Eurographics / ACM SIGGRAPH Symposium on Computer Animation (2016)
Ladislav Kavan and Chris Wojtan (Editors)

Repurposing Hand Animation for Interactive Applications

Stephen W. Bailey[1,2] Martin Watt[2] James F. O'Brien[1]

[1]University of California, Berkeley [2]DreamWorks Animation

Figure 1: Four frames of a synthesized roar animation for Toothless the dragon

Abstract

In this paper we describe a method for automatically animating interactive characters based on an existing corpus of key-framed hand-animation. The method learns separate low-dimensional embeddings for subsets of the hand-animation corresponding to different semantic labels. These embeddings use the Gaussian Process Latent Variable Model to map high-dimensional rig control parameters to a three-dimensional latent space. By using a particle model to move within one of these latent spaces, the method can generate novel animations corresponding to the space's semantic label. Bridges link each pose in one latent space that is similar to a pose in another space. Animations corresponding to a transitions between semantic labels are generated by creating animation paths that move though one latent space and traverse a bridge into another. We demonstrate this method by using it to interactively animate a character as it plays a simple game with the user. The character is from a previously produced animated film and the data we use for training is the data that was used to animate the character in the film. The animated motion from the film represents an enormous investment of skillful work. Our method allows this work to be repurposed and reused for interactively animating the familiar character from the film.

1. Introduction

Feature animation is a labor and time intensive process that results in characters with compelling and unique personalities. Taking one of these characters into an interactive application presents a challenge. The traditional approach is to hand animate large numbers of motion clips which can then be evaluated in a motion graph. This becomes expensive due to the large number of possible actions required. Even a single action can require multiple clips to avoid obvious visual repetition when idling in a specific pose.

In this paper we repurpose the original hand animated content from a film by using it as a training set which is then used to generate new animation in real time that can retain much of the personality and character traits of the original animation. Due to this choice of training data, we assume that we will have tens of minutes of usable animation. Furthermore, because we use animation for a film-quality character, there is a large number of rig parameters that our synthesis algorithm will need to control. Thus, we use a form of the Gaussian Process Latent Variable Model (GPLVM) to embed the rig parameters of the animation in a lower dimensional space, and we synthesize new animations using this model.

Our work presents a new method to scale the input data to the GPLVM to account for the nonlinear mapping between a character's rig parameters and its evaluated surface mesh. Further, we present a novel method to synthesize new animation using the GPLVM. Our method is based on a particle simulation, and we demonstrate its effectiveness at generating new facial animation for a non-human character. We found that GPLVMs trained with a few homogeneous animations produce visually better results than

one trained with many animations of varying types of motions. Our method uses multiple GPLVMs, and we present a novel method to synthesize smooth animations that transition between models. To demonstrate the effectiveness of our work, we develop an interface for our method to receive directions to control the animation in real-time. We developed an interactive application to interface with our method to show that our algorithm can synthesize compelling and expressive animation in real-time.

2. Related Work

Statistical methods have been used to analyse and synthesize new motion data [BH00, MK05, LBJK09]. In particular, the Gaussian Process Latent Variable Model (GPLVM) [Law06] has been used for a number of applications in animation such as satisfying constraints or tracking human motion [GMHP04, UFHF05, WFH08] as well as interactive control [YL10, LWH*12]. This model is used to reduce the dimension of the motion data and to create a statistical model of the animation. Modifications to the GPLVM have been proposed to make it better suited for modeling motion data. The GPLVM tends to keep far data separated in the reduced dimensional space, but it makes no effort to keep similar data points close together. A number of methods have been proposed to address this limitation. Back constraints [LQnC06] have been applied to the GPLVM to preserve local distances. Dynamic models [WFH06, Law07] have also been introduced to model the time dependencies in animation data. A connectivity prior [LWH*12] has been proposed to ensure a high degree of connectivity among the animation data embedded in the low-dimensional latent space. Prior methods that model animation data with a GPLVM have been applied to full-body motion capture data. In contrast with past work, we apply a similar technique to hand-crafted animation for a film-quality character. One key difference between motion capture data and film-quality hand animation is that the hand animation lies in a significantly higher dimensional space than the motion capture data in terms of the number of parameters needed to specify a pose.

Data-driven approaches to character control and animation synthesis have focused on full-body tasks, which are based on motion graphs [AF02, KGP02, LCR*02, TLP07, LZ08, LLP09, MC12]. These methods use a graph structure to describe how motion clips from a library can be connected and reordered to accomplish a task. These approaches perform well with large training set; however, smaller data sets might not be well-suited for motion graphs because a lack of variety and transitions in the motions. Other methods for character control include data-driven and physics-based approaches [CBvdP09, MLPP09, LWH*12, TGLT14]. All of these methods are applied to full-body human motion or hand motion [AK13]. The tasks the controllers are trained can be quantifiably measured such as locomotion or reaching tasks. In contrast, we use our method to animate a non-human character's face. Tasks for facial animation are not as easy to quantify, and we therefore develop a novel particle simulation-based method to control facial animation.

Facial animation of non-human characters can be controlled by retargetting recorded expressions. A commonly used method is blendshape mapping [BFJ*00, CXH03, SSK*11, BWP13, CWLZ13], which maps expressions from an input model onto corresponding expressions from the target character. Motion is generated by then blending between the different facial shapes of the character. This approach uses an input model such as a video recording of a human to drive the animation of the character. Unlike the blenshape mapping approaches, our method does not control facial animation with recordings of a model. Furthermore, we do not require that the character's face be animated with blendshapes. We make no assumptions about the character's rig, but specifically the face rig we used in our results is animated using a combination of bones, blendshapes, and free-form deformations. Other methods use speech recordings to control the facial animation [LP86, WL94, ET97, Bra99]. Our method does not use video or speech recordings to control the facial animation. Instead we use user interaction with an interactive application as input for our animation synthesis algorithm. Another method for modeling facial expressions allows users to manipulate the face directly and avoids unnatural faces by learning model priors [LCXS09].

Animated characters are controlled through an underlying rig, which deforms a surface mesh that defines the character. A variety of methods exist to map a character's rig controls to deformations of the surface mesh [Bar84, SP86, MTLT88, SF98, LCF00] as well as the inverse from a skeleton to rig space [HSK15]. Our method makes no assumptions about rig controls and treats mapping from the character rig to the surface mesh as an arbitrary nonlinear function, similar to the assumptions made in [HTC*13].

3. Overview

Our work computes a low dimensional embedding for a set of training animation and uses the resulting model to generate new animation. The animation data is represented as character rig parameters, which can be evaluated to generate a surface mesh of the character. We make no assumptions about the mapping from rig parameters to the mesh. Because the mapping is typically nonlinear, variation in the rig controls might not necessarily correspond with a similar variation in the surface mesh. We therefore scale each component of the rig parameters based on an approximation of the influence each control has on the mesh.

Next, we embed the scaled rig parameters in a low dimensional space. We first use principal component analysis (PCA) to reduce the data to an intermediate space. We then use then use a form of the GPLVM to further reduce the dimension of the data. Our GPLVM variant keeps similar poses in the animation close in the latent space and keeps temporally close poses near each other as well. For pose synthesis, we compute the maximum a posteriori estimate for the most likely rig parameters given a low-dimensional latent point. We use the learned models to synthesize new animations in real-time. The current pose of a synthesized animation is represented as a particle in the latent space. We apply forces to the particle to push it towards user-defined targets. At each time step in the simulation, we use the current location of the particle in the latent space to generate the next pose in the animation using the GPLVM. We found that this method creates expressive facial animations.

Because we train a separate GPLVM for each type of action, the particle simulation by itself cannot generate animations that transition between models. To overcome this limitation, we compute

matching points between the models. These matching points are locations in the latent spaces that map to similar rig parameters. Transitions between models are performed by moving the particle to one of these matching points, switching models, and starting a new simulation at the corresponding matching point in the new model.

4. Low Dimensional Embedding

Given a large set of training animation, represented as a sequence of rig control parameters, our method learns a mapping between a low dimensional latent space and rig parameters. This mapping is generated in three stages. First, each rig control in the training animation is scaled to weight the controls proportional to changes in the final mesh. Second, the training animation is reduced linearly using Principal Component Analysis (PCA). Finally, the data is mapped to a lower dimensional latent space using a form of the Gaussian Process Latent Variable Model (GPLVM). After we have found an embedding of the training data in the latent space, we can then map any arbitrary point in the low dimensional space to values for the rig controls.

4.1. Scaling Rig Controls

We assume that the character rig parameters $\mathbf{p}$, when evaluated, produces a surface mesh. The i^{th} vertex of this mesh is given by the function $\mathbf{e}_i(\mathbf{p})$. We only assume that the rig evaluation function $\mathbf{e}(\mathbf{p})$ is continuous. Otherwise, we make no other assumptions about the function to keep our method as general as possible. Thus, the evaluation function will typically be highly nonlinear.

Depending on how the evaluation function $\mathbf{e}(\mathbf{p})$ is defined, large changes in some rig parameters might result in small changes in the output surface mesh while small changes for other parameters might result in large changes in the mesh. Specifically for some setting of the rig parameters $\mathbf{p}$, the value $\left\| \frac{\partial \mathbf{e}(\mathbf{p})}{\partial p_i} \right\|$ might be large for the i^{th} rig parameter, but the value $\left\| \frac{\partial \mathbf{e}(\mathbf{p})}{\partial p_j} \right\|$ might be small for some other rig control. Thus, there could exist some rig controls that have a very small effect on the surface mesh but have a large variance across the training animation. Because we will be using PCA, we want to scale each component of the data so that the principal axes of the transformation do not align with these controls with high variance but low influence on the mesh.

To avoid this situation, we want to scale the rig parameters about the sample average to obtain $\mathbf{z} = \mathbf{W}(\mathbf{p} - \bar{\mathbf{p}}) + \bar{\mathbf{p}}$ where $\mathbf{W}$ is a diagonal matrix and w_i is the amount to scale the i^{th} rig parameter. We choose $\mathbf{W}$ such that a unit change in the scaled rig parameter space corresponds with approximately a unit change in the surface mesh. Specifically for the i^{th} rig parameter,

$$\left\| \frac{\partial}{\partial z_i} \mathbf{e}(\mathbf{W}^{-1}(\mathbf{z} - \bar{\mathbf{p}}) + \bar{\mathbf{p}}) \right\| = 1 \tag{1}$$

where $\mathbf{z}$ is any possible value of the scaled rig parameters.

We use $\mathbf{p} = \mathbf{W}^{-1}\mathbf{z}$ and the chain rule to find that

$$\left\| \frac{\partial \mathbf{e}(\mathbf{p})}{\partial p_i} \frac{\partial}{\partial z_i} \left[w_i^{-1}(z_i - \bar{p}_i) + \bar{p}_i \right] \right\| = 1. \tag{2}$$

We can use Equation 2 to solve for the weights and find that $w_i = \left\| \frac{\partial \mathbf{e}(\mathbf{p})}{\partial p_i} \right\|$. Because $\mathbf{e}(\mathbf{p})$ is a generally nonlinear function, Equation 2 cannot be satisfied for all possible values of $\mathbf{p}$ for a fixed $\mathbf{W}$. Instead, we approximate the norm of the partial derivative by evaluating the rig at the sample mean $\bar{\mathbf{p}}$ of the training data and at several points about the mean. For rig parameter i, we construct a least squares error problem to approximate the norm of the partial derivative by

$$\left\| \frac{\partial \mathbf{e}(\mathbf{p})}{\partial p_i} \right\| \approx \underset{w}{\operatorname{argmin}} \sum_{n=-2}^{2} \left(\|\mathbf{e}(\bar{\mathbf{p}}) - \mathbf{e}(\bar{\mathbf{p}} + n\sigma_i)\| - w\|n\sigma_i\| \right)^2 \tag{3}$$

where σ_i is a vector with the sample standard deviation of the i^{th} rig parameter in the i^{th} position and zeros elsewhere. The values $n \in \{-2, -1, 0, 1, 2\}$ were chosen experimentally, and this set was found to produce good results. We solve this least squares problem separately for each w_i.

4.2. Linear Dimensionality Reduction

Typically, a fully-rigged main character for a feature film will have on the order of thousands of rig controls. Some of these rig controls might not be used in the training data, and some might have a small, almost imperceptible effect on the animation. To remove these controls and simplify the data, we linearly reduce the dimension of the data by using Principal Component Analysis. This method will treat the small variations in the data as noise and remove it. This initial linear reduction helps improve the results of the GPLVM that is used later.

Let $\mathbf{z}$ represent the scaled rig parameters of a single frame of animation. Suppose that there are D_{rig} parameters and that there are N total number of frames of animation in the training set. The scaled animation data can be represented as $\mathbf{Z} = [\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3, ..., \mathbf{z}_N]$. We then compute the singular value decomposition of the data $\bar{\mathbf{Z}} = \mathbf{U}\Sigma\mathbf{V}^T$ where the matrix $\bar{\mathbf{Z}}$ is the matrix $\mathbf{Z}$ with the sample mean subtracted from each column of the matrix. We choose the number of principal components d_{pca} to use by considering the explained variance of the model. The explained variance is given by $v(d) = \sum_{i=1}^{d} \sigma_i^2 / \sum_{i=1}^{k} \sigma_i^2$, where σ_i^2 is the i^{th} singular value of the normalized matrix $\bar{\mathbf{Z}}$ and k is the rank of the matrix. In our experiments for our models, we chose d_{pca} such that $v(d_{pca}) \approx 0.85$. With the number of principal components chosen, we define the transformation matrix $\mathbf{T}_{pca}$, which contains the first d_{pca} columns of the matrix $\mathbf{U}$. We then represent the training data as the matrix $\mathbf{Y} = \mathbf{T}_{pca}^T \bar{\mathbf{Z}}$.

We evaluated the difference between running PCA on the original and scaled rig parameters to determine the effect scaling the parameters has on the quality of the dimensionality reduction. We found that when enough principal components are used to ensure

that the explained variance is at or above 85%, there is no discernible difference quality of the animations between the scaled and original rig parameters, but the GPLVMs described in the following section tended to perform better with the scaled rig parameters. The difference between the original rig parameters and the compressed data, measured as $\left\|\mathbf{z}-\mathbf{T}_{pca}\mathbf{T}_{pca}^{T}\mathbf{z}\right\|$, is much larger when using the scaled rig parameters compared to the unscaled parameters. When we use a small number of principal components, animations compressed with the scaled rig parameters are visually better than the animations compressed with the unscaled data. Furthermore, the unscaled version often contains objectively undesirable meshes, such as the jaw of a character passing through the roof of its mouth. Therefore, we conclude that quantitative comparisons in the rig parameter space will not be sufficient to evaluate the effectiveness of our method.

4.3. Nonlinear Dimensionality Reduction

Given the linearly reduced data in the matrix $\mathbf{Y}$, we now compute a low-dimensional embedding through the use of a Gaussian Process Latent Variable Model [Law06]. The GPLVM is a generative, probabilistic model that we use to map nonlinearly the PCA transformed data $\mathbf{Y}$ to a set of points $\mathbf{X}$ in a latent space of dimension d_{gplvm} where $d_{gplvm} < d_{pca}$. We model dynamics in the latent space by placing a Gaussian process prior on the points $\mathbf{X}$ as described in [Law07]. This dynamics prior will thus keep temporally close data points close together spatially. Because we train our models using multiple segments of animation, the GPLVM with a dynamics prior will tend to keep separate segments far apart in the latent space. This separation is caused by the GPLVM placing dissimilar frames of animation far apart without trying to place similar frames near each other. Therefore, we use the connectivity prior described in [LWH*12] in order to pull together similar frames of animation from separate segments.

The GPLVM models the training data $\mathbf{Y}$ as the outputs of a Gaussian process from the low dimensional embedding of the points $\mathbf{X}$. We assume that each output of the GP is independent so that

$$\begin{aligned}\log p(\mathbf{Y}|\mathbf{X}) &= \sum_{i=1}^{d_{pca}} \log N(\mathbf{y}_{i,:}|\mathbf{0},\mathbf{K}_x) \\ &= -\frac{d_{pca}}{2}|\mathbf{K}_x| - \frac{1}{2}\operatorname{tr}\left(\mathbf{K}_x^{-1}\mathbf{Y}\mathbf{Y}^T\right) + \text{const.}\end{aligned} \tag{4}$$

We denote the i^{th} row of $\mathbf{Y}$ as $\mathbf{y}_{i,:}$. For the entries in the kernel matrix $\mathbf{K}_x$, we use the radial basis function, which is given by:

$$k_X(\mathbf{x}_i,\mathbf{x}_j) = \sigma_{rbf}^2 \exp\left(-\frac{1}{2l_x^2}\left\|\mathbf{x}_i-\mathbf{x}_j\right\|^2\right) + \delta_{ij}\sigma_{white}^2. \tag{5}$$

The kernel parameters σ_{rbf}^2, σ_{white}^2, and l^2 are optimized when the GPLVM is trained.

Our input data is composed of multiple segments of animation, and we would like to model the dynamics of each segment. We place a Gaussian process prior on the latent points $\mathbf{X}$. The input to the GP is time $\mathbf{t}$ of each frame. Each segment of animation is independent from all others; thus, the prior places a Gaussian process on each segment separately. The dynamics prior is given by

$$\psi_D(\mathbf{X},\mathbf{t}) = \sum_{i=1}^{d_{gplvm}} \log N(\mathbf{X}_{\mathbf{i},:}|\mathbf{0},\mathbf{K}_\mathbf{t}). \tag{6}$$

The entries of the kernel matrix $\mathbf{K}_t$ are computed by the radial basis function. Furthermore, $K_t^{ij} = 0$ when frames i and j belong to separate animation segments. See the description of the simple hierarchical model in [Law07] for more details.

The connectivity prior provides a method to model the degree of connectivity among the latent points $\mathbf{X}$ by using graph diffusion kernels. We denote this prior with $\psi_C(\mathbf{X})$. See the description of the connectivity prior in [LWH*12] for more details.

Combining the dynamics and connectivity priors, we can express the conditional probability of $\mathbf{X}$ as $p(\mathbf{X}|\mathbf{t}) \propto \exp\psi_D(\mathbf{X},\mathbf{t})\exp\psi_C(\mathbf{X})$. We estimate the latent points $\mathbf{X}$ and the hyper-parameters σ_{rbf}, σ_{white}, and l_x through maximum a posteriori (MAP) estimation. Thus, we want to maximize

$$\begin{aligned}&\log p(\mathbf{X},\sigma_{rbf},\sigma_{white},l_x|\mathbf{Y},\mathbf{t}) = \\ &\qquad \log p(\mathbf{Y}|\mathbf{X}) + \psi_D(\mathbf{X},\mathbf{t}) + \psi_C(\mathbf{X}).\end{aligned} \tag{7}$$

To maximize Equation (7), we use scaled conjugate gradient. The initial guess for the latent points is the first d_{gplvm} rows of $\mathbf{Y}$. We manually set the hyper-parameters for the dynamics prior and do not optimize these values. In Figure 2, we show a plot of several animation curves embedded in a three dimensional latent space.

4.4. Mapping to Rig Controls

Once we have a trained a model, we are now able to reconstruct rig control values from a new point $\mathbf{x}'$ in the latent space. We first find the most likely point in the d_{pca} dimensional space given the new point and the GPLVM model. Next, we multiply by the matrix of principal components to obtain the scaled rig parameters. Finally, we divide by the scaling factors and add the mean to each parameter.

The distribution of a new point $\mathbf{y}$ given the corresponding latent point $\mathbf{x}$ and the GPLVM model M is a Gaussian distribution where

$$p(\mathbf{y}|\mathbf{x},M) = N(\mathbf{y}|\mathbf{Y}\mathbf{K}_x^{-1}\mathbf{k}_x(\mathbf{x}), k_x(\mathbf{x},\mathbf{x}) - \mathbf{k}_x(\mathbf{x})^T\mathbf{K}_x\mathbf{k}_x(\mathbf{x})) \tag{8}$$

where $\mathbf{k}_x(\mathbf{x})$ is a column vector whose i^{th} entry is given by $\mathbf{k}_x(\mathbf{x})_i = k_x(\mathbf{x}_i,\mathbf{x})$. Because the distribution is Gaussian, the most likely point in the d_{pca} dimensional space is given by the mean $\mathbf{Y}\mathbf{K}_x^{-1}\mathbf{k}_x(\mathbf{x})$. The product $\mathbf{Y}\mathbf{K}_x^{-1}$ can be precomputed, which would allow this pose reconstruction problem to run in time linear to the size of the training data for the model.

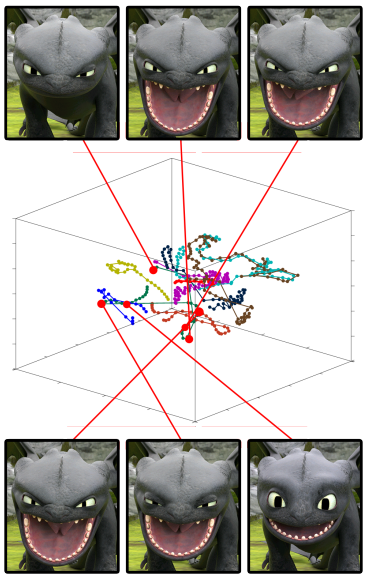

Figure 2: Three dimensional latent space learned for a training set of 9 examples of a roar with a total of 393 frames.

5. Animation Synthesis in Latent Space

New animations can be synthesized by generating a new path $\mathbf{P} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_t]$ through the latent space. The rig parameters for each point in the path can be computed by mapping the point from the latent space to the high dimensional rig control space. Because the latent space provides a continuous mapping any smooth curve in this low-dimensional space will result in smooth animation curves for each rig parameter.

To synthesize a new path, we simulate a particle moving through the latent space and track its position over time. We control the particle using a Lagrange multiplier method to enforce constraints on the system. For example, if we desire a path that does not stray too far from a user-defined point, we define a constraint to enforce this behavior. To add variations and noise to the path, we apply a random force. We found that this particle simulation method works well for synthesizing facial animations.

In order to achieve real-time performance, the number of training points in the GPLVM must be small. Therefore, the training animation needs to be divided into sufficiently small subsets. Each subset of animation corresponds with a specific type of expression or facial action such as a roar. A separate GPLVM is trained on each subset of animation. Because these latent spaces are separate, we need a method to map points from one model to another. With such a mapping, the particle simulation can transition between models, which allows for the synthesis of facial animations across multiple subsets of the animation.

We conclude this sections with a descriptions of a set of low-level "commands" to provide control of the synthesized animation. These commands are used to control the particle in the latent space, which thus gives control of the synthesized animation. The motivation for these commands is to develop a system reminiscent of the method an artist might use to plan an animation of a character's face. These commands allow for a user or an application to specify key poses in time, and our animation synthesizer generates motion that transitions between the poses.

5.1. Particle Simulation

We synthesize curves in the latent space by tracking the position of a particle in this space over time.

The input to our simulation is a path $\mathbf{p}(t)$ that the particle follows through time. We apply two constraints to the system and a "random" force to add noise to the path. The first constraint ensures that the particle does not move too far from the path. The second constraint ensures that the particle remains in areas of high probability in the GPLVM. Because there could be times when both constraints cannot be satisfied simultaneously, we model the path-following constraint as a hard constraint that must be satisfied, and the other constraint is modeled as a soft constraint that can be violated.

Given some path $\mathbf{p}(t)$ parametrized by time, we want to ensure that the particle does not drift too far away from the curve. To enforce this requirement, we apply the inequality constraint $\|\mathbf{x} - \mathbf{p}(t)\|^2 - r^2 \leq 0$ to ensure that the particle at location $\mathbf{x}$ stays within a distance r of the point $\mathbf{p}(t)$ at time t. Forward simulation with this constraint is computed using the Lagrange multiplier method described in [Bau72].

Let $\mathbf{F}$ be the force acting on the particle at time t. We use the Lagrange multiplier method to compute an additional force $\mathbf{F}_c$ that we apply to the particle to ensure that the constraint is satisfied. The constraint force is given by $\mathbf{F}_c = \lambda \mathbf{g}$ where $\mathbf{g} = \mathbf{x}(t) - \mathbf{p}(t)$. The multiplier λ for a particle of unit mass is given by

$$\lambda = \frac{-\mathbf{g}^T \mathbf{F} + G}{\mathbf{g}^T \mathbf{g}}. \tag{9}$$

The scalar G is given by

$$G = (\dot{\mathbf{x}}(t) - \dot{\mathbf{p}}(t))^T (\dot{\mathbf{x}}(t) - \dot{\mathbf{p}}(t)) + 2\alpha(\mathbf{g}^T \dot{\mathbf{x}}(t) - \mathbf{g}^T \dot{\mathbf{p}}(t)) + \frac{1}{2}\beta^2(\mathbf{g}^T \mathbf{g} - r^2). \tag{10}$$

The parameters α and β are selected by the user to control how quickly a system violating the constraints returns to a state satisfying them. We set $\beta = \alpha^2$, which is suggested in [Bau72]. The term $\mathbf{F}_c$ described above will apply a force to satisfy the equality constraint $\|\mathbf{x}(t) - \mathbf{p}(t)\|^2 - r^2 = 0$. To allow the particle to move freely within the radius around the target point, we constrain the force $\mathbf{F}_c$ to only point towards the target point $\mathbf{p}(t)$. This is accomplished by setting $\lambda = 0$ whenever $\lambda > 0$.

Our second constraint pushes the particle towards high probability regions in the latent space. The GPLVM provides a probability distribution over the latent space $p(\mathbf{x}(t)|M)$, and we use this distribution to push the particle towards "probable" regions, which can provide better reconstructed poses than less probable regions of the

latent space. However, we found that models trained with facial animations can synthesize reasonable poses from less likely regions of the latent space. We found that generally these lower probability poses do not contain visual defects such as an overly stretched face or interpenetrating meshes. Therefore, keeping the particle in a high probability region is not critical and can be violated if necessary to satisfy the path constraint. We model this likelihood constraint as a force applied to the particle that points in the direction of the gradient of the PDF. The magnitude of the force is determined by the value of the PDF evaluated at the particle's current location. If the value is above some empirically chosen quantity v, the magnitude is small, and if the value is below v, the magnitude is large. We model this as a sigmoid function so that the force function is continuous for numerical integration. The magnitude is expressed as

$$S(t) = a\left(1+\exp\left(\frac{p(\mathbf{x}(t)|M)-v}{l}\right)\right)^{-1}, \tag{11}$$

and the constraint force is expressed as

$$\mathbf{F}_{GPLVM}(t) = S(t)\frac{\partial p(\mathbf{x}(t)|M)}{\partial \mathbf{x}} \Big/ \left\|\frac{\partial p(\mathbf{x}(t)|M)}{\partial \mathbf{x}}\right\|. \tag{12}$$

The parameters a and l are defined by the user, and control the magnitude of the force when the constraint is not satisfied and how quickly the magnitude approaches a. Computing the partial derivatives of the Gaussian process takes time quadratic to the size of the training data. If the size of the training set is small, this can be computed in real-time.

In addition to these constraint forces, we apply a random force $\mathbf{F}_{rand}(t)$ to add variation to the particle's path. We model this force as a randomly drawn, zero-mean Gaussian process: $\mathbf{F}_{rand}(t) \sim \mathcal{GP}(0, k(t,t'))$. Each component of $\mathbf{F}_{rand}(t)$ is independent of all others. The covariance function is given by $k(t,t') = \alpha \exp\left(-(2\gamma)^{-1}(t-t')^2\right)$, where α and γ are user-defined parameters that control the magnitude and smoothness of the random force.

This random force adds noise and variations to the particle's movement through the latent space. Thus, a particle following the same path multiple times will have slight variations in each repetition, which will generate unique animations with small but noticeable differences. Variations in the animation could be achieved through other means such as perturbing the path $\mathbf{p}(t)$; however, we did not evaluate these other possibilities.

In our experiments, we simulate the particle forward in time using a fourth-order Runge-Kutta integration method. We used a piecewise linear function for the path $\mathbf{p}(t)$, which is defined by a set of points $[\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_n]$ such that $\mathbf{p}(t_i) = \mathbf{p}_i$ and t_i is the time of the i^{th} frame of animation. We do not integrate across multiple frames of animation to avoid integrating over discontinuities in the piecewise path function $\mathbf{p}(t)$. Section 5.3 describes methods to define $\mathbf{p}(t)$.

5.2. Mapping Between Models

A large set of heterogeneous motions cannot be accurately embedded in a low dimensional ($d \leq 5$) latent space. Therefore, we divide the training animation into small sets of similar expressions and compute the embedding in the latent space for each subset separately. The drawback of training separate models is that animations transitioning between multiple models cannot be synthesized using our particle simulation method. This problem arises because a continuous path between models does not exist. In this section, we describe a method to synthesize smooth animations that transition between latent spaces.

To create a path between two models M_1 and M_2, we first precompute a set S of corresponding points in both latent spaces. A pair of matching points $(\mathbf{x}_1, \mathbf{x}_2)$ where $\mathbf{x}_1 \in M_1$ and $\mathbf{x}_2 \in M_2$ is included in S if $\|\mathbf{g}(\mathbf{x}_1; M_1) - \mathbf{g}(\mathbf{x}_2; M_2)\|^2 < \varepsilon$ where $\mathbf{g}(\mathbf{x}; M)$ is the function that maps $\mathbf{x}$ to the rig parameter space. Thus, we want to identify pairs of points in the latent spaces whose reconstructed poses are similar. The set of matching points identifies points in the two models, which can be used as bridges between the two models. To create a curve that moves between model M_1 to M_2, we create a path in M_1 that ends at a point in S for the model and then create a path that starts at the matching point in M_2.

To identify a pair of matching points for models M_1 and M_2, we fix a point $\mathbf{x}_1 \in M_1$ and compute the reconstructed rig parameters $\mathbf{z}_1 = \mathbf{g}(x_1; M_1)$. The point $\mathbf{x}_1$ can be any point; however, in our implementation, we restricted $\mathbf{x}_1$ to be from the set of latent points corresponding to the training animation for the model. Next, the point $\mathbf{z}_1$ is transformed by the linear dimensionality reduction specified by model M_2

$$\hat{\mathbf{y}}_1 = \mathbf{T}_2^T[\mathbf{W}_2(\mathbf{z}_1 - \mathbf{m}_2)] \tag{13}$$

where $\mathbf{T}_2$ is the first d principal components of the PCA transformation given in model M_2, $\mathbf{W}_2$ is the diagonal matrix of scale values for each component, and $\mathbf{m}_2$ is the mean of the training data used in model M_2.

The next step is to find the point $\mathbf{x}_2$ in the latent space of model M_2 such that

$$\mathbf{x}_2 = \underset{\mathbf{x}}{\operatorname{argmin}} \left\|\hat{\mathbf{y}}_1 - \underset{\mathbf{y}}{\operatorname{argmax}} \log p(\mathbf{y}|\mathbf{x}, M_2)\right\|^2. \tag{14}$$

Because $y_i = f(\mathbf{x}) + \varepsilon$ where ε is additive Gaussian white noise, the maximum of $p(\mathbf{y}|\mathbf{x}, M_2)$ occurs when $\mathbf{y} = \mathbf{f}_*$ where $f_* = \mathbf{K}_*[\mathbf{K}_x]^{-1}\mathbf{Y_2}$ is the noise-free output for the test point $\mathbf{x}$. Therefore, Equation (14) can be written as

$$\mathbf{x}_2 = \underset{\mathbf{x}}{\operatorname{argmin}} \left\|\hat{\mathbf{y}}_1 - \mathbf{K}_*[\mathbf{K}_x]^{-1}\mathbf{Y_2}\right\|^2. \tag{15}$$

The problem of finding the best matching $\mathbf{x}_2 \in M_2$ giving the point $\mathbf{x}_1 \in M_1$ is now formulated as a nonlinear optimization problem. We solve this problem by using the scaled conjugate gradient algorithm. However, because the function is multi-modal, we

Figure 3: Four examples of the best-matching poses found between two models. In each pair, the pose on the left is generated from a model trained on animations with grumpy-looking animations, and the pose on the right is generated from happy-looking animations.

run the optimization algorithm multiple times with randomly selected initial values to attempt to find the global minimizer. Furthermore, care needs to be taken not to take large steps during the optimization routine because the gradient of the objective function quickly goes to zero as $\mathbf{x}_2$ moves away from the training points in the model.

In our implementation, we identified pairs of matching points between models M_1 and M_2 by computing matching points $\mathbf{x}_2$ for each latent point of the training data for model M_1. We then evaluated the Euclidean distance between the reconstructed rig space poses for each pair of matching points. Pairs with distances below some user-defined threshold were kept while all other pairs were discarded. With this method, we obtained between 10-50 transition points between each pair of models. For models trained on similar-looking animations, the transition points were spread throughout the latent space. Models trained with distinct animations tended to have the transition points clustered around one or two small regions of the latent space.

To create an animation that transitions between two models, we generate a curve in the first model that ends at one of the precomputed transition points and a curve in the second model that starts at the corresponding transition point from the first model. The animation is synthesized by reconstructing the poses along the curves and placing the animation from the second model right after the first. As seen in Figure 3, the poses reconstructed from matching latent points in two models are not necessarily identical. As a result, there will be a discontinuity in the animation at the transition between the two models. To overcome this problem, we perform a short blend between the two poses in the rig parameter space at the transition point.

5.3. Synthesis Control

We use the particle simulation method described above to synthesize animation for the face of a non-human character and develop a set of commands to provide intuitive control of the character's expression. The high-level reasoning for using these commands is that we want to provide control over what pose the character has at a specific time in an animation. With these poses, our synthesis algorithm then generates transitions between the poses and models specified in the commands.

MOVE: The move command takes a target point $\mathbf{t}$ in the latent space as input. The synthesized animation is controlled by moving the particle from its current position in the latent space to the target point. This is accomplished by setting the particle's path function $\mathbf{p}(t)$. We tested two methods to generate the path. The first method creates a straight line from the current point to the target. The second method uses the shortest path in a complete weighted graph G of the training data. In the graph, we represent each frame of data as a vertex, and the weights between vertices are computed by $w(\mathbf{x}_i, \mathbf{x}_j) = \left\|\mathbf{x}_i - \mathbf{x}_j\right\|^{-p}$, which is similar to the graph constructed for the connectivity prior [LWH*12]. In our implementation, we found that setting $p = 4$ yielded good results. We also add the start and end points as vertices in the graph G. We re-sample the resulting path so that $\left\|\frac{\partial \mathbf{p}(t)}{\partial t}\right\|$ is constant for all t. This ensures that the particle follows the path at a consistent speed. We found that both path-generating methods create compelling animation. The only difference between the two is that the straight line path is shorter, and thus a particle following this path will reach the target in less time.

IDLE: When the animated character is not performing an action, we would like for the character to have an "idling" animation, and we would like to control the expression of the character as it idles. We found that we can synthesize idling animations by picking a point $\mathbf{p}$ in the latent space corresponding with a user-specified pose. This pose is a hand-selected expression. We let the particle move randomly within a radius r about the point to create variations of that pose. Keeping the particle within the radius is accomplished by setting the particle's path following function to $\mathbf{p}(t) = \mathbf{p}$ for the time we want idle about the point. To add variety to the animation, multiple user-specified points can be used. With multiple points, the synthesis can be controlled by first picking a point from the set to move to. Next, the particle hovers about that point for a fixed amount of time. Finally, an new point is selected, and the simulation repeats by moving to this new point and hovering. See the accompanying video for examples of synthesized idling animations.

TRANSITION: The transition command is used to generate a continuous animation between two models. This command uses the previously described MOVE and IDLE commands. To transition from model M_1 to model M_2, our method moves the particle from its current position in model M_1 to the nearest precomputed matching point in the latent space. When the particle is close to the point, it then idles about that point and the particle in M_2 also begins to idle about the corresponding matching point. We finish the transition by performing a blend between the high-dimensional rig parameters from the two models while the particles are idling. Please see the video for examples of transitions.

PLAY SEGMENT: Occasionally, we might want to play part of an animation unmodified directly from the training set. We play the animation by using the embedding of the sequence in the latent space. We use the MOVE command to position the particle near the starting pose of the animation. When the particle is close enough, we stop the simulation and move the particle along the path of the embedded animation. When moving the particle to the start, we adjust the radius r to ensure that it has moved close to the start to avoid discontinuities when the animation segment starts playing.

Figure 4: Set of frames from an animation synthesized using a model trained on a set of "surprise" expressions.

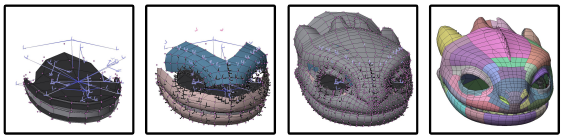

Figure 5: A visualization of the layered Deformation System for Toothless's facial rig that enables real time free-form facial control shaping.

6. Results

We used the method described above to synthesize animations at interactive frame rates. The input to our algorithm is film-quality hand animation. For a feature film, a main character might have about 20 minutes of animation. We manually separated the data into sets of similar expressions and also removed any visually bad data. For example, a character might be off screen and is not animated, or a character might be animated for one specific camera angle and does not look acceptable from all possible viewing angles. Using our method, we trained a separate model for each type of expression that we manually labeled in the training data. To evaluate the effectiveness of our method, we compared transitions synthesized with our method to transitions generated using Motion Graphs [KGP02]. Additionally, we synthesized scripted animations off-line and created an interactive game featuring synthesized real-time animation using our algorithm to demonstrate the application of our method.

We used the animation data from the hero dragon character Toothless in the feature film *How to Train Your Dragon 2*. This data is sampled at 24 FPS, and 742 face rig controls are used in our algorithm. Toothless's facial rig is a multi-layered design [PHW*15], which provides control ranging from coarse to fine deformations. Figure 5 shows the layers of the face rig. There are four main layers of the face rig that involve both bones and blenshapes. First, the bones control large, gross deformations of the mesh. Second, intermediate blendshapes are applied for coarse control. Third, fine-control blendshapes are used. Finally, free-form deformations are applied to allow custom shapes after the first three layers have been evaluated.

To demonstrate how well our method can reuse previous animation, we use only data from this film and do not hand animate any data specific for our applications. We identified eight expression sets: happy, grumpy, bored, curious, and neutral, roar, head shake, and surprise. We manually labeled animations that fit into these categories and trained a GPLVM on each one separately. The labeling task required several hours to complete. Each model contained between 100 to 800 frames of animation, and the latent space for each model has three dimensions. We chose three dimensions experimentally by training models with different dimensions. We found that for our small data sets, the quality of animations synthesized with models of three dimensions or higher were perceptually similar. Therefore, we chose the smallest dimension to minimize the number of unknown variables we solve for when training the GPLVM. In total, we included 3745 usable frames of animation in our training data, which is equivalent to 156 seconds of animation.

Because our method solves a problem similar to Motion Graphs and methods based on Motion Graphs, we compare expression transitions synthesized with our method to those we synthesized using Motion Graphs described in [KGP02]. In our method, we used on average 12 frames to blend between two models. Therefore, we used the same number of frames to synthesize the blends between segments of animation using Motion Graphs for comparison. In the accompanying video, we show transitions synthesized using both methods. For Motion Graphs, we picked transitions between two sets of animation by picking transitions points between animation sequences with small distances in the rig parameter space as described in their paper. Visually, we found that in some cases, transitions synthesized using Motion Graphs appear sudden and unnatural. We found that these sudden transitions occur when the two animations do not contain large movements. However, Motion Graph blends are not noticeable when transitioning between motions containing large movements. Our method, on the other hand is able to synthesize smooth transitions between different expressions regardless of the amount of motion before and after the transition.

We found that because our sets of training animation are small and contain heterogeneous motions, the Motion Graph algorithm was unable to find transitions with small distances going towards or away from most animation segments. Thus, a motion graph built on this data would use a small fraction of the data. Our method, however, makes use of the entire data set and is capable of transitioning to and from any pose.

We also evaluate our method by synthesizing scripted animations. We directly used our interface for the synthesis algorithm. We provided control over which command is sent to the system and when. This give the user the ability to specify poses that the character needs to make at a scripted time. Because the animation can be computed in real-time, the user can quickly see how changes in the script affect the animation. All of the off-line animations shown in our accompanying video are synthesized with this method. We found that scripting an animation allows for some-

one without an artistic background to author novel and expressive animations quickly.

We demonstrate the effectiveness of our algorithm through an interactive game of Tic-Tac-Toe, in which the user plays against the computer. We synthesize animation for Toothless's face to react in real time with the results of the game. During Toothless's turn, he holds a ponderous expression. Although the computer logic for Tic-Tac-Toe strategy can be computed in milliseconds, we intentionally extend Toothless's deliberation time to allow for expressions as if he were playing a cognitively difficult game. During the player's turn, he squints and scowls as if he were intimidating the player. When Toothless loses a round in the game, he roars and expresses anger, and when the he wins, he expresses happiness. If Toothless misses a move to block the player from winning, he displays an expression of surprise. All of these expressions are scripted using commands described in Section 5.3.

We found that eye movement is context specific. Because synthesizing new animation with eye movement lead to unrealistic animation, we fixed the eyes to look forward and do not include the eyes' rig parameters in the synthesis model.

For each emotional state animated in the game, we created a set of scripts containing specific commands. When the game needed to synthesize a particular emotional expression, it randomly picked a script from the corresponding set to run. Only the head shaking animation was scripted using the PLAY command. All other animations are scripted using TRANSITION, MOVE, and IDLE.

We tested our application on an HP Z840 workstation with two Intel Xeon E5-2687w processors running at 3.1GHz, providing 16 cores in total. The machine has 32GB RAM. To compute the surface meshes, we use LibEE [WCP*12], a multithreaded evaluation engine for calculating Toothless's surface mesh.

To achieve interactive frame rates for rig evaluation, the resolution of Toothless's final skin mesh was reduced by a factor of 5. This was done non-uniformly to ensure resolution was retained in the most critical areas for expression, e.g. eyes and wrinkles around the nose. Apart from the mesh resolution reduction, no other changes were made to the face rig compared with the original production rig used in the film. LibEE is also the same engine used to evaluate the rig during the production of the film; therefore, the animation and deformations are all the same as used in production. We render the mesh for the real-time application using OpenGL. The application runs successfully at 24 frames per second. Please see the supplementary video for a recording of the application running in real time.

7. Discussion

Our labeled training data for each expression formed small sets ranging from 100 to 800 frames of animation. Because of the small size of these sets, GPLVMs worked well to model the variation in the motion for each expression. However, dividing the data into separate sets of expressions has limitations. We cannot mix expressions because the models are separate. For example, our method is unable to combine "happy" and "surprise" expressions to synthesize a hybrid expression from both models. Generating these mixed expressions could be possible by training a GPLVM on a large, combined data set. However, we found that a GPLVM trained on this mixed set did not perform well because of the dissimilarities in the motions from the separate expressions. Additionally, the computation time required to train the model grows cubically with the size of the training data, and we found that the training times were unfeasibly long without using Gaussian process approximation techniques.

Our method's ability to synthesize transitions between models depends on its ability to find matching points between two expression models. Suppose that two GPLVM models are so different that no pair of similar points can be found. Then synthesizing transitions between the two might need to pass through a third model that has matching points with the two. For example, a transition going from happy to grumpy expressions might need to pass through a neutral expression if the happy and grumpy models share no similar points.

Acknowledgements

We thank Bo Morgan for his helpful discussions throughout the project. We also thank David Otte his assistance in working with the character rig for Toothless.

References

[AF02] ARIKAN O., FORSYTH D. A.: Interactive motion generation from examples. *ACM Trans. Graph. 21*, 3 (July 2002), 483–490. 2

[AK13] ANDREWS S., KRY P.: Goal directed multi-finger manipulation: Control policies and analysis. *Computers & Graphics 37*, 7 (2013), 830 – 839. 2

[Bar84] BARR A. H.: Global and local deformations of solid primitives. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1984), SIGGRAPH '84, ACM, pp. 21–30. 2

[Bau72] BAUMGARTE J.: Stabilization of constraints and integrals of motion in dynamical systems. *Computer Methods in Applied Mechanics and Engineering 1* (June 1972), 1–16. 5

[BFJ*00] BUCK I., FINKELSTEIN A., JACOBS C., KLEIN A., SALESIN D. H., SEIMS J., SZELISKI R., TOYAMA K.: Performance-driven hand-drawn animation. In *NPAR 2000 : First International Symposium on Non Photorealistic Animation and Rendering* (June 2000), pp. 101–108. 2

[BH00] BRAND M., HERTZMANN A.: Style machines. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., pp. 183–192. 2

[Bra99] BRAND M.: Voice puppetry. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1999), SIGGRAPH '99, ACM Press/Addison-Wesley Publishing Co., pp. 21–28. 2

[BWP13] BOUAZIZ S., WANG Y., PAULY M.: Online modeling for real-time facial animation. *ACM Trans. Graph. 32*, 4 (July 2013), 40:1–40:10. 2

[CBvdP09] COROS S., BEAUDOIN P., VAN DE PANNE M.: Robust task-based control policies for physics-based characters. *ACM Trans. Graph. 28*, 5 (Dec. 2009), 170:1–170:9. 2

[CWLZ13] CAO C., WENG Y., LIN S., ZHOU K.: 3d shape regression for real-time facial animation. *ACM Trans. Graph. 32*, 4 (July 2013), 41:1–41:10. 2

[CXH03] CHAI J.-X., XIAO J., HODGINS J.: Vision-based control

of 3d facial animation. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2003), SCA '03, Eurographics Association, pp. 193–206. 2

[ET97] ESCHER M., THALMANN N. M.: Automatic 3d cloning and real-time animation of a human face. In *Proceedings of the Computer Animation* (Washington, DC, USA, 1997), CA '97, IEEE Computer Society, pp. 58–. 2

[GMHP04] GROCHOW K., MARTIN S. L., HERTZMANN A., POPOVIĆ Z.: Style-based inverse kinematics. *ACM Trans. Graph. 23*, 3 (Aug. 2004), 522–531. 2

[HSK15] HOLDEN D., SAITO J., KOMURA T.: Learning an inverse rig mapping for character animation. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (New York, NY, USA, 2015), SCA '15, ACM, pp. 165–173. 2

[HTC*13] HAHN F., THOMASZEWSKI B., COROS S., SUMNER R. W., GROSS M.: Efficient simulation of secondary motion in rig-space. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2013), SCA '13, ACM, pp. 165–171. 2

[KGP02] KOVAR L., GLEICHER M., PIGHIN F.: Motion graphs. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2002), SIGGRAPH '02, ACM, pp. 473–482. 2, 8

[Law06] LAWRENCE N. D.: The Gaussian process latent variable model. *Technical Report no CS-06-05* (2006). 2, 4

[Law07] LAWRENCE N. D.: Hierarchical gaussian process latent variable models. In *In International Conference in Machine Learning* (2007). 2, 4

[LBJK09] LAU M., BAR-JOSEPH Z., KUFFNER J.: Modeling spatial and temporal variation in motion data. *ACM Trans. Graph. 28*, 5 (Dec. 2009), 171:1–171:10. 2

[LCF00] LEWIS J. P., CORDNER M., FONG N.: Pose space deformation: A unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., pp. 165–172. 2

[LCR*02] LEE J., CHAI J., REITSMA P. S. A., HODGINS J. K., POLLARD N. S.: Interactive control of avatars animated with human motion data. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2002), SIGGRAPH '02, ACM, pp. 491–500. 2

[LCXS09] LAU M., CHAI J., XU Y.-Q., SHUM H.-Y.: Face poser: Interactive modeling of 3d facial expressions using facial priors. *ACM Trans. Graph. 29*, 1 (Dec. 2009), 3:1–3:17. 2

[LLP09] LEE Y., LEE S. J., POPOVIĆ Z.: Compact character controllers. *ACM Trans. Graph. 28*, 5 (Dec. 2009), 169:1–169:8. 2

[LP86] LEWIS J. P., PARKE F. I.: Automated lip-synch and speech synthesis for character animation. *SIGCHI Bull. 17*, SI (May 1986), 143–147. 2

[LQnC06] LAWRENCE N. D., QUIÑONERO CANDELA J.: Local distance preservation in the gp-lvm through back constraints. In *Proceedings of the 23rd International Conference on Machine Learning* (New York, NY, USA, 2006), ICML '06, ACM, pp. 513–520. 2

[LWH*12] LEVINE S., WANG J. M., HARAUX A., POPOVIĆ Z., KOLTUN V.: Continuous character control with low-dimensional embeddings. *ACM Transactions on Graphics 31*, 4 (2012), 28. 2, 4, 7

[LZ08] LO W.-Y., ZWICKER M.: Real-time planning for parameterized human motion. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2008), SCA '08, Eurographics Association, pp. 29–38. 2

[MC12] MIN J., CHAI J.: Motion graphs++: A compact generative model for semantic motion analysis and synthesis. *ACM Trans. Graph. 31*, 6 (Nov. 2012), 153:1–153:12. 2

[MK05] MUKAI T., KURIYAMA S.: Geostatistical motion interpolation. In *ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), SIGGRAPH '05, ACM, pp. 1062–1070. 2

[MLPP09] MUICO U., LEE Y., POPOVIĆ J., POPOVIĆ Z.: Contact-aware nonlinear control of dynamic characters. In *ACM SIGGRAPH 2009 Papers* (New York, NY, USA, 2009), SIGGRAPH '09, ACM, pp. 81:1–81:9. 2

[MTLT88] MAGNENAT-THALMANN N., LAPERRIÈRE R., THALMANN D.: Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics Interface '88* (Toronto, Ont., Canada, Canada, 1988), Canadian Information Processing Society, pp. 26–33. 2

[PHW*15] POHLE S., HUTCHINSON M., WATKINS B., WALSH D., CANDELL S., NILSSON F., REISIG J.: Dreamworks animation facial motion and deformation system. In *Proceedings of the 2015 Symposium on Digital Production* (New York, NY, USA, 2015), DigiPro '15, ACM, pp. 5–6. 8

[SF98] SINGH K., FIUME E.: Wires: A geometric deformation technique. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1998), SIGGRAPH '98, ACM, pp. 405–414. 2

[SP86] SEDERBERG T. W., PARRY S. R.: Free-form deformation of solid geometric models. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1986), SIGGRAPH '86, ACM, pp. 151–160. 2

[SSK*11] SEOL Y., SEO J., KIM P. H., LEWIS J. P., NOH J.: Artist friendly facial animation retargeting. *ACM Trans. Graph. 30*, 6 (Dec. 2011), 162:1–162:10. 2

[TGLT14] TAN J., GU Y., LIU C. K., TURK G.: Learning bicycle stunts. *ACM Trans. Graph. 33*, 4 (July 2014), 50:1–50:12. 2

[TLP07] TREUILLE A., LEE Y., POPOVIĆ Z.: Near-optimal character animation with continuous control. *ACM Trans. Graph. 26*, 3 (July 2007). 2

[UFHF05] URTASUN R., FLEET D. J., HERTZMANN A., FUA P.: Priors for people tracking from small training sets. In *Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1 - Volume 01* (Washington, DC, USA, 2005), ICCV '05, IEEE Computer Society, pp. 403–410. 2

[WCP*12] WATT M., CUTLER L. D., POWELL A., DUNCAN B., HUTCHINSON M., OCHS K.: Libee: A multithreaded dependency graph for character animation. In *Proceedings of the Digital Production Symposium* (New York, NY, USA, 2012), DigiPro '12, ACM, pp. 59–66. 9

[WFH06] WANG J. M., FLEET D. J., HERTZMANN A.: Gaussian process dynamical models. In *In NIPS* (2006), MIT Press, pp. 1441–1448. 2

[WFH08] WANG J. M., FLEET D. J., HERTZMANN A.: Gaussian process dynamical models for human motion. *IEEE Trans. Pattern Anal. Mach. Intell. 30*, 2 (Feb. 2008), 283–298. 2

[WL94] WATERS K., LEVERGOOD T.: An automatic lip-synchronization algorithm for synthetic faces. In *Proceedings of the Second ACM International Conference on Multimedia* (New York, NY, USA, 1994), MULTIMEDIA '94, ACM, pp. 149–156. 2

[YL10] YE Y., LIU C. K.: Synthesis of responsive motion using a dynamic model. *Computer Graphics Forum 29*, 2 (2010), 555–562. 2