One-to-Many: Example-Based Mesh Animation Synthesis

Changxi Zheng
Columbia University

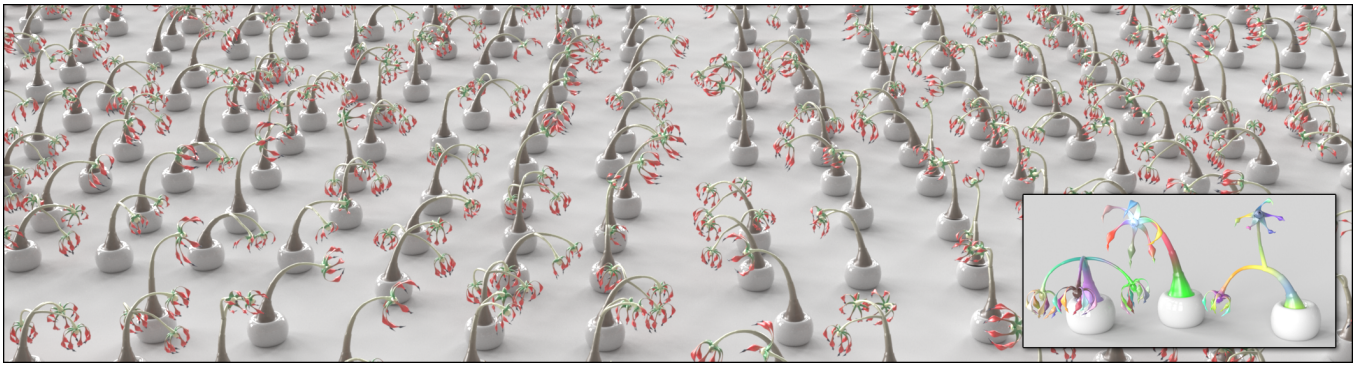


Figure 1: 1000 starving feeding creatures *are synthesized from a provided short clip of deformable mesh animations (in the inset). The output animations can have arbitrary length, each presenting different motions. Our method is fast, requires no knowledge of the models for creating the examples, and supports various types of deformable mesh animations.*

Abstract

We propose an example-based approach for synthesizing diverse mesh animations. Provided a short clip of deformable mesh animation, our method synthesizes a large number of different animations of arbitrary length. Combining an automatically inferred linear blending skinning (LBS) model with a PCA-based model reduction, our method identifies possible smooth transitions in the example sequence. To create smooth transitions, we synthesize reduced deformation parameters based on a set of characteristic key vertices on the mesh. Furthermore, by analyzing cut nodes on a graph built upon the LBS model, we are able to decompose the mesh into independent components. Motions of these components are synthesized individually and assembled together. Our method has the complexity independent from mesh resolutions, enabling efficient generation of arbitrarily long animations without tedious parameter tuning and heavy computation. We evaluate our method on various animation examples, and demonstrate that numerous diverse animations can be generated from each single example.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Object representations I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: animation, synthesis, transition, bone graph, cut node

1 Introduction

Creating deformable mesh animations is a key task in computer graphics. Traditionally, it is either manually crafted by skilled animators or automatically generated using physics-based simulations. Although proven to be very successful, these methods can still suffer from high production cost when we strive to capture the tremendous richness of our real-world motions: no tree leaves move in exactly the same way; flags in a wind flap differently; and people walk, run and dance with a variety of styles. To produce rich variations, one has to laboriously adjust underlying motion curves or change related parameters and repeat simulations.

In this paper, we explore a different approach of creating diverse deformable mesh animations by addressing a key question: can we automatically synthesize different mesh animations from a single deforming sequence (see Figure 1)? Indeed, many other aspects of computer graphics have witnessed successful example-based synthesis methods (such as character animation [Kovar et al. 2002], texture synthesis [Wei et al. 2009], shape design [Kalogerakis et al. 2012], mesh posing [Der et al. 2006], noise pattern design [Galerne et al. 2012], and simulation [Martin et al. 2011]). For mesh animation synthesis, however, major challenges arise from the curse of dimensionality and the need to ensure spatial and temporal coherence: it can be difficult to synthesize high-dimensional animations, such as mesh deformations, with limited amount of data; meanwhile, it is critical to ensure all mesh vertices to move in a spatially and temporally coordinated fashion; additionally, we hope to generate a large variety of animations, all of which satisfy user-specified constraints.

Our approach starts by estimating a *linear blend skinning* (LBS) model [James and Twigg 2005; Kavan et al. 2010; Le and Deng 2012] from the provided example. Using it as a reduced deformation model, we manage the geometric complexity of detailed meshes and ensure spatial coherence of their deformations. However, our key goal is different from all the previous work about mesh skinning and deformation editing: we aim to synthesize motions of all the LBS bones and thereby create diverse mesh animations.

Contributions Our approach has the following three major contributions. (i) First we propose a difference metric of mesh deformation state for identifying possible transitions in the example sequence. We define the difference metric in the local frame of reference of a selected bone for eliminating any linear transformations, and employ a PCA-based reduced coordinates for fast evaluation. (ii) Unlike character animation synthesis [Kovar et al. 2002; Arikan and Forsyth 2002], simply blending mesh animations at transitions

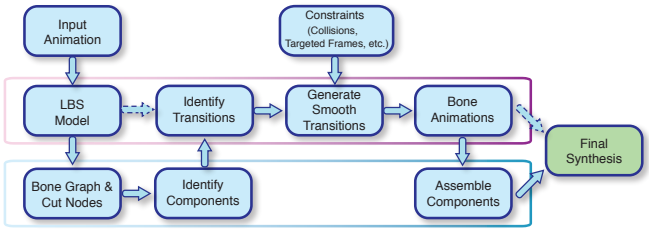


Figure 2: Overview: *We synthesize various mesh animations from a single input by inferring an LBS model and synthesizing LBS bone motions (in magenta rectangle). Furthermore, by identifying weakly coupled LBS components, we can synthesize animations asynchronously in multiple components (in blue rectangle). The dash arrows indicate a possible data flow when performing single-components synthesis.*

can lead to unpleasant artifacts, because for high-dimensional mesh deformations it can be rare, if not impossible, to find close enough states from the provided example. We instead blend the deformation gradients at a set of selected characteristic key vertices, and thereby estimate temporally coherent bone motions and the resulting mesh deformations. (iii) Furthermore, we decompose the mesh into multiple regions whose deformations are weakly coupled with each other. We find that these regions can be identified by the cut nodes of a bone graph, a graph extracted from the LBS model. We then independently synthesize the bone motions of different graph components separated by the cut nodes. Finally, combining the motions of those components asynchronously produces numerous different animations (see Figure 1).

Applications Our method offers an efficient and resolution-independent way of synthesizing a large number of similar yet different animations. It is particularly powerful for creating an animation database as well as generating "motion textures" in a large scale animating environment. In addition to the efficiency and diversity, directly generating motion curves for LBS bones allows our method to be easily integrated into a standard animation pipeline: an animator can easily post-edit synthesized animations by adjusting generated LBS motion curves [Der et al. 2006], or assemble together animations of different mesh components (see Section 4).

2 Related Work

Example-based synthesis has seen wonderful success in many aspects of computer graphics. Among those listed above, most closely related to our work are several data-driven character animation techniques. Pullen and Bregler [2000] created cyclic character motions by sampling joint angles using constructed wavelets and Laplacian pyramid. Our approach shares some similarities with motion graphs [Kovar et al. 2002; Arikan and Forsyth 2002; Lee et al. 2002; Gleicher et al. 2003], which encapsulate connections among a database of motion clips. User-directed character motions are synthesized by graph walks that meet user specifications, and simple linear blending techniques are used to generate transitions between two clips. Although sufficient for those low-dimensional character motion data, for our goal it is hard to blend two sequences of mesh deformations without introducing artifacts. Therefore, we propose a gradient-domain blending technique to address it. Moreover, these methods often require thousands of input frames to produce plausible results. However, creating such a large corpus of deformable mesh animations can be prohibitively expensive. Less input frames are used in [Lau et al. 2009], in which the method learned a Dynamic Bayesian Network to model variations of human motions from hundreds of frames. However, this method only aims to produce subtle variations close to input motions. In contrast, our method is able to produce large distinctions. Lastly, all these methods are particularly suitable for character animations which can be efficiently represented by low-dimensional datasets, such as hierarchical kinematic skeletons together with time-varying joint angles. Unfortunately, this representation leads to the curse of dimensionality when it comes to animate high-dimensional phenomena, such as mesh deformations. Our method, complementary to all these methods, is designed to synthesize deformable mesh animations.

Regarding to mesh animation synthesis, the most closely related work is Mesh Ensemble Motion Graphs (MEMG) [James et al. 2007], which played back motion clips of deformable ensembles. Given a candidate transition location, it asynchronously offsets the transitions of ensembles to nearby frames to avoid interpenetrations. While their work is focused on looping the input animations of a group of ensembles, our method has a different goal, aiming to produce various animations of a single deformable object. Key technical differences include: (i) we address the problem of selecting plausible transition locations up to arbitrary rigid transformations using an LBS model, whereas MEMG assumes all meshes are fixed at some locations and uses reduced modal coordinates to represent moderate deformations. (ii) we use gradient domain blending to avoid transition artifacts, rather than a simple linear blending of reduced coordinates as used in MEMG. And (iii) we address the problem of decomposing a mesh into weakly coupled regions for independent asynchronous animation synthesis.

Combing with various physics-based simulation techniques, data-driven animation synthesis has proven to be successful in many specific cases. For example, James and Fatahalian [2003] tabulated specialized impulse responses to enable transitions between precomputed motion clips. Specifically for cloth animations, data-driven methods have been developed to enrich the details of a cloth simulation [Wang et al. 2010; Kavan et al. 2011]. Moreover, user-provided examples have been successfully used to guide simulation methods to mimic the dynamics of given examples for cloth [Wang et al. 2011] and elastic bodies [Martin et al. 2011]. Different from these work, our method, assuming no knowledge of underlying physics, synthesizes animations kinematically and supports arbitrary mesh deformations.

Our method is partially inspired by "video textures" [Schödl and Essa 2002; Agarwala et al. 2005; Kwatra et al. 2005], in which a set of transition frames of a video is detected and is used for asynchronous synthesis of new video clips. In our problem, we consider the 3D deformable animation synthesis. The fundamentally different animation data from video clips necessitate new metrics for asynchronous transition identification and new algorithms for blending deformation sequences.

Finally, our method relies on an LBS model, which can be automatically learned from an input animation. To this end, we use [Le and Deng 2012] because it guarantees an unit sum of bone weights, an important property for ensuring reliable gradient-domain computation in our scheme (see Section 3.3). Other techniques [James and Twigg 2005; Kavan et al. 2010] have also been devised, while some of them (such as [James and Twigg 2005]) use the unit sum of bone weights as a soft constraints that might not be satisfied exactly in the solution. Finally, we note that an inferred LBS model as a reduced deformable model has been used for character pose editing [Der et al. 2006; Wang et al. 2007]. In this paper, we employ it for animation synthesis.

3 Single-Component Animation Synthesis

An overview of our method is depicted in Figure 2. In this section, we start from presenting our synthesis method for a deformable mesh as a single component. Later in section 4, we extend our method to decompose a deformable mesh into weakly coupled com-

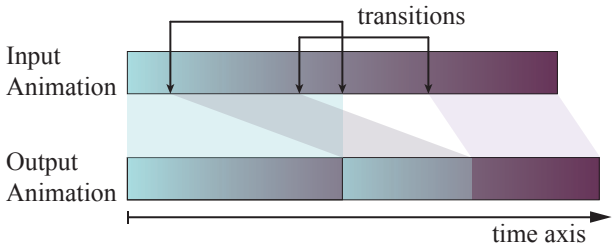


Figure 3: Single-Component Animation Synthesis: *(Top) We identify transitions of a single input animation. (Bottom) New animations are synthesized by splicing input animation segments together and creating smooth transitions.*

ponents and synthesize their animations independently. Provided an input animation sequence of N frames, our basic strategy is to find smooth transitions between two frames, and optionally splice animation segments at those transitions (see Figure 3).

3.1 Background on the LBS Model

A deformable mesh can have thousands of vertices, resulting in a large number of degree of freedoms (DoFs). However, in a typical animation, all vertices must move in a spatially coherent way; individual vertices never move independently with respect to their neighbors. This suggests that the complexity of a plausible mesh animation is much less than its geometric complexity. Similar to [Der et al. 2006], we exploit the correlation of vertex movement using an LBS model.

An LBS model consists of a set, $\mathcal{B}$, of bones, each of which is described by an affine transformation $\{\mathsf{R}_b, \boldsymbol{t}_b\}$. A deformed vertex $\boldsymbol{v}_i$ is then transformed from its undeformed point $\bar{\boldsymbol{v}}_i$ to

$$\boldsymbol{v}_i = \sum_{b\in\mathcal{B}} w_{ib}(\mathsf{R}_b\bar{\boldsymbol{v}}_i + \boldsymbol{t}_b), \tag{1}$$

where w_{ib} are pre-defined skinning weights, which are constant scalars throughout the entire animation. In this paper, we simply refer $\{\mathsf{R}_b, \boldsymbol{t}_b\}$ as the *position* of a bone, b. Notice that although R_b can be any affine matrices, we start from an LBS model with only rigid rotations, R_b, because a rotation matrix R_b together with a translation $\boldsymbol{t}_b$ defines an orthogonal local frame of reference for its LBS bone. These well-defined local frames of reference provide convenience for our animation synthesis method. Later in Section 3.3, we will relax R_b and allow them to be general affine matrices.

Although an LBS model can be manually created, provided an input animation, we infer it automatically following the recent method in [Le and Deng 2012]. This method guarantees that R_b is always pure rotations, and that the unit constraint (i.e., $\sum_{b\in\mathcal{B}} w_{ib} = 1$), an important property for our method as explained in Section 3.3, is satisfied. The output of LBS precomputation includes (i) the number of bones, $|\mathcal{B}|$, (ii) the skinning weights w_{ib} for every bone and vertex, and (iii) a time series of bone positions $\{\mathsf{R}_b(t), \boldsymbol{t}_b(t)\}, t = 1\ldots N$ at each frame of the input animation.

3.2 Identifying Smooth Transitions

The first step of our method is to identify transition frames. Namely, we need to find pairs of frames, (f_i, f_j), whose spacetime states are close enough for creating a smooth transition from f_i to f_j. First, we define a transition cost function $D(i, j)$ by incorporating the following two components.

Local-Frame Bone Position A mesh deformation is invariant up to a rigid transformation. In other words, when splicing two animation clips, we can apply any rigid transformation to either clip without changing the mesh deformation sequences. Therefore, when evaluating a transition cost, we need a compatible frame of reference for comparing the states of two animation frames. For this purpose, we select an LBS bone as a *reference bone*, b_r, and compute the positions of other bones b_i (referred as *non-reference* bones) with respect to the local frame of reference of b_r for each animation frame t (see Figure 5):

$$\begin{aligned} \mathsf{R}_{i\to r}(t) &= \mathsf{R}_r^T(t)\mathsf{R}_i(t), \text{ and} \\ \boldsymbol{t}_{i\to r}(t) &= \mathsf{R}_r^T(t)\left[\boldsymbol{t}_i(t) - \boldsymbol{t}_r(t)\right]. \end{aligned} \tag{2}$$

While we can use any LBS bone as the reference bone, b_r, to get reliable results in practice we always choose the most influential bone that has the maximum sum of weights $\sum_{i\in\mathcal{V}} w_{ib}$ as b_r, where $\mathcal{V}$ is the set of all mesh vertices. This local-frame translation (2) guarantees the elimination of any rigid transformation on the entire mesh; it also eases the creation of a smooth transition in section 3.3.

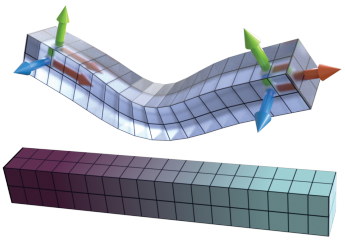


Figure 4: A simple LBS model *is illustrated using a bar deformed by two bones (top). The vertex-bone weights are color-mapped on the mesh (bottom).*

Reduced Bone Coordinates Next, we represent all the local-frame bone positions, $\{\mathsf{R}_{i\to r}(t), \boldsymbol{t}_{i\to r}(t)\}$, using reduced coordinates. Since the rotations are in a nonlinear Lie group, $\mathbf{SO}(3)$, we use the matrix logarithm function to map them into the corresponding Lie algebra, $so(3)$, of skew symmetric matrices, and represent them using 3×1 axis-angle vectors. Assembling all N example frames together, we construct a matrix A_r as follows:

$$\mathsf{A}_r = \begin{bmatrix} \boldsymbol{p}_1^1 & \boldsymbol{p}_1^2 & \cdots & \boldsymbol{p}_1^N \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{p}_{|\mathcal{B}|}^1 & \boldsymbol{p}_{|\mathcal{B}|}^2 & \cdots & \boldsymbol{p}_{|\mathcal{B}|}^N \end{bmatrix}, \text{ where } \boldsymbol{p}_i^j = \begin{bmatrix} \boldsymbol{t}_{i\to r}(j) \\ \Phi\left(\mathsf{R}_{i\to r}(j)\right) \end{bmatrix}. \tag{3}$$

Here $\Phi\left(\mathsf{R}_{i\to r}(j)\right)$ denotes the 3×1 axis-angle representation of rotation matrix $\mathsf{R}_{i\to r}(j)$, and hence $\boldsymbol{p}_i^j$ is a 6×1 vector describing a single bone position at frame j. Next, we compute a truncated SVD [Golub and Van Loan 1996] (thresholded at 0.005),

$$\mathsf{A}_r \approx \mathsf{U}_r\mathsf{S}_r\mathsf{V}_r^T = \mathsf{U}_r\mathsf{Q}_r. \tag{4}$$

Then each column vector $\boldsymbol{q}_i, i = 1\ldots N$ in the matrix Q_r, is a reduced-coordinate vector that compactly describes the mesh deformation at frame i and is not affected by any rigid transformation. In all our examples, $\boldsymbol{q}_i$ has a length no larger than 38, enabling fast evaluations of our transition cost function.

Transition Cost Function After laying out the requisite terminology, we have the ingredients to define the transition cost function $D(i, j)$ between frame i and j. We use a squared state-space distance metric,

$$D(i,j) = \|\boldsymbol{q}_i - \boldsymbol{q}_j\|_2^2 + \alpha\|\dot{\boldsymbol{q}}_i - \dot{\boldsymbol{q}}_j\|_2^2 + \beta\|\boldsymbol{r}_i - \boldsymbol{r}_j\|_2^2, \tag{5}$$

Here the first two terms are similar to the metric used in [James and Fatahalian 2003]. $\boldsymbol{q}_i$ is a reduced-coordinate vector as computed in (4); $\dot{\boldsymbol{q}}_i$ is computed using a finite difference approximation (i.e., $\dot{\boldsymbol{q}}_i \approx \boldsymbol{q}_i - \boldsymbol{q}_{i-1}$). The third term is to reflect the change of the

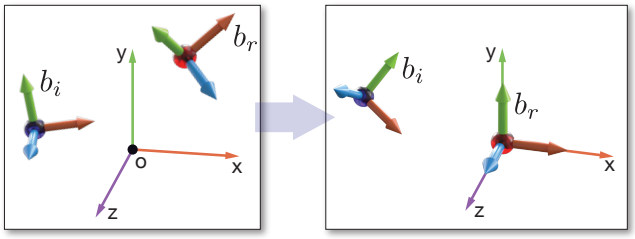


Figure 5: Local-Frame Bone Positions: *The LBS bone positions in the world frame (left) are translated into the local frame of reference of the reference bone b_r (right). We evaluate our space-time difference metric and create smooth transitions in the local frame of b_r, allowing arbitrary rigid transformations to be applied on animation segments while preserving spatiotemporal correlations.*

reference bone b_r at a transition. Since the reference bone at frame i and j can always perfectly match under a rigid transformation, we only consider the difference of its first derivatives at i and j. Therefore, we define $\boldsymbol{r}_i$ as the position of b_r at animation frame $i-1$ with respect to its local frame of reference at animation frame i. Similar to $\boldsymbol{p}_i^j$ in (3), $\boldsymbol{r}_i$ is the 6×1 vector

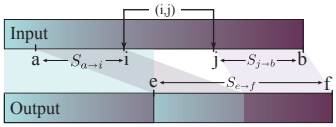
$$\boldsymbol{r}_i = \begin{bmatrix} \mathsf{R}_r^T(i)[\boldsymbol{t}_r(i-1) - \boldsymbol{t}_r(i)] \\ \Phi(\mathsf{R}_r^T(i)\mathsf{R}_r(i-1)) \end{bmatrix}. \tag{6}$$

The weights, α and β, balance the three terms so that none of them dominates the others, and in our implementation we use $\alpha = \frac{\max_{i=1..N}\|\boldsymbol{q}_i\|_2^2}{\max_{i=1..N}\|\dot{\boldsymbol{q}}_i\|_2^2}$, and $\beta = \frac{\max_{i=1..N}\|\boldsymbol{q}_i\|_2^2}{\max_{i=1..N}\|\boldsymbol{r}_i\|_2^2}$.

Finally, we evaluate $D(i,j)$ for every pair of frames (i,j) of the input animation, and only keep the pairs whose cost function value is below a threshold (i.e., $D(i,j) < T_c$) as our candidate transitions. The threshold T_c can be controlled by a user. A small value results in less variations in output animations; a large value produces more distinct results, but may potentially introduce transition artifacts. In all our examples, we use $T_c = 40 \max_{i=1..N} D(i, i-1)$.

3.3 Creating Smooth Transitions

Let $S_{a\to i}$ denote a segment of input animation from frame a to i, and $S_{j\to b}$ denote another segment from frame j to b. If (i,j) is one of our candidate transitions, we are ready to splice these segments together (see Figure 3) to form a new animation segment $S_{e\to f}$, where $f - e = i - a + b - j$.

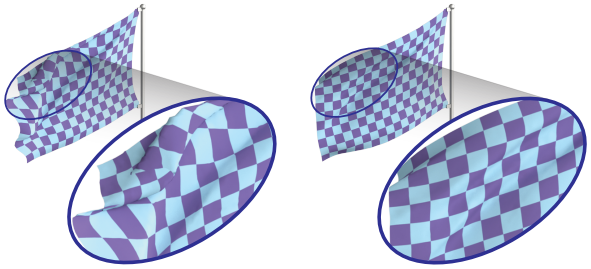


We first match b_r's position at the transition (i,j). For every frame of $S_{a\to i}$, we simply copy b_r's position into the output, $S_{e\to f}$, namely, $\tilde{\mathsf{R}}_r(e+k) = \mathsf{R}_r(a+k)$ and $\tilde{\boldsymbol{t}}_r(e+k) = \boldsymbol{t}_r(a+k)$ for $k = 0...(i-a)$. For every frame of $S_{j\to b}$, we apply a fixed rigid transformation to its time series of b_r to match it against the last frame of $S_{a\to i}$,

$$\begin{aligned} &\tilde{\mathsf{R}}_r(e+i-a+k) = \mathsf{R}_r(i)\mathsf{R}_r^T(j)\mathsf{R}_r(j+k), \text{ and} \\ &\tilde{\boldsymbol{t}}_r(e+i-a+k) = \mathsf{R}_r(i)\mathsf{R}_r^T(j)[\boldsymbol{t}_r(j+k) - \boldsymbol{t}_r(j)] + \boldsymbol{t}_r(i), \end{aligned} \tag{7}$$

for $k = 0...(b-j)$. Notice that because of the third term in our transition cost function (5) this rigid transformation is sufficient to achieve a smooth velocity change of b_r as well.

Next, consider the non-reference bones $b_i \neq b_r$. We now splice their motions in both segments so that they match exactly at the transition (i,j) and meanwhile preserve spatiotemporal correlations. One straightforward approach is to linearly blend the bone coordinates or solve a 1D Poisson equation for every bone position coordinate $\{\mathsf{R}_i(k), \boldsymbol{t}_i(k)\}$ with fixed boundary values at a and b and an equality constraint at (i,j). Unfortunately, such a simple scheme strives to keep temporal smoothness, but neglects spatial correlations of the bone positions, leading to undesirable artifacts as shown in Figure 6 and supplemental video. Notice that simple linear blending causes no problems in character animations [Kovar et al. 2002; Arikan and Forsyth 2002], because those methods use more input frames, each with a small number of DoFs for representing a character motion, and hence it is possible to find close enough transitions. It also works well in Mesh Ensemble Motion Graphs [James et al. 2007], because it assumes moderate deformations, and numerous ensembles with asynchronous transitions allow effective diffusion of transition artifacts in space-time.



(a) Poisson Blending (b) Our Method

Figure 6: Comparison of Blending Methods: *Simple blending near transition frames using Poisson solves leads to unpleasant artifacts (left). Our method (right) based on sampled deformation gradients preserves spatiotemporal coherence and produces more natural transitions (also refer to supplemental video). Input animation comes from [Briceño et al. 2003].*

Inspired by gradient domain shape editing techniques [Yu et al. 2004; Sumner et al. 2005; Der et al. 2006], we preserve spatial coherence of bone positions by solving the bone positions based on blended mesh deformation gradients. To overcome geometric complexity, we avoid evaluating deformation gradients on the entire mesh; instead we compute them at a set of selected vertices. This strategy is closely related to the ones used by [Der et al. 2006] and [Meyer and Anderson 2007].

Selection of Key Vertices Concretely, we select a set $\mathcal{V}$ of mesh vertices based on the learned LBS model. For each bone b, we add into $\mathcal{V}$ one vertex v_b that has the maximum weight associated with b (i.e., $v_b = \arg\max_{v\in\mathcal{V}} w_{vb}$). In addition, for each pair of bones (b_i, b_j), we select a vertex v_{ij} using $v_{ij} = \arg\max_{v\in\mathcal{V}} w_{vi}w_{vj}$, and add it into $\mathcal{V}$ if both b_i and b_j have non-zero weights on it (i.e., $w_{vi}w_{vj} > 0$). Intuitively, these vertices serve as coupling points to capture how the bones move spatially coherently.

Deformation Gradients at Key Vertices We precompute a deformation gradient for each vertex $v \in \mathcal{V}$ at every input animation frame. Since we have applied a rigid transformation to align the reference bone b_r in (7), we compute vertex deformation gradients also in the local frame of reference of b_r, so we can use them to estimate the local-frame bone positions later. In particular, given a set of local-frame bone positions, $\{\mathsf{R}_{b\to r}(t), \boldsymbol{t}_{b\to r}(t)\}, b = 1...|\mathcal{B}|$, at an animation frame t, the resulting local-frame deformation gradient at a key vertex $\boldsymbol{v}_i$ is derived by differentiating (1),

$$\frac{\partial \boldsymbol{v}_i(t)}{\partial \bar{\boldsymbol{v}}_i} = \sum_{b\in\mathcal{B}} (\mathsf{R}_{b\to r}(t)\bar{\boldsymbol{v}}_i + \boldsymbol{t}_{b\to r}(t)) \frac{\partial w_{ib}}{\partial \bar{\boldsymbol{v}}_i} + w_{ib}\mathsf{R}_{b\to r}(t). \tag{8}$$

Caution needs to be taken when precomputing the spatial derivatives, $\frac{\partial w_{ib}}{\partial \bar{\boldsymbol{v}}_i}$, because the LBS model only defines w_{ib} on the mesh

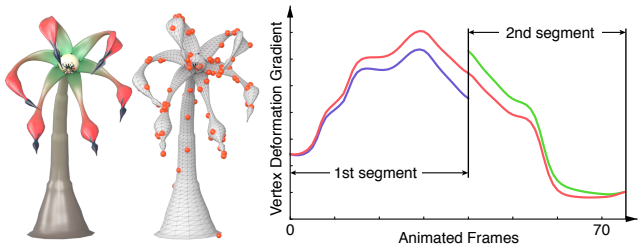


Figure 7: Blending Deformation Gradients of Key Vertices: *Given the mesh of a feeding creature (left), we select a set of key vertices (middle). When splicing two segments, we blend the time series of deformation gradients of key vertices from both segments (shown by blue and green curves) by solving 1D Poisson equations. The resulting smooth time varying deformation gradients (red curve) is then used in* (9) *for solving bone positions.*

surface not in the 3D space. To avoid artifacts for estimating bone positions in the next step, we need to guarantee a unit sum of bone weights (i.e., $\sum_{b\in|\mathcal{B}|} w_{ib} = 1$) when estimating the LBS model, and also ensure $\sum_{b\in|\mathcal{B}|} \frac{\partial w_{ib}}{\partial \bar{\boldsymbol{v}}_i} = 0$, since both properties guarantee a deformation gradient to be an affine matrix F when the entire mesh undergoes an affine transformation F. Otherwise, an undeformed mesh still results in non-identity deformation gradients. Unfortunately, computing $\frac{\partial w_{ib}}{\partial \bar{\boldsymbol{v}}_i}$ using finite element approximations on arbitrary meshes as used in [Der et al. 2006] cannot guarantee the latter property due to the mesh irregularity. Instead, we propose to compute $\frac{\partial w_{ib}}{\partial \bar{\boldsymbol{v}}_i}$ for all key vertices using a constrained least-square solver which robustly ensures both properties. Detailed derivations are presented in Appendix A.

Representation of Deformation Gradient Each deformation gradient is a 3×3 matrix $\mathsf{D}_i(t) = \frac{\partial \boldsymbol{v}_i(t)}{\partial \bar{\boldsymbol{v}}_i}$ describing local mesh rotations and stretches at frame t. To create smooth transitions, we synthesize new vertex deformation gradients at the frames nearby the transition location. Since a deformation gradient contains nonlinear rotational part, simple blending can lead to artifacts. Similar to [Sumner et al. 2005; Huang et al. 2011], we represent each $\mathsf{D}_i(t)$ in rotation-strain coordinates by computing the polar decomposition [Golub and Van Loan 1996], $\mathsf{D}_i(t) = \mathsf{U}_i(t)\mathsf{S}_i(t)$, where $\mathsf{U}_i(t)$ is a rotation matrix, and $\mathsf{S}_i(t)$ is a symmetric stretch matrix. The rotation-strain coordinates of $\mathsf{D}_i(t)$ is a 9×1 vector $\boldsymbol{d}_i(t)$, in which the first 3 elements is the axis-angle coordinates $\Phi(\mathsf{U}_i(t))$, and the rest 6 elements stack the upper triangle part of $\mathsf{S}_i(t)$. In summary, at the end of the precomputation stage, we have a time series of $\boldsymbol{d}_i(t)$ for each key vertex $v_i \in \mathcal{V}_k$.

Solving Local-Frame Bone Positions Now we proceed to compute bone positions for creating smooth transitions. Recall that our goal is to form a new animation segment $S_{e\to f}$ using $S_{a\to i}$ and $S_{j\to b}$. We first blend the rotation-strain coordinates $\tilde{\boldsymbol{d}}_i(t)$ on time axis by solving a 1D Poisson equation for each coordinate component (see Figure 7),

$$\tilde{\boldsymbol{d}}_i''(t) = \boldsymbol{g}(t), t = e...f, \text{ s.t. } \tilde{\boldsymbol{d}}_i(e) = \boldsymbol{d}_i(a) \text{ and } \tilde{\boldsymbol{d}}_i(f) = \boldsymbol{d}_i(b),$$

where $\boldsymbol{g}(t)$ is computed using the finite difference approximation at the corresponding frames of the precomputed $\boldsymbol{d}_i(t)$. Each 1D Poisson solve amounts to solving a tridiagonal linear system. Next, we compute the bone positions based on deformation gradients $\tilde{\mathsf{D}}_i(t)$ constructed from $\tilde{\boldsymbol{d}}_i(t)$. As derived in [Der et al. 2006], the deformation gradient in (8) can be written as a linear operator with respect to the bone positions $\{\mathsf{R}_{b\to r}(t), \boldsymbol{t}_{b\to r}(t)\}$; hence (8) is of

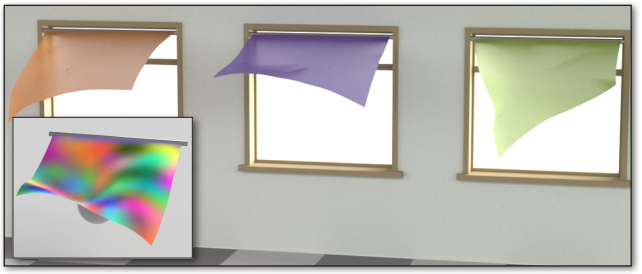


Figure 8: Avoiding collisions: *We detect collisions while splicing animation segments (as described in §3.4), resulting in 60 different blowing clothes from a single example. While the cloth in the provided animation (shown in the inset) may flap down and touch a sphere, none of the synthesized clothes swings down and gets deformed by the sphere that does not exist in the output scene (see the video for all clothes).*

a form $\boldsymbol{d} = \mathsf{G}\boldsymbol{p}$, where $\boldsymbol{d}$ stacks the deformation gradients D_i at key vertices into a vector, and $\boldsymbol{p}$ stacks all the bone positions. At each output frame $t, t = e...f$, provided the blended deformation gradients $\tilde{\mathsf{D}}_i(t)$, we compute bone positions $\{\tilde{\mathsf{F}}_{b\to r}(t), \tilde{\boldsymbol{t}}_{b\to r}(t)\}$ for all the non-reference bones by stacking $\tilde{\mathsf{D}}_i(t)$ into a vector $\tilde{\boldsymbol{d}}$ and solving a least-square system,

$$\mathsf{G}\boldsymbol{p} = \tilde{\boldsymbol{d}}. \tag{9}$$

Notice that here we use $\tilde{\mathsf{F}}_{b\to r}$ to indicate that from now on a non-reference bone has general affine matrices instead of pure rotations, because the least-square solves have no guarantee to ensure pure rotations in the results. Nevertheless, there is no problem to use them in the LBS model for computing mesh deformations. Now that the solved bone positions are in the local frame of the reference bone b_r, we finally translate them into the world frame based on the synthesized reference bone positions. In particular, we compute

$$\begin{aligned} &\tilde{\mathsf{F}}_b(k) = \tilde{\mathsf{R}}_r(k)\tilde{\mathsf{F}}_{b\to r}(k), \text{ and} \\ &\tilde{\boldsymbol{t}}_b(k) = \tilde{\mathsf{R}}_r(k)\tilde{\boldsymbol{t}}_{b\to r}(k) + \tilde{\boldsymbol{t}}_r(k), \ k = e...f. \end{aligned} \tag{10}$$

At this point, we have the basic ingredients to generate various animations: we randomly select segments that start and end at transition frames and splice them together using the presented algorithm.

3.4 Constrained Animation Synthesis

While our primary goal is to produce many different animations from a single example, our method can easily incorporate user-specified constraints. With selected candidate transitions, a motion graph is constructed similar to approaches for character animation synthesis [Kovar et al. 2002; Lee et al. 2002]: each node of the graph represents a segment of the input animation, and each edge corresponds to a candidate transition. The algorithm of searching on the graph is flexible, depending on specific applications. The graph walking strategies in character motion synthesis can of course be naturally applied. In what follows, we present the details of our graph walking implementation.

Our implementation incorporates two types of constraints, targeted frames and collision avoidance. Targeted frames specify that a set of frames $f_i, i = 1...M$ from the input animation must appear at frame $\tilde{f}_j, j = 1...M$ in the output animation. Collision avoidance requires that the synthesized deformable mesh should not interpenetrate with other objects in the environment. Notice that it is unlikely that our single-component synthesis introduces self-collisions if the input animation contains no self-collisions, because it reuses input animation frames with only moderate changes near transitions.

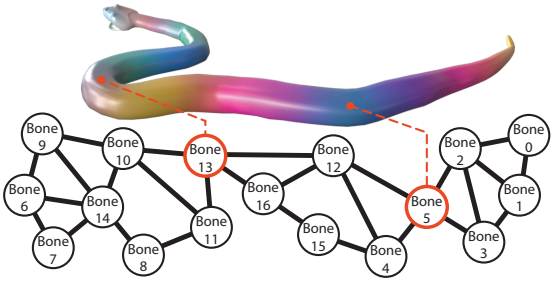


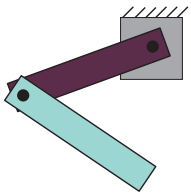
Figure 9: A bone graph *is illustrated using a deformable snake from [Briceño et al. 2003]. (Top) The snake is colormapped according to its LBS bone distribution. (Bottom) The corresponding bone graph consists 17 bones with 2 cut bones in red circles.*

The targeted frames split the output time axis into intervals. For each interval, the start and end frames are given. We use multi-start bidirectional search [Russell and Norvig 2009] to find sequences of input animation segments forming the desired intervals. There are multiple start and end nodes on the graph, because it is likely that many input segments contain the same targeted frame. When searching on the graph, we conduct AABB-tree-based collision detection, and discard the current searching path if the newly created transitions introduces collisions. We also label each input mesh deformation frames in which a collision occurs and avoid using these frames in the synthesis (see Figure 8). Additionally, we compare the current path with existing results. If it overlaps largely with some existing result, we discard the current search to avoid generating animations utterly alike each other. Lastly, it is possible that the start frame or the end frame or both of them are not specified for an interval, then bidirectional search is unnecessary, and we simply conduct a random walk on the graph to generate diverse results.

4 Multi-Component Animation Synthesis

In this section, we extend the presented single-component animation synthesis algorithm to create different transitions at multiple spatial regions of the deformable mesh. Then, individual synthesis for different regions allows their animations to be assembled asynchronously, resulting in more diverse output.

Why Is Multi-Component Asynchronous Synthesis Reasonable? For a complex deformable mesh, there usually exist multiple weakly coupled, if not completely independent, deformable regions. For example, consider the upper and lower body of a dancer, both parts can deform independently in an animation sequence. This suggests that their animations can be synthesized separately. Moreover, from the perspective of the dynamic system analysis [Strogatz 2000], deformation dynamics with a large number of DoFs very often exhibits both periodic and chaotic behaviors. We detect transitions by essentially identifying periodic deformations. Meanwhile, motions cross multiple regions can experience chaotic dynamics (such as the classic double pendulum), in which very little coordination among these regions is exhibited. Therefore, animations of those regions can be synthesized independently and assembled without noticeable artifacts. Our key goal in this section is to detect those independently deformed mesh regions, synthesize their animations asynchronously and assemble them seamlessly.



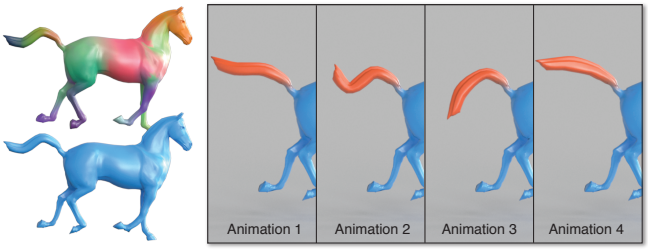


Figure 10: Multi-component synthesis method *produces a variety of animations even for a perfectly periodic horse gallops from [Sumner and Popović 2004]. Single component synthesis (Left bottom) generates no variations for the input (left top). Multi-component synthesis automatically separates the tail from the body, and produces different tail motions (right) while the body galloping motion is still perfectly periodic (also refer to the video).*

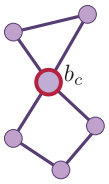
4.1 Decomposition of LBS Bones

To harness this basic insight, our first step is to identify independent regions. Fortunately, we find that the structure of LBS bones provides a helpful abstraction to this end.

Bone Graph First, we create a bone graph. We construct a weighted undirected graph where each node corresponds to an LBS bone. There exists an edge e_{ij} between node i and node j if and only if there is at least one vertex v that has non-zeros weights with respect to both bones (i.e., $w_{vi}w_{vj} > 0$). The corresponding weight d_{ij} is set as

$$d_{ij} = \max_{v \in \mathcal{V}} w_{vi} w_{vj}. \tag{11}$$

A large weight d_{ij} suggests that at least one vertex is heavily affected by bone i and j. Consequently, when we synthesize bone motions, both bones must move in a correlated way to deform their associated vertices properly. On the other hand, a small d_{ij} indicates that almost no vertices are affected by both bones, and hence they can move independently. As illustrated in Figure 9, the structure of a bone graph reflects the coupling between LBS bones, thus helping for identifying weakly coupled or independent components.



Cut Nodes We then cut off the bone graph edges that indicate very weak coupling. We discard the edges whose weights are smaller than a threshold T_e. For all the examples, we always use $T_e = 0.1 \max_{i,j} d_{ij}$ without breaking the graph into disconnected components. Next, we find all *cut nodes* (or *cut vertices*) of the resulting graph. A node is called a cut node if removing this node from the graph results in separated components (see Figure. 9). Our implementation follows the efficient algorithm [Thulasiraman and Swamy 1992] for finding all cut nodes in a complexity of $O(n + m)$, where n and m are respectively the number of nodes and edges in the graph. Suppose that b_c is a cut node, which, after being removed, separates the graph into two components $\mathcal{B}_1$ and $\mathcal{B}_2$. In this case, we have two independent bones components, $\mathcal{B}_1 \cup \{b_c\}$ and $\mathcal{B}_2 \cup \{b_c\}$. In summary, by analyzing the bone graph, we identify multiple components separated by cut nodes.

4.2 Assembling Multi-Component Animations

Let $\mathcal{C}_i, i = 1...T$ denote the resulting weakly connected components of the bone graph. We independently synthesize the bone motions of each component following the presented algorithm in Section 3. Suppose for each $\mathcal{C}_i$, we have synthesized m_i animations, each with a length of M frames. We now present the details

Algorithm 1: merge bone animations of component $\mathcal{C}_i$ and $\mathcal{C}_j$

procedure: `merge_bone_animation` $(\mathcal{C}_i, \mathcal{C}_j, b_c)$
begin
  foreach m_i *of synthesized bone animations for component* $\mathcal{C}_i$ **do**
    foreach m_j *of synthesized bone animations for component* $\mathcal{C}_j$ **do**
      compute $D_c(i,j)$ using (12)
      if $D_c(i,j) \geq T_c$ **then return**
      foreach *frame* $f = 1...M$ **do**
        copy the sequence of bone positions in $\mathcal{C}_i$
        update b_c's positions in the resulting animation using (13)
        foreach *bone* $b \in \mathcal{C}_j, b \neq b_c$ **do**
          compute b's positions in the resulting animation using (14)

of assembling them together to produce more variations.

For each pair of components $\mathcal{C}_i$ and $\mathcal{C}_j$, if there exists a cut node b_c separating them (i.e., $\{b_c\} = \mathcal{C}_i \cap \mathcal{C}_j$), we combine their motions based on b_c's position at each animation frame. As outlined in Algorithm 1, consider the independently synthesized bone motions of $\mathcal{C}_i$ and $\mathcal{C}_j$. Let $\{\mathsf{F}_{c,i}(t), \boldsymbol{t}_{c,i}(t)\}$ and $\{\mathsf{F}_{c,j}(t), \boldsymbol{t}_{c,j}(t)\}, t = 1...M$ denote respectively the time series of b_c's positions in both components. We are able to combine the motions of both components naturally if b_c's motions are similar up to a rigid transformation. In particular, we compute polar decompositions, $\mathsf{F}_{c,i}(t) = \mathsf{U}_{c,i}(t)\mathsf{S}_{c,i}(t)$ and $\mathsf{F}_{c,j}(t) = \mathsf{U}_{c,j}(t)\mathsf{S}_{c,j}(t)$, and a difference metric,

$$D_c(i,j) = \sum_{t=1}^{M} \|\mathsf{S}_{c,i}(t) - \mathsf{S}_{c,j}(t)\|_F^2. \tag{12}$$

We combine the two motions if $D_c(i,j) < T_c$, where T_c is a constant for balancing the diversity and plausibility of the results. In practice, we use $T_c = 0.05 \max_{t=1..F}\{\|\mathsf{S}_{c,i}(t)\|_F^2, \mathsf{S}_{c,j}(t)\|_F^2\}$. When merging them, we use the motions from one component, $\mathcal{C}_i$, as an "anchor", and match the motions of $\mathcal{C}_j$ against it using rigid transformations. Namely, we first update b_c's position using

$$\mathsf{F}_c(t) = \frac{1}{2}\mathsf{U}_{c,i}(t)\,(\mathsf{S}_{c,i}(t) + \mathsf{S}_{c,j}(t)) \text{ and } \boldsymbol{t}_c(t) = \boldsymbol{t}_{c,i}(t), \tag{13}$$

Here simply averaging the corresponding stretch matrices from $\mathcal{C}_i$ and $\mathcal{C}_j$ as b_c's output stretch matrix produces no artifacts, since our selected T_c ensures that they are close enough. All the other bone motions in $\mathcal{C}_i$ are unchanged. And the motion of a bone $b \in \mathcal{C}_j, b \neq b_c$ in the combined animation is computed by

$$\begin{aligned} &\mathsf{F}_b(t) = \mathsf{U}_{c,i}(t)\mathsf{U}_{c,j}^T(t)\mathsf{F}_{b,j}(t), \text{ and} \\ &\boldsymbol{t}_b(t) = \mathsf{U}_{c,i}(t)\mathsf{U}_{c,j}^T(t)[\boldsymbol{t}_{b,j}(t) - \boldsymbol{t}_{c,j}(t)] + \boldsymbol{t}_{c,i}(t). \end{aligned} \tag{14}$$

Here at each animation frame t, all the bones $b \in \mathcal{C}$ are applied the same rigid transformation, and hence their spatial correlation is preserved. These rigid transformations are temporally coherent, since they are computed from the continuously changing positions of b_c. As a result, the resulting merged bone motions are temporally correlated as well.

The merged components $\mathcal{C}_i$ and $\mathcal{C}_j$ form a new component $\mathcal{C}_{ij}$, which is then used to merge with another component if they share a cut node. We repeat this procedure until all the components are assembled together. From there we know all the bone positions in a synthesized sequence, and finalize the resulting deformable mesh animation. During this process, each merge produces a number of different animations (see Figure 10). Meanwhile, we check if the user-specified constraints are satisfied by the merged animations. In our implementation, we optionally discard results that introduce any self-collisions.

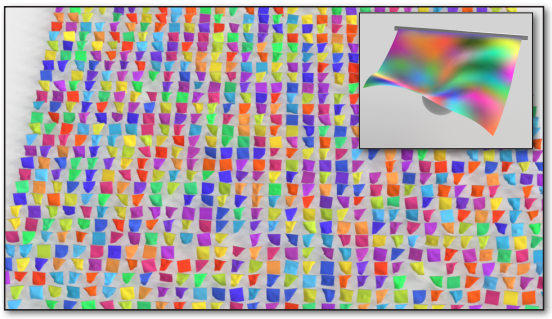


Figure 11: 1000 different blowing clothes *are synthesized from a single example shown in the inset where the colormap indicates the LBS bone distribution.*

5 Results

We have applied our method to a range of animations, including available animations of previous work [Briceño et al. 2003; Sumner and Popović 2004; Zheng and James 2012] as well as our home-brew animations, including cloth simulations, human and animal animations, and flexible objects. The results are synthesized on an Intel quad-core i7-3770 (3.4 GHz) processor with 8 GB RAM, and are summarized in Table 1. Our implementation exploited possible parallelism in the synthesis using Intel's thread building block library.

Constrained Synthesis We demonstrate the collision avoidance in our animation synthesis using the blowing cloth example (see Figure 8). The targeted frames are demonstrated using the dancers (see Figure 13), for which we enforce all the animations start from the same frame and end at another frame. This produces 1000 dancers that start and end at the same poses but dance differently in between (refer to the supplemental video).

Cloth Animations Only a single component on the bone graph is detected for all our cloth animations (see Figure 11 and 12). This is because clothes usually experience large deformations, and we need many correlated LBS bones to approximate their animations. The resulting bone graphs tend to be highly connected, raising a challenge on maintaining spatial coherence in the resulting animation (see a comparison in Figure 6). Consequently, for cloth animations with many small scale deformations (such as the wrinkles in Figure 12), only a small number of plausible transition pairs are detected, resulting in fewer variations for the flag example.

Multi-Component Synthesis We identified weakly coupled components for the galloping horse, the dancer, and feeding crea-

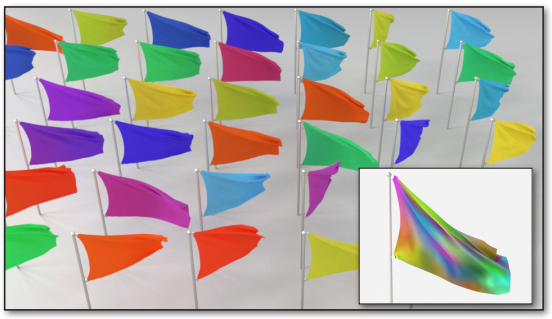


Figure 12: 36 flags *flap differently in a wind.*

Example	Input Complexity			Analysis					Synthesis		
	Frames	Vtx	Tri	Bones, $\|\mathcal{B}\|$	t_{LBS}	Trans	Components	t_{analy}	Anim	Frames	t_{synth}
flag*	210	6909	13436	50	20.4 min	54	1	3.8 min	36	360	3.6 sec
dancer 1*	180	7061	14118	27	11.2 min	58	1	2.1 min	1000	600	80.6 sec
dancer 2*	180	7061	14118	27	11.2 min	98	3	3.2 min	1000	600	162.7 sec
horse†	48	8431	16843	22	9.6 min	24	2	1.4 min	4	336	1.2 sec
flowing cloth‡	230	25921	51200	46	32.6 min	204	1	3.5 min	1000	480	20.4 sec
cloth window‡	230	25921	51200	46	32.6 min	204	1	3.5 min	60	480	296.3 sec
feeding creature	280	21721	38458	91	16.5 min	526	26	9.7 min	1000	750	203.5 sec

Table 1: Example statistics *for # input animation frames, the mesh complexity (# vertices and # triangles), # LBS bones ($|\mathcal{B}|$), LBS computation time (t_{LBS}), # transitions, # weakly coupled components, animation analysis time (t_{analy}), # different output animations, # output frames, and the synthesis time (t_{synth}). The rows of* dancer 1 *and* dancer 2 *are respectively for single-component and multi-componenent synthesis, both using the same input example. The multi-component transition number is the total number from all individual components. We indicate examples from prior work using $\star$ for [Briceño et al. 2003], † for [Sumner and Popović 2004], and ‡ for [Zheng and James 2012].*

Figure 13: 1000 dancers *are synthesized from a single example. The output dancers exhibit different dancing sequences.*

ture examples. The input of galloping horse consists of 48 frames forming two perfectly periodic galloping cycles. Single component synthesis detect transitions connecting corresponding frames into both cycles, resulting in no variations. In contrast, multi-component synthesis separates the tail from the horse body, and detect more transitions for the tail part (see Figure 10). To synthesize the 1000 dancers (see Figure 13), we use an extra constraint that allows the reference bones to be rigidly transformed only on the ground plane (i.e., the X-Y plane), so that the synthesized dancers always move on the ground.

6 Conclusion

We have presented a method for synthesizing deformable mesh animations from a single input animation sequence. Requiring no knowledge of the models for creating the input animation, our method can be used for generating a wide range of mesh animations without repeating the creation process of the source animation. Our experiments suggest that many mesh animations can be decoupled. Reusing those individual components asynchronously is an efficient way to produce various animations with user-specified constraints.

Limitations and Future Work The main contribution of this work is to identify transitions and splice mesh animations smoothly. This method works well when the input animation exhibits periodic and ideally chaotic motions. Our multi-component analysis decomposes the mesh into smaller components and hence is able to detect even locally periodic motions for constructing motion graphs. In the worst case, if the input animation shows a sequence of very distinct deformations, then the resulting synthesized animations are quite limited. Our current synthesis ignores the interaction of the resulting animations with the surrounding environment. For example, the resulting cloth animation cannot response correctly against wind forces. This problem can be alleviated by introducing high-level constraints in the graph search. And there is much room for further improving the efficiency of searching the created motion graphs to satisfy different types of constraints. In general, data-driven animation synthesis has the difficulty of exploring new mesh deformations out of the scope of input animations. Multi-component synthesis is able to produce new deformations by combing different components asynchronously, but is limited to a few identifiable components. Another promising future direction is to combine this method with physics-based simulations: simulation is performed only when we cannot reuse existing results, and hence produces new deformations. Finally, it could be interesting to explore other types of animations (such as fluids) with the same idea of example-based synthesis.

Acknowledgments We would like to thank the anonymous reviewers for their constructive feedback. We also thank Xiaochen Hu for his help of early testing implementation, Papoj Thamjaroenporn for proofreading and improving a few figures, and Eitan Grinspun for his feedback. This work was supported by Columbia University young faculty startup fund.

A Spatial Derivative of Bone Weights

Evaluating vertex deformation gradients using equation (8) requires the precomputation of spatial derivatives of bone weights, $\frac{\partial w_{ib}}{\partial \bar{\boldsymbol{v}}_i}$, at selected key vertices. If the entire mesh undergoes an affine transformation, all the LBS bones have the same position $\{\mathsf{F}, \boldsymbol{t}\}$, and the deformation gradient must be F. The property is guaranteed in (8) only when

$$\sum_{b\in|\mathcal{B}|} \frac{\partial w_{ib}}{\partial \bar{\boldsymbol{v}}_i} = 0 \tag{15}$$

is ensured in our estimation of $\frac{\partial w_{ib}}{\partial \bar{\boldsymbol{v}}_i}$. Consider a key vertex v_i that has P neighbor vertices $v_j, j = n_1..n_P$. For an arbitrary irregular mesh, finite element approximation considering the difference of weights between v_i and v_j cannot guarantee the satisfaction of (15) by the resulting derivative estimations. Our new approach starts from the approximation using the definition of scalar field spatial derivative:

$$\frac{\partial w_{ib}}{\partial \bar{\boldsymbol{v}}_i}(\boldsymbol{v_j} - \boldsymbol{v_i}) \approx w_{jb} - w_{ib}. \tag{16}$$

Stacking all neighbors together according to (16) yields a linear least-square equation of a form $\mathsf{A}_i \boldsymbol{d}_b = \boldsymbol{c}_b$ for a single bone b. We weight each row of the system by the inverse of $\sqrt{\|\boldsymbol{v_j} - \boldsymbol{v_i}\|_2^2}$, so a closer neighbor results in a better approximation in (16). The constraint (15) couples the bone weight derivatives together. Therefore, we need to stack together the equations for all bones, and solve a

least-square system of a form

$$\begin{bmatrix} \mathsf{A}_i & & \\ & \mathsf{A}_i & \\ & & \ddots \end{bmatrix} \boldsymbol{d} = \boldsymbol{c} \tag{17}$$

with the equality constraint (15). Here $\boldsymbol{d}$ stacks the unknown weight derivatives for all the related bones of the vertex v_i, and $\boldsymbol{c}$ stacks weight differences between v_i and all its neighbors for all the related bones. Fortunately, the size of this system is small, since in a typical LBS model, each vertex is affected by at most 4 bones, rendering a system (17) at most $4P \times 12$. We solve this constrained least-square system by projecting $\boldsymbol{d}$ into the space that satisfies (15) (using QR factorization on the system of (15)) and solving the resulting least-square problems.

References

AGARWALA, A., ZHENG, K. C., PAL, C., AGRAWALA, M., COHEN, M., CURLESS, B., SALESIN, D., AND SZELISKI, R. 2005. Panoramic video textures. *ACM Trans. on Graphics (SIGGRAPH 2005) 24*, 3 (July), 821–827.

ARIKAN, O., AND FORSYTH, D. A. 2002. Interactive motion generation from examples. *ACM Trans. on Graphics (SIGGRAPH 2002) 21*, 3 (July), 483–490.

BRICEÑO, H. M., SANDER, P. V., MCMILLAN, L., GORTLER, S., AND HOPPE, H. 2003. Geometry videos: a new representation for 3d animations. In *2003 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 136–146.

DER, K. G., SUMNER, R. W., AND POPOVIĆ, J. 2006. Inverse kinematics for reduced deformable models. *ACM Trans. on Graphics (SIGGRAPH 2006) 25*, 3 (July), 1174–1179.

GALERNE, B., LAGAE, A., LEFEBVRE, S., AND DRETTAKIS, G. 2012. Gabor noise by example. *ACM Trans. on Graphics (SIGGRAPH 2012) 31*, 4 (July), 73:1–73:9.

GLEICHER, M., SHIN, H. J., KOVAR, L., AND JEPSEN, A. 2003. Snap-together motion: assembling run-time animations. In *Proceedings of the 2003 symposium on Interactive 3D graphics*, 181–188.

GOLUB, G. H., AND VAN LOAN, C. F. 1996. *Matrix Computations*, 3rd ed. Johns Hopkins University Press, Baltimore, MD.

HUANG, J., TONG, Y., ZHOU, K., BAO, H., AND DESBRUN, M. 2011. Interactive shape interpolation through controllable dynamic deformation. *IEEE Transactions on Visualization and Computer Graphics 17*, 7, 983–992.

JAMES, D. L., AND FATAHALIAN, K. 2003. Precomputing interactive dynamic deformable scenes. *ACM Trans. on Graphics (SIGGRAPH 2003) 22*, 3 (July), 879–887.

JAMES, D. L., AND TWIGG, C. D. 2005. Skinning mesh animations. *ACM Transactions on Graphics 24*, 3 (Aug.), 399–407.

JAMES, D. L., TWIGG, C. D., COVE, A., AND WANG, R. Y. 2007. Mesh ensemble motion graphs: Data-driven mesh animation with constraints. *ACM Trans. on Graphics 26*, 4 (Oct.).

KALOGERAKIS, E., CHAUDHURI, S., KOLLER, D., AND KOLTUN, V. 2012. A probabilistic model for component-based shape synthesis. *ACM Trans. on Graphics 31*, 4 (July).

KAVAN, L., SLOAN, P.-P., AND O'SULLIVAN, C. 2010. Fast and Efficient Skinning of Animated Meshes. *Computer Graphics Forum (Eurographics 2010) 29*, 2.

KAVAN, L., GERSZEWSKI, D., BARGTEIL, A., AND SLOAN, P.-P. 2011. Physics-inspired upsampling for cloth simulation in games. *ACM Trans. on Graphics (SIGGRAPH 2011) 30*, 4.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. *ACM Trans. on Graphics (SIGGRAPH 2002) 21*, 3 (July), 473–482.

KWATRA, V., ESSA, I., BOBICK, A., AND KWATRA, N. 2005. Texture optimization for example-based synthesis. *ACM Trans. on Graphics (SIGGRAPH 2005) 24*, 3 (July), 795–802.

LAU, M., BAR-JOSEPH, Z., AND KUFFNER, J. 2009. Modeling spatial and temporal variation in motion data. *ACM Trans. on Graphics (SIGGRAPH Asia 2009) 28*, 5 (Dec.), 171:1–171:10.

LE, B. H., AND DENG, Z. 2012. Smooth skinning decomposition with rigid bones. *ACM Trans. on Graphics (SIGGRAPH Asia 2012) 31*, 6 (Nov.).

LEE, J., CHAI, J., REITSMA, P. S. A., HODGINS, J. K., AND POLLARD, N. S. 2002. Interactive control of avatars animated with human motion data. *ACM Trans. on Graphics (SIGGRAPH 2002) 21*, 3 (July), 491–500.

MARTIN, S., THOMASZEWSKI, B., GRINSPUN, E., AND GROSS, M. 2011. Example-based elastic materials. *ACM Trans. on Graphics (SIGGRAPH 2011) 30*, 4 (July), 72:1–72:8.

MEYER, M., AND ANDERSON, J. 2007. Key point subspace acceleration and soft caching. *ACM Transactions on Graphics 26*, 3 (July).

PULLEN, K., AND BREGLER, C. 2000. Animating by multi-level sampling. In *Proceedings of the Computer Animation*, CA '00.

RUSSELL, S. J., AND NORVIG, P. 2009. *Artificial Intelligence: A Modern Approach*, third ed. Prentice Hall.

SCHÖDL, A., AND ESSA, I. A. 2002. Controlled animation of video sprites. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

STROGATZ, S. H. 2000. *Nonlinear Dynamics and Chaos*. Westview.

SUMNER, R. W., AND POPOVIĆ, J. 2004. Deformation transfer for triangle meshes. *ACM Trans. on Graphics (SIGGRAPH 2004) 23*, 3 (Aug.), 399–405.

SUMNER, R. W., ZWICKER, M., GOTSMAN, C., AND POPOVIĆ, J. 2005. Mesh-based inverse kinematics. *ACM Trans. on Graphics (SIGGRAPH 2005) 24*, 3 (July), 488–495.

THULASIRAMAN, K., AND SWAMY, M. N. S. 1992. *Graphs: Theory and Algorithms*, first ed. Wiley-Interscience.

WANG, R. Y., PULLI, K., AND POPOVIĆ, J. 2007. Real-time enveloping with rotational regression. *ACM Trans. on Graphics (SIGGRAPH 2007) 26*, 3 (July).

WANG, H., HECHT, F., RAMAMOORTHI, R., AND O'BRIEN, J. 2010. Example-based wrinkle synthesis for clothing animation. *ACM Trans. on Graphics (SIGGRAPH 2010) 29*, 4 (July).

WANG, H., O'BRIEN, J. F., AND RAMAMOORTHI, R. 2011. Data-driven elastic models for cloth: modeling and measurement. *ACM Trans. on Graphics (SIGGRAPH 2011) 30*, 4 (Aug.).

WEI, L.-Y., LEFEBVRE, S., KWATRA, V., AND TURK, G. 2009. State of the art in example-based texture synthesis. In *Eurographics'09, State of the Art Report*.

YU, Y., ZHOU, K., XU, D., SHI, X., BAO, H., GUO, B., AND SHUM, H.-Y. 2004. Mesh editing with poisson-based gradient field manipulation. *ACM Trans. on Graphics (SIGGRAPH 2004) 23*, 3 (Aug.), 644–651.

ZHENG, C., AND JAMES, D. L. 2012. Energy-based self-collision culling for arbitrary mesh deformations. *ACM Trans. on Graphics (SIGGRAPH 2012) 31*, 4 (July), 98:1–98:12.