Globally Consistent Space-Time Reconstruction

by

Ian James South-Dickinson

B.Sc., Oregon State University, 2007

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA
(Vancouver)

September 2012

Abstract

We present a novel algorithm for space-time reconstruction of deforming meshes. Based on partial meshes at every frame, and sparse optical flow information between frames, we reconstruct a globally consistent, cross-parameterized, and hole filled sequence of meshes. Our method is based on pair-wise merging of frame sequences while correcting for changes in topology, filling in missing geometry, and repairing inconsistencies. We also introduce a robust method for filling in missing geometry in each frame of the sequence using geometry from another frame. Using this method we can propagate geometry over the full frame sequence, correcting errors and filling in holes even in regions of the object that are not observed in the input meshes for extended periods of time. Unlike other approaches, our method does not require template geometry, nor is it limited to narrow classes of objects or purely isometric deformations.

Preface

The system in this paper was a joint development between Prof. Alla Sheffer, Prof. Wolfgang Heidrich, Tiberiu Popa, Derek Bradley and myself.

Prof. Sheffer and Prof. Heidrich supervised the project, and Derek Bradley helped with the capture setup and processing, and contributed to the optical flow tracking.

Tiberiu Popa and I together developed the core system. We co-developed the optical flow (Section 4.1) and local patch-based parametrization (Section 4.2). Tiberiu developed the Parameterization Assembly (Section 4.3) and Analysis and Correction (Section 4.4), while I developed the Geometry Completion (Chapter 5).

Part of this thesis was published in the following paper [28]:

- T. Popa, I. South-Dickinson, D. Bradley, A. Sheffer, and W. Heidrich. Globally consistent space-time reconstruction. Computer Graphics Forum, 29(5):1633-1642, 2010.

Table of Contents

List of Figures

Acknowledgements

First of all, I would like to thank my supervisor Prof. Alla Sheffer for her guidance and support during my graduate studies. I would also like to thank my collaborator on this project, Tiberiu Popa, it has been a pleasure working with you. In addition, I would like to thank my second reader, Prof. Wolfgang Heidrich, for his helpful feedback.

I would like to thank all the great collaborators and friends I worked with during my time in the the DGP and Imager research groups, Derek, Vlady, Hongbo, James, Xi, and Cody.

Finally, I would like to thank the Graphite developers for providing the essential underlying tools that were used to create this work, and Andrei Sharf for the use of his virtual scanning software that was used to create synthetic data.

Dedication

I would like to dedicate this thesis to my wonderful family, who supported me in going to another country for graduate school, and to all the wonderful friends I made here at UBC. In addition, I'd like to dedicate this to Kat and Edwin, who put up with me while I got this finished up.

Chapter 1

Introduction

1.1 Motivation

Geometry scanning and capture techniques in combination with surface reconstruction methods provide a powerful and convenient way to create virtual replicas of real-world objects. Advanced scanning methods are now capable of providing raw capture data, typically point clouds, for dynamically deforming shapes. The reconstruction of geometry and motion from this data is the natural next step. While a variety of methods allow for the independent reconstruction of geometry in each frame, significantly better results can be obtained by accumulating information over time, thus leveraging knowledge of the temporal behaviour of the captured objects (e.g. [7, 13, 27, 32]). This prior knowledge is particularly critical to the reconstruction of the object's motion or frame-to-frame geometry correspondence.

Most deforming objects change their shape gradually both in terms of Euclidean coordinates and in terms of intrinsic surface shape. This holds true for any articulated shape, humans, animals, garments, and many other objects in our everyday surrounding. This *gradual change* observation effectively implies that no discrete changes that drastically affect the intrinsic shape, such as a change in the object genus, are possible. Note that the fact that the change is gradual does not prevent large changes in the shape over time, such as the dog's paws and chest motion in Figure 1.1. The space-time reconstruction of such shapes should ideally satisfy this prior.

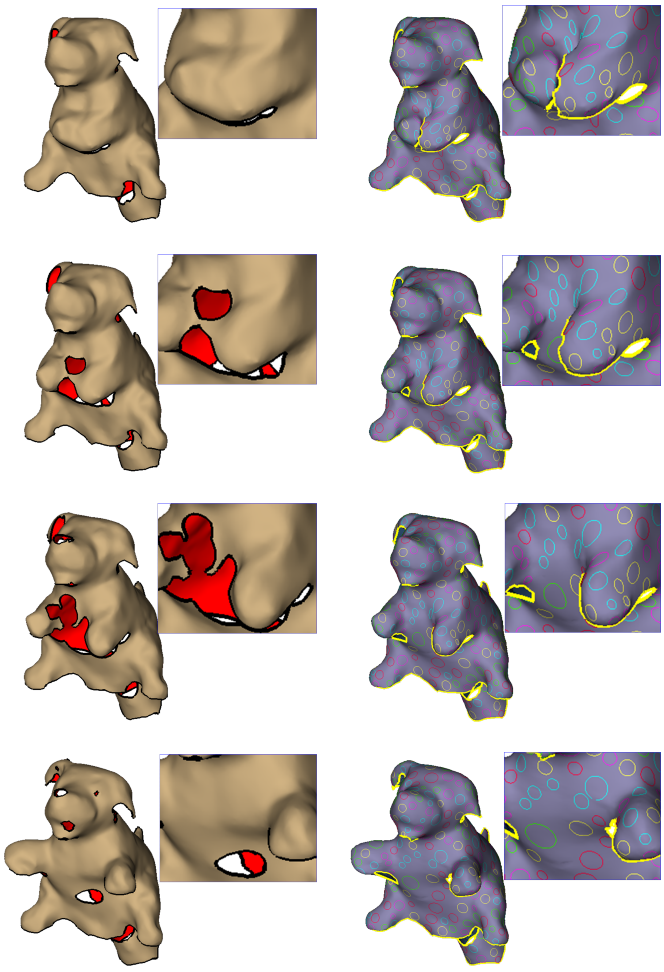


Figure 1.1: Dog hand-puppet: (left) inconsistent per-frame surface reconstruction, (right) globally consistent reconstruction, ellipses visualize the reconstructed motion. Note the correct reconstruction of the arms and chest in the zoomed in region even when the arms are folded.

One of the most difficult challenges when reconstructing gradually changing shapes is the handling of self-contact situations, where the moving surface comes in contact with itself, such as when the fingers of a human hand touch (Figure 1.2). Such self-contacts occur quite frequently in typical motions. Without additional priors, reconstruction of the contact sub-sequence would

glue the contact areas together. However, switching between glued and unglued geometries (Figure 1.2, top) introduces drastic intrinsic change in the reconstructed geometry, violating the gradual-change expectation. Many methods, e.g. [7], introduce erroneous motion or geometry when such drastic changes happen. Others use a geometric template as an additional shape prior and discard or ignore geometries inconsistent with the template. The template is either provided externally [13, 38] or one frame in the sequence is selected to act as a template [14, 15, 27, 35]. Clearly, the use of a template is limiting, as an external template may not be readily available and often no single frame contains complete or even correct, i.e. contact-free, geometric information for the reconstructed shape.

1.2 Overview

In our work we present a novel reconstruction method capable of processing sequences with multiple contact changes without any additional priors. We start by reconstructing individual-frame geometry and then proceed to assemble those individual frames into a consistent space-time frame sequence, simultaneously completing missing information across time and correcting inconsistencies. To the best of our knowledge, ours is the first method to develop such an explicit temporal correction mechanism.

In our method we compute a correspondence between consecutive frames in the sequence. We observe that since the change is gradual, the correspondence, or mapping between correctly reconstructed consecutive frames should exhibit very little stretch, as the intrinsic surface distances change very little with a small change in time. However, for the initial independently reconstructed frames such a mapping may not exist a priori, since due to occlusions and inconsistencies the frames might have different genus and other large intrinsic differences. To overcome these differences we developed a specialized mapping mechanism that uses a local to global approach, assembling a global mapping from a set of local ones, resulting in a map which

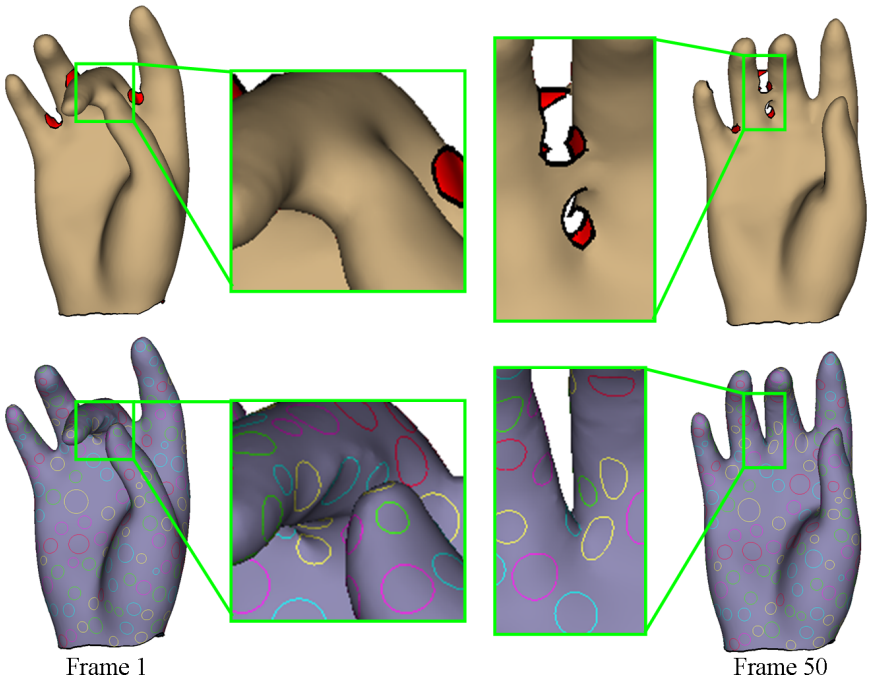


Figure 1.2: Topological correction. Two frames of the hand sequence: (top) per-frame reconstruction; (bottom) our globally consistent reconstruction - the ellipses highlight the global correspondence.

allows for easy detection of inconsistencies and hole completion and which provides a low-stretch feasible correspondence in the consistent regions.

We introduce a robust method for merging the geometric information of consecutive frames together to create a consistent space-time frame pair. We merge together every pair of consecutive frames in a hierarchical fashion, creating a single consistent space-time sequence. Pairs are merged together by using the global mapping to perform geometry completion on one of the frames, filling its holes with information from the other frame, thus creating a union of all the geometric information in both frames. We robustly complete complex holes by introducing a splitting mechanism that divides complex geometry into smaller non-complex parts that we process individually, allowing more of the geometry in the consecutive frames to be merged together.

As demonstrated by our results in (Section 5.3), our completion method is able to robustly complete a wide variety of holes with complex geometry. In (Chapter 6), we demonstrate that our method can handle a variety of contact situations correctly, as it reconstructs the scanned shapes and accumulates both geometric details and motion information across the frame sequences.

1.3 Organization

In (Chapter 2) we review related work in reconstruction and geometry completion methods, and in (Chapter 3) we give an overview of our reconstruction algorithm. In (Chapter 4) we describe our method for constructing the cross-parameterization between individually reconstructed frames. In (Chapter 5) we describe in detail our geometry completion method. In addition, we show results for the completion of individual frame pairs. In (Chapter 6) we show results for the full reconstruction of several datasets. In (Chapter 7) and (Chapter 8) we discuss future work and conclude this thesis.

This project was a collaboration with others, and I will focus primarily on my main contribution to the project. Specifically, I will focus on the geometry completion algorithm in Chapter 5, with other aspects of the system included for completeness.

Chapter 2

Related Work

2.1 Motion Priors

The main challenge of spatio-temporal reconstruction is to effectively use the temporal component, accumulating geometry and motion information over time. This process requires prior knowledge on the scanned object's behaviour over time. Frequently used priors include quasi-isometry [7, 18, 36], and piece-wise rigid motion [10, 27]. Some methods utilize priors that allow for drastic shape and topology changes in the reconstructed models [16, 32]; however, such changes are likely to be undesirable when reconstructing many typical motions, such as the movements of a human or other animal. In our work we expect the scanned shape to change gradually, a prior that is satisfied by the motion of most typically scanned objects. Both quasi-isometry and piece-wise motion priors implicitly assume gradual change.

One of the challenges in spatio-temporal reconstruction is to not only complete locally missing data, but also reconcile inconsistent per-frame inputs. Inconsistent in this context means that straightforward, naive combination of the per-frame scans violates the shape or motion priors. This frequently happens in scans of surfaces coming into contact with themselves. A temporally local reconstruction would glue the parts in contact, resulting in a drastically different intrinsic shape in the contact and non-contact parts of the frame sequence violating the gradual change prior. Inconsistencies can also happen because of errors in local reconstruction.

Methods that focus on reconstructing only the motion of the scanned object,

or frame-to-frame registration [1, 23, 37], can often ignore such inconsistencies. However without corresponding geometry reconstruction and data completion, their results are not as useful. Methods that reconstruct both geometry and motion but ignore the possibility of drastic intrinsic changes [7] can either provide unnatural results or fail completely when inconsistencies are present.

2.2 Shape Priors

The most common way to resolve inconsistencies is to introduce a strong shape prior, by using either an external template, e.g. [13, 20, 38], or one of the frames in the sequence as a template, e.g. [14, 15, 35]. The drawback of the first approach is that an external template may often be unavailable. When using the second solution it becomes quite difficult to complete geometric data missing in the selected frame. Moreover, there may be sequences in which each frame exhibits some local self-contact not present in another part of the motion. In this case, none of the frames can serve as a template without user correction. Several techniques [3, 24] replace the use of a template with a combination of a skeleton and markers. In comparison to the aforementioned approaches, our method manages to correct inconsistencies, reconstructing globally consistent geometry and motion, without relying on any type of template or other structural information. At the same time, our results are comparable to those of state-of-the art methods in cases where no inconsistencies exist.

Wand et al [39] were the first, to our knowledge, to specifically address contact situations, introducing a global space-time reconstruction framework. They use the data to construct a template-like surface representation and then deform it to approximate the inputs. The approximation often smooths out details or introduces non-existent details coming from the template. In contrast, our method preserves the input per-frame geometry nearly everywhere, discarding local geometry only if it is inconsistent with other parts

of the sequence.

2.3 Geometry Completion

Geometry completion is the process of filling in the missing data in a static surface that is caused by an erroneous and incomplete reconstruction of the surface. The most common approach used is to identify the holes and fill them with a smooth surface. Some methods [21, 41] accomplish this by creating an initial triangulation of the hole boundary and improve it until some condition is met. In [21], the hole is triangulated without adding extra vertices in the hole, and then proceed to subdivide it until the sampling density is approximately that of the surrounding surface. Alternatively, Zhao et al. [41] finds an initial poor quality triangulation of the hole with extra vertices, and then the smooth vertex positions are found such that their normal smoothly varies across the hole. Other methods [8, 12] define the hole filling surface with an isosurface and then trianglulate it. Davis et al. [12] construct the isosurface by initializing the inside/outside volume on a voxel grid around the known surface and then diffuses the voxel values through the hole using approximate heat diffusion. In [8] the isosurface for a hole's neighborhood is approximated with Radial Basis Functions and the hole is filled by extrapolating the isosurface into the region of the hole. While some of these methods are able to handle holes of arbitrary size, for most surfaces a smooth surface is not an accurate representation of the missing geometry unless the hole being filled is sufficiently small and does not encompass significant features.

Several methods address the problem of complex missing details by filling the holes with detailed geometry that is synthesized from the mesh itself. Many of these methods [4, 9, 25, 31, 40] have been inspired by texture synthesis and image completion algorithms that fill in missing pixels in an image. These methods fill the holes by identifying geometry patches that could be pasted along a hole boundary and iteratively fill the hole with transformed

patches. Such methods are not able to complete missing features that are not already present in the mesh in a similar form. Other methods complete the geometry by transferring details from a template mesh that contains the missing features. Pauly et al. [26] reconstructs and fills in missing data in a point cloud by using a matching database to automatically select multiple candidate templates, and then proceeds to compose a new mesh by blending together segments of the templates that best match the point cloud. Kraevoy and Sheffer [19] completes the geometry in a mesh by creating a cross parameterization between the mesh and a template using a coarse base mesh, using it to complete the holes in the mesh with the matched geometry from the template. Our geometry completion method is similar in that we use a cross-parameterization to transfer matched geometry, but we do not construct a base mesh to aid in the completion.

Chapter 3

Algorithm Overview

The goal of our system is to reconstruct a *consistent frame sequence* from a given space-time point cloud using precomputed optical flow correspondences as an aid, robustly handling self-contacts and other local inconsistencies. We define a consistent frame sequence as a sequence of meshes with the same connectivity and with per-frame vertex positions that satisfy the gradual change assumptions of small spatial and intrinsic changes in the mesh. The shared connectivity explicitly defines the shape's motion over time.

3.1 Initialization

When performing the reconstruction we aim to maximize the use of per-frame geometric information. To this end our method starts by independently reconstructing per-frame meshes from the input point clouds [6]. We can also use per-frame meshes obtained through other means.

3.2 Hierarchical Assembly

We aim to propagate reliable geometric information available in even a single frame across the entire sequence. To achieve this goal we use a hierarchical sequence assembly procedure that, at each step, combines pairs of consecutive consistent frame sub-sequences into a single consistent sequence (Figure 3.1), while merging the geometric information available in both. At

level zero, we pair individual frames. At level one, we pair sub-sequences of length two, and so forth.

The hierarchical assembly mechanism can propagate geometric information across any number of frames, as the geometry merging propagates this information up through the hierarchy. This is demonstrated in Figure 3.1, where the complete geometry for the left sphere is available only in frame VIII and is successfully propagated across the entire sequence.

3.3 Pair-wise Sequence Combination

When combining the two subsequences we aim to obtain the union of the geometric information available in each, as illustrated in Figures 3.2 and 4.1, and to resolve any inconsistencies between them. As part of the process we also compute a common connectivity for the combined sequence, establishing an explicit one-to-one mapping throughout.

To combine the sub-sequences we first pair the last frame of the first subsequence, referred to as *source mesh*, and the first frame of the second subsequence, referred to as *target mesh*. We then propagate the combined result throughout both sequences, generating a single consistent sequence. The source and target pairing is done as follows:

3.3.1 Cross-Parameterization

We first compute a mapping between the source and target meshes that will capture the motion between them. To this end our mapping minimizes the intrinsic change between the source and its map on the target and is consistent with the optical flow between them (Figure 4.1, row two, and Section 4).

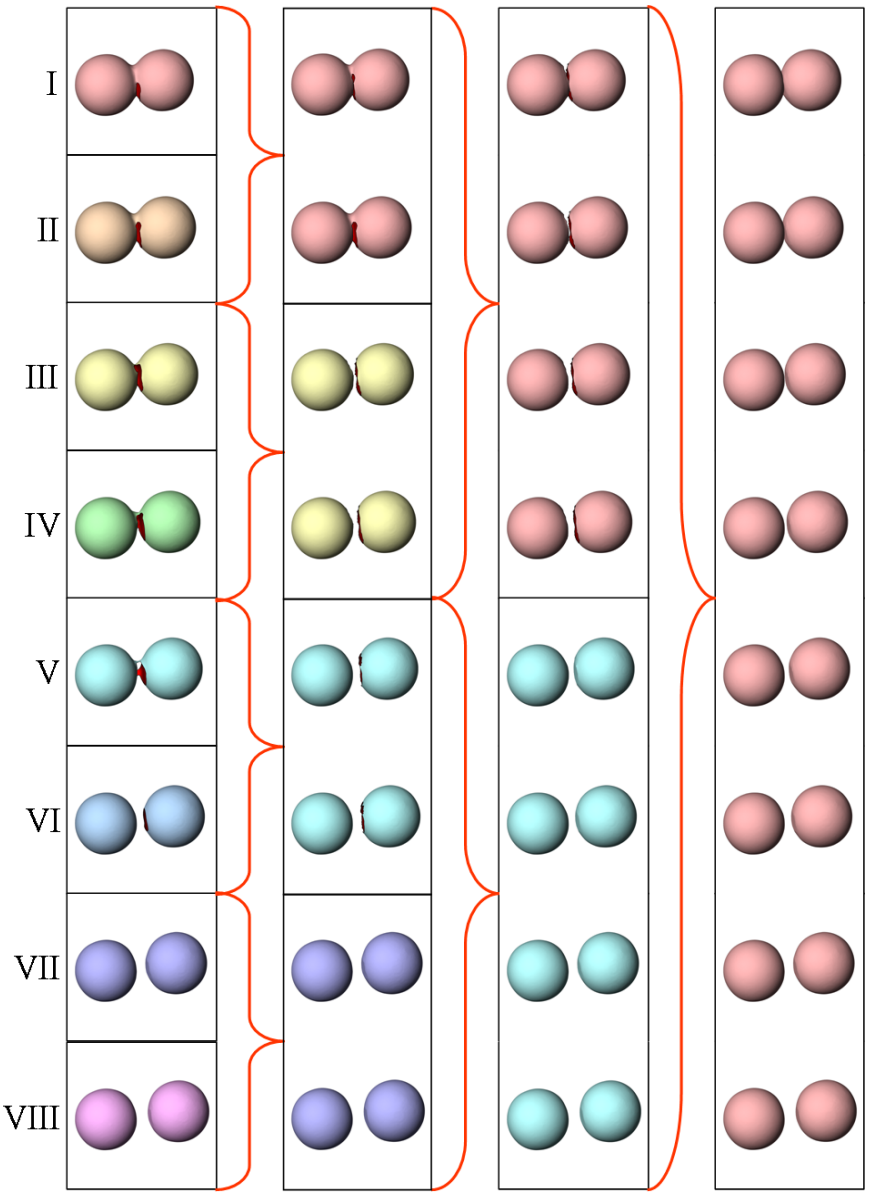


Figure 3.1: Hierarchical assembly of a consistent frame sequence. From left to right are sequences of length 1, 2, 4, and 8 respectively. Note the correction step applied to frames III and IV, V and VI, and again to the first two sequences in the second column.

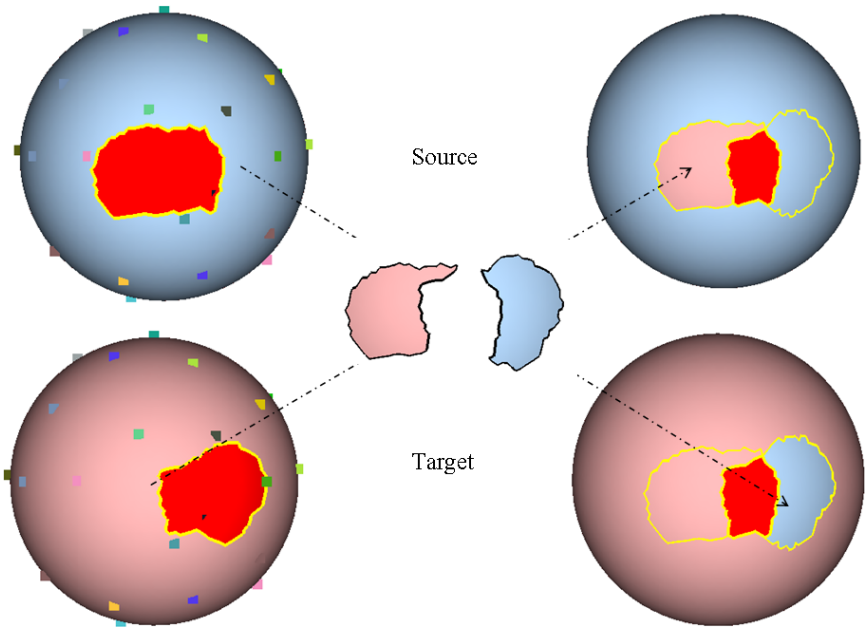


Figure 3.2: Combining the source and target frames (optical flow correspondences highlighted) to generate a consistent frame sequence. Holes highlighted in red.

Analysis and Correction The computed mapping represents the motion between the frames and thus should satisfy the gradual change prior. Regions on the source where the mapping exhibits either stretch or inconsistent (large) spatial motion are indicative of inconsistent source and target reconstructions; this is typically due to self-contact or inaccurate per-frame reconstruction (Figure 4.1, row two). We identify these regions, and correct the meshes accordingly, by deleting erroneous geometry (Section 4.4, Figure 4.1, row three).

3.3.2 Geometry Completion

We use the mapping to combine the geometric information from the source and target frames as illustrated in Figure 3.2, and generate a common connectivity for both frames. We use the mapping to perform geometry com-

pletion on the source, filling in missing geometry with geometry that is in the target. We use this completed frame for the new connectivity of the sequence, and propogate this change throuhgout both sequences, while preserving the local details of the new geometry (Chapter 5).

3.4 Temporal Stretch Optimization

The pairwise maps that we obtain in the previous steps have very low stretch. But when combining large sub-sequences, even a small stretch error, when propagated throughout the sequence, can yield undesirable artifacts. We alleviate this problem by performing a stretch optimization step similar to the one used in [29] on every pair of consecutive frames. Since adjacent frames in the compatible mesh are expected to be nearly isometric, the transformation gradient of any given triangle from one frame to the next should be a near rigid transformation. We therefore compute the per-triangle deformation gradients between consecutive frames and, using polar decomposition, extract the rotational component of the transformation. We then use it as the new transformation gradient from the first frame to the second. The second mesh is then reconstructed as described in [29] yielding a very small stretch between consecutive frames. This optimization procedure is very effective as well as efficient. Since all meshes have the same connectivity we only need to invert a single matrix and backsubstitute different right-hand sides. This process can be repeated, although in all our examples one iteration was sufficient.

Chapter 4

Cross-Parameterization

The goal of this step is to compute a map between the source and target meshes that captures the motion between them. Since we assume the motion or change to be gradual, we aim for a mapping that minimizes such change. We explicitly minimize the mapping stretch (intrinsic change) while using optical flow based initialization (Section 4.1) to bound the spatial change. The challenge we face is that the source and target meshes may not a priori support a bijective low stretch map. First, because the meshes can be incomplete, each mesh can contain regions with no corresponding counterpart on the other (Figure 3.2). Second, and much more challenging, the meshes may be inconsistent, with drastic intrinsic differences preventing a low stretch mapping. To overcome both problems, we compute the global parameterization as an assembly of local low-stretch maps (Section 4.2). The assembly process (Section 4.3) provides a partial mapping in which points with no corresponding map indicate regions on one mesh that correspond to missing geometry on the other, and regions of high stretch indicate local inconsistencies between the source and target.

4.1 Optical Flow Tracking

We aim to obtain a mapping consistent with the expected motion from source to target. However, more than one map can satisfy the gradual change prior. By using video-based capture, we incorporate optical flow [5] as an additional prior on the likely motion. Unlike other methods that rely on a dense, accurate optical flow to track the geometry through time

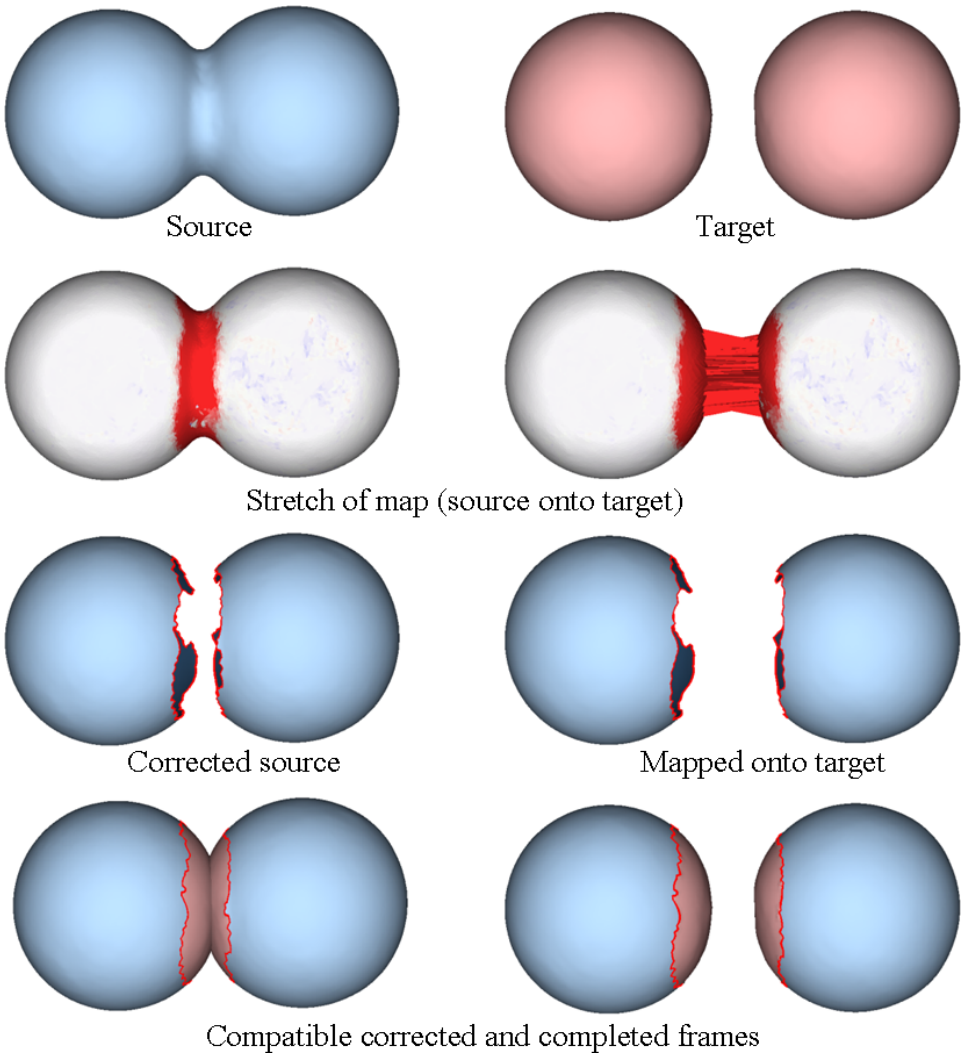


Figure 4.1: Correcting source and target frames. In row two the coloring represents stretch varying from 0.9 (blue) to 1.1 (red) (ideal stretch is one).

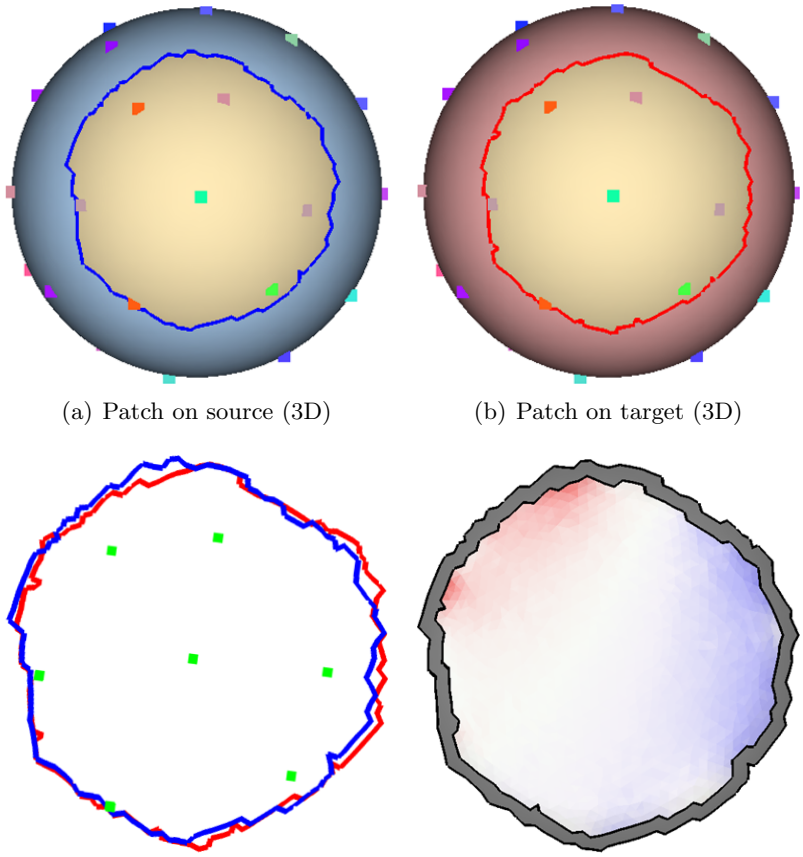
(e.g. [15]), we use the optical flow results only as a secondary prior, providing sparse soft *anchors* to guide the parameterization. Therefore, we can be very conservative in selecting optical flow matches, pruning unreliable correspondences.

We track the optical flow between frames in all camera views, and obtain

a 2D vector field describing the image-space motion of the object from one frame to the next. We then query the 3D motion of a vertex by projecting it into two cameras, advancing it through time using the 2D optical flow in camera space and projecting it back onto the model at the next frame. Next, we perform two levels of pruning to eliminate unreliable correspondences. First, we check the forward and backward flow in 2D for consistency, and only keep the optical flow samples that fall in the same pixel after moving forward and backward in time. We then check that the actual 3D point correspondences are consistent when projected back into the camera space. We further thin the remaining anchors to obtain a fairly uniform distribution across the model. The uniformity makes the subsequent mapping step more robust by using the same parameters across the entire model. The number of retained anchors is on the order of one to three percent of the number of mesh vertices. Even after pruning, the optical flow matches may have slight inaccuracies at the sub-pixel scale, but since we only use them as an initial guide for parameterization, such errors have a very minor impact on our results.

4.2 Local Patch-Based Parameterization

We use a local patch-based parameterization technique as a stepping stone toward computing a global parameterization. As noted by Bradley et al. [7], if two near-isometric patches are mapped to the same domain using stretch-minimizing parameterization, then their maps are likely to be identical; in other words, using the map from one patch to the common domain and then the inverse map from the common domain to the second patch should lead to a near-isometric parameterization between the patches (Figure 4.2). Based on this observation, we grow patches on both models centered around matching anchors and find patch-to-patch maps by parameterizing the patches into the plane and aligning them using the anchors. This process provides us with overlapping low-distortion local parameterizations in most of the consistent areas of the two models. We then use the overlapping local patch maps to



(a) Patch on source (3D)

(b) Patch on target (3D)

(c) Patches flattened and aligned in 2D using anchors

(d) Source mapped to target in 3D (grey area excluded from mapping)

Figure 4.2: Local patch parameterization. In the final map (d) the color visualizes stretch varying from 0.95 (blue) to 1.05 (red).

piece together a global mapping (Section 4.3).

4.2.1 Patch Growth

We start by growing patches simultaneously from all the source anchors and their corresponding points on the target, while preserving patch compactness and disk topology. The compactness criterion is important because it significantly improves the amount of overlap of the parameterized patches

in 2D. The growth process terminates once the patches contain three to seven matching anchors (Figure 4.2(a)). Three anchors are sufficient for subsequent processing, but the parameterization is more robust to noise in the optical flow when the number of anchors per patch is higher. At the same time, the overall mapping stretch is likely to increase with patch size (recall that we do not expect the meshes to be truly isometric). Hence, if the patches grow beyond a certain radius but not enough anchors are found, the growth is aborted and the patch is discarded.

Patches grown only from source anchors may not cover the entire source model. To improve the coverage, we perform a second iteration of patch growth, growing patches from the uncovered vertices. Note that initially regular source mesh vertices do not have a known match on the target mesh. Therefore, when growing a patch from a regular vertex, we first grow the patch on the source mesh until an anchor is encountered, and then grow a patch on the target mesh around the matching anchor until its radius is equal to the radius of the first patch. From here we proceed with regular patch growth. Vertices that remain uncovered at the end of this process typically indicate source regions that lack matching target geometry.

4.2.2 Patch Cross-Parameterization

The two matching patches are first parameterized independently in the plane. We use ABF++ [33] for computing the planar parameterizations, because it provides a reasonable trade-off between minimizing stretch and efficiency. The two parameterizations are then aligned in the 2D plane using an affine transformation that aligns matching anchors in a least-squares sense (Figure 4.2(c)), implicitly providing a parameterization from one patch to the other.

For isometric patches this provides a mapping with minimal stretch. Our patches are not isometric as the triangulations of the input meshes differ and the intrinsic geometry is not identical. However, in most cases the cross-parameterization stretch (Figure 4.2(d)) is concentrated along the

patch boundaries, which are much more sensitive to differences in triangulation. For parameterization purposes we therefore ignore the mapping in the boundary region (greyed out in Figure 4.2(d)).

4.3 Parameterization Assembly[1]

After the local parameterizations are computed, most vertices on the source mesh have multiple possible mappings on the target based on the number of patches they belong to. To obtain a one-to-one map, we first select for each vertex a single local mapping from this set and then apply an optimization procedure to improve the resulting parameterization.

4.3.1 Map Selection

Our goal is to select the patch with the best local mapping for each vertex, such that the resulting global map exhibits minimal stretch. This goal translates into a combinatorial optimization problem, which unfortunately is very difficult to solve. Instead we opt for the following efficient heuristic, which combined with the subsequent optimization yields the desired result. We observe that in the local mappings stretch is concentrated mostly near patch boundaries (Figure 4.2(d)). Hence, for each vertex we select the map corresponding to the patch in which the vertex is closest to the center. This heuristic results in global parameterizations that exhibit very low stretch for triangles in the interior of most local patches, and higher stretch in the patch boundary regions. Triangles whose vertices are mapped using different patches typically exhibit the worst stretch, and in extreme cases can even be flipped (Figure 4.3, left).

[1] The parameterization assembly step is not a contribution of this thesis and is included for completeness.

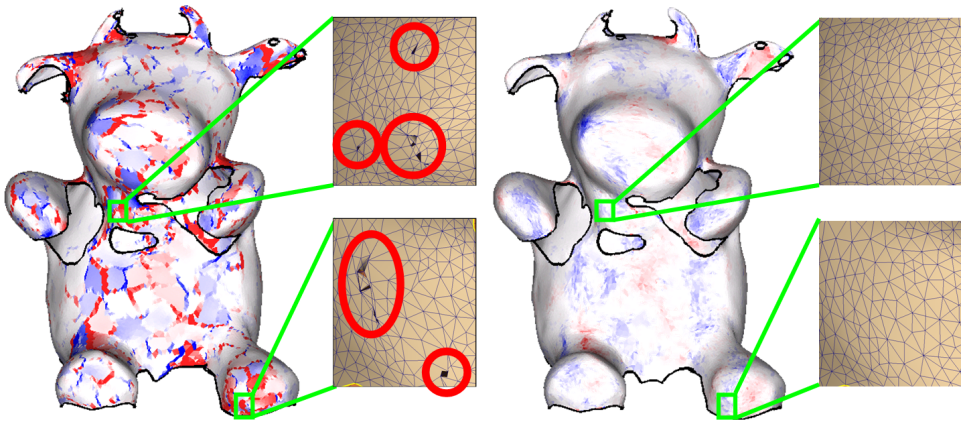


Figure 4.3: Mapping optimization: (left) Initial global parameterization with stretch and flipped triangles most prominent when mapping has switched patches; (right) stretch after optimization is applied (six global iterations).

4.3.2 Optimization

To improve the parameterization we need an optimization procedure that, given the initial map, simultaneously reduces the stretch and fixes flipped triangles. In the presence of flips, direct stretch optimization (e.g. [30]) often gets stuck in local minima, because sliding a vertex along the mesh in the correct target direction increases the local stretch before decreasing it again. In addition, direct minimization of stretch as described by Schreiner et al. is very time consuming, requiring runtimes of an hour or more for meshes of interesting size.

We observe that because the change in the intrinsic shape from one frame to the next is small, a local shape-preserving functional is consistent with stretch minimization in our setting. Thus, rather than directly minimizing stretch, we choose to optimize a functional preserving the Laplacian coordinates of the source vertices [22] when mapped to the target. Evaluating the Laplacian is significantly faster than evaluating stretch, but more importantly, the Laplacian functional clearly detects flipped triangles, and optimizing it eliminates them. Since the distortion is concentrated in small

regions, we can effectively utilize a local relaxation technique, iteratively moving one source mesh vertex at a time over the target mesh to reduce

$$\min_{\tilde{v}} \parallel L\tilde{v} - Lv \parallel_2^2 . \tag{4.1}$$

Here L is the Laplacian operator, v are the original positions in the source mesh, and $\tilde{v}$ the mapped positions on the target mesh. We constrain the solution space to the target mesh, thus preserving the initial target geometry. To search for a local minimum, we use a random walk approach. To speed optimization, we process the vertices in a decreasing order of their error (Equation 4.1). Usually only a few (five or six) global iterations of Laplacian relaxation are sufficient to reduce stretch to acceptable levels across much of the mesh(Figure 4.3, right). Once the flipped triangles are corrected, it is possible to switch to direct stretch optimization, but in our experiments this did not improve results significantly.

Areas that have local high stretch even in the optimized map are indicative of geometric inconsistencies between the processed frames (Figure 4.1, row two). Vertices on either model that remain unmapped at the end of this step indicate geometry missing in the other model and are incorporated into that model by the completion step (Section 4.4).

4.4 Analysis and Correction[2]

The cross-parameterization step computes a map between the source and target meshes which aims to approximate the motion between the frames. Thus under our gradual change assumption, mapped triangles should exhibit low stretch and the vertex motion prescribed by the map should be sufficiently small. A violation of either of these conditions indicates a problem in the local reconstruction of either the source or target meshes and

[2]The analysis and correction step is not a contribution of this thesis and is included for completeness.

their corresponding sub-sequences (Figure 4.1, row two). Based on this observation it becomes trivial to detect such erroneous reconstructions in our setting by simply analyzing triangle stretch and vertex motion prescribed by the cross-parameterization.

Manually setting a threshold on acceptable magnitude of vertex motion is quite problematic as it strongly relates to the local motion speed. Instead, we found that this test can be robustly replaced by checking for motion similarity between neighbouring vertices and between vertices and neighbouring anchors. Vertices belonging to triangles that exhibit high-stretch, and vertices whose motion is much larger compared to neighbouring vertices or anchors are flagged as potentially incorrect and clustered into regions. We do not know a priori whether the inconsistencies are caused by problems with the geometry in the source or target meshes, but we do know that at least one of the corresponding source and target regions has incorrect geometry. Therefore, we need to identify which, if any, of the source or target regions is consistent with our gradual change assumptions and which is not. For each of source and target, we first delete the region in one mesh and complete the hole with the corresponding region from the other, using the completion mechanism described in the next section. We then test to see whether a mapping using this new geometry obeys our gradual change assumptions.

There are three possible outcomes. The first, and most frequent one is that only the geometry of one of the meshes satisfies the gradual change criteria, in which case this geometry is selected for both frames. The second possible outcome is that both completions violate the gradual change assumption, indicating that neither is acceptable. In this case the inconsistent regions are deleted from both frame sequences. In the third case both completions lead to consistent results. In this case, for simplicity, we pick the region from the source.

Chapter 5

Geometry Completion

The final step of pair-wise sequence combination is to merge the two subsequences into a single consistent frame sequence, consisting of a common mesh connectivity with positions for each vertex in each frame. The resulting output sequence contains all of the surface geometry present in both input sequences. Consequently, when the last frame pair is processed at the top level of the hierarchy, the final output sequence will have accumulated all of the geometry in the entire capture sequence.

In this section we present a robust algorithm for merging the sequences using the global parametrization computed to perform *geometry completion*. Geometry completion is the process of filling in (or *completing*) the missing geometry of a mesh using the geometry information from another mesh. Our algorithm performs geometry completion on the source mesh using the target mesh to fill in the missing geometry, creating a new mesh that has all of the geometry of both the source and target. We use the connectivity of this mesh for the connectivity of the complete consistent frame sequence that represents the entire range of the source and target, and we propagate the connectivity to all the frames using a deformation technique to find the vertex positions in each frame.

Our algorithm receives as input the source and target frame sequences along with the mapping from the source mesh to the target mesh (5.1(top)). We use the mapping to guide our geometry completion algorithm, which completes the source using the target information and filling in some of the holes (5.1(middle)). This outputs a completed source that represents the union of

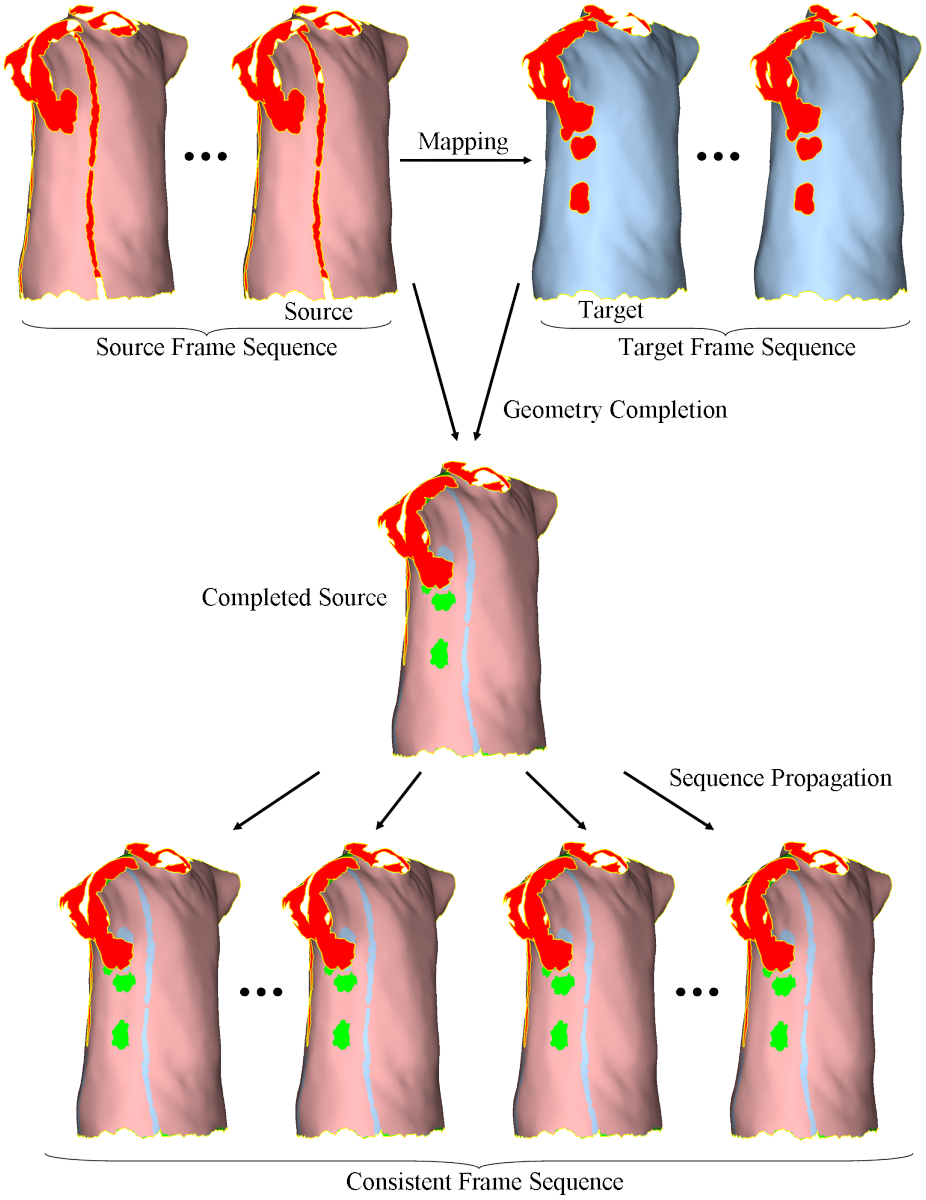


Figure 5.1: Completion overview: (top) Input source and target sequences with a mapping from the source to target, (center) completed source representing the full geometry, (bottom) propagating the source geometry into all frames, creating the output consistent frame sequence.

the source and target geometry. Finally, we propagate the completed source into each frame of the output sequence (5.1(bottom)). Given the mapping, only the pink source vertex positions are known in all frames, while the green and the blue vertex positions are only available in the first half or the second half of the sequence, respectively. We determine the positions of the unknown regions by deforming them from a known frame into the unknown frames while preserving the local details. The resulting consistent frame sequence is used as input for the next level of hierarchical assembly, given it is not the last level of the hierarchy.

This chapter is organized into two sections. In Section 5.1 we describe in detail the geometry completion algorithm that we use to generate the completed source. In Section 5.2 we describe our deformation scheme for propagating the completed source to the entire sequence.

5.1 Source Completion

We use the source mesh connectivity as a basis for the connectivity of the merged consistent frame sequence. We generate this merged connectivity, that represents the full geometry in the sequence, by performing geometry completion on the source mesh using the target, utilizing an algorithm similar to Kraevoy et al [19]. This process fills in the missing geometry in the source, resulting in a new source connectivity that represents the full geometry available in both the source and target.

In the first step of source completion we first identify the target geometry that we can use to complete the source. To do this we segment the target into two categories, *missing* or *not missing* in the source, and use each missing set to construct the completion geometry that fills the source holes. We segment the target by using the mapping to project the source boundaries onto the target surface (Figure 5.2(left bottom)), which intersect and surrounds the missing geometry (Figure 5.2(middle bottom)). This segmentation generally produces one large set of geometry that is not missing in the source, and

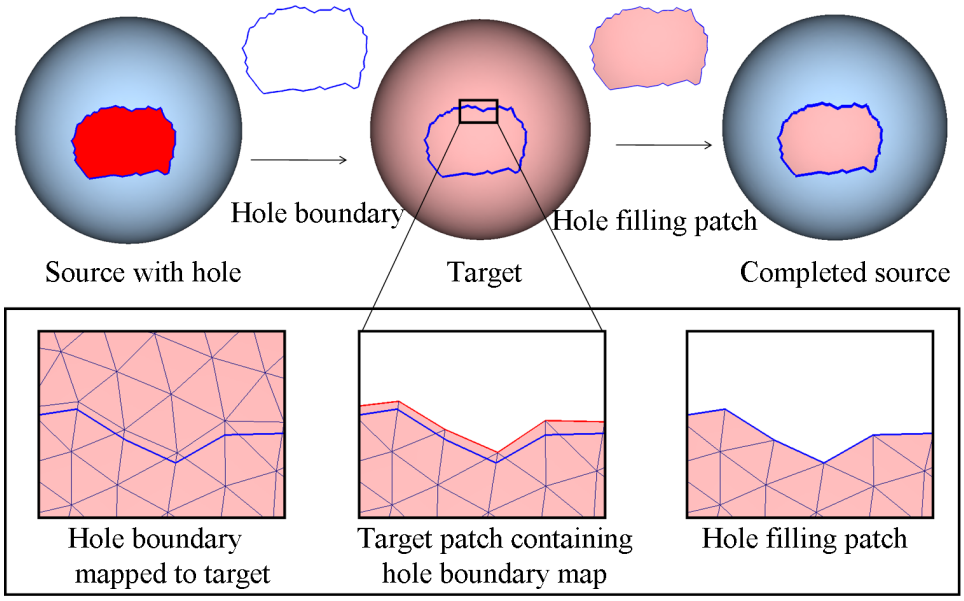


Figure 5.2: Completion: (left) Mapping the boundary, (center) extracting missing geometry, and (right) completing the geometry. Missing geometry highlighted in red.

multiple sets of missing geometry (Figure 5.2(middle top)). We extract each set of missing geometry and remesh it to create a new connectivity that conforms to the source boundaries (Figure 5.2(right bottom)), so that it seamlessly completes the source. We then generate the new vertex positions for the missing geometry in the source (Figure 5.2(right top)), as described in Section 5.2.

In Section 5.1.1 we describe our method for segmenting the target into missing and not missing sets, and in Section 5.1.2 we describe how we create the new connectivity for the missing geometry in the source.

5.1.1 Segmentation

To complete the source with target geometry, we first identify the *completion geometry*, the subset of target geometry that is missing from the source that

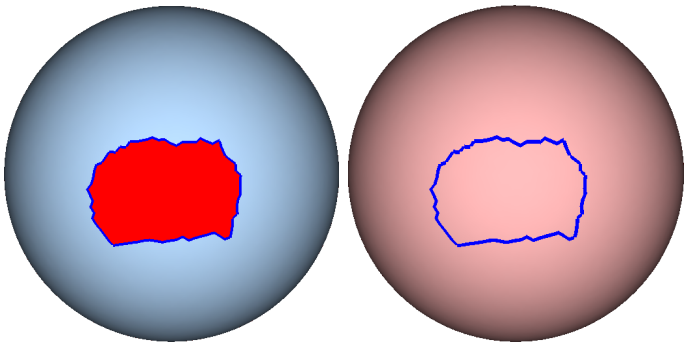
we will use to complete the source. We do this by segmenting the target triangles into two sets, one of which contains the completion geometry.

Our algorithm for segmenting the target proceeds as follows: First we map the source boundaries onto the target and determine if there exists adjacent target geometry that we can attach to the boundary. If such geometry does exist, we mark that source boundary as *completable*. The mapping of the completable source boundaries separates the target geometry, with completion geometry on one side of the boundary and non-completion geometry on the other (Figure 5.3). We approximate the mapping of each completable source boundary using the sequence of triangles intersected by the mapping. Finally, we complete the segmentation by considering each intersected triangle to be in the completion geometry set and flood fill inward, adding adjacent triangles to the completion geometry set.

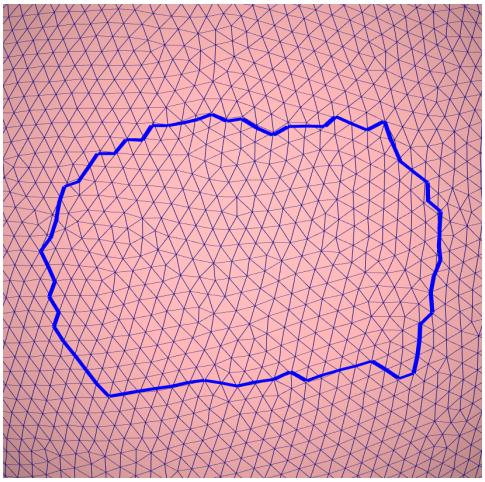
Source Boundary Projection We observe that the missing geometry may only attach to the source mesh at the boundaries, and that the mapping of those boundaries will be adjacent to the completion geometry. We identify the *completable boundary sequences*, the sequences of boundary vertices that have associated target geometry, by analyzing the mapped location in the target. After identifying each completable boundary sequence, we use the mapped location to segment the target.

We differentiate between three types of vertices based on the mapping: *unmapped*, which do not have a mapped location in the source, *mapped*, which do have a mapped location, and *boundary mapped*, which map close to a target boundary. Further details on these different types of mappings and how we use them to determine the completable boundary sequences are as follows:

- **Unmapped Vertices:** The unmapped vertices correspond to geometry that is in the source but not the target. Therefore there does not exist nearby target geometry that can be attached to unmapped boundary vertices.



(a) Source (b) Target



(c) Mapped Boundary

Figure 5.3: The mapping of the source boundary onto the target surrounds the completion geometry.

- **Mapped Vertices:** The *mapped vertices* are the source vertices with a mapped location. The location of the mapped vertex position is determined by the target triangle T that it maps into and the Barycentric coordinates $(\lambda_0, \lambda_1, \lambda_2)$ of the position within T. The mapped position is calculated as $\sum_{j=0}^{2} \lambda_j P_{T_j}$, where P_{T_j} is the position of the j-th vertex in the triangle T.
- **Boundary Mapped Vertices:** Some boundary vertices will map

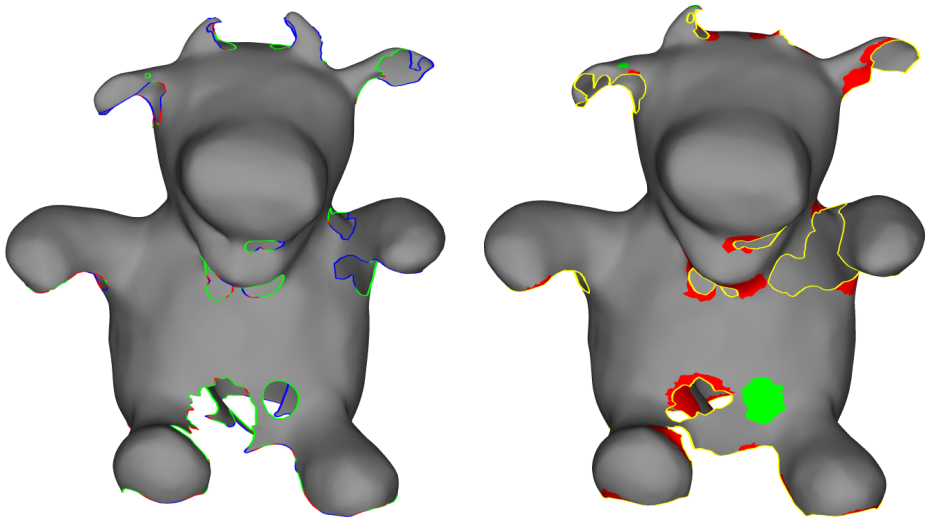
close to a target boundary. If we treated these the same as other mapped vertices they would only be completed with a small amount of target geometry, for example the red vertices in (Figure 5.4(a)), so for the purposes of our algorithm we consider them to be unmapped. We consider a vertex to be close to the boundary if it maps to a triangle with at least one boundary vertex, and the on-boundary neighbors dominate the weighting of the position such that the sum of their weights is greater than 0.9.

- **Completable Boundary Sequences:** We group adjacent mapped boundary vertices together to form the completable boundary sequences. Some completable boundaries will encompass an entire boundary, while others will only be partial boundaries. In Figure 5.4, the boundary in (a) that is associated with the green geometry in (b) is a full boundary, while the boundaries associated with red geometry are partial boundaries. Full boundaries will tend to be entirely filled with target geometry, while partial boundaries are filled with geometry that is connected to the target boundaries.

We discard short completable boundary sequences with less than four vertices, and use the remaining sequences to segment the target and identify the missing geometry. We segment the target by using the path of triangles intersected the mapped sequence. These paths of triangles segment the target by acting as the boundary between the missing and non missing geometry.

Intersecting Triangle Paths We segment the target using the sequence of points found by mapping the completable boundary sequences, and connect them together to form a seamless path on the target surface that separates the missing geometry from the non missing. Rather than use the geodesic path between points to represent the mapping of the boundary, we approximate it using a *triangle path*, an edge-adjacent sequence of triangles (Figure 5.5(a)).

We construct a triangle path for each completable boundary sequence by

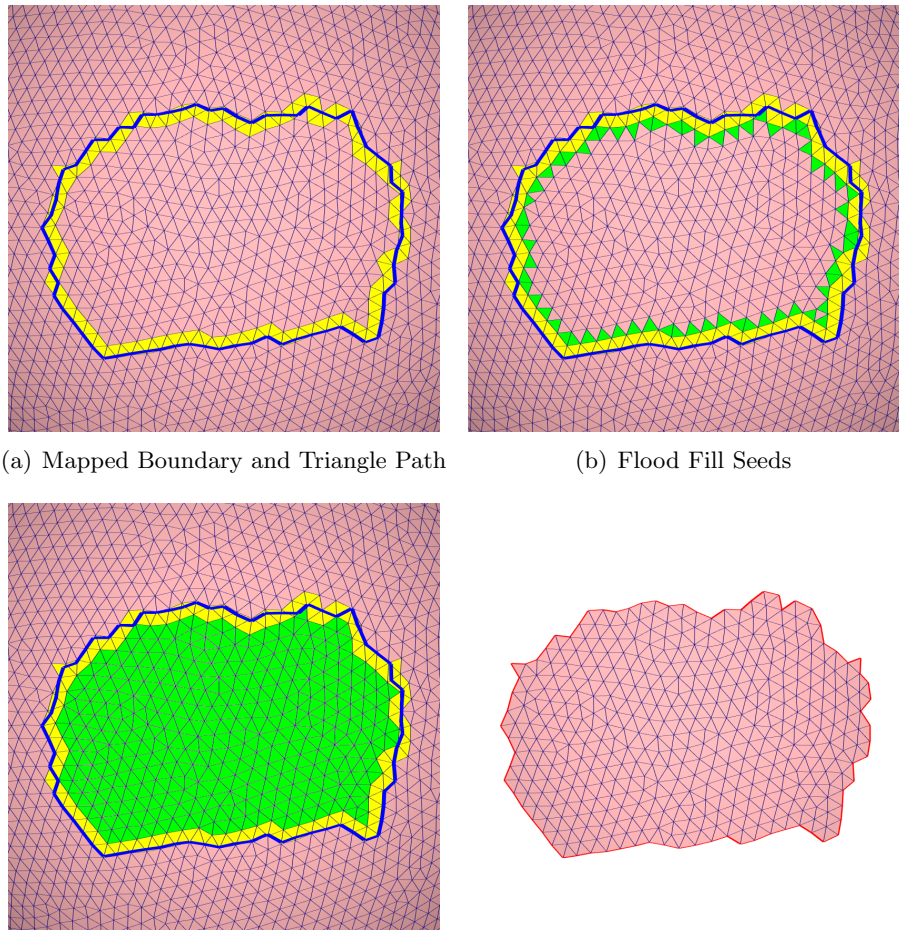


(a) Source (b) Target

Figure 5.4: Identifying completion geometry: (a) Boundary vertices: Mapped boundary vertices are green, unmapped boundary vertices are blue, and vertices mapping close to the boundary are are red, (b) Segmented target: Colored regions are completion geometry, with green regions completely bounded by a mapped path, and red regions are bounded by mapped path and the target boundary.

connecting together the mapped triangles for each vertex in the boundary. The triangles are not guaranteed to connect together seamlessly along the edges, so we connect them together using the shortest path between sequential triangles. In addition, if a completable boundary sequences does not encompass an entire boundary, we complete the path by connecting the terminal triangles to a boundary by finding the shortest path to a boundary. We perform these queries on the *dual graph* of the target. Details on the dual graph and how we perform the shortest path queries are as follows:

- **Dual Graph:** The dual graph of a mesh represents the adjacency of the triangles. Each triangles has a vertex in the dual graph, and there is an edge between two vertices in the dual graph if and only if the triangles share an edge. For our purpose the edge weight between two



(a) Mapped Boundary and Triangle Path

(b) Flood Fill Seeds

(c) Flood Filled

(d) Completion Geometry

Figure 5.5: Extracting completion geometry on the target. (a) Triangle path and mapped boundary (b) flood fill seed triangles to the left of the path (c) completion geometry from flood filling from the seeds (d) extracted completion geometry copied and cut from the target.

triangles is the distance between the Barycenters of those triangles.

- **Shortest Path Between Triangles:** We connect together the mapped triangles by finding the shortest path between them in the dual graph. We use Djikstra's shortest path algorithm to find it.

- **Shortest Path to Boundary:** We find this by modifying Djikstra's shortest path algorithm to terminate when the first boundary triangle is popped off the priority queue.

The set of all triangles paths seamlessly separates the completion geometry and non-completion geometry, so we flood fill to identify the completion geometry.

Flood filling We find the completion geometry by flood filling the geometry surrounded by the triangle paths. The geometry surrounded by a path is determined by the orientation of the triangle path. A completion boundary on the target traverses in counterclockwise (CCW) order around the hole in the target, therefore the triangle path will traverse in CCW order around the completion geometry. We discover the completion geometry by flood filling starting from a set of of *flood fill seeds*, which are the triangles that are topologically to the left of the path (Figure 5.5(b)) in the dual graph. A triangle is topologically to the left of the path if and only if it is oriented to the left of the dual of the path in the target's dual graph. Flood filling from these seeds gives us the completion geometry (Figure 5.5(c)).

We then extract the flood filled geometry and create a copy of it (Figure 5.5(d)) to complete the holes in the source.

5.1.2 Remeshing

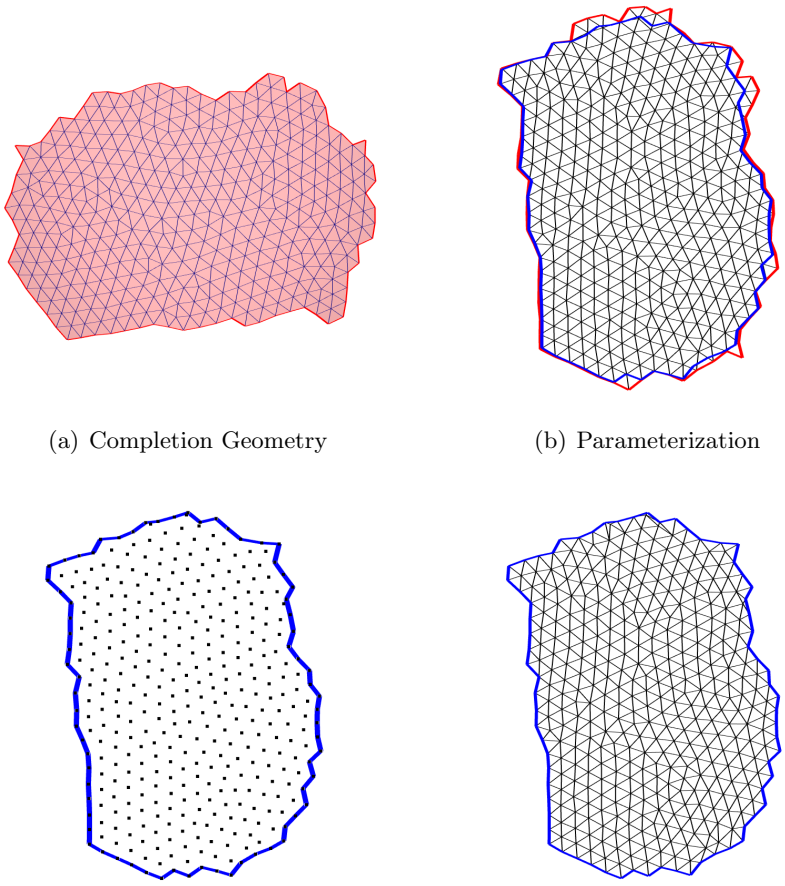
In this section we will describe how we *remesh*, or create a new connectivity, of the completion geometry we found in the segmentation, such that the new connectivity conforms to the source connectivity. The completion geometry we previously found contains a superset of the vertices we will, in effect, be copying into the source mesh. The connectivity of these target vertices are not compatible with the source connectivity, so we remesh the vertices to conform to the source connectivity. We define each set of remeshed target vertices as a *completion patch*. We create each completion patch by creating

the connectivity robustly in 2D, allowing us to create a general connectivity for the new vertices in the source, such that the connectivity can be used for the entire consistent frame sequence.

We create the completion patch by selecting a subset of the vertices from the extracted completion geometry, and generating a new connectivity for these vertices while conforming to the source boundary connectivity. We parameterize the completion geometry (Figure 5.6(a)) giving us a representation of it in 2D (Figure 5.6(b)), and then the completable boundary sequences are mapped into the parameterized completion geometry (Figure 5.6(b)). We use the completable boundary sequences to create a 2D polygon, and we discard any vertices outside of the polygon (Figure 5.6(c)). We then triangulate the polygon and the remaining target vertices to create the new connectivity that conforms to the source connectivity (Figure 5.6(d)).

Parametrization We generate the connectivity for the completion patch by first parametrizing the completion geometry, giving us a 2D representation of the geometry. The quality of the connectivity in 3D when it completes the source depends on the quality of the parametrization. If the parametrization from 3D to 2D induces a large distortion, then a quality triangulation in 2D may not be a quality 3D surface triangulation. For this reason we use a low-distortion parametrization method, ABF++ [33], using the implementation provided in Graphite [17]. We then use this parametrization to find the 2D boundary and vertices of the completion patch.

2D Boundary and Internal Points Using the 2D parametrization of the completion geometry, we find the 2D representation of the source boundary that we will conform to by constructing the *boundary polygon*, the polygon representing the boundary of the completion patch in 2D. We construct the boundary polygon from the mapping of the completable boundary sequences that are adjacent to the completion geometry. We choose the *internal points*, the 2D position of vertices that will be in the completion patch, by choosing only the target vertices that lie inside this polygon, and discard the rest.



(a) Completion Geometry

(b) Parameterization

(c) Boundary Polygon and Internal Points

(d) Remeshed Geometry

Figure 5.6: Remeshing completion connectivity: (a) completion geometry from Figure 5.5 (b) Parametrized version with mapping of boundary in 2D (blue) (c) Boundary polygon and selected internal vertices (d) Connectivity of completion by triangulating (c).

- **Boundary Polygon:** We construct the polygon from the 2D mapped positions of the completable boundaries that are adjacent to the completion geometry. In the trivial case, when the completable boundary encompasses an entire source boundary, the polygon can be constructed entirely from the completable boundary. Otherwise, the map-

ping of the completable boundary is a set of connected line segments that we must connect together to form a polygon. We connect them together by adding the line segments of the completion geometry boundary that are boundaries in the target. We do this by first associating the closest target boundary vertex to each endpoint of completable boundaries. We traverse the boundary from one endpoint until we encounter another endpoint, connecting together the endpoints with the 2D line segments of the boundary encountered. This gives us a closed polygon representing the conforming boundary of the final completion patch.

- **Internal Points:** Using the boundary polygon, we discard the vertices from the completion geometry that are not going to be included in the final completion patch. We discard any vertices that are outside the polygon by using an in-polygon test on all of the 2D positions of the completion geometry vertices. In addition, we discard any vertices that are close to the polygon, if its distance from a polygon edge is less than 0.1 times the average polygon edge length. By removing these vertices, we prevent small and sliver triangles from appearing in the final connectivity.

We use the boundary polygon and the internal points to construct the connectivity of the patch.

Connectivity Assuming that we are given a low-distortion parametrization, a high quality triangulation of the boundary polygon and the interior points will result in a high quality mesh in 3D. For this reason, we use a constrained Delaunay triangulation of the points, using the Triangle [34] software. A constrained Delaunay triangulation (CDT) is a Delaunay triangulation of a set of points, with some vertices constrained to be connected by an edge. We constrain the boundary polygon to be preserved by constraining each individual polygon edge. The boundary of a Delaunay triangulation is the convex hull of the points therefore we remove those triangles that are

exterior to the polygon. The connectivity of this triangulation conforms to the source hole and will give us a high quality triangulation in 3D.

Handling Degenerate Cases In some cases of complicated completion geometry, the ABF++ algorithm will fail to create a valid parametrization. If the completion geometry is not a topological disc and it has holes, then it cannot be parametrized with ABF++. Sometimes ABF++ will produce a parametrization that overlaps itself and the method described above will fail. We handle degenerate completion geometry by first cutting edges until there are no holes and cutting it into multiple disconnected pieces until there are no overlaps. We then remesh each piece separately as described above, and merge the remeshed pieces together to create the full completion geometry patch.

- **Hole Cutting:** We eliminate each hole by cutting the completion geometry using the Djisktra shortest path algorithm to find the shortest path between the hole and the boundary. We find the shortest path by adding a fake vertex to the graph that has an edge with each vertex on the boundary of the hole, and using an edge weight of 0 for these edges. We start the search from the fake vertex, and terminate the search on the first boundary vertex that is pulled off of the priority queue. We then cut the completion geometry along the edges in this path, eliminating the internal hole and connecting it to the exterior boundary. This will frequently introduce an overlap in the parametrization, requiring further cutting.

- **Overlap Cutting:** After cutting all holes, we eliminate overlaps in the resulting parametrization. To do this, we cut the geometry using Variational Shape Approximation (VSA) [11], utilizing the implementation provided in Graphite [17], until each individual piece does not overlap itself in the parametrization.

We then generate the connectivity as described earlier, while constraining the cut edges in the CDT. We then recombine the connectivity of all the remeshed pieces together at the cut edges, thus creating a connectivity that conforms to the original boundary.

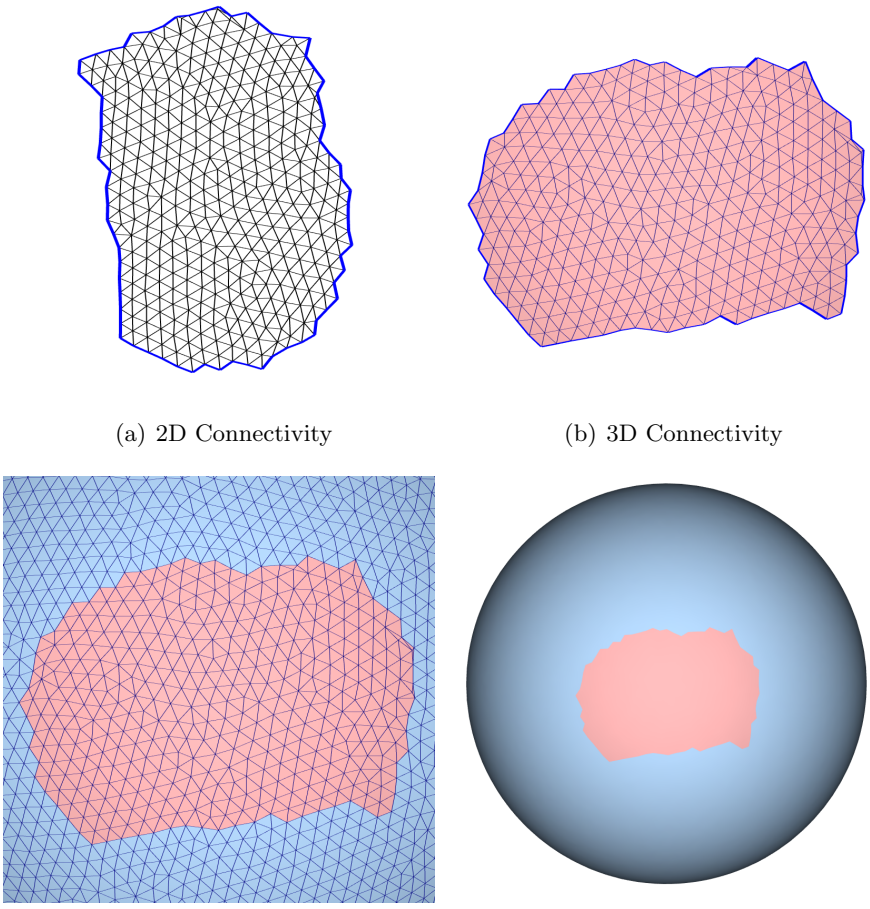
After we have generated all of the completion patches, we connect their boundaries to their associated source boundaries (5.7(c)), thus creating the completed source that contains connectivity for all of the geometry available in both the source and target (5.7(d)). We then generate vertex positions for the completed source in all the frames.

5.2 Sequence Propagation

After completing the source, we create the single consistent frame sequence by taking the completed source connectivity and generating the vertex positions for each frame. The completed source contains both source and target vertices, and therefore we already have available the vertex positions for source vertices in the source frames, and positions for the target vertices in the target frames. In addition, the mapping gives us the mapped source vertex positions in each of the target frames. This leaves two different types of unknown vertex positions: the unmapped source vertices in the target frames, and the target vertices that completed the source in the source frames. We find these unknown per-frame vertex positions by deforming the corresponding geometry from a known frame, called the *origin frame*, into the unknown frame, called the *destination frame*.

5.2.1 Deformation

We use a Laplacian deformation technique similar to Lipman et al [22] to deform the origin frame into the destination frame. We call each set of connected unknown vertices in the completed source connectivity a *deformation set*, and we deform each deformation set individually from its origin frame



(a) 2D Connectivity

(b) 3D Connectivity

(c) Glued into Source

(d) Completed Source

Figure 5.7: Completing the source with the new connectivity.

into each destination frame. For target destination frames we use the source as the origin frame, and for source destination frames we use the target as the origin frame. We also include in the deformation set the *border vertices*, which is the set of vertices that are edge-adjacent to the deformation set. The positions of the border vertices are known in both the origin and destination frames.

We deform the vertices by finding vertex positions in the destination frame

such that the Laplace vectors in the origin are preserved. In addition, the deformation set may undergo rotation between the origin and destination frame, so we determine this rotation and find vertex positions that preserve the rotated Laplace vectors.

We perform our deformation in three steps: In the first step we determine the rotation the border undergoes from the origin frame to the destination frame. We then propagate the rotations from the border to the vertices in the deformation set such that the rotation varies smoothly between vertices. Finally, we find the vertex positions for the destination frame such that the Laplace vectors closely match the rotated Laplace vectors from the origin frame.

Border Rotations We find the rotation a border vertex undergoes from the origin to the destination by first constructing local coordinate frames for the border vertex in the origin and destination frames. We use the rotation the local coordinate frame undergoes from the origin to the local coordinate frame in the destination. The local coordinate frame for a border vertex is created by constructing a orthonormal basis from the vertex normal $\mathbf{n}_i$ and the vector of an outgoing edge $\mathbf{e}_{ij}$:

$$\begin{aligned}\hat{\mathbf{z}}_i &= \mathbf{n}_i \\ \hat{\mathbf{y}}_i &= \frac{\hat{\mathbf{z}}_i \times \mathbf{e}_{ij}}{\|\hat{\mathbf{z}}_i \times \mathbf{e}_{ij}\|} \\ \hat{\mathbf{y}}_i &= \hat{\mathbf{y}}_i \times \hat{\mathbf{z}}_i\end{aligned}$$

We construct the 3×3 matrix $\mathbf{F}_i$ that represents the local frame basis:

$$\mathbf{F}_i = \begin{pmatrix} \hat{\mathbf{x}}_i^{\mathrm{T}} \\ \hat{\mathbf{y}}_i^{\mathrm{T}} \\ \hat{\mathbf{z}}_i^{\mathrm{T}} \end{pmatrix}.$$

Finally, we construct the rotation matrix for the transformation that $\mathbf{F}_i$ undergoes from the origin to the destination by the change of basis from $\mathbf{F}_i$ to $\mathbf{F}'_i$, where $\mathbf{F}'_i$ is the local frame constructed in the destination. This rotation is given by $\mathbf{R}_i = \mathbf{F}_i'^{\mathrm{T}}\mathbf{F}_i$. We then smoothly propagate these rotation matrices to the vertices in the deformation set.

Rotation Propagation We desire a smooth propagation of border rotations to the deformation set. We do this by minimizing the squared Frobenius norm of the difference between the logarithm of neighboring rotation matrices:

$$\min_{\forall i\, \mathbf{R}_i} \sum_{(i,j)\in\mathbf{E}} \|\log \mathbf{R}_i - \log \mathbf{R}_j\|_F^2 \quad \text{subj to} \quad \mathbf{R}_k = \overline{\mathbf{R}}_k,\ \forall k \in C,$$

where C is the set of border vertices, and $\overline{\mathbf{R}}_k$ is the rotation found for border vertex k. By performing the minimization with the logarithm of the rotation matrices we keep the solution within the space of rotation matrices, while allowing the minimization to be found by solving a linear system for each component of the matrix. We find the logarithm of the minimizing rotations, and let $\mathbf{R}_i = e^{\log \mathbf{R}_i}$. To calculate the exponential and logarithm of the matrices we use the methods described by Alexa [2]. We then use these matrices to rotate the Laplace vectors of the deformation set vertices in the origin frame when deforming them.

Finally, we use the rotated Laplace vectors to find the vertex positions.

Positions We find the positions by minimizing the difference between the destination Laplacian coordinates and the rotated origin Laplace coordinates. Specifically, we minimize

$$\min_{\forall i\, v_i} \sum_i \left\| v_i - \frac{1}{d_i} \sum_{e=(i,j)} v_j - \mathbf{R}_i \mathbf{l}_i \right\|^2 \quad \text{subj to} \quad v_k = \overline{v}_k,\ \forall k \in C,$$

where

$$\mathbf{l}_i = v'_i - \frac{1}{d_i} \sum_{e=(i,j)} v'_j$$

is the standard Laplace vector for the origin, v_i are the destination positions, d_i is the degree of vertex i, v'_i is the position in the origin for vertex i, and $\overline{v}_k$ is the known position of the border vertex k in the destination frame. By finding the minimizing vertex positions, we find the new positions in the destination that preserve the local details in the origin.

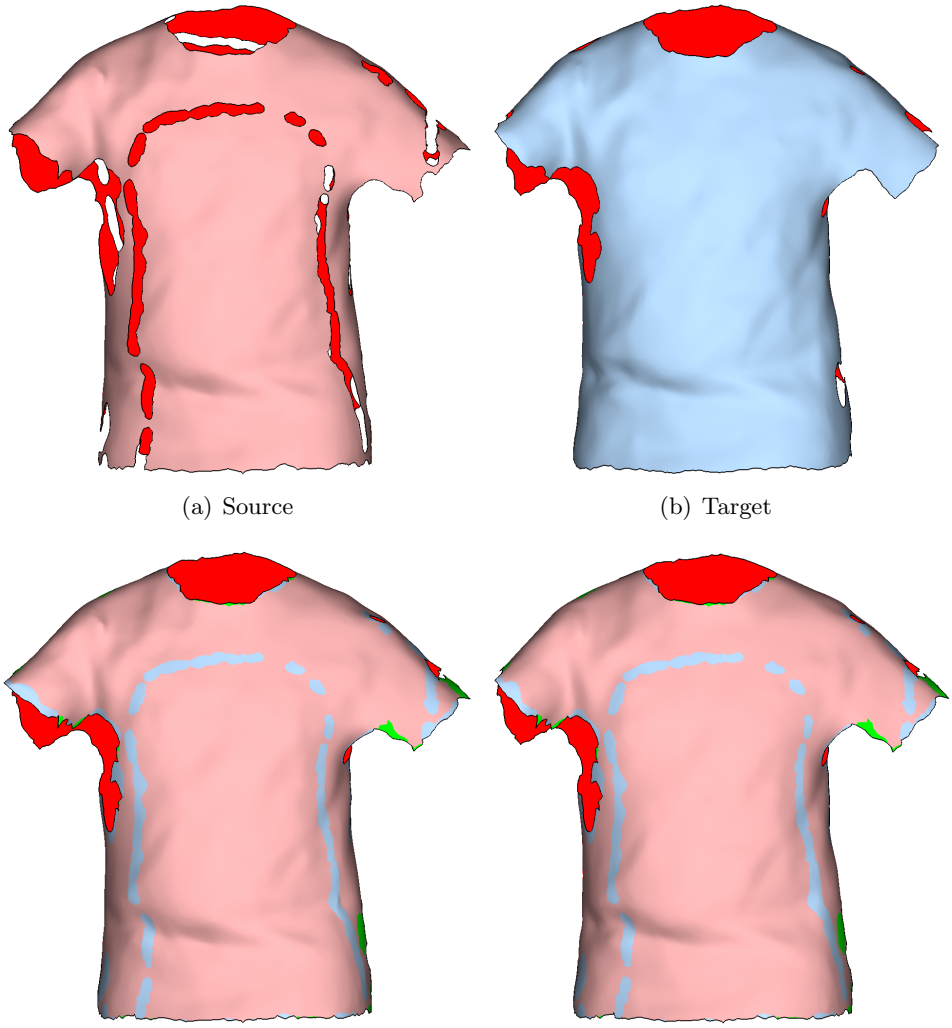
We perform the above steps on all deformation sets for all unknown vertices, giving us a single consistent frame sequence for both source and target frames, with a common mesh connectivity and positions for all geometry present in both the source and target.

5.3 Completion Results

We present the results of our geometry completion algorithm on selected frames from the *T-shirt* (Figures 5.8 and 5.9) dataset, and the *dog hand-puppet* (Figure 5.10) dataset. For more info on these datasets, see Chapter 6. For the completed images, the light blue geometry originated in the source frame, and the pink geometry originated in the target frame. In addition, we color the unmapped source geometry as green.

The T-shirt results (Figures 5.8 and 5.9) demonstrate cases of large amounts of missing source geometry that is completed with target geometry. Our algorithm is able to seamlessly complete the long strips of missing data on the front of the shirt, and complete the missing geometry under the arms.

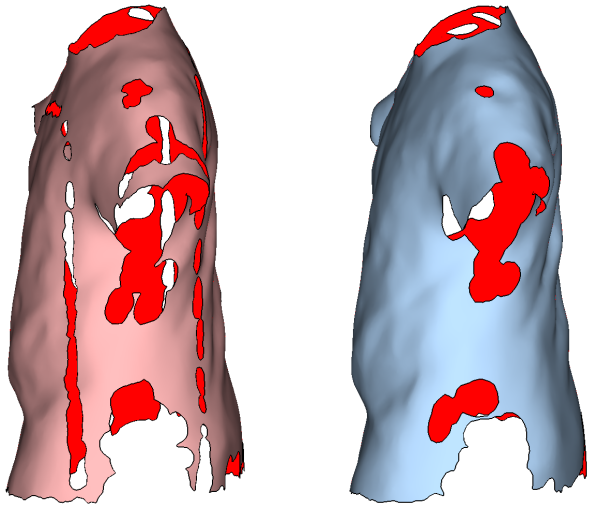
In this frame of the the dog hand-puppet dataset (Figure 5.10) the wrist of the puppeteer is missing in the source frame (Figure 5.10(a)), but is present in the target frame. Our algorithm is able to robustly complete the non-planar wrist geometry.



(a) Source (b) Target

(c) Completed Source (d) Completed Target

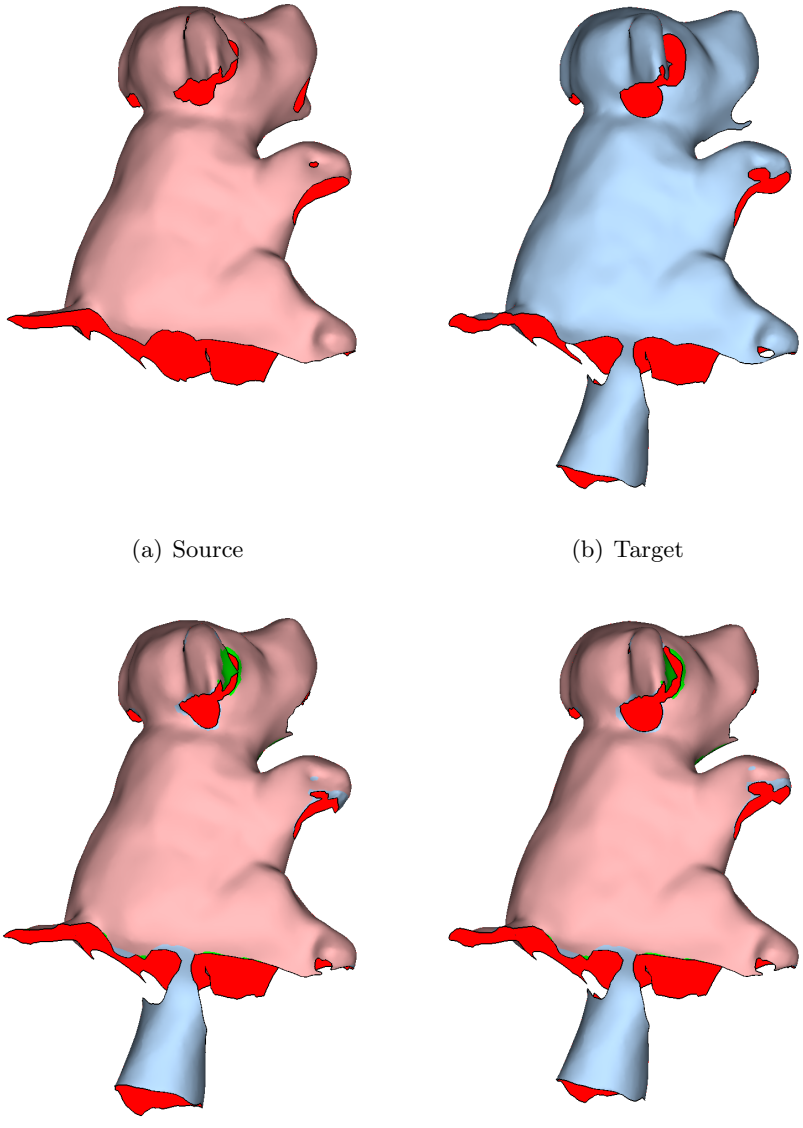
Figure 5.8: *T-shirt* completion results.



(a) Source (b) Target

(c) Completed Source (d) Completed Target

Figure 5.9: *T-shirt* completion results.



(a) Source (b) Target

(c) Completed Source (d) Completed Target

Figure 5.10: *Dog hand-puppet* completion results.

Chapter 6

Results

We demonstrate our reconstruction method's results on three datasets (two spheres, a dog hand-puppet, and a t-shirt). The *two spheres* dataset in Figure 3.1 is synthetic, generated using virtual cameras which simulate occlusion and other artifacts [32]. To simulate optical flow we use known correspondences on around 1% randomly sampled mesh vertices. Synthetic data is easier to control, generating specific contact situations we wanted to test for. All other datasets are captures of real objects. The initial per-frame meshes for all the models were reconstructed using [6].

Results Visualization We display the input per-frame meshes with back facing triangles highlighted in red, effectively showing the holes and missing geometry. The output consistent geometry is drawn with random colored circles that are consistent on meshes with the same connectivity. This visualization effectively shows the cross-parameterization between different frames.

The *spheres* dataset (Figure 3.1) is a short 8 frame sequence that illustrates the main features of the method, highlighting both inconsistency correction and geometry propagation.

The *dog hand-puppet* dataset (Figure 6.1) is obtained from multi-view video capture and demos our method's ability to reconstruct complex contact situations, such as when the dog's paws are brought in front of the chest. Our method successfully recovers the paw shape as well as the occluded chest geometry in this situation.

The *T-Shirt* sequence (Figure 6.2) from [7] highlights the robustness of our completion mechanism. Even though no template or strong priors are used, our results are nearly identical to those of Bradley et al. The difference is only in areas where no data is available in the inputs and where Bradley et al. use the template for completion, such as under the arms.

As demonstrated by these examples our method is capable of correctly reconstructing globally consistent geometry and motion even in the presence of complex contacts and missing data. In contrast to many previous techniques it does so without relying on strong priors such as a template or skeleton, which are often non-available. It also avoids over-smoothing and loss of details common to many other methods [13, 38, 39] preserving per-frame geometry except for regions where it violates the gradual change prior.

Runtimes Most of the tested per-frame meshes had on the order of $50K$ triangles. Since our algorithm is highly parallelizable, we ran it on a cluster of 27 Intel Xeon 3Ghz CPUs. The run-time is split roughly equally among three operations: initial patch-based parameterization, optimization and completion. The first two take about five minutes per frame. While per-frame completion is very fast, a few seconds, the completion time grows to five to ten minutes at higher levels of the hierarchy where it is propagated across longer and longer sequences.

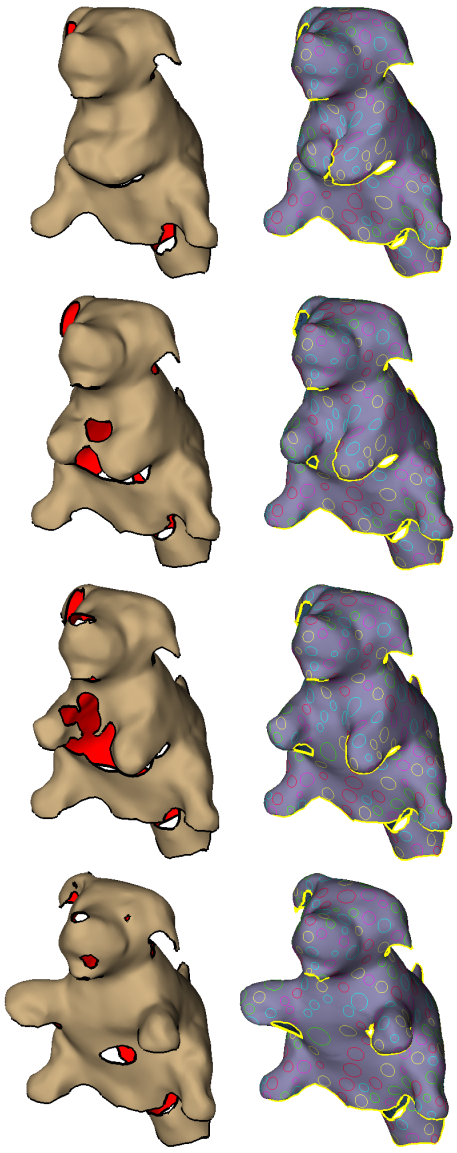


Figure 6.1: Dog hand-puppet reconstruction: (left) input per-frame meshes, (right) consistent geometry and motion reconstruction.

Figure 6.2: T-shirt reconstruction: (left) input per-frame meshes, (right) consistent geometry and motion reconstruction.

Chapter 7

Future Work

Our method has a number of limitations. Our current implementation of geometry reconstruction for close-by or contact surfaces can lead to surface self-intersections. These can be detected fairly easily, but would require some variation of temporally consistent deformation [29] to resolve. Another limitation of our method is temporal locality when checking for inconsistencies. Currently we only consider the source and target frames, and give preference to the source in situations where both geometries provide acceptable interpretation. By considering a larger temporal window we might be able to better detect situations where the target geometry is the correct one.

Because our method only uses the geometry available in the initial per frame reconstruction, we cannot reconstruct details that are never present in the input. This can be improved by creating a tool for an artist to design the missing details in a single frame, and propogating those details throughout the entire sequence in the same way we propogate missing geometry in Section 5.2. Such an approach could also be used to smooth out the jagged boundaries of the output models.

When propagating geometry we only use the boundary as a constraint while preserving the local detail, but the local details can undergo other changes. A possible improvement would be to use the optical flow as a soft constraint in the propagation. This approach is similar to the approach that Popa et al. [29] take, where they used the images from the capture to restore wrinkle details lost in the reconstruction of garments.

Chapter 8

Conclusion

We have presented a method for reconstructing a consistent frame sequence from a sequence of point clouds captured using multiple video streams, using the gradual change assumption as prior. As demonstrated by the examples we are able to robustly complete missing features with the matching geometry from another frame. Our method uses optical flow to create a cross-parameterization between sequential frames which we use to identify geometry that will complete the holes in a single frame. This approach allows us to accumulate all of the geometric information available in the sequence by completing geometry between each frame pair in a hierarchical manner, and we have demonstrated results of this applied to an entire captured sequence.

Bibliography

[1] N. Ahmed, C. Theobalt, C. Roessl, S. Thrun, and H.-P. Seidel. Dense correspondence finding for parameterization-free animation reconstruction from video. In *Proc. CVPR*, pages 1–8, 2008.

[2] Marc Alexa. Linear combination of transformations. *ACM Trans. Graph.*, 21(3):380–387, 2002.

[3] Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. Scape: shape completion and animation of people. *ACM Trans. Graph.*, 24(3):408–416, 2005. ISSN 0730-0301. doi: http://doi.acm.org/10.1145/1073204.1073207.

[4] Gerhard H. Bendels, Ruwen Schnabel, and Reinhard Klein. Detail-preserving surface inpainting. In *The 6th International Symposium on Virtual Reality, Archaeology and Cultural Heritage (VAST)*, pages 41–48. Eurographics Association, Eurographics Association, November 2005.

[5] J.-Y. Bouguet. Pyramidal implementation of the lucas kanade feature tracker: Description of the algorithm. Technical report, Intel Corporation, Microprocessor Research Labs, 1999.

[6] D. Bradley, T. Boubekeur, and W. Heidrich. Accurate multi-view reconstruction using robust binocular stereo and surface meshing. In *Proc. CVPR*, 2008.

[7] Derek Bradley, Tiberiu Popa, Alla Sheffer, Wolfgang Heidrich, and

Tamy Boubekeur. Markerless garment capture. *ACM Trans. Graph.*, 27(3):99, 2008.

[8] John Branch, Flavio Prieto, and Pierre Boulanger. Automatic hole-filling of triangular meshes using local radial basis function. In *Proceedings of the Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06)*, 3DPVT '06, pages 727–734. IEEE Computer Society, 2006.

[9] Toby P. Breckon and Robert B. Fisher. Non-parametric 3d surface completion. In *Proceedings of the Fifth International Conference on 3-D Digital Imaging and Modeling*, 3DIM '05, pages 573–580, Washington, DC, USA, 2005. IEEE Computer Society.

[10] Will Chang and Matthias Zwicker. Automatic registration of articulated shapes. *Computer Graphics Forum (Proc. SGP)*, (5):1459–1468, 2008.

[11] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. Variational shape approximation. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 905–914. ACM, 2004.

[12] J. Davis, S.R. Marschner, M. Garr, and M. Levoy. Filling holes in complex surfaces using volumetric diffusion. In *3D Data Processing Visualization and Transmission, 2002. Proceedings. First International Symposium on*, pages 428 –441, june 2002.

[13] Edilson de Aguiar, Carsten Stoll, Christian Theobalt, Naveed Ahmed, Hans-Peter Seidel, and Sebastian Thrun. Performance capture from sparse multi-view video. *ACM Trans. Graph.*, 27(3):98, 2008.

[14] Yasutaka Furukawa and Jean Ponce. Dense 3d motion capture from synchronized video streams. In *Proc. CVPR*, pages 1–8, 2008.

[15] Yasutaka Furukawa and Jean Ponce. Dense 3d motion capture for human faces. In *Proc. CVPR*, pages 1–8, 2009.

[16] B. Goldlucke, I. Ihrke, C. Linz, and M. Magnor. Weighted minimal hypersurface reconstruction. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(7):1194–1208, 2007. ISSN 0162-8828.

[17] Graphite, 2003. http://www.loria.fr/∼levy/Graphite/index.html.

[18] Q.-X. Huang, B. Adams, M. Wiche, and L.-J. Guibas. Non-rigid registration under isometric deformations. *Computer Graphics Forum (Proc. SGP)*, 27(5):1449–1457, 2008.

[19] Vladislav Kraevoy and Alla Sheffer. Template-based mesh completion. In *Proc. Symposium on Geometry Processing (SGP)*, page 13, 2005.

[20] Hao Li, Bart Adams, Leonidas J. Guibas, and Mark Pauly. Robust single-view geometry and motion reconstruction. *ACM Trans. Graph.*, 28(5), 2009.

[21] Peter Liepa. Filling holes in meshes. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, SGP '03, pages 200–205. Eurographics Association, 2003.

[22] Yaron Lipman, Olga Sorkine, Marc Alexa, Daniel Cohen-Or, David Levin, Christian Rössl, and Hans-Peter Seidel. Laplacian framework for interactive mesh editing. *International Journal of Shape Modeling (IJSM)*, 11(1):43–61, 2005.

[23] N. J. Mitra, S. Flory, M. Ovsjanikov, N. Gelfand, L. Guibas, and H. Pottmann. Dynamic geometry registration. In *Proc. Symposium on Geometry Processing (SGP)*, pages 173–182, 2007.

[24] Sang Il Park and Jessica Hodgins. Data-driven modeling of skin and muscle deformation. *ACM Trans. Graph.*, 27(3):96, 2008.

[25] Seyoun Park, Xiaohu Guo, Hayong Shin, and Hong Qin. Surface completion for shape and appearance. *Vis. Comput.*, 22(3):168–180, March 2006.

[26] Mark Pauly, Niloy J. Mitra, Joachim Giesen, Markus Gross, and Leonidas J. Guibas. Example-based 3d scan completion. In *Proceedings of the third Eurographics symposium on Geometry processing*, SGP '05. Eurographics Association, 2005.

[27] Yuri Pekelny and Craig Gotsman. Articulated object reconstruction and markerless motion capture from depth video. *Computer Graphics Forum (Proc. Eurographics)*, 27(2):399 – 408, 2008.

[28] T. Popa, I. South-Dickinson, D. Bradley, A. Sheffer, and W. Heidrich. Globally consistent space-time reconstruction. *Computer Graphics Forum*, 29(5):1633–1642, 2010.

[29] Tiberiu Popa, Qingnan Zhou, Derek Bradley, Vladislav Kraevoy, Hongbo Fu, Alla Sheffer, and Wolfgang Heidrich. Wrinkling captured garments using space-time data-driven deformation. *Computer Graphics Forum (Proc. Eurographics)*, 28(2):427–435, 2009.

[30] J. Schreiner, A. Asirvatham, E. Praun, and H. Hoppe. Inter-surface mapping. *ACM Trans. Graph.*, 23(3):870–877, 2004.

[31] Andrei Sharf, Marc Alexa, and Daniel Cohen-Or. Context-based surface completion. *ACM Trans. Graph.*, 23(3):878–887, August 2004.

[32] Andrei Sharf, Dan A. Alcantara, Thomas Lewiner, Chen Greif, Alla Sheffer, Nina Amenta, and Daniel Cohen-Or. Space-time surface reconstruction using incompressible flow. *ACM Trans. Graph.*, 27(5): 110, 2008.

[33] Alla Sheffer, Bruno Lévy, Maxim Mogilnitsky, and Alexander Bogomyakov. ABF++: fast and robust angle based flattening. *ACM Trans. Graph.*, 24(2):311–330, 2005.

[34] Jonathan Richard Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In Ming C. Lin and Dinesh

Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, May 1996. From the First ACM Workshop on Applied Computational Geometry.

[35] Jochen Sussmuth, Marco Winter, and Gunther Greiner. Reconstructing animated meshes from time-varying point clouds. *Computer Graphics Forum*, 27(5):1469–1476, 2008.

[36] Art Tevs, Martin Bokeloh, Michael Wand, Andres Schilling, and Hans-Peter Seidel. Isometric registration of ambiguous and partial data. In *Proc. CVPR*, pages 1185–1192, 2009.

[37] Kiran Varanasi, Andrei Zaharescu, Edmond Boyer, and Radu P. Horaud. Temporal surface tracking using mesh evolution. In *Proc. ECCV*, volume Part II, pages 30–43, 2008.

[38] Daniel Vlasic, Ilya Baran, Wojciech Matusik, and Jovan Popović. Articulated mesh animation from multi-view silhouettes. *ACM Trans. Graph.*, 27(3):97, 2008.

[39] Michael Wand, Bart Adams, Maksim Ovsjanikov, Alexander Berner, Martin Bokeloh, Philipp Jenke, Leonidas Guibas, Hans-Peter Seidel, and Andreas Schilling. Efficient reconstruction of nonrigid shape and motion from real-time 3d scanner data. *ACM Trans. Graph.*, 28(2):15, 2009.

[40] Chunxia Xiao, Wenting Zheng, Yongwei Miao, Yong Zhao, and Qunsheng Peng. A unified method for appearance and geometry completion of point set surfaces. *Vis. Comput.*, 23(6):433–443, May 2007.

[41] Wei Zhao, Shuming Gao, and Hongwei Lin. A robust hole-filling algorithm for triangular mesh. *Vis. Comput.*, 23(12):987–997, November 2007.