Non-manifold Level Sets: A multivalued implicit surface representation with applications to self-collision processing

Nathan Mitchell Mridul Aanjaneya Rajsekhar Setaluri Eftychios Sifakis
University of Wisconsin-Madison

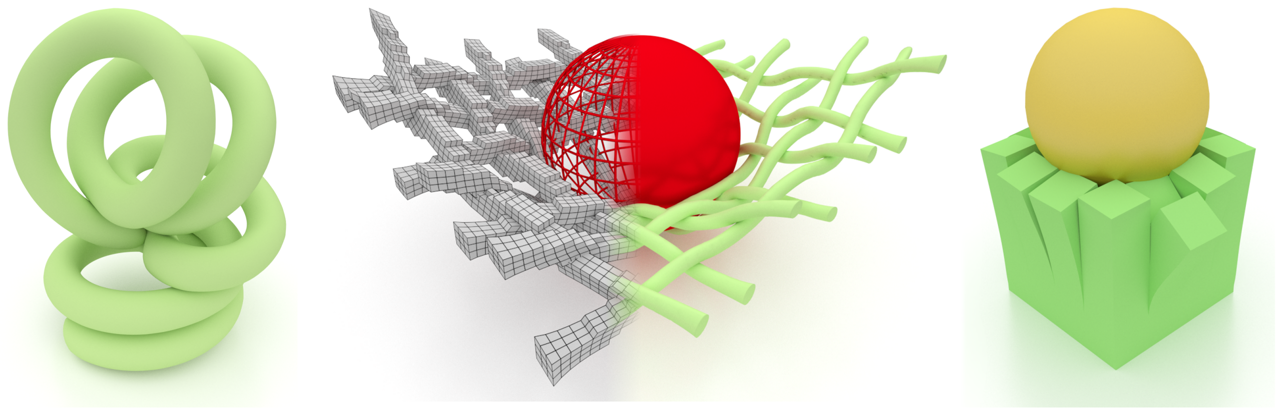


Figure 1: *(Left) A self-colliding coil. (Middle) The level set of a deformable net (shown colliding with a rigid sphere) is stored in a single mesh structure. (Right) A partially sliced elastic cube is impacted by a rigid ball to demonstrate robust collision handling of zero-width cuts.*

Abstract

Level sets have been established as highly versatile implicit surface representations, with widespread use in graphics applications including modeling and dynamic simulation. Nevertheless, level sets are often presumed to be limited, compared to explicit meshes, in their ability to represent domains with thin topological features (e.g. narrow slits and gaps) or, even worse, material overlap. Geometries with such features may arise from modeling tools that tolerate occasional self-intersections, fracture modeling algorithms that create narrow or zero-width cuts by design, or as transient states in collision processing pipelines for deformable objects. Converting such models to level sets can alter their topology if thin features are not resolved by the grid size. We argue that this ostensible limitation is not an inherent defect of the implicit surface concept, but a collateral consequence of the standard Cartesian lattice used to store the level set values. We propose storing signed distance values on a regular hexahedral mesh which can have multiple collocated cubic elements and non-manifold bifurcation to accommodate non-trivial topology. We show how such non-manifold level sets can be systematically generated from convenient alternative geometric representations. Finally we demonstrate how this representation can facilitate fast and robust treatment of self-collision in simulations of volumetric elastic deformable bodies.

CR Categories: I.3.7 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling;

Keywords: level sets, collision processing, deformable models

1 Introduction

Modeling of geometric objects is one of the fundamental challenges of computer graphics and visual computing at large. Nearly three decades after their introduction [Osher and Sethian 1988], level sets have evolved into one of the most widely used representations of geometry, alongside traditional alternatives such as meshes, splines and subdivision surfaces. A level set implicitly represents a domain boundary $\mathcal{S} = \partial\Omega$ as the zero-value isosurface (i.e. zero *level set*)

$$\mathcal{S} = \{\vec{x} \in \mathbf{R}^n \mid \phi(\vec{x}) = 0\} \qquad (1)$$

of a scalar field $\phi(\vec{x})$ measuring signed distances to the boundary of the object $\Omega \subset \mathbf{R}^n$. Level sets allow for fast $O(1)$ time point-object intersection queries or point projections to the object surface. Model deformation is also possible, including topological split and merge operations, simply by varying the underlying scalar field. Level sets are used in a diverse range of applications including surface editing [Museth et al. 2002], tetrahedral meshing [Labelle and Shewchuk 2007], scattered point interpolation [Zhao et al. 2001], fluid simulation [Osher and Fedkiw 2002] and collision processing for deformable solids [Gascuel 1993], rigid bodies [Guendelman et al. 2003] and skinning animations [Vaillant et al. 2013].

In principle, based on equation (1) a level set could represent any object $\Omega \subset \mathbf{R}^n$. In practice, however, the scalar field $\phi(\vec{x})$ is never provided analytically, but sampled instead at discrete points in $\mathbf{R}^n$. As a consequence, the expressive ability of discrete level sets is limited by the sampling resolution and the interpolation scheme used. In the common practice where ϕ values are sampled on the nodes of a uniform Cartesian grid, and trilinear interpolation is used to define a continuous scalar field, models with multiple boundary crossings per grid edge (near narrow gaps or strips, see Figure 2) cannot be represented. These issues can be alleviated to some extent by using adaptive schemes [Losasso et al. 2004; Museth 2013] to concentrate resolution near fine features, or hybridizing with point-based methods [Enright et al. 2002] to capture details at a sub-cell level. Nevertheless, gratuitously increasing the level set sampling resolution is a brute-force remedy which quickly becomes impractical if the thickness of topological gaps approaches zero, as is commonly the case with geometries arising from cutting and fracture modeling pipelines (see Figure 8(top)). It is also unfortunate that even though

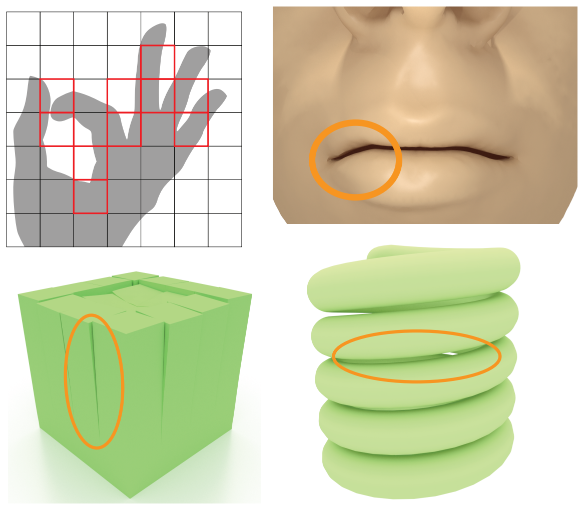


Figure 2: *Scenarios where standard Cartesian grid-based level sets lack the expressive ability to resolve thin features. For the vector art hand (top left), the cells highlighted in red show features that will not be resolved. While many cases can be resolved with fine enough resolutions, the fractured cube (bottom left) is an instance that cannot be resolved with conventional level sets.*

level sets are perfectly capable of localizing the implicit surface to sub-cell resolution (trilinearly interpolated level sets on Cartesian grids converge quadratically to surfaces of bounded curvature) they cannot resolve multiple interface crossings within a single cell.

We argue that these apparent limitations of level sets are not intrinsic defects of the implicit representation (equation 1), but consequences of the data structure (e.g. Cartesian grid) conventionally used to store the signed distance values. Instead of using $\mathbf{R}^n$ as the domain of $\phi(\vec{x})$, we propose to define this scalar field over an explicit quadrilateral (2D) or hexahedral (3D) mesh. We use regular (square or cube) elements in these meshes, identical in shape to the cells of a conventional Cartesian grid. However, the explicit connectivity in our mesh allows us to have multiple overlapping elements associated with geodesically distant regions (see Figure 4). Furthermore, this enables us to introduce non-manifold connectivity to capture topological bifurcation at the tip of a crack or incision, or in the vicinity of highly concave regions (see Figure 5).

The algorithmic interface to an implicit surface data structure is highly dependent on the application it caters to. A fluid simulation application would likely mandate support for dynamic evolution of the interface. Geometry processing tasks may be more dependent on contouring or differential properties such as curvature. The data structure we propose, which is essentially a multivalued signed distance field, might require a different interface depending on the end application. For example, applications in multiphase fluids [Losasso et al. 2006], multi-material surface tracking [Da et al. 2014] or volumetric meshing [Sacht et al. 2013] may contribute their own interpretations to what a non-manifold feature may be and what algorithmic routines need to be supported. In order to avoid such ambiguities we focus our scope to a specific driving application: collision detection and penalty-based response on volumetric simulation of elastic solids. In section 3 we review how this specific application utilizes conventional (manifold) level sets, identify scenarios where the limitations of the traditional formulation hinder the efficacy of collision processing or even make it inapplicable, and explain how the non-manifold level set concept can alleviate those obstacles. In summary, our key contributions are as follows:

- We extend level set representations by storing discrete values on nodes of a non-manifold Cartesian grid, enabling them to represent domains with thin gaps or slivers, and even encode boundaries of overlapping material domains.
- We detail a systematic pipeline for converting mesh-based geometry representations into multivalued level sets, and discuss how other geometric descriptions could also be used as input.
- We extend a number of algorithmic concepts and geometric predicates, such as signed distance initialization and closest-point projection to the non-manifold level set paradigm.
- We show how non-manifold level sets can broaden the scope of penalty-based self-collision processing for elastic solids.

2 Previous work

Level set methods were first introduced by Osher and Sethian [1988] for tracking moving interfaces in the context of Hamilton-Jacobi equations. Subsequently, Adalsteinsson and Sethian [1994] proposed substantial runtime savings by restricting computations to a thin band of active voxels near the interface. Sethian [1998] proposed fast marching methods for monotonically advancing fronts as well as for redistancing the level set using values seeded only on the narrow band. Besides fast computation, a number of methods have also been proposed for efficiently storing level sets including octrees [Losasso et al. 2004], RLE representations [Houston et al. 2006; Irving et al. 2006; Chentanez and Müller 2011], the VDB data structure [Museth 2013] which evolved from Dynamic Tubular Grids [Nielsen and Museth 2006] and the DB+Grid data structure [Museth 2011], and the virtual-memory based SPGrid data structure [Setaluri et al. 2014].

Methods have been proposed for computing implicit representations of non-manifold surfaces [Bloomenthal and Ferguson 1995; Yuan et al. 2012]. Similar ideas were used for simulating bubbles [Zheng et al. 2006] and multiphase fluids [Losasso et al. 2006]. Our work diverges from these approaches as we enhance the expressive capability of a *single* level set by embedding signed distance values on an explicit mesh. Our work is related to the practice of embedding high-resolution geometry in regular meshes, a concept that was first leveraged by Müller et al. [2004] for deformable body simulations and fracture. In addition to hexahedral embeddings, methods such as the virtual node algorithm [Molino et al. 2004] have been used to create non-manifold tetrahedral lattices that correspond to thin topological features in the embedding geometry. Virtual node concepts are also similar to XFEM methods which were used for crack modeling [Moës et al. 1999] and for cutting and fracturing thin shells [Kaufmann et al. 2009]. This principle has continued to evolve with many of the topological limitations in prior approaches being raised by Sifakis et al. [2007] and has been successfully used in production tools as well [Hellrung et al. 2009].

Our non-manifold level set approach is inspired by these methods, but it needs to be made cognizant of further topological limitations that the signed distance field imposes on our representation (see Section 4). Notably, when dealing with collisions near thin features, all of the aforementioned approaches employed detection and response techniques based on surface meshes [Bridson et al. 2002] that rely on the availability of good surface meshes, are computationally expensive, presume collision-free history or use impulses which makes implicit integration challenging. To accelerate collision detection and response while allowing for implicit integration, methods have been proposed using implicit surface representations [McAdams et al. 2011] which work even in near-interactive settings, but require enough level set resolution to avoid any non-manifold features altogether. Recently, image-based techniques [Faure et al. 2008; Wang et al. 2012] have been proposed which provide an interesting alternative. Finally, implicit surfaces have also been recently used in real-time skinning applications [Vaillant et al. 2013; Vaillant et al. 2014].

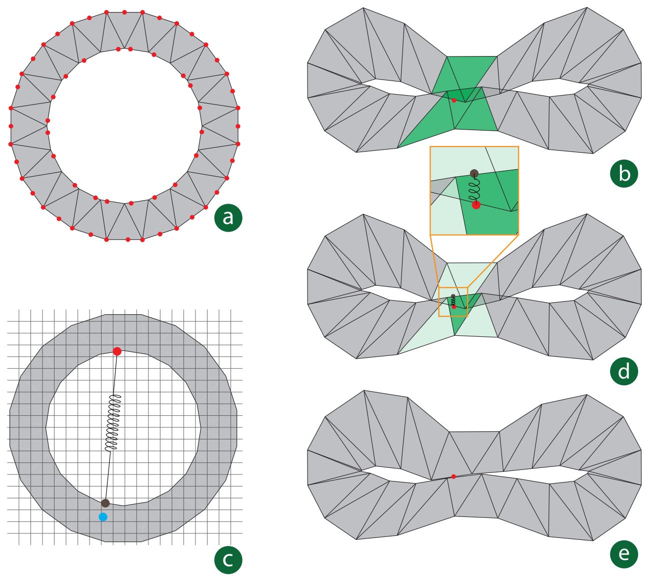


Figure 3: *Illustration of our self-collision pipeline: (a) A triangulated torus model is pictured in its undeformed configuration. Collision proxies on the surface shown in red. (b) The torus is deformed into a self-colliding state. A bounding box hierarchy yields initial candidates of triangles colliding with the proxy. (c) After pruning false positives, the material location that the proxy collided onto is identified, and mapped back to the undeformed configuration (blue dot). The level set (stored on the pictured grid) is used to project to the closest surface point (brown dot). (d) a zero-rest spring is initialized between the proxy and its surface-projected target. (e) The deformed torus after the self-collision is resolved.*

3 Level set collisions for volumetric solids

We present our new implicit surface formulation in the context of collision processing for elastic volumetric bodies. Self-collision processing is paramount in generating visually attractive and realistic shapes, as is evident in character skinning pipelines [McAdams et al. 2011; Vaillant et al. 2013]. Handling collisions in volumetric solids can be quite different than the typical cloth collision pipeline. With volumetric solids there is a clear distinction between an *inside* and an *outside* region, making it possible to process collisions in a single time instance of a simulated deformation. In contrast, if collisions are detected in a cloth simulation, we need to rely on deformation history to determine how the cloth surface is to be untangled (unless global intersection analysis [Baraff et al. 2003] is performed). While cloth simulations typically strive for a collision-free state at all times, commonly enforced via impulses in semi-implicit integration schemes [Bridson et al. 2003], volumetric objects can tolerate occasional, limited interpenetration, and respond to collision with penalty forces which are more easily coupled with explicit integration schemes. In this section, we review a level-set assisted technique that capitalizes on these opportunities to handle volumetric object collisions in large time-step implicit integration schemes, without requiring a collision-free deformation history. In section 4 we explain when standard level sets are inadequate for this task, and use this as motivation for our non-manifold variant.

Our collision pipeline consists of two stages: In the detection stage, discrete material points (labeled *collision proxies*) are checked for collision against the object interior. In the response stage, we use a spring-like penalty force to push each colliding proxy to the object surface [Teran et al. 2005b; McAdams et al. 2011]. Since simulated solids are typically endowed with elastic material models that prevent (or discourage) inversion, in all our examples we chose to only seed collision proxies on the object surface, as internal non-inversion combined with boundary non-collision would imply a globally non-intersecting state. Interior collision proxies can also be used, if desired, with no algorithmic change. Figure 3 illustrates the detection and response process on an elastic torus model squished into self-collision. For any colliding proxy, we identify the offending (internal) material location that the proxy collided with. The closest surface point to that material location is calculated, and a zero rest-length spring is introduced between that surface location and the original proxy. This spring remains active just until the collision detection phase is repeated; typically for one step of the time integration method employed, or for just a single Newton iteration in an implicit scheme. The most costly predicates in this process are (i) detecting whether a proxy intersects the object interior, and (ii) projecting the offending location to the model surface. Both predicates could be answered in $O(1)$ if a level set representation of the model was available; unfortunately, the continuous deformation makes updating an implicit representation impractical. Hence, we opt for an *approximate* algorithm [McAdams et al. 2011] that only relies on a level set representation of the *undeformed* model.

For simplicity, let us assume that the deformable volumetric solids are tetrahedralized. Let E_i denote the i-th simulation element in the undeformed configuration and e_i denote the same element in the deformed configuration. Similarly, let P_i denote the location of the i-th collision proxy in the undeformed configuration and p_i denote its deformed counterpart. Let ϕ denote the level set function for the simulated volume in the undeformed configuration. The collision handling routine performs the following steps for each proxy p_i :

Step 1 The set of (deformed) elements $\mathcal{E} = \{e_{i_1}, e_{i_2}, \ldots, e_{i_k}\}$ are checked against p_i for intersection. This is performed as follows:

(a) We use an axis-aligned bounding box hierarchy, defined over all deformed elements, to identify all elements whose bounding box intersects p_i, i.e. $\mathcal{E}_{\text{int}} = \{e_k \in \mathcal{E} | \texttt{Box}(e_k) \cap p_i \neq \emptyset\}$.

(b) We identify the element E_i that contains the proxy P_i in the undeformed configuration. This may be more than one element, e.g. if P_i was a mesh vertex. We trivially have that $e_i \in \mathcal{E}_{\text{int}}$, as p_i is embedded in it. We prune e_i along with all of its immediate topological neighbors from $\mathcal{E}_{\text{int}}$, as false positives (we do not collide a proxy with itself, or its immediate neighborhood).

(c) We perform an exact intersection test between any elements e_t that have not been already pruned. We do so by computing the barycentric coordinates of p_i with respect to e_t, and discard elements if those coordinates are out of bounds.

Step 2 For every colliding proxy, we identify the location X_t in the *undeformed* configuration of the material point the proxy impacted. We do so using the barycentric coordinates computed in step 1(c) to interpolate X_t from the *undeformed* colliding element E_t.
Step 3 Elements E_t with $\phi(X_t){>}0$ are dismissed as non-colliding (this could be due to discretization discrepancy between mesh and level set, or if an embedded simulation approach is used where elements in $\mathcal{E}$ reach beyond the extent of the simulated model).
Step 4 Using the level set, point X_t is projected to the surface point $Y_t = X_t - \phi(X_t)\nabla\phi(X_t)$, for all elements E_t, where $e_t \in \mathcal{E}_{\text{int}}$.
Step 5 In the deformed configuration, a zero rest-length spring is initialized between points p_i and y_t to resolve the collision.

In step 5, y_t corresponds to the point Y_t in the deformed configuration. Note that our algorithm, in steps 3 and 4, relied upon a level set representation of the *undeformed* shape of the simulated model. The cost paid for this convenience is that the surface location y_t (the collision target) is only an approximate surface projection in the *deformed* configuration; nevertheless, this disparity vanishes as the effect of the collision springs progressively brings the penetration depth closer to zero. Figure 3 illustrates the individual steps of the algorithm on a torus in two spatial dimensions.

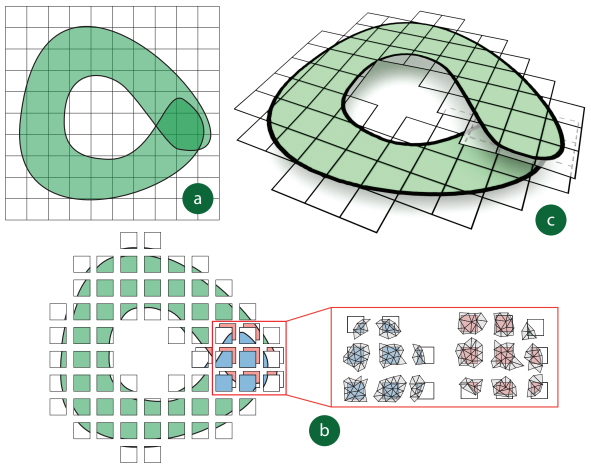


Figure 4: *(A) A self-overlapping 2D model with template mesh overlaid. (B) Duplicate elements created during non-manifold embedded mesh generation, along with their associated material fragments. (C) Final non-manifold embedding mesh.*

4 Non-manifold level sets

The self-collision algorithm outlined in Section 3 works well when a good quality level set can be computed from the model's undeformed configuration. In such cases, it provides the opportunity for excellent performance, even allowing interactive simulation for highly detailed models [McAdams et al. 2011], as it allows very aggressive integration time steps (tolerating occasional mild interpenetration) and exploits the fast intersection/projection level set queries. The approach breaks down, however, in cases where the object contains narrow gaps that cannot be resolved by the level set resolution (see Figure 2). As a brute-force remedy, it might be possible to pose a model in a *reference configuration* that avoids thin features (e.g. modeling a hand such that the fingers are generously separated [McAdams et al. 2011]). However, this pre-processing can be tedious (e.g. for faces with narrow clearance between the lips), unnatural (if the "reference pose" is not really a *rest* pose, see the elastic coil in Figure 2), or impossible to perform a priori if the thin features arise during simulation (e.g. cracks and cuts). We propose a principled remedy, designing a new implicit geometry data structure that fully supports the necessary geometric predicates, but accommodates models with narrow gaps or even material overlap.

4.1 Basic non-manifold embedding

Models such as the ones illustrated in Figure 2 have been known to pose challenges not just for level set generation, but also for certain dynamic simulation techniques even in the absence of collision processing. Of course, simulation of elastic deformation is a straightforward proposition, e.g. using the Finite Element Method [Sifakis and Barbic 2012], if an explicit tetrahedral mesh representation of the model is available. However, performance-optimized deformation techniques [Rivers and James 2007; McAdams et al. 2011] often choose to simulate a regular mesh that merely *embeds* the model geometry, rather than strictly conforming to it. Such techniques would run the risk of "tying" together disconnected material regions if they are separated by a distance smaller than the embedding mesh resolution (e.g. adjacent helices of the coil, or the two lips of the face model pictured in Figure 2). Fortunately, a class of embedding approaches have successfully addressed this very challenge [Molino et al. 2004; Teran et al. 2005a; Sifakis et al. 2007; Nesme et al. 2009]. These methods add non-manifold connectivity to the embedding mesh, duplicating elements and degrees of freedom as necessary to best capture the embedded model topology.

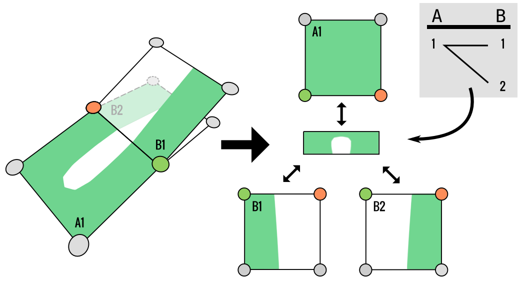


Figure 5: *(Left) An example where material bifurcates at an edge in a non-manifold Cartesian embedding. (Right) Level sets can only store a single interface transition at an edge. In the non-manifold level set bifurcations are explicitly recorded in* transition faces *that record a connectivity graph between all cells on the left and right.*

Since our non-manifold level set structure is inspired by the same principles, we review a common formalism of the non-manifold embedding process [Sifakis et al. 2007] before discussing our level set-specific modifications. The algorithm is illustrated in Figure 4.

Input: (a) A geometric description of the shape to be embedded (the green-shaded area in Figure 4). For simplicity, we may assume the geometry is given as a triangulated model, which allows us to express the self-overlap in our specific example. (b) A mesh which will be used as a *template* for our embedding process. In Figure 4(a) this is the regular quadrilateral mesh pictured in the foreground.
Step 1 [Element separation] We separate each element of the template (quadrilateral) mesh, keeping track of the subset of our material model that is contained in each such element (e.g. taking note of all material triangles that intersected each quadrilateral).
Step 2 [Element duplication for disconnected components] For each embedding element, we identify all disjoint connected components of material contained therein (e.g. by checking connectivity of the respective material triangles). We generate a *duplicate* embedding (quadrilateral) element for each material component. Note that, at this point, all embedding elements are still disconnected.
Step 3 [Restoring connectivity] For any pair of *geometrically* adjacent embedding elements, we check if there is material continuity across their common face (e.g. by checking if they both intersect the same material triangle on that face). If such continuity exists, we collapse all vertices along their common face. This collapse is transitive; in the example of Figure 5(left) all three elements near a convex material region have acquired a common face (with non-manifold connectivity) due to transitive pair-wise vertex collapses.

The result is shown in Figure 4(c); after discarding embedding elements with no material content, the final embedding mesh has been fully assembled, with overlapping duplicates of elements properly connected, respecting the topology of the embedded material.

4.2 Mesh bifurcation and transition faces

The intent of our proposed level set data structure would be to store signed distance values on the *nodes* of the embedding mesh produced by the algorithm just described (to our knowledge, these *non-manifold* embedding meshes have only been previously used to store deformation data, not level set values). Of course, such signed distances would be computed geodesically, along the embedding mesh, rather than in the Euclidean sense. Subsequently, a continuous signed distance field would be computed on the embedding mesh via standard bilinear (2D) or trilinear (3D) interpolation. We note that in "simple" cases such as the example of Figure 4 (where we have element overlap, but no non-manifold connectivity) this approach would have been fully sufficient. Unfortunately, scenarios such as the one illustrated in Figure 5(left) reveal a newfound challenge: Elements hinged in a non-manifold configuration

Algorithm 1 Non-Manifold Level Set Mesh Construction

```
Require: Template Embedding Mesh $\mathcal{T}$, Material Description $\mathcal{M}$
 1: procedure CONSTRUCT_NONMANIFOLD_LEVELSET_MESH
 2:   ▷ Phase 1: Duplicate elements by connected components
 3:   for all Elements in $\mathcal{T}$ : $E_i$ do
 4:     $\mathcal{C} \leftarrow$ CONNECTED_COMPONENTS($\mathcal{M} \cap E_i$)
 5:     for all Components in $\mathcal{C}$ : $m_j$ do
 6:       $D_{i,j} \leftarrow$ CREATE_DUPLICATE($T_i$ , $m_j$)
 7:   ▷ Phase 2: Reconnect or build transition faces
 8:   for all Geometrically adjacent element pairs: $(E_k, E_l)$ do
 9:     $\mathbf{G} \leftarrow$ INITIALIZE_BIPARTITE_GRAPH($\mathcal{D}_k, \mathcal{D}_l, \{\}$ )
10:     for all Duplicates from $E_k$ and $E_l$: $(D_{(k,q)}, D_{(l,r)})$ do
11:       if MATERIAL_CONTINUOUS($D_{(k,q)}$,$D_{(l,r)}$ ) then
12:         INSERT_EDGE($\mathbf{G}, D_{(k,q)}, D_{(l,r)}$)
13:     for all Connected subgraphs of $\mathbf{G}$: $C_i$ do
14:       if #EDGES($C_i$) = 1 then                ▷ Face is Manifold
15:         COLLAPSE(Vertices on common face)
16:       if #EDGES($C_i$) $> 1$ then              ▷ Face is Non-Manifold
17:         REGISTER_TRANSITION_FACE($C_i$)
```

on a common face may disagree on the *sign* of the signed distance value stored on one of their common vertices. In Figure 5(left), elements A1 and B2 record the vertex in orange as being inside the material domain (hence carrying a negative level set value), while the same vertex is outside the embedded domain (with a positive level set value) as far as element B1 is concerned. At this point, we should emphasize that any discrete level set is an inherently approximate representation of geometry, as it depends on interpolation of signed distance value samples. The severity of this phenomenon, however, is much greater as it carries the risk of eliminating parts of the model boundary, or forcing it to spuriously appear in parts of the embedding mesh that it did not originally traverse. Note that this behavior does not affect non-manifold embedding for simulation purposes, since such techniques explicitly track the material embedded in each element, rather than using interpolated vertex.

We posit that, for the proper resolution of non-manifold connectivity, the algorithm of Section 4.1 cannot be allowed to indiscriminately collapse vertices (in Step 3) based solely on material continuity, if this yields a contradiction in the nodal signed distance values across connected elements. Thus, we introduce the concept of a *transition face* which encodes connectivity between incompatible (in terms of the signs of nodal distance values) materially connected elements. This construct is illustrated in Figure 5(right). The transition face is envisioned as an infinitesimally thin connective strip between between elements A1, B1 and B2 with the appropriate internal structure as to connect the material of each element *as reconstructed from their nodal values* via bilinear interpolation. For example, we see that element A1 is considered to be fully interior to the domain, once described by the signed distance values stored at its nodes. We explicitly store a transition face as a *connected* bipartite graph as seen in Figure 5, which records pairwise material continuity of elements on either side, which would normally be lost once only nodal level set values are retained for each element.

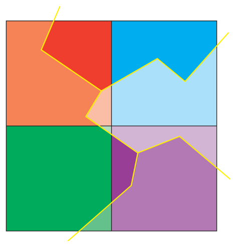
4.3 Non-manifold level set mesh algorithm

Using the transition face mechanism, we can now describe our new algorithm for generating the embedded mesh whose nodes will be used to store the signed distance values of our non-manifold level set. The entire process is outlined in Algorithm 1. The first phase of the algorithm is identical to steps 1-2 of the stock embedding algorithm outlined in section 4.1. As before, given a geometric material description $\mathcal{M}$ (e.g. a tessellation of the model) we identify the material region $\mathcal{M} \cap E_i$ contained within the embedding element E_i from an embedding "template" mesh $\mathcal{T}$. We identify connected components $\{m_j\}_j$ in this set, and create a duplicate embedding element $D_{i,j}$ associated with each material component. As before, all duplicate elements $D_{i,j}$ are completely disconnected at this point.

Subsequently, we analyze material continuity on adjacent embedding elements, with the goal of reconnecting the previously separated elements into the final embedding mesh. For any two elements E_k, E_l that were adjacent in the template mesh $\mathcal{T}$, we identify the sets $\mathcal{D}_k = \{D_{k,q}\}_q$ and $\mathcal{D}_l = \{D_{l,r}\}_r$ of duplicate elements that were respectively spawned from them. We examine each possible pair $(D_{k,q}, D_{l,r})$ drawn from these sets for material continuity across their common face. At this point, however, instead of collapsing vertices on the common face of such pairs that are found to be materially connected, we simply record this connectivity with an edge in a bipartite graph $\mathbf{G}$ defined over the sets $\mathcal{D}_k$ and $\mathcal{D}_l$. Once all pairs from $\mathcal{D}_k$ and $\mathcal{D}_l$ have been processed, we proceed to split the graph $\mathbf{G}$ into its connected components (in terms of graph connectivity, not material connectivity as in Phase 1). For every connected component (subgraph) of $\mathbf{G}$, we proceed as follows:

- If a connected subgraph contains *exactly* one edge, the duplicate elements $D_{k,q}$ and $D_{l,r}$ connected by that edge are guaranteed to be compatible relative to the sign of the distance value stored on their nodes, since they agree exactly on the material intersecting their (geometrically) common face. This is a consequence of this edge being a connected component of $\mathbf{G}$, indicating that no other element is independently connected to either $D_{k,q}$ or $D_{l,r}$. In this case, we are free to collapse the vertices of the two duplicate elements across their common face, exactly as we did in section 4.1.
- If a connected subgraph contains two or more edges (see Figure 5(right)), we cannot collapse all vertices on the duplicate elements' common face, since some of these elements may disagree on the sign of the distance field stored on their nodes. In this case, we simply generate a transition face, which is encoded using the same connected subgraph, allowing the duplicate elements that are juxtaposed on that transition face to retain independent signed distance values on their nodes. As we will see in the next sections, a transition face is semantically equivalent to a "hard" topological connection (a collapsed face) for operations that traverse the final embedding mesh, with the exception of its ability to allow separate signed distance values on each duplicate element it connects.

Implementation notes For simplicity of exposition we have thus far assumed that the description of the material model $\mathcal{M}$ is given in the form of an explicitly meshed object (e.g. a tetrahedralized volume in 3D). However, this is not strictly necessary and *any* material description that can answer the predicates of Algorithm 1, lines 4 (connectivity within an element) and 11 (material continuity across elements) can be trivially used in the same framework.



For example, we demonstrate cutting of volumetric elastic models (Figure 10), where a user-specified fracture surface is used to explicitly subdivide each element of the template mesh $\mathcal{T}$ into disjoint polyhedra. This polyhedral decomposition [Sifakis et al. 2007] natively provides connected component information, and can easily detect material continuity across adjacent embedding elements by checking if their polyhedral material regions share a face on their common boundary, as seen in the insert image above. Finally, the transitive Collapse operation (line 15) is implemented in practice using a Union-Find structure which records transitive equivalences of vertex identifiers.

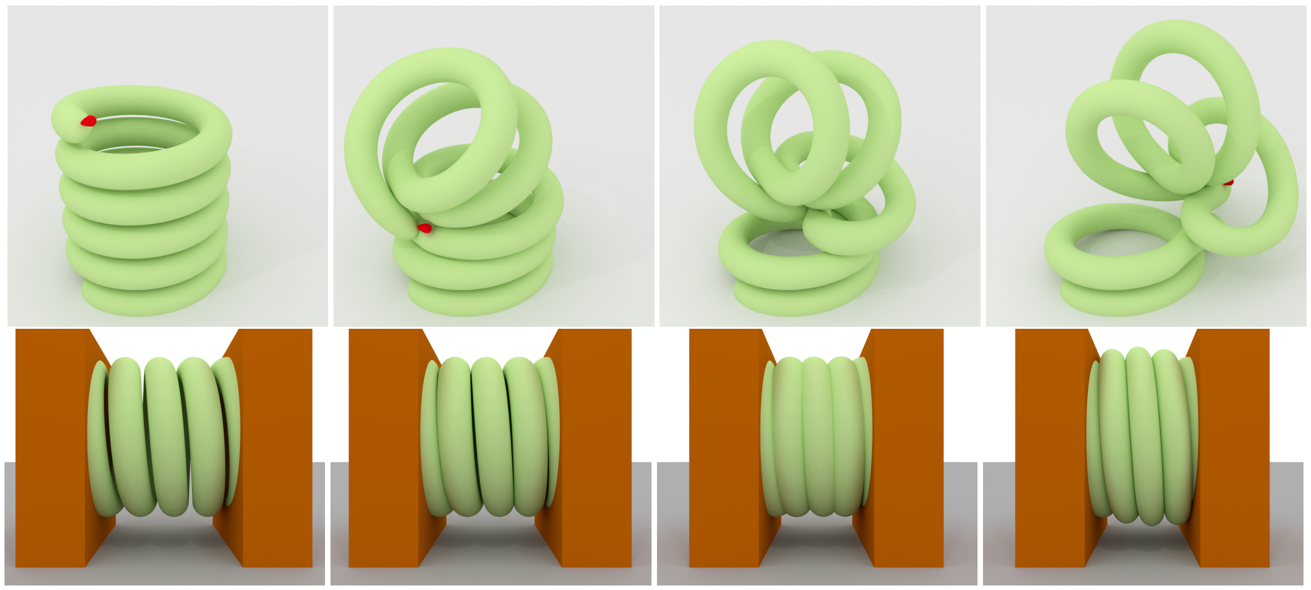


Figure 6: *(Top) A volumetric coil self-collides under user manipulation. (Bottom) A coil is compressed against two walls. Subsequently, collision handling is disabled and the geometry self-intersects (third frame in row). Self-collisions are turned back on and the coil recovers.*

4.4 Level Set operations on nonmanifold meshes

Initialization of signed distances Once the topology of the embedding mesh has been constructed, including the creation of the necessary transition faces, the embedding mesh nodes must be populated with the proper signed distance values. We start by explicitly computing such distances on embedding elements that intersect the object boundary. Since we possess an explicit description of the material contained in each element, for each of their nodes we compute the minimum (absolute) distance from all material contained in that element. We also compute the sign depending on whether the node is inside or outside the embedded material component. Adjoining elements that have had common vertices collapsed (topologically; not connected via transition faces) will agree on the sign of the signed distance field at shared nodes, but not necessarily the magnitude. We retain the distance value with the *minimum* magnitude, across all elements incident to this node. Of course, no such reduction is performed on nodes connected via transition faces. Subsequently, we propagate the signed distance field in the interior of the object using the $O(n \log n)$ Fast Marching Method [Sethian 1998], with the only modification that this Dijkstra-type algorithm is allowed to propagate through transition faces in exactly the same fashion as through explicitly connected nodes. While we only compute a scalar signed distance field, it would be straightforward to also compute a normal field [Kobbelt et al. 2001] to support higher quality reconstructions.

Distance queries and surface projection The basic level set predicates required in the collision pipeline of section 3 include a lookup of the signed distance value $\phi(\vec{x})$ at an arbitrary location $\vec{x}$ in space, and the projection of a material point to the closest location $\mathsf{Proj}(\vec{x};\Gamma)$ on the model surface Γ. Since our embedding mesh may contain several overlapping elements, it is no longer sufficient to define such predicates as functions of just the spatial location $\vec{x}$ being queried; we also need to identify the appropriate branch of material being referred to. Thus, we reformulate these predicates as $\phi(\vec{x},D_i)$ and $\mathsf{Proj}(\{\vec{x},D_i\};\Gamma)$, where the element D_i embeds the material point $\vec{x}$ in the non-manifold level set mesh. Subsequently, the result of the projection operator is also a tuple $(\vec{x}^\star,D_j)$ denoting a material point $\vec{x}^\star$ and its respective embedding element D_j.

Given an embedding element D_l and a location $\vec{x}$ embedded in it, level set value and gradient are computed via trilinear interpolation:

$$\phi(\vec{x},D_l)=\sum_{i,j,k=0}^{1}\mathcal{N}_{ijk}(\vec{x})\phi_{ijk},\quad \nabla\phi(\vec{x},D_l)=\sum_{i,j,k=0}^{1}\nabla\mathcal{N}_{ijk}(\vec{x})\phi_{ijk}$$

where $\mathcal{N}_{ijk}$ denotes the trilinear basis functions and ϕ_{ijk} are the signed distance values at the nodes of D_l. It is known that the gradient of the level set function, i.e. the steepest ascent direction of the distance field, is a unit normal which points in the direction of the closest point on the surface. Thus, the closest point to $\vec{x}$ on the model surface is to be found in the direction of $\vec{n}=\nabla\phi(\vec{x},D_l)$, at a distance of $|\phi(\vec{x},D_l)|$. Thus, analytically:

$$\mathsf{Proj}(\{\vec{x},D_l\})=\vec{x}-\phi(\vec{x},D_l)\nabla\phi(\vec{x},D_l)$$

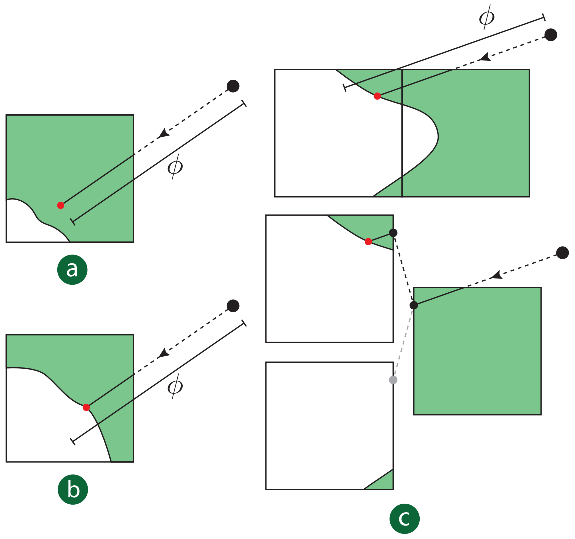


Figure 7: *Different scenarios of surface projection. (a) Backtracing terminates after covering a distance of ϕ without intersecting the interface. (b) Backtracing terminates at the location of the first interface crossing. (c) Backtracing hits a transition face and continues into a connected neighbor with the largest negative ϕ value.*

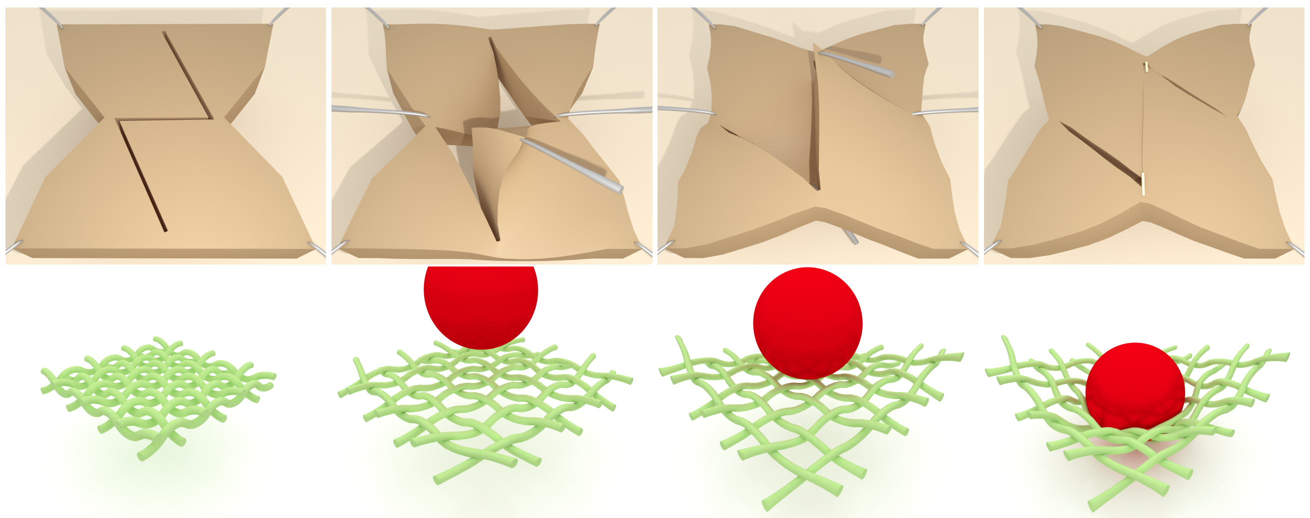


Figure 8: *(Top) Surgical simulation of a z-plasty procedure, with self-collision processing. (Bottom) A net is stretched out, twisted to a saddle configuration and a ball is subsequently dropped on it. A single level set is used for the entire net during self-collision processing.*

To compute the projection $\mathsf{Proj}(\{\vec{x}, D_l\}; \Gamma)$ we topologically backtrace the non-manifold level set mesh along $\vec{n}$ element-by-element, to ensure that we follow a geodesic path along the embedding mesh, as shown in Figure 7. If while traversing a distance $\phi(\vec{x}, D_l)$ along $\vec{n}$ we land in an element D_m that is crossed by the interface, then we use bisection search to compute the interface point $\vec{x}^\star$ and return the tuple $(\vec{x}^\star, D_m)$ (Figure 7(b)). If we have traversed a distance equal to $\phi(\vec{x}, D_l)$ without crossing any interface, we stop the backtrace operation and report the location reached after the requisite distance has been traveled (Figure 7(a)); we do so to avoid grazing by a nearby interface without actually stopping there. Finally, if the backtracing process crosses a transition face f, then we compute the point $\vec{x}_f$ where the ray from $\vec{x}$ along $\vec{n}$ crosses f. We then compute the value of ϕ at $\vec{x}_f$ for all elements on the other side (connected through the transition face) and choose the one that gives the largest negative value. If no such element is present, then we assume that the interface lies *exactly* at $\vec{x}_f$ and return this point along with the cell from which we entered the transition face as the result of the projection. We note that although this projection is approximate, the error is comparable with conventional, grid-based level sets, and perfectly acceptable for our collision handling scheme.

5 Examples

We simulated a number of examples to demonstrate the efficacy of our method in several challenging scenarios. Figure 6(top) shows a user pulling a three dimensional volumetric coil at the red handle creating complex self-collisions. Figure 6(bottom) shows the same coil being compressed against two moving walls. Self-collisions are turned off at some point to make the geometry self-intersecting, and subsequently turned back on again resulting in the coil bulging outwards. This example shows that our method does not require any history information for resolving self-collisions. Figure 8(top) shows a simulation of a common maneuver in plastic surgery, called a *Z-plasty*, while Figure 8(bottom) shows a ball dropping on a net that has been stretched outwards and twisted into a saddle configuration. Our method uses a single level set for the entire net during self-collision processing, obviating the need for multiple collision level sets and circumventing the complexity in bookkeeping associated with such scenarios. Figure 9(b) shows an example where the lower jaw of a face model is pulled down and subsequently pushed back up, opening and closing the mouth in the process. Note the slight bulge in the cheeks due to self-collisions at the lips when the mouth is closed because the jaw is pushed further up compared to the rest state. Figure 9(c) shows a user moving around two points on the lips (shown in orange) to demonstrate complex self-collisions that our method can resolve. Finally, Figure 10 shows an example where a cube is partially sliced by six planes using the method of [Sifakis et al. 2007]. This results in sixteen fingers which are pushed apart when squashed by a ball from the top. Note that a standard Cartesian grid-based level set cannot be used for resolving this structure irrespective of its resolution.

Model	Level set Gen (s)	Solve (s)	Collision Proc. (s)	Backtrace Total (ms)	Proxy Count
Z-plasty	222.6	1.961	0.0671	0.0479	7121
Coil	580.5	13.22	0.4651	14.8	31126
Net	524.0	23.47	0.4123	5.38	48042
Face	271.0	24.46	0.2977	0.620	48851

The table below captures the performance impact of our collision handling methodology. The first column lists the computation times for generating the non-manifold level set mesh; we emphasize that this is a one-time precomputation cost, before dynamic simulation even starts. The following columns list the cost for each step of our Backward Euler implicit integration scheme, divided into the solution of the linearized equations, the cost of collision processing, and specifically the aggregate cost of all backtracing operations for projecting proxies to the object surface. It can easily be seen that the cost of collision processing is a minute fraction of the overall simulation. This is a consequence of our scheme not requiring a history of collision free states, and thus being able to take significantly more aggressive steps than semi-implicit schemes that strictly disallow interpenetration [Bridson et al. 2002].

6 Limitations and future work

Our pipeline for non-manifold level sets has been specifically created for self-collision processing in deformable body simulations. There are several diverse applications of the standard (manifold) level set concept such as representing geometry, tracking dynamically evolving interfaces, as a geometric query structure, etc. However, we believe that the current methodology may require application-specific embellishments to cater to broader tasks in modeling and simulation, beyond what was needed for our collision-oriented proof of concept. For example, when two dynamically evolving interfaces are brought together, they may either

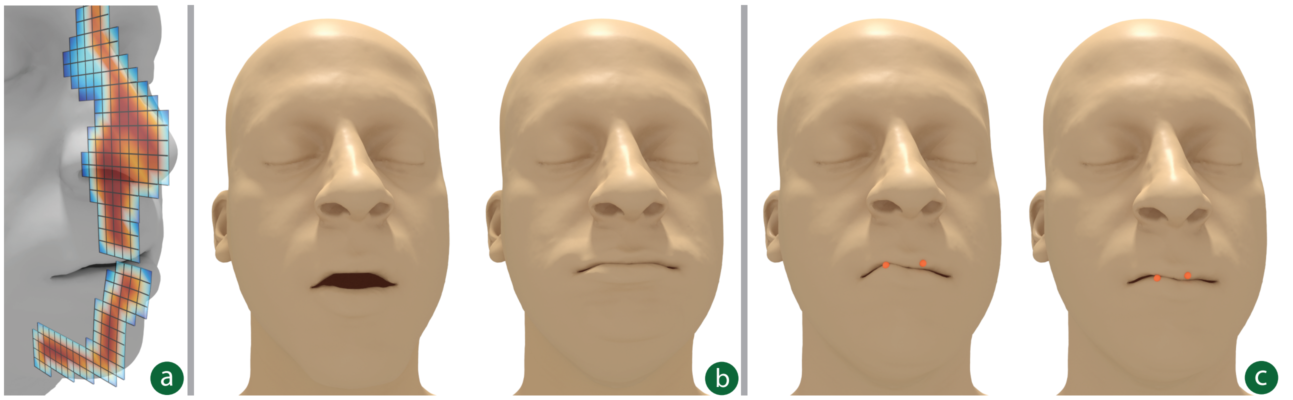


Figure 9: *(a) Cutaway view of the non-manifold level set generated on a face model, (b) The lower jaw is displaced vertically, opening and closing the mouth. Note the small bulges in the cheek due to self-collisions at the lips because the jaw is pushed further up compared to the rest state. (c) A user moves around two points on the lips (orange) to demonstrate the robustness of our method in resolving self-collisions.*

be allowed to merge (as is typical in fluid simulations) or overtake one another (e.g. the two separate branches of the lips during self-collision). Thus, application-specific semantics might be needed to use non-manifold level sets for dynamic interfaces.

Currently, we do not extend the non-manifold level set values into a narrow band outside the interface because we used the same grid both for simulation and for storing level set values. If the extent is predetermined, then we could generate a coarse non-manifold level set mesh first and refine it as a post-process. Alternatively, one could also "grow" the non-manifold level set by following characteristics, but as mentioned above, one may or may not wish to merge overlapping characteristics depending on the context. This issue is also related to the question of "evolving" a non-manifold level set. In future work, we wish to address these questions targeting the specific applications of multiphase flow and crack propagation.

For standard Cartesian grid-based level sets, there exists an established theoretical foundation for accurately computing high order gradients even in the vicinity of singularities. For non-manifold level sets, an extra level of complexity is introduced because there can be a topological bifurcation in addition to a singularity. To obtain valid values in such complex situations, our current scheme reverts to first order element-wise trilinear interpolation. It would be interesting to explore in future work if there can be a more accurate representation for the interface in these cases. Our current representation is still discrete in the sense that the expressive ability of the non-manifold level set mesh only gets discretely modified when we allow to separate a cell or bifurcate a cell. This is contrast to XFEM methods which can include discontinuities that continuously evolve through material, and as a consequence, it can create a continuous path of deformation or topological change as an evolving front or a crack propagates through material elements.

Finally, our current implementation used Cartesian grids as template meshes for generating the non-manifold level set. In future work, we would like to explore different representations such as octrees [Losasso et al. 2004], RLE representations [Houston et al. 2006; Irving et al. 2006; Chentanez and Müller 2011], the VDB data structure [Museth 2013] and SPGrid [Setaluri et al. 2014].

Acknowledgements

The authors are grateful to the SIGGRAPH & SIGGRAPH Asia anonymous reviewers for their insightful feedback. We would also like to thank Ronald Fedkiw and Matthew Cong for valuable early discussions that contributed to the motivation for this method. This work was supported in part by National Science Foundation Grants IIS-1253598, CNS-1218432, CCF-1423064 and CCF-1438992.

References

ADALSTEINSSON, D., AND SETHIAN, J. A. 1994. A fast level set method for propagating interfaces. *JCP 118*, 269–277.

BARAFF, D., WITKIN, A., AND KASS, M. 2003. Untangling cloth. *ACM Trans. Graph. 22*, 3, 862–870.

BLOOMENTHAL, J., AND FERGUSON, K. 1995. Polygonization of non-manifold implicit surfaces. SIGGRAPH '95, 309–316.

BRIDSON, R., FEDKIW, R., AND ANDERSON, J. 2002. Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph. 21*, 3, 594–603.

BRIDSON, R., MARINO, S., AND FEDKIW, R. 2003. Simulation of clothing with folds and wrinkles. SCA '03, 28–36.

CHENTANEZ, N., AND MÜLLER, M. 2011. Real-time eulerian water simulation using a restricted tall cell grid. SIGGRAPH '11, 82:1–82:10.

DA, F., BATTY, C., AND GRINSPUN, E. 2014. Multimaterial mesh-based surface tracking. *ACM TOG 33*, 4, 112:1–112:11.

ENRIGHT, D., MARSCHNER, S., AND FEDKIW, R. 2002. Animation and rendering of complex water surfaces. *ACM Trans. Graph. 21*, 3, 736–744.

FAURE, F., BARBIER, S., ALLARD, J., AND FALIPOU, F. 2008. Image-based collision detection and response between arbitrary volume objects. SCA '08, 155–162.

GASCUEL, M.-P. 1993. An implicit formulation for precise contact modeling between flexible solids. In *SIGGRAPH '93*, 313–320.

GUENDELMAN, E., BRIDSON, R., AND FEDKIW, R. 2003. Non-convex rigid bodies with stacking. *ACM TOG 22*, 3, 871–878.

HELLRUNG, J., SELLE, A., SHEK, A., SIFAKIS, E., AND TERAN, J. 2009. Geometric fracture modeling in bolt. In *SIGGRAPH 2009: Talks*, SIGGRAPH '09, 7:1–7:1.

HOUSTON, B., NIELSEN, M. B., BATTY, C., NILSSON, O., AND MUSETH, K. 2006. Hierarchical RLE level set: A compact and versatile deformable surface representation. *ACM Trans. Graph. 25*, 1, 151–175.

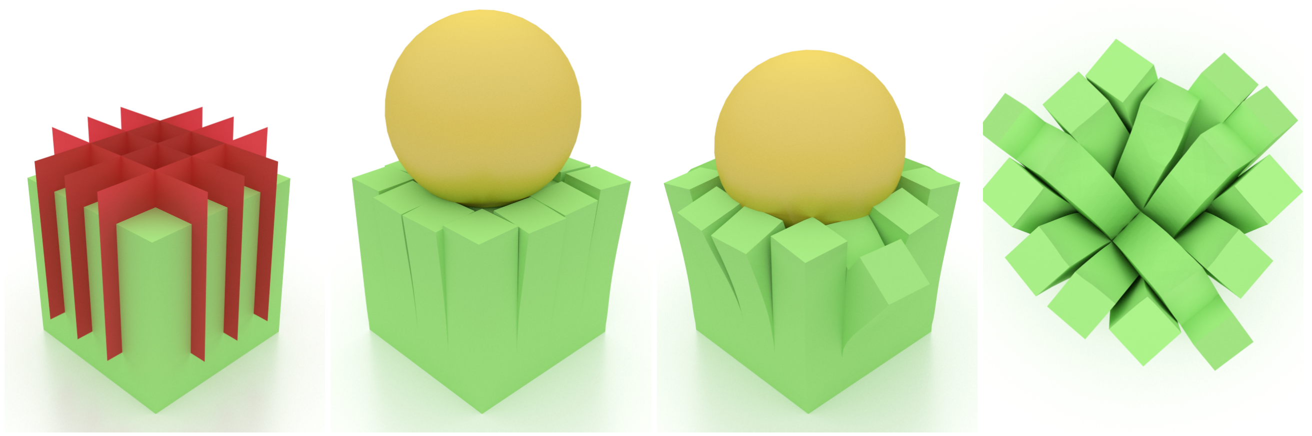


Figure 10: *A cube is partially sliced by 6 planes and a ball subsequently squashes it to push the resulting 16 fingers apart. Our non-manifold level set can robustly resolve zero width cuts which could not be resolved with standard Cartesian grid-based level sets.*

IRVING, G., GUENDELMAN, E., LOSASSO, F., AND FEDKIW, R. 2006. Efficient simulation of large bodies of water by coupling two and three dimensional techniques. SIGGRAPH, 805–811.

KAUFMANN, P., MARTIN, S., BOTSCH, M., GRINSPUN, E., AND GROSS, M. 2009. Enrichment textures for detailed cutting of shells. *ACM Trans. Graph. 28*, 3, 50:1–50:10.

KOBBELT, L. P., BOTSCH, M., SCHWANECKE, U., AND SEIDEL, H.-P. 2001. Feature sensitive surface extraction from volume data. SIGGRAPH '01, 57–66.

LABELLE, F., AND SHEWCHUK, J. 2007. Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles. *ACM TOG 26*, 3.

LOSASSO, F., GIBOU, F., AND FEDKIW, R. 2004. Simulating water and smoke with an octree data structure. SIGGRAPH '04, 457–462.

LOSASSO, F., SHINAR, T., SELLE, A., AND FEDKIW, R. 2006. Multiple interacting liquids. *ACM TOG 25*, 3, 812–819.

MCADAMS, A., ZHU, Y., SELLE, A., EMPEY, M., TAMSTORF, R., TERAN, J., AND SIFAKIS, E. 2011. Efficient elasticity for character skinning with contact and collisions. *ACM Trans. Graph. 30*, 4, 37:1–37:12.

MOËS, N., DOLBOW, J., AND BELYTSCHKO, T. 1999. A finite element method for crack growth without remeshing. *IJNME 46*, 131–150.

MOLINO, N., BAO, Z., AND FEDKIW, R. 2004. A virtual node algorithm for changing mesh topology during simulation. *ACM Trans. Graph. 23*, 3, 385–392.

MULLER, M., TESCHNER, M., AND GROSS, M. 2004. Physically-based simulation of objects represented by surface meshes. CGI '04, 26–33.

MUSETH, K., BREEN, D., WHITAKER, R., AND BARR, A. 2002. Level set surface editing operators. In *ACM TOG*, 330–338.

MUSETH, K. 2011. DB+Grid: A novel dynamic blocked grid for sparse high-resolution volumes and level sets. SIGGRAPH '11.

MUSETH, K. 2013. VDB: High-resolution sparse volumes with dynamic topology. *ACM Trans. Graph. 32*, 3 (July), 27:1–27:22.

NESME, M., KRY, P. G., JEŘÁBKOVÁ, L., AND FAURE, F. 2009. Preserving topology and elasticity for embedded deformable models. *ACM Trans. Graph. 28*, 3, 52:1–52:9.

NIELSEN, M. B., AND MUSETH, K. 2006. Dynamic tubular grid: An efficient data structure and algorithms for high resolution level sets. *J. Sci. Comput. 26*, 3 (Mar.), 261–299.

OSHER, S., AND FEDKIW, R. 2002. *Level Set Methods and Dynamic Implicit Surfaces*. Springer.

OSHER, S., AND SETHIAN, J. 1988. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *J. Comput. Phys. 79*, 12–49.

RIVERS, A., AND JAMES, D. 2007. FastLSM: Fast lattice shape matching for robust real-time deformation. *ACM TOG 26*, 3.

SACHT, L., JACOBSON, A., PANOZZO, D., SCHÜLLER, C., AND SORKINE-HORNUNG, O. 2013. Consistent volumetric discretizations inside self-intersecting surfaces. *Computer Graphics Forum 32*, 5, 147–156.

SETALURI, R., AANJANEYA, M., BAUER, S., AND SIFAKIS, E. 2014. SPGrid: A sparse paged grid structure applied to adaptive smoke simulation. *ACM Trans. Graph. 33*, 6, 205:1–205:12.

SETHIAN, J. A. 1998. Fast marching methods. *SIAM Review 41*, 199–235.

SIFAKIS, E., AND BARBIC, J. 2012. Fem simulation of 3d deformable solids: A practitioner's guide to theory, discretization and model reduction. In *ACM SIGGRAPH 2012 Courses*, SIGGRAPH '12.

SIFAKIS, E., DER, K. G., AND FEDKIW, R. 2007. Arbitrary cutting of deformable tetrahedralized objects. SCA '07, 73–80.

TERAN, J., SIFAKIS, E., BLEMKER, S. S., NG-THOW-HING, V., LAU, C., AND FEDKIW, R. 2005. Creating and simulating skeletal muscle from the visible human data set. *IEEE Transactions on Visualization and Computer Graphics 11*, 3, 317–328.

TERAN, J., SIFAKIS, E., IRVING, G., AND FEDKIW, R. 2005. Robust quasistatic finite elements and flesh simulation. SCA '05, 181–190.

VAILLANT, R., BARTHE, L., GUENNEBAUD, G., CANI, M.-P., ROHMER, D., WYVILL, B., GOURMEL, O., AND PAULIN, M. 2013. Implicit skinning: Real-time skin deformation with contact modeling. *ACM Trans. Graph. 32*, 4, 125:1–125:12.

VAILLANT, R., GUENNEBAUD, G., BARTHE, L., WYVILL, B., AND CANI, M.-P. 2014. Robust iso-surface tracking for interactive character skinning. *ACM TOG 33*, 6, 189:1–189:11.

WANG, B., FAURE, F., AND PAI, D. K. 2012. Adaptive image-based intersection volume. *ACM Trans. Graph. 31*, 4, 97:1–97:9.

YUAN, Z., YU, Y., AND WANG, W. 2012. Object-space multi-phase implicit functions. *ACM TOG 31*, 4, 114:1–114:10.

ZHAO, H.-K., OSHER, S., AND FEDKIW, R. 2001. Fast surface reconstruction using the level set method. VLSM '01, 194–202.

ZHENG, W., YONG, J.-H., AND PAUL, J.-C. 2006. Simulation of bubbles. SCA '06, 325–333.