Published in ACM Transactions on Graphics (Special Issue for SIGGRAPH 2008)

Real-Time Data Driven Deformation Using Kernel Canonical Correlation Analysis

Wei-Wen Feng Byung-Uck Kim Yizhou Yu
University of Illinois at Urbana-Champaign

Abstract

Achieving intuitive control of animated surface deformation while observing a specific style is an important but challenging task in computer graphics. Solutions to this task can find many applications in data-driven skin animation, computer puppetry, and computer games. In this paper, we present an intuitive and powerful animation interface to simultaneously control the deformation of a large number of local regions on a deformable surface with a minimal number of control points. Our method learns suitable deformation subspaces from training examples, and generate new deformations on the fly according to the movements of the control points. Our contributions include a novel deformation regression method based on kernel Canonical Correlation Analysis (CCA) and a Poisson-based translation solving technique for easy and fast deformation control based on examples. Our run-time algorithm can be implemented on GPUs and can achieve a few hundred frames per second even for large datasets with hundreds of training examples.

CR Categories: I.3.7 [**Computer Graphics**]: Three Dimensional Graphics and Realism—Animation

Keywords: Animation, Skinning, Regression, Poisson Equation

1 Introduction

Achieving intuitive control of animated surface deformation while observing a specific style is an important but challenging task. Very often, the surface undergoing deformation does not have an intrinsic skeleton. For example, creating interesting and meaningful facial expressions requires surface deformation induced by an orchestrated coordination of a number of muscles. An intuitive and powerful animation interface should be able to simultaneously control the deformation of a large number of local regions on such a deformable surface with a minimal number of control points (animation parameters). Solutions to this task can find many applications in data-driven skin animation and computer puppetry. This is especially true with the recent trend in computer game design, which makes in-game avatars and virtual environments subject to user-level control and customization.

With the rapid progress in data acquisition and physically based simulation techniques, one effective solution would be learning suitable deformation subspaces from acquired or simulated examples, and generate new deformations on the fly according to the translational movements of a sparse set of control points. Thus, one challenge we need to overcome in intuitive control of arbitrary deformations would be learning the mapping between sparse control point movements and the deformation of an entire surface with

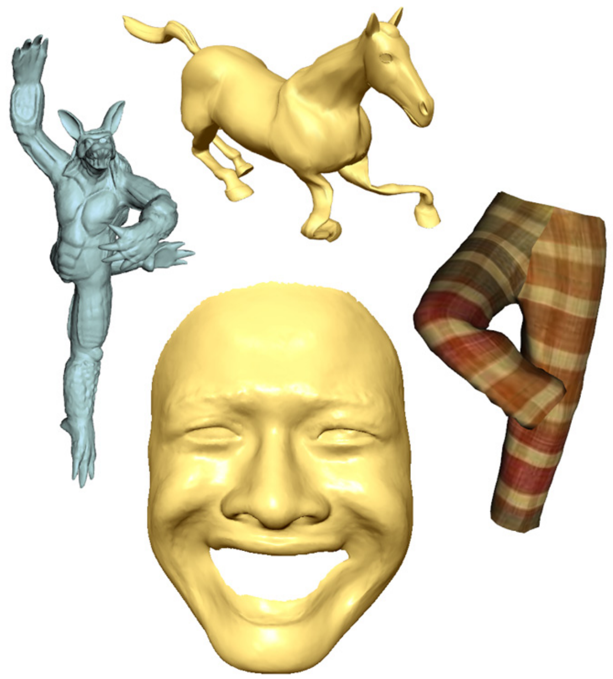
Figure 1: Novel deformations of various styles can be generated in real time with our data-driven method.

at least thousands of DOFs. Fortunately, deformations at different regions of the surface might be highly correlated. For example, moving an eyebrow should not only deform the eye, but also affect the cheek and mouth in a meaningful expression. Moving one region of clothing might create wrinkles on other regions. Therefore, it is crucial to learn these potentially nonlinear relationships from the examples and let them guide novel deformations.

Another challenge in achieving our goal is the performance requirement in real-time applications such as gaming. An update rate of thirty frames per second is the basic requirement in gaming applications which have a large portion of their system resources devoted to AI and rendering instead of animation. Thus the run-time algorithm for deformations needs to be extremely fast and reach a frame rate much higher than 30fps. Preferably it should make use of modern GPU's parallel processing power. This consideration implies that the framework should not involve sophisticated run-time computation that is hard to parallelize effectively on GPUs.

In this paper, we present a statistically based framework to learn deformation styles for a skinned mesh from example configurations. Our contributions include a novel deformation regression method based on kernel Canonical Correlation Analysis (CCA) [Hotelling 1936; Melzer et al. 2003] and a Poisson-based translation solving technique for easy and fast deformation control based on training examples. Our run-time algorithm can be implemented on GPUs and can achieve a few hundred frames per second even for large datasets with hundreds of training examples.

In our method, we first extract from example meshes a sparse set of control points as well as a skinned mesh with bones and bone influence weights. The goal of a subsequent learning process is to capture connections between control points and bone deformations in the example data and train a predictor generating novel bone deformations from control point movements. To achieve this, we first perform nonlinear model reduction by applying kernel CCA to find a pair of nonlinear subspaces that maximize the correlation between the pairs of example configurations. The original data are then projected into the subspaces to obtain their reduced coordinates. Standard regression techniques can be performed on the reduced coordinates to train a desired predictor.

At run-time, the deformation predictor is used to generate novel bone deformations according to control point movements. The prediction process is reformulated into matrix-vector multiplications and kernel transformations, both of which can be efficiently implemented on GPUs for parallel processing. For deformations with large bone rotations, the predicted bone translations can be improved using a real-time Poisson-based linear solver which only requires a single matrix-vector multiplication if the pseudo inverse of its coefficient matrix has been precomputed. The resulting bone transformations from previous steps can be directly used in a GPU-based skinning algorithm. High performance has therefore been achieved in all of our experiments.

2 Related Work

Mesh skinning has been widely used in games for skeleton driven skin deformation. Linear blend skinning (SSD) [Magnenat-Thalmann et al. 1988] is a popular technique that transforms every vertex using a weighted sum of nearby bone transformations. Bone influence weights can be automatically estimated from examples. Direct blending of rotation matrices by SSD suffers from collapsing joints and the "candy wrapper" artifact when joints are overly bent or twisted. Techniques have been proposed to compensate such inaccuracies. The method in [Mohr and Gleicher 2003] can well model example-based muscle deformations by adding extra joints around the area with large skinning errors. Their joint placement is compact and can resolve artifacts from SSD with little performance impact. Dual-quaternion skinning [Kavan et al. 2007a] have also demonstrated good results in terms of both deformation quality and performance. In addition to these fast skinning algorithms, there exist more expensive example-based techniques [Anguelov et al. 2005; Weber et al. 2007] that effectively integrate recent mesh deformation algorithms to generate high-quality skin deformations.

Meanwhile, fully automatic techniques have been developed to compute proxy bones and their influence weights from mesh animations [James and Twigg 2005]. In [Park and Hodgins 2006], the extracted proxy bones were used for segmenting markers into rigid segments. High quality skin deformations can be reconstructed via a second-order skinning scheme followed by RBF-based interpolation of the residual errors. The algorithm in [Wang et al. 2007] also relies on proxy bones in addition to rotational regression to overcome the limitations of SSD. Although the technique in [James and Twigg 2005] is primarily targeted at articulated models, it can be extended to highly deformable surfaces by fitting a suitable number of bones [Kavan et al. 2007b]. Nevertheless, the relatively large number of necessary proxy bones make it hard to intuitively control the animation of deformable surfaces.

Example-based mesh deformation driven by a small number of translational handles instead of a skeleton have been previously investigated in the framework of MeshIK [Sumner et al. 2005; Der et al. 2006], where deformation gradients are interpolated from example poses and new vertex positions or new bone transformations are solved via a relatively expensive nonlinear optimization, which simultaneously involves all examples. In contrast, this paper adopts a learning approach which has an offline training stage in addition to the run-time animation stage. Because of the precomputation in the training stage, impressive real-time performance has been made possible during the run-time stage. In addition, our method can accommodate a larger number of training examples to faithfully learn a deformation style because its run-time stage has linear scalability in terms of training examples while the time complexity of the algorithm in [Der et al. 2006] is a cubic polynomial of the number of example poses. Thus, our method is better suited for real-time data-driven animation.

Facial expressions are difficult to animate due to correlated deformations in multiple regions, the variety of expressions and the existence of small features such as wrinkles. Blendshape face is a popular real-time technique for facial animation. By establishing relationships between motion capture data and blendshape weights, new facial animations can be generated from facial MoCAP data [Joshi et al. 2003; Deng et al. 2006]. However, unlike our method, these techniques require relatively dense marker movements. In FaceIK [Zhang et al. 2004], spatially varying blending weights based on control point positions are computed to generate novel expressions from multiple example faces. In Face poser [Lau et al. 2007], a nonlinear optimization is formulated in a maximum a posteriori (MAP) framework to find optimal PCA coefficients. These face animation techniques are relatively expensive and are not well suited for real-time applications.

Cloth is another challenge for both deformation acquisition and creation. A pattern-based cloth capturing method has been proposed in [White et al. 2007] to reconstruct a deforming cloth mesh from multiple views, but a real-time interface is still needed to create potentially novel animations consistent with the style of the captured data. On the other hand, the main objective of the editing techniques in [Kircher and Garland 2006; Xu et al. 2007] was to modify an existing mesh animation but not to create a novel one according to control point movements.

This paper also shares the same motivation with previous work on skeletal animation driven by low-dimensional control signals [Grochow et al. 2004; Chai and Hodgins 2005]. Specifically, the method in [Grochow et al. 2004] performs global nonlinear model reduction using a particular form of Gaussian processes. At run-time, a relatively expensive nonlinear optimization is still necessary to reconstruct a complete skeletal configuration from low-dimensional signals. In comparison, our method performs local nonlinear model reduction for every proxy bone using kernel CCA and does not require nonlinear optimization at the run-time. Note that CCA with the linear kernel has been applied in [Dontcheva et al. 2003] for aligning two serial signals in skeletal animation.

3 Overview

Our goal is to achieve real-time example-based surface deformation and animation using sparse control points. The original surface can be an arbitrary deformable surface without a skeleton. Given a set of surface deformation examples, we first build two different abstractions for the surface. The first abstraction is designed as an animation interface. It is a very sparse set of control points whose locations on the surface are such that they can unambiguously convey the intended deformation. The second abstraction is a sparse set of abstract bones whose deformation parameters are collectively used for generating the deformation of every vertex on the surface in real time. To build connections between these two abstractions, a crucial preprocessing step is to train deformation predictors $\mathbf{d}(\mathbf{c})$ from pairs of configurations of control points and bone deformations, $(\mathbf{c}, \mathbf{d})$. At run-time, these predictors effectively generate new bone deformations, and in turn new surface deformations faithful to the style in the training examples, from novel control point movements. Since the prediction models are precomputed, generating new deformations at run-time is efficient and uses little computational resource. The work flow of our system is summarized in Figure 2.

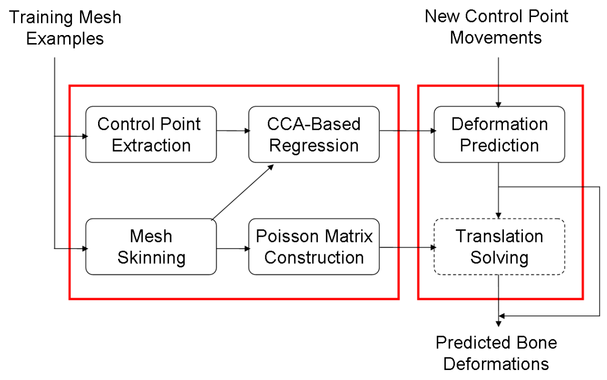

Figure 2: The work flow of our method.

Training Stage. Given meshes with n_p different deformations, we identify several mesh vertices in highly deformable regions and use them as the control points. We adapt the method in [Meyer and Anderson 2007] to guide our control point selection. The method works by performing principal component analysis (PCA) on vertex positions from mesh deformation examples to obtain a set of bases. Varimax rotation is used to rotate each PCA basis vector into a more localized version. Two representative points with largest differences are then selected automatically from each rotated basis vector. Finally, we interactively choose n_c samples from these representative points that are placed near the semantic features of the input mesh such as eyes or the mouth. These points are then concatenated together to form the vector $\mathbf{c}$. Note that user interaction is only for choosing one of multiple points automatically identified near the same semantic feature and can be finished in a few minutes.

We also build a skinned mesh which includes a set of bone deformations and the bone influence weights for each vertex. The skinned mesh is generated by grouping original triangles with similar transformations into the same bone [Wang et al. 2007]. The error $E_{A\rightarrow B}$ of joining bone A to bone B is defined to include not only vertex prediction errors, induced by applying B's transformation to A's vertices, but also edge prediction errors in terms of both edge orientation and length. Each bone acts as an abstract representative for transformations, and its influence weights on a vertex are then obtained by minimizing the fitting error of vertex positions in all examples. In terms of bone transformations, we choose to fit a rigid transformation in the form of a dual-quaternion using the method in [Kavan et al. 2007b].

Once the set of control point sequences and corresponding deformations have been collected for each bone from the examples, we perform nonlinear model reduction using kernel CCA [Hotelling 1936; Melzer et al. 2003] and build a deformation predictor using linear regression. We first transform every data pair,$(\mathbf{c}, \mathbf{d})$, into reduced coordinates $(\mathbf{c_r}, \mathbf{d_r})$ by performing kernel CCA, which finds pairs of projection bases which maximize the correlation between $\mathbf{c_r}$ and $\mathbf{d_r}$. We then apply linear regression on the reduced coordinates $(\mathbf{c_r}, \mathbf{d_r})$ to compute the deformation predictor that maps $\mathbf{c_r}$ to $\mathbf{d_r}$. The mapping is in the form of a regression matrix, and can be computed by minimizing least-square errors. Both the CCA bases and regression matrix are computed for each bone in the skinned mesh.

Run-time Stage. At the run-time, the CCA bases are used to transform new control point coordinates into reduced coordinates. The regression matrix is then applied to the reduced coordinates as the predictor to obtain new dual quaternion transformations for each bone. For highly deformable meshes, instead of using the predicted bone translations, we recompute them by solving the Poisson equation [Yu et al. 2004] in real time to distribute errors more uniformly over the entire mesh as well as to better satisfy positional constraints specified by the control points. Note that we still use the rotational part of the prediction as usual. Most of the run-time process in our method can be formulated as simple matrix-vector multiplications plus kernel function evaluations, which can be implemented on GPUs very efficiently. The run-time performance of our method can achieve hundreds of frames per second in our experiments.

4 CCA Based Regression

We apply a statistically based method called *canonical correlation analysis* [Hotelling 1936] to perform model reduction and obtain optimal basis pairs that reveal the functional dependency between control points and bone deformations. While principal component analysis (PCA) performs feature extraction for a single set of variables, CCA extracts pairs of features that yield maximum correlation between two sets of variables and, thus, is better suited as a preprocessing step for regression. We choose to use CCA because of its ability to capture data dependency while avoiding overfitting. If we directly apply linear regression or kernel-based regression methods such as RBFs on the input data, the resulting predictor can fit the example data very well, but there is the risk of overfitting if the input examples are sparse. The obtained predictor might not be able to learn the essence of the actual deformation model, especially in terms of non-linear facial deformations, and produce unnatural novel deformation sequences (Figure 3). Moreover, for example pairs that are nonlinearly correlated, the kernel trick can be applied in the CCA formulation to establish nonlinear dependency between variables, which enrich the classes of deformation our system can handle.

In the following subsections, we review the mathematical background of CCA, and describe the details of our regression method.

4.1 Canonical Correlation Analysis

Given two sets of variables $\{\mathbf{c}, \mathbf{d}\}$ with $\mathbf{c} \in R^n$ and $\mathbf{d} \in R^m$, CCA finds pairs of bases $\{\mathbf{u_c}, \mathbf{u_d}\}$ such that the projections $c_r = \mathbf{u_c}^T\mathbf{c}$ and $d_r = \mathbf{u_d}^T\mathbf{d}$ have their correlation ρ maximized. Here we follow the derivation from [Melzer et al. 2003]. Specifically,

$$\rho = \frac{E[c_r d_r]}{\sqrt{E[c_r^2]E[d_r^2]}} = \frac{E[\mathbf{u_c}^T\mathbf{c}\mathbf{d}^\mathbf{T}\mathbf{u_d}]}{\sqrt{E[\mathbf{u_c}^T\mathbf{c}\mathbf{c}^\mathbf{T}\mathbf{u_c}]E[\mathbf{u_d}^T\mathbf{d}\mathbf{d}^\mathbf{T}\mathbf{u_d}]}}. \quad (1)$$

The maximization can be formulated as :

$$\max_{\mathbf{u_c},\mathbf{u_d}} \rho = \max_{\mathbf{u_c},\mathbf{u_d}} \frac{\mathbf{u_c}^T\Sigma_{cd}\mathbf{u_d}}{\sqrt{\mathbf{u_c}^T\Sigma_{cc}\mathbf{u_c}\mathbf{u_d}^T\Sigma_{dd}\mathbf{u_d}}} \quad (2)$$

where $\Sigma_{cd} = cov(\mathbf{c}, \mathbf{d})$ is the cross-covariance matrix of $\mathbf{c}, \mathbf{d}$, and Σ_{cc}, Σ_{dd} are defined similarly. The solution for $\{\mathbf{u_c}, \mathbf{u_d}\}$ can be obtained via singular value decomposition (SVD) of the matrix :

$$\Sigma_{cc}^{-\frac{1}{2}}\Sigma_{cd}\Sigma_{dd}^{-\frac{1}{2}} = \mathcal{U}\mathcal{D}\mathcal{V}^\mathcal{T} \quad (3)$$

where $\mathcal{U}, \mathcal{D}, \mathcal{V}$ are the resulting decomposition of SVD. The i-th basis pair can be obtained by computing

$$\mathbf{u_c}^i = \Sigma_{cc}^{-\frac{1}{2}}\mathcal{U}^i \quad (4)$$

and

$$\mathbf{u_d}^i = \Sigma_{dd}^{-\frac{1}{2}}\mathcal{V}^i \quad (5)$$

where $\mathcal{U}^i$ and $\mathcal{V}^i$ are the i-th column of matrices $\mathcal{U}$ and $\mathcal{V}$. Note that there are a maximum number of $\min(m, n)$ basis pairs, where $\min(m, n)$ is equal to the smaller dimension of $\mathbf{c}$ and $\mathbf{d}$.

4.2 The Kernel Trick

Instead of representing CCA bases in a linear subspace, we can also construct a nonlinear version of the algorithm via kernel functions. The kernel method was originally used to extend a support vector machine (SVM) to its non-linear version. It works by mapping the original data into a higher dimensional feature space and solving a corresponding nonlinear version of the problem in that feature space. Suppose $\phi : R^s \rightarrow R^t, t > s$ is a mapping that transforms $\mathbf{x}$ into the feature space. A kernel function, $k(\mathbf{x}, \mathbf{y})$, can be used to define the dot product in the feature space. That is, $k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^T\phi(\mathbf{y})$. In many cases, the kernel function has a simple closed-form expression even when the mapping itself is hard to formulate explicitly. The kernel trick means if the original problem can be reformulated to depend only on the dot product of the original data, the nonlinear version of the problem can be formulated to depend only on the kernel function. As a simple example, let $\phi(\mathbf{x})$ be a mapping which transforms a vector $\mathbf{x} = (x_1, x_2)$ into the vector of all second degree monomials $(x_1^2, x_2^2, x_1x_2, x_2x_1)$. We can clearly see that $\phi(\mathbf{x})^T\phi(\mathbf{y}) = k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y})^2$. Here $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y})^2$ is the second degree polynomial kernel.

The nonlinear version of CCA in our current context can be derived via the kernel trick. Given n_p pairs of example data $\{\mathbf{c}^i, \mathbf{d}^i\}_{i=1}^{n_p}$ with $\mathbf{c}^i \in R^n$ and $\mathbf{d}^i \in R^m$, we define $\mathbf{C} = (\mathbf{c^1} \ldots \mathbf{c}^{n_p})$ and $\mathbf{D} = (\mathbf{d^1} \ldots \mathbf{d}^{n_p})$ as data matrices with their i-th columns being $\mathbf{c}^i$ and $\mathbf{d}^i$, respectively. The covariance matrices for nonlinearly mapped data can be written as :

$$\Sigma_{cd} = \phi(\mathbf{C})\mathbf{D}^T, \Sigma_{cc} = \phi(\mathbf{C})\phi(\mathbf{C})^T, \Sigma_{dd} = \mathbf{D}\mathbf{D}^T,$$

where $\phi(\mathbf{C})$ is a nonlinear version of matrix $\mathbf{C}$ by applying the mapping ϕ on each column of $\mathbf{C}$. We choose to transform only the input matrix $\mathbf{C}$ instead of both $(\mathbf{C}, \mathbf{D})$ because a nonlinear mapping of $\mathbf{D}$ would result in a nonlinear reconstruction at run time from reduced coordinates to original ones, which would be expensive considering the performance requirement of our method. Because the basis pair $\{\mathbf{u_c}, \mathbf{u_d}\}$ always lies in the span of the mapped data $\{\phi(\mathbf{C}), \mathbf{D}\}$, we can further express a pair of bases $\{\mathbf{u_c}, \mathbf{u_d}\}$ as $\mathbf{u_c} = \phi(\mathbf{C})\mathbf{f_c}$ and $\mathbf{u_d} = \mathbf{D}\mathbf{f_d}$, where $\mathbf{f_c}, \mathbf{f_d} \in R^{n_p}$ are coefficient vectors with their dimensions equal to the number of example pairs. Therefore we can write the nonlinear version of (2) using the kernel as follows:

$$\max_{\mathbf{f_c}, \mathbf{f_c}} \rho = \max_{\mathbf{f_c}, \mathbf{f_c}} \frac{\mathbf{f_c}^T K_c K_d \mathbf{f_d}}{\sqrt{\mathbf{f_c}^T (K_c)^2 \mathbf{f_c} \mathbf{f_d}^T (K_d)^2 \mathbf{f_d}}} \tag{6}$$

where $K_c = \phi(\mathbf{C})^T\phi(\mathbf{C})$ and $K_d = \mathbf{D}^T\mathbf{D}$. Note that the entries of K_c can be computed by the kernel function k_c instead of by $\phi(\mathbf{c})$ explicitly. This is the dual form of (2), and can be solved in a similar manner via SVD as (4) and (5).

Suppose we retain n_f pairs of coefficient vectors from SVD to form two matrices, $\mathbf{F_c} = (\mathbf{f_c}^1 \ldots \mathbf{f_c}^{n_f})$ and $\mathbf{F_d} = (\mathbf{f_d}^1 \ldots \mathbf{f_d}^{n_f})$. The projection of an input $\mathbf{c}$ onto the basis $\mathbf{u_c}^i$ can be computed as

$$\phi(\mathbf{c})^T\mathbf{u_c}^i = \sum_{j=1}^{n_p} f_{cj}^i \phi(\mathbf{c})^T\phi(\mathbf{c}^j) = \sum_{j=1}^{n_p} f_{cj}^i k_c(\mathbf{c}, \mathbf{c}^j).$$

Thus, the vector of reduced coordinates, $\mathbf{c_r} \in R^{n_f}$, can be expressed in a matrix-vector form as follows.

$$\mathbf{c_r} = \mathbf{F_c}^T \xi_\mathbf{c}, \tag{7}$$

where $\xi_\mathbf{c} \in R^{n_p}$ is the kernelized vector whose j-th entry is $k_c(\mathbf{c}, \mathbf{c^j})$.

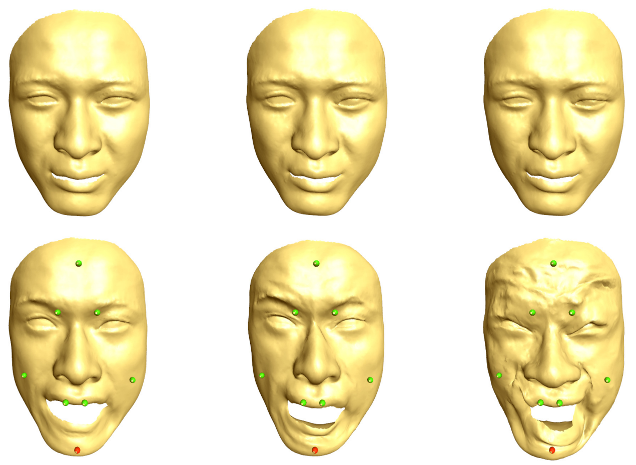

Figure 3: A comparison among our method, regression based on both PCA and RBFs, and direct RBF-based regression. The top row shows the fitting quality of a training example, and the bottom row shows a predicted deformation once the user pulls a control point. The colored spheres are the control points, with the red one indicating the point being edited by the user. All three methods can fit the training data very well without noticeable differences. However, RBF and joint PCA-RBF regression fail to generalize beyond original examples and produce distorted results.

4.3 Regression

We use regression to build up connections between control point coordinates and bone deformations. We use dual quaternions for representing bone deformations [Kavan et al. 2007a]. A dual quaternion $\mathbf{q_0} + \epsilon\mathbf{q}_\epsilon$ represents a rigid transformation. It involves two classic quaternions, $\mathbf{q_0}$ and $\mathbf{q}_\epsilon$, therefore, a bone deformation involves eight parameters, $\mathbf{d} \in R^8$. Linear blending of dual quaternions can be applied to blend multiple corresponding rigid transformations, and the resulting dual quaternion still represents a rigid transformation. We choose to represent a bone deformation using a dual quaternion instead of an affine transformation matrix because it has better interpolation property than the transformation matrix and can avoid the "candy wrapper" artifact [Kavan et al. 2007b] when used for skinning. It also has fewer parameters than an affine transformation matrix, and is therefore a more suitable choice for regression.

Regression is performed once for each bone. The outcome of regression is a predictor, $\mathbf{d}(\mathbf{c})$, that is able to predict a bone deformation, $\mathbf{d}$, given the concatenation of all control point coordinates, $\mathbf{c} \in R^{n_c \times n_{DOF}}$ where n_c is the number of control points and n_{DOF} is the degree of freedom of each control point. Note that a control point can have at most 3 DOFs in its 3D position. Given n_p surface deformation examples, we first extract from each example a pair of data, $(\mathbf{c}^i, \mathbf{d}^i)$ for the bone. Kernel CCA is then performed on all extracted data pairs to obtain the set of n_f CCA bases and the collection of reduced coordinates, $\mathbf{C_r} \in R^{n_f \times n_p}$ and $\mathbf{D_r} \in R^{n_f \times n_p}$, of all data pairs after projection onto the bases. While PCA can only have a common set of bases for all bones, there is a distinct set of CCA bases specifically tailored for each bone. Given those pairs of reduced coordinates, we compute an optimal predictor $\mathbf{B_c} \in R^{n_f \times n_f}$ by performing the following linear regression:

$$\min_{\mathbf{B_c}} \sum_{i=1}^{n_p} \|\mathbf{B_c}\mathbf{c_r}^i - \mathbf{d_r}^i\|^2 \tag{8}$$

where $\mathbf{c_r}^i$ and $\mathbf{d_r}^i$ are the i-th column of $\mathbf{C_r}, \mathbf{D_r}$, respectively. We

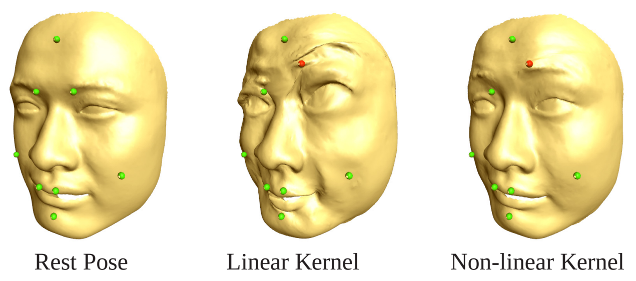

Figure 4: A comparison of kernel functions in our CCA-based model reduction. Predictors based on nonlinear polynomial kernels produce higher quality deformations without artifacts when compared with results based on the linear kernel.

choose a simple linear predictor because the example data pairs have already been nonlinearly transformed using kernel CCA to maximize their dependency and linear regression can already produce accurate predictions in all our experiments.

Since the output deformation from our predictor is in the form of reduced coordinates as well, they need to be used to further reconstruct the eight parameters of a dual quaternion. Different from PCA, the bases generated by CCA are not necessarily orthogonal to each other. Therefore we cannot directly reconstruct these parameters by considering the reduced coordinates as weights over the CCA bases. In theory, dual bases of the CCA bases need to be computed to achieve accurate reconstruction. Since we do not kernelize the example deformations in our CCA formulation, the matrix of dual bases can be represented as a linear mapping $\mathbf{H_d} \in R^{8\times n_f}$ which transforms the reduced coordinates $\mathbf{D_r}$ to optimally approximate the example deformations $\mathbf{D}$. Thus, it is the solution of the following linear least-squares problem:

$$\min_{\mathbf{H_d}} \sum_i \|\mathbf{H_d}\mathbf{d_r}^i - \mathbf{d}^i\|^2. \tag{9}$$

Note that since the matrices $\mathbf{F_c}, \mathbf{B_c}, \mathbf{H_d}$ are consecutive linear mappings, we can concatenate them into a single linear operator $\mathbf{M_b} = \mathbf{H_d}\mathbf{B_c}\mathbf{F_c}$ that predicts the resulting deformation $\mathbf{d}$ given the kernelized vector $\xi_\mathbf{c}$ from the input $\mathbf{c}$. In our run-time implementation, this combined predictor $\mathbf{M_b}$ is used in place of $\mathbf{F_c}, \mathbf{B_c}$ and $\mathbf{H_d}$ for efficient execution.

We have verified the merit of CCA-based nonlinear model reduction by comparing our results with regression based on radial basis functions (RBF). We tested both direct RBF regression and RBF-based regression from reduced coordinates obtained by applying PCA to control point configurations. In both schemes, a regression model is computed by minimizing least-squares errors. As we can see from Figure 3, although the fitting error for the training examples is small in all of the methods, the direct regression scheme tends to overfit the training data and produce unnatural predicted results. On the other hand, while PCA-based model reduction can prevent overfitting, there are still distortions in the predicted results. This is because the same set of PCA bases are used in the regression for all bone transformations, and it cannot necessarily give rise to a good functional dependency for every bone.

We have also compared the quality of predicted deformations from linear and polynomial kernels in CCA-based model reduction. As shown in Figure 4, for certain deformation styles such as facial animation, nonlinear kernels yield more natural results.

5 Poisson Translation Solver

At run-time, the transformation for each bone is generated independently using the predictor learned in the previous section. Usually we can directly use this resulting transformation for dual quaternion skinning. However, when nearby bones undergo very different 3D rotations, small prediction errors in translation may be amplified in the visual results and create artifacts in regions jointly controlled by multiple bones. We solve this problem by computing a new set of translations for the bones in a Poisson formulation. The motivation is that while bone rotations define deformed local shapes, it is the bone translation that integrates these local shapes together. By solving for new translations, we can ensure that the resulting transformations will be consistent among nearby bones while preserving the deformed local shapes from our prediction. In addition, control point positions can be enforced as soft constraints in the Poisson formulation.

As a preprocessing step for the Poisson solver, we perform linear blend skinning for both vertices and edges on the deformable mesh surface. Let v_i^0 be the rest position of the i-th vertex v_i, $\{w_i^b\}_{b=1}^{n_b}$ be its skinning weights for all bones, and $\{\mathbf{R_b}, \mathbf{t_b}\}_{b=1}^{n_b}$ be the rotation matrices and translation vectors of all bones, the skinned vertex position $v_i^s = \sum_b \mathbf{w}_i^b(\mathbf{R_b}v_i^0 + \mathbf{t_b})$. We also fit a set of skinning weights $\{\hat{w}_k^b\}_{b=1}^{n_b}$ for each edge $e_k = v_{k0} - v_{k1}$ in the mesh. We treat $\hat{w}_k^b$ as a skinning weight for edge e_k analogous to w_i^b for v_i. Specifically, we compute $\{\hat{w}_k^b\}_{b=1}^{n_b}$ by minimizing the following fitting error of the edge:

$$\min_{\hat{\mathbf{w}}_\mathbf{k}} \sum_{j=1}^{n_p} \|e_k^j - \sum_{b=1}^{n_b} \hat{w}_k^b(\mathbf{R_b}{}^j e_k^0)\|^2 \tag{10}$$

Note that we use the same set of bone transformations when fitting weights for edges. Since the edge orientation is a vector, we have removed the translation component in the error formulation.

Our Poisson-based translation solver is formulated as an optimization problem which minimizes the differences of two edge predictions. The objective function of this minimization is formulated as follows.

$$\min_{\mathbf{t}} \sum_{k=1}^{n_e} \|(v_{k0}^s - v_{k1}^s) - e_k^s\|^2 + \beta \sum_{l=1}^{n_c} \|v_{i_l}^s - c_l\|^2 \tag{11}$$

where β is a weighting factor for positional constraints, $e_k^s = \sum_b \hat{w}_k^b(\mathbf{R_b}e_k^0)$ is the skinned edge representation, c_l represents the current position of the l-th control point, v_{i_l} is the vertex corresponding to the l-th control point, and n_e is the number of edges. Here the rotation matrix $\mathbf{R_b}$ is directly from prediction, and new translation vectors $\mathbf{t} = \{t_0 \ldots t_{n_b}\}$ are sought as the solution of this minimization. It can be easily verified that (11) is actually a linear least-squares problem, $\min_\mathbf{t} \|\mathbf{At} - \mathbf{T}\|^2$, where $\mathbf{T}$ contains all bone rotation matrices and control point positions, and matrix $\mathbf{A}$ can be computed from rest-pose vertex positions and the skinning weights for both vertices and edges. Since entries in matrix

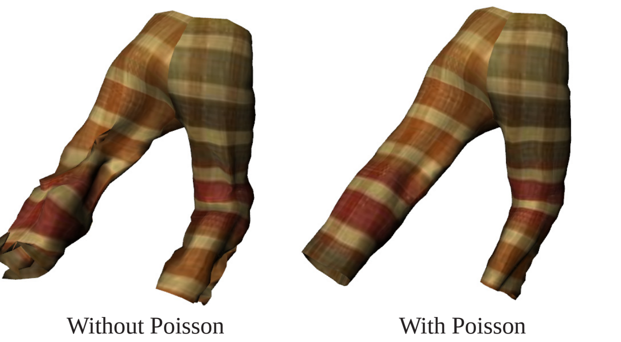

Figure 5: Deformation prediction without an additional translation solver may yield distorted results. However, once our Poisson-based translation solver has been applied, the resulting deformation becomes natural without artifacts.

$\mathbf{A}$ do not change at run-time, we precompute its pseudo inverse $\mathbf{P} = (\mathbf{A^TA})^{-1}\mathbf{A}^T$ in a preprocessing stage. Therefore the solution of (11) can be directly found by a matrix-vector multiplication $\mathbf{t} = \mathbf{PT}$. This simple solution technique makes it straightforward to implement on GPUs.

The insightful reader might notice that dual-quaternion skinning has been replaced by linear blend skinning in our Poisson formulation. The reason for this approximation is that the evaluation of dual quaternions makes the solution nonlinearly depend on the predicted rotations. Thus the problem becomes a nonlinear optimization, which is much more expensive to solve at run-time. In practice, we have found the resulting translations from this approximation works very well in all our experiments.

To validate our translation solving process, we have compared the predicted deformations from novel control point movements with and without the translation solver. As we can see in Figure 5, the predicted deformation is much more distorted without the Poisson-based translation solver because of the inconsistency of the translations among nearby bones. On the other hand, new translations obtained from our translation solver can better position nearby bones and are much more natural visually.

Remark A solution in the form of matrix-vector multiplication for the Poisson problem has previously been used for solving for transformations of a reduced deformable model [Wang et al. 2007]. However, their method was derived in the context of deformation gradients and both the new matrix transformations and translations were solved. In our method, we propose a novel view of the problem by treating edges as additional elements for skinning, and therefore do not require any specific treatment for deformation gradients or recomputing the rotation matrices.

6 GPU Implementation

Since all the run-time components can be reduced to linear operations, we can implement most of them efficiently on GPUs. During each frame, the data sent to the GPU from CPU only includes the current control point coordinates, which have less than 150 bytes in all of our examples. Both the deformation prediction and translation solving processes are performed entirely on GPUs.

The overall GPU implementation can be separated into three stages: two matrix-vector multiplications for CCA-based prediction, one matrix-vector multiplication for solving translations, and a dual-quaternion skinning step in a vertex shader. In both prediction and translation solving steps, there are additional kernel transformations and quaternion conversion for each bone. But overall the computation is dominated by straightforward matrix-vector multiplications. In our implementation, we use CUDA [nVidia CUDA], a general GPU programming framework by nVidia, for the first two GPGPU stages.

6.1 Deformation Prediction

Given uploaded control point coordinates $\mathbf{c}$, we first compute the kernelized vector $\xi_\mathbf{c}$ from data matrix $\mathbf{C}$, which is defined at the end of Section 4.2. The predicted deformation $\mathbf{d_b}$ for bone b can then be obtained via a matrix-vector multiplication $\mathbf{d_b} = \mathbf{M_b}\xi_\mathbf{c}$, where $\mathbf{M_b}$ is defined in Section 4.3 as the product of multiple matrices. For computational efficiency, we concatenate $\mathbf{M_b}$ from all bones into one large matrix $\mathbf{M}$, and perform the multiplication only once per pass. Note that both $\mathbf{M}$ and $\mathbf{C}$ can be precomputed and preloaded to GPU before the run-time process.

6.2 Translation Solving

Once we have obtained the predicted dual quaternions, we convert their non-dual parts into rotation matrices and concatenate them into a single vector $\mathbf{T}$. Similar to the previous stage, we can precompute and preload the pseudo inverse matrix $\mathbf{P}$ to the GPU memory. The translations $\mathbf{t}$ can then be computed directly by $\mathbf{t} = \mathbf{PT}$. Once we have obtained $\mathbf{t}$, we use it to generate a new set of dual quaternions $\mathbf{d}'$ by updating the translation part of $\mathbf{d}$ with $\mathbf{t}$. We map the resulting dual quaternions $\mathbf{d}'$ into an OpenGL texture buffer to make it available in the skinning stage.

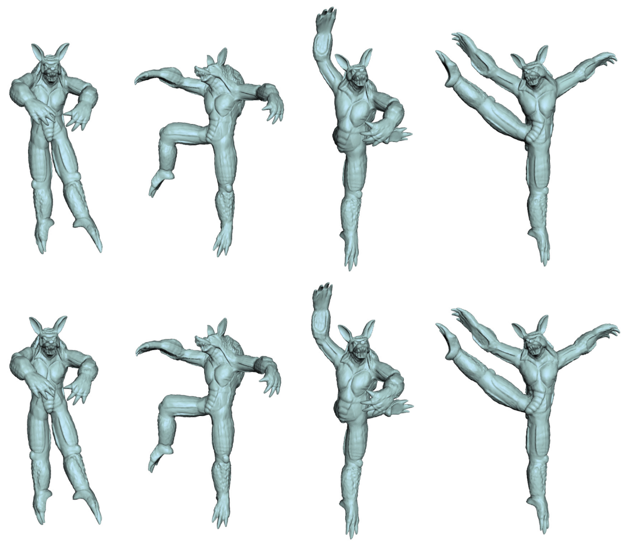
Figure 6: Comparisons with ground truth on articulated mesh deformation. The top row shows the ground truth which was not part of the training data, and the bottom row shows synthetically generated results from our method with the Gaussian kernel.

6.3 DQ-Palette Skinning

The last part of our GPU implementation performs dual-quaternion skinning [Kavan et al. 2007a] to actually deform the mesh. Dual-quaternion skinning for vertex v_i is formulated as

$$\hat{v_i^s} = (\sum_{b=1}^{n_b} w_i^b \mathbf{d_b})\hat{v_i^0}\overline{(\sum_{b=1}^{n_b} w_i^b \mathbf{d_b})}^{-1}$$

where w_i^b is a vertex skinning weight, $\bar{\mathbf{d}} = \mathbf{q_0} - \epsilon\mathbf{q}_\epsilon$ is the dual quaternion conjugation, $\mathbf{d^{-1}} = \mathbf{d}^* = \mathbf{q_0}^* + \epsilon\mathbf{q}_\epsilon^*$ is the inverse of unit dual-quaternion $\mathbf{d}$ with $\|\mathbf{d}\| = 1$, and $\hat{v} = 1 + \epsilon(v_x i + v_y j + v_z k)$ is the dual quaternion representation for vertex $v = (v_x, v_y, v_z)$.

We perform the above skinning in a vertex shader in an intermediate pass before normal computation and shading. The rest-pose vertex positions and their skinning weights are preloaded to GPU as a Vertex Buffer Object (VBO). In the vertex shader, we compute at each vertex a weighted sum of revised dual quaternions from the previous pass. The interpolated dual quaternion is then applied to the vertex to obtain its deformed position. In a final rendering pass, per-vertex normal vectors are computed on the fly using the deformed vertex positions from the previous pass. This results in accurate normal vectors for shading.

7 Experimental Results

We chose three different types of example deformations, including facial animation, articulated mesh animation, and secondary deformation of clothing driven by underlying articulated motion, for our experiments. Among them, the facial animation and clothing deformation examples were real-world data acquired using computer vision techniques [Zhang et al. 2004; White et al. 2007]. In the CCA-based regression stage, we chose the fifth degree inhomogeneous polynomial kernel, $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y} + c)^5$, for facial animation, and the Gaussian kernel for other deformations. The differences between these two kernel functions are not significant in predicted results. The Gaussian kernel worked well in all our

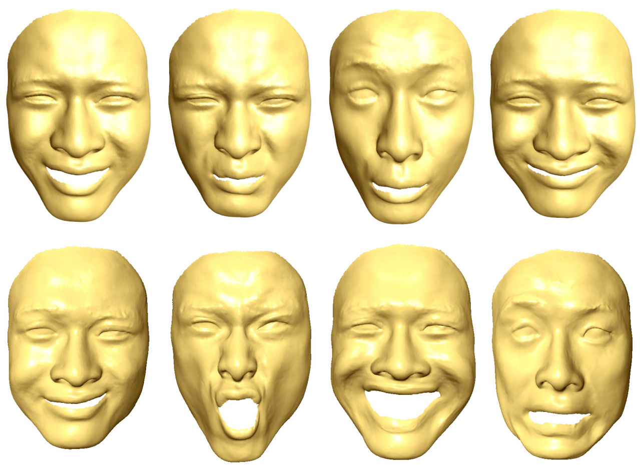

Figure 7: Frames from predicted facial deformations generated using our method with a polynomial kernel. The frames in the top row belong to a testing facial animation sequence. The bottom row shows novel expressions generated from interactive editing.

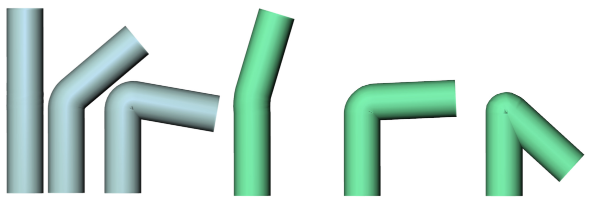

Figure 9: A bending style is trained from given examples to generate novel deformations with the same style. The blue models on the left are training examples. The green ones on the right are novel deformations generated using the Gaussian kernel.

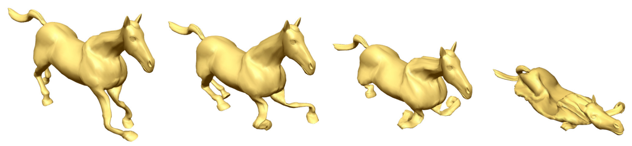

Figure 10: A simultaneous galloping and collapsing sequence generated from a deformation predictor trained using two separate galloping and collapsing sequences. The Gaussian kernel is used.

experiments while the polynomial kernel produced slightly better result for facial animation.

The parameters of the kernel functions are estimated automatically. For the Gaussian kernel, its standard deviation is set to the standard deviation of control point positions from the training examples. In the chosen polynomial kernel, c is set to the average norm of control points scaled by a constant factor (0.1 in our case).

At run-time, novel bone deformations can be predicted in real time in our system (Table 1) with novel control point movements. We demonstrate this by generating novel animated surface deformation with the motion trajectories of a sparse (< 10) set of control points. Here we obtain the control point trajectories directly from the testing mesh sequences though they can also be marker movements from MoCAP data. Although surface deformation is underdetermined with such a sparse set of constraints in the original deformation space, our regression model is able to predict the deformations from the learned subspaces based on the training examples. The resulting deformations are shown in Figures 6, 7, and 8. The top rows of Figures 6 and 8 show ground truth that was not part of the training data. We have also designed an interface to let the user drag any of the control points either in a 3D space or on a 2D projection plane. The interactively defined control point positions can also be used for generating novel surface deformations.

We show the capability of our method in learning different deformation styles in Figure 9 and 15. Given sparse training examples of a bending style, our method can produce novel deformations with the same style. We can also combine examples of different deformation styles to produce a hybrid deformation predictor. In Figure 10, a simultaneous galloping and collapsing sequence is generated from our predictor using training examples from separate horse galloping and collapsing sequences.

We have compared our results with FaceIK [Zhang et al. 2004] and PCA-based blendshape on facial animation. We used the same set of control points and their moving sequences for all three methods and recorded the prediction errors for each method. The RMS errors for FaceIK, PCA-based blendshape and our method are 0.0242, 0.0143 and 0.0108, respectively, when the largest dimension of the bounding box of the face is scaled to have a unit length. Visual comparison results are also shown in Figure 11. While the numerical errors are not large for all methods, the visual result from our method is more natural and closer to the ground truth than others. FaceIK is not a real-time technique and sometimes produces results with obvious distortion around facial features such as the mouth. For PCA-based blendshape, we built a set of blendshape bases using PCA and solved for optimal blendshape weights from new control point positions using least-squares. Be-

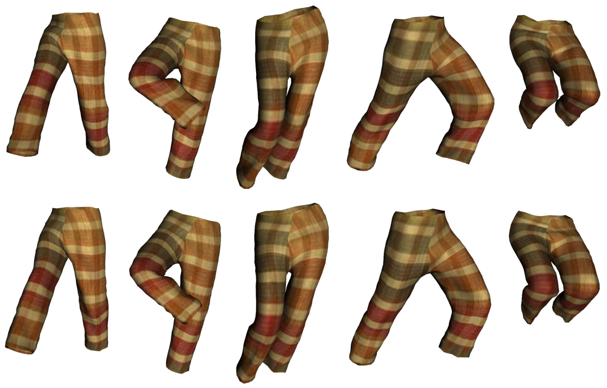

Figure 8: Comparisons with ground truth on deformation of clothing. The top row shows the ground truth which was not part of the training data, and the bottom row shows synthetically generated results from our method with the Gaussian kernel.

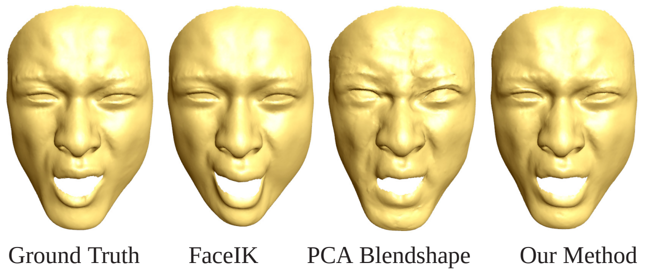

Figure 11: A comparison of predicted deformation among our method, FaceIK [Zhang et al. 2004] and PCA-based blendshape. Our method generalizes well within the deformation subspace learned from training examples, and produces results closer to the ground truth than the other two methods.

Examples	Vertices	Bones	cpts	#CCA	#Train	#Test	Prep	fps
Face	23,728	260	8	8	38	384	20 min	502
Pants	1,453	150	7	8	80	1691	15 min	466
Armadillo	33,000	80	5	8	29	298	18 min	408
Horse	8,431	150	8	8	27	n/a	13 min	506
Cylinder	2,000	40	2	2	3	40	2 min	798
Bar	8,000	90	2	2	3	40	6 min	632

Table 1: Statistics and Timings. All performance measurements were taken from a 3.0GHz Pentium D processor with nVidia Geforce 8800GTS 640MB VRAM. 'cpts' means the number of control points, '#CCA' means the number of CCA bases used for each bone, '#Train' means the number of training examples, '#Test' means the number of testing examples, and 'Prep' means the total time for all preprocessing steps. Our Poisson-based translation solver were not used for Face and Cylinder. Instead of the total number of vertices, the number of bones, training examples and the translation solver are more influential factors affecting the final frame rate.

cause the sparse set of control points are not sufficient to robustly solve for the blendshape weights, there are visible artifacts in the resulting deformations. We have also compared the performance of our method with Face poser [Lau et al. 2007]. In their method, a nonlinear optimization is performed on CPU at run-time and requires significantly more time than our method. It can only solve for less than six frames per second while our method is around two orders of magnitude faster.

We have compared the accuracy of our regression with SAD [Kavan et al. 2007b] using the same number of proxy bones. Since SAD cannot be directly used for generating novel deformations, we only perform the comparison on the training examples. Our results were obtained by predicting the deformations from control point configurations in the training examples, and SAD results were from direct skinning of training examples. The comparison is shown in Figure 12, where our method can faithfully reproduce the deformations from original training examples after regression while SAD fails to accurately fit the training examples with the same number of bones. Since SAD uniformly places proxy bones on the mesh, its skinning results are less optimal.

To demonstrate the scalability of our method, we have tested the performance of our system with an increasing number of proxy bones and training examples. As shown in Figure 13, our system can still achieve more than 100 fps even for the extreme case of 650 bones with the Poisson-based translation solver, which has a quadratic dependence on the number of bones. In Figure 14, we demonstrate the performance of our system by generating the deformations for a group of 15 pairs of pants on the fly at 35 fps.

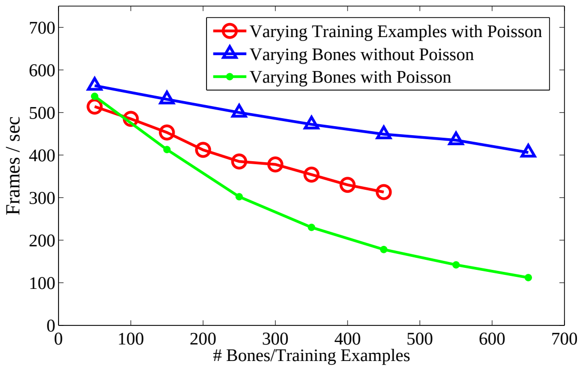

Figure 13: Performance plots of our method with or without the Poisson-based translation solver, using an increasing number of bones and training examples. For plots with a varying number of bones, we use 38 training examples for the regression model. For the plot with a varying number of training examples, we use 100 proxy bones.

8 Conclusions and Future Work

We have presented an intuitive and powerful user interface to simultaneously control the deformation of an entire deformable surface with a minimal number of control points. Our contributions include a novel deformation regression method based on kernel CCA, a Poisson-based translation solving technique, and an efficient GPU implementation. Our run-time algorithm can achieve a few hundred frames per second even for large datasets with hundreds of examples. Comparisons show our method can achieve better results than existing ones on challenging tasks such as handle-based facial animation.

There exist a few limitations in our method that need further research. First, the control points need to be specified before regression and fixed in subsequent animations. However, our current regression implementation only took less than 30 seconds in all of our experiments. With further optimization, it is possible to change control points and rebuild the regression model at run-time. Second, the predicted deformations are not always localized when moving control points. This is undesirable when precise control is necessary. However, in typical data-driven animations, the movements of control points are not independent but highly correlated. Therefore, such a limitation would not create serious problems in practice. Finally, since our method is data-driven, the quality of predicted deformations depend on training examples. As shown in Figure 15, when examples are completely missing in certain directions, our system can only generate a simple shear. This can be

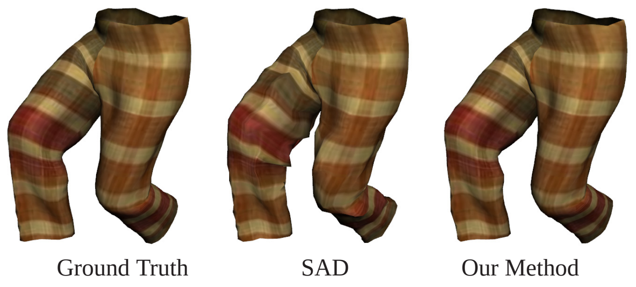

Figure 12: A comparison of fitting quality between our method and SAD [Kavan et al. 2007b]. Our method generates more accurate and natural results.

Figure 14: A group of 15 pairs of pants are animated simultaneously at 35 FPS. Deformations for individual pairs are generated independently in real time.

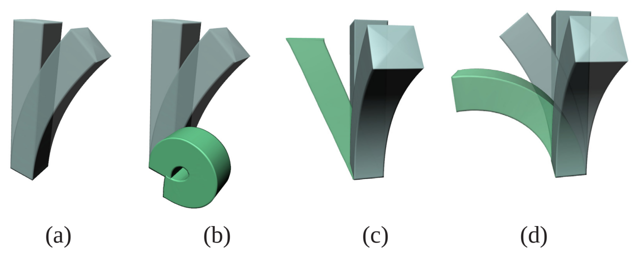

Figure 15: Our data-driven method depends on training examples in (a) to produce an extreme new deformation in (b), which demonstrates a strong extrapolation capability. However, if examples are completely missing in the perpendicular direction, the predicted deformation becomes a simple shear as shown in (c), which can be improved by inserting an extra training example in that direction as shown in (d).

alleviated by adding extra examples in those directions. Since our system is highly scalable with respect to the number of training examples, adding a few extra examples would not hurt performance in practice.

Acknowledgments

We would like to thank the anonymous reviewers for valuable comments and the authors of [Sumner and Popović 2004; Zhang et al. 2004; Shi et al. 2006; White et al. 2007] for sharing the mesh sequences used in our experiments. The second author was supported by the Korea Research Foundation Grant funded by the Korean Government (MOEHRD) (KRF-2007-357-D00213).

References

ANGUELOV, D., SRINIVASAN, P., KOLLER, D., THRUN, S., RODGERS, J., AND DAVIS, J. 2005. Scape: shape completion and animation of people. *ACM Transactions on Graphics 24*, 3, 408–416.

CHAI, J., AND HODGINS, J. 2005. Performance animation from low-dimensional control signals. *ACM TOG 24*, 3, 686–696.

DENG, Z., CHIANG, P.-Y., FOX, P., AND NEUMANN, U. 2006. Animating blendshape faces by cross-mapping motion capture data. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, 43–48.

DER, K., SUMNER, R., AND POPOVIĆ, J. 2006. Inverse kinematics for reduced deformable models. *ACM Transactions on Graphics 25*, 3, 1174–1179.

DONTCHEVA, M., YNGVE, G., AND POPIVIC, Z. 2003. Layered acting for character animation. *ACM TOG 22*, 3, 409–416.

GROCHOW, K., MARTIN, S., HERTZMANN, A., AND POPIVIC, Z. 2004. Style-based inverse kinematics. *ACM TOG 23*, 3, 520–529.

HOTELLING, H. 1936. Relations between two sets of variates. *Biometrika 28*, 321–377.

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

JOSHI, P., TIEN, W., DESBRUN, M., AND PIGHIN, F. 2003. Learning controls for blend shape based realistic facial animation. In *Proceedings of the 2003 Eurographics/SIGGRAPH symposium on computer animation*, 162–174.

KAVAN, L., COLLINS, S., ZARA, J., AND O'SULLIVAN, C. 2007. Skinning with dual quaternions. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, 39–46.

KAVAN, L., MCDONNELL, R., DOBBYN, S., ZARA, J., AND O'SULLIVAN, C. 2007. Skinning arbitrary deformations. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, 53–60.

KIRCHER, S., AND GARLAND, M. 2006. Editing arbitrarily deforming surface animations. *ACM Transactions on Graphics 25*, 3, 1098–1107.

LAU, M., CHAI, J., XU, Y.-Q., AND SHUM, H.-Y. 2007. Face poser: Interactive modeling of 3d facial expressions using model priors. In *ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA 2007)*, 161–170.

MAGNENAT-THALMANN, N., LAPERRIRE, R., AND THALMANN, D. 1988. Joint-dependent local deformations for hand animation and object grasping. In *Graphics Interface*, 26–33.

MELZER, T., REITERA, M., AND BISCHOFB, H. 2003. Appearance models based on kernel canonical correlation analysis. *Pattern Recognition 36*, 9, 1961–1971.

MEYER, M., AND ANDERSON, J. 2007. Key point subspace acceleration and soft caching. *ACM Transactions on Graphics 26*, 3, 74.1–74.8.

MOHR, A., AND GLEICHER, M. 2003. Building efficient, accurate character skins from examples. *ACM Transactions on Graphics 22*, 3, 562–568.

NVIDIA CUDA. Compute unified device architecture (cuda). http://developer.nvidia.com/object/cuda.html.

PARK, S., AND HODGINS, J. 2006. Capturing and animating skin deformation in human motion. *ACM Transactions on Graphics 25*, 3, 881–889.

SHI, L., YU, Y., BELL, N., AND FENG, W.-W. 2006. A fast multigrid algorithm for mesh deformation. *ACM Transactions on Graphics 25*, 3, 1108–1117.

SUMNER, R., AND POPOVIĆ, J. 2004. Deformation transfer for triangle meshes. *ACM Transactions on Graphics 23*, 3, 397–403.

SUMNER, R., ZWICKER, M., GOTSMAN, C., AND POPOVIĆ, J. 2005. Mesh-based inverse kinematics. *ACM Transactions on Graphics 24*, 3, 488–495.

WANG, R., PULLI, K., AND POPOVIĆ, J. 2007. Real-time enveloping with rotational regression. *ACM Transactions on Graphics 26*, 3, 73.1–73.9.

WEBER, O., SORKINE, O., LIPMAN, Y., AND GOTSMAN, C. 2007. Context-aware skeletal shape deformation. *Computer Graphics Forum (Eurographics 2007) 26*, 3, 265–274.

WHITE, R., CRANE, K., AND FORSYTH, D. 2007. Capturing and animating occluded cloth. *ACM Transactions on Graphics 26*, 3, 34.1–34.8.

XU, W., ZHOU, K., YU, Y., TAN, Q., PENG, Q., AND GUO, B. 2007. Gradient domain editing of deforming mesh sequences. *ACM Transactions on Graphics 26*, 3, 84.1–84.10.

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 Transactions on Graphics (special issue for SIGGRAPH 2004) 23*, 3, 641–648.

ZHANG, L., SNAVELY, N., CURLESS, B., AND SEITZ, S. M. 2004. Spacetime faces: High-resolution capture for modeling and animation. *ACM Transactions on Graphics 23*, 3, 548–558.