Spatiotemporal Atlas Parameterization for Evolving Meshes

FABIÁN PRADA and MISHA KAZHDAN, Johns Hopkins University
MING CHUANG*, ALVARO COLLET†, and HUGUES HOPPE‡, Microsoft Corporation

We convert a sequence of unstructured textured meshes into a mesh with incrementally changing connectivity and atlas parameterization. Like prior work on surface tracking, we seek temporally coherent mesh connectivity to enable efficient representation of surface geometry and texture. Like recent work on evolving meshes, we pursue local remeshing to permit tracking over long sequences containing significant deformations or topological changes. Our main contribution is to show that both goals are realizable within a common framework that simultaneously evolves both the set of mesh triangles and the parametric map. Sparsifying the remeshing operations allows the formation of large spatiotemporal texture charts. These charts are packed as prisms into a 3D atlas for a texture video. Reducing tracking drift using mesh-based optical flow helps improve compression of the resulting video stream.

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

Additional Key Words and Phrases: textured mesh sequences, surface tracking, remeshing, spatiotemporal packing, texture video atlas

ACM Reference format:
Fabián Prada, Misha Kazhdan, Ming Chuang, Alvaro Collet, and Hugues Hoppe. 2017. Spatiotemporal Atlas Parameterization for Evolving Meshes. *ACM Trans. Graph.* 36, 4, Article 58 (July 2017), 12 pages.
https://doi.org/10.1145/3072959.3073679

1 INTRODUCTION

Several techniques capture a performance for interactive viewing within a virtual environment [e.g., Collet et al. 2015; Dou et al. 2016; Newcombe et al. 2015; Starck and Hilton 2007; Zitnick et al. 2004]. This quickly advancing area is variously referred to as volumetric video, VR holograms, or free-viewpoint video. One commonly used representation is a *temporal sequence of colored meshes*. Each mesh consists of a set of triangles with shared vertex positions. Surface color is specified using per-vertex attributes, projected camera images, or a parameterized texture atlas image.

The most flexible approach is to store an independent mesh for each time frame, as this supports arbitrarily varying scenes with topological changes and large deformations. Creating an unstructured sequence of meshes is natural for many surface reconstruction techniques. However, the flexibility comes with significant runtime cost. Each mesh must be encoded separately due to its unique connectivity. The set of vertices and/or texture parameterization also changes each frame, so one cannot easily exploit temporal coherence of surface appearance. In short, *unstructured meshes* lead to high data bandwidth and memory use.

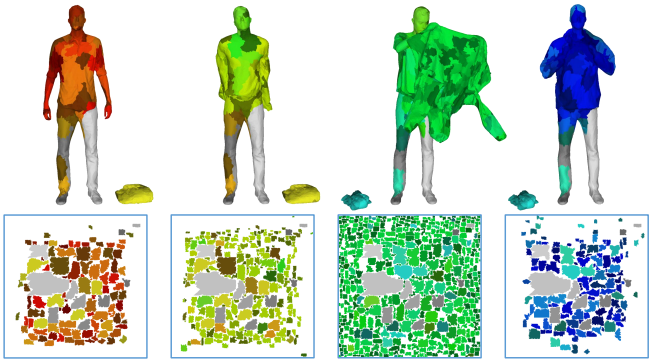


Fig. 1. Four frames of the "clothing" sequence and their 2D texture slices within the 3D atlas parameterization. Hue and saturation visualize the midpoint and length of each chart's lifespan, from red (early) to blue (late), with saturated colors for short lifespans.

Instead, many approaches perform inter-frame tracking, whereby the mesh deforms geometrically but retains the same connectivity (and parameterization) over consecutive frames. Often, mesh tracking is successively applied over dozens of frames, until some triangles become overly deformed or topological changes occur. Collet et al. [2015] explore finding an optimal subset of *keyframes* at which to reinitialize the mesh connectivity. Within each keyframe subsequence, the shared mesh connectivity facilitates geometry compression [Yu et al. 2011] and the shared parameterization lets texture images be encoded efficiently as a video stream.

Bojsen-Hansen et al. [2012] make the important observation that mesh tracking failures are often spatially localized. Their approach is to identify erroneous regions and apply *local remeshing*, thereby creating *evolving meshes*. They also establish surface correspondences across remeshing operations to propagate attributes like surface albedo over mesh vertices during animation. However, their scheme does not support a temporally coherent surface parameterization.

Our main contribution is to enable local remeshing in conjunction with a texture atlas parameterization (Figure 1). Use of a texture provides efficient storage of high-resolution surface detail and allows inter-frame compression using hardware-accelerated video coding schemes like MPEG-4. The key challenge relative to [Bojsen-Hansen et al. 2012] is to spatiotemporally sparsify the remesh operations to allow the creation of large texture charts that persist across many frames. Naturally, the approach performs best when remeshing is minimized. To this end, we present techniques to both improve mesh tracking and reduce the remeshing extent.

The next step is to construct a set of parametric charts over the evolving mesh. These charts correspond to (right) prisms in the spatiotemporal domain of the texture video. We present

*Now at Apple Inc. †Now at Facebook Inc. ‡Now at Google Inc.

This work is supported by the NSF award 1422325.

0730-0301/2017/7-ART58 $15.00
https://doi.org/10.1145/3072959.3073679

combinatorial algorithms to both define these prisms and pack them into the texture domain. This 3D spatiotemporal packing can be seen as a generalization of traditional 2D texture atlas packing.

In summary, our contributions are:

- temporally coherent atlas parameterization of evolving meshes,
- improved geometric tracking to reduce the need for remeshing,
- reduction in tracking drift using mesh-based optical flow,
- new remeshing strategy that localizes mesh modifications, and
- efficient 3D spatiotemporal packing of parameterized charts.

2 RELATED WORK

Mesh parameterization. Many techniques partition a static mesh into charts, flatten the charts, and pack them into a 2D texture atlas [e.g., Lévy et al. 2002; Sander et al. 2001; Sheffer et al. 2006]. When a fixed connectivity template is used to track a sequence of animation frames, the same parameterization can be used for all frames. We are not aware of any work that builds persistent atlas parameterizations for meshes with temporally evolving connectivity.

Mesh tracking and deformation. Several techniques fit scanned 3D data by deforming a mesh template over successive frames while maintaining fixed mesh connectivity [e.g., Budd et al. 2013; Li et al. 2009, 2008; Starck and Hilton 2007; Zollhöfer et al. 2014]. The deformation process optimizes both the fit to the input data and some regularizing deformation energy, e.g., the as-rigid-as-possible (ARAP) metric evaluated over either a coarse deformation graph [Sumner et al. 2007] or fine-scale mesh neighborhoods [Sorkine and Alexa 2007]. Like [Bojsen-Hansen et al. 2012], we successively apply both coarse- and fine-scale tracking.

Tevs et al. [2012] present a sparse-to-dense correspondence approach for synthesizing a mesh template from an input sequence. They track correspondences across non-consecutive frames to allow fill-in of missing data (due to, for example, occlusion).

Few existing tracking techniques exploit the surface color field. Li et al. [2012] use SURF descriptors and image-based optical flow to track features in recorded videos and thereby set sparse temporal point correspondences. Our approach builds dense correspondence maps on the mesh surfaces using mesh-based optical flow [Prada et al. 2016] to iteratively refine mesh tracking.

Surface correspondence. A closely related problem is to compute correspondence maps between surfaces or point clouds [Kim et al. 2011; Maron et al. 2016; Solomon et al. 2016]. These global methods could improve tracking robustness but are expensive to compute.

Evolving meshes. Wojtan et al. [2009] and Bojsen-Hansen et al. [2012] apply local remeshing operations (edge collapses, splits, and swaps) to maintain well-shaped triangles as a mesh deforms. They use a voxel grid to identify regions where the deformed mesh fails to align with the target. Specifically, they compute isosurfaces from the signed-distance fields of the two meshes and examine the topological consistency of the isosurfaces in the voxel grid. We relax the constraint on local topological consistency between source and target. Instead we identify surface points with significant geometric error and grow remesh regions about those points. We use a voxel grid both to grow each remesh region and to stitch its boundary.

Bojsen-Hansen et al. [2012] associate surface attributes with mesh vertices and propagate these attributes temporally through point correspondences across remeshing operations. In contrast, we store surface attributes using a texture parameterization that is evolved over the remesh modifications. We cast atlas chartification, parameterization, and packing as 3D problems.

3 OVERVIEW

The input to our algorithm is a temporal sequence of unstructured colored meshes $\{F_i = (T_i, V_i, \rho_i)\}$, where each frame F_i is represented by a triangulation T_i, a set of 3D vertex positions V_i, and a map ρ_i from the mesh surface (T_i, V_i) to colors.

Our goal is to create a *time-evolving mesh* $\left\{F'_i = (T'_i, V'_i, U'_i, I'_i)\right\}$, with 2D vertex texture coordinates U'_i and texture images I'_i, that looks similar to the input sequence but has temporal coherence in connectivity, geometry, parameterization, and texture.

The creation of the time-evolving mesh $\{F'_t\}$ proceeds in 4 steps:

Tracking (§4): Given the new geometry (T'_i, V'_i) at time i, we deform its vertex positions, to obtain the geometry $(T'_i, \widetilde{V}_{i+1})$ that fits the input geometry (T_{i+1}, V_{i+1}) from the next frame.

Remeshing (§5): We identify regions in the deformed mesh $(T'_i, \widetilde{V}_{i+1})$ that fail to match the input and replace these with input geometry, obtaining the remeshed geometry (T'_{i+1}, V'_{i+1}) for the next frame.

Parameterization (§6): We leverage the temporal coherence of the triangles T'_i in the output geometry to define coherent texture atlases U'_i for better compressibility.

Texture sampling (§7): We sample the input surface color ρ_i into the new texture I'_i using the map U'_i and encode the result as a video.

4 TRACKING

To reuse mesh connectivity, we deform the output geometry (T'_i, V'_i) from frame i (referred to as *source*) to match the input geometry (T_{i+1}, V_{i+1}) from frame $i+1$ (referred to as *target*) by assigning new vertex positions, $V'_i \rightarrow \widetilde{V}_{i+1}$. Our tracking strategy has three phases: we use a deformation graph combined with a new correspondence-pruning technique to coarsely align the source to the target (§4.1); we use mesh-based optical flow to correct for tangential drift (§4.2); and we refine the registration using a per-vertex ARAP energy (§4.3).

4.1 Coarse alignment

While the discussion below describes correspondences from the source to the target, in practice we symmetrize the process by using correspondences in both directions.

Our coarse tracking algorithm is based on traditional non-rigid deformation algorithms [e.g., Li et al. 2008; Sumner et al. 2007] augmented with an improved point correspondence. We start by constructing a coarse deformation graph where each node corresponds to a local linear transformation and iteratively alternate between three steps: identifying point correspondences, estimating deformation parameters, and updating the pose. (We use 600 nodes.)

As in the work of [Li et al. 2008], the deformation parameters are represented by a rotation R_j and translation t_j at each node of the

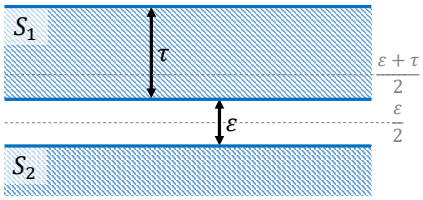


Fig. 2. Without normal consistency, nearest-point correspondences are valid for motions smaller than the separating distance. With, this validity extends to motions on the order of the separating distance plus the model thickness.

deformation graph and are obtained as the minimizers of an energy combining smoothness, interpolation, and rigidity terms:

$$E = \alpha \cdot E_S + \beta \cdot E_I + \gamma \cdot E_R$$

where $E_S(\{R_j, t_j\})$ measures the variation in deformation parameters between adjacent nodes in the deformation graph, $E_I(\{R_j, t_j\})$ measures the distance between corresponding points, $E_R(\{R_j\})$ measures the extent to which the transformations are rotations, and the weights (α, β, γ) balance between the energies.

Correspondence pruning. Li et al. assign point-to-point correspondences from the source to the target using a closest-point query with normal consistency. We extend this by using region-based pruning to discard subsets of bad correspondences.

Figure 2 visualizes our motivation. If the source is composed of shapes S_1 and S_2 that are a distance of ε apart, and the target is obtained by fixing $T_1 = S_1$ and setting T_2 to a translation of S_2, then points on S_j will remain closer to points on T_j as long the translation is smaller than $\varepsilon/2$, (e.g. translating upwards by more than $\varepsilon/2$ will bring T_2 closer to S_1 than S_2). Thus, nearest-point correspondences are only guaranteed for well-separated surfaces.

If we assume that S_1 has thickness τ, then the use of normal consistency extends the guarantee to displacements up to $(\varepsilon + \tau)/2$, and correspondences will be correct even when the surfaces are not well-separated, so long as they are "thick".

We mitigate the "thickness" assumption by noting that while bad correspondences can occur with normal consistency, they tend to form over small regions. We identify *continuity clusters* on the source and discard correspondences from clusters that are too small.

More precisely, setting $\phi : S \to T$ to be the nearest-neighbor map and $d(\cdot, \cdot)$ the geodesic distance on T, we assign a continuity score to an edge $e = (p, q)$ in S by measuring distance on T:

$$c(e) = d(\phi(p), \phi(q)).$$

We remove all edges in the mesh S whose continuity weight is above a threshold and identify the connected components of the pruned graph. Finally, we discard correspondences from components whose area is below a threshold. (We set the thresholds for distance and area to 5 cm and 1/5-th of total area, respectively.)

For efficiency, we do not compute geodesic distances for all $(p', q') = (\phi(p), \phi(q))$. Instead, we perform furthest-point sampling to find landmarks $\{l_1, \ldots, l_\ell\} \subset T$ on the target. We find the solution to the single-source geodesic problem for each landmark [Surazhsky et al. 2005] and store the ℓ-dimensional vector of distances, $\vec{d}_{p'} = \{d(p', l_1), \cdots, d(p', l_\ell)\}$. Using the triangle inequality, we approximate the distance $d(p', q')$ by $\|\vec{d}_{p'} - \vec{d}_{q'}\|_\infty$. (We set $\ell = 6$.)

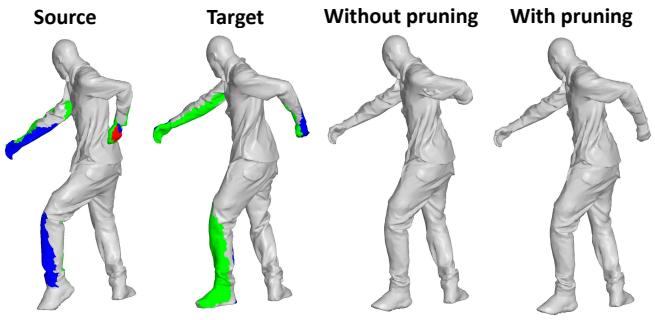


Fig. 3. Example with a hand moving off the body between the source and the target frames. Correspondences are pruned by proximity (>10 cm), normal consistency (>90°), and continuity-cluster size (<20%), shown in blue, green, and red, respectively. Without continuity-cluster pruning, correspondences exist between source points on the hand and target points on the body, pushing the arm towards the torso in the deformation. Pruning discards these bad correspondences, letting the arm move freely to match the target.

Figure 3 shows an example of the improvements gained by pruning small continuity clusters. Without the pruning, correspondences are kept between points on the (thin) hand of the source and points on the body of the target, resulting in a deformation that fails to separate the two. With pruning, such correspondences are removed and the hand moves cleanly away from the body.

4.2 Texture-based registration

The coarse alignment process uses only geometric information to define correspondences and thus cannot account for tangential drift on continuously symmetric surfaces. In our work, avoiding this drift is desirable to preserve the alignment of texture detail, ultimately making the texture videos easier to compress. (Video codecs have motion prediction, but their block-based strategy performs poorly across the parametric discontinuities associated with texture chart boundaries, as confirmed in the results of Table 2.)

We use mesh-based optical flow [Prada et al. 2016] to detect tangential drift and adjust the correspondences as follows. Starting with the coarse alignment described above, we compute the Laplacian of the color signals (to eliminate constant color offsets that result from lighting changes) and transfer the color Laplacian from the deformed source to the target by sampling from the closest, normally consistent, point on the source. Next, we compute the vector field $\vec{v}$ aligning the Laplacians of the input and sampled color fields on the target, and define a new correspondence map

$$\tilde{\phi}(p) = \mathrm{Adv}_{\vec{v},1}(\phi(p))$$

where $\mathrm{Adv}_{\vec{v},s}(p)$ is the map advecting $p \in T$ along $\vec{v}$ for duration s.

Given the adjusted correspondence $\tilde{\phi}$, we rerun coarse alignment to get an improved registration. This process is iterated using the deformed source from the previous pass to define new correspondences and a new optical flow field. (We use two optical-flow-adjusted passes in addition to the initial coarse registration.)

Figure 4 highlights the importance of optical-flow correction. The models are obtained by deforming the positions of a mesh with fixed connectivity and fixed texture coordinates to match different frames of an animation sequence. Blending the parts of the three

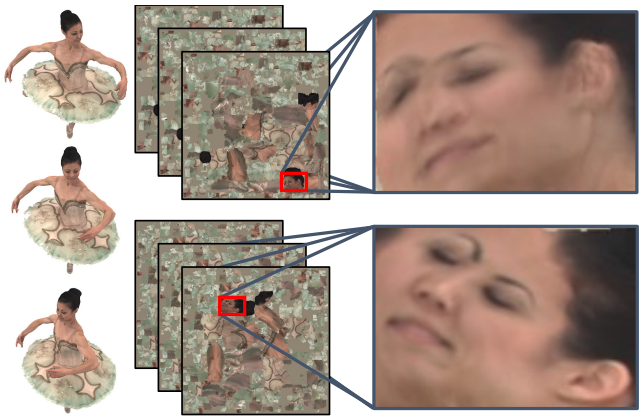


Fig. 4. Comparison of texture atlases without (top) and with (bottom) optical-flow adjustment, showing three frames of a sequence (left), the associated texture maps (middle), and a blend of the face textures (right).

texture maps corresponding to the face of the ballerina, we see pronounced ghosting when optical flow is not used (top). In contrast, incorporating optical-flow preserves the locations of fine features like the eyes and ears in the texture (bottom).

4.3 Fine registration

Given the small number of nodes in the deformation graph, coarse alignment has insufficient degrees of freedom to register fine detail. We refine the coarse alignment using the ARAP deformation of [Sorkine and Alexa 2007]. (We chose ARAP rather than a regularized normal offset as in Bojsen-Hansen et al. [2012] because ARAP constrains the deformation to preserve triangle shape, ensuring that distortions do not accumulate over successive deformations.)

We briefly recall the ARAP approach. ARAP deformation alternates between local and global optimizations. In the local phase, a rotation is computed at each vertex that best aligns the source and target one-rings. In the global phase, a vector-field is computed by using the rotations to transform the difference between neighboring source vertices. A Poisson system is then solved to find new vertex positions whose gradients (softly) match the prescribed vector-field and whose positions lie near the corresponding target points.

In our implementation, we define the correspondences using normal consistency and continuity cluster pruning, as in Section 4.1. To solve for the rotation at a vertex, we use the geometry of the one-ring in the progenitor frame (the output frame that first contains the vertex and its one-ring) to ensure that triangle quality does not deteriorate over time. (We perform 10 local/global iterations.)

5 REMESHING

Due to potentially significant changes in the geometry and topology of the input, it is often impossible to track a single mesh through the entire sequence. In this section, we describe how an evolving mesh addresses this problem. We identify seed points on both the (deformed) source and the target where tracking fails (§5.1) and replace the deformed source geometry with the target geometry in such regions (§5.2).

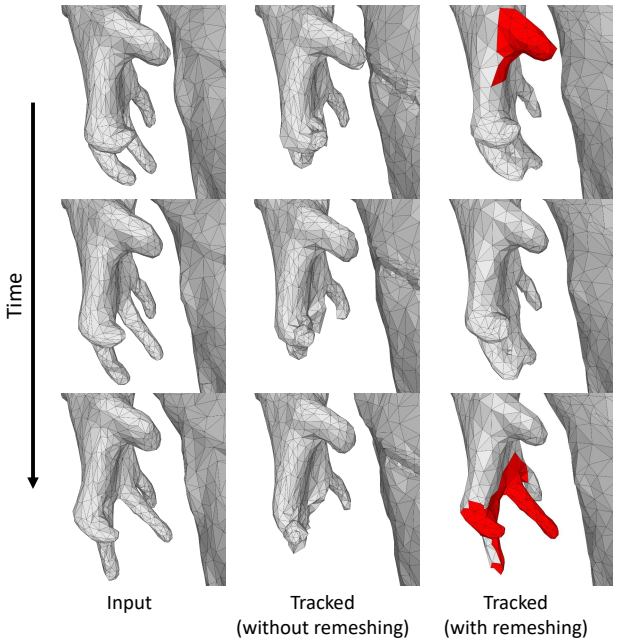


Fig. 5. Close-up on fingers in three successive frames, comparing tracking quality with fixed mesh connectivity and with our adaptive remeshing. Newly remeshed geometry is highlighted in red.

As a motivating example, Figure 5 shows close-ups of three consecutive frames, comparing the target (left), tracked (middle), and remeshed (right) geometries. Due to the fine detail of the fingers, tracking alone fails to provide a good deformation from the source to the target and this failure is propagated over the course of the sequence. Using remeshing, we obtain a geometry that not only is a better fit to the current target, but also provides a higher-quality source for deformation in subsequent frames.

5.1 Identifying tracking failure

We identify seed points on the source $(T'_i, \widetilde{V}_{i+1})$ and target (T_{i+1}, V_{i+1}) for remeshing, using both geometric and parametric measures of failure. In contrast to prior work, local topological changes in the target mesh (such as tiny handles or tunnels) do not necessarily trigger remeshing events.

Geometric failure. We detect if a vertex of the source (resp. target) mesh is far from the target (resp. source). We modulate the distance by the "importance" of the vertex, measured in terms of its visibility and geometric detail. The motivation is to de-emphasize hidden surface regions (e.g., underside of a jacket) while promoting regions with small features (e.g., fingers). For a source vertex v we set:

$$E_G(v) = d(v)\left(1 + \mathrm{vis}(v)\left(\alpha + \beta \cdot \min\left(\frac{m(v)}{m_{\max}}, 1\right)\right)\right)$$

where $d(v)$ is the distance to the target mesh, $\mathrm{vis}(v)$ is a proxy for ambient occlusion (measured using rays in the cone that makes a 45° angle with the vertex normal), and $m(v)$ is the mean curvature. We then mark a vertex as a geometric failure if $E_G(v)$ exceeds a prescribed threshold. (We set the threshold to 10 cm and fix $(\alpha, \beta) = (1, 4/3)$ and $m_{\max} = 5\ \mathrm{cm}^{-1}$.)

In addition, we also consider flipped source triangles as instances of geometric failure. In our implementation we declare a triangle to be flipped if casting a ray in the negative normal direction does not intersect another surface. If we find a sufficiently large edge-connected component of such triangles, we mark them as seeds. (We mark components consisting of five or more triangles.)

Parametric failure. Because our framework holds constant the texture coordinates of tracked triangles, we identify triangles that grow too much in the course of tracking (and hence would cause texels to be stretched over the surface [Sander et al. 2001]), marking such triangles for remeshing. Specifically, we compute the Jacobian J of the map from each original triangle to its current shape in the deformed mesh and test if the squared Frobenius norm $\|J\|_F^2$ exceeds a threshold. If we find a large edge-connected component of such triangles, we mark them as seeds. (We set the threshold to $\|J\|_F^2 > 8$ and identify components consisting of ten or more triangles.)

5.2 Remeshing poorly tracked regions

For each remeshing seed identified on either the source or the target, we determine a pair of corresponding regions on the two meshes and replace the geometry in the source with that of the target.

Identifying corresponding regions. To define corresponding regions, we adapt the voxel-based approach of [Wojtan et al. 2009]. Their technique embeds the source and target meshes in a voxel grid and grows regions by progressively adding voxels until the boundaries of the intersections with the source and target match.

The voxel set $\mathcal{V}$ is initialized using just the voxel containing the seed and is grown into face-adjacent voxels (in a breadth-first manner) until the two meshes are consistent with respect to all the faces on the voxel set boundary. Two meshes are defined to be consistent with respect to a voxel face if the two meshes intersect the same set of edges in the same order.

Figure 6 (top left) shows a visualization with the initial seed (green sphere) on the target geometry. It shows the grown voxel set $\mathcal{V}$ in green wireframe (second column) and highlights the consistency (at voxel granularity) of the two meshes along its boundary.

Replacing source with target geometry. As our goal is to best preserve the input geometry, we cannot use the isosurfacing approach of [Wojtan et al. 2009] to introduce target geometry into the source. Instead, we explicitly join the triangulations of the source outside the voxel set $\mathcal{V}$ with the triangulation of the target inside $\mathcal{V}$.

Specifically, given the voxel set $\mathcal{V}$, we clip the source and target meshes to the boundary of $\mathcal{V}$, remove the source triangles in $\mathcal{V} \cap S$ and fuse the remainder with the target triangles in $\mathcal{V} \cap T$. We perform the fusion by stitching together the two boundaries and then creating a seamless blend across the stitch (Figure 6, bottom).

Stitching. From the face-based consistency constraints, we have a one-to-one correspondence between cycles of edges on the boundaries of the clipped source and target. For each such pair, we parameterize the source and target cycles uniformly over the unit interval and introduce new interior source (resp. target) vertices at the corresponding parametric positions of the interior target (resp. source) vertices, performing edge splits as we do.

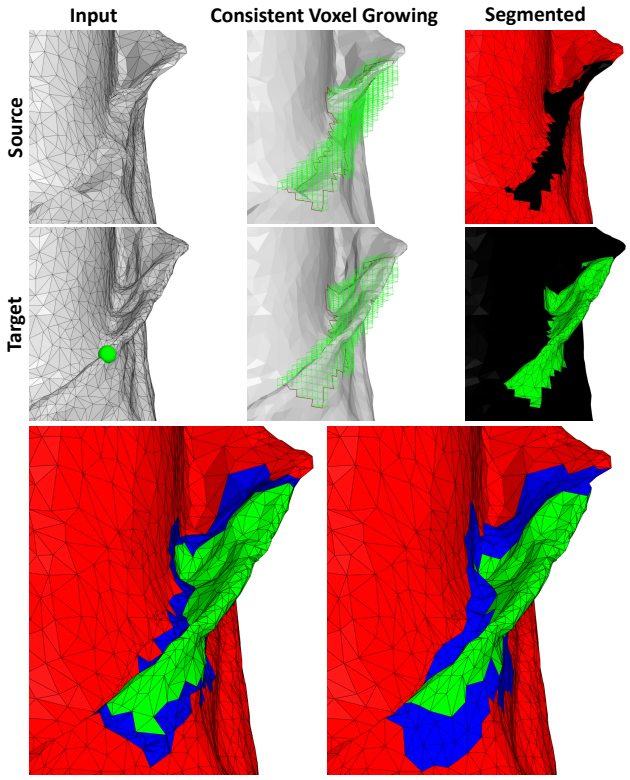


Fig. 6. Visualization of remeshing, showing the source and target mesh (with the remeshing seed as a green sphere), the result of consistent voxel growing, and the segmented source and target regions. We use correspondences between boundary vertices to stitch these together (bottom left) and blend the geometry to obtain a seamless remesh (bottom right).

Now that the source and target regions have boundaries with vertices in one-to-one correspondence, we stitch the surfaces by replacing each corresponding vertex pair with its average.

Blending. To improve the transition between the source and target surfaces, we compute the two-ring neighborhood of vertices along the stitching seam, lock the vertices outside this region, and iterate between (Laplacian) smoothing the vertex positions [Taubin 1995] and removing poorly shaped triangles using edge collapses. Due to the locking, this approach generates a surface that is not smooth at the seam. Smoothness could be obtained, for example, by using bi-Laplacian / Willmore energies [Bobenko and Schröder 2005].

6 ATLAS PARAMETERIZATION

Tracking produces a sequence of meshes in which a large subset of triangles is tracked from one frame to the next. These tracked triangles form temporal chains. Our goal is to create a partition of all triangles into spatiotemporal charts, where each chart consists of a connected component of chains with identical lifespans. Intuitively, a chart is a surface region that is tracked from a given start frame to a given end frame. We assign to each chart a fixed 2D texture parameterization (i.e., constant vertex texture coordinates for all tracked triangles in the chart). In determining the partition into charts, we seek to maximize both temporal and spatial coherence.

We measure *temporal coherence* as the average triangle lifespan. We favor longer lifespans to reduce the number of stored texture coordinates and to maintain temporal coherence in the resulting texture video.

We measure *spatial coherence* as the average number of charts per frame. We favor fewer charts to reduce unused gutter texels needed to separate charts and to minimize the presence of texture discontinuity curves on the rendered meshes.

Optimizing either coherence term alone is straightforward. Maximizing temporal coherence corresponds to creating maximally long chains of tracked triangles, independent of neighboring triangles, but this often leads to small charts. Maximizing spatial coherence is achieved by making all charts have a lifespan of a single frame, so that all triangles in a frame can be parameterized together.

In this section we describe our approach for simultaneously optimizing both objectives (§6.1), parameterizing the resulting charts (§6.2), and assembling them into a 3D texture map (§6.3). We begin, however, with some notation.

Notation. Let T_i' be the triangles in the remeshed output of frame i. Given triangles $t_i \in T_i'$ and $t_{i+1} \in T_{i+1}'$, we write $t_i \mapsto t_{i+1}$ when triangle t_i is deformed into triangle t_{i+1} by the tracking.

- A chain is a subset of nodes $\{t_s, \ldots, t_e\}$ with $t_i \mapsto t_{i+1}$.
- The lifespan of a chain $\{t_s, \ldots, t_e\}$ is the interval $[s, e]$.
- The neighbor graph on $\bigcup T_i'$, denoted $\mathcal{N}$, is the graph with an edge between two triangles/nodes if either they are adjacent in the same frame, or one tracks to the other (in adjacent frames).
- A chart is a connected subgraph $C \subset \mathcal{N}$ whose nodes form chains with identical lifespan, which is denoted $I_C = [s_C, e_C]$.
- Two charts are neighbors if they are connected by an edge in $\mathcal{N}$.
- The cross-section of a chart at time i is $C(i) = C \cap T_i'$.
- An atlas is a set of charts that partitions the neighbor graph $\mathcal{N}$.

Figure 7 (top) shows a visualization of a neighbor graph with subgraphs C_1 and C_2 highlighted. Though both are composed of two chains, only C_1 is a chart. Figure 7 (bottom) shows two different atlases on $\mathcal{N}$. The one on the left maximize temporal coherence and, among all such maxima, also maximizes spatial coherence. The one on the right maximizes spatial coherence and, among all such maxima, also maximizes temporal coherence.

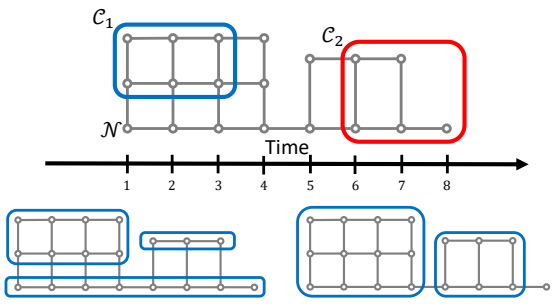


Fig. 7. Visualization of a neighbor graph ($\mathcal{N}$). The top row shows two subgraphs, one of which is a chart (C_1) and one of which is not (C_2). The bottom row shows two possible atlases.

Geometric interpretation. We think of a chart C as a (right) prism through the neighbor graph, with time as its axis, and the cross-section at time i given by $C(i)$. This allows us to define the cap area as twice the average of cross-sectional areas and the side area as the sum of the perimeter lengths of the cross-sections:

$$\|C\|_C = \frac{2}{|I_C|} \cdot \sum_{s \in I_C} |C(s)| \qquad \text{and} \qquad \|C\|_S = \sum_{s \in I_C} |\partial C(s)| \,.$$

6.1 Charting

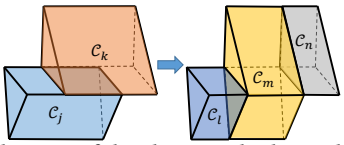
To find a good atlas, we define an energy on the space of atlases, design editing operators to transform one atlas into another, and greedily choose edits that reduce the energy.

Optimization energy. We define the energy of an atlas $\mathcal{A}$ to be the sum of temporal and spatial coherence terms, summed over the atlas charts. We note that a chart ceases to be temporally coherent at its temporal boundaries (its two caps). Similarly, it ceases to be spatially coherent along its spatial boundaries (its sides) and becomes less coherent as it narrows. Taking these in conjunction, we set:

$$E(\mathcal{A}) = \sum_{C \in \mathcal{A}} E(C) \quad \text{with} \quad E(C) = \left(\|C\|_C + \alpha \cdot \frac{\|C\|_S}{\|C\|_C} \right).$$

(We fix α to 10 times the average radius of a triangle.)

Atlas editing operators. To support exploration of the space of atlases, we define an atlas edit operator as the composition of primitive transformations. The *spatial merge* takes two neighboring charts with identical lifespans and joins them into a single chart. It keeps the temporal coherence energy fixed and improves spatial coherence by removing boundaries and increasing the cross-sectional area. The *temporal split* takes a chart and divides it along a cross-sectional slice. It worsens temporal coherence, but lets us trim the lifespans of adjacent charts so they can be merged.



Then, given spatially neighboring charts C_j and C_k, we construct an edit operator composed of at most two temporal splits (to align the lifespans of the charts) followed by a spatial merge of the charts with identical lifespans. This edit creates at most three charts C_l, C_m, and C_n and its change in energy is

$$\Delta E = \Big(E(C_l) + E(C_m) + E(C_n) \Big) - \Big(E(C_j) + E(C_k) \Big).$$

Greedy energy descent. We define a base atlas by computing maximal chains in the neighbor graph and clustering neighboring chains with identical lifespans. This atlas minimizes the temporal coherence energy and, of all such minimizers, it is the one that minimizes the spatial coherence energy.

We optimize the atlas by maintaining a heap of candidate edits, sorted by their associated change in energy. We pop a candidate edit off the heap, perform the edit if the associated charts have not been modified already, create new candidate edits between the new chart and its neighbors, and insert these candidates into the heap if they have negative energy change ΔE.

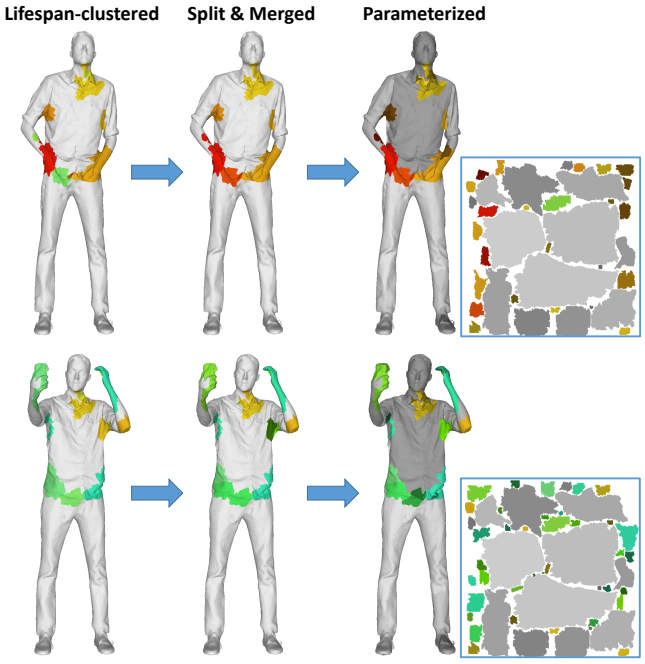


Fig. 8. Visualizations of the charts at two frames of the "selfie" sequence.

6.2 Unwrapping

Given an atlas, we unwrap each chart, mapping it into texture space. For a chart C we take geometry from the first cross-section, $C(s_C)$, and use UVAtlas [Microsoft 2011] to obtain the parameterization of the 2D cross-section. This parameterization is then extruded across the lifespan of the chart.

We note that even though each chart is connected, we are not guaranteed that the unwrapping will occupy a contiguous region in texture space because the parameterization may divide a chart into multiple components to limit geometric distortion. These components form a refined chartification, which we call the *parameterized atlas*.

Figure 8 visualizes the cross-sections of the charts at two intermediate frames of the "selfie"' sequence. The images show cross-sections of the initial lifespan-clustered, optimized, and parameterized charts as distinct patches on the surface as well as the mapping of the parameterized atlas into texture space. (Charts are colored as in Figure 1, using hue and saturation to denote the mid-point and duration of a chart's lifespan.)

As this sequence does not exhibit large distortion, our method can successfully track big charts with long lifespan, visualized by the large gray patches with fixed position.

6.3 Packing

Given the unwrapped charts, we pack them into a 3D texture atlas. Our design strategy aims to support a streamable representation whereby a lightweight client need only store and access a single frame of texture video per rendered frame. Also, we constrain the 3D texture charts to be *right* prisms, so vertices have constant texture coordinates over their lifespans.

To assign texture coordinates to the unwrapped charts, we sort charts by cap area and incrementally place them in the texture volume. (We also tried sorting by chart volume and lifespan, but found that cap area gives the most efficient packing.)

Using the fact that the unwrapped charts are extrusions of 2D cross-sections, we can reduce the placement to a 2D problem. Specifically, given the chart C, we consider all previously placed charts which overlap the lifespan of C, flatten them onto the 2D plane and search for locations in 2D that are empty of previously placed charts and can fit the cross-section of C.

We find such locations by using the approach of [Lévy et al. 2002] which defines *horizons* for both the current texture map and the chart to be inserted. The horizon is an envelope defined around a line, with one horizon contained in the free space of the texture domain and the other containing the chart. Shifting one horizon along the other provides an efficient way to identify locations in texture space which can accommodate the new chart.

While a direct implementation successfully places the charts, it is not efficient. This is because the original method uses a single texture horizon at the top of the 2D domain, and places the new charts as close to the bottom as possible. As the packing is performed from bottom to top, the horizon provides a good representation of the remaining free space. In our context, we perform the packing in 2D texture space, but then extrude the result into the 3D texture volume, with different charts extruded by different lifespans. Thus, even if charts appear to be tightly packed at the bottom of one 2D flattening, the packing may not be tight when flattened onto a different lifespan.

To work around this, we use multiple horizon lines for the 2D texture space, testing each one to identify candidate locations. Once the locations are identified, we place the chart at the location closest to the projected center of *all* the charts already placed into the atlas. (We use 16 evenly spaced horizontal and vertical lines in our experiments and have found diminishing returns when using more.)

7 TEXTURE SAMPLING

In the final step of our processing, we assign color values to both the interior and gutter texels of the texture video.

Assigning interior texels. Given the packed charts, we assign a color value to an interior texel at time i by sampling the input color ρ_i. If the color ρ_i is originally defined from projected camera images, sampling is performed by back-projecting into the most forward-facing camera as in [Collet et al. 2015]. Otherwise, we sample ρ_i at the closest point on the input geometry.

Diffusing into gutter texels. To improve video compression, we smoothly diffuse the texel values at chart boundaries into the gutters. We do this by solving a 3D Laplace equation, using an approximate coarse-to-fine solver. At each level of the hierarchy, we label the texels as either *gutter* or *boundary*, discretize the system using a 7-point Laplacian with Neumann constraints at the boundaries of texture space and Dirichlet constraints at seam boundaries, and perform Gauss-Seidel relaxation to update the solution. At the finest resolution, the gutter texels and boundary values are given. At coarser resolutions, a texel is in the gutter if *all* of its eight children are gutter texels, and the value of a boundary texel is set to the

		Geometric accuracy (distance in mm)			Parameterization coherence Temporal: lifespan			Parameterization coherence Spatial: charts/frame			Packing efficiency (3D occupancy)		
Sequence	Frames	KF1	KF2	EV	KF1	KF2	EV	KF1	KF2	EV	KF1	KF2	EV
Selfie	360	1.0	0.3	0.6	45.0	9.5	257.2	23	24	45	70%	71%	72%
Macarena	520	1.7	0.5	0.7	16.0	13.3	306.4	28	33	65	73%	71%	72%
Slick	607	1.5	0.5	1.2	12.4	10.1	179.3	25	26	102	69%	70%	65%
Soccer	471	1.6	0.5	0.8	14.7	5.8	134.7	25	24	94	72%	71%	65%
Breakers	491	1.8	0.4	0.8	14.0	6.4	108.6	17	16	141	65%	68%	64%
Jumping Jacks	259	1.9	0.5	0.9	8.4	4.1	81.2	28	25	123	71%	70%	63%
Chair	542	0.9	0.5	1.0	4.3	3.0	96.3	53	55	210	72%	73%	62%
Girl	700	1.3	0.4	0.9	5.2	3.3	48.5	27	27	158	70%	70%	59%
Ballerina	220	1.9	0.4	1.1	5.1	1.9	34.9	69	63	235	75%	73%	65%
Clothing	731	0.9	0.4	0.9	3.8	1.7	31.8	42	42	226	72%	72%	43%

Table 1. Geometric and parametric properties of the output tracked meshes for three approaches (mesh keyframes and our evolving mesh), comparing the average residual distance (in mm) of the tracked meshes, the average triangle lifespan (in frames), the average number of charts per frame, and percentage occupancy of the texture video. (Ideal results should have small distance residuals, long lifespans, fewer charts, and high occupancy.)

average of the values of its *non-gutter* children. Once the system is relaxed at the coarser resolution, the solution is prolonged into the next resolution by replicating the values of each gutter and boundary texel at the coarse level into its (at most eight) gutter children. (We use 10 passes of Gauss-Seidel relaxation at each level.)

8 RESULTS AND DISCUSSION

To evaluate our method, we process ten sequences, ranging in length from 220 frames to 731 frames (at 30 fps), where each frame consists of a mesh of 40K triangles, textured from projected camera views as in the work of [Collet et al. 2015]. All sequences exhibit some change in surface genus over their duration, with the complexity ranging from a simple sequence in which a character removes a cell-phone from his pocket to take a picture ("selfie") to a complex one in which a character removes a shirt and puts on a jacket ("clothing"). Table 1 orders the sequences from simplest to most complex. We generate time-evolving meshes with approximately 40K triangles per frame and 1024×1024 textures.

Figure 9 shows several frames from six sequences, along with the parameterized charts at the first and last frames. Again the sequences are ordered from simplest to most complex, and we see a commensurate increase in short-lifespan charts (shown with saturated colors).

We compare the results of our evolving mesh approach (EV) to the keyframe meshes (KF) of [Collet et al. 2015]. Both approaches leverage tracking to obtain a temporally coherent representation of the mesh, with tracking failure used to initiate keyframing/remeshing. The difference is that while we replace only the subset of the geometry, the keyframing approach replaces the entire mesh. We compare to two variants of the keyframe approach. In the first (KF1), we define tracking failure strictly in terms of distance to the target and set the threshold to obtain results similar to those described in [Collet et al. 2015]. In the second (KF2), we use the same measure of tracking failure as for our evolving mesh and incorporate optical-flow adjustment in the tracking process as well.

Geometric quality. Table 1 (left) compares the accuracy of the surfaces produced using keyframing with the accuracy produced using an evolving mesh. The distance between two surfaces is given by the symmetrized average distance between a point on one surface and the nearest point on the other [Jacobson et al. 2016].

The table shows that all three methods generate accurate output geometry, with an average error of 0.5–2.0 mm, or roughly 1/1000th of the model size. When using keyframe meshes with the same measure of tracking failure as our approach, the approach of [Collet et al. 2015] produces output with smaller average distances, because each keyframe replaces the entire geometry at a frame of tracking failure so the distance is reset to zero everywhere.

Coherence. The middle of the table compares the parameterization coherence of the approaches. Temporal coherence is measured by the average lifespan of triangles in the parameterized atlas, while spatial coherence is measured by the average number of disconnected charts in a cross-section.

The table reveals the trade-off between optimizing for temporal and spatial coherence. Both variations of keyframe meshes construct a parameterization using the geometry from a single frame, resulting in 2D texture maps that are optimally spatially coherent. In contrast, our approach begins with an optimally temporally coherent mesh and transforms it to balance the two criteria. As a result, our evolving mesh produces triangles with significantly higher average lifespan while creating more disconnected 2D cross-sections. (Though not shown here, the temporally optimal atlas from Section 6.1 produces charts with a lifespan that is 35% longer, on average.)

The spatial optimality of both variants of keyframing is reflected in the fact that the two implementations generate similar numbers of charts per frame. The difference becomes evident when considering temporal coherence. Using a more conservative measure for tracking failure (KF2) creates keyframes at a higher frequency, resulting in substantially shorter lifespans. The use of a laxer measure (KF1) alleviates this but comes at the cost of degraded geometric fidelity.

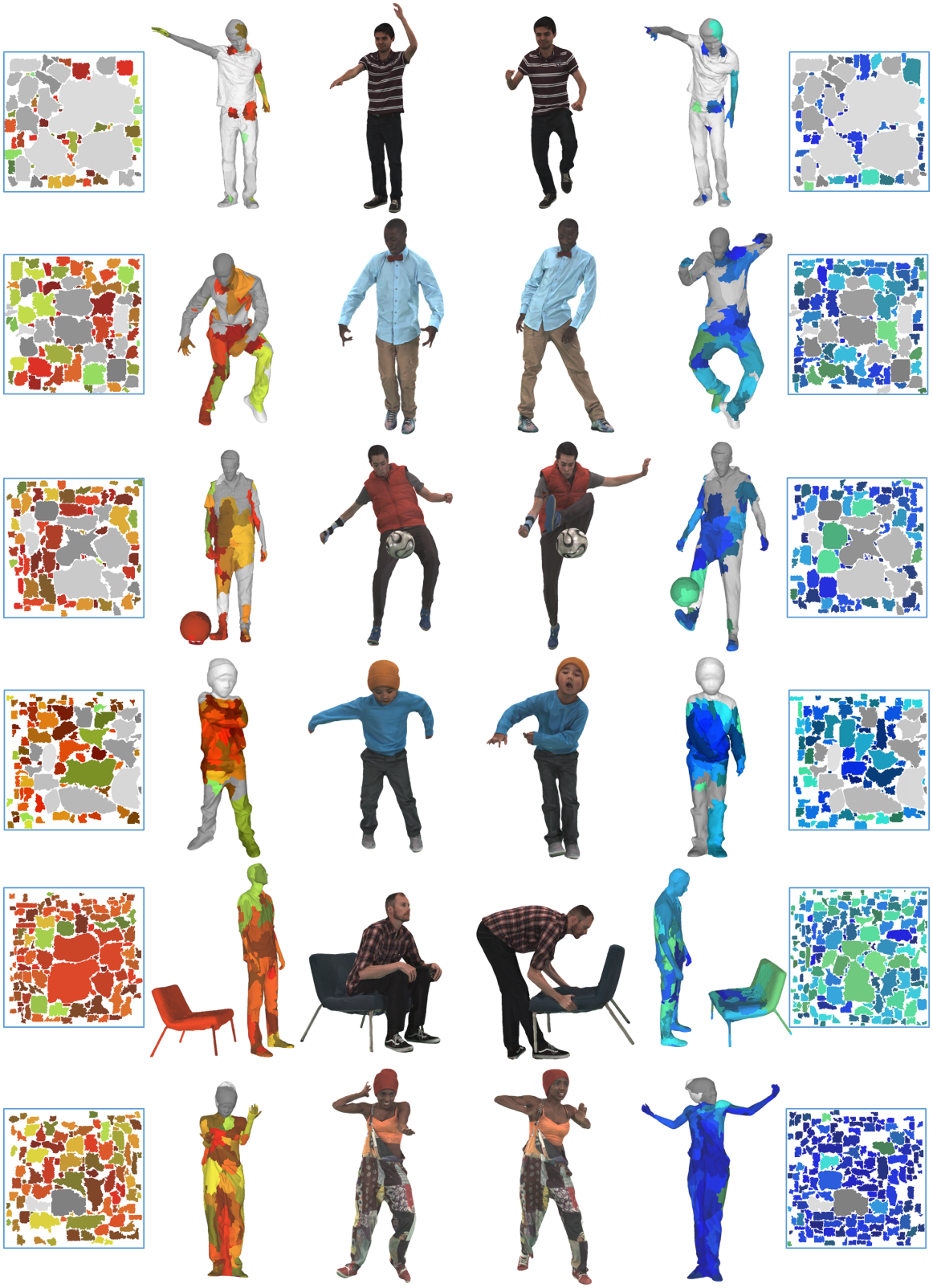


Fig. 9. Frames from the "Macarena", "slick", "soccer", "breakers", "chair", and "girl" sequences, showing the parameterized charts on the surface and in texture space. Hue and saturation visualize the midpoint and length of each chart's lifespan. (Red is early, blue is late, and saturated colors have shorter lifespan.)

Sequence	Input	KF1	KF2	EV−−	EV+−	EV++
Selfie	7.8 (64.3)	1.8 (6.6)	2.4	1.8	1.7	1.5 (6.3)
Macarena	5.8 (64.1)	2.3 (8.7)	2.1	2.2	2.0	1.9 (7.3)
Soccer	6.0 (64.9)	3.0 (12.0)	3.2	2.9	2.7	2.7 (9.8)
Girl	9.7 (65.6)	5.9 (19.8)	5.9	6.4	5.6	5.4 (11.5)
Ballerina	9.1 (65.8)	6.5 (21.7)	7.1	6.5	6.4	5.8 (14.5)
Chair	7.8 (64.8)	3.6 (19.7)	4.2	3.4	3.4	3.1 (10.4)

Table 2. Texture video bitrates (Mbps) for the unstructured input, keyframe meshes, and an evolving mesh without optical flow or harmonic fill-in enabled (−−), with optical flow but without harmonic fill-in enabled (+−), and with both optical flow and harmonic fill-in enabled (++). (Geometry bitrates are given in parentheses.)

(§4) Tracking	66.3
Coarse alignment / Texture-based registration / Fine registration	29.4 / 30.3 / 6.6
(§5) Remeshing	1.3
Identifying tracking failure / Remeshing poorly tracked regions	0.3 / 1.0
(§6) Parameterization	0.3
Charting / Unwrapping / Packing	0.1 / 0.1 / 0.1
(§7) Texture Sampling	14.9
Assigning interior texels / Diffusing into gutter texels	14.6 / 0.3
Average frame processing time	82.8

Table 3. Average time in seconds to process each frame of the Slick sequence.

Packing efficiency. The right of Table 1 compares the packing efficiency of the three methods and highlights a challenge of using an evolving mesh. In contrast to keyframe meshing which packs charts in 2D, our approach must solve the more difficult problem of packing charts in 3D. One implication of this is that texels have a "size" that persists across the 3D texture volume, so the fraction of area occupied in a 2D texture slice is bounded by the ratio of the surface area seen in that frame to the maximum surface area seen in any frame of the sequence. In particular, for sequences like "clothing" where large parts of the geometry are only revealed in a small subset of the sequence, the texture atlas will not be packed efficiently. (This is evident in Figure 1, where the 2D texture space in the first, second, and fourth frames is only partially occupied because of the larger surface area in the third.) Fortunately, the texture gutter areas compress well using standard video encoders.

Video compression. As a measure of coherence, we use the bitrate of the texture video compressed with H.264/MPEG-4 at a fixed quality level, i.e., the default constant rate factor (CRF) of 23. Table 2 assesses the impact of the individual steps in our approach on coherence. It shows the bitrate for the unstructured input, keyframe meshes, an evolving mesh without optical flow or harmonic fill-in, an evolving mesh with optical flow but without harmonic fill-in, and an evolving mesh with optical flow and harmonic fill-in. To adjust for the different packing efficiencies, we adjust the resolutions of the texture atlases such that the number of interior texels for a given sequence matches that resulting from our EV++ approach.

As Table 2 demonstrates, the different steps of our adaptive remeshing all improve the compression quality, resulting in a roughly 12% reduction in compressed texture storage. (Similar experiments using a lower-quality CRF of 29 show more significant reductions, of roughly 18%.)

Geometry compression. Though we do not address this problem in our work, we use naïve encoding to evaluate the effects of our method on mesh storage. We represent a chart (with N triangles per frame and a lifespan of ℓ) using 16-bit integers for vertex indices (connectivity), quantized vertex positions, and quantized texture coordinates. Connectivity and texture coordinates need only be stored for the first frame of the chart, requiring roughly $8 \cdot N$ bytes. Vertex positions are encoded at each frame and are represented using absolute position values in the first frame and changes (deltas) in subsequent frames, requiring roughly $3 \cdot N \cdot \ell$ bytes. Finally, we leverage repetitions in the deltas, storing the data as a .zip file with default compression.

The parenthesized values in Table 2 are the geometry bitrates for the input, keyframe mesh approach, and our time-evolving representation. The bitrate for the input geometry is high because connectivity, texture coordinates, and absolute position values must all be stored at each frame. Both the keyframe approach and our method show marked improvement due to the sparse storage of connectivity and texture parameterization, as well as the compressibility of the quantized change in position values. The table also highlights that with the increased lifespan of our charts, we are able to achieve a roughly 30% reduction in geometry storage size over the keyframe approach, on average.

Performance. Table 3 breaks down the average time for processing the frames of the Slick sequence on a PC with a quad-core i7-5700HQ processor and 16GB of memory. The tracking stage, which is run for all the 606 pairs of consecutive frames, takes the longest. The remeshing stage is called only when a poorly tracked region is detected. (For the Slick sequence this happens in 122 frames.) Though the parametrization stage is global, the associated computation takes a small fraction of the total runtime. Texture sampling is a relatively expensive stage but is easily parallelized.

Our running time is comparable to the keyframe-based approach [Collet et al. 2015]. Both approaches incur an overhead when tracking fails: for evolving meshes this triggers a remeshing step, and for the keyframe approach it requires the selection of an optimal reference geometry. As shown in Table 3, this overhead is negligible when compared to the time spent on tracking, parametrization, and texture sampling, steps that are required by both approaches.

Limitations. Figure 10 illustrates limitations of our approach. The top row shows an artifact resulting from setting the geometric penalty too large: fingers are introduced on the sleeve in one frame (left) and persist even after the hand has moved away from the sleeve (right). The middle row shows an artifact resulting from setting the geometric penalty too small: while the tracked mesh (left) provides a reasonable visual approximation of the target frame, the presence of high frequency noise in the subsequent frame forces a local remeshing operation (right). The bottom shows the impact of optical flow failure: when the flow fails to align texture due to fast tangential motion, drift becomes evident in the texture domain.

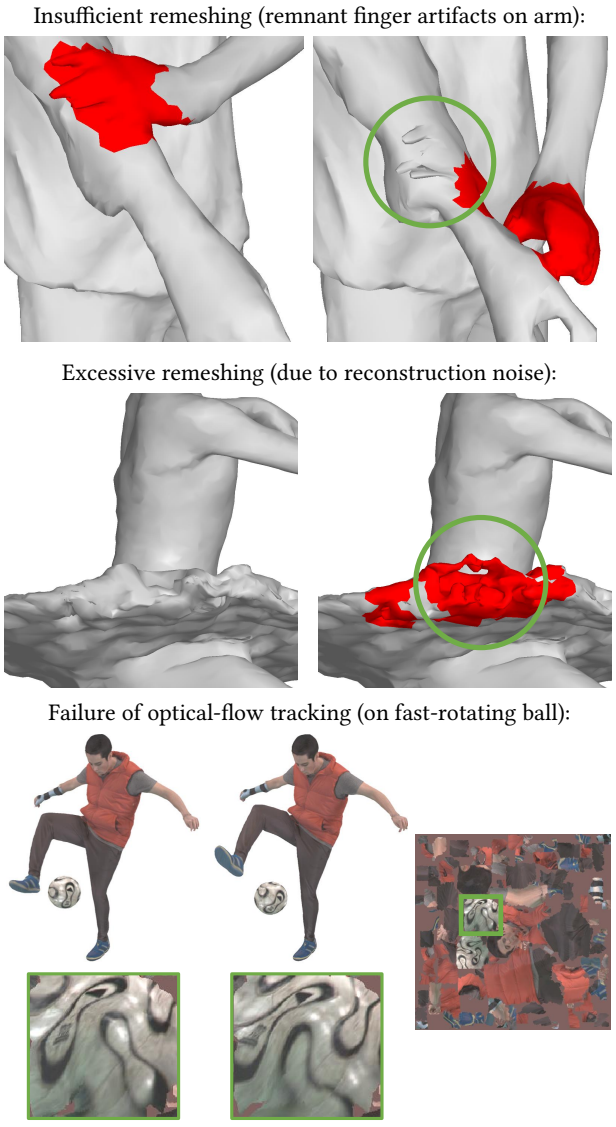


Fig. 10. Limitations of our approach.

9 SUMMARY AND FUTURE WORK

We present a new approach for representing sequences of textured, animated meshes. Starting from unstructured meshes, our approach identifies spatial and temporal coherence to create a time-evolving mesh, in which triangles persist across multiple frames, maintaining a fixed parameterization into a texture atlas, wherein the texture content is aligned across frames to improve compressibility.

We improve on earlier tracking methods by introducing a correspondence-pruning approach and leveraging mesh-based optical flow to correct for tangential drift. A new remeshing scheme selectively replaces geometry in regions of tracking failure, avoiding the resampling required by marching cubes. We present the first description of the space of 3D texture atlases, and implement an optimization algorithm to search this space for atlases with improved spatiotemporal coherence.

We demonstrate the practicality of our framework on several high-quality captured sequences of varying complexity, and quantify the improvements in both coherence and compressibility.

In the future, we would like to consider several extensions, including new metrics for evaluating the quality of meshes with evolving connectivity, alternate strategies for exploring the space of 3D atlases, new data structures for representing temporally evolving meshes, and less greedy approaches for packing charts into the texture volume. We would also like to consider more general packing paradigms that trade off simpler texture storage and access for better packing efficiency. These include allowing texture content to be displaced temporally and allowing chart prisms to be sheared for more flexible 3D packing.

The ultimate goal is to construct time-evolving meshes that are perceptually similar to the input meshes yet allow a compressed streamed representation that is as compact as possible.

ACKNOWLEDGMENTS

We are very grateful to Spencer Reynolds for helping provide test data for our experiments.

REFERENCES

Alexander I Bobenko and Peter Schröder. 2005. Discrete Willmore flow. *Symposium on Geometry Processing* (2005).
Morten Bojsen-Hansen, Hao Li, and Chris Wojtan. 2012. Tracking surfaces with evolving topology. *ACM Trans. Graph.* 31, 4, Article 53 (July 2012).
Chris Budd, Peng Huang, Martin Klaudiny, and Adrian Hilton. 2013. Global non-rigid alignment of surface sequences. *Int. J. Comput. Vision* 102 (2013).
Alvaro Collet, Ming Chuang, Pat Sweeney, Don Gillett, Dennis Evseev, David Calabrese, Hugues Hoppe, Adam Kirk, and Steve Sullivan. 2015. High-quality streamable free-viewpoint video. *ACM Trans. Graph.* 34 (2015).
Mingsong Dou, Sameh Khamis, Yury Degtyarev, Philip Davidson, Sean Ryan Fanello, Adarsh Kowdle, Sergio Orts Escolano, Christoph Rhemann, David Kim, Jonathan Taylor, Pushmeet Kohli, Vladimir Tankovich, and Shahram Izadi. 2016. Fusion4D: Real-time performance capture of challenging scenes. *ACM Trans. Graph.* 35, 4, Article 114 (July 2016).
Alec Jacobson, Daniele Panozzo, and others. 2016. libigl: A simple C++ geometry processing library. http://libigl.github.io/libigl/. (2016).
Vladimir G. Kim, Yaron Lipman, and Thomas Funkhouser. 2011. Blended intrinsic maps. *ACM Trans. Graph.*, Article 79 (2011).
Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. 2002. Least squares conformal maps for automatic texture atlas generation. In *Proc. ACM SIGGRAPH.*
Hao Li, Bart Adams, Leonidas J. Guibas, and Mark Pauly. 2009. Robust single-view geometry and motion reconstruction. *ACM Trans. Graph.* 28 (2009).
Hao Li, Linjie Luo, Daniel Vlasic, Pieter Peers, Jovan Popović, Mark Pauly, and Szymon Rusinkiewicz. 2012. Temporally coherent completion of dynamic shapes. *ACM Trans. Graph.* 31, 1, Article 2 (Feb. 2012).
Hao Li, Robert W. Sumner, and Mark Pauly. 2008. Global correspondence optimization for non-rigid registration of depth scans. In *Symposium on Geometry Processing.* 10.
Haggai Maron, Nadav Dym, Itay Kezurer, Shahar Kovalsky, and Yaron Lipman. 2016. Point registration via efficient convex relaxation. *ACM Trans. Graph.* 35, 4, Article 73 (July 2016).
Microsoft. 2011. UVAtlas. https://github.com/Microsoft/UVAtlas. (2011).
Richard A. Newcombe, Dieter Fox, and Steven M. Seitz. 2015. DynamicFusion: Reconstruction and tracking of non-rigid scenes in real-time. In *IEEE CVPR.*
Fabián Prada, Misha Kazhdan, Ming Chuang, Alvaro Collet, and Hugues Hoppe. 2016. Motion graphs for unstructured textured meshes. *ACM Trans. Graph.* 35, 4, Article 108 (July 2016).
Pedro Sander, John Snyder, Steven Gortler, and Hugues Hoppe. 2001. Texture mapping progressive meshes. In *Proc. ACM SIGGRAPH.* 409–416.
Alla Sheffer, Emil Praun, and Kenneth Rose. 2006. Mesh parameterization methods and their applications. *Foundations and Trends in Computer Graphics and Vision* 2, 2 (2006).
Justin Solomon, Gabriel Peyré, Vladimir G. Kim, and Suvrit Sra. 2016. Entropic metric alignment for correspondence problems. *ACM Trans. Graph.* 35, 4, Article 72 (July 2016), 13 pages.
Olga Sorkine and Marc Alexa. 2007. As-rigid-as-possible surface modeling. In *Symp. on Geometry Processing.*

Jonathan Starck and Adrian Hilton. 2007. Surface capture for performance-based animation. *IEEE Computer Graphics and Application* 27 (2007).

Robert W. Sumner, Johannes Schmid, and Mark Pauly. 2007. Embedded deformation for shape manipulation. *ACM Trans. Graph.* 26 (2007).

Vitaly Surazhsky, Tatiana Surazhsky, Danil Kirsanov, Steven J. Gortler, and Hugues Hoppe. 2005. Fast exact and approximate geodesics on meshes. *ACM Trans. Graph.* 24, 3 (July 2005).

Gabriel Taubin. 1995. A signal processing approach to fair surface design. In *Proc. ACM SIGGRAPH*. 8.

Art Tevs, Alexander Berner, Michael Wand, Ivo Ihrke, Martin Bokeloh, Jens Kerber, and Hans-Peter Seidel. 2012. Animation cartography: Intrinsic reconstruction of shape and motion. *ACM Trans. Graph.* 31, 2 (2012).

Chris Wojtan, Nils Thürey, Markus Gross, and Greg Turk. 2009. Deforming meshes that split and merge. *ACM Trans. Graph.* 28, 3, Article 76 (July 2009).

Faxin Yu, Zheming Lu, Hao Luo, and Pinghui Wang. 2011. *Three-dimensional model analysis and processing.* Springer Science & Business Media.

C Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon Winder, and Richard Szeliski. 2004. High-quality video view interpolation using a layered representation. *ACM Trans. Graph.* 23 (2004).

Michael Zollhöfer, Matthias Nießner, Shahram Izadi, Christoph Rehmann, Christopher Zach, Matthew Fisher, Chenglei Wu, Andrew Fitzgibbon, Charles Loop, Christian Theobalt, and Marc Stamminger. 2014. Real-time non-rigid reconstruction using an RGB-D camera. *ACM Trans. Graph.* 33 (2014).