Motion Graphs for Unstructured Textured Meshes

Fabián Prada* Misha Kazhdan* Ming Chuang† Alvaro Collet† Hugues Hoppe†

Abstract

Scanned performances are commonly represented in virtual environments as sequences of textured triangle meshes. Detailed shapes deforming over time benefit from meshes with dynamically evolving connectivity. We analyze these unstructured mesh sequences to automatically synthesize motion graphs with new smooth transitions between compatible poses and actions. Such motion graphs enable natural periodic motions, stochastic playback, and user-directed animations. The main challenge of unstructured sequences is that the meshes differ not only in connectivity but also in alignment, shape, and texture. We introduce new geometry processing techniques to address these problems and demonstrate visually seamless transitions on high-quality captures.

Keywords: shape similarity, seamless transitions, shape morphing, looping, stochastic motion, video textures, Markov chain

Concepts: •**Computing methodologies** → *Computer graphics;*

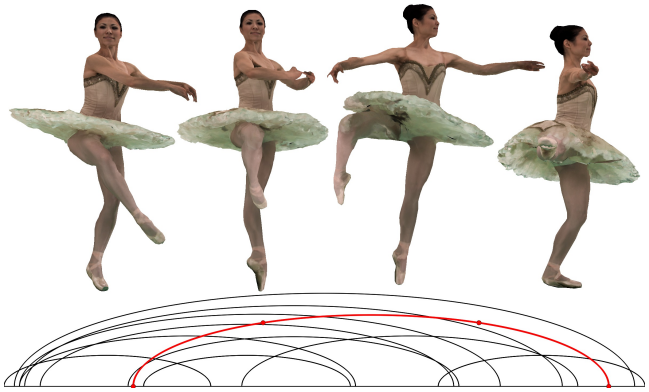

Figure 1: *Example of a synthesized mesh sequence introduced to smoothly transition between similar poses. Each mesh corresponds to a colored node on the transition arc above the timeline.*

1 Introduction

Immersive virtual environments benefit from dynamic content scanned from real-world performances. In such environments, the viewer location is not known a priori and can change rapidly, so the content must be renderable from arbitrary viewpoints. Several techniques achieve such *free-viewpoint video*, as reviewed in the next section. Our work focuses on approaches that reconstruct a textured triangle mesh at each time frame. The sequence of meshes may have arbitrarily changing connectivity.

The input mesh sequence may be thought of as a directed path graph, in which each node corresponds to a mesh pose, and there is an edge between each successive pair of meshes. Our goal is to extend this to a more general *motion graph* [Kovar et al. 2002], with additional edges that correspond to new, synthesized motion transitions between similar frames (Figure 1). Related work on skeletal animation [Kovar et al. 2002; Heck and Gleicher 2007] and temporally coherent meshes [Casas et al. 2012] has shown that motion graphs enable several useful scenarios:

Periodic motions: The captured sequence may contain motions that are naturally periodic (e.g., walking or running), so graph cycles should correspond to seamless looping of such motions.

Stochastic motion: The graph can be randomly traversed to synthesize infinite motion, while maintaining smooth transitions, avoiding obvious repetition, and maximizing content variety.

User-directed action: During motion playback, the user can interactively specify a target frame (e.g., from a walking/running/jumping sequence), and a path is found within the graph to quickly transition to that frame, again while preserving visual smoothness.

Our contribution is to extend these ideas to sequences of unstructured textured meshes [Bojsen-Hansen et al. 2012]. Specifically, we address the following challenges:

- Compute a similarity matrix to identify temporal windows with similar poses and use it to select new transitions to introduce.
- For each transition, track a template through the frames so that all frames share the same mesh connectivity.
- Compute a sequence of textured meshes that achieves a smooth visual transition across the two temporal windows.
- Define transition probabilities that allow stochastic traversal of the graph such that the steady-state probabilities of visiting nodes are as uniform as possible.

The main difficulty lies in the construction of smooth transitions, which requires morphing of the mesh geometry together with ghost-free texture interpolation. This is challenging because even in natural periodic motion, frames are often misaligned, the mesh shapes deform (e.g., arm positions, cloth folds), and reconstructed texture features change (e.g., facial expressions). These differences become more significant when transitioning between unrelated motions.

2 Related work

Free-viewpoint video There are numerous techniques for recording a live performance using multiple cameras such that it may later be experienced from a full 360-degree range of inward-looking viewpoints [e.g., Kanade et al. 1997; Carranza et al. 2003; Vlasic et al. 2009; Bojsen-Hansen et al. 2012; Casas et al. 2014; Zitnick et al. 2004; Zollhöfer et al. 2014; Huang et al. 2014; Collet et al. 2015]. Most techniques reconstruct an explicit surface representation (triangle mesh), which is then shaded using a merged texture atlas or a view-dependent combination of the original videos.

*Johns Hopkins University †Microsoft Corporation

SIGGRAPH '16 Technical Paper, July 24-28, 2016, Anaheim, CA
ISBN: 978-1-4503-4279-7/16/07
DOI: http://dx.doi.org/10.1145/2897824.2925967

These techniques can be broadly categorized according to whether they reconstruct a new surface at each time frame (e.g., using shape from silhouettes or multiview stereo), deform a canonical or precomputed geometric model (e.g., body mesh) to fit each frame, or use a hybrid combination of per-frame reconstruction and inter-frame tracking. Reusing the same *keyframe* mesh connectivity over several consecutive frames (or even the entire sequence) can improve robustness and encoding efficiency. On the other hand, allowing dynamic mesh connectivity offers greater flexibility in capturing general scenes, e.g., with free-flowing clothing or interacting objects.

Our work supports input with arbitrarily changing mesh connectivity and is therefore broadly applicable. We demonstrate it using datasets from both [Starck and Hilton 2007] and [Collet et al. 2015].

Motion graphs Given a skeletal motion capture sequence, motion graphs [Arikan and Forsyth 2002; Kovar et al. 2002; Lee et al. 2002] are constructed by identifying temporal windows of frames with similar pose and constructing smooth transitions between them, e.g., by spherical linear interpolation of joint rotations. These graphs are used to synthesize new motion paths that closely follow given constraint curves or respond to interactive controls.

Several papers extend motion graphs to shape sequences. Starck et al. [2005] construct motion graphs over human shapes by representing these as geometry images using spherical parameterizations. Huang et al. [2009] identify pairs of similar mesh frames and transition from one action to another at the matching frames, but without geometry or texture interpolation. The work of Casas et al. [2012] is quite related to ours in that it also constructs motion graphs over input mesh sequences and defines smooth transitions between different actions. Huang et al. [2015] extend this framework to enable the use of separately captured skeletal motion data to drive the character animation. Volino et al. [2015] enable interactive animation control in a web-based browser.

There exists a key distinction from our work though: prior techniques for creating smooth transitions assume that all input meshes are well-registered and share the same connectivity. This is a strong assumption to make, as in practice it is difficult to register a static template to all scanned frames [Bojsen-Hansen et al. 2012] (due to fast motion, incompatible topology, flexible and free-form materials, etc.). In contrast, we make no such assumption. We explicitly construct *locally* consistent connectivity for the transition frames. This enables a broader range of input from more complex scenarios.

Image-based transitions Xu et al. [2011] achieve image-based synthesis of realistic human video animations. The surface geometry is used to guide the warping of 2D images into the output frame and is not itself rendered into the final output. Changing the viewpoint in this system requires re-running the matching/deformation pipeline. Several techniques address texture misalignment by warping 2D screen-space renderings using image-based optical flow at runtime [Eisemann et al. 2008; Casas et al. 2014, 2015]. The images are rendered, warped according to the flow field, and blended for a specific viewpoint. Casas et al. [2015] partially avoid costly runtime computations by precomputing the image-based optical flow from multiple viewpoints. In contrast, we compute optical flow directly over the triangle mesh, storing the final corrected texture into an atlas and avoiding view-dependent runtime computations.

3 Overview

The input $\mathcal{F} = \{(F_0, I_0), \ldots, (F_{n-1}, I_{n-1})\}$ consists of a sequence of textured mesh frames, where each frame is a triangle mesh $F_i = (K_i, V_i)$ with connectivity K_i and vertex positions V_i, and I_i is the associated texture. A *motion graph* $\mathcal{G} = (\mathcal{F}, \mathcal{E})$ is a directed graph in which each node corresponds to a frame and each edge $e \in \mathcal{E} \subseteq \mathcal{F} \times \mathcal{F}$ represents the possible transition from one

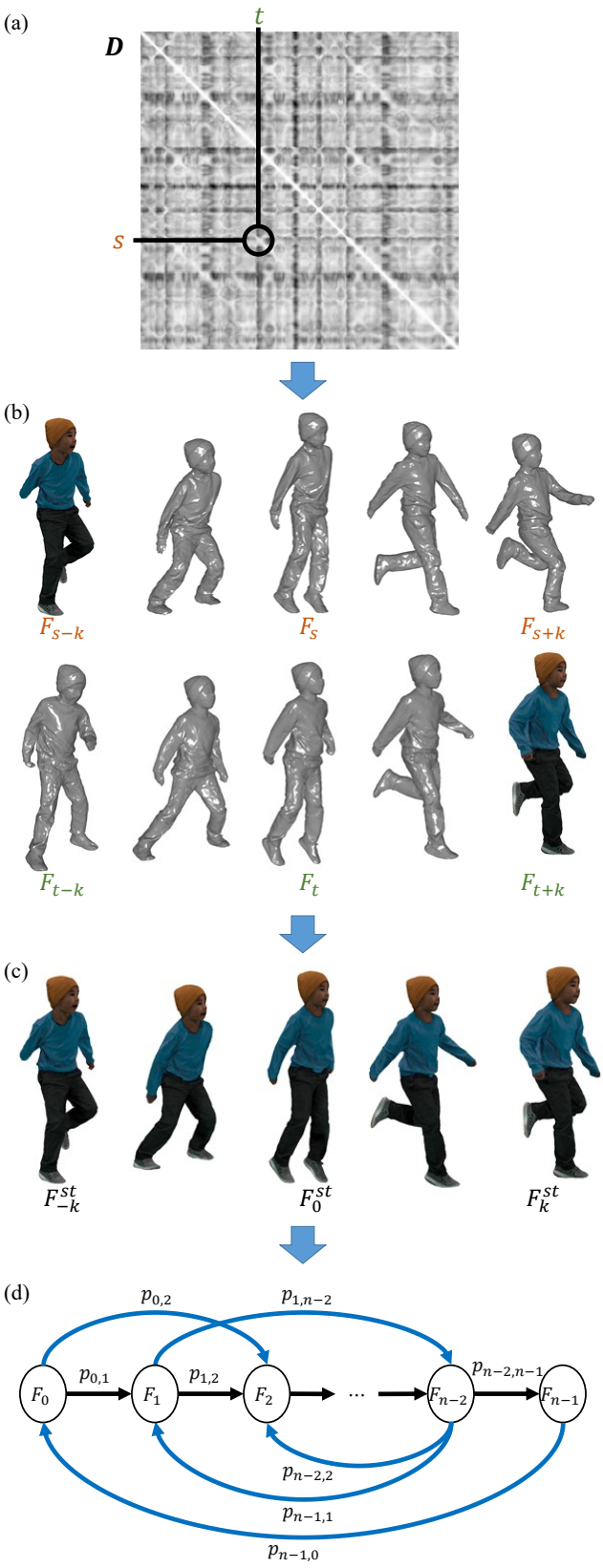

Figure 2: *Illustration of the steps in our processing pipeline: (a) compute the inter-window dissimilarity matrix* $\mathbf{D}$*, (b) fit a fixed-connectivity template to the frames of similar windows, (c) synthesize the textured transition frames, and (d) assign transition probabilities to the derived motion graph.*

frame to another. Thus, for the initial sequence, the motion graph is a *directed path graph* as there is an edge between two consecutive frames. Note that the first frame F_0 has no incoming edge, and the last frame F_{n-1} has no outgoing edge. Because this graph is acyclic, it does not permit any repeating motion. And, because no node has more than one outgoing edge, it only permits a single path.

Our goal is to augment this graph with a new set of motion transitions. A simple scheme would be to successively identify a pair of similar frames using a distance function D, e.g., to find a source frame F_s and a target frame F_t, such that $D(F_s, F_{t-1}) < \epsilon$ and $D(F_{s+1}, F_t) < \epsilon$ for some error threshold ϵ, and to simply augment the motion graph with the new edge (F_s, F_t). However, as the corresponding frames may come from different animation sequences, this new transition can easily result in a visible discontinuity during playback.

Instead, our approach is to find loosely matching windows $W_s = \{F_{s-k}, \ldots, F_{s+k}\}$ and $W_t = \{F_{t-k}, \ldots, F_{t+k}\}$ and then *deform* the windows by synthesizing new transition frames between F_{s-k} and F_{t+k}. The transition window $W_{s\to t}$ consists of a short fixed-size sequence of synthesized frames (Figure 3):

$$W_{s\to t} = (F^{st}_{-k}, F^{st}_{-k+1}, \ldots, F^{st}_{k-1}, F^{st}_{k}),$$

where $\{F^{st}_i\}$ are newly synthesized frames and $F^{st}_{-k} \approx F_{s-k}$ and $F^{st}_{k} \approx F_{t+k}$ (with equivalence up to tessellation). This sequence itself forms a path graph, and is merged with the motion graph.

After adding a set of transitions $\{W_1, \ldots, W_l\}$, the new motion graph contains the nodes $\mathcal{F} \cup \{F^{W_1}\} \cup \ldots \cup \{F^{W_l}\}$ and contains the union of the original graph edges and all new transition paths. We transform this graph into a Markov chain graph by assigning transition probabilities to each node, such that a random walk visits all nodes with approximately equal probability.

A visualization of our pipeline is shown in Figure 2 with the matrix of dissimilarity measures shown in the top row, a candidate transition pair in the second row, the synthesized transition in the third, and the motion graph in the bottom.

The next sections discuss how to identify candidate pairs of similar frames s, t (Section 4), how to choose and fit a single template to the meshes in the source and target windows (Section 5), how to interpolate between the source and target to obtain the new synthesized window $W_{s\to t}$ (Section 6), and how to assign transition probabilities on the graph edges so that the steady-state frequency of visiting nodes during a random walk is close to uniform (Section 7).

4 Finding candidate transition windows

To identify potential transitions we search for pairs of frames F_s and F_t, with the property that the window around F_s is similar to the window around F_t. The use of a window also ensures that the source and target surfaces have similar velocities.

This relates the problem of choosing transitions to the problem of measuring shape similarity, a problem that has been well-studied in the graphics literature with numerous shape descriptors proposed for efficient retrieval [e.g., Shilane et al. 2004; Tangelder and Veltkamp 2007; Huang et al. 2010]. In our application the shape matching is performed offline, allowing us to use techniques that are more discriminating but also more expensive to compute.

In particular, we leverage the exhaustive search approach proposed by Funkhouser et al. [2004], where a dissimilarity score between two pairs of models is computed by first translating each model so that its center of mass is aligned with the origin, and then searching over *all* rotations about the origin for the one minimizing the sum-of-squared distances between the shapes.

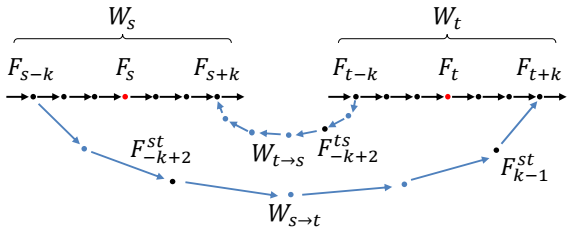

Figure 3: *Given two similar windows centered about frames F_s and F_t in the input sequence, we construct two new mesh sequences $W_{s\to t}$ and $W_{t\to s}$ which provide smooth transitions between opposite endpoints of the two windows.*

Letting χ_s and χ_t be the indicator functions of the two surfaces and letting EDT^2_s and EDT^2_t be the (squared) Euclidean Distance Transforms, this distance is expressed as:

$$D_{st}(R) = \langle \text{EDT}^2_s, R(\chi_t)\rangle + \langle \chi_s, R(\text{EDT}^2_t)\rangle$$

with $R(\Phi)$ the rotation of Φ (i.e. $R(\Phi)(p) = \Phi(R^{-1}(p))$).

Following the work of [Huang et al. 2009], we adapt this approach in two ways. To ensure that we match the action around frames F_s and F_t we compare local windows rather than individual frames. And, since actions are defined in relation to the ground, we only search for frames that are similar up to a rotation about the (vertical) z-axis. This gives a windowed dissimilarity matrix $\mathbf{D}$ with entries:

$$\mathrm{d}_{st} = \min_{R\in SO(2)} \sum_{i=-k}^{k} \omega(k) \cdot D_{s+k,t+k}(R)$$

where the rotation R is taken from the group of rotations about the z-axis and $\omega : [-k, k] \to \mathbb{R}$ is a kernel, centered at zero, that gives more weight to the matching score of frames that are closer to F_s and F_t respectively. As in [Funkhouser et al. 2004], we perform the matching efficiently using the Fast Fourier Transform, avoiding the more expensive brute-force search used by [Huang et al. 2009].

Given the dissimilarity matrix we define a set of candidate transitions by iteratively choosing pairs of frames that give low distance scores (so that we are more likely to succeed in synthesizing a transition) and are sufficiently far from previously chosen pairs (to avoid creating many transitions between the same pairs of actions). As we iterate the pair selection, we try to synthesize transitions (Section 5 and 6). If the synthesis fails, we discard the failed pair to allow for the selection of other candidate transitions in the vicinity.

As shown in Figure 2 (second row), the windowed matching generates candidate pairs that have similar action, while the weighting kernel ensures that the center frames F_s and F_t are close.

Choice of descriptor We use the descriptor in [Funkhouser et al. 2004] because it is easy to compute, is efficient to match using the FFT, and has been shown to retrieve models with high precision. While other descriptors have been proposed for matching dynamic meshes, there has been little work comparing such descriptors. (Although Huang et al. [2010] show that their volumetric shape histograms outperform the spherical harmonic descriptor of [Kazhdan et al. 2003], the practical implications are unclear for two reasons. First, the descriptor of [Kazhdan et al. 2003] has been shown to be less discriminating than the descriptor of [Funkhouser et al. 2004] that we use. Second, [Huang et al. 2010] compare shape histograms with exhaustive search to the spherical harmonic representation with rotation invariance. As rotation invariance discards salient information, the diminished performance of the spherical harmonic representation is expected and it is unclear how the descriptors would compare if exhaustive search were used for both.)

5 Tracking the windows consistently

Having found frames F_s and F_t with similar windows $W_s = \{F_{s-k}, \ldots, F_{s+k}\}$ and $W_t = \{F_{t-k}, \ldots, F_{t+k}\}$, our goal is construct *two* new transition sequences: one from the beginning of W_s to the end of W_t and the other from the beginning of W_t to the end of W_s, as shown in Figure 3. (Note that $W_{s\to t}$ and $W_{t\to s}$ do not correspond to opposite edges in the motion graph because they are adjacent to different nodes.)

In this section, we describe the first phase of this processing – fitting a tetrahedral template mesh to all frames in W_s and W_t, so that all frames are tessellated with a common connectivity. In Section 6, we use this shared tessellation to synthesize the new transition frames.

Pseudocode **TrackTemplate** outlines the steps of the fitting phase. An initial surface template is chosen and extended to a volumetric template. Then the template is propagated through (and across) the frames in W_s and W_t to obtain the tracked windows $\overline{W}_s$ and $\overline{W}_t$.

TrackTemplate(W_s , W_t , ϵ , ε , L)

```
1   F_a ← SelectTemplate( W_s , W_t )
2   F̄_a ← Tetrahedralize( F_a )
3   Q ← {a}
4   while Q ≠ ∅
5     q ← Pop( Q )
6     for r ∈ UnprocessedNeighbors( q )
7       F̃_r ← PropagateSurface( ∂F̄_q )
8       F̃_r ← DetailTransfer( F̃_r , F_r , ϵ , ε , L )
9       F̄_r ← PropagateVolume( F̄_q , F̃_r )
10      Push( Q , r )
```

5.1 SelectTemplate

We select the connectivity from among the meshes in W_s and W_t, and use the frame F_a which best satisfies the criteria proposed by Collet et al. [2015] (in decreasing order of priority): more connected components, lower genus, larger surface area.

5.2 Tetrahedralize

We tetrahedralize the interior of F_a using a constrained Delaunay triangulation that introduces Steiner points and has area-edge ratio bounded by 1.8 [Si 2015] to get the volumetric template $\overline{F}_a$.

5.3 UnprocessedNeighbors

Noting that adjacent poses in the source (respectively, target) have similar geometries, we propagate the template within the source (respectively, target) by registering tracked frames to their untracked neighbors. Using the similarity of frames F_s and F_t, we propagate from the source to the target (or vice versa in the case that $F_a \in W_t$) by registering $\overline{F}_s$ to F_t (Figure 4).

5.4 PropagateSurface

We begin the tracking process by leveraging the similarity between adjacent frames (and between F_s and F_t) and using the implementation of non-rigid ICP described by Li et al. [2009] to register the boundary of the tracked template, $\partial\overline{F}_q$, to the neighbor's geometry, F_r. The Hausdorff distance between $\partial\overline{F}_q$ and F_r is measured to determine whether the tracking is successful. In our examples, we abort the transition synthesis when the Hausdorff distance exceeds 10 cm.

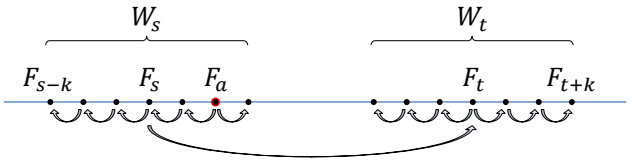

Figure 4: *Tracking propagation order, starting from the heuristically selected best frame F_a and propagating across the two windows W_s and W_t at the identified most similar frames F_s and F_t. The end result is a tracked mesh F for each frame $F \in W_s \cup W_t$, all with the same connectivity as F_a.*

5.5 DetailTransfer

Having performed ICP to coarsely register the template's surface to the input, we further deform the geometry of the template to better match the input's detail.

When the volumetric meshes have a consistent tessellation, local coordinate frames are related and the differential coordinates [Sorkine and Alexa 2007] of the input can be transferred to the template [Casas et al. 2014]. As we do not assume that the input and template meshes share a common tetrahedralization, such an approach is not feasible. Instead, we use the surface normals (which do not require relating local frames) to encode the detail.

As in the work of Li et al. [2009] we transfer detail by offsetting vertices on the template so that the deformed mesh better matches the input. However, rather than seeking the offsets that bring the template vertices closer to the input, we first independently model the template and input's detail as offsets along a smoothed normal field, and then replace the template's offsets with those of the input.

Figure 5 compares the two approaches, with the tracked template and input geometry shown on the left and the results for [Li et al. 2009] and our method shown on the right. (To better reveal the detail, two passes of 1-to-4 subdivision are applied to the tracked mesh.) The figure highlights two limitations of the work of Li et al.: The smooth offset constraint required to mitigate the effects of misregistration also results in less effective transfer of the higher-frequency content. And, even using a smoothness constraint, the offset surface is noisy when the underlying normal field is not smooth.

Figure 5 also illustrates a limitation of our approach. Topological errors in the reconstructed surface may lead to inconsistent detail layers. For instance, near the elbow on the left side of the image, the template (respectively, input) has positive (respectively, negative) mean curvature so the smooth surface is obtained by insetting (respectively, offsetting). Consequently, the template surface is inset twice, shrinking the area around the elbow.

Pseudocode **DetailTransfer** outlines the steps of the algorithm. First we *hierarchically* define the template and input meshes as normal offsets from smoother base-domains (Figure 6). Then we replace the offsets of the template with those of the input.

Noting that the normal offset that smooths a surface corresponds to the mean-curvature, this approach is akin to replacing (hierarchically) the mean-curvature of the template with the mean-curvature of the input.

NormalOffsetToBase Given a mesh and smoothness parameters for the normal field and the target base layer, we first compute the smooth normals and then compute the normal offsets that generate the smooth layer.

SmoothNormals A naive method for smoothing the normals would first compute the Gauss map and independently diffuse the coordinate functions. The problem is that this approach ignores the

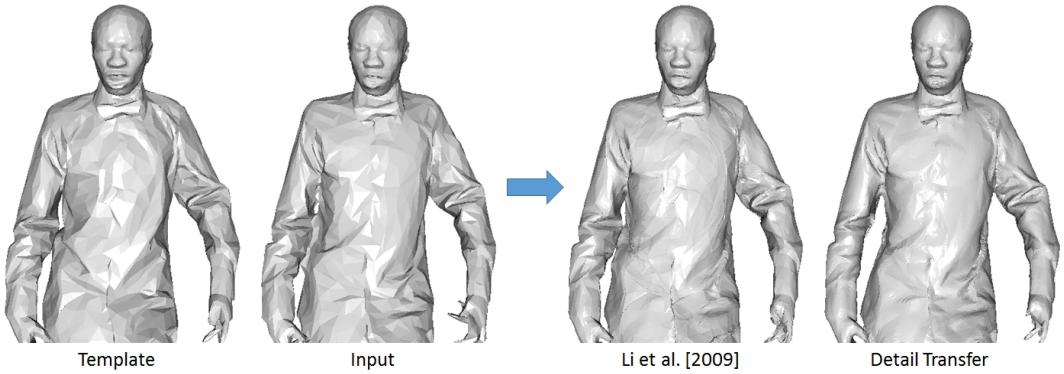

Figure 5: *Transferring detail from the tracked geometry to the input frame.*

DetailTransfer(F_0 , F_1 , ϵ , ε , L)

for $\alpha \in \{0,1\}$:
  $\tilde{F}_\alpha \leftarrow F_\alpha$
  for $l \in [0, L)$:
    ($\vec{n}^l_\alpha$, h^l_α) $\leftarrow$ **NormalOffsetToBase**($\tilde{F}_\alpha$, $\epsilon \cdot 4^l$, ε)
    $\tilde{F}_\alpha \leftarrow \tilde{F}_\alpha + h^l_\alpha \cdot \vec{n}^l_\alpha$
$\Phi^* \leftarrow$ **PullBack**($\tilde{F}_0$, $\tilde{F}_1$)
for $l \in [0, L)$:
  $\tilde{F}_0 \leftarrow \tilde{F}_0 - \Phi^*(h^l_1) \cdot \vec{n}^l_0$
return $\tilde{F}$

NormalOffsetToBase(F , ϵ , ε)

$\vec{n} \leftarrow$ **SmoothNormals**(F , ϵ)
$h \leftarrow$ **EstimateOffsets**(F , $\vec{n}$, ε)
return ($\vec{n}$, h)

fact that the normal field lies on the sphere. In practice, this can result in undesirable artifacts such as the simultaneous vanishing of the coordinate functions (making it impossible to define a unit-normal) or the convergence of each of the coordinate functions to a constant (which would not provide a meaningful normal field).

Instead, we simulate the diffusion of the normal field as a map between manifolds. That is, we constrain the normals to change in the tangent plane of the corresponding point on the sphere. Specifically, initializing $\vec{n}$ to be the Gauss map, we solve for the tangent offset field $\vec{t}$ that minimizes the energy:

$$E(\vec{t}) = \int_F \left[\left\|\vec{t}\right\|^2 + \epsilon \cdot \left\|\nabla(\vec{n} + \vec{t})\right\|^2 \right] dp,$$

with the first term penalizing large changes and the second encouraging smoothness of the resulting map.

Using the Euler-Lagrange formulation, the minimizer $\vec{t}$ is the solution to the screened-Poisson equation:

$$(1 - \varepsilon \cdot \Delta)\vec{t} = -\epsilon \cdot \pi(\Delta \vec{n}) \tag{1}$$

where the Laplace-Beltrami operator, Δ, is applied to each of the coordinate functions of $\vec{t}$ and $\vec{n}$ independently and π is the operator projecting onto the tangent space.

Lastly, we update the normal field by applying the offset and rescaling, $\vec{n} \leftarrow (\vec{n} + \vec{t})/\|\vec{n} + \vec{t}\|$.

This approach resolves both problems: Shifting a point on the sphere by a vector in the point's tangent space (1) only moves the point further from the origin, so it is impossible for it to be zero, and (2) after normalization, the shift only moves a point to a position in the same hemisphere, making it impossible to have all points move to the same position if the initial mapping was surjective.

Note that the diffusion can be iterated and, if it converges, the result is a *harmonic map* from F to the sphere. In particular, when F is genus-0 the resulting map is a *conformal* spherical parameterization. For examples we refer the reader to the supplemental material.

EstimateOffsets Given the initial embedding of the surface, $\phi : F \to \mathbb{R}^3$ (with $\phi(p) = p$ for all $p \in F$) and given the smoothed normal field $\vec{n}$, we solve for the normal offsets h minimizing:

$$E(h) = \int_F \left[\left\|h\right\|^2 + \varepsilon \cdot \left\|\nabla(\phi + h \cdot \vec{n})\right\|^2 \right] dp,$$

with the first term penalizing large offsets and the second encouraging smoothness of the resulting embedding.

Using the Euler-Lagrange formulation, the minimizer h is the solution to the screened-Poisson equation:

$$\big(1 - \varepsilon \cdot (\vec{n}^* \cdot \Delta \cdot \vec{n})\big)h = -\varepsilon \cdot (\vec{n}^* \cdot \Delta) \cdot \phi \tag{2}$$

where $\vec{n}^*$ is the dual of the normal field (taking a vector in $\mathbb{R}^3$ and returning the dot-product with the normal).

To mitigate the shrinking effects of mean-curvature flow, we adjust the offset function so that it has zero mean, $h \leftarrow h - \int_F h / \int_F 1$.

PullBack Since the surfaces F and G have already been registered using ICP, we simply define a correspondence map as the map taking a point on the source to the closest point on the target:

$$\Phi(p) = \operatorname*{arg\,min}_{q \in G} \|p - q\|^2, \qquad \forall p \in F. \tag{3}$$

The pull-back of this map is then used to transform the offsets computed on the target to offsets on the source:

$$\Phi^*(h_G) \equiv h_G \circ \Phi.$$

5.6 PropagateVolume

Finally, having registered the surface of the tracked mesh $\partial\overline{F}_q$ to the neighbor $\widetilde{F}_r$ we extend the registration to the interior of the volume. To do this, we solve the volumetric As-Rigid-As-Possible registration problem [Chao et al. 2010], seeking the position of the interior vertices of $\overline{F}_r$ that minimize the ARAP energy from the initial template $\overline{F}_a$ to $\overline{F}_r$ while locking the boundary vertices to the positions of $\widetilde{F}_r$.

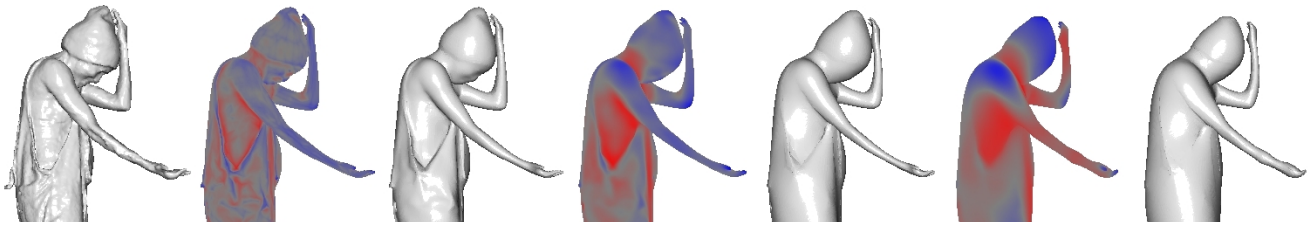

Figure 6: *Hierarchical detail extraction. Using the base layer from one pass of detail-transfer as the input to the next lets us extract details across different frequencies.*

6 Synthesizing seamless transitions

Having tracked the windows consistently, we now construct an animation that transitions from the first frame of the source window to the last frame of the target.

Pseudocode **SynthesizeTransition** outlines the construction steps. First, the target window is rigidly aligned to the source. Next, starting and ending textures are obtained by projecting the input textures, I_{s-k} and I_{t+k}, onto the tracked geometries, F^{st}_{-k} and F^{st}_{k}, and the optical flow field is estimated. Then, intermediate frames are constructed by minimizing a rigidity-invariant objective function and intermediate textures are generated by advecting the start texture forward, the end texture backward, and blending.

SynthesizeTransition($\overline{W}_s$, $\overline{W}_t$, ϵ , ε , L)

$(o, R) \leftarrow$ **AlignWindows**($\overline{F}_s$, $\overline{F}_t$)
$I^{st}_{-k} \leftarrow$ **ProjectTexture**(F_{s-k} , I_{s-k} , F^{st}_{-k})
$I^{st}_{k} \leftarrow$ **ProjectTexture**(F_{t+k} , I_{t+k} , F^{st}_{k})
$\vec{v} \leftarrow$ **MeshOpticalFlow**(F_a , I^{st}_{-k} , I^{st}_{k} , ϵ , ε , L)
for $i \in [-k, k]$:
  $\overline{F}^{st}_i \leftarrow$ **FitARAP**($\overline{F}_{s+i}$, $\overline{F}_{t+i}$, R)
  $\overline{F}^{st}_i \leftarrow$ **Pose**($\overline{F}_{s+i}$, $\overline{F}_{t+i}$, $\overline{F}^{st}_i$, R , o)
  $\lambda_i \leftarrow (i + k)/2k$
  $I^{s}_i \leftarrow$ **Advect**(F_a , I^{st}_{-k} , $\lambda_i \cdot \vec{v}$)
  $I^{t}_i \leftarrow$ **Advect**(F_a , I^{st}_{k} , $-(1 - \lambda_i) \cdot \vec{v}$)
  $I^{st}_i \leftarrow I^{s}_i \cdot (1 - \lambda_i) + I^{t}_i \cdot \lambda_i$

6.1 AlignWindows

We solve for the optimal translation and rotation registering $\overline{F}_t$ to $\overline{F}_s$ by using Horn's method [Horn et al. 1988] to find the translation $o \in \mathbb{R}^2$ and rotation $R \in SO(2)$ that minimize the weighted sum of squared differences:

$$E(R, o) = \sum_i w_i \cdot m_i \cdot \left\| p^s_i - (R(p^t_i) + o) \right\|^2$$

with $\{p^s_i\}$ (respectively, $\{p^t_i\}$) the vertices in $\overline{F}_s$ (respectively, $\overline{F}_t$), m_i the mass of the i-th vertex (the sum of the volumes of the incident tetrahedra in $\overline{F}_s$ and $\overline{F}_t$), and w_i a weight that gives higher importance to vertices that are closer to the ground plane.

Biasing the registration towards the lower parts of the body helps avoid undesirable "gliding" motion and we set the weight as:

$$w_i = \exp\left(-\frac{\left(z^s_i + z^t_i\right)^2}{2\sigma^2_z} \right)$$

with z^s_i (respectively, z^t_i) the z-coordinate of the i-th vertex in $\overline{F}_s$ (respectively, $\overline{F}_t$) and σ_z is the rms height from the ground:

$$\sigma^2_z = \frac{\sum_i m_i \cdot \left(z^s_i + z^t_i\right)^2}{\sum_i m_i}.$$

6.2 ProjectTexture

When the calibrated camera images of the original scanned performance are available, we texture the frames at the boundary of the transition by projecting these images as in [Collet et al. 2015], where the contributions from non-occluded images are weighted by dot-product of camera orientations and surface normals. If the original camera images are not available, we can use nearest-point sampling to assign texture colors.

6.3 MeshOpticalFlow

To texture the frames in the interior of the transition we estimate the optical flow field $\vec{v}$ that takes the texture from the first frame in the source window to the texture in the last frame of the target window. Given such a flow field, we synthesize in-between textures by flowing the texture from the first frame of the source forward, the texture from the last frame backward, and blending.

We do this by extending the classical optical flow algorithm [Lucas and Kanade 1981; Tomasi and Kanade 1991; Shi and Tomasi 1994] from signals on images to signals on meshes. Computing optical flow directly on the surface offers a solution that is geometry-aware and agnostic to occlusions and texture seams.

For images, optical flow is often implemented over a multiresolution pyramid, using the lower resolutions to define the low-frequencies of the flow (that register the lower frequencies of the source and target images) and refining at successively higher resolutions.

As meshes do not carry a hierarchical structure, we adapt the algorithm by introducing a penalty term that encourages the flow to be smooth. Initially, the source and target signals are smoothed and the penalty term is given a high weight so that we focus on estimating the lower frequencies of the flow. The source and target are then advected along the estimated flow so that they are roughly aligned, and the process is repeated with less smoothing of the source and target and smaller smoothness penalty to allow for high-frequency correction.

Figure 7 compares results of linear blending (middle) and our optical flow (right) when synthesizing the texture for the middle frame of a 21-frame sequence. Due to the texture motion, linear blending exhibits significant ghosting artifacts in the face and shirt while optical flow better preserves the detail. Since the true texture is also captured (left), we can compare to it directly. Unfortunately, traditional similarity measures like RMS and SSIM fail to capture the improved detail preservation because the underlying L^2 distance is actually greater when detail is (even slightly) misregistered than when it is blurred. In contrast, comparing color histograms using the Earth Mover's Distance shows that optical flow better preserves the distribution of texel values, but this measure fails to incorporate

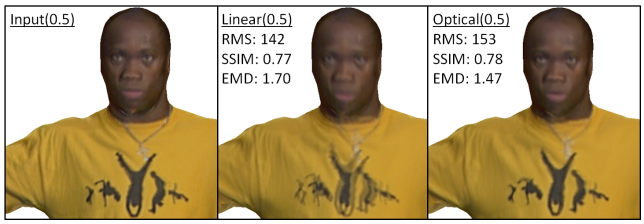

Figure 7: *Comparison of the in-between texture synthesized using linear blending (middle) and the texture synthesized using our optical flow (right) with the "ground-truth" acquired texture (left).*

spatial information. Finding a good measure of texture similarity is an interesting area for future work.

MeshOpticalFlow(F , I_0 , I_1 , ϵ , ε , L)

$\vec{v} \leftarrow 0$
for $l \in [0, L)$
  $\vec{v} \leftarrow \vec{v}$+**RefineFlow**(F , I_0 , I_1 , $\vec{v}$, $\epsilon/4^l$, $\varepsilon/4^l$)
return $\vec{v}$

RefineFlow At each hierarchy level, the vector-field $\vec{v}$ is refined by smoothing the input signals, flowing the smoothed signals halfway along $\vec{v}$ so that they are coarsely registered to each other, estimating the higher-resolution flow $\vec{\nu}$ that aligns the coarsely registered signals, and incorporating the refining flow $\vec{\nu}$ back into the vector field $\vec{v}$.

RefineFlow(F , I_0 , I_1 , $\vec{v}$, ϵ , ε)

$\tilde{I}_0 \leftarrow$ **Advect**(F , **SmoothSignal**(I_0 , ϵ) , $\vec{v}/2$)
$\tilde{I}_1 \leftarrow$ **Advect**(F , **SmoothSignal**(I_1 , ϵ) , $-\vec{v}/2$)
$\vec{\nu} \leftarrow$ **EstimateFlow**(F , $\tilde{I}_0$, $\tilde{I}_1$, ε)
$\lambda \leftarrow$ **GetScale**(F , $\tilde{I}_0$, $\tilde{I}_1$, $\vec{\nu}$)
return $\lambda \cdot \vec{\nu}$

SmoothSignal To smooth the signal I on surface F, we solve for the signal $\tilde{I}$ that minimizes the energy:

$$E(\tilde{s}) = \int_F \left[\left(\tilde{I}(p) - I(p) \right)^2 + \epsilon \cdot \|\nabla I(p)\|^2 \right] dp,$$

with the first term penalizing deviation from the input signal and the second encouraging smoothness.

Using the Euler-Lagrange formulation, the minimizer $\tilde{I}$ is the solution to the screened-Poisson equation:

$$(1 - \epsilon \cdot \Delta)\tilde{I} = I \tag{4}$$

where Δ is the Laplace-Beltrami operator on F.

Advect To flow a signal I along a vector field $\vec{w}$, we compute the streamlines of the vector field:

$$\gamma_p^{\vec{w}} : [0, 1] \to F$$

and set the value of the advected signal by following the flow line backwards and sampling:

$$\tilde{I}(p) = I\left(\gamma_p^{-\vec{w}}(1) \right). \tag{5}$$

EstimateFlow To refine the flow, we follow the optical flow implementation for images, seeking the flow $\vec{\nu}$ such that the change obtained by flowing I_0 along $\vec{\nu}$ explains the difference between I_1 and I_0. And, symmetrically, flowing I_1 along $-\vec{\nu}$ explains the difference between I_0 and I_1.

Using a first-order approximation, the change in the value of a signal I advected by a vector field $\vec{w}$ is given by the negative of the dot-product of I with the vector field: $-\langle \nabla I, \vec{w} \rangle$.

Thus, the correction vector field (approximately) satisfies:

$$-\langle \vec{\nu}, \nabla I_0 \rangle = I_1 - I_0 \qquad \text{and} \qquad \langle \vec{\nu}, \nabla I_1 \rangle = I_0 - I_1.$$

Setting $\delta(p) = I_0(p) - I_1(p)$ to be the difference between the signals, the flow is given as the minimizer of the energy:

$$E(\vec{\nu}) = \int_F \sum_{i=0}^{1} \left(\langle \nabla I_i(p), \vec{\nu}(p) \rangle - \delta(p) \right)^2 + \varepsilon \cdot \|\nabla \vec{\nu}(p)\|^2 \tag{6}$$

with the first term penalizing the failure of the vector field to explain the difference in signals, and the second encouraging smoothness of the flow.

Again, using the Euler-Lagrange formulation, the minimizer $\vec{\nu}$ is the solution to a linear system:

$$(\rho_0^* \cdot \rho_0 + \rho_1^* \cdot \rho_1 - \varepsilon \cdot \Delta)\vec{\nu} = (\rho_0^* + \rho_1^*)\delta \tag{7}$$

where ρ_i is the linear operator taking vector fields to scalar fields by evaluating the point-wise dot-product of the vector field with the gradient of I_i:

$$\rho_i(\vec{w})(p) = \langle \vec{w}(p), \nabla I_i(p) \rangle,$$

ρ_i^* is its adjoint, and Δ is the Hodge Laplacian.

GetScale A problem with using a soft smoothness constraint is that the penalty not only encourages the vector field to be smooth, but also small. This has the effect of dampening the flow. We correct the dampening by solving a 1D optimization problem to get the correct scale.

Fixing $\vec{\nu}$, we forgo the smoothness penalty from Equation (6) and define the energy of the scale λ as:

$$E(\lambda) = \int_F \sum_{i=0}^{1} \left(\langle \nabla I_i, \lambda \cdot \vec{\nu} \rangle - \delta \right)^2 dp.$$

Setting the derivative to zero, we get:

$$\lambda = \frac{\int_F \delta \langle \nabla I_0 + \nabla I_1, \vec{\nu} \rangle \, dp}{\int_F \langle \nabla I_0, \vec{\nu} \rangle^2 + \langle \nabla I_1, \vec{\nu} \rangle^2 \, dp}. \tag{8}$$

Discussion Though our work is the first to consider the problem of optical flow for signals on surfaces, we have recently discovered that a similar scale-space approach had been proposed for images by Alvarez et al. [1999]. The two approaches differ in several ways.

1. To extend optical flow to surfaces, we replace (linear) translation by the flow field $\vec{v}$ with (non-linear) exponentiation/advection in order to account for the curvature of the surface.
2. We formulate the regularization in terms of the Hodge Laplacian (rather than the Laplacians of the flow field coefficients) as there is no canonical coordinate frame for representing $\vec{v}$.
3. While Alvarez et al. use a fixed smoothing weight ε for all levels of the hierarchy, we reduce this weight at finer scales. (We have found that this generates better flows in fewer iterations.)

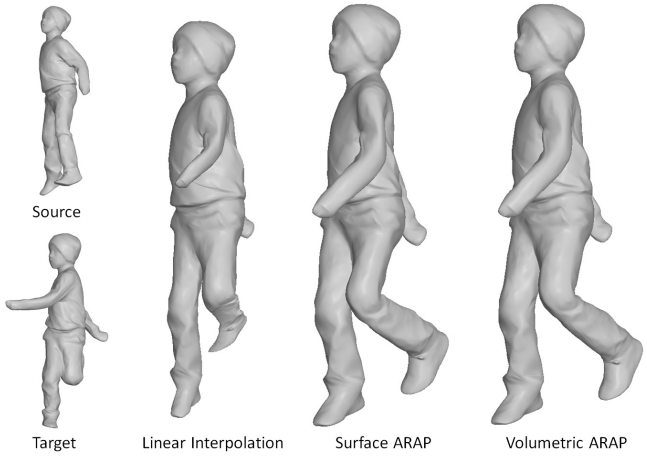

Figure 8: *Comparison of intermediate pose synthesis using linear interpolation, surface ARAP, and volumetric ARAP.*

4. To mitigate the shrinking of the flow field due to the smoothness constraint, we perform a subsequent scaling estimation/correction step, **GetScale**. In [Alvarez et al. 1999] the shrinking may be accounted for by using more refinement iterations.

5. To support boundary-aware flow fields, Alvarez et al. scale the smoothness term by a weight that decays with the size of the signal's gradient. We don't currently support this.

6.4 FitARAP

We synthesize the in-between frame by solving for the geometry that minimizes the weighted sum of as-rigid-as-possible energies between the new frame and the source and target. Figure 8 shows a visualization of the results of frame synthesis for frames F_s and F_t (first column). Using naive linear interpolation to synthesize the half-way mesh (second column) results in noticeable distortion and shrinking, but is a good initialization for the ARAP optimization. Surface ARAP (third column) provides good results even for dissimilar poses, but does not preserve the volume as well as volumetric ARAP. (See the left arm and thigh.)

Our intrinsic pose interpolation is obtained following the approach of Von-Tycowicz et al. [2015] – we minimize a weighted sum of ARAP energies (for each connected component). Given source and target frames, $\overline{F}_{s+i}$ and $\overline{F}_{t+i}$, we solve for the frame $\overline{F}_i^{st}$ minimizing:

$$E(\overline{F}_i^{st}) = (1-\lambda_i)\cdot E_{\text{ARAP}}(\overline{F}_i^{st}, \overline{F}_{s+i}) + \lambda_i \cdot E_{\text{ARAP}}(\overline{F}_i^{st}, \overline{F}_{t+i})$$

with $E_{\text{ARAP}}(\cdot,\cdot)$ the same volumetric ARAP energy as in Section 5.6 (but without locking boundary vertices this time), and λ_i the interpolation weight given by the cubic blending kernel:

$$\lambda_i = 3\left(\frac{i+k}{2k}\right)^2 - 2\left(\frac{i+k}{2k}\right)^3.$$

We use a cubic blending function rather than the simpler linear function because the vanishing derivative at the end-points $i = \pm k$ ensures that we not only interpolate the frames F_{s-k} and F_{t+k} but also the velocity of the animations at these frames. (See Figure 9.)

6.5 Pose

Given the source and target frames, $\overline{F}_{s+i}$ and $\overline{F}_{t+i}$, the result of the ARAP registration gives a transition frame $\overline{F}_i^{st}$ that is only defined up to rigid transformation. To pose the (connected component of the)

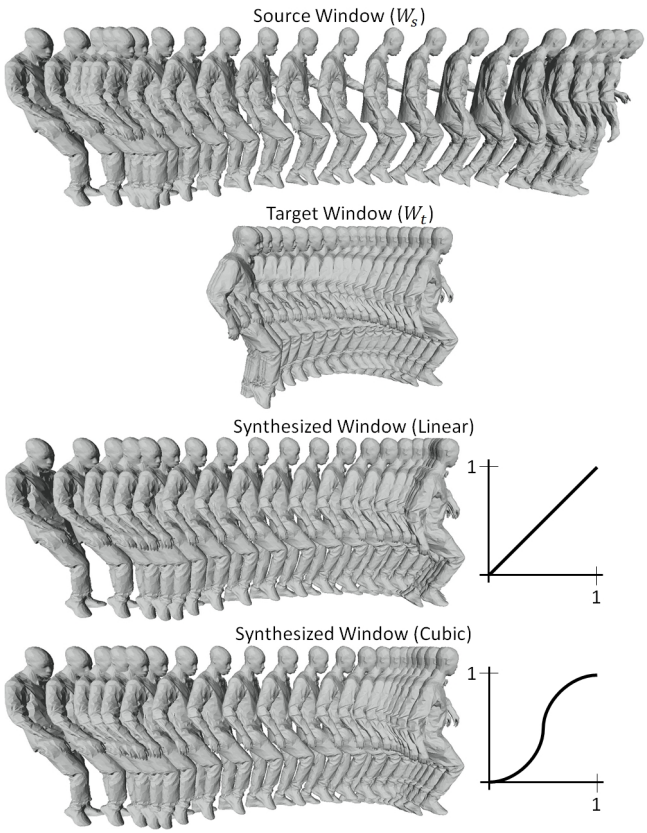

Figure 9: *Given spatially aligned transition windows W_s and W_t (first and second rows), we interpolate between corresponding frames to synthesize the frames transitioning from F_{s-k} to F_{t+k}. Using a linear blending function fails to preserve the velocity of the animation at the starting and ending frames (third rows). This is resolved by using a cubic blending kernel (bottom row).*

synthesized frame, we solve for the translation $\tilde{o} \in \mathbb{R}^3$ and rotation $\tilde{R} \in SO(3)$ that minimizes the weighted sum of registration errors:

$$\begin{aligned} E(\tilde{o},\tilde{R}) \quad = \quad & \sum_j m_j \Big[(1-\lambda_i) \left\| \left(\tilde{R}(p_j^{st}) + \tilde{o}\right) - p_j^{s+i} \right\|^2 \\ & + \lambda_i \left\| \left(\tilde{R}(p_j^{st}) + \tilde{o}\right) - \left(R(p_j^{t+i}) - o\right) \right\|^2 \Big] \end{aligned}$$

with p_j^{st} the j-th vertex of the synthesized frame $\overline{F}_i^{st}$, (o, R) the rigid registration from **AlignWindows** that best aligns the target to the source, and m_j the mass of the j-th vertex. We then apply the rigid transformation to the vertices of $\overline{F}_i^{st}$.

7 Motion graph traversal

Having constructed a motion graph by synthesizing transitions, we create new motions by traversing the graph, either stochastically or deterministically.

In either case, graph traversal requires updating the actor's world pose. As in [Kovar et al. 2002], we assign a 2D rigid transformation to each graph edge. For edges of the input sequence, this transformation is the identity. For edges corresponding to synthesized transitions, it is the rigid transformation computed for window alignment (Section 6.1). As we traverse each graph edge, the transform is applied to the actor's pose.

In the following discussion, we assume that the graph $\mathcal{G}$ is trimmed to its largest strongly connected component. Also, we reduce the complexity of the graph by removing all vertices whose indegree and outdegree are both one. Consequently, the number of graph edges becomes at least twice the number of graph vertices.

7.1 Random walk

For random traversal, we generalize the motion graph to a *Markov chain* $\mathcal{G} = (\mathcal{F}, \mathcal{E}, P)$ by defining a (sparse) transition probability matrix P which assigns to each directed edge $e = (F_i, F_j) \in \mathcal{E}$ a transition probability $p_{i,j}$. Outgoing probabilities at each node must sum to unity, i.e., $P\,\mathbf{1} = \mathbf{1}$.

We explore a new technique to assign transition probabilities P such that the steady-state distribution $\mathcal{S}(P)$ of the Markov Chain, i.e., the probability of being at any given node after a long random walk, is as close to uniform as possible. More precisely, $\mathcal{S}(P)$ is the unique left eigenvector of P with eigenvalue one [Levin et al. 2009].

In practice, $\mathcal{S}(P)$ can approximate a uniform distribution after activating just a few transitions in $\mathcal{G}$. For example, if there exists a transition from the last input frame to the first, enabling just this single transition makes the steady-state distribution uniform. To avoid these undesirable trivial solutions, we add the Frobenius norm $\|P\|_F$ as a regularization term. It is equivalent to the sum of L^2 norms of the rows of P and encourages uniformity of outgoing transition probabilities at all nodes.

Denoting the uniform distribution as $\pi = \mathbf{1}/|\mathcal{F}|$, our objective is

$$\begin{aligned} \min_{P} \quad & \alpha \cdot |\mathcal{F}| \cdot \|\mathcal{S}(P) - \pi\|^2 + (1-\alpha) \cdot \frac{|\mathcal{E}|}{|\mathcal{F}|^2} \cdot \|P\|_F^2 \\ \text{subject to} \quad & P\,\mathbf{1} = \mathbf{1}, \\ & p_{i,j} \geq 0, \\ & p_{i,j} = 0 \;\text{ if }\; (F_i, F_j) \notin \mathcal{E} \end{aligned}$$

with $|\mathcal{F}|$ and $|\mathcal{E}|/|\mathcal{F}|^2$ balancing the two penalty terms.

This is a constrained nonlinear objective whose derivatives are difficult to estimate accurately. We minimize it using the direct solver in the MATLAB `patternsearch` routine, using a maximum of 10^4 function evaluations and setting $\alpha = 0.08$ in all our examples. As a starting state for the nonlinear minimization, we first minimize the quadratic proxy energy

$$\alpha \cdot |\mathcal{F}| \cdot \|\pi P - \pi\|^2 + (1-\alpha) \cdot \frac{|\mathcal{E}|}{|\mathcal{F}|^2} \cdot \|P\|_F^2$$

subject to the same constraints. This energy is motivated by the fact that $\mathcal{S}(P)$ is the only distribution satisfying $\mathcal{S}(P)P = \mathcal{S}(P)$ on irreducible Markov chains [Levin et al. 2009].

Figure 10 shows an example result on the Breaker dataset, demonstrating that the use of optimized transition probabilities generates random walks that visit the input frames more uniformly.

One can assign the target distribution π in the optimization to be non-uniform, if it is desirable to make certain portions of the captured sequence occur less frequently as they would in a natural human performance. An interesting direction for future work would be to automatically learn such adapted distribution.

7.2 Shortest paths

For deterministic path generation, we implement a viewer program that supports interactive graph traversal. By default, it plays the input sequence. However, when the user selects a target frame, the program finds (and traverses) the shortest-path in the motion graph that goes from the current frame to the one prescribed. We refer the reader to the accompanying videos for a real-time screen-capture of a user interacting with the motion graph.

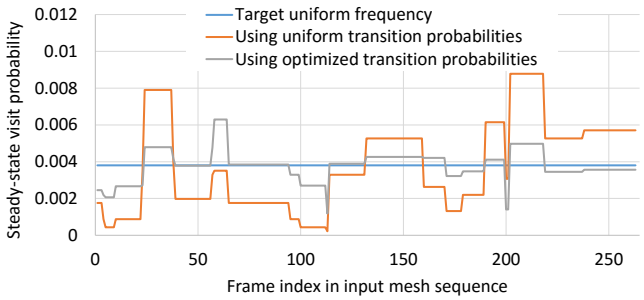

Figure 10: *Plot of probability distributions on the input frames of the Breaker dataset for random walks over its motion graph. The horizontal blue line shows the target uniform distribution. The orange curve shows the steady-state distribution using uniform transition probabilities. The gray curve shows the same for our optimized transition probabilities. The optimized probabilities lead to a much more uniform coverage of the input frames.*

8 Results and discussion

We have evaluated our approach on a number of datasets, ranging from short captures (∼200 frames) to longer ones (∼2000 frames). With the exception of Street Dancer and Dan, all the input sequences are unstructured. Parameter settings for the algorithms are provided in the appendix. These settings are identical across datasets.

Figure 11 visualizes a number of these datasets by showing a uniform sampling of frames from across the input sequence (indicated by the colored dots along the red line at the bottom). Despite the complexities and variability of the input, our approach successfully synthesizes a large number of transitions, providing rich motion graphs. In the images, the synthesized transitions are visualized by the black arcs connecting different points along the input sequence. (Note that the input sequence runs from left to right, while the synthesized transitions come in pairs.)

Datasets Example frames uniformly sampled from individual transitions are shown in Figure 12. Though not part of the input data, these transitions represent plausible complex actions (e.g.,bending, kicking, waving hands) and do not exhibit unnatural articulation in the geometry or unwanted ghosting in the texture. For a dynamic visualization of both the input and the synthesized transitions, we refer the reader to the accompanying video.

Table 1 summarizes for each capture sequence the number of frames in the input and the (largest strongly connected component) output, as well as the number of candidate windows, successful windows, and output windows. The Street Dancer and Dan captures are composed of several performances (6 and 10 respectively) with common parametrization. For these, we explicitly construct seamless transitions between boundaries of consecutive performances (Section 6); note that the presence of a common parametrization implies that any "candidate transition" is trivially successful as there is no need for window tracking.

Detailed running times Table 2 focuses on the Slick Magic sequence, factoring the computation time into the independent phases described in Sections 3-7. (Steps with negligible computation times are omitted.)

The Slick Magic input sequence consists of $n = 2283$ frames, each represented by a mesh with roughly 10K faces and a 1Mpixel texture atlas. Using our similarity-based pair selection, we identified $\tilde{l} = 53$

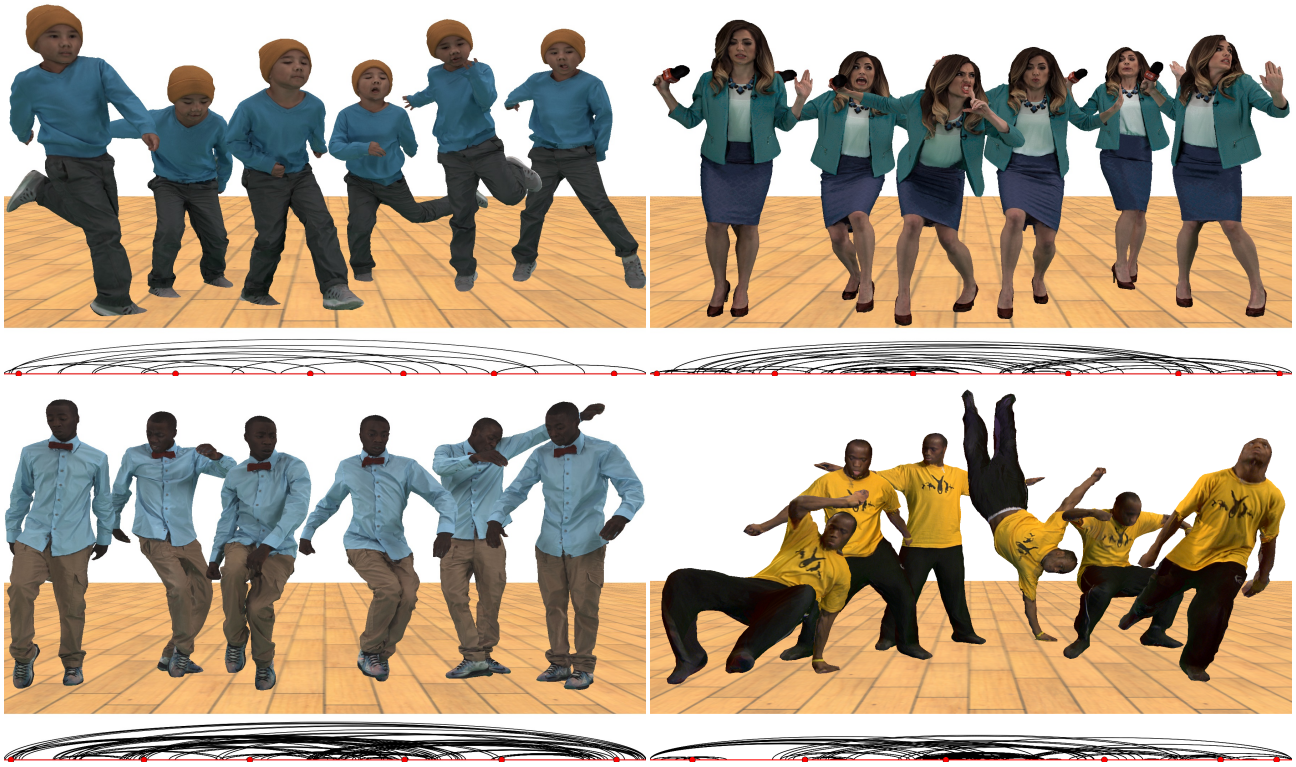

Figure 11: *Results of motion graph construction, showing a uniform sampling of the frames in the input sequences for Breakers, Reporter, Slick Magic, and Street Dancer. The images also show the synthesized transitions (represented by the arcs).*

Capture	Frames	Transitions
Ballerina	282 : 125	18 : 18 : 11
Breakers	443 : 253	42 : 16 : 13
Dan	587 : 563	54 : 54 : 54
Soccer Guy	593 : 430	86 : 76 : 64
Reporter	612 : 572	58 : 48 : 43
Girl	989 : 164	42 : 20 : 13
Street Dancer	1800 : 1637	100 : 100 : 83
Slick Magic	2283 : 2097	106 : 76 : 72

Table 1: *Frame size (input:output) and transition counts (candidate:successful:output) for all processed sequences. (Street Dancer and Dan sequence courtesy of [Starck and Hilton 2007]; all others courtesy of [Collet et al. 2015].)*

candidate transition pairs. We successfully tracked $l = 38$ of these pairs to get 76 forward and backward transitions, each of which was $2k + 1 = 21$ frames long. Extracting the largest strongly connected component, we obtained a final motion graph covering 2097 of the input frames and containing 72 synthesized transitions.

Running on a PC with a quad-core i7-5700HQ processor and 16GB of memory, it took roughly 10 minutes to identify candidate window pairs, 5 hours to fit a template to each of the window pairs, 3 hours to synthesize the transition frames, and one minute to compute the transition probabilities.

Identifying candidate window pairs requires computing shape descriptors for each of the n frames and then performing an $n \cdot (n-1)/2$ comparisons between the descriptors.

Tracking the windows requires computing a tetrahedralization of the template for each of the $\tilde{l}$ window pairs. Then the template surface is propagated to the other $\tilde{l} \times (4k + 1)$ frames. Having identified the l windows for which surface propagation succeeded,

(§4) Finding candidate transition windows	9
descriptor computation	<1
descriptor comparison	9
(§5) Tracking the windows consistently	295
tetrahedralizing the template	5
propagating the surface template	168 + 36
transferring geometric detail	38
propagating the volumetric template	37
(§6) Synthesizing seamless transitions	172
texture projection	29
optical flow computation	56
frame synthesis	86
(§7) Assigning transition probabilities	<1
Total time	478

Table 2: *Running times (in minutes) for the different phases of processing the Slick Magic sequence.*

the input detail is remapped to the fit template surface and the surface propagation is extended to the volume of each of the other $l \times (4k+1)$ frames. In the table, we give the surface propagation times for the l successful transitions (168 minutes) and the $\tilde{l} - l$ transitions failing the Hausdorff distance test (36 minutes) separately.

To synthesize the transition frames, we project the textures onto the two boundary templates of each of the $2l$ windows and estimate the optical flow for each of the $2l$ transitions. Then, for each of the $2l \times (2k - 1)$ non-boundary frames, the new frame geometry is synthesized and texture is advected from the boundary frames.

Finally, direct search is used to compute the probabilities for the maximally recurrent component of the graph.

Figure 12: *Samples of the frames synthesized for transitions in the Dan, Soccer Guy, and Girl datasets.*

Limitations In sequences with few repeated poses, it is difficult to construct a graph that covers a significant portion of the original capture. This happens for the Girl sequence whose largest strongly connected component only covers 164 of the original 989 frames.

For surfaces that have rotational symmetry (e.g., the ball in the Soccer Guy sequence) template tracking may not correct for rotation. As a result, the input textures may be too distant for optical flow.

Similarly, when texture detail appears and disappears (e.g., the wrinkles in the shirt of the Slick Magic sequence) optical flow cannot account for the change in color values, and ghosting may appear.

Our technique does not attempt to quantify physical realism in the synthesized transition geometry. In fact, there is no explicit check for surface intersections (i.e., collisions). In practice, we have not encountered problems with these issues, likely because the window-based similarity prunes poor candidate transitions.

9 Summary and future work

We have explored a new approach for constructing seamless motion graphs from time-varying, textured meshes. The key insight is to replace the problem of computing a template tracked across the entire sequence with the simpler problem of computing local templates tracked across pairs of short transition windows. The approach comprises four steps: (1) finding candidate transition windows; (2) imposing a *locally* consistent connectivity across the windows; (3) synthesizing new textured frames to transition between the windows; and (4) defining transition probabilities for the graph.

The resulting system takes as input a captured sequence of unstructured meshes and generates a motion graph that supports both stochastic and user-guided traversal. We validate our system on a number of captures, ranging in size from 200 to 2000 frames, showing successful synthesis of motion graphs that enable efficient and seamless transition between different distant frames.

There are several directions for improving the system:

- Shape similarity could incorporate texture colors to improve matching performance as in [Huang et al. 2015].
- For registering the fine-resolution input geometry with the tracked template, one could explore other ways to transfer the detail, e.g., using functional maps [Ovsjanikov et al. 2012].
- Mesh-based optical flow could be made more efficient at low frequencies by leveraging multiresolution geometry processing structures [e.g., Aksoylu et al. 2005; Chuang et al. 2009].
- Processing could be accelerated by automatically recognizing when volumetric ARAP can be replaced by faster surface ARAP, by parallelizing across frames, and by using a GPU-based surface tracking approach [e.g., Zollhöfer et al. 2014].

References

AKSOYLU, B., KHODAKOVSKY, A., and SCHRÖDER, P. 2005. Multilevel solvers for unstructured surface meshes. *SIAM Journal of Scientific Computing*, 26:1146–1165.

ALVAREZ, L., ESCLARÍN, J., LEFÉBURE, M., and SÁNCHEZ, J. 1999. A PDE model for computing the optical flow. In *XVI Congreso de Ecuaciones Diferenciales y Aplicaciones*.

ARIKAN, O. and FORSYTH, D. A. 2002. Interactive motion generation from examples. *ACM Trans. Graph.*, 21:483–490.

BOJSEN-HANSEN, M., LI, H., and WOJTAN, C. 2012. Tracking surfaces with evolving topology. *ACM Trans. Graph.*, 31.

CARRANZA, J., THEOBALT, C., MAGNOR, M. A., and SEIDEL, H.-P. 2003. Free-viewpoint video of human actors. *ACM Trans. Graph.*, 22.

CASAS, D., RICHARDT, C., COLLOMOSSE, J., THEOBALT, C., and HILTON, A. 2015. 4D model flow: precomputed appearance alignment for real-time 4D video interpolation. *Computer Graphics Forum*, 34(7).

CASAS, D., TEJERA, M., GUILLEMAUT, J.-Y., and HILTON, A. 2012. 4D parametric motion graphs for interactive animation. In *Symp. on Interactive 3D Graphics and Games*, pages 103–110.

CASAS, D., VOLINO, M., COLLOMOSSE, J., and HILTON, A. 2014. 4D video textures for interactive character appearance. *Comput. Graph. Forum*, 33.

CHAO, I., PINKALL, U., SANAN, P., and SCHRÖDER, P. 2010. A simple geometric model for elastic deformations. *ACM Trans. Graph.*, 29.

CHUANG, M., LUO, L., BROWN, B., RUSINKIEWICZ, S., and KAZHDAN, M. 2009. Estimating the Laplace-Beltrami operator by restricting 3D functions. *Symposium on Geometry Processing*.

COLLET, A., CHUANG, M., SWEENEY, P., GILLETT, D., EVSEEV, D., CALABRESE, D., HOPPE, H., KIRK, A., and SULLIVAN, S. 2015. High-quality streamable free-viewpoint video. *ACM Trans. Graph.*, 34.

DE GOES, F., DESBRUN, M., and TONG, Y. 2015. Vector field processing on triangle meshes. In *SIGGRAPH Asia 2015 Courses*, pages 17:1–17:48.

DZIUK, G. 1988. Finite elements for the Beltrami operator on arbitrary surfaces. In *Partial Differential Equations and Calculus of Variations, Lecture Notes in Mathematics*, volume 1357. Springer.

EISEMANN, M., DE DECKER, B., MAGNOR, M., BEKAERT, P., DE AGUIAR, E., AHMED, N., THEOBALT, C., and SELLENT, A. 2008. Floating textures. *Computer Graphics Forum*, 27(2).

FUNKHOUSER, T., KAZHDAN, M., SHILANE, P., MIN, P., KIEFER, W., TAL, A., RUSINKIEWICZ, S., and DOBKIN, D. 2004. Modeling by example. *ACM Trans. Graph.*, 23(3).

HECK, R. and GLEICHER, M. 2007. Parametric motion graphs. In *Symposium on Interactive 3D Graphics and Games*, ACM, pages 129–136.

HORN, B., HILDEN, H., and NEGAHDARIPOUR, S. 1988. Closed form solutions of absolute orientation using orthonormal matrices. *Journal of the Optical Society*, 5:1127–1135.

HUANG, C.-H., BOYER, E., NAVAB, N., and ILIC, S. 2014. Human shape and pose tracking using keyframes. In *Proc. CVPR*.

HUANG, P., HILTON, A., and STARCK, J. 2009. Human motion synthesis from 3D video. In *Proc. CVPR*.

HUANG, P., HILTON, A., and STARCK, J. 2010. Shape similarity for 3D video sequences of people. *International Journal of Computer Vision*, 89:362–381.

HUANG, P., TEJERA, M., COLLOMOSSE, J., and HILTON, A. 2015. Hybrid skeletal-surface motion graphs for character animation from 4D performance capture. *ACM Trans. Graph.*, 34:17.

JACOBSON, A., PANOZZO, D., and OTHERS. 2016. libigl: A simple C++ geometry processing library. http://libigl.github.io/libigl/.

KANADE, T., RANDER, P., and NARAYANAN, P. J. 1997. Virtualized reality: Constructing virtual worlds from real scenes. *IEEE Multimedia*, 4.

KAZHDAN, M. 2013. ShapeSPH. http://www.cs.jhu.edu/~misha/Code/ShapeSPH/ShapeAlign/.

KAZHDAN, M., FUNKHOUSER, T., and RUSINKIEWICZ, S. 2003. Rotation invariant spherical harmonic representation of 3D shape descriptors. In *Symposium on Geometry Processing*.

KOVAR, L., GLEICHER, M., and PIGHIN, F. 2002. Motion graphs. *ACM Trans. Graph.*, 21:473–482.

LEE, J., CHAI, J., REITSMA, P. S., HODGINS, J. K., and POLLARD, N. S. 2002. Interactive control of avatars animated with human motion data. *ACM Trans. Graph.*, 21:491–500.

LEVIN, D. A., PERES, Y., and WILMER, E. L. 2009. *Markov chains and mixing times*. American Mathematical Society.

LI, H., ADAMS, B., GUIBAS, L. J., and PAULY, M. 2009. Robust single-view geometry and motion reconstruction. *ACM Trans. Graph.*, 28.

LUCAS, B. and KANADE, T. 1981. An iterative image registration technique with an application to stereo vision. In *Proc. Intnl. Joint Conf. on Artificial Intelligence*, pages 674–679.

OVSJANIKOV, M., BEN-CHEN, M., SOLOMON, J., BUTSCHER, A., and GUIBAS, L. 2012. Functional maps: A flexible representation of maps between shapes. *ACM Trans. Graph.*, 31.

PINKALL, U. and POLTHIER, K. 1993. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics*, 2.

PRADA, F. and KAZHDAN, M. 2015. Unconditionally stable shock filters for image and geometry processing. *Computer Graphics Forum*, 34:201–210.

RUSINKIEWICZ, S. 2004. Trimesh v2. http://gfx.cs.princeton.edu/proj/trimesh2/.

SHI, J. and TOMASI, C. 1994. Good features to track. In *Proc. CVPR*, pages 593–600.

SHILANE, P., MIN, P., KAZHDAN, M., and FUNKHOUSER, T. 2004. The Princeton shape benchmark. In *Shape Modeling International*.

SI, H. 2015. TetGen, a Delaunay-based quality tetrahedral mesh generator. *ACM Trans. on Math. Software*, 41.

SORKINE, O. and ALEXA, M. 2007. As-rigid-as-possible surface modeling. In *Symp. on Geometry Processing*, pages 109–116.

STARCK, J. and HILTON, A. 2007. CVSSP3D datasets. http://cvssp.org/data/cvssp3d/.

STARCK, J., MILLER, G., and HILTON, A. 2005. Video-based character animation. In *Symposium on Computer animation*.

TANGELDER, J. and VELTKAMP, R. 2007. A survey of content based 3D shape retrieval methods. *Multimedia Tools and Applications*, 39:441–471.

TOMASI, C. and KANADE, T. 1991. Detection and tracking of point features. Technical Report CMU-CS-91-132.

VLASIC, D., PEERS, P., BARAN, I., DEBEVEC, P., POPOVIĆ, J., RUSINKIEWICZ, S., and MATUSIK, W. 2009. Dynamic shape capture using multi-view photometric stereo. *ACM Trans. Graph.*, 28.

VOLINO, M., HUANG, P., and HILTON, A. 2015. Online interactive 4D character animation. In *ACM SIGGRAPH Web3D Conference*.

VON-TYCOWICZ, C., SCHULZ, C., SEIDEL, H.-P., and HILDEBRANDT, K. 2015. Real-time nonlinear shape interpolation. *ACM Trans. Graph.*, 34.

XU, F., LIU, Y., STOLL, C., TOMPKIN, J., BHARAJ, G., DAI, Q., SEIDEL, H.-P., KAUTZ, J., and THEOBALT, C. 2011. Video-based characters: Creating new human performances from a multi-view video database. *ACM Trans. Graph.*, 30(4).

ZITNICK, C. L., KANG, S. B., UYTTENDAELE, M., WINDER, S., and SZELISKI, R. 2004. High-quality video view interpolation using a layered representation. *ACM Trans. Graph.*, 23.

ZOLLHÖFER, M., NIESSNER, M., IZADI, S., REHMANN, C., ZACH, C., FISHER, M., WU, C., FITZGIBBON, A., LOOP, C., THEOBALT, C., and STAMMINGER, M. 2014. Real-time non-rigid reconstruction using an RGB-D camera. *ACM Trans. Graph.*, 33.

A Implementation details

In this appendix we provide implementation details for various steps of the geometry processing. First we introduce some notation.

Linear systems We represent matrices by bold, upper-case Roman characters (e.g. $\mathbf{M} \in \mathbb{R}^{M \times N}$) and vectors by lower-case Roman characters (e.g. $\mathbf{v} \in \mathbb{R}^N$). We represent the coefficients of matrices and vectors with lower-case Roman characters (e.g. $\mathrm{m}_{ij} \in \mathbb{R}$ and $\mathrm{v}_i \in \mathbb{R}$).

Given $\mathbf{v} \in \mathbb{R}^{dN}$, we can think of $\mathbf{v}$ as an N-dimensional vector whose coefficients are themselves d-dimensional column vectors. Similarly, given $\mathbf{M} \in \mathbb{R}^{dN \times N}$, we can think of $\mathbf{M}$ as an $N \times N$-dimensional matrix whose coefficients are d-dimensional column vectors. Using this abuse of notation, we denote by $\mathrm{diag}_d(\mathbf{v})$ the $N \times N$ diagonal matrix whose diagonal entries are the d-dimensional coefficients of $\mathbf{v}$.

Given matrices $\mathbf{M}_1$ and $\mathbf{M}_2$, we denote by $\mathbf{M}_1 \oplus \mathbf{M}_2$ the matrix with $\mathbf{M}_1$ and $\mathbf{M}_2$ along the diagonal:

$$\mathbf{M}_1 \oplus \mathbf{M}_2 = \begin{pmatrix} \mathbf{M}_1 & 0 \\ 0 & \mathbf{M}_2 \end{pmatrix}.$$

Given a matrix $\mathbf{M}$ and integer k, we denote by $\mathbf{M}^{\oplus k}$ the block diagonal matrix with k blocks, $\mathbf{M}^{\oplus k} = \mathbf{M} \oplus \cdots \oplus \mathbf{M}$.

Given matrices $\mathbf{M}_1$ and $\mathbf{M}_2$ (with the same number of rows), we denote by $\mathbf{M}_1 \big| \mathbf{M}_2$ the (horizontal) concatenation:

$$\mathbf{M}_1 \big| \mathbf{M}_2 = \begin{pmatrix} \mathbf{M}_1 & \mathbf{M}_2 \end{pmatrix}.$$

Geometry We assume that we are given a triangle mesh $\mathcal{M}$, and denote by $(\mathcal{V}, \mathcal{T})$ the vertices and triangles of the mesh.

Discrete functions are represented using the hat basis as elements of $\mathbb{R}^{|\mathcal{V}|}$ [Dziuk 1988] and vector fields are represented using the conforming basis of gradients and rotated gradients as elements of $\mathbb{R}^{2|\mathcal{V}|}$ [de Goes et al. 2015].

We denote by $\mathbf{M} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ and $\mathbf{S} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ the (lumped) mass and stiffness matrices. The former is the diagonal matrix with one third of the area of a vertex's one-ring on the diagonal, and the latter is the ubiquitous cotangent Laplacian [Dziuk 1988; Pinkall and Polthier 1993]. And we denote by $\mathbf{H}$ the Hodge-Laplacian for vector fields, $\mathbf{H} = \left(\mathbf{S} \cdot \mathbf{M}^{-1} \cdot \mathbf{S}\right)^{\oplus 2}$.

Given an orthonormal tangent frame at each triangle, we denote by $\mathbf{J} \in \mathbb{R}^{2|\mathcal{T}| \times 2|\mathcal{T}|}$ the operator that rotates each triangle's tangent vector by 90°, we denote by $\mathbf{G} \in \mathbb{R}^{|\mathcal{V}| \times 2|\mathcal{T}|}$ the operator that returns the gradient of a scalar function, and we denote by $\mathbf{B} \in \mathbb{R}^{2|\mathcal{T}| \times 2|\mathcal{V}|}$ the matrix that takes a vector field expressed in terms of the conforming basis and returns the coefficients with respect to the per-triangle tangent frame, $\mathbf{B} = \mathbf{G} \big| \mathbf{J} \circ \mathbf{G}$.

A.1 Shape matching

To compute the similarity matrix $\mathbf{D}$ we proceed in two steps.

Per-frame computation We use ShapeSPH [Kazhdan 2013] to compute the discrete indicator and Euclidean Distance Transforms functions for each shape, represented as a grid of $2N \times N/2 \times N$ samples in cylindrical coordinates, with indices corresponding to angle, radius, and height. We then compute the 1D FFT across each row of angles to get a representation of frame F_i by the complex Fourier coefficients of the indicator and Euclidean Distance Transform:

$$\widehat{\mathbf{X}}^i, \widehat{\mathbf{E}}^i \in \mathbb{C}^{2N \times N/2 \times N}.$$

This computation is performed for each of the n frames and has complexity $O(n \cdot N^3 \log N)$.

Comparing pairs of frames Given the Fourier coefficients, we compute the similarity between frames s and t by correlation, multiplying the Fourier coefficients into the vector $\widehat{\mathbf{c}}^{st}$:[1]

$$\widehat{\mathrm{c}}_i^{st} = \sum_{l=-k}^{k} \omega_k \cdot \sum_{m=1}^{N/2} m \cdot \sum_{n=1}^{N} \left(\widehat{\mathrm{x}}_{imn}^{s+k} \cdot \overline{\widehat{\mathrm{e}}_{imn}^{t+k}} + \widehat{\mathrm{e}}_{imn}^{s+k} \cdot \overline{\widehat{\mathrm{x}}_{imn}^{t+k}} \right),$$

taking the inverse Fourier Transform to get the correlation values in the spatial domain, and then finding the minimal correlation value:

$$\mathrm{d}_{st} = \min_{i \in [0, 2N)} \mathrm{c}_i^{st}.$$

This computation is performed for each of the n^2 pairs of frames and has complexity $O(n^2 \cdot N^3)$.

Parameters To compute distances, we use $N = 64$ for the resolution of the voxel grid and set ω to be the fifth-order B-spline. To define candidate transitions, we discard transition (s, t) if:

- The frames are too close to each other: $|s - t| < 60$.
- The candidate transition is too close to an existing transition $(\tilde{s}, \tilde{t})$: $|s - \tilde{s}| < 20 \quad \text{and} \quad |t - \tilde{t}| < 20$.
- The dissimilarity is too large: $d_{st} > 3 \left(\frac{\sum_i \mathrm{d}_{i,i+1}}{i-1} \right)$.

A.2 As-rigid-as-possible registration

To compute the volumetric ARAP energy we use the tetrahedron-based implementation of libigl [Jacobson et al. 2016]. Given two tetrahedral meshes $(\mathcal{V}_1, \mathcal{K})$ and $(\mathcal{V}_2, \mathcal{K})$ with the same tessellation $\mathcal{K}$, the ARAP energy is given by

$$E(\mathcal{K}, \mathcal{V}_1, \mathcal{V}_2) = \sum_{\sigma \in \mathcal{K}} |\sigma|_1 \cdot \min_{R \in SO(3)} \|R - L_\sigma\|_F^2$$

with L_σ the linear component of the affine map taking the embedding of σ through $\mathcal{V}_1$ to its embedding through $\mathcal{V}_2$, $|\sigma|_1$ the volume of σ under the embedding of $\mathcal{V}_1$, and $\| \cdot \|_F^2$ the squared Frobenius norm.

We minimize this nonlinear energy by running 20 iterations of the local/global approach of Sorkine and Alexa [2007], alternating between solving for the optimal rotations and solving for the optimal vertex positions.

When performing the tracking (Section 5.6) we initialize the solver with the vertex positions of the tracked frame when the two frames are in the same window. When the frames are in different windows (e.g., tracking from F_s to F_t or vice-versa) we apply the rigid transformation whose translational component aligns the centers of mass and rotational component minimizes the dissimilarity:

$$D_{st}(R) = \langle \mathrm{EDT}_s^2, R(\chi_t) \rangle + \langle \chi_s, R(\mathrm{EDT}_t^2) \rangle$$

from Section 4. When synthesizing the geometry of in-between frames (Section 6.4) we use the weighted linear blend of the vertex positions in the source and aligned target as the initial guess.

A.3 Detail transfer

We assume that we have source and target meshes $\mathcal{M}_\alpha = (\mathcal{V}_\alpha, \mathcal{T}_\alpha)$, with $\alpha \in \{0, 1\}$. We denote the vertex positions and normals by the $3|\mathcal{V}_\alpha|$-dimensional vectors of (linearized) coefficients, $\mathbf{v}_\alpha$ and $\mathbf{n}_\alpha$.

[1] The multiplication by the radius m is required to account for the circumference of the circle.

Normal smoothing (Equation 1) Given the mesh normals, we compute the (linearized) tangent vectors, $\mathbf{t}_1, \mathbf{t}_2, \in \mathbb{R}^{3|\mathcal{V}|}$, where the i-th tangents are perpendicular to the i-th normal. We set $\mathbf{T} = \text{diag}_3(\mathbf{t}_1)|\text{diag}_3(\mathbf{t}_2)$ to be the operator that takes tangent coefficients and returns the (per-vertex) tangent vectors.

The tangential offsets $\mathbf{o}$ with respect to this frame are given by the solution to the $2|\mathcal{T}| \times 2|\mathcal{T}|$ linear system:

$$\left(\mathbf{T}^t \cdot (\mathbf{M} + \varepsilon \cdot \mathbf{S})^{\oplus 3} \cdot \mathbf{T}\right) \mathbf{o} = -\varepsilon \cdot (\mathbf{T}^t \cdot \mathbf{S}^{\oplus 3}) \cdot \mathbf{n}.$$

And, we define the new normal positions by offsetting the original normals in the tangent direction, $\tilde{\mathbf{n}} = \mathbf{n} + \mathbf{T} \cdot \mathbf{o}$, and normalizing so that the normals have unit length.

We note that though the implementation requires choosing tangents at each vertex, the smoothed normals are independent of this choice.

Offset estimation (Equation 2) Given the smoothed normals, $\tilde{\mathbf{n}}$, the offsets $\mathbf{h}$ are given by the solution to the $|\mathcal{V}| \times |\mathcal{V}|$ linear system:

$$\left(\text{diag}_3^t(\tilde{\mathbf{n}}) \cdot (\mathbf{M} + \varepsilon \cdot \mathbf{S})^{\oplus 3} \cdot \text{diag}_3(\tilde{\mathbf{n}})\right) \mathbf{h} = -\varepsilon \cdot (\text{diag}_3^t(\tilde{\mathbf{n}}) \cdot \mathbf{S}) \cdot \mathbf{v}.$$

Then, setting $\mathbf{1} \in \mathbb{R}^{|\mathcal{V}|}$ to be the vector whose entries are all 1, we adjust $\mathbf{h}$ to have zero mean by setting:

$$\mathbf{h} \leftarrow \mathbf{h} - \frac{\mathbf{1}^t \cdot \mathbf{M} \cdot \mathbf{h}}{\mathbf{1}^t \cdot \mathbf{M} \cdot \mathbf{1}} \cdot \mathbf{1}.$$

Registration (Equation 3) We define the pull-back of the registration as the matrix $\mathbf{P} \in \mathbb{R}^{|\mathcal{V}_0| \times |\mathcal{V}_1|}$ whose i-th row entries are defined by:

- Finding the point on $\mathcal{M}_1$ closest to the i-th vertex of $\mathcal{M}_0$.
- Computing the barycentric coordinates of the point relative to the vertices of the containing triangle.
- Setting the entries corresponding to these vertices to the barycentric weights (and all other row entries to zero).

For computing the closest point, we use the TriMesh kD-tree implementation [Rusinkiewicz 2004].

Parameters In implementing the detail transfer, we use $\epsilon = 10^{-3}$ for the normal smoothing weight, $\varepsilon = 10^{-4}$ for the offset smoothing weight, and $L = 3$ for the number of hierarchy levels.

A.4 Optical flow

We assume that we have a single mesh $\mathcal{M}$ with source and target signals $\mathbf{s}_0$ and $\mathbf{s}_1$. For simplicity, the discussion will assume that the signals are single-channel, though the extension to multi-channel signals is straightforward.

Preprocess We begin the optical flow pipeline by transforming the texture map into a signal on the mesh. This is done by setting the value at each vertex to the average value of the texture map on the vertex's one-ring. In practice, we also subdivide the mesh by splitting edges longer than a threshold and re-triangulating in order to ensure that the signal captures sufficient texture detail.

Runtime Our implementation of the optical flow consists of four steps: Smoothing, advection, flow estimation, and scale adjustment.

Signal smoothing (Equation 4) The smoothed signal $\tilde{\mathbf{s}}$ is given as the solution to the $|\mathcal{V}| \times |\mathcal{V}|$ linear system

$$(\mathbf{M} + \epsilon \cdot \mathbf{S})\tilde{\mathbf{s}} = \mathbf{M} \cdot \mathbf{s}.$$

Advection (Equation 5) Given a point $p \in \mathcal{M}$, we evaluate the scalar field $\mathbf{s}$ advected along $\vec{\mathbf{v}}$ for time t by taking N geodesic steps along the mesh. At each step we evaluate the vector field $\vec{\mathbf{v}}$ at p (determined by the triangle containing p) and update the position by taking a unit steps backwards along the geodesic from p in direction $-\vec{\mathbf{v}}(p)/N$. This is implemented by following the straight line defined by the tangent and unfolding adjacent faces if a triangle edge is reached. (See, for example, the implementation in [Prada and Kazhdan 2015].) Finally, we set the value of the advected signal by sampling the input signal at the new position p.

When the p is a vertex, we offset p by a small amount into each adjacent triangle and proceed as above, using the average over adjacent triangles to set the value of the advected scalar field at p.

Flow estimation (Equation 7) To discretize the flow estimation, we define the following auxiliary operators:

- We set $\mathbf{M}_T \in \mathbb{R}^{|\mathcal{T}| \times |\mathcal{T}|}$ to be the *triangle* mass matrix – the diagonal matrix whose i-th entry is the area of the i-th triangle.
- We set $\mathbf{D}_\alpha \in \mathbb{R}^{|\mathcal{T}| \times 2|\mathcal{V}|}$ to be the matrix which takes a vector field and returns the dot-product with the gradient of $\mathbf{s}_\alpha$, evaluated at each triangle:

$$\mathbf{D}_\alpha = \left(\text{diag}_2\left(\mathbf{G}(\mathbf{s}_\alpha)\right)\right)^t \cdot \mathbf{B}.$$

We also set $\mathbf{d} \in \mathbb{R}^{|\mathcal{T}|}$ to be the per-triangle difference between the source and target, obtained by computing the difference of the average values of $\mathbf{s}_0$ and $\mathbf{s}_1$ at the vertices.

Using these, the discretization of Equation (7) gives the correction flow as the solution to the $2|\mathcal{V}| \times 2|\mathcal{V}|$ linear system:

$$\left(\sum_{\alpha \in \{0,1\}} (\mathbf{D}_\alpha^t \cdot \mathbf{M}_T \cdot \mathbf{D}_\alpha) - \varepsilon \cdot \mathbf{H}\right) \vec{\mathbf{v}} = \left(\sum_{\alpha \in \{0,1\}} \mathbf{D}_\alpha^t \cdot \mathbf{M}_t\right) \mathbf{d}.$$

Scale adjustment (Equation 8) Using the notation above, the optimal scale is given as:

$$\lambda = \frac{\mathbf{d}^t \cdot \left(\sum_{\alpha \in \{0,1\}} \mathbf{M}_T \cdot \mathbf{D}_\alpha\right) \cdot \vec{\mathbf{v}}}{\vec{\mathbf{v}}^t \cdot \left(\sum_{\alpha \in \{0,1\}} \mathbf{D}_\alpha^t \cdot \mathbf{M}_T \cdot \mathbf{D}_\alpha\right) \cdot \vec{\mathbf{v}}}.$$

Postprocess After estimating the flow field on the mesh we use it to advect the texture: For each texel center, we locate its corresponding position within a triangle of the mesh, follow the stream-line defined by the flow, and sample the input texture at the advected position. We then set the value at the starting texel position to the sampled value.

Parameters In implementing the optical flow we use $\epsilon = 3 \cdot 10^{-3}$ for the signal smoothing weight, $\varepsilon = 10^6$ for the offset smoothing weight, and $L = 7$ for the number of hierarchy levels.

Note A limitation of using this conforming basis is that for genus-g surfaces, the span of the basis does not include the harmonic vector fields – the $2g$-dimensional space of curl- and divergence-free vector fields. We do not find this to be a problem for our applications as our datasets consist of human characters which are genus-0. However, extending the approach to surfaces of arbitrary genus would require either explicitly estimating a basis for the harmonic vector fields and incorporating that within the system, or using a different (e.g. edge-based) discretization of vector fields.