *et al.* [SDF07] as these operations are extremely costly (see Li and Barbič [LB18]) and our approach is robust to over-counting.

We consider a region to be interior if any grid node is geometrically coincident to a negatively signed vertex in $\mathbf{x}^V$. All other regions are considered exterior, and we discard them (Figure 11). We create at least one hexahedron mesh for each interior region I. Multiple copies of interior meshes are created in some regions to account for

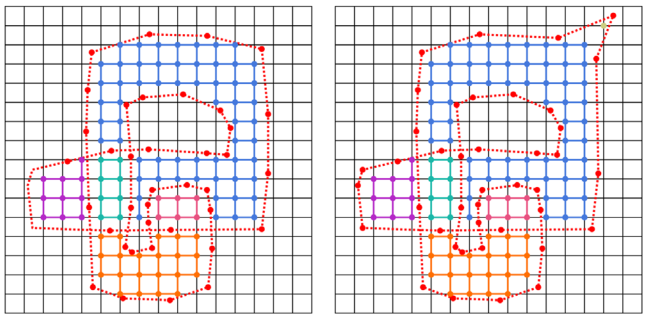


Figure 10: *Region over-count. As the process of partitioning the grid only uses connectivity based on grid edges, it is possible for a contiguous region to be split into multiple regions. Shifting some of the vertices of $\mathcal{S}$ on the left results in the geometry on the right, which contains an additional region in the upper-right corner since no edge connects this grid node to the larger blue region.*

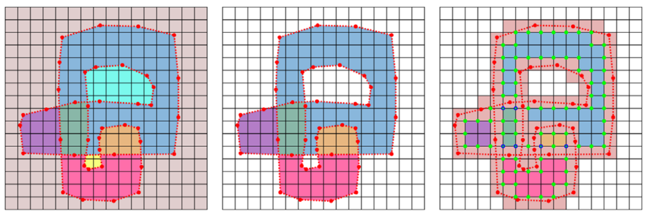


Figure 11: *Connected regions. (Left) The surface partitions the background grid into contiguous regions. (Middle) The exterior regions are removed. (Right) The volumetric extension $\mathcal{U}$ is shown, along with the negatively signed vertices in green. Multiple geometrically coincident vertices are indicated using blue circles with green centres.*

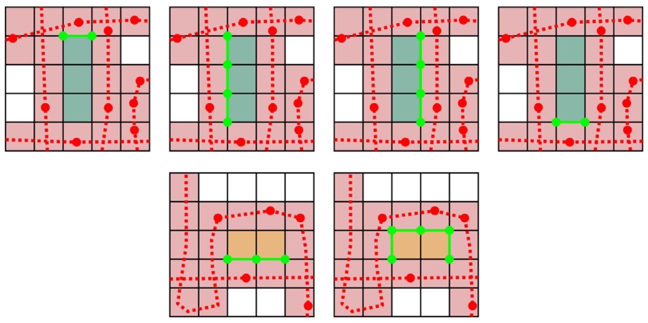


Figure 12: *Copy counting. The two regions from Figure 11 having multiple copies are shown. Each copy is displayed with its corresponding connected component of vertices with negative sign.*

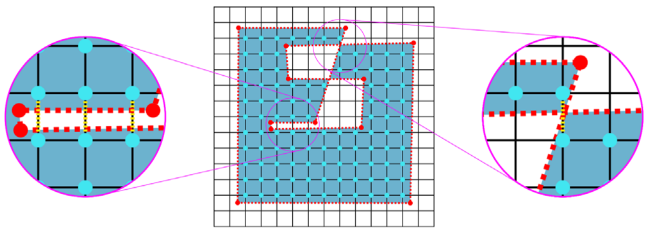


Figure 13: *Edge cut criterion. Grid nodes $\mathbf{x_i}$ of a region are shown, along with two examples showing that adjacent grid nodes may have their common edge cut by a triangle (cut edges are indicated by the dashed yellow lines). In this case, adjacencies are not built between the corresponding vertices in $\mathcal{V}_{\mathbf{i}}^{I,0}$ to avoid unwanted sewing.*

multiple overlapping portions of the volumetric domain. We denote copy c of the hexahedron mesh for region I by $\mathcal{V}^{I,c} = (\mathbf{x}^V, \mathbf{m}^{V^{I,c}})$. As before, each of these meshes uses the common vertex array $\mathbf{x}^V$.

A region I will have more than one copy if any grid node of I is geometrically coincident with more than one negatively signed vertex in $\mathbf{x}^V$. We count the number of copies as follows. Consider the set of negatively signed vertices in $\mathbf{x}^V$ geometrically coincident to some grid node of I. The number of copies is equal to the connected components of this set where two vertices i_1 and i_2 are adjacent if both are coincident to a common hexahedron in $\mathcal{U}$, as shown in Figure 12.

We construct the first copy of $\mathcal{V}^{I,0}$ for region I from precursor hexahedron meshes $\mathcal{V}_{\mathbf{i}}^{I,0} = (\mathbf{x}^V, \mathbf{m}_{\mathbf{i}}^{V^{I,0}})$ where $\mathbf{i}$ indexes grid nodes of region I. Note that these are not vertices in $\mathbf{x}^V$. For each node $\mathbf{i}$, $\mathbf{m}_{\mathbf{i}}^{V^{I,0}}$ consists of eight hexahedra which are geometrically coincident with the eight local background grid cells incident to $\mathbf{i}$; the vertices of these hexahedra are defined by the positions of $\mathbf{i}$ and the surrounding 26 background grid nodes, which are added to $\mathbf{x}^V$. Of these 27 vertices, we call the vertex corresponding to $\mathbf{i}$ itself a central vertex (which will be used in Section 7.1). We again merge these precursors as described in Section 4.1 where adjacencies between the vertices in $\mathbf{x}^V$ are defined as follows. Two vertices i_1 and i_2 are adjacent if

- i_1 and i_2 are geometrically coincident,
- there are grid nodes $\mathbf{i}$ and $\mathbf{j}$ of I such i_1 is a vertex of $\mathcal{V}_{\mathbf{i}}^{I,0}$ and i_2 is a vertex of $\mathcal{V}_{\mathbf{j}}^{I,0}$, and
- $\mathbf{i}$ and $\mathbf{j}$ are incident to a common edge of $\mathcal{G}_{\Delta x}$ which does not intersect $\mathcal{S}$.

The third criterion prevents connection between geometrically close but topologically distant features, as illustrated in Figure 13. The final array $\mathbf{m}^{V^{I,0}}$ is then formed by concatenating all of the arrays $\mathbf{m}_{\mathbf{i}}^{V^{I,0}}$ and removing any duplicate hexahedra. The remaining copies $\mathcal{V}^{I,c}$ are created by duplicating $\mathbf{m}^{V^{I,0}}$ with new vertices distinct from those corresponding to $\mathcal{V}^{j^I,0}$ and any other copy.

In general, our copy counting process can result in an over-count. We note that our process is analogous to the cell creation portion of the method of Li and Barbič [LB18]. They show that the correct number of copies is equal to the winding number of the (simple) region. We do not compute the winding number since our over-count is typically resolved during the merging process described in Section 7. However, unresolvable failure cases occur when the background uniform grid $\mathcal{G}_{\Delta x}$ cannot resolve thin features or high-curvature in $\mathcal{S}$. To resolve these cases, we refine the background grid using a strategy similar to that of Wang *et al.* [WJST14] and

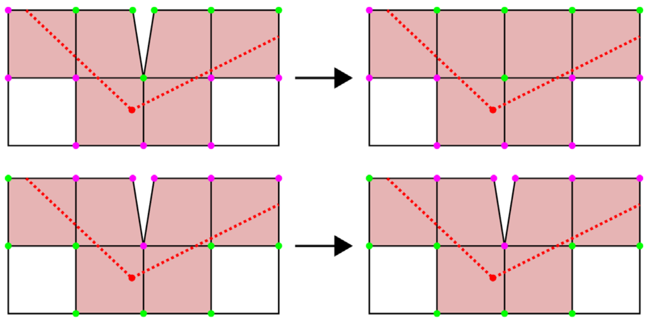


Figure 14: *Preliminary merge. The construction of the volumetric extension $\mathcal{U}$ may result in geometrically coincident vertices which do not come from topologically distant parts of the mesh. Green vertices have negative signs, while purple vertices have positive sign. Above: The process in Section 7.1 merges these vertices into a single vertex. Below: We do not merge coincident positive vertices, to avoid unnecessarily sewing the exterior.*

re-run the algorithm. We then use a topology-preserving coarsening strategy (see Section 8) to achieve the target resolution. We also note again that unlike Li and Barbič [LB18], we cannot handle non-simple immersions.

7. Interior Extension Region Merging

Having created the interior extensions $\mathcal{V}^{I,c}$, we merge these meshes with the volumetric extension $\mathcal{U}$ and with each other (to account for possible over-counting in their creation) in four main steps. We first merge hexahedra from $\mathcal{U}$ into $\mathcal{V}^{I,c}$ in a process described below. We then determine which of the interior extensions should merge to each other, using information from the previous step to generate lists of overlapping hexahedra between meshes of different regions and copies. Next, we use these lists to determine which copies of the same region are over-counts and merge the duplicates together. Finally, the lists are used to define the adjacencies between vertices for the final merge.

7.1. Merge with boundary

Recall from Section 6 that in regions I with multiple copies, we count the copies using connected components of negatively signed vertices in $\mathbf{x}^V$ geometrically coincident with nodes of I. We use $\mathcal{C}^{I,c}$ to denote this set of vertices for component c and region I. For regions with only one copy, we similarly define $\mathcal{C}^{I,0}$ to be the collection of all negatively signed vertices in $\mathbf{x}^V$ geometrically coincident with nodes in region I. Note that for these single copy regions, the vertices of $\mathcal{C}^{I,0}$ need not be connected (see the geometry of Figure 15, where the vertices $\mathcal{C}^{I,0}$ are composed of two connected components on the outer and inner boundaries). We perform a preliminary merge of vertices in $\mathcal{C}^{I,c}$ for each region I and copy c. Vertices i and j in $\mathbf{x}^V$ are adjacent if they are geometrically coincident and both in $\mathcal{C}^{I,c}$ for some I and c. This preliminary merge closes unwanted interior voids without 'sewing' the exterior or merging topologically distant vertices of $\mathcal{U}$ as shown in Figure 14.

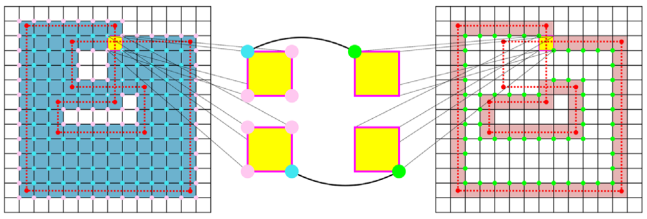


Figure 15: *Vertex adjacency. The merge process between vertices of $\mathcal{V}^{I,c}$ and $\mathcal{C}^{I,c}$. For the cell highlighted in yellow, there are two hexahedra from $\mathcal{V}^{I,c}$ and therefore four pairs of geometrically coincident vertices. The two negatively signed vertices (in green) from $\mathcal{C}^{I,c}$ are matched to the vertices which came from an interior connected component (marked in cyan) and not the ones which did not (marked in pink).*

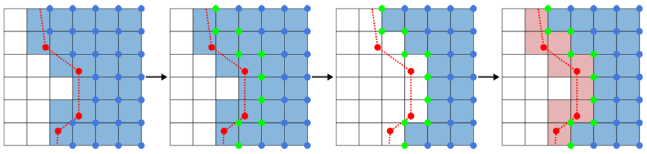


Figure 16: *Merge with boundary. We illustrate the process of Section 7.1 following the preliminary merge of negatively signed vertices. First, specific vertices of $\mathcal{V}^{j^I,c}$ are merged with vertices of $\mathcal{C}_c^{j^I}$. Next, hexahedra to be replaced are removed from the $\mathcal{V}^{j^I,c}$. Finally, copies of hexahedra from $\mathcal{U}$ are added to this mesh.*

The merge between the vertices of $\mathcal{V}^{I,c}$ and $\mathcal{C}^{I,c}$ is then defined by the following adjacency. Vertices i_1, i_2 in $\mathbf{x}^V$ are adjacent if

- i_2 and i_2 are geometrically coincident,
- i_1 is a vertex in $\mathcal{V}^{I,c}$ and i_2 is a vertex in $\mathcal{C}^{I,c}$,
- the connected component from Section 6 corresponding to i_1 contains a central vertex (defined in Section 6).

The last requirement means that vertices of $\mathcal{C}^{I,c}$ should only merge to the vertices of $\mathcal{V}^{I,c}$ which are actually interior to the region, and not those which are overlapping from a topologically far part of $\mathcal{V}^{I,c}$ as illustrated in Figure 15. After this merge has been performed, we update the indices in $\mathcal{C}^{I,c}$ accordingly as this set will be used in further steps of the merging procedure.

We next use a strategy different to that in Section 4.1 for merging hexahedra of $\mathcal{U}$ to their geometrically coincident counterparts in $\mathcal{V}^{I,c}$. This modified strategy is designed to prefer the structure of $\mathcal{U}$ over that of $\mathcal{V}^{I,c}$ (*e.g.* if two hexahedra of $\mathcal{U}$ are geometrically coincident but share only vertices on one face, then they will still have this connectivity after this merge). For copy c and region I, we denote the set of all hexahedra h of $\mathcal{U}$ incident to some $i \in \mathcal{C}^{I,c}$ by $\mathcal{H}^{I,c}$. Note that it is possible that some hexahedra of $\mathcal{U}$ are not included in any such set for any region I and copy c. We first remove hexahedra from $\mathbf{m}^{V^{I,c}}$ that are geometrically coincident with a hexahedron in $\mathcal{H}^{I,c}$ and incident to a vertex in $\mathcal{C}^{I,c}$. Finally, a copy of each hexahedra in $\mathcal{H}^{I,c}$ is added to $\mathbf{m}^{V^{I,c}}$. We outline this process in Figure 16.

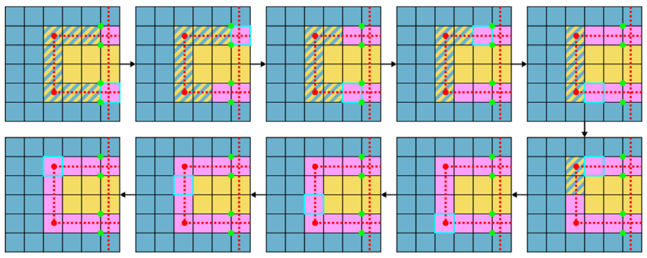


Figure 17: *Overlap lists. A closeup of the overlap region from the geometry of Figure 15 is shown here. At the upper left, the seeds for the overlap between the two copies are shown in purple, as well as the incident negative vertices (green) to the seeds from each copy. At each step, the current seed is marked with a cyan border. New geometrically coincident neighbours of the seed hexahedra are then added in the next step. When all seeds have been traversed, the process stops.*

7.2. Overlap lists

We next merge the meshes $\mathcal{V}^{I,c}$ of different regions along their appropriately defined common boundaries. The boundary region between any two meshes $\mathcal{V}^{I_1,c_1}$ and $\mathcal{V}^{I_2,c_2}$ is grown from initial seeds, which we define to be a pair of hexahedra in $\mathcal{V}^{I_1,c_1}$ and $\mathcal{V}^{I_2,c_2}$ that are equal to each other and to a hexahedron in $\mathcal{U}$. An initial seed is defined as a pair of hexahedra h_1, h_2 in $\mathcal{V}^{I_1,c_1}$ and $\mathcal{V}^{I_2,c_2}$, respectively, such that there exists a hexahedron h_3 in $\mathcal{U}$ with $\mathbf{m}_{h_1}^{\mathcal{V}^{I_1,c_1}} = \mathbf{m}_{h_2}^{\mathcal{V}^{I_2,c_2}} = \mathbf{m}_{h_2}^{\mathcal{U}}$. A pair of regions and copies I_1, c_1 and I_2, c_2 for which an initial seed exists will be denoted by $\mathbf{q} = (I_1, c_1, I_2, c_2)$; these are the pairs with common boundaries. We now construct an array $\mathbf{p}^{\mathbf{q}}$, which we call an overlap list, which represents the overlapping boundary between I_1, c_1 and I_2, c_2. We refer to the elements of this array as seeds and denote the ith seed by $\mathbf{s}_i^{\mathbf{q}}$. We initially set $\mathbf{p}^{\mathbf{q}}$ to be the array of all initial seeds and then grow the list using an iterative procedure:

- we first flag every grid cell geometrically coincident to the hexahedra in the seeds as visited.
- Starting from $\mathbf{s}_0^{\mathbf{q}} = (h_1, h_2)$, we compute the neighbour hexahedra of h_1 and h_2.
- If a neighbour of h_1 is geometrically coincident with a neighbour of h_2 and they are not geometrically coincident with a visited grid cell, the pair is appended to $\mathbf{p}^{\mathbf{q}}$ as a new seed and the geometrically coincident grid cell is marked as visited.
- We repeat this process on each subsequent seed in $\mathbf{p}^{\mathbf{q}}$ until all seeds are exhausted.

At the end of this expansion, $\mathbf{p}^{\mathbf{q}}$ is a list of overlapping hexahedra pairs that will be used to ‘sew’ the regions together. We illustrated this process in Figure 17.

7.3. Deduplication

As mentioned in Section 6, the number of copies is generally an over-count. We use the overlap lists $\mathbf{p}^{\mathbf{q}}$ to deduce which copies c of a region I are redundant. First, for each hexahedron h in $\mathcal{U}$, we construct a set D_h of regions and copies (I, c) as follows: for each pair $\mathbf{q} = (I_1, c_1, I_2, c_2)$ and seed $\mathbf{s}^{\mathbf{q}} = (h_1, h_2)$ in $\mathbf{p}^{\mathbf{q}}$, if $h = h_1$ or $h = h_2$

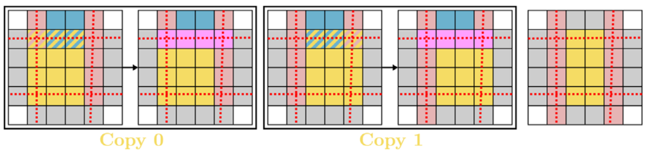


Figure 18: *Deduplication. We show two of the four copies of the central region (yellow), corresponding to the right and left segments of $\mathcal{U}$. Each copy creates an overlap list with the upper region (blue). The overlap list for copy 0 creates a pair between a non-boundary yellow hexahedron and a boundary hexahedron from the blue region. This boundary hexahedron is in a pair with a boundary hexahedron of copy 1, allowing us to deduce that copies 0 and 1 of the yellow region are duplicates.*

then both (I_1, c_1) and (I_2, c_2) are in D_h. This creates a set of regions and copies which have a hexahedra that is either equal to h or will subsequently merge with h. We note that the condition $h = h_1$ or $h = h_2$ is to account for the fact that not all seeds in $\mathbf{p}^{\mathbf{q}}$ are initial seeds, and as such one or both hexahedra in the seed may not be equal to hexahedra from $\mathcal{U}$. We next deduce if copies c_1 and c_2 of a region I are redundant (meaning that they represent the same copy) using the following criterion: the copies are redundant if (I, c_1) and (I, c_2) are both elements of D_h for some h. This process is shown in Figure 18.

We merge redundant copies using the process of Section 7.1. For each region I, we compute connected components of its copies using duplication as the notion of adjacency. For each connected component of copies, we take the copy c_i with the smallest index as the representative copy. As each redundant copy's mesh contains a different part of the boundary hexahedra from $\mathcal{U}$, we process the representative copy to contain the boundaries from all copies by repeating the merge with boundary process of Section 7.1 on updated data. Specifically, we replace C^{I,c_i} with the union of C^{I,c_j} where c_j ranges over the connected component of copies. We then form an updated collection of incident hexahedra $\mathcal{H}^{I,c_i}$ before repeating the boundary merge process. Finally, we update each overlap list; any overlap list corresponding to a duplicated copy is recreated using the representative copy to account for updated hexahedron ordering. Redundant overlap lists resulting from this update are then discarded.

7.4. Final merge

We now merge the vertices in $\mathbf{x}^V$ with adjacencies defined by the overlap lists: for each seed $\mathbf{s}_i^{\mathbf{q}}$ in an overlap list, the geometrically coincident nodes of the hexahedra pair in $\mathbf{s}_i^{\mathbf{q}}$ are adjacent. We then create the final mesh $\mathcal{V}$ by combining all of the arrays $\mathbf{m}^{\mathcal{V}^{I,c}}$ from copies which are either the minimum representative or not duplicated. Recall from Section 7.1 that some hexahedra of $\mathcal{U}$ are not copied into any copy's mesh. We add all such hexahedra to $\mathcal{V}$ to guarantee that $\mathcal{U}$ is contained in this final mesh, completing the interior extension region merging process.

8. Coarsening

For this section, it will be convenient to consider mesh arrays as flat arrays as in the example of Figure 4. Our method requires

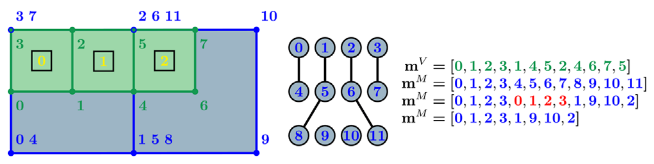

3 7
2 6 11
10
3
2
5
7
0
1
2
0
1
4
6
0 4
1 5 8
9
0 1 2 3
4 5 6 7
8 9 10 11
$\mathbf{m}^V = [0,1,2,3,1,4,5,2,4,6,7,5]$
$\mathbf{m}^M = [0,1,2,3,4,5,6,7,8,9,10,11]$
$\mathbf{m}^M = [0,1,2,3,0,1,2,3,1,9,10,2]$
$\mathbf{m}^M = [0,1,2,3,1,9,10,2]$

Figure 19: *Coarsening. An example of fine mesh connections. Hexahedra 0 and 1 are totally connected, while hexahedra 1 and 2 are connected by a face. After merging the vertices of the coarse mesh (blue), the duplicated hexahedron (indicated in red) is removed.*

high-resolution (small Δx) background grids for high-curvature/detailed surfaces. We provide a topology-aware coarsening strategy to provide user control over the final volumetric mesh resolution/element counts. After the hexahedron mesh $\mathcal{V}$ is created, we coarsen the underlying grid by doubling Δx. We then create a maximal coarse mesh $\mathcal{M}$ based on the fine mesh $\mathcal{V}$. For each index m^V_j in $\mathcal{V}$, we define the initial connectivity for $\mathcal{M}$ as $m^M_j = j$; *i.e.* every fine hexahedron h^V corresponds to a maximal coarse hexahedron h^M and none of the hexahedra h^M share vertices. For each maximal coarse hexahedron h^M, we bin the centre of the corresponding fine hexahedron h^V into the coarsened grid, *i.e.* we record the multi-dimensional grid index $\mathbf{i}^{h^M}$ of the grid cell containing the centre of h^V. We then initialize the position array $\mathbf{x}^M$ for $\mathcal{M}$ from the coarse grid cell corners of cell $\mathbf{i}^{h^M}$. Specifically, for each hexahedron h^M of $\mathcal{M}$, we define $\mathbf{x}^M_{8h^M+i^e} = \mathbf{x}^{2\Delta x}_{\mathbf{i}^{h^M}} + \mathbf{o}_{i^e}$, where $0 \leq i^e < 8$ indexes the corners of the hexahedron and $\mathbf{o}_{i^e}$ is an offset from the coarse cell centre $\mathbf{x}^{2\Delta x}_{\mathbf{i}^{h^M}}$ to corner i^e of the coarse grid cell $\mathbf{i}^{h^M}$. To build the final coarsened mesh, we merge portions of the maximal coarse mesh using the method of Section 4.1 where adjacencies are defined from a hexahedron-wise notion of connectivity as follows. Two maximal coarse hexahedra h^M_0 and h^M_1 are connected if their corresponding fine hexahedra h^V_0 and h^V_1 share a face in $\mathcal{V}$ (*i.e.* they share the four vertices on a face). We define two types of connection: totally connected and partially connected. Maximal coarse hexahedra are totally connected if they have the same coarse grid index $\mathbf{i}^{h^M_0} = \mathbf{i}^{h^M_1}$ and their corresponding fine hexahedra h^V_0 and h^V_1 are not geometrically coincident. Maximal coarse hexahedra are partially connected if they are connected but are not totally connected. We then define vertex adjacency from our notions of hexahedron connectivity. If two hexahedra h^M_0 and h^M_1 in the maximal coarse mesh are totally connected, then their eight respective geometrically coincident vertices are defined to be adjacent, *i.e.* vertex $m^M_{8h^M_0+i^e}$ is adjacent to vertex $m^M_{8h^M_1+i^e}$, $0 \leq i^e < 8$. If they are partially connected, then their corresponding fine hexahedra h^V_0, h^V_1 share a face. We then identify an analogous face in each of h^M_0 and h^M_1, and only the geometrically coincident vertices corresponding to the analogous face are adjacent. There are two cases that define the analogous face. First, if the corresponding fine hexahedra h^V_0, h^V_1 are geometrically coincident, then the analogous face is the one on the analogous side of the coarse hexahedron. If they are not geometrically coincident, then the analogous face is the one geometrically coincident with the face shared by the fine hexahedra. The general coarsening procedure is illustrated in Figure 19.

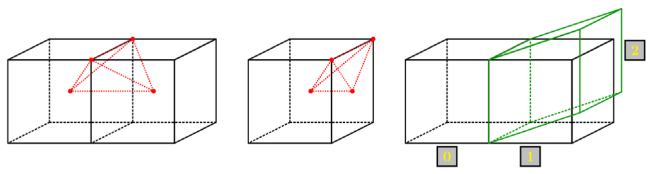


Figure 20: *Hexahedra tetrahedralization. (Left) A standard interior face in $\mathcal{V}$. The centres of the two incident hexahedra are combined with two face vertices to form the tetrahedra (red). (Middle) A standard boundary face uses a face centre instead of the missing incident hexahedron centre. (Right) A non-standard interior face is shown. The right-most incident hexahedra are geometrically coincident. We form hexahedra pairs/faces (0,1), (0,2) and treat them, respectively, as standard interior, as in the left-most image.*

9. Hexahedron Mesh To Tetrahedron Mesh Conversion

We design a topologically aware BCC-based approach for the creation of a tetrahedron mesh $\mathcal{T}$ from the hexahedron mesh $\mathcal{V}$. We initialize the particle array for the tetrahedron mesh $\mathbf{x}^T$ to be the same as $\mathbf{x}^V$, but we add a new vertex in the centre of each hexahedron and each boundary face. Tetrahedra are computed from the faces in the mesh $\mathcal{V}$. Normally, a face in $\mathcal{V}$ would have one (boundary face) or two (interior face) incident hexahedra. However, since $\mathcal{V}$ is comprised of many geometrically coincident hexahedra there are more cases. We classify them as standard boundary face (one incident hexahedron), standard interior face (two non-geometrically coincident incident hexahedra), non-standard interior (more than two incident hexahedra, some geometrically coincident and some not geometrically coincident) and non-standard boundary (more than one incident hexahedron, all geometrically coincident). Each face contributes four tetrahedra to $\mathcal{T}$ in the case of standard boundary and standard interior faces. The tetrahedra consist of two vertices from the face and the cell centres on either side of the face in the case of standard interior faces. In the case of standard boundary faces, the face centre is used in place of the second hexahedron centre. For non-standard interior faces, we take all pairs of non-geometrically coincident incident hexahedra and add tetrahedra as if their common face was a standard interior face. For non-standard boundary faces, tetrahedra are added for each incident hexahedron as if it were incident to a standard boundary face. We illustrate this procedure in Figure 20.

10. Examples

We consider a variety of examples in both two and three dimensions. To illustrate the capabilities of the final mesh connectivities, we treat the objects as deformable solids and run a finite element (FEM) simulation [SB12]. Performance statistics for the 3D examples are presented in Table 1. All experiments were run on a workstation with a single Intel® Core$^{\text{TM}}$ i9-10980XE CPU at 3.00 GHz.

10.1. 2D examples

10.1.1. *Ribbon*

Figure 21 shows a deformable FEM simulation using a volumetric mesh produced by our algorithm. One end of a ribbon shape passes

Table 1: *Performance of generating volumetric meshes using our algorithm for various 3D examples. All times are in seconds and represent the total runtime of the algorithm. Relative Δx is ratio of Δx to the shortest side length of the bounding box.*

Example	Grid dim.	Relative Δx	# Hex	Time (s)
Two boxes	$66 \times 64 \times 86$	0.0158730	256,368	2.80219
Simple overlap	$194 \times 64 \times 194$	0.0158730	1,606,296	24.0179
Double Möbius	$294 \times 288 \times 64$	0.0158730	903,653	33.6324
Twin bunnies	$162 \times 166 \times 128$	0.00787402	1,525,821	31.1815
Dragon	$512 \times 690 \times 520$	0.00195313	20,110,457	303.301
Fancy ball	$130 \times 132 \times 128$	0.00787402	515,400	25.8388
Head	$512 \times 830 \times 718$	0.00195313	62,444,819	839.951
Sacht	$52 \times 104 \times 42$	0.0243902	135,736	6.97091

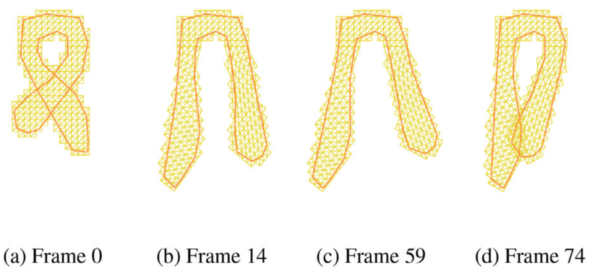

(a) Frame 0 (b) Frame 14 (c) Frame 59 (d) Frame 74

Figure 21: *A ribbon with a more complicated initial self-intersection is also treated properly by our method.*

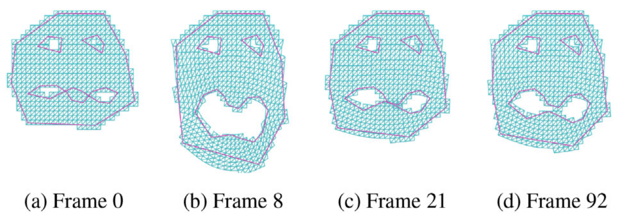

(a) Frame 0 (b) Frame 8 (c) Frame 21 (d) Frame 92

Figure 22: *A face with multiple boundary components and initially self-intersecting lips is successfully animated.*

through the other, partitioning the surface into several components. These intersections are successfully resolved, and the mesh is allowed to move without gluing.

10.1.2. *Face*

Figure 22 demonstrates a similar scenario. In this case, the lips of the face geometry initially overlap; and, as an added challenge, the boundary of the input geometry consists of multiple disconnected components. Our method successfully treats cases like these by design.

10.2. 3D examples

10.2.1. *Two boxes and simple overlap*

We begin our 3D examples by demonstrating that our algorithm is able to quickly generate consistent meshes for simple self-intersecting geometries. In Figure 23, basic hand-made geometries are allowed to separate and unfurl from their initial self-intersecting states. The two boxes in the left-hand side of each sub-figure were meshed using a background grid resolution of $66 \times 64 \times 86$ cells and $\Delta x = 0.00955671$, taking 2.80219 s to generate the resulting 256,368 hexahedra in the output mesh. The simple overlapping shape in the right-hand side of each sub-figure was meshed using a grid with $194 \times 64 \times 194$ cells and $\Delta x = 0.00328125$, resulting in 1,606,296 hexahedra in the output mesh.

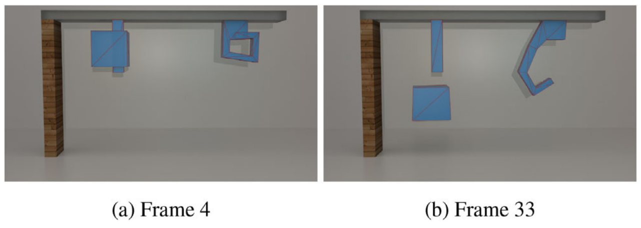
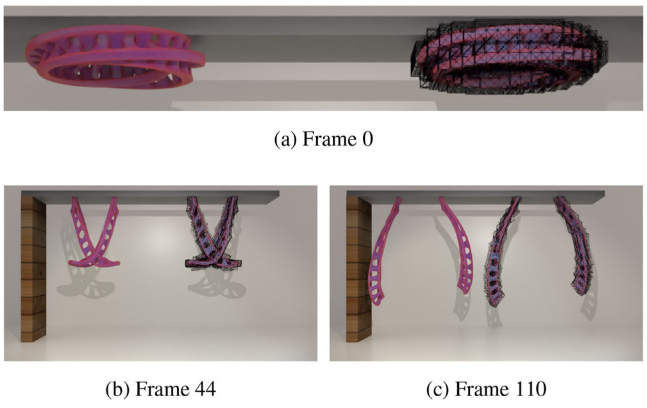

(a) Frame 4 (b) Frame 33

Figure 23: *Simple self-intersecting 3D geometries are able to separate and unfurl with our algorithm.*


(a) Frame 0

(b) Frame 44 (c) Frame 110

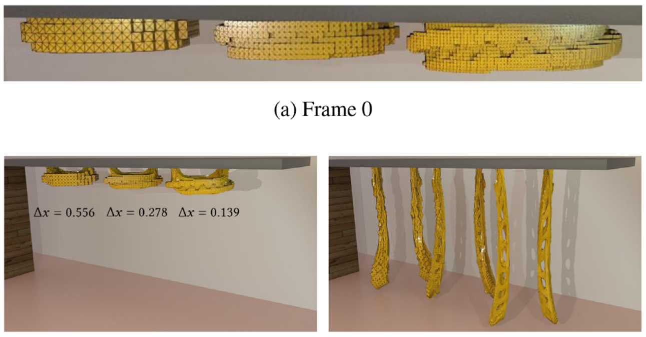
Figure 24: *Two intersecting Möbius-strip-like geometries (pink) naturally fall and separate under our method. The associated hexahedron meshes are shown in the right half of each frame.*

10.2.2. *Double Möbius*

Figure 24 shows two Möbius-strip-like geometries[1] falling and separating under the effects of gravity, despite substantial intersections at the start of the simulation. This example was run using a background grid with $294 \times 288 \times 64$ cells and a Δx of 0.0347391. The resulting hexahedron mesh has 903,653 elements. Generating the volumetric mesh using our algorithm takes 33.6324 s.

We also repeat this example at multiple spatial resolutions in order to demonstrate the effect of resolution on the quality of meshing results (see Figure 25). The coarsest grid (corresponding to the leftmost meshes in each sub-figure) is $21 \times 19 \times 5$ with $\Delta x = 0.556$. An intermediate grid resolution of $39 \times 37 \times 9$ cells with

[1] 'Mobius Bangle' by Creative_Hacker is licensed under CC BY 4.0.

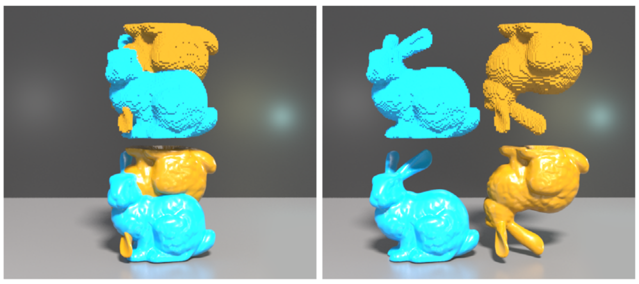


(a) Frame 0

(b) Frame 16

(c) Frame 84

Figure 25: *Running the example shown in Figure 24 at different spatial resolutions. In each frame, from left to right, the background grids have $\Delta x = 0.556$, 0.278 and 0.139.*



(a) Frame 1

(b) Frame 54

Figure 26: *Two overlapping bunnies naturally separate. The top part of each sub-figure shows the meshes generated by our algorithm, while the bottom part of each sub-figure shows the corresponding surface meshes.*

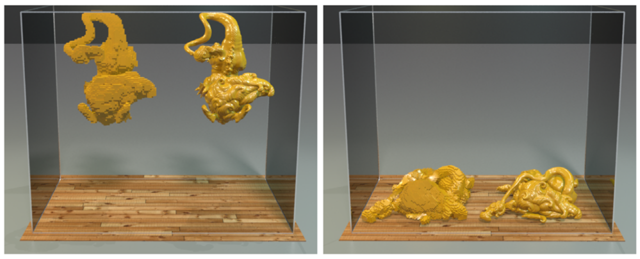
$\Delta x = 0.278$ corresponds to the middle meshes in each sub-figure. The rightmost meshes in each sub-figure come from using a grid with $75 \times 73 \times 17$ cells with $\Delta x = 0.139$. Proper separation is achieved at all three of these tested resolutions, and in particular, our algorithm performs quite well on this example even at extremely low spatial resolution.

10.2.3. *Twin bunnies*

Another standard example is the Stanford bunny. Figure 26 demonstrates that two almost completely overlapping bunny meshes can naturally separate under our method. No issues are encountered as different segments of the bunnies pass through one another. This example uses a grid resolution of $162 \times 166 \times 128$ cells with $\Delta x = 0.0203027$, resulting in a mesh with 1,525,821 hexahedra.

10.2.4. *Dragon*

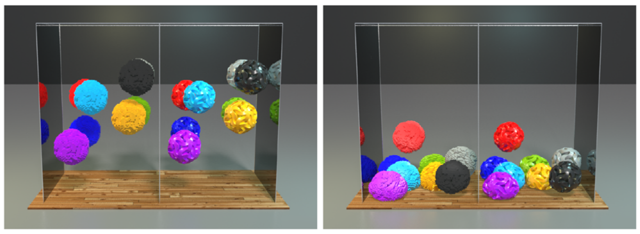
The most complicated geometry we test our method on is the dragon[2] shown in Figure 27 (and also shown in Figure 9). Adequate resolution is required in order to resolve all the fine-scale features of this mesh; accordingly, we use a grid resolution of $512 \times 690 \times 520$ cells with $\Delta x = 0.0708709$. Our final mesh, generated in 5 min, contains just over 20 million hexahedra.



(a) Frame 0

(b) Frame 300

Figure 27: *A complex mesh of a dragon is allowed to fall under gravity. The left-hand side of each subfigure shows the deforming mesh we generate, and each right-hand side shows the corresponding surface mesh.*



(a) Frame 20

(b) Frame 80

Figure 28: *Several ball-like geometries with intricate slices and holes are successfully meshed with our algorithm and then deform and collide under an FEM simulation.*

10.2.5. *Fancy ball*

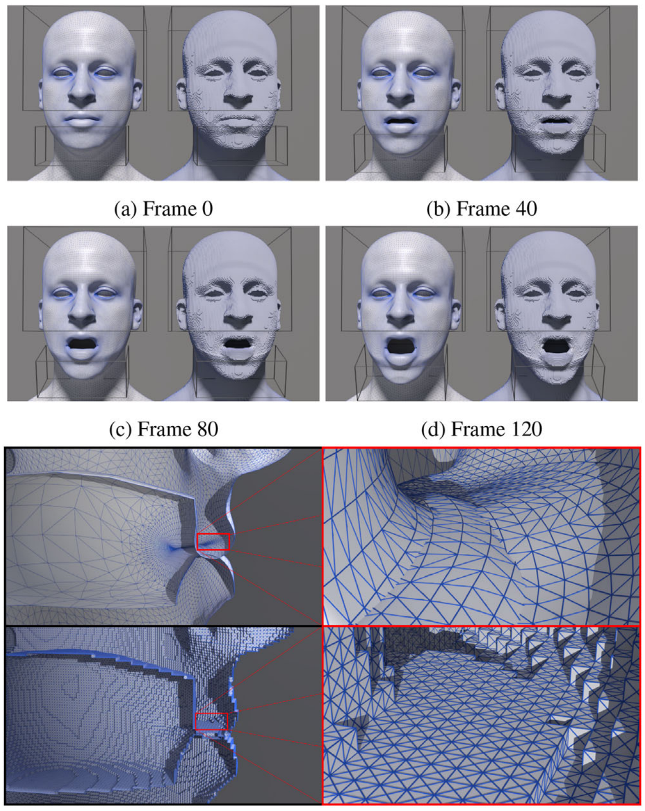
Figure 28 shows another interesting case where several ball-like geometries[3] deform and collide after being meshed with our algorithm. Each ball has a number of thin cuts and fine-scale features, which our algorithm is able to resolve using a grid with $130 \times 132 \times 128$ cells and $\Delta x = 2.82671$. The 515,400 resulting hexahedra are generated in 25.8388 s.

10.2.6. *Head*

Modelling of the human body often gives rise to self-intersection. This is particularly common in the faces, where lip geometries often self-intersect. To that end, we consider a real-world head geometry in Figure 29. Note that the lips separate effectively. This example results in a volumetric mesh with over 62 million elements, using a background grid resolution of $512 \times 830 \times 718$ cells and $\Delta x = 0.000501962$. Generating the hexahedron mesh takes 839.951 s.

[2] 'Asian Dragon' by Lalo-Bravo.

[3] 'Abstract object' by sonic art.

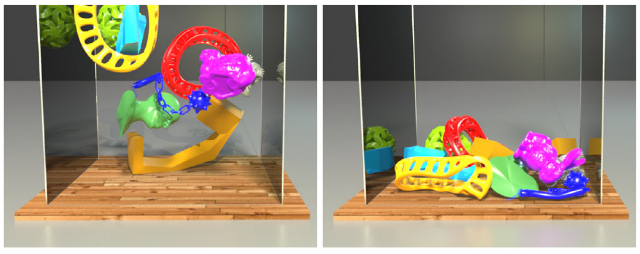


(a) Frame 0 (b) Frame 40

(c) Frame 80 (d) Frame 120

(e) Interior view of lips

Figure 29: *A face surface with self-intersecting lips is successfully meshed. The right-hand side of each of the first four frames shows the deformed hexahedron mesh, while each left-hand side shows the corresponding surface mesh. The wireframe boxes represent Dirichlet boundary condition regions. In the bottom four sub-figures, lip intersection is visualized in the input surface and subsequent hexahedron mesh.*

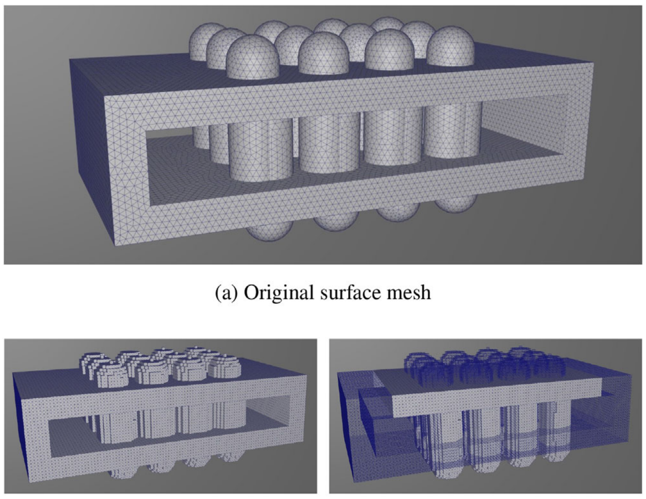


(a) Frame 80 (b) Frame 200

Figure 30: *We simulated dropping our 3D examples into a box with a FEM sim.*

10.2.7. *Collection*

Various objects from 3D examples are dropped in a tank in Figure 30. The objects naturally deform and collide without meshing or simulation issues.



(a) Original surface mesh

(b) Output tetrahedral mesh (c) A contiguous subsection

Figure 31: *Our method can successfully resolve the self intersecting geometry proposed in Sacht* et al. *[SJP*13]. We visualize the surface of the output after conversion to a tetrahedron mesh. We emphasize a sub-section of the mesh on the right sub-figure; the bristles are attached to the correct parts of the torus and are not connected to the bristles from the opposite side of the torus.*

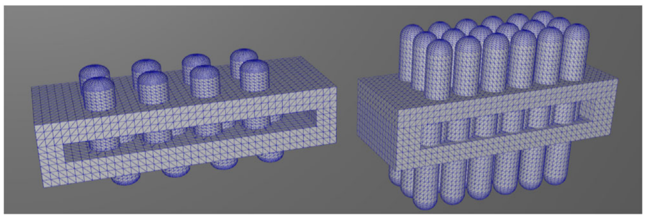


Figure 32: *Versions of the Sacht geometry from VegaFEM. Left is torus-easy, right is torus-difficult.*

10.2.8. *Sacht et al. mesh*

Finally, we demonstrate that our method, like that of Li and Barbič [LB18], can successfully resolve the self intersections of the geometry shown in Figure 31 that is not supported by the method of Sacht *et al.* [SJP*13]. In Sacht *et al.* [SJP*13], the bristles in this geometry get locked by the surrounding torus. However, both our method and Li and Barbič [LB18] properly resolve all self-intersections. We compare our method against the implementation of Li and Barbič [LB18] in the VegaFEM library [BSS12] using two versions of the Sacht mesh provided in VegaFEM, torus-easy and torus-difficult. These meshes are shown in Figure 32. Li and Barbič's method takes 14.773 s with 112,554 (output) tetrahedra and 32.845 s with 225,338 tetrahedra for torus-easy and torus-difficult, respectively. Our method takes 4.708 s with 120,591 (output) hexahedra and 12.949 s with 224,083 hexahedra. We also compare our method on the non-torus helix and beam meshes provided in VegaFEM. The helix is highly self intersecting, with every loop intersecting the previous and next loops. Our method takes 59.973 s with 60,925 hexahedra while their method takes 123.758 s with 36,184 tetrahedra. The beam geometry is simply a rectangular box with no self-intersections. Our method takes 0.145 s with 495 hexahedra while their method takes 0.255 s with 450 tetrahedra. See Gagniere *et al.* [GHC*23] for visualizations of these two geometries. Our method runs noticeably faster as nearly all steps are designed to be highly parallelizable. We additionally note that our method takes only a triangle mesh as input whereas VegaFEM takes a triangle mesh and a glued tetrahedron mesh.

11. Discussion and Limitations

Our method has various limitations, most of which are attributed to our reduced use of exact/adaptive precision arithmetic. The most prominent limitations of our approach are in the types of input surface mesh $\mathcal{S}$ that we support. Fine-scale features, *e.g.* thin parallel sheets, can cause negatively signed vertices to be located in regions of the grid corresponding to an incorrect region. In these pathological cases, the output mesh will have undesirable extraneous collections of hexahedra. We present a 2D example of this failure in Section A of the supplementary document [GHC*23]. Such cases result from a background grid which is too coarse relative to the size of polygonal faces in the finer regions. Hence, we resolve this by iteratively halving the cell width until fine scale features can be accurately resolved. We note that such a heuristic strategy may result in an undesirable level of refinement. However, our coarsening approach is designed to mitigate this. Even using added resolution and subsequent coarsening, our methodological simplifications prevent us from handling certain classes of cases that Li and Barbič [LB18] can handle, *e.g.* we cannot resolve non-simple immersions. It would be interesting to investigate whether our minimal-exact-arithmetic approach could be extended to handle non-simple immersions as well. A further failure case arises from one surface mesh being fully contained within the volume defined by a larger surface mesh. Our method will generally fail to create enough copies of the smaller volume. This is not a serious limitation, however, as our focus is on self-intersection and such situations can easily be avoided by translating separate meshes by a sufficient amount. Future work includes improvements to the algorithm to handle known pathological cases without the need for refinement and subsequent coarsening, as well as improved detection mechanisms for such cases. In particular, additional use of exact arithmetic in the form of segment-triangle intersections should allow for the detection of fine scale signing failures. It should be possible to combine such a detection mechanism with an adaptive refinement/coarsening scheme to create an intermediate step which eliminates the need for heuristic global refinement and allows for graceful failure (*i.e.* when a refinement limit is reached).

Lastly, Figure 2 illustrates an interesting case which neither our approach, that of Li and Barbič [LB18] nor that of Sacht *et al.* [SJP*13] can handle. In this case, which is common near *e.g.* elbows and even shoulders in an upper torso, a portion of the domain overlaps in such a way that $\boldsymbol{\phi}^{S}_{\tilde{S}}$ must have negative Jacobian determinant in some regions. Our approach returns a mesh for this case, but it does not properly copy the overlap region and one of the two copies that would be required is rejected. In other words, our approach does not give a result consistent with creating a mesh in $\tilde{\mathcal{S}}^V$ and pushing it forward under $\boldsymbol{\phi}^{S}_{\tilde{S}}$. In Li and Barbič [LB18], this is noted as a case for which an immersion does not exist and Sacht *et al.* [SJP*13] explicitly require the Jacobian determinant of $\boldsymbol{\phi}^{S}_{\tilde{S}}$ to be non-negative. However, this is a commonly occurring case which would be beneficial to resolve.

Acknowledgements

The authors have nothing to report.

References

[ACWK06] Angelidis A., Cani M.-P., Wyvill G., King S.: Swirling-sweepers: Constant-volume modeling. *Graph. Models 68*, 4 (2006), 324–332.

[Att10] Attene M.: A lightweight approach to repairing digitized polygon meshes. *Vis. Comput. 26*, 11 (2010), 1393–1406.

[BB99] Belytschko T., Black T.: Elastic crack growth in finite elements with minimal remeshing. *Int. J. Num. Meth. Engr. 45*, 5 (1999), 601–620.

[BDS*18] Barill G., Dickson N., Schmidt R., Levin D., Jacobson A.: Fast winding numbers for soups and clouds. *ACM Trans. Graph. 37*, 4 (2018), 1–12.

[Béz70] Bézier P.: *Numerical Control: Mathematics and Applications*. Masson & Cie, Paris (1970).

[BFG*20] Brönnimann H., Fabri A., Giezeman G.-J., Hert S., Hoffmann M., Kettner L., Pion S., Schirra S.: 2D and 3D linear geometry kernel. In *CGAL User and Reference Manual* (5.2 edition). CGAL Editorial Board (2020). https://doc.cgal.org/5.2/Manual/packages.html#PkgKernel23

[Bla67] Blank S.: Extending Immersions of the Circle. PhD thesis, Brandeis University, 1967.

[BSMF20] Botsch M., Sieger D., Moeller P., Fabri A.: Surface mesh. In *CGAL User and Reference Manual* (5.2 edition). CGAL Editorial Board (2020). https://doc.cgal.org/5.2/Manual/packages.html#PkgSurfaceMesh

[BSS12] Barbič J., Sin F. S., Schroeder D.: Vega FEM Library. http://www.jernejbarbic.com/vega (2012). Accessed 2023.

[BWS*09] Brunton A., Wuhrer S., Shu C., Bose P., Demaine E.: Filling holes in triangular meshes by curve unfolding. In *Proceedings of the 2009 IEEE International Conference on Shape Modeling and Applications* (2009), pp. 66–72. https://doi.org/10.1109/SMI.2009.5170165

[CBE*15] Cong M., Bao M., Jane L. E., Bhat K. S., Fedkiw R.: Fully automatic generation of anatomical face simulation models. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2015), pp. 175–183.

[CBF16] Cong M., Bhat L., Fedkiw R.: Art-directed muscle simulation for high-end facial animation. In *Proceedings of the 2016 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2016), Eurographics Association, pp. 119–127.

[DCB13] DORAN C., CHANG A., BRIDSON R.: Isosurface stuffing improved: Acute lattices and feature matching. In *ACM SIGGRAPH 2013 Talks* (2013).

[EB14] EDWARDS E., BRIDSON R.: Detailed water with coarse grids: Combining surface meshes and adaptive discontinuous Galerkin. *ACM Trans Graph 33*, 4 (2014), 136:1–136:9.

[EFW20] EVANS P., FASY B., WENK C.: Combinatorial properties of self-overlapping curves and interior boundaries. In *Proceedings of the 36th International Symposium on Computational Geometry (SoCG 2020), Leibniz International Proceedings in Informatics (LIPIcs)* (Dagstuhl, Germany, 2020), S. Cabello, D. Z. Chen (Eds.). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, volume *164*, pp. 41:1–41:17. https://drops.dagstuhl.de/opus/volltexte/2020/12199, https://doi.org/10.4230/LIPIcs.SoCG.2020.41

[EM09] EPPSTEIN D., MUMFORD E.: Self-overlapping curves revisited. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms* (2009), SIAM, pp. 160–169.

[FTS06] FUNCK W. V., THEISEL H., SEIDEL H.-P.: Vector field based shape deformations. *ACM Trans. Graph. 25*, 3 (2006), 1118–1125.

[GC11] GRAVER J., CARGO G.: When does a curve bound a distorted disk? *SIAM Journal on Discrete Mathematics 25*, 1 (2011), 280–305.

[GCX*20] GAO J., CHEN W., XIANG T., JACOBSON A., MCGUIRE M., FIDLER S.: Learning deformable tetrahedral meshes for 3D reconstruction. In *Advances in Neural Information Processing Systems* (2020), H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Eds.), Curran Associates, Inc., vol. *33*, pp. 9936–9947. https://proceedings.neurips.cc/paper/2020/file/7137debd45ae4d0ab9aa953017286b20-Paper.pdf

[GD01] GAIN J., DODGSON N.: Preventing self-intersection under free-form deformation. *IEEE Trans Viz Comp Grap 7*, 4 (2001), 289–298.

[GHC*23] GAGNIERE S., HAN Y., CHEN Y., HYDE D., MARQUEZ-RAZON A., TERAN J., FEDKIW R.: Robust Grid-Based Meshing for Self-Intersections: Supplementary Document. *Tech. Rep.* 2023.

[HL95] HU Z.-J., LING Z.-K.: Geometric modeling of a moving object with self-intersection. In *Proceedings of the International Design Engineering Technical Conferences and Computers and Information in Engineering Conference* (1995), American Society of Mechanical Engineers, vol. *17162*, pp. 141–148.

[HPSZ11] HARMON D., PANOZZO D., SORKINE O., ZORIN D.: Interference-aware geometric modeling. *ACM Transactions on Graphics (TOG) 30*, 6 (2011), 1–10.

[HSW*20] HU Y., SCHNEIDER T., WANG B., ZORIN D., PANOZZO D.: Fast tetrahedral meshing in the wild. *ACM Trans. Graph. 39*, 4 (2020), 117:1–117:18.

[HT89] HORN W., TAYLOR D.: A theorem to determine the spatial containment of a point in a planar polyhedron. *Comp Vis Graph Imag Proc 45*, 1 (1989), 106–116.

[HZG*18] HU Y., ZHOU Q., GAO X., JACOBSON A., ZORIN D., PANOZZO D.: Tetrahedral meshing in the wild. *ACM Trans. Graph. 37*, 4 (July 2018), 60:1–60:14. https://doi.org/10.1145/3197517.3201353

[JAYB15] JAMIN C., ALLIEZ P., YVINEC M., BOISSONNAT J.-D.: CGALmesh: A generic framework for Delaunay mesh generation. *ACM Trans. Math. Soft. 41*, 4 (2015), 1–24.

[JKSH13] JACOBSON A., KAVAN L., SORKINE-HORNUNG O.: Robust inside-outside segmentation using generalized winding numbers. *ACM Trans. Graph. 32*, 4 (2013), 1–12.

[KBT17] KOSCHIER D., BENDER J., THUEREY N.: Robust extended finite elements for complex cutting of deformables. *ACM Trans Graph 36*, 4 (2017), 55:1–55:13. https://doi.org/10.1145/3072959.3073666

[KSBC12] KAZHDAN M., SOLOMON J., BEN-CHEN M.: Can mean-curvature flow be modified to be non-singular? *Comput. Graph. Forum 31*, 5 (Aug. 2012), 1745-1754. https://doi.org/10.1111/j.1467-8659.2012.03179.x

[KT10] KIM H.-J., TAUTGES T.: EBMesh: An embedded boundary meshing tool. In *Proceedings of the 19th International Meshing Roundtable* (Berlin, Heidelberg, 2010), S. Shontz (Ed.), Springer, Berlin Heidelberg, pp. 227–242.

[LB18] LI Y., BARBIČ J.: Immersion of self-intersecting solids and surfaces. *ACM Trans. Graph. 37*, 4 (July 2018). https://doi.org/10.1145/3197517.3201327

[Li11] LI W.: Detecting ambiguities in 3D polygons with self-intersecting projections. In *Proceedings of the 2011 12th International Conference on Computer-Aided Design and Computer Graphics* (2011), pp. 11–16. https://doi.org/10.1109/CAD/Graphics.2011.31

[LRLTY20] LORIOT S., ROUXEL-LABBÉ M., TOURNOIS J., YAZ I.: Polygon mesh processing. In *CGAL User and Reference Manual* (5.2 edition). CGAL Editorial Board, 2020. https://doc.cgal.org/5.2/Manual/packages.html#PkgPolygonMeshProcessing

[LS07] LABELLE F., SHEWCHUK J.: Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles. In *SIGGRAPH'07: ACM SIGGRAPH 2007* (New York, NY, USA, 2007), ACM, pp. 57. https://doi.org/10.1145/1275808.1276448

[Mar74] MARX M.: Extensions of normal immersions of $\mathcal{S}^1$ into $\mathcal{R}^2$. *Transactions of the American Mathematical Society 187* (1974), 309–326.

[MASS15] MITCHELL N., AANJANEYA M., SETALURI R., SIFAKIS E.: Non-manifold level sets: A multivalued implicit surface representation with applications to self-collision processing. *ACM Trans. Graph. 34*, 6 (Oct. 2015). https://doi.org/10.1145/2816795.2818100

[MBF03] Molino N., Bridson R., Fedkiw R.: Tetrahedral mesh generation for deformable bodies. In *Proceedings of the Symposium on Computer Animation* (2003), pp. 8.

[MBF04] Molino N., Bao Z., Fedkiw R.: A virtual node algorithm for changing mesh topology during simulation. *ACM Trans Graph 23*, 3 (2004), 385–392. https://doi.org/10.1145/1015706.1015734

[MBTF03] Molino N., Bridson R., Teran J., Fedkiw R.: A crystalline, red green strategy for meshing highly deformable objects with tetrahedra. In *Proceedings of the International Meshing Roundtable Conference* (2003), Citeseer, pp. 103–114.

[Muk14] Mukherjee U.: Self-overlapping curves: Analysis and applications. *Computer-Aided Design 46* (2014), 227–232.

[OF03] Osher S., Fedkiw R.: *Level Set Methods and Dynamic Implicit Surfaces*. Applied Mathematical Science. Springer, New York, N.Y., 2003.

[SB09] Song J.-H., Belytschko T.: Cracking node method for dynamic fracture with finite elements. *Int. J. Num. Meth. Engr. 77*, 3 (2009), 360–385.

[SB12] Sifakis E., Barbic J.: Fem simulation of 3D deformable solids: A practitioner's guide to theory, discretization and model reduction. In *SIGGRAPH'12: Proceedings of the ACM SIGGRAPH 2012 Courses* (New York, NY, USA, 2012), ACM, pp. 20:1–20:50. https://doi.org/10.1145/2343483.2343501

[SDF07] Sifakis E., Der K., Fedkiw R.: Arbitrary cutting of deformable tetrahedralized objects. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2007), pp. 73–80.

[Si15] Si H.: TetGen, a Delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Softw. 41*, 2 (Feb. 2015). https://doi.org/10.1145/2629697

[SJP*13] Sacht: Consistent volumetric discretizations inside self-intersecting surfaces. In *SGP'13: Proceedings of the Eleventh Eurographics/ACMSIGGRAPH Symposium on Geometry Processing* (Goslar, DEU, 2013), Eurographics Association, pp. 147–156. https://doi.org/10.1111/cgf.12181

[SOS04] Shen C., O'Brien J., Shewchuk J.: Interpolating and approximating implicit surfaces from polygon soup. In *SIGGRAPH'04: Proceedings of the ACM SIGGRAPH 2004 Papers* (New York, NY, USA, 2004), Association for Computing Machinery, pp. 896–904. https://doi.org/10.1145/1186562.1015816

[SP86] Sederberg T., Parry S.: Free-form deformation of solid geometric models. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques* (1986), pp. 151–160.

[SV92] Shor P., Van Wyk C.: Detecting and decomposing self-overlapping curves. *Computational Geometry 2*, 1 (1992), 31–50. https://www.sciencedirect.com/science/article/pii/092577219290019O

[TBFL19] Tao M., Batty C., Fiume E., Levin D.: Mandoline: Robust cut-cell generation for arbitrary triangle meshes. *ACM Trans. Graph. 38*, 6 (Nov. 2019). https://doi.org/10.1145/3355089.3356543

[The20] The CGAL Project: *CGAL User and Reference Manual* (5.2 edition). CGAL Editorial Board, 2020. https://doc.cgal.org/5.2/Manual/packages.html

[Tit61] Titus C.: The combinatorial topology of analytic functions of the boundary of a disk. *Acta Mathematica 106*, 1-2 (1961), 45–64.

[TSB*05] Teran J., Sifakis E., Blemker S., Ng-Thow-Hing V., Lau C., Fedkiw R.: Creating and simulating skeletal muscle from the visible human data set. *IEEE Trans Vis Comp Graph 11*, 3 (2005), 317–328.

[WDG*19] Wang S., Ding M., Gast T., Zhu L., Gagniere S., Jiang C., Teran J.: Simulation and visualization of ductile fracture with the material point method. In *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, ACM, vol. *2*, pp. 18.

[WJST14] Wang Y., Jiang C., Schroeder C., Teran J.: An adaptive virtual node algorithm with robust mesh cutting. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2014), Eurographics Association, pp. 77–85.

[WWD15] Wu J., Westermann R., Dick C.: A survey of physically based simulation of cuts in deformable bodies. *Comp Graph Forum 34*, 6 (2015), 161–187. https://doi.org/10.1111/cgf.12528

[ZDZ*18] Zhang J., Duan F., Zhou M., Jiang D., Wang X., Wu Z., Huang Y., Du G., Liu S., Zhou P., Shang X.: Stable and realistic crack pattern generation using a cracking node method. *Front. Comp. Sci. 12*, 4 (2018), 777–797.

Supporting Information

Additional supporting information may be found online in the Supporting Information section at the end of the article.

Supporting Information