Vis Comput
DOI 10.1007/s00371-010-0474-6

ORIGINAL ARTICLE

Real-time data driven deformation with affine bones

Byung-Uck Kim · Wei-Wen Feng · Yizhou Yu

Abstract Data driven deformation is increasingly important in computer graphics and interactive applications. From given mesh example sequences, we train a deformation predictor and manipulate a specific style of surface deformation interactively using only a small number of control points. The latest approach of learning the connection between rigid bone transformations and control points uses a statistically based framework, called *canonical correlation analysis*. In this paper, we extend this approach to a skinned mesh with affine bones, each of which conveys a nonrigid affine transformation. However, it is difficult to discover the underlying relationship between control points and nonrigid transformations. To address this issue, we present a two-layer regression framework; one layer being from control points to rigid and the other layer being from rigid to nonrigid transformations. Our contributions also include bone-vertex weight smoothing, enabling the distribution of each bone's influence across neighboring vertices. We can alleviate distortion around regions where nearby bones undergo various transformations and improve deformations reaching beyond the learned subspaces. Experimental results show that our method can achieve more general deformations including flexible muscle bulges or twists. The performance of our implementation is comparable to the latest approach.

B.-U. Kim (✉) · W.-W. Feng · Y. Yu
Department of Computer Science, University of Illinois at Urbana-Champaign, 201 N. Goodwin Ave., Urbana, IL 61801, USA
e-mail: kbu@illinois.edu

W.-W. Feng
e-mail: wfeng2@illinois.edu

Y. Yu
e-mail: yyz@illinois.edu

Published online: 17 April 2010

Keywords Deformation · Canonical correlation analysis · Regression · Weight smoothing

1 Introduction

With recent advances in data acquisition technology, data driven techniques become increasingly important in computer graphics and interactive applications since we can apply them to effectively process captured mesh sequences for generating various animations. One of the important techniques is automatic skinning for arbitrary mesh animations. From a given skeleton-free mesh example sequence, we can build a skinned mesh with proxy bones by assigning triangles with similar transformations into the same proxy bone cluster. Then we can estimate bone transformations and fit their influence weights at each vertex by minimizing the error of linear blend skinning. At the run-time stage, each vertex can be approximated by linearly blending a weighted sum of multiple bone transformations. It provides a compact representation for general mesh animations.

Data driven deformation works by building a mapping from lower dimensional control signals, which could be a few control points, to surface deformations based on training mesh poses. The goal is to learn a deformation predictor that can faithfully reproduce the deformation style and to achieve high performance at the run-time stage. The utilization of a skinned mesh with proxy bones provides an important model reduction for data driven deformation by finding the mapping between the control points and bone transformations. The work of [6] reveals that a surface deformation exhibits correlation among different surface regions and its relation with control points can be formulated effectively in their reduced subspaces using a statistically based framework, called *Canonical Correlation Analysis* (CCA). The

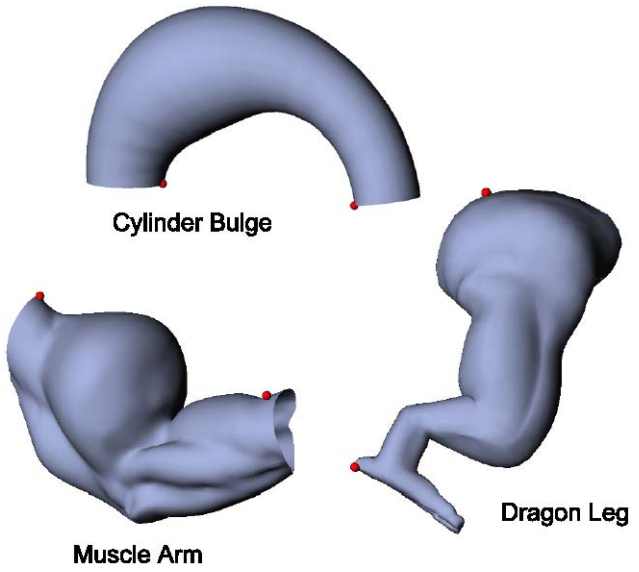


Fig. 1 Novel mesh deformations with our approach. We use one structural model, *Cylinder Bulge*, and two anatomical models, *Muscle Arm* and *Dragon Leg*. All three deformations are not presented in the original mesh sequences or animations. The vertices *in red* are a set of control points guiding surface deformation

deformation predictor from CCA based regression has been one of the successful tools in real-time data driven deformation. However, each bone transformation is restricted to be rigid. This means nonrigid components such as shear and scale have to be implicitly encoded in the translation components when fitting the proxy bone transformations. This restricts the resulting deformation to be less flexible. The situation becomes even worse in cases where we compute a new set of translations through the Poisson solver since we lose the nonrigid information encoded in the previously predicted bone translations.

In this paper, we extend the previous approach to a skinned mesh with affine bones, each of which conveys a nonrigid affine transformation. From the benefits of affine bones, we can achieve a better approximation of mesh animations and reproduce flexible deformations such as muscle bulges or volume changes. However, a direct regression for affine transformations are less meaningful and makes it difficult to discover the underlying relationship between control points and affine bones. To address this, we present a two-layered regression framework for the deformation predictor. We perform the CCA based regression from control points to rigid components in the first layer, followed by a regression from rigid to nonrigid transformations in the second layer, where the second regression can be regarded as the indirect mapping from control points to nonrigid components. In addition, we are free from the side effects of the Poisson based translation solver since non-rigid information is dealt with separately in our training stage, and does not rely on bone translations to represent shear and scale components.

Another contribution of our paper is using bone-vertex weight smoothing (BVWS) to distribute each bone's influence effectively across neighboring vertices. Previous methods compute bone influence weights by minimizing the least square fitting errors. While this produces optimal fitting for the input mesh poses, it cannot guarantee to produce new deformation results without artifacts when the bone transformations are generated on the fly. We observe that if the neighboring vertices are not smoothly influenced by different bones and these bones undergo various transformations, then the blending result may be prone to unpleasant artifacts. Thus, we develop a weight fitting scheme based on Laplacian mesh smoothing to compute a smoother distribution of bone influence weights over the mesh surface while minimizing the error of predicted vertex positions.

Experimental results show that our approach allows us to reproduce stretching and squashing deformations faithfully and alleviate noisy deformation predictions through BVWS, especially for deformations falling outside the subspace spanned by the training examples. The performance of our implementation on GPUs can achieve a few hundred frames per second with hundreds of bones, which is similar to the previous approach with only rigid transformations.

2 Background

Linear blend skinning [10], also called *Skeleton Subspace Deformation* (SSD), is the popular technique in computer graphics and animations, where each vertex is bound to skeletons, or rigid bones, and its position can be computed as a weighted sum of associated skeleton transformations. SSD is easy to implement, but it suffers from artifacts such as well-known collapsing joints and "candy wrapper" effect caused by linearly blending of rotation matrices [9, 11]. We can also generate the new pose in their pose space [9] where a specific pose can be synthesized as a function of the given example meshes.

For a skeleton-free mesh, we can build a skinned mesh from mesh sequences, where clustering triangles with similar rotation sequences can be identified as a near-rigid structure of the mesh [8], called proxy bone. The proxy bone transformations and bone-vertex influence weights can then be estimated automatically from example mesh spaces by minimizing the position error between the deformed vertices and the ground truth vertices at each frame [8, 16].

The skeletal animation can be driven by low-dimensional control signals [2, 7] and this idea can be extended to example-based mesh deformations in the framework of inverse kinematic [4, 15], where a small subset of vertices act as constraints for optimization, finding the best pose whose vertex positions or bone transformations satisfy those constraints. However, real-time solutions are difficult, as this

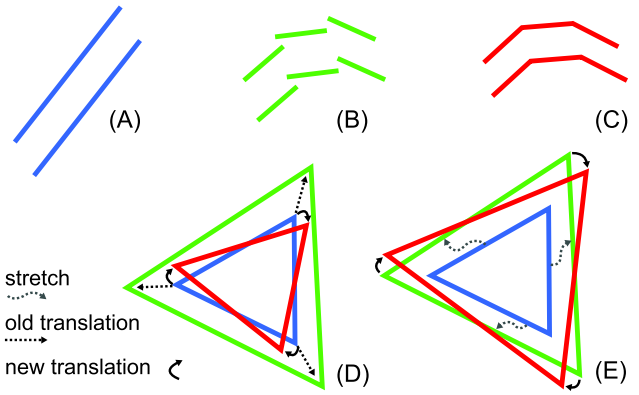


Fig. 2 *Top row*: the prediction errors in translations can be amplified and result in the great visual artifacts such as discontinuities. This can be recovered with a new set of translations through the Poisson based solver. *Bottom row*: the new translations from Poisson solver may nullify the stretch which was encoded into the previous translations but if we handle stretch separately from translation then we can maintain area changes due to stretch

process requires quite an expensive nonlinear optimization and the performance drops quickly when the number of example poses grows. To achieve the real-time performance, the off-line learning system would be more efficient. The work of [16] is based on SSD where the deformation gradient is trained from mesh examples and can be predicted according to the configuration of skeletons. In addition to rotation regression, the shear/scale can be trained in terms of axis-angle, and thus affine deformations can be reproduced well. However, the method still relies on skeletons and can only be applied on articulated deformations.

On the other hand, the work of [6] can work on the arbitrary deformable surface without skeletons. In their approach, CCA is used for building the connection between a set of control points and each rigid bone transformation. They also introduce the nonlinear version of CCA, called *kernel CCA*, to better capture the nonlinear relationships between underlying pairs of input data. Because all these trainings can be preprocessed in the off-line, they can predict bone transformations in real-time with a few matrix–vector multiplications on GPUs. Once bone transformations are predicted, each vertex position is computed with the standard way of linear blending skinning in the modern graphics hardware. Thus, the performance is fast with a few hundred frames per second.

However, the deformation prediction based techniques may suffer from the errors in translations and they usually produce the visual artifacts due to the mesh discontinuities as shown in top row of Fig. 2. New translations can be recovered through the Poisson equation in the gradient domain such as deformation gradient [1, 14, 16] or edge gradient [6] to alleviate the artifacts. Unfortunately, there are problems with the Poisson based translation solver for rigid transformations. Rigid bones have only rotations and translations and, therefore, stretch need to be encoded into either translation or rotation. Scale can be encoded into translation and shear can be simulated by the combination of translation and rotation [13].

Figure 2(D) shows the example of the side effects of the Poisson based translation solver with a rigid transformation. The original (blue) triangle is deformed to the green one where stretch is mostly encoded into translation. Unfortunately, the new translation from the Poisson equation will nullify the previously encoded scale because it just plays on the role of recovering the errors in the predicted translation. This causes inaccurate deformation results different from training examples. On the other hand, as shown in Fig. 2(E), if we encode stretch into a nonrigid transformation separately from translation then we can be free from the side effects of the Poisson based translation solver. Thus, we propose a new learning framework to handle the nonrigid transformations explicitly.

We also propose a new method to improve the predicted surface deformation by smoothing bone-vertex weights across the vertices. The motivation of the method originated from the work of mesh smoothing [5]. Our main goal is maintaining a high quality skin approximation over several mesh examples while smoothing bone-vertex weights across the neighboring vertices. Therefore, we regard the weight smoothing as an additional criteria in the optimization and integrate it into the objective function when computing vertex skin weights.

3 Deformation with affine bones

We propose a learning framework similar to the one in [6], but we explicitly handle the stretch deformations by using the affine bones instead of rigid bones in the previous work. In our approach, each nonrigid bone transformation is decomposed a rigid (rotation) and a nonrigid (stretch) component. At run-time, a configuration of control points will drive rigid bones and then the rigid bones induce nonrigid stretch to add affine deformation. Finally, we compute new translations through the Poisson solver to generate new surface deformations.

In this section, we begin with a skinned mesh with affine bones and perform a two-layered regression from control points to rotation, followed by from rotation to stretch. Finally, we describe how affine bone transformations involve Poisson equation.

3.1 Skinned mesh with affine bones

From the training examples with P different poses, we can construct the skinned mesh with B proxy bones. We perform the hierarchical clustering [16] in which triangles with

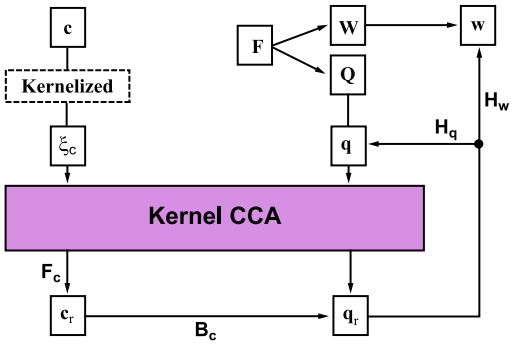


Fig. 3 The overview of our learning framework where $\mathbf{B}_c$ and $\mathbf{H}_w$ is the coefficient matrices for the two-layered regression, respectively

the similar transformation are progressively grouped into the same bone. After initial clustering, we can identify which vertices are belonging to which bones and their transformations $\mathbf{T}_b^j$ where $j = 1 \dots P$ and $b = 1 \dots B$. Then we should estimate bone-vertex influence weight w_{bi} which indicates how much b bone transformations influence on vertex i. We can compute vertex weights by minimizing the fitting error between the deformed vertex position and the ground truth vertex position v_i^j at pose j. Specifically,

$$\min_{\mathbf{w}_i, \mathbf{T}} \sum_{j=1}^{P} \left\| v_i^j - \sum_{b=1}^{B} w_{bi} \left(\mathbf{T}_b^j v_i \right) \right\|^2, \quad i = 1 \dots N, \tag{1}$$

where v_i is the vertex position in the rest pose with N vertices and the objective function is solved with the non-negative least square (NNLS) method to avoid the overfitting [8]. Similarly, the affine bone transformations $\mathbf{T} = (\mathbf{F}, \mathbf{t})$ can be solved using (1) as well by fixing the vertex weights. Here, $\mathbf{F}$ is a nonrigid component, a 3×3 matrix being formed by composing matrices of rotation and stretch, and $\mathbf{t}$ is a translation.

3.2 CCA based regression for rigid bone transformations

We can build the connection between a set of control points $\mathbf{c}$ and a bone deformation $\mathbf{d}$ in their reduced subspaces [6]. The difficulties are that we may not be able to apply CCA based regression directly from control points coordinate to a nonrigid affine bone transformation because 12 entries of affine matrix is less meaningful. For a more meaningful way, the linear matrix $\mathbf{F}$ should be decomposed into rotation matrix $\mathbf{Q}$ and stretch matrix $\mathbf{W}$ through Polar Decomposition [13]. For better interpolation and efficient computation, we can transform rotation $\mathbf{Q}$ to quaternion $\mathbf{q} \in \mathbb{R}^4$ and represent a symmetric stretch matrix $\mathbf{W}$ with only right upper 6 entries $\mathbf{w} \in \mathbb{R}^6$.

In this way, we can extract rigid and nonrigid components from bone transformations and handle them separately; rotations for posing bones and stretches for adding flexible surface. For rotations, we perform nonlinear model reduction using kernel CCA for every data pair of $(\mathbf{c}, \mathbf{q})$ which transforms into reduced coordinates $(\mathbf{c}_r, \mathbf{q}_r)$ with the coefficient matrix $\mathbf{F_c}$ from $\mathbf{c}$ to $\mathbf{c}_r$. We provide Appendix I for the mathematical description for kernel CCA. After performing nonlinear model reduction, we can find the linear mapping from $\mathbf{c}_r$ to $\mathbf{q}_r$ with regression.

$$\min_{\mathbf{B}_c} \sum_{j}^{P} \left\| \mathbf{B}_c \mathbf{c}_r^j - \mathbf{q}_r^j \right\|^2,$$

where $\mathbf{c}_r^j$ and $\mathbf{q}_r^j$ is the values in the reduced coordinates at the training example j.

In practice, the bases generated by CCA are not necessarily orthogonal to each other. Thus, we should compute the optimal reconstructor from the reduced coordinate $\mathbf{q}_r$ to $\mathbf{q}$ in the original space. Specifically,

$$\min_{\mathbf{H}_q} \sum_{j}^{P} \left\| \mathbf{H}_q \mathbf{q}_r^j - \mathbf{q}^j \right\|^2,$$

where $\mathbf{q}^j$ is quaternion at the training example j.

3.3 Regression for nonrigid bone transformations

To handle affine bone transformations, we should train a mapping to stretch matrix according to the control point coordinates. In our approach, we assume that each bone stretch has a close relationship with its corresponding bone rotation in addition to control point coordinates. Thus, we choose to compute the linear mapping from $\mathbf{q}_r$ to $\mathbf{w}$ for each bone, instead of a mapping directly from control points $\mathbf{c}$ to $\mathbf{w}$. It is obvious that this regression can provide not only the direct mapping from a bone rotation in the reduced coordinate to a bone stretch but also the indirect mapping from control points to a bone stretch because we have already establish the maximized correlation between control points and each bone rotation in their reduced coordinates. Specifically,

$$\min_{\mathbf{H}_w} \sum_{j}^{P} \left\| \mathbf{H}_w \mathbf{q}_r^j - \mathbf{w}^j \right\|^2,$$

where $\mathbf{w}^j$ is the right symmetric entries of stretch matrix at the training example j.

Note that we can stack the matrix $\mathbf{H}_q$ and $\mathbf{H}_w$ into a single matrix $\mathbf{H}_d$ because they both have $\mathbf{q}_r$ as input for regression. Then we represent the deformation predictor $\mathcal{D}(\mathbf{c})$ to generate $\mathbf{d} \in \mathbb{R}^{10}$ which consists of $\mathbf{q} \in \mathbb{R}^4$ and $\mathbf{w} \in \mathbb{R}^6$, by concatenating three matrix sequences, $\mathbf{M} = \mathbf{H}_d \mathbf{B}_c \mathbf{F}_c$. Finally, the reconstructed $\mathbf{q}$ and $\mathbf{w}$ are converted to the full 3×3 matrices, rotation $\mathbf{R}$ and stretch $\mathbf{W}$ to obtain the nonrigid affine component $\mathbf{F} = \mathbf{RW}$.

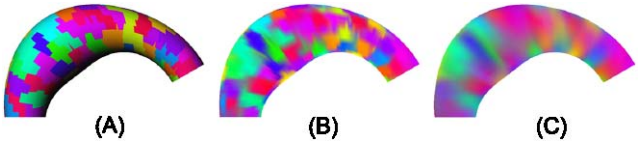


Fig. 4 Bone-vertex weight smoothing. (A) Initial clustering, (B) initial bone-vertex weight fitting, (C) BVWS with 3 iterations

Once we obtain **F**, we also need to compute bone translations to complete surface deformations. We use the same Poisson equation as in the work of [6]. However, instead of using a rotation, we use a nonrigid affine component **F** to derive the edge gradients. Then we can reformulate the Poisson equation into a linear least square system, $\mathbf{Ax} = \mathbf{b}$, where **x** is new translation vectors, **b** consists of each nonrigid bone transformation **F**, and **A** is computed from the rest-pose vertex positions and bone-vertex/bone-edge skinning weights. Finally, the linear system can be solved in the form of $\mathbf{x} = \mathbf{Pb}$ where $\mathbf{P} = (\mathbf{A}^\mathbf{T}\mathbf{A})^{-1}\mathbf{A}^\mathbf{T}$ and it can be precomputed because **A** is fixed at run-time.

4 Bone-vertex weight smoothing (BVWS)

In addition to the predicted bone transformations, another important factor that may affect the quality of the resulting deformations is the bone influence weights. Since each vertex may be influenced by several bone transformations, the blended transformations may not be smooth when nearby bones undergo different movements, and consequently cause visual artifacts near the boundaries of different bones. One effective solution for these artifacts is to ensure that the blending weights are smoothly distributed over the surface, and thus the resulting blended transformations are also smooth. Therefore, we propose a novel method to generate smooth blending weights based on Laplacian smoothing.

By definition of *Laplacian Smoothing*, the continuous form of smoothing equation (*diffuse process*) is as follows:

$$\frac{\partial X}{\partial t} = \lambda \mathcal{L}(X). \tag{2}$$

The basic idea is to move the mesh vertices along the direction of Laplacian to produce a smooth surface. By integrating (2) over time, we can distribute noises over its neighbor fast while keeping the overall shape of the mesh. One important observation is that the diffusion process reaches equilibrium state when $\mathcal{L}(X) = 0$. An intuitive interpretation is that the vector of $\mathcal{L}(X)$ can be regarded as the mean-curvature normal from differential geometry. Therefore, minimizing $\mathcal{L}(X)$ can be regarded as minimizing the total curvature on the surface, which implies the surface smoothing. Since our goal is to smooth the vertex weights instead of mesh geometry, we apply the discrete Laplace operator to vertex weight w_{bi} for vertex v_i. Specifically,

$$\mathcal{L}(w_{bi}) = \frac{1}{m} \sum_{j \in E(i,j)} (w_{bj} - w_{bi}),$$

where $E(i, j)$ is edges between vertex i and j and m is the number of one-ring neighbors to vertex i. Note that our goal is not only to smooth the vertex weights across the mesh by minimizing $\mathcal{L}(X)$, but also to minimize the fitting error between the skinned vertex positions and the ground truth vertex positions. Thus, we can rewrite the objective function of (1) as follows:

$$\min_{\mathbf{w}_i, \mathbf{T}} \sum_{j=1}^{P} \left(\left\| v_i^j - \sum_{b=1}^{B} w_{bi} (\mathbf{T}_b^j v_i) \right\|^2 + \rho \sum_{b=1}^{B} \left\| \mathcal{L}(w_{bi}) \right\|^2 \right), \quad i = 1 \ldots N, \tag{3}$$

where ρ is the control parameter for smoothness and set to 0.01 for all our experiments.

Since the original $\mathbf{T}_b^j$ would not be optimal anymore under the new vertex weights, we also perform refitting bone transformations by fixing vertex weights from (3) and repeat a few iterations for spreading the weights smoothly. In our experiments, we use 3 iterations of BVWS.

5 GPU implementation

Our implementation on GPUs is very straightforward. We use CUDA [3] for matrix-vector multiplications and OpenGL Shading Language [12] based GPGPU techniques for linear blend skinning. As stated before, we upload to the graphics card memory the precomputed matrices, **M** and **P**, for deformation predictor and Poisson based translation solver, respectively.

At every frame, the control points sequence **c** is sent to GPUs and then its kernelized vector ξ_c can be computed by dot product in the feature space using kernel function. In our simulation, we use RBF kernel function for our all experiments. For each bone deformation, we perform the matrix-vector multiplications $\mathbf{d} = \mathbf{M}\xi_c$ where **d** consists of quaternion **q** and the right stretch entries **w** and they are transformed into **R** and **W**, followed by composing the linear matrix $\mathbf{F} = \mathbf{RW}$. For a new set of translation, we perform another matrix-vector multiplication $\mathbf{t} = \mathbf{PF}$. Finally, we store the new bone transformations $\mathbf{T} = (\mathbf{F}, \mathbf{t})$ into the texture memory which can be accessed by vertex shader at linear blend skinning stage as shown in Fig. 5.

We can perform the standard linear blend skinning by binding the bone/weight information per vertex using Vertex Buffer Objects. With per vertex-linear blend skinning computation, we can obtain the deformed vertex position

Table 1 Data for simulation where #V is the number of vertices, #B for bones, #C for the reduced dimension of CCA, #Tr for training examples, #Te for the original meshes, #M and #P is the matrix size for $\mathcal{D}(c)$ and Poisson based translation solver, #T is the total size for matrices to be uploaded to the graphics card. The unit of data size is KB and the size of (A/B) means A is estimated with dual quaternion and B with our method

Examples	#V	#B	#C	#Tr	#Te	#M	#P	#T
Cylinder bulge	2000	50	2	4	40	9.49/10.66	29.69	39.19/40.36
Cylinder bulge*	2000	100	2	4	40	18.87/23.55	117.97	136.84/141.52
Dragon leg	2210	100	2	19	86	66.09/82.50	117.97	184.06/200.47
Muscle arm	5256	100	2	5	50	22.02/25.14	117.97	139.98/143.11

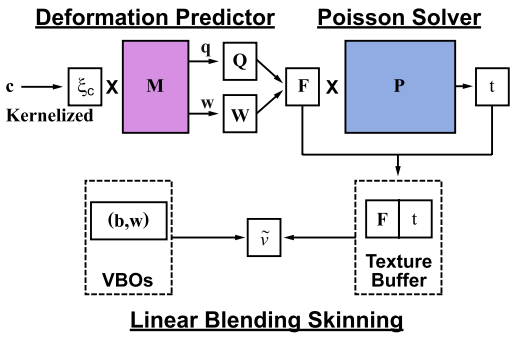


Fig. 5 Overview of our GPU implementation which consists of three stage: Deformation Predictor, Poisson based translation solver, and Linear Blending Skinning. The first two stages facilitate the computing power of CUDA and the last stage does a general GPGPU technique with the shading programming

simply by an weighted sum of bone transformations, $\tilde{v}_i = \sum_{b=1}^{B} w_{bi}(\mathbf{F_b} v_i + \mathbf{t}_b)$.

6 Experimental results

In our experiments, we use one structural model, "Cylinder Bulge," and two anatomical models, "Dragon Leg" and "Muscle Arm" as shown in Fig. 1. The structural model demonstrates the fundamental deformation for stretching and the anatomical models include the complicated muscle changes at bulging and twisting. We have sampled training examples from the original mesh sequences which may be obtained from the motion captures or animation synthesized programs. Table 1 shows the summary of our simulation data.

Figure 6 demonstrates the limitations of the previous method with rigid bone transformations and the advantage of our system with nonrigid affine bone transformations. In this experiment, we have sampled four training examples from the original mesh sequences. As shown in Fig. 6, (A) is the ground truth from the last frame of the original mesh sequences, which is also the last training example. From this pose, we generate the new deformation by dragging one of the control points with green color up to the position with red color as shown in (B), (C), and (D). Here, (B) and (C) are the results from the previous method using dual-quaternion as bone transformations while (D) is our result using affine transformations. We also apply the Poisson based translation solver to obtain new bone translation in (C) and (D). As shown in (B), we can reproduce the bulge effects well with enough number of bones even we use rigid bone transformations. However, the control point may not be able to guide the mesh deformation exactly without the constraints from translation solver. Therefore, it is hard to find the solution to reach beyond the training example spaces. We can overcome this limitation by solving the Poisson equation involving the soft constraints with control point positions. Unfortunately, we lose stretches with the new bone translations as shown in (C). (D) shows the advantage of our system. We cannot only preserve surface stretches successfully but also exactly drive the deformation guided by the control point coordinates.

Figure 7 demonstrates muscle bulging and twisting with the anatomical arm which involves not only an apparent muscle flexing at biceps but also subtle muscle changes along with arms when tightening/relaxing at elbow or wrist twisting. We have simulated the mesh deformation by moving the control point positions over the trajectories of the original mesh sequences. (A) is the ground truth, (B) and (C) are results with dual quaternion, (D) is the result with our method. Top rows are the captured images at the regions of inner side muscles when the wrist is twisting. Bottom rows are from the deformations when the arm is flexed fully. As we can expect, the prediction errors in the translations result in the visual distortions in (B). These artifacts can be alleviated with the Poisson based translation solver in (C). However, the resulting deformation in (C) cannot preserve muscle bulging; muscle volumes are lost compared to the ground truth. Our method in (D) can generate these flexible deformations and the result is almost identical with the ground truth.

Figure 8 demonstrates the skinning position errors on some selected training examples from the anatomical model of Dragon Leg. We compute vertex skinning error as the norm of offset between the ground truth positions and the

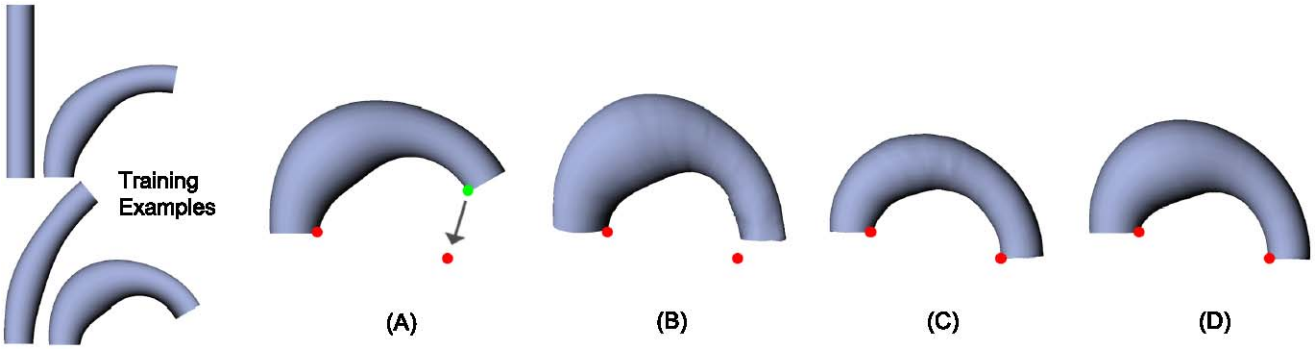


Fig. 6 Cylinder bulge deformation. (A) The skinned mesh, (B) deformation is predicted with dual quaternion, (C) deformation is predicted with dual quaternion and Poisson translation solver, (D) deformation is predicted with flexible bones

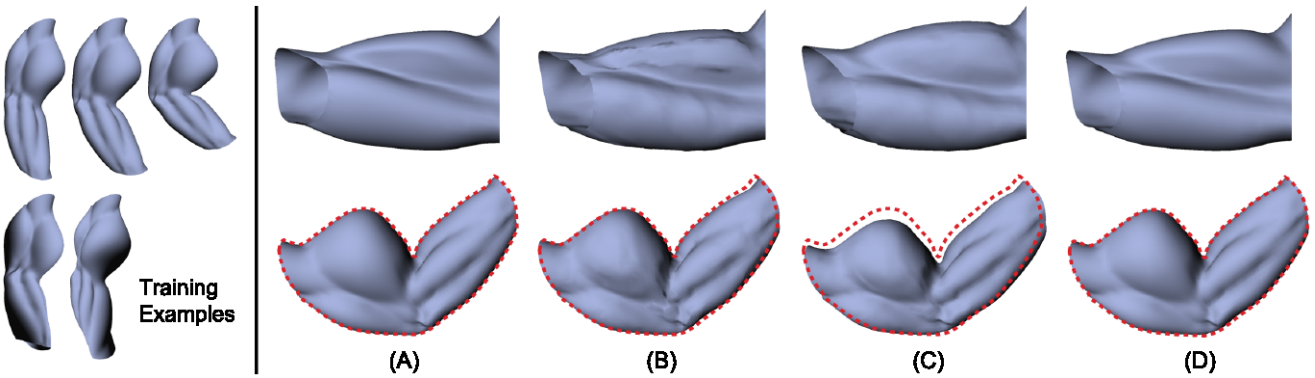


Fig. 7 Muscle arm deformation. (A) The ground truth, (B) deformation predicted with dual quaternion, (C) deformation predicted with dual quaternion and Poisson solver, (D) our method

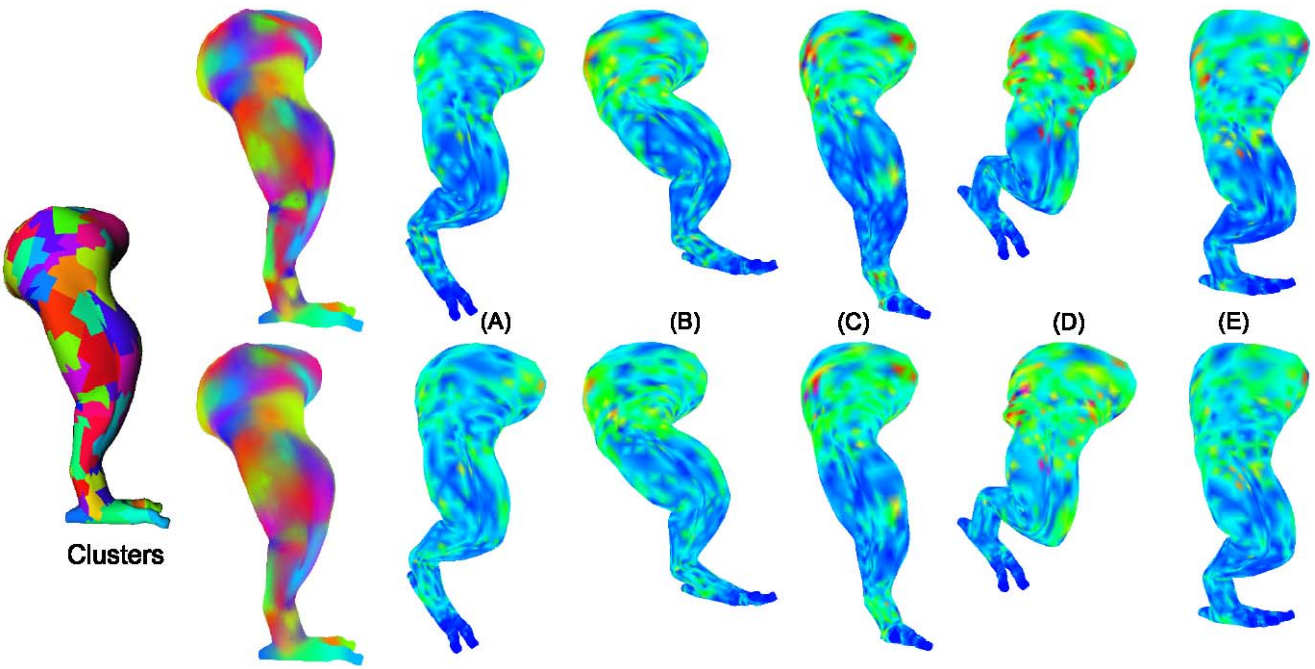


Fig. 8 Dragon leg skinning prediction. *Top rows* are skinning prediction with the usual vertex/bone transformation fitting method. *Bottom rows* with the vertex/bone transformation fitting method with vertex weight smoothing. They both are performed with 3 iterations

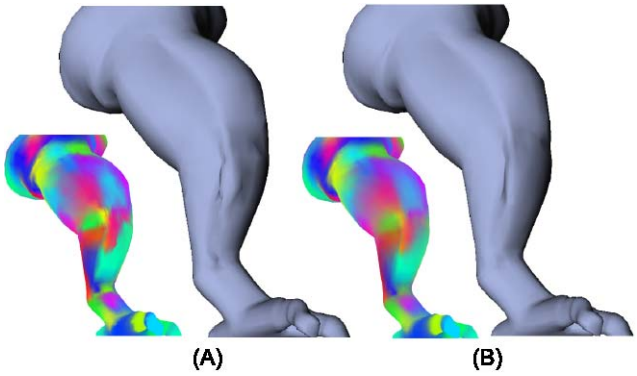


Fig. 9 Dragon leg deformation prediction. *Top* and *bottom* rows are generated from our method without and with vertex weight smoothing, respectively. We also visualize the vertex weights

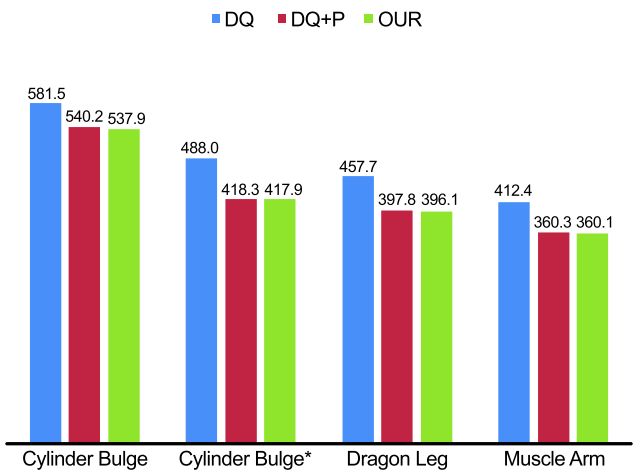


Fig. 10 Performance with FPS. All performance were taken from a 1.8 GHz Core2Duo processor with nVidia 8800GTX with 512 MB video memory

deformed positions. Top rows are from the conventional NNLS fit of vertex skinning weights and bottom rows are the results from BVWS. As shown in the top rows, the errors are not distributed evenly and there may be large distortions at some local regions. On the other hand, BVWS distributes the skinning errors over the vertices more evenly.

Figure 9 demonstrates the comparison of surface deformations with and without BVWS. We have generated the deformations with the same interactive movement of control points. There are artifacts at the regions of the calf muscles on the leg without vertex weight smoothing. On the other hand, our weight smoothing method helps eliminate such artifacts in the predicted deformations successfully.

Figure 10 demonstrates the performance comparisons for the previous method and our method. In the previous method, the Poisson based translation solver can be optional because we can predict the appropriate translation with dual quaternion for small transitional changes such as facial animation. For our method, however, it is best though to use our system with the Poisson reconstruction step. From the graph, our system shows a few hundred of FPS on the conventional personal computer with the modern graphics card and is competitive with the previous method.

7 Conclusions

In this paper, we have extended the previous technique of CCA based data driven deformation to affine bones. By using bones with affine transformations, we can faithfully generate arbitrary deformations with volume changes. The performance is similar to the previous method since the matrix size for the deformation predictor only increases slightly, which has little impact on the performance.

Our contributions also include a method for bone-vertex weight smoothing. It allows us to alleviate visual artifacts by blending bone transformations smoothly over vertices. Moreover, BVWS provides a general scheme for computing vertex weights with an objective function that can be combined with many clustering algorithms. In future, we hope to further reduce skinning errors by computing smooth weights adaptively at highly deformable regions or user selected regions.

Acknowledgements We would like to thank the reviewers for their suggestions and valuable comments. We would also like to thank Robert Y. Wang for sharing the mesh sequences "Dragon Leg" and "Muscle Arm" [16].

Appendix: Kernel canonical correlation analysis

For given P training examples, we can define the data matrices $\mathbf{C} = (\mathbf{c}^1 \dots \mathbf{c}^P)$ and $\mathbf{E} = (\mathbf{q}^1 \dots \mathbf{q}^P)$ from pairs of control points coordinate and quaternion $\{\mathbf{c}^j, \mathbf{q}^j\}_{j=1}^P$. For a nonlinear version of CCA, we can apply the mapping $\phi(\cdot)$ to transform the original data into a higher dimensional feature space (In practice, we use the nonlinear mapping of only $\mathbf{C}$ for the performance reasons). Then we can express a pair of bases for the linear version of CCA as $\{\phi(\mathbf{C})\mathbf{f_c}, \mathbf{E}\mathbf{f_e}\}$, where $\mathbf{f_c}, \mathbf{f_e} \in \mathbb{R}^P$ are coefficient vectors. Thus, we can write kernel CCA as follows:

$$\max_{\mathbf{f_c},\mathbf{f_e}} \rho = \max_{\mathbf{f_c},\mathbf{f_e}} \frac{\mathbf{f_c^T}\phi(\mathbf{C})^{\mathbf{T}}\phi(\mathbf{C})\mathbf{E}^{\mathbf{T}}\mathbf{E}\mathbf{f_e}}{\sqrt{\mathbf{f_c^T}(\phi(\mathbf{C})^{\mathbf{T}}\phi(\mathbf{C}))^2\mathbf{f_c}\mathbf{f_e^T}(\mathbf{E}^{\mathbf{T}}\mathbf{E})^2\mathbf{f_e}}},$$

where we can obtain κ pairs of coefficient vectors for two matrices, $\mathbf{F_c} = (\mathbf{f_c^1} \dots \mathbf{f_c^\kappa})$ and $\mathbf{F_e} = (\mathbf{f_e^1} \dots \mathbf{f_e^\kappa})$ from SVD.

At run-time, we should project the control points coordinate $\mathbf{c}$ into the reduced coordinates $\mathbf{c_r} \in \mathbb{R}^\kappa$ and it can be represented as a matrix-vector form as follows:

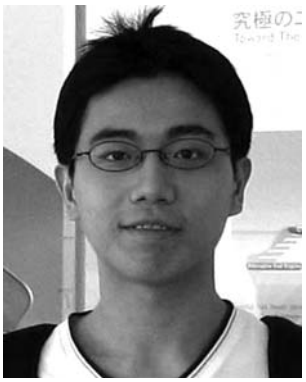
$$\mathbf{c_r} = \mathbf{F_c^T}\boldsymbol{\xi}_\mathbf{c},$$

where $\boldsymbol{\xi}_{\mathbf{c}} \in \mathbb{R}^{\kappa}$ is the kernelized vector whose jth entry is $k_c(\mathbf{c}, \mathbf{c^j})$. In our simulation, we use the Gaussian as the kernel function, $k_c(\mathbf{c}, \mathbf{c^j}) = \exp(-\frac{\mathbf{c}^T\mathbf{c}^j}{\beta})$ with $\beta = 40.0$.

References

1. Anguelov, D., Srinivasan, P., Koller, D., Thrun, S., Rodgers, J., Davis, J.: Scape: shape completion and animation of people. ACM Trans. Graph. **24**(3), 408–416 (2005). http://doi.acm.org/10.1145/1073204.1073207
2. Chai, J., Hodgins, J.K.: Performance animation from low-dimensional control signals. ACM Trans. Graph. **24**(3), 686–696 (2005). http://doi.acm.org/10.1145/1073204.1073248
3. nVidia CUDA: Compute unified device architecture (CUDA). http://developer.nvidia.com/ (2009)
4. Der, K.G., Sumner, R.W., Popović, J.: Inverse kinematics for reduced deformable models. ACM Trans. Graph. **25**(3), 1174–1179 (2006). http://doi.acm.org/10.1145/1141911.1142011
5. Desbrun, M., Meyer, M., Schröder, P., Barr, A.H.: Implicit fairing of irregular meshes using diffusion and curvature flow. In: SIGGRAPH '99: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, pp. 317–324. ACM Press/Addison-Wesley Publishing Co., New York (1999). http://doi.acm.org/10.1145/311535.311576
6. Feng, W.W., Kim, B.U., Yu, Y.: Real-time data driven deformation using kernel canonical correlation analysis. ACM Trans. Graph. **27**(3), 1–9 (2008). http://doi.acm.org/10.1145/1360612.1360690
7. Grochow, K., Martin, S.L., Hertzmann, A., Popović, Z.: Style-based inverse kinematics. ACM Trans. Graph. **23**(3), 522–531 (2004). http://doi.acm.org/10.1145/1015706.1015755
8. James, D.L., Twigg, C.D.: Skinning mesh animations. ACM Trans. Graph. **24**(3), 399–407 (2005). http://doi.acm.org/10.1145/1073204.1073206
9. Lewis, J.P., Cordner, M., Fong, N.: Pose space deformation: A unified approach to shape interpolation and skeleton-driven deformation. In: SIGGRAPH '00: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, pp. 165–172. ACM Press/Addison-Wesley Publishing Co., New York (2000). http://doi.acm.org/10.1145/344779.344862
10. Magnenat-Thalmann, N., Laperrire, R., Thalmann, D., Montréal, U.D.: Joint-dependent local deformations for hand animation and object grasping. In: Proceedings on Graphics Interface '88, pp. 26–33 (1988)
11. Mohr, A., Gleicher, M.: Building efficient, accurate character skins from examples. ACM Trans. Graph. **22**(3), 562–568 (2003). http://doi.acm.org/10.1145/882262.882308
12. Rost, R.J.: OpenGL(R) Shading Language, 2nd edn. Addison-Wesley, New York (2006)
13. Shoemake, K., Duff, T.: Matrix animation and polar decomposition. In: Proceedings of the Conference on Graphics Interface '92, pp. 258–264. Morgan Kaufmann Publishers Inc., San Francisco (1992)
14. Sumner, R.W., Popović, J.: Deformation transfer for triangle meshes. ACM Trans. Graph. **23**(3), 399–405 (2004). http://doi.acm.org/10.1145/1015706.1015736
15. Sumner, R.W., Zwicker, M., Gotsman, C., Popović, J.: Mesh-based inverse kinematics. ACM Trans. Graph. **24**(3), 488–495 (2005). http://doi.acm.org/10.1145/1073204.1073218
16. Wang, R.Y., Pulli, K., Popović, J.: Real-time enveloping with rotational regression. ACM Trans. Graph. **26**(3), 73 (2007). http://doi.acm.org/10.1145/1276377.1276468

Byung-Uck Kim received the B.S., M.S., and Ph.D. degrees in computer science from Yonsei University, Korea, in 1996, 1998, and 2004, respectively. He had worked with Samsung Electronics Co. Ltd. as a senior engineer until 2007. He is currently a Postdoctoral Researcher in the Department of Computer Science, University of Illinois at Urbana-Champaign. His research interests include computer graphics, geometry processing, and data-driven mesh deformation.



Wei-Wen Feng is currently a Ph.D. candidate in the Department of Computer Science, University of Illinois at Urbana-Champaign, where he received his M.S. degree in 2006. His Ph.D. research is on data-driven methods for mesh deformation, mesh skinning, and real-time rendering. He received his B.S. degree in Computer Science at National Chiao Tung University in Taiwan.

Yizhou Yu received the B.S. degree in computer science and the M.S. degree in applied mathematics from Zhejiang University, China, in 1992 and 1994, respectively, and the Ph.D. degree in computer science from the University of California at Berkeley in 2000. He is currently an associate professor in the Department of Computer Science, University of Illinois at Urbana-Champaign. He is on the editorial board of the Visual Computer Journal and International Journal of Software and Informatics. His current research interests include computer animation, geometry processing, texture analysis, and visual analytics.