Tracking Surfaces with Evolving Topology

Morten Bojsen-Hansen*
IST Austria

Hao Li†
Columbia University

Chris Wojtan*
IST Austria

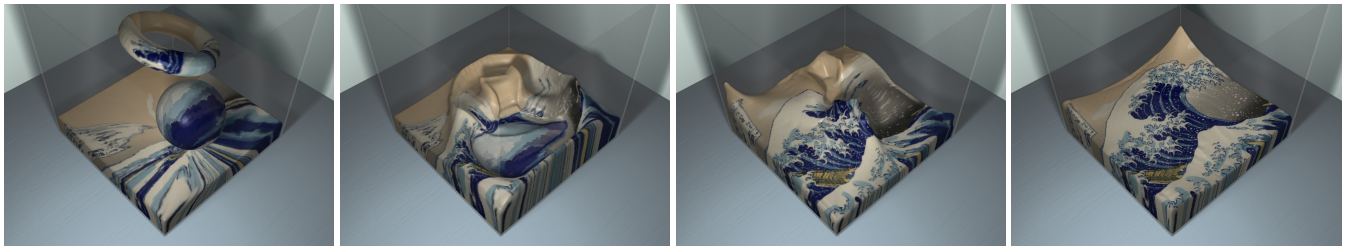

Figure 1: *Our method recovers a sequence of high-quality, temporally coherent triangle meshes from any sequence of closed surfaces with arbitrarily changing topology. We reliably extract correspondences from a level set and track textures backwards through a fluid simulation.*

Abstract

We present a method for recovering a temporally coherent, deforming triangle mesh with arbitrarily changing topology from an incoherent sequence of static closed surfaces. We solve this problem using the surface geometry alone, without any prior information like surface templates or velocity fields. Our system combines a proven strategy for triangle mesh improvement, a robust multi-resolution non-rigid registration routine, and a reliable technique for changing surface mesh topology. We also introduce a novel topological constraint enforcement algorithm to ensure that the output and input always have similar topology. We apply our technique to a series of diverse input data from video reconstructions, physics simulations, and artistic morphs. The structured output of our algorithm allows us to efficiently track information like colors and displacement maps, recover velocity information, and solve PDEs on the mesh as a post process.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

Keywords: mesh deformation, non-rigid tracking, implicit surfaces, fluid simulation

Links: DL PDF

1 Introduction

Robust computational representations of deforming surfaces are considered indispensable within many scientific and industrial fields. Medical scientists deduce clues about the human body from the level sets of time-varying voxel data, physicists extract geometric information from simulations and acquisitions of fluid interfaces, and computer graphics professionals generate animations and capture performances in order to entertain audiences. As tools that generate time-evolving surfaces become increasingly commonplace, it is essential that we, as computer graphics researchers, provide better tools for the analysis and computational processing of these forms of animated geometry.

One particular class of evolving surfaces, namely surfaces that change topology through time, is particularly difficult to deal with. Because these surfaces are able to bend, split apart, reconnect themselves, and disappear through time, it is impossible to make any convenient assumptions about their shape and connectivity. For this reason, *implicit surfaces* such as contoured voxel data and metaballs, are extremely popular for representing such time-evolving surfaces. Unfortunately, these implicit surfaces are poorly suited for many important geometric tasks, such as mapping how surface points at one particular time correspond to surface points sometime later.

In this paper, we provide a general, robust method for tracking correspondence information through time for an arbitrary sequence of closed input surfaces. We do not require any context clues such as velocity information or shape priors, and we allow the surfaces to change topology through time. We solve this problem by combining a robust non-rigid registration algorithm, a reliable method for computing topology changes in triangle meshes, and a mesh-improvement routine for guaranteeing numerical accuracy and stability. The output of our method is a series of temporally coherent triangle meshes, as well as an *event list* that tracks how surface vertices correspond through time.

We apply our method to data sets generated by different methods, such as physics simulations using two separate surfacing algorithms, morphing surfaces generated by implicit surfaces, and performance capture data reconstructed from videos. We show that we can reliably extract correspondence information that was absent from the original geometry, and we utilize this information to significantly enhance the input data. Using our algorithm, we are able to preserve important surface features, apply textures and displacement maps, simulate partial differential equations on the surface, and even propagate visual information *backwards* in time. When applied to dynamic shape reconstruction problems, we are able to reliably track the input without making any assumptions about how the data was generated. One can argue that this template-free tracking is an important tool for scientific experiments where it is essential to remove bias from the tools used for information discovery. The contributions of our work are as follows:

*(mortenbh|wojtan)@ist.ac.at
†hao@hao-li.com

- We provide the first comprehensive framework for tracking a series of closed surfaces where topology can change.
- Our algorithm is able to greatly enhance existing datasets with valuable temporal correspondence information. Some examples include displacement mapping of fluid simulations and texture mapping of level set morphs.
- We introduce a novel topology-aware wave simulation algorithm for enhancing the appearance of existing liquid simulations while significantly reducing the noise present in similar approaches.
- Because our method robustly extracts surface information from input data alone, we provide a reliable way to automatically track markerless performance capture data without the need for a template.

2 Related Work

Our work is closest to a recent publication of Stam and Schmidt [2011]. They showed that, by examining the input parameters for an implicit surface algorithm, one can derive the surface velocity to create motion blur and more coherent surface animations. By integrating surface velocity through time, they presented a method to approximate point-to-point correspondences which can be used to track texture information. This inspirational work introduced some exciting applications for tracking correspondences through complicated deformations, and we believe that it brought the community a significant step closer to solving the general problem of tracking a topology-evolving surface. Our method is different from theirs in a number of ways. Firstly, we wish to solve the more general problem of tracking an arbitrary input surface sequence, so we do not assume that we know the parameters behind the surface dynamics. Secondly, their correspondence information is only as accurate as their velocity integration, so it is prone to numerical drift. Our method uses a nonlinear shape matching optimization to minimize this drift, and the difference is particularly apparent in the presence of large rotations.

To the best of our knowledge, our method is the first to provide a solution to the problem of registration combined with topology change. For the remainder of this section, we divide the work most related to ours into two camps: those related to deformable shape matching and registration, and those related to surface evolution with topology changes.

Deformable Shape Matching and Registration. The field of dynamic geometry processing is actively involved with the problem of extracting correspondences between inconsistent time-varying meshes [Chang et al. 2010]. Dense and accurate correspondences are critical for temporal shape analysis and surface tracking, making applications such as marker-free human performance capture and shape reconstruction from streams of incomplete 3D data possible. We will focus our discussion on methods that take sequences of meshes or point clouds as input.

Most methods that establish full surface correspondences through time rely on an existing template model or construct it in a separate step. With a *fixed topology* and known geometric state, template models are popular because they simplify the problem of reconstructing geometry and motion. The techniques introduced in [Mitra et al. 2007; Süßmuth et al. 2008] aggregate scan sequences into a 4D space-time surface to build a more complete template. In addition to being limited to fairly small deformations, both methods do not allow the input data to change topology. The statistical framework introduced by Wand et al. [2007] and later improved in [Wand et al. 2009] estimates a globally consistent template model with a fixed topology. While also being restricted to slowly-varying surface deformations, their methods can identify topology variations in the scans. On the other hand, the framework presented in [Li et al. 2009] does use a rough approximation of a template as a prior, preventing wrong topology computations, but focuses on handling deformations that are significantly larger than previous methods using a robust non-rigid registration algorithm. While highly disruptive motions are explicitly treated in the system of Tevs et al. [2012], largely incomplete acquisitions can still damage the template extraction.

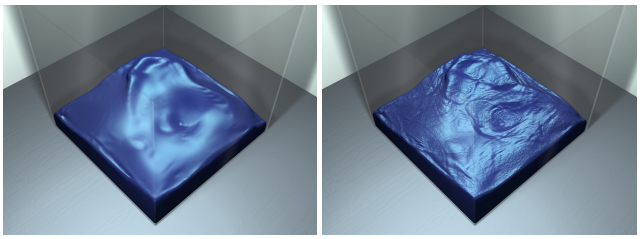

Figure 2: *Our framework allows us to synthesize high-frequency details using a separate wave simulation (right) on top of a lower resolution pre-simulated fluid surface (left).*

Although correspondences are desirable for many geometric analysis and manipulation purposes, a few state-of-the-art reconstruction methods skip the requirement of extracting a template model but aim at simply filling incomplete capture data. The technique presented in [Sharf et al. 2008] is able to produce a watertight surface sequence from extremely noisy input scans using a volumetric incompressible flow prior but suffers from significant flickering in the reconstruction. In the context of fluid capture, Wang et al. [2009] demonstrated a framework to fill holes in partially captured liquid surfaces using a physically guided model. Their method achieves time-coherent reconstructions of dynamic surfaces but is restricted to fluid simulations since frame-to-frame correspondences are guided by a simulated velocity field. Recently, Li et al. [2012] demonstrated a shape completion framework for temporally coherent hole filling of incomplete and flickering-affected scans of human performances. Their method makes minimal assumptions about the surface deformation by establishing correspondences within a small time window and avoid the extraction of globally consistent correspondences through time.

The proposed technique is able to establish full correspondences across time-series of input meshes and is not limited to a fixed topology like template-based methods. Our method is grounded on a general purpose non-rigid registration algorithm similar to [Li et al. 2009; Li et al. 2012] and can therefore be applied widely, ranging from fluid surface dynamics, human body performances, and arbitrary shape morphings.

Surface Evolution with Topology Changes. Several methods exist for tracking topology-changing surfaces through time with the aid of prescribed motions or velocity fields. Level set methods [Osher and Fedkiw 2003] and particle level set methods [Enright et al. 2002] are popular techniques for representing a dynamic implicit surface. These methods consider the zero level set of a voxelized signed distance function, and they integrate velocity information in order to move the function. This integration displaces the zero set of the function, resulting in a moving surface. Müller [2009] used a strategy of repeatedly re-sampling an evolving Lagrangian triangle mesh in order to provide fast surface tracking for fluid surfaces. Semi-Lagrangian contouring [Bargteil et al. 2006a] also utilizes Lagrangian information in the form of extracted surface geometry in order to improve surface tracking. These methods can be used to propagate surface information through time, but they cannot reli-

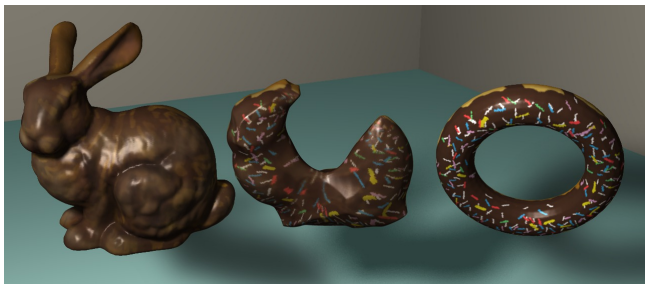

Figure 3: *A morphing example where surface textures are tracked. Unlike existing techniques, our method does not exhibit ghosting artifacts.*

ably track surface correspondences over large deformations without diffusion because of their strategy of continual re-sampling. Similar to our method, Dinh et al. [Dinh et al. 2005] also tracks texture information on a topology-changing surface. Their method requires the solution of a PDE over space-time, which limits its application to low resolution surfaces over a short amount of time. Our method treats each time step independently, so it is able to handle highly detailed input.

The surface evolvers most similar to ours are mesh-based surface tracking methods [Du et al. 2006; Wojtan et al. 2010; Brochu et al. 2010]. The idea behind these techniques is to evolve a triangle mesh according to a velocity field, which allows for better preservation of geometric features and correspondence information than an implicit surface. These mesh-based methods go hand-in-hand with robust numerical methods for changing mesh topology [Brochu and Bridson 2009; Wojtan et al. 2009; Campen and Kobbelt 2010; Zaharescu et al. 2007; Pons and Boissonnat 2007]. Within our framework, we use a method similar to Wojtan et al. [2009] for changing mesh topology, because of its speed and versatility (Further details are explained in §4.3).

While each of these works on surface evolution certainly helped inspire ours, we would like to remind the reader that our method solves a significantly different problem of tracking *without any velocity information*. In this light, we do not perceive our method as a competitor to existing fluid simulation techniques, but as a powerful enhancement tool — it allows a user to convert the output from *any simulation type* into a temporally coherent deforming mesh sequence. Our tracked surfaces are a great improvement over implicit surfaces in the information they provide, the details they preserve, and the useful applications that they aid.

3 Problem Statement

This paper is concerned with the problem of taking a series of closed surfaces through time as input, and then replacing these surfaces with a sequence of temporally coherent deforming triangle mesh. We wish to allow these input surfaces to have arbitrary shapes and topology, and these shapes and topology are allowed to change significantly from one surface to the next. Because such data can come from a range of diverse sources in practice, we cannot assume any specific domain knowledge, nor can we assume that we are given additional information such as velocity fields. While surface tracking and registration is a widely studied problem, we are unaware of any tracking methods that are both robust to large deformations and arbitrarily complicated topology changes while retaining correspondence information. This is unfortunate, because frequent topology changes result from many common sources such as fluid dynamics, morphing, and erroneous scanned data.

To adequately solve this problem, we must define what it means

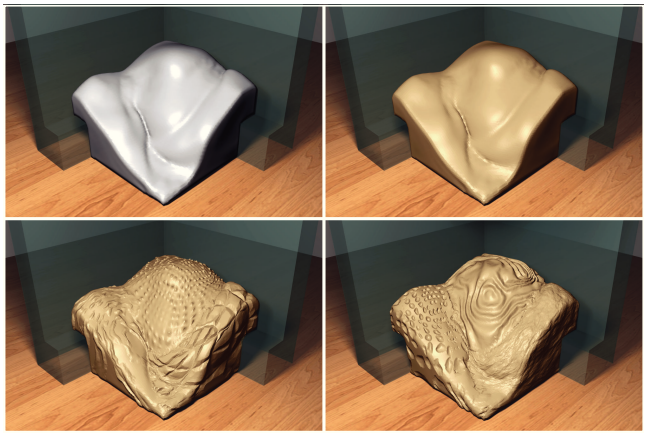

Figure 4: *Our method can turn a temporally incoherent mesh sequence (upper left) into a coherent one (upper right). We use this tracked mesh to add displacement maps as a post-process without having to re-simulate any physics.*

for two shapes to correspond in the presence of topology changes and find the most appropriate mapping between consecutive pairs of input surfaces. This correspondence information should gracefully propagate through changes in surface topology. We require our method to handle arbitrarily large plastic deformations through time while keeping the computation tractable.

4 Method

Our algorithm consists of several interwoven operations: mesh improvement (§4.1), non-rigid alignment (§4.2), and topological change (§4.3). The mesh improvement operation ensures that our output mesh $\mathbf{M}$ retains high-quality triangles while only minimally re-sampling geometry. The non-rigid alignment step ensures that $\mathbf{M}$ actually conforms to the desired shapes through time, and the topology change step ensures that the topology of $\mathbf{M}$ conforms to that of the desired input shapes $\{\mathbf{S}_n\}_{n=1}^{N}$ in each frame. We show that these three operations alone are enough to generate smoothly deforming meshes with high-quality geometry. However, in order to utilize these deforming meshes to their full extent, we also record correspondence information along the way (§4.4). Finally, we explain how to use the recorded correspondence information to efficiently propagate information forwards and backwards through time as a post-process (§4.6).

4.1 Mesh Improvement

A detailed surface mesh with well-shaped triangles is essential for a wide variety of beneficial computations. In addition to enhancing numerical stability in our non-rigid registration solver (§4.2) as well as the geometric intersection code in our topology change routine (§4.3), a triangle mesh free from degeneracies is necessary for such basic operations as interpolation, ray tracing, and collision detection. As we explain later in §5, the guaranteed mesh quality from our algorithm allows us to densely sample complex textures, generate displacement maps which are less prone to self-intersections, and solve partial differential equations on a deforming mesh using a finite element method.

In our framework, we follow the mesh improvement procedures outlined in the survey by Wojtan et al. [2011]. When edges become too long, we split them in half by adding a new vertex at the midpoint. When edges become too short, or when triangle interior angles or dihedral angles become too small, we perform an edge collapse by replacing an edge with a single vertex. Although we

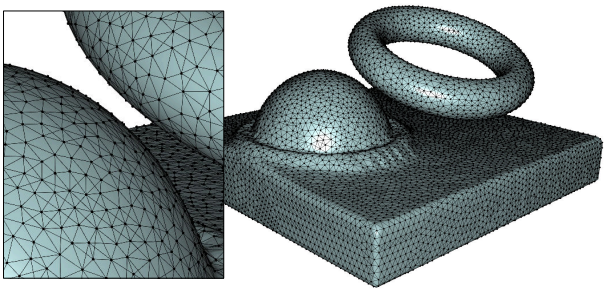

Figure 6: *Our mesh is augmented with a deformation graph for robust coarse-level non-rigid registration. We use geodesic distances to construct the graph in order to avoid edge connections between surfaces close in Euclidean space but far along geodesics.*

did not implement them in our framework, edge flips are also another excellent mesh re-sampling operation.

When improving a dynamically-deforming mesh, the main challenge is to find the right balance between high-quality triangles and excessive vertex re-sampling. Though we are free to customize these mesh improvement parameters however we like, we used similar parameters for all of the examples in this paper. We used a minimum interior angle of 10 degrees, a minimum dihedral angle of 45 degrees, and a maximum:minimum edge length ratio of 4:1. For a more in depth discussion on choosing parameters for these operations, please see [Wojtan et al. 2011].

4.2 Non-Rigid Alignment

Our goal is to establish correspondences between a source $\mathbf{M}$ and a target $\mathbf{S}_n$. If we assume that the two shapes have the same topology, we can solve this problem by warping $\mathbf{M}$ onto $\mathbf{S}_n$ while minimizing surface distances and shape distortion. In general, this assumption does not hold, but we may still use non-rigid registration to align the shapes. By simultaneously maximizing geometric similarity and rigidity, surface regions on $\mathbf{M}$ that are compatible with those on $\mathbf{S}_n$ will be aligned, providing dense correspondences within these regions.

We adapt the state-of-the-art bi-resolution registration framework by Li et al. [2009] for non-rigid alignment. Their method is split into two parts to maximize robustness and efficiency: a non-linear optimization that takes care of coarse alignment, and a linearized optimization that aligns fine-scale details. We describe these parts in the following two sections.

Coarse Non-Linear Alignment. Li et al. [2009]'s non-rigid iterative closest point algorithm alternates between estimating correspondences from $\mathbf{M}$ to $\mathbf{S}_n$, and non-rigid deformation of $\mathbf{M}$ that allows correspondences to slide along $\mathbf{S}_n$. Rigidity of the deformation model is relaxed whenever convergence is detected to avoid local minima.

Deformation is achieved using a coarse *deformation graph* that is constructed by uniformly sub-sampling $\mathbf{M}$ such that the distance between deformation graph nodes is four times larger than the average edge length of $\mathbf{M}$. Instead of computing displacements for each vertex of $\mathbf{M}$, we solve for an affine transformation $(\mathbf{A}_i, \mathbf{b}_i)$ for each graph node. The graph node transformations are transferred to the remaining vertices via linear blend skinning. Letting $\mathcal{N}(i)$ denote the $k = 4$ graph nodes nearest to $\mathbf{x}_i$, we describe the motion of $\mathbf{x}_i$ by a linear combination of the computed graph node transformations. Each $j \in \mathcal{N}(i)$ is weighted as $w_{ij} = (1 - d(\mathbf{x}_i, \mathbf{x}_j)/d_{\text{max}})^2$ and normalized such that $\sum_{j\in\mathcal{N}(i)} w_{ij} = 1$. Here, $d(\cdot,\cdot)$ denotes geodesic distance and d_{max} is the distance to the $(k+1)$th nearest graph node. The choice of using geodesic distances was made to ensure that a vertex is not influenced by graph nodes that are close in Euclidean distance but far along geodesics. This is important, *e.g.* in case of a breaking wave whose tip might be close to the surface in Euclidean but not geodesic distance. Instead of using the same connectivity as the original triangle mesh, a graph edge is formed whenever there exists a vertex in $\mathbf{M}$ influenced by two graph nodes.

When estimating correspondences, the original formulation matches vertices on $\mathbf{M}$ with the closest point on $\mathbf{S}_n$. We instead choose to project a vertex $\mathbf{x}_i$ onto $\mathbf{S}_n$ in the direction of the surface normal to obtain $\mathbf{c}_i$. We have found that this heuristic is significantly better at picking correspondences during large non-rigid deformations, especially where surfaces spread out into thin sheets. To avoid inconsistent alignments, we prune correspondences where the surface normals at $\mathbf{x}_i$ and $\mathbf{c}_i$ are more than 60 degrees apart, or where $\mathbf{c}_i$ is more than three times further from $\mathbf{x}_i$ than the closest point on $\mathbf{S}_n$.

To solve for the affine transformation, we minimize an energy functional that consists of a fitting and some regularization terms. The fitting energy measures how far $\mathbf{M}$ is from $\mathbf{S}_n$ according to the correspondences found above.

$$E_{\text{fit}} = \sum_{i\in\mathcal{V}} (\alpha_{\text{point}} \|\mathbf{x}_i - \mathbf{c}_i\|^2 + \alpha_{\text{plane}} \langle \mathbf{n}_i, \mathbf{x}_i - \mathbf{c}_i \rangle^2)$$

Here, $\mathcal{V}$ is the set of deformation graph nodes, $\mathbf{c}_i$ denotes the point on $\mathbf{S}_n$ mapped from $\mathbf{x}_i$ and $\mathbf{n}_i$ denotes the surface normal at $\mathbf{c}_i$. The parameters α_{point} and α_{plane} determine the relative importance of the corresponding point-to-point and point-to-plane energy terms. We use $\alpha_{\text{point}} = 0.1$ and $\alpha_{\text{plane}} = 1$ in all our examples. The larger weight for the point-to-plane term allows the correspondences to slide along $\mathbf{S}_n$ when solving for the deformation, leveraging the coupling between correspondence and deformation optimization.

A second term maximizes the rigidity of the affine transformation, thus minimizing distortion and scaling. This is accomplished by measuring how far $\mathbf{A}_i$ is from a true rotation matrix. Letting $\mathbf{a}_{i1}, \mathbf{a}_{i2}, \mathbf{a}_{i3}$ be the columns of $\mathbf{A}_i$, we obtain

$$\begin{aligned} E_{\text{rigid}} = \sum_{i\in\mathcal{V}} (&\langle \mathbf{a}_{i1}, \mathbf{a}_{i2} \rangle^2 + \langle \mathbf{a}_{i1}, \mathbf{a}_{i3} \rangle^2 + \langle \mathbf{a}_{i2}, \mathbf{a}_{i3} \rangle^2 \\ &+ (1 - \|\mathbf{a}_{i1}\|)^2 + (1 - \|\mathbf{a}_{i2}\|)^2 + (1 - \|\mathbf{a}_{i3}\|)^2) \end{aligned}$$

A final term ensures smoothness between edge connected nodes.

$$E_{\text{smooth}} = \sum_{i\in\mathcal{V}} \sum_{j\in\mathcal{N}(i)} \|\mathbf{A}_i(\mathbf{x}_j - \mathbf{x}_i) + \mathbf{x}_i + \mathbf{b}_i - (\mathbf{x}_j + \mathbf{b}_j)\|^2$$

The total energy $E_{\text{total}} = \alpha_{\text{fit}} E_{\text{fit}} + \alpha_{\text{reg}} (E_{\text{rigid}} + 0.1 E_{\text{smooth}})$ is minimized using a standard Gauss-Newton solver based on Cholesky decomposition. We alternate between correspondence estimation and surface deformation until convergence, and gradually relax the regularization by dividing α_{reg} by 10. For each $\mathbf{S}_n$ we initialize the optimization with $\alpha_{\text{fit}} = 0.1$ and $\alpha_{\text{reg}} = 1000$.

Fine-Scale Linear Alignment. While the coarse level optimization makes sure that large deformations between $\mathbf{M}$ and $\mathbf{S}_n$ are recovered, a second warping step uses a more efficient (but rotation-sensitive) linear mesh deformation technique to capture high-frequency geometric details in $\mathbf{S}_n$. For each vertex of $\mathbf{M}$, we trace an undirected ray in the normal direction and find the closest intersection point $\mathbf{c}_i$ on $\mathbf{S}_n$.

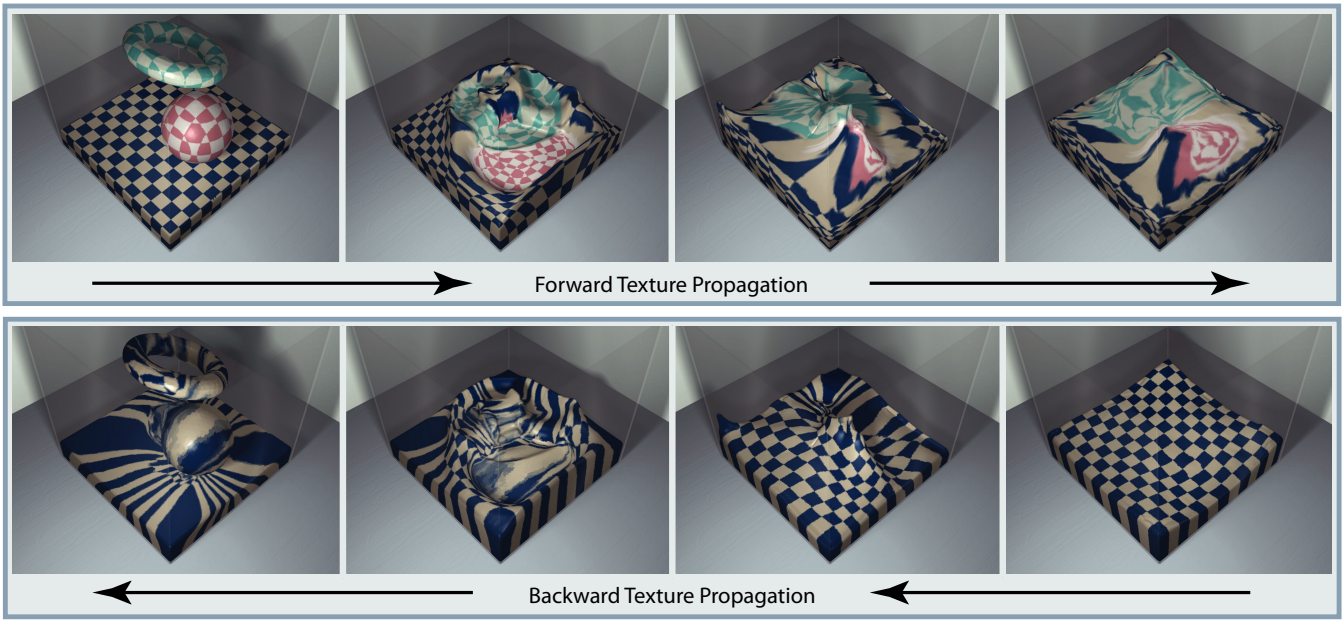

Figure 5: *These animations show how we can use our algorithm to propagate a texture both forwards and backwards through time. In the bottom animation, the fluid simulation naturally splashes around as it settles into a checker texture.*

The optimization uses a point-to-point fitting term $E_{\text{fit}} = \sum_{i \in \mathcal{V}} \|\mathbf{x}_i - \mathbf{c}_i\|^2$ and solves for the displacement of each vertex by minimizing the difference between adjacent vertex displacements using $E_{\text{reg}} = \sum_{(i,j) \in \mathcal{E}} |d_i - d_j|^2$.

To avoid self intersections, we prune correspondences that are further than a threshold $\sigma = 0.1$ given a scene bounding box diagonal of 1. Finally, we synthesize fine-scale details from the target on the pre-aligned mesh by minimizing $E_{\text{tot}} = E_{\text{fit}} + E_{\text{reg}}$ using an efficient conjugate gradient solver. Despite the robustness of the proposed non-rigid registration approach, we do not guarantee that every target surface region will have a corresponding source point. Such cases require a change in topology.

4.3 Topological Change

This paper considers a more general class of input deformations than most previous methods — we aim to track surfaces that are not only highly deformable, but that may change topology arbitrarily through time. For example, we allow new surface components to appear from nowhere in the middle of an animation, and we expect that entirely disparate surface regions may suddenly merge together. In order to accurately track such extreme behavior in the input data, we build new tools to constrain the topology of our mesh to that of an arbitrary closed input surface.

We base our topology change method on that of Wojtan et al. [2009] with subdivision stitching [2010] as explained in their SIGGRAPH course [2011]. The method takes as input a triangle mesh $\mathbf{M}$ and voxelizes its signed distance function $\phi_{\mathbf{M}}$ onto a volumetric grid. A cubic cell in the volumetric grid is classified as topologically complex if the intersection of $\mathbf{M}$ with the cell is more complex than what can be represented by a marching cubes reconstruction of $\phi_{\mathbf{M}}$ inside the cell. Topologically complex cells are candidates for re-sampling, and triangles of $\mathbf{M}$ inside such cells will be replaced by marching cubes triangles reconstructed from $\phi_{\mathbf{M}}$. This strategy forces $\mathbf{M}$ to change such that its topology matches that of $\phi_{\mathbf{M}}$, effectively making sure that $\mathbf{M}$ changes topology in the event that it intersects itself. See Wojtan et al. [2011] for a detailed exposition.

We chose to use this method primarily because of its flexibility and robustness. We would like the surface to change topology not only when the mesh intersects itself, but also whenever the input geometry happens to change its own topology. Furthermore, because this method is independent of surface velocity, it adds another layer of robustness to our algorithm; in the event that our registration routine produces inaccurate displacement information, the topology algorithm will correct the final shape by drawing new surface geometry directly from $\mathbf{S}_n$.

To do this, we generalize the idea of Wojtan et al.; instead of constraining the topology of the input mesh to match that of its own signed distance function, we constrain the input mesh to match the topology of *any voxelized implicit surface*. We simply voxelize an arbitrary implicit surface Θ, and replace the signed distance function $\phi_{\mathbf{M}}$ in the original with our new function Θ. The algorithm then compares the topology of the mesh $\mathbf{M}$ to the topology of Θ, and replaces $\mathbf{M}$'s triangles with triangles from the extracted isosurface of Θ wherever $\mathbf{M}$ and Θ have a different local topology. We can refer to this generalized topology change routine as ConstrainTopology$(\mathbf{M}, \Theta)$. Using this terminology, the original algorithm of Wojtan et al. can be executed by calling ConstrainTopology$(\mathbf{M}, \phi_{\mathbf{M}})$.

Within our deformation framework, we use this generalized topology change algorithm in two ways: first to ensure that the deforming mesh changes topology if it intersects itself, and second, to ensure that the deforming mesh has the same topology as the target input data. These actions can be computed by calling ConstrainTopology$(\mathbf{M}, \phi_{\mathbf{M}})$ and ConstrainTopology$(\mathbf{M}, \phi_{\mathbf{S}_n})$, respectively, where $\mathbf{S}_n$ is the target mesh from the input data. We will specify the exact order in which to call these functions in section §4.5.

4.4 Recording Correspondence Information

Throughout the computation of our deforming mesh $\mathbf{M}$, we want to track how its correspondences evolve through time. The previously mentioned mesh modification routines can cause significant changes in correspondence information, and we must track how these changes occur.

The mesh deformation algorithm described in §4.2 is Lagrangian in nature, so it moves individual vertices to their new locations at each frame in the animation sequence. Consequently, the vast majority of vertex locations in our mesh at a given frame number correspond exactly to the location of that same vertex at earlier and later frame numbers. For these vertices, information about their corresponding position at different points in the sequence is implicit; vertex i in frame number $n-1$ corresponds exactly with i in frame n.

The only vertices which do *not* have this trivial correspondence with vertices in different frames are the few vertices which were created or destroyed due to re-sampling. Within our framework, the only way to create new vertices is via topological change (§4.3) or edge and triangle subdivision (§4.1). The only way for us to destroy vertices is via topological change (§4.3) or edge collapse (§4.1). Note that some other potential mesh improvement procedures like mesh fairing [Jiao 2007; Brochu and Bridson 2009; Stam and Schmidt 2011] improve triangle quality at the expense of re-sampling correspondence information by diffusing it along the surface. For this reason, we did not use such fairing procedures in §4.1.

For each transition between two frames, we track these re-sampling events (edge subdivision, triangle subdivision, edge collapse, topology change) in what we call an *event list*. The event list stores detailed information about each re-sampling event, and it is sorted by the order in which the re-sampling events took place. Each event in the list records information of the form $(\mathcal{V}_{\text{in}}, \mathcal{V}_{\text{out}}, f(\mathcal{V}_{\text{in}}), b(\mathcal{V}_{\text{out}}))$, where $\mathcal{V}_{\text{in}}$ is a set of the input vertices, $\mathcal{V}_{\text{out}}$ is a set of the output vertices, $f(\mathcal{V}_{\text{in}})$ is a function that assigns information to $\mathcal{V}_{\text{out}}$ as a function of $\mathcal{V}_{\text{in}}$, in case we want to propagate information forwards. Similarly, $b(\mathcal{V}_{\text{out}})$ is a function that assigns information to $\mathcal{V}_{\text{in}}$ as a function of $\mathcal{V}_{\text{out}}$, in case we want to propagate information backwards.

When we subdivide an edge (i, j) between two vertices i, j, a new vertex k is created on the line connecting i and j. In this case, the event list records $(\{i, j\}, \{i, j, k\}, k \mapsto (i, \alpha, j, 1-\alpha), \emptyset)$, where α denotes the barycentric coordinate of k. Notice that we omit the trivial correspondences $i \mapsto (i, 1)$ and $j \mapsto (j, 1)$. Similarly, when we subdivide a triangle (i, j, k) by adding a new vertex l somewhere inside the triangle, we record $(\{i, j, k\}, \{i, j, k, l\}, l \mapsto (i, \alpha_i, j, \alpha_j, k, \alpha_k), \emptyset)$. As before, $\alpha_i, \alpha_j, \alpha_k$ denote barycentric coordinates. Finally, when we collapse an edge (i, j) and replace the two endpoints i, j by a new vertex k at the barycenter of i and j, we record $(\{i, j\}, \{k\}, k \mapsto (i, 0.5, j, 0.5), \{i \mapsto (k, 1), j \mapsto (k, 1)\})$.

When a topological change occurs, surfaces can split wide open and entire patches of new geometry can be created. For each patch of new geometry after the topology change, we propagate information from the vertices on the boundary of the patch inward, using a breadth-first graph marching algorithm (similar to Yu et al. [2012]). Though several propagation strategies are valid at this point (during the marching algorithm, each new vertex could simply copy information from its nearest neighbor, it could distribute information evenly throughout the patch, e.g. by solving an elliptic PDE, etc.), we chose a strategy of each vertex taking the average of the information from its visited neighbors during the breadth-first march. For each new vertex that is created, our event list records the list of boundary vertices, the new vertex, and the linear combination of boundary vertices that results from this marching and averaging. There is no backward correspondence assignment for these vertices.

Lastly, vertices can be deleted in a topological merge. We treat such operations the same way that we treat new vertices that result from a topology change, but in reverse: before the patch of vertices is destroyed, we march inward from the boundary of the patch of deleted vertices and propagate information using the same averaged vertex scheme. For each vertex that is deleted, our event list records the list of boundary vertices, the new vertex, a null forward operation, and the linear combination of boundary vertices that results from the marching and averaging operation.

```
 1: Mesh M = LoadTargetMesh(S_1)
 2: ImproveMesh(M)
 3: for frame n = 2 → N do
 4:    LoadTargetMesh(S_n)
 5:    CoarseNonRigidAlignment(M, S_n)
 6:    FineLinearAlignment(M, S_n)
 7:    ImproveMesh(M)
 8:    φ_M := CalculateSignedDistance(M)
 9:    ConstrainTopology(M, φ_M)
10:    φ_{S_n} := CalculateSignedDistance(S_n)
11:    ConstrainTopology(M, φ_{S_n})
12:    ImproveMesh(M)
13:    SaveEventListToDisk(n)
14:    SaveMeshToDisk(M)
15: end for
```

Algorithm 1: *Pseudocode for our mesh-tracking algorithm.*

4.5 Summary of the Tracking Algorithm

We review the steps of our tracking method in Algorithm 1. Our method begins by initializing a triangle mesh $\mathbf{M}$ to the first frame of the input mesh sequence $\{\mathbf{S}_n\}_{n=1}^N$. We then immediately call our mesh improvement routine (§4.1) to ensure that $\mathbf{M}$ consists of high-quality geometry. Next, we enter the main loop of our algorithm, which visits each of the input meshes $\mathbf{S}_n$ in turn. In each iteration we use our course non-rigid alignment routine (§4.2) to align the low-resolution features of $\mathbf{M}$ as closely as possible with those of $\mathbf{S}_n$. Once the coarse alignment has terminated, we perform a fine-scale linearized alignment in order to ensure that all of the high-resolution details of $\mathbf{M}$ line up with $\mathbf{S}_n$. At this point in the algorithm, we have deformed our mesh $\mathbf{M}$ such that it lines up with the input data frame $\mathbf{S}_n$. This deformation may cause the triangles of $\mathbf{M}$ to stretch and compress arbitrarily, so we again perform a mesh improvement in order to clean up the overly deformed elements.

Next, we must account for the fact that $\mathbf{M}$ may have self-intersections. We execute the basic topology change algorithm in §4.3 by first computing a voxelized signed distance function near the surface of $\mathbf{M}$ and then ensuring that $\mathbf{M}$ has the same topology as the zero isosurface of this function. This step mainly cleans up any large self-intersections in the mesh by merging surface patches together. Next, we execute a topology change algorithm again, but this time we constrain $\mathbf{M}$ to match the topology of $\mathbf{S}_n$. This step ensures that we split apart any surfaces in $\mathbf{M}$ which stretches over gaps in $\mathbf{S}_n$, as well as merge any separate regions of $\mathbf{M}$ that are actually merged in $\mathbf{S}_n$. The extra topology constraint additionally acts as a fail-safe by re-sampling parts of $\mathbf{M}$ in the rare event that the alignment algorithm was unable to find correspondences for all of $\mathbf{M}$.

At this point in the algorithm, our mesh $\mathbf{M}$ can consist of triangles with arbitrarily poor aspect ratios, because the topological sewing algorithm only cares about the connectivity of the mesh and not the condition of the individual mesh elements. We therefore call our mesh improvement routine once again to ensure that the mesh is fit for another round of tracking. Note that throughout this entire algorithm, we document any re-sampling operations that occur (potentially in lines 2, 7, 9, 11, and 12 of Algorithm 1) and add them to our event list (§4.4). In the final two steps of this loop, we save our event list and the mesh $\mathbf{M}$ itself to disk. We then start the loop again with the next frame of animation $\mathbf{S}_{n+1}$.

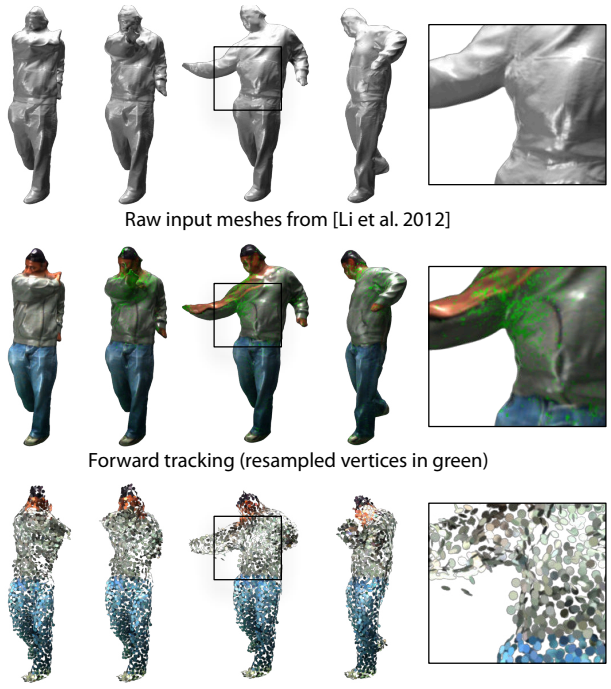

Figure 7: *Top: Input performance capture data has inconsistent vertices across frames and exhibits topological variations. Middle: Our method seamlessly handles topology changes and ensures high-quality triangles. Resampled vertices from our mesh improvement algorithm are marked in green. Bottom: The method of Tevs et al. (visualized as a point cloud) is prohibitively expensive for long, detailed mesh sequences and fails to capture the correct motion.*

4.6 Propagating Information as a Post-Process

After we have finished tracking the input geometry (after all of the steps in §4.5 have run until completion), we have a series of temporally coherent animation frames of a mesh $\mathbf{M}$ that deforms and changes topology. Furthermore, we also have a per-frame event list that describes exactly how correspondences propagate throughout the animation. We can use this list to pass information like surface texture and surface velocity from one frame to the next. To pass information forward in time, we run through the event list in the order that each event took place, and, using the notation from section 4.4, we pass information to re-sampled vertices using the function $f(\mathcal{V}_{\text{in}})$. Similarly, we pass information backwards in time by running backwards through the event list and using $b(\mathcal{V}_{\text{out}})$.

5 Applications

Having detailed our method for obtaining a temporally coherent parameterization of an arbitrary sequence of closed manifold meshes (§4), we shift our focus to applications. We show how we can apply our method to track a broad range of different incoherent surfaces and how we can exploit extracted correspondence information to significantly enhance the meshes in a variety of different ways.

Displacement Maps. Displacement maps provide an efficient way of adding geometric detail to an animation as a post-process, avoiding costly re-simulation or geometry acquisition. We recover a temporally coherent mesh sequence from a physically-based Eulerian viscoelastic simulation [Goktekin et al. 2004] with a periodically re-sampling surface tracker similar to [Müller 2009] (Figure 4). Our method faithfully conforms to the target shape in every frame with minimal re-sampling.

To showcase our temporally coherent parameterization and high mesh quality, we apply two different displacement maps to the mesh sequence. We represent a displacement map as a per-vertex scalar designating the normal direction displacement of each vertex. Using our data structure (§4.6), we propagate displacements applied in the first frame to all later frames. Compared to tracking, propagation is almost instantaneous, taking only a few seconds for the entire animation. Swapping in a different displacement map is thus fast and effortless. Compare this to the state of the art without our method, where an animator instead would have to re-run the entire simulation to change the geometry.

Color. It is often useful to texture implicit surfaces in production [Sumner et al. 2003; Wiebe and Houston 2004]. Because of the large computational costs of liquid simulation, it is particularly convenient to add detail to a lower resolution simulation as a post-process, *e.g.* by applying foam or deep water textures. Figure 5 shows a splashy liquid scene which comes from a standard Eulerian solver using the Level Set Method [Osher and Fedkiw 2003] to track the free-surface. We track an incoherent sequence of marching cubes reconstructions of the level sets from the simulation.

Similar to displacement maps, we propagate colors applied in the first frame to all later frames. Our accompanying video shows a checkerboard pattern and a lava texture propagated through time. Further exploiting our temporal data structure, we propagate colors applied in the last frame *backwards* in time to the first frame (Figure 5). This technique allows us to enhance the splashy animation with an interesting artistic expression where an image is slowly revealed as the dynamics settle (Figure 1).

Wave simulation. Texture is one way of enhancing a low resolution liquid simulation, however, correct computation of light transport for effects like caustics is easier with real geometry. Our method is not limited to static displacement maps (mentioned above), but allows for procedural displacements as well. In particular we may improve the fidelity of the splashy liquid simulating mentioned above (Figure 2) by adding a dynamic displacement map. Because our method yields particularly high-quality surface triangles with minimal re-sampling, we are able to use the resulting mesh to solve partial differential equations. Inspired by recent fluid animation research [Thürey et al. 2010; Yu et al. 2012], we augment our surfaces with a time-varying displacement map, computed as the solution to a second order wave equation:

$$\frac{\partial^2 h}{\partial t^2} = c^2 \nabla^2 h. \tag{1}$$

Here, h is wave displacement in the normal direction, ∇^2 is the discrete Laplace operator computed with cotangent weights [Botsch et al. 2010], and c is a user-chosen wave speed. We use our event list to transfer the state variables (wave heights h and velocities in the normal direction v) from one frame to the next, and we integrate the system using symplectic Euler integration with several sub-cycled time steps per input frame. One may optionally choose to add artificial damping to the simulation for artistic reasons by multiplying h by a $(1-\varepsilon)$ factor in each step. No artificial damping was used in our simulations.

Our wave simulation method is novel in that it retains tight control over wave energy sources. We only add wave heights precisely at the locations in space-time where topological changes occur. This stands in opposition to previous work, which recomputes wave heights every time step based on surface geometry. The result

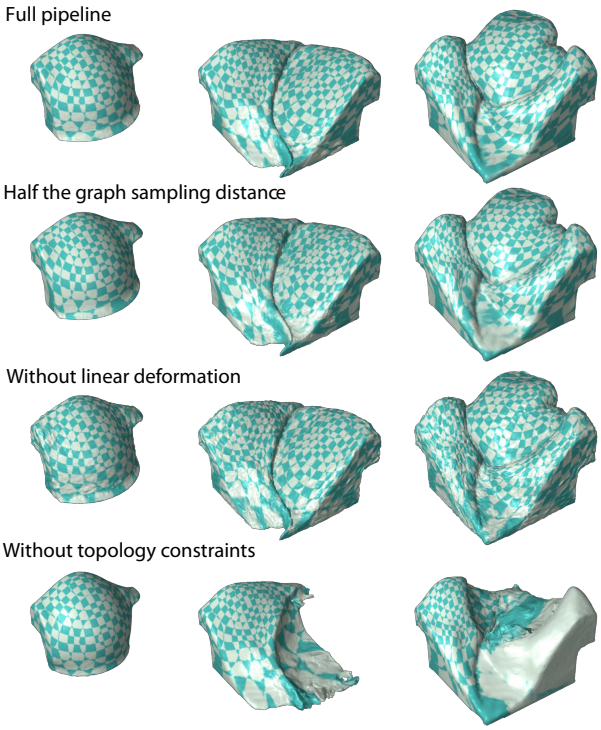

Figure 8: *Comparison between our full pipeline and leaving out individual stages of our surface tracking framework.*

of this distinction is that our simulations are much less likely to introduce energy due to numerical errors. Our simulations have a dramatically high signal-to-noise ratio – we can clearly see interesting wave interference patterns persist throughout the entire simulation.

Morph. Another application of our method is transferring colors through morphs that change topology between arbitrary genera (Figure 3). We use a simple linear blend between signed distance functions to create the morph, and we subsequently obtain a coherent mesh by tracking it with our framework. We start by propagating color backwards from the final frame, and then we use the colors which were propagated to the first frame to obtain a base texture. In this way an artist can see where important feature points end up on the target shape to aid in creating a more natural morph. To obtain the morph in Figure 3, we additionally blend between the two forward and backward propagated colors.

Performance Capture. Performance capture has numerous applications such as video games and filmmaking. Due to noise and occlusion, captured data often exhibits non-physical topology changes. Unlike previous methods, we are able track captured data with topology changes while obtaining temporally coherent correspondences (Figure 7). We apply a texture in the first frame and propagate it forward. Regions that are unoccluded throughout the sequence are tracked faithfully.

6 Evaluation

We performed an extensive series of tests to evaluate our method. We used the viscoelastic simulation (Figure 4) as a testbed while we varied parameters, turned off various parts of our code, and attempted alternative approaches. Please see our accompanying video for visualizations of these tests.

In Figure 9, we show how our method compares to the naïve approach of simply projecting the tracked mesh $\mathbf{M}$ onto the input $\mathbf{S}_n$. Tangential drift is severe even in the case of simple translation. Next, we compare our method to one without fine-scale registration (line 6 of Algorithm 1). Since the graph-based registration works on a coarse scale and only influences vertices in $\mathbf{M}$ through linear blend weights, this modified method is unable to correctly register small features. Such errors accumulate over time, causing a rough, lumpy surface that ignores the fine-scale details of the input. Our full algorithm clearly does not exhibit these problems, showing why the fine-scale optimization of Section 4.2 is necessary.

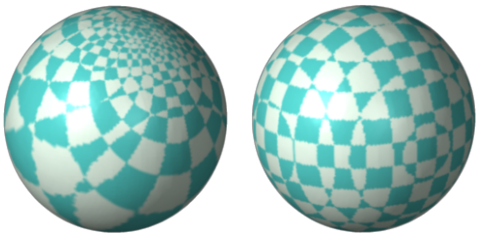

Figure 9: *The difference between projection (left) and our non-rigid registration technique (right). Simple projection causes severe distortion of the surface, while our registration reliably provides accurate correspondences.*

Our tests also show that the topology constraint (§4.3, line 11 of Algorithm 1) is essential for robust tracking. The tests in our video illustrate how a method without this constraint is unable to cope with drastic changes in input topology. An obvious example in the viscoelastic simulation is the sudden introduction of new components in later frames — when the topology constraint is turned off, the non-rigid registration algorithm was unable to recognize these components without manually creating a template. Another important feature of the topology constraint is that it acts as a convenient failsafe. Should the registration routine fail to fully conform to the target shape, the topology constraint fills in regions of mismatched geometry. As a result, our full algorithm is quite robust to poor parameter choices for the alignment, and poor alignment only leads to additional re-sampling (as opposed to an unrecoverable failure).

Within a given frame, time complexity of our method is dominated by coarse non-rigid alignment (§4.2, line 5 of Algorithm 1). Sampling density of the deformation graph is the most critical parameter to the time complexity of the non-rigid alignment, since it dictates the number of variables in the non-linear optimization problem. Sensitivity of our algorithm to different sampling densities is examined in the supplementary video. In addition to the density used in Figure 4, we also ran the algorithm with both half- and quarter-sampling density. The video shows that reduced sampling densities lead to increased re-sampling, but the result remains similar to our high-quality tracking. Conveniently, this allows use of a lowered sampling density to get a fast approximation of the algorithm's output before committing to solving with a high sampling density.

Another way to reduce the time complexity of our method is to use sparser input. We experimented with five, ten and twenty-five times sparser input than the results shown in Figure 4. As seen in the video, our method is robust to sparse input and produces reasonable correspondences, even for the example where we use only sixteen out of the original 400 frames in the input for tracking.

The memory complexity of our algorithm is similarly dominated by the non-rigid alignment. However, because we only do pairwise alignment, memory consumption is independent of the length of the sequence of input data. In other words the space complexity scales with the number of vertices in $\mathbf{M}$.

We have gathered statistics for all of our application examples. We summarize these results in Table 1. All measurements were per-

	ViscElast	Splash	Morph	PerfCap
Vertices	60k-300k	280k-380k	77k-96k	214k-369k
Frames	400	500	100	111
Frame time	45-153s	105-220s	17-21s	150-174s
Coarse reg.	87-93%	81-89%	67-73%	70-73%
Fine reg.	3-8%	11-18%	19-23%	24-27%

Table 1: *Summery of statistics for our application examples. Time spent on mesh improvement and topology changes is negligible compared to alignment and is omitted in the table. Timings exclude file I/O operations.*

formed on a standard PC with an Intel i7-2600K processor and 16 GB of memory. We note that our implementation has not been optimized for performance and is mostly sequential.

Comparison to other methods. As detailed in section 2, the method of Stam et al. [2011] is significantly different from ours. While this is an admittedly biased comparison, we show how our method performs with their example of three blended blobs rotating about the origin (see Figure 10). Our algorithm explicitly solves for the globally most rigid deformation, so we obtain practically perfect tracking whereas Stam et al. show slight tangential drift and color diffusion. We imagine their problem would be exacerbated with larger time steps, while ours remains accurate.

7 Discussion

Since our tracking approach is sequential and does not rely on higher level deformation priors, we do not guarantee drift free tracking. For purposes such as tracking extended performance capture recordings, dynamic body shape statistics and elastic deformation models could be incorporated to prevent accumulations of tracking errors. More generally one could exploit the temporal mesh sequence by combining the result of forwards and backwards tracking [Kim et al. 2007; Kagaya et al. 2011]. Nevertheless, none of our examples, including the performance capture example, exhibited any noticeable drift when propagating the texture from the first frame to the end despite the drastic topology variations and large deformations in the input data. Therefore, we did not further investigate these temporal schemes.

Our temporally coherent meshes retain high-quality elements and low-valence vertices, even in the presence of highly non-rigid deformations and topology changes, as can be seen in our supplementary video. Our bound on triangle and dihedral angles in particular make sure that high-valence vertices and skinny triangles are avoided. Since we do not currently implement edge flips, we resample vertices more often than we would otherwise, especially when the surface is compressing or stretching. As our results show, this does not turn our to be a big problem in practice, however, we would like to implement edge flips in the future.

Our method is meant to find surface correspondences through arbitrary deformations while remaining faithful to the input motion. When given a severely stretched deformation as input, an exactly tracked set of surface correspondences will inevitably exhibit severe stretching as well. In situations such as this, a minimally distorted mapping through time is actually incorrect behavior as far as our algorithm is concerned. Our strategy of matching geometry while minimizing tangential drift unavoidably causes distortion when propagating visual data such as texture, as can be seen in *e.g.* Figure 5. Our method does, however, allow the user to relax the energy term that punishes tangential drift, thereby giving some control over distortion. If specific requirements are sought, such as maximal conformality of textures, one would have to tailor our method

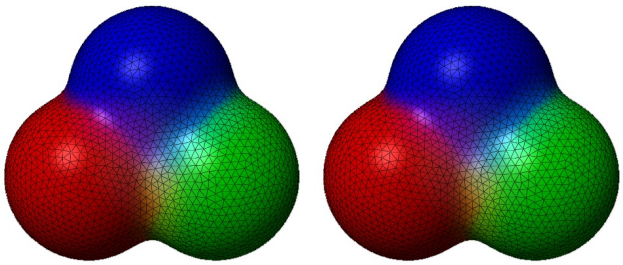

Figure 10: *Stam and Schmidt introduced this shape as a benchmark for evaluating the accuracy of an implicit surface tracking algorithm. After one complete rotation, our algorithm's output (right) is virtually identical to the analytical solution (left).*

for that particular use-case. We view this as an interesting direction for future work. Texture synthesis [Bargteil et al. 2006b; Kwatra et al. 2007] is but one possible solution to the challenging problem of texture stretching.

Because our method is based on shape matching, we are unable to track surfaces invariant under our energy functions; a surface with no significant geometric features (like a rotating sphere) will not be tracked accurately. However, it would be easy to augment our method with additional priors such as velocity information in order to handle such featureless cases.

The biggest limitation of our method is the fact that we are currently limited to closed manifold surfaces due to the algorithm we use for performing topology changes. This method assumes that for any arbitrary point in space we must unambiguously decide whether it is inside or outside the surface.

8 Conclusion

We have presented a novel approach that takes a sequence of arbitrary closed surfaces and produces as output a temporally coherent sequence of meshes augmented with vertex correspondences. The output of our algorithm is useful for a variety of applications such as (dynamic) displacement maps, texture propagation, template-free tracking and morphs. We have also demonstrated the robustness of the method to parameters as well as input. In the future we would like to extend the method to handle non-closed surfaces, as well as explore problem-specific applications of our general-purpose framework.

Acknowledgements

We would like to thank Xiaochen Hu for implementing mesh conversion tools, Duygu Ceylan for helping with the rendering, and Art Tevs for the human performance data comparison. We also thank Nils Thuerey and Christopher Batty for helpful discussions. This work is supported by the SNF fellowship PBEZP2-134464.

References

BARGTEIL, A., GOKTEKIN, T., O'BRIEN, J., AND STRAIN, J. 2006. A semi-lagrangian contouring method for fluid simulation. *ACM Transactions on Graphics (TOG) 25*, 1, 19–38.

BARGTEIL, A., SIN, F., MICHAELS, J., GOKTEKIN, T., AND O'BRIEN, J. 2006. A texture synthesis method for liquid animations. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, Eurographics Association, 345–351.

BOTSCH, M., KOBBELT, L., PAULY, M., ALLIEZ, P., AND LEVY, B. 2010. *Polygon mesh processing*. AK Peters Ltd.

BROCHU, T., AND BRIDSON, R. 2009. Robust topological operations for dynamic explicit surfaces. *SIAM Journal on Scientific Computing 31*, 4, 2472–2493.

BROCHU, T., BATTY, C., AND BRIDSON, R. 2010. Matching fluid simulation elements to surface geometry and topology. *ACM Transactions on Graphics (TOG) 29*, 4, 47:1–47:9.

CAMPEN, M., AND KOBBELT, L. 2010. Exact and robust (self-) intersections for polygonal meshes. *Computer Graphics Forum (Eurographics) 29*, 2, 397–406.

CHANG, W., LI, H., MITRA, N. J., PAULY, M., AND WAND, M. 2010. Geometric registration for deformable shapes. In *Eurographics 2010: Tutorial Notes*.

DINH, H., YEZZI, A., TURK, G., ET AL. 2005. Texture transfer during shape transformation. *ACM Transactions on Graphics (TOG) 24*, 2, 289–310.

DU, J., FIX, B., GLIMM, J., JIA, X., LI, X., LI, Y., AND WU, L. 2006. A simple package for front tracking. *Journal of Computational Physics 213*, 2, 613–628.

ENRIGHT, D., MARSCHNER, S., AND FEDKIW, R. 2002. Animation and rendering of complex water surfaces. *ACM Transactions on Graphics (TOG) 21*, 3, 736–744.

GOKTEKIN, T., BARGTEIL, A., AND O'BRIEN, J. 2004. A method for animating viscoelastic fluids. *ACM Transactions on Graphics (TOG) 23*, 3, 463–468.

JIAO, X. 2007. Face offsetting: A unified approach for explicit moving interfaces. *Journal of computational physics 220*, 2, 612–625.

KAGAYA, M., BRENDEL, W., DENG, Q., KESTERSON, T., TODOROVIC, S., NEILL, P. J., AND ZHANG, E. 2011. Video painting with space-time-varying style parameters. *IEEE Transactions on Visualization and Computer Graphics (TVCG) 17*, 74–87.

KIM, B., LIU, Y., LLAMAS, I., AND ROSSIGNAC, J. 2007. Advections with significantly reduced dissipation and diffusion. *IEEE Transactions on Visualization and Computer Graphics (TVCG) 13*, 135–144.

KWATRA, V., ADALSTEINSSON, D., KIM, T., KWATRA, N., CARLSON, M., AND LIN, M. 2007. Texturing fluids. *IEEE Transactions on Visualization and Computer Graphics (TVCG) 13*, 939–952.

LI, H., ADAMS, B., GUIBAS, L. J., AND PAULY, M. 2009. Robust single-view geometry and motion reconstruction. *ACM Transactions on Graphics (TOG) 28*, 5, 175:1–175:10.

LI, H., LUO, L., VLASIC, D., PEERS, P., POPOVIĆ, J., PAULY, M., AND RUSINKIEWICZ, S. 2012. Temporally coherent completion of dynamic shapes. *ACM Transactions on Graphics (TOG) 31*, 1, 2:1–2:11.

MITRA, N. J., FLORY, S., OVSJANIKOV, M., GELFAND, N., GUIBAS, L., AND POTTMANN, H. 2007. Dynamic geometry registration. In *Proceedings of the fifth Eurographics Symposium on Geometry Processing (SGP)*, Eurographics Association, 173–182.

MÜLLER, M. 2009. Fast and robust tracking of fluid surfaces. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, ACM, 237–245.

OSHER, S., AND FEDKIW, R. 2003. *Level set methods and dynamic implicit surfaces*, vol. 153. Springer.

PONS, J., AND BOISSONNAT, J. 2007. Delaunay deformable models: Topology-adaptive meshes based on the restricted delaunay triangulation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 1–8.

SHARF, A., ALCANTARA, D. A., LEWINER, T., GREIF, C., SHEFFER, A., AMENTA, N., AND COHEN-OR, D. 2008. Space-time surface reconstruction using incompressible flow. *ACM Transactions on Graphics (TOG) 27*, 5, 110:1–110:10.

STAM, J., AND SCHMIDT, R. 2011. On the velocity of an implicit surface. *ACM Transactions on Graphics (TOG) 30*, 3, 21:1–21:7.

SUMNER, N., HOON, S., GEIGER, W., MARINO, S., RASMUSSEN, N., AND FEDKIW, R. 2003. Melting a terminatrix. In *ACM SIGGRAPH 2003 Sketches & Applications*, ACM.

SÜSSMUTH, J., WINTER, M., AND GREINER, G. 2008. Reconstructing animated meshes from time-varying point clouds. *Computer Graphics Forum (SGP) 27*, 5, 1469–1476.

TEVS, A., BERNER, A., WAND, M., IHRKE, I., BOKELOH, M., KERBER, J., AND SEIDEL, H.-P. 2012. Animation cartography - intrinsic reconstruction of shape and motion. *ACM Transactions on Graphics (TOG) 31*, 2, 12:1–12:15.

THÜREY, N., WOJTAN, C., GROSS, M., AND TURK, G. 2010. A multiscale approach to mesh-based surface tension flows. *ACM Transactions on Graphics (TOG) 29*, 4, 48:1–48:10.

WAND, M., JENKE, P., HUANG, Q.-X., BOKELOH, M., GUIBAS, L., AND SCHILLING, A. 2007. Reconstruction of deforming geometry from time-varying point clouds. In *Proceedings of the fifth Eurographics Symposium on Geometry Processing (SGP)*, Eurographics Association, 49–58.

WAND, M., ADAMS, B., OVSJANIKOV, M., BERNER, A., BOKELOH, M., JENKE, P., GUIBAS, L., SEIDEL, H.-P., AND SCHILLING, A. 2009. Efficient reconstruction of nonrigid shape and motion from real-time 3d scanner data. *ACM Transactions on Graphics (TOG) 28*, 2, 15:1–15:15.

WANG, H., LIAO, M., ZHANG, Q., YANG, R., AND TURK, G. 2009. Physically guided liquid surface modeling from videos. *ACM Transactions on Graphics (TOG) 28*, 3, 90:1–90:11.

WIEBE, M., AND HOUSTON, B. 2004. The tar monster: Creating a character with fluid simulation. In *ACM SIGGRAPH 2004 Sketches & Applications*, ACM.

WOJTAN, C., THÜREY, N., GROSS, M., AND TURK, G. 2009. Deforming meshes that split and merge. *ACM Transactions on Graphics (TOG) 28*, 3, 76:1–76:10.

WOJTAN, C., THÜREY, N., GROSS, M., AND TURK, G. 2010. Physics-inspired topology changes for thin fluid features. *ACM Transactions on Graphics (TOG) 29*, 4, 50:1–50:8.

WOJTAN, C., MÜLLER-FISCHER, M., AND BROCHU, T. 2011. Liquid simulation with mesh-based surface tracking. In *ACM SIGGRAPH 2011 Courses*, ACM.

YU, J., WOJTAN, C., TURK, G., AND YAP, C. 2012. Explicit mesh surfaces for particle based fluids. *Computer Graphics Forum (Eurographics) 31*, 2, 41–48.

ZAHARESCU, A., BOYER, E., AND HORAUD, R. P. 2007. TransforMesh: a topology-adaptive mesh-based approach to surface evolution. In *Proceedings of the Eighth Asian Conference on Computer Vision*, Springer, vol. II of *LNCS 4844*, 166–175.