Real-Time Enveloping with Rotational Regression

Robert Y. Wang[1] Kari Pulli[1,2] Jovan Popović[1]

[1]Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology

[2]Nokia Research Center

Abstract

Enveloping, or the mapping of skeletal controls to the deformations of a surface, is key to driving realistic animated characters. Despite its widespread use, enveloping still relies on slow or inaccurate deformation methods. We propose a method that is both fast, accurate and example-based. Our technique introduces a rotational regression model that captures common skinning deformations such as muscle bulging, twisting, and challenging areas such as the shoulders. Our improved treatment of rotational quantities is made practical by model reduction that ensures real-time solution of least-squares problems, independent of the mesh size. Our method is significantly more accurate than linear blend skinning and almost as fast, suggesting its use as a replacement for linear blend skinning when examples are available.

CR Categories: I.3.7 [Computer Graphics]: Three Dimensional Graphics and Realism—Animation; I.6.8 [Simulation and Modeling]: Types of Simulation—Animation

Keywords: Enveloping, skinning, deformation, model reduction, animation

1 Introduction

Enveloping (or skinning) is a common and fundamental task in computer graphics. Whenever an animator controls a character via a skeleton, enveloping is used to map these controls to deform a mesh surface. There is no standard way to envelope. An artist may run a physical simulation to model muscle deformations or tune complex systems of shape blending for more direct control. For interactive applications, *linear blend skinning* enjoys the most popularity. It is easy to use and can be accelerated on graphics hardware. However, linear blend skinning suffers from artifacts such as joint collapse and restricts the range of deformations available to the artist.

The shortcomings of linear blend skinning are well-known, and there has been a rich body of work on alternatives. Much of this work proposes new modeling approaches. However, given the intricacy of existing modeling tools, an artist may not want to or be able to switch. For this case, there are techniques that train on examples exported from other enveloping tools. Most of these example-based approaches learn a model of corrective displacements on top of linear blend skinning. However, displacement models extend poorly to correcting the rotational errors of linear blend skinning.

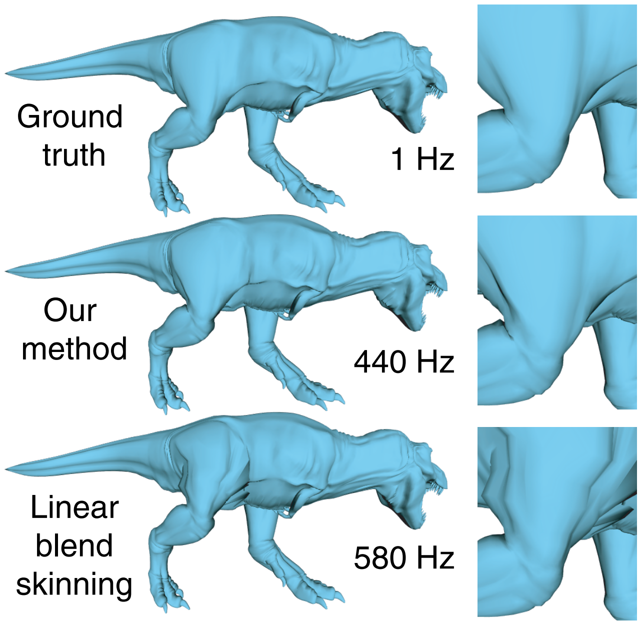


Figure 1: *Anatomy-based techniques produce good results but are too slow for many real-time applications. Our technique accurately captures muscle deformations from a set of examples generated by the black-box anatomical model and efficiently evaluates them on graphics hardware. Our technique is more accurate than linear blend skinning and almost as fast.* (Credit: Joel Anderson)

The methods that directly model rotational deformations rely on strong assumptions about where and how these deformations occur. Finally, most techniques are simply not fast enough for use in applications such as video games and training simulations.

We believe that an artist should be able to use any enveloping tool, or even run a muscle simulation, without worrying about the computational efficiency of the resulting model. Thus, our technique learns from examples exported from any black-box enveloping tool and generates a model suitable for fast evaluation on graphics hardware. We also anticipate that a common set of skinning behaviors would include deformations such as twisting, muscle bulges, and shoulder deformations. We designed our rotational regression model with these behaviors in mind. All in all, we are proposing our technique as a more general, less artifact-ridden, replacement for linear blend skinning.

We formulate the enveloping problem as learning a mapping $\mathbf{y}(\mathbf{q})$ from a set of example skeleton-mesh pairs $\{(\mathbf{q}^i, \mathbf{y}^i)\}$. We choose to learn our mapping in the space of deformation gradients. Deformation gradients encode the local orientation, scaling, and shearing of a deformed mesh with respect to the mesh in the *rest pose*. We train and evaluate deformation gradient predictors $D(\mathbf{q})$ that can relate our input, bone transformations, to deformation gradients of our output mesh surface (Section 3). From these predictions, we can reconstruct the mesh vertex positions by solving a Poisson

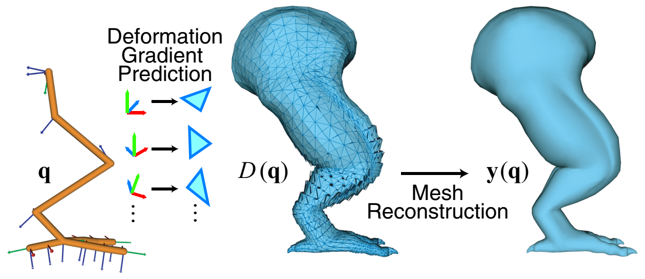


Figure 2: *Overview. Our model maps skeletal pose* $\mathbf{q}$ *to its corresponding mesh* $\mathbf{y}(\mathbf{q})$. *The mapping has two steps. First, we predict deformation gradients of the mesh based on the skeletal pose. We then reconstruct vertex positions from these predictions by solving a Poisson equation.*

equation (Section 4). We begin by describing a prediction step that evaluates the deformation gradient of every triangle, and a reconstruction step that solves for the position of every vertex. However, we can exploit coherency and coordination of the vertices and triangles (Section 5) to reduce the number of deformation gradients we have to predict and the number of coordinates we have to solve for at runtime. Our reduction not only makes CPU evaluation much faster, but also takes the particular form of matrix-palette skinning, allowing us to perform the bulk of our runtime evaluation on the GPU (Section 6).

2 Related Work

Physically based and anatomy-based approaches have produced some of the most realistic results in enveloping [Scheepers et al. 1997; Wilhelms and Gelder 1997; Teran et al. 2005]. Commercial packages for muscle simulations are readily available and commonly used in movie production and special effects [cgCharacter 2003; Comet 2006]. Other approaches use physics but not anatomy to describe muscle deformations [Hyun et al. 2005; Capell et al. 2002; Capell et al. 2005; Guo and Cheong Wong 2005]. We complement these approaches by transforming the models they produce into a form suitable for evaluation on graphics hardware. In fact, our main application is to take custom rigged models that are costly to evaluate and learn an efficient form for use in a training simulation or video game.

Linear blend skinning is the most pervasive enveloping technique used in interactive applications. This unpublished technique is also known as single-weight enveloping and skeletal subspace deformation, whose acronym SSD we use for the remainder of this paper. The benefit of SSD lies in its ease of use and applicability to modern graphics hardware in the form of *matrix-palette skinning* [Lee 2006]. SSD transforms each vertex with a linear blend of bone rotations. The blend weights are usually hand-painted, but there are also well-known techniques for optimally training them from a set of examples [Mohr and Gleicher 2003; James and Twigg 2005].

Linearly blending rotations leads to well-known artifacts such as collapsing joints and the "candy wrapper effect" [Lewis et al. 2000; Mohr and Gleicher 2003]. There have been many techniques that address these problems. Wang and Phillips [2002] and Merry et al. [2006] propose variations to SSD that trains more blend weights per vertex per joint. While these models are more expressive, they also require more training data to prevent overfitting. The remaining techniques fall into two broad categories: displacement interpolating approaches and rotation interpolating approaches.

Displacement interpolating approaches take as input a baseline SSD model and a set of training poses composed of skeleton-mesh pairs [Lewis et al. 2000; Sloan et al. 2001; Kry et al. 2002; Allen et al. 2006; Rhee et al. 2006]. They compute corrective displacements to the SSD model based on the training data and interpolate these displacements in pose space. Adding displacements works well to correct minor errors of SSD. However, we show that interpolating displacements to correct SSD rotational errors, such as those found in twisting motion, becomes unwieldy, requiring abundant training data. Our technique complements the displacement approaches above because it provides a superior baseline technique that better approximates rotational deformations.

Rotation interpolating approaches such as the work of Weber [2000], and Mohr and Gleicher [2003] extend the expressive power of SSD by introducing additional spherical linearly interpolated "half-way" joints. However, these joints are added *a priori*, without regard to the actual deformations given by example poses. We extend the idea of introducing additional joints by identifying precisely where they are needed (Section 5) and by fitting the behavior of these joints to surface deformations. We show that this improves upon existing techniques, especially for the case of joints with both bending *and* twisting. Kavan and Žára [2005] take an existing SSD model and non-linearly blend quaternions instead of transformation matrices. This technique corrects twisting defects but cannot approximate muscle deformations that were never present in the original SSD model.

Deformation gradients have been used by a variety of techniques for pose modeling [Sumner and Popović 2004; Anguelov et al. 2005; Der et al. 2006]. We share with these techniques a common Poisson reconstruction step, but differ in how we model the deformation gradients. Der et al. [2006] describe a pose space with the non-linear span of a set of deformation gradients extracted from example meshes. This pose space can then be explored by mesh-based inverse-kinematics. While this is useful for certain applications, animators often want to control a mesh with a specific skeleton or precisely drive non-end-effector joints. Furthermore, Der et al. [2006] model the pose space non-parametrically, incurring an evaluation cost cubic in the number of training poses. The SCAPE pose model predicts deformation gradients from bone rotations much like our own model [Anguelov et al. 2005]. SCAPE can even handle different identities in addition to different poses. On the other hand, our method more accurately predicts rotational deformations and is orders of magnitude faster to evaluate.

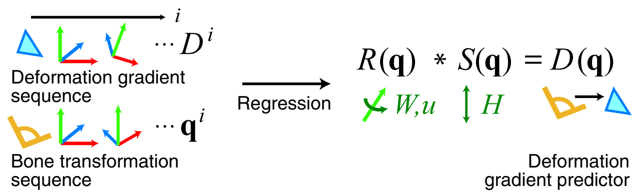
3 Deformation Gradient Prediction

The deformation gradient D is a local description of orientation, scaling, and shearing of a deformed mesh surface $\mathbf{y}$ relative to a rest (or undeformed) surface $\hat{\mathbf{y}}$. For a triangle mesh, the deformation gradient for each triangle is simply a 3×3 matrix [Sumner and Popović 2004; James and Twigg 2005].

Our task is to predict these deformation gradients D from bone transformations $\mathbf{q}$ given a set of examples. For an articulated rigid-body, this mapping is the identity; the deformation gradients of each rigid segment are equal to the bone rotation affecting that segment. Linear blend skinning generalizes this mapping, allowing mesh deformations to depend on a linear blend of bone transformations. We've designed our deformation gradient predictors $D(\mathbf{q})$ to capture additional deformations such as twisting and muscle bulges.

Each deformation gradient D can be separated into a rotational component R and a scale/shear component S using polar decomposition, and these components need to be treated differently. We predict the former with a rotational regression model and the latter with a scale/shear regression model. Together, these two predictions form our deformation gradient predictors $D(\mathbf{q})$ (Figure 3).

To perform the regression above, we require sequences of bone rotations $\mathbf{q}^i$ and deformation gradients D^i. We extract the former from the example skeletal poses and the latter from the corresponding example meshes. While we describe the construction of deformation gradient predictors on a per-triangle basis below, our model can be applied to any sequence of deformation gradients—a property we exploit for our reduced reconstruction step.



i
⋯ D^i
Deformation gradient sequence
⋯ q^i
Bone transformation sequence
Regression
R(q) * S(q) = D(q)
W,u
H
Deformation gradient predictor

Figure 3: *We learn a mapping from a sequence of bone transformations to a sequence of deformation gradients. We build separate regression models for rotation and scale/shear, learning parameters W and u for rotation and H for scale/shear. The rotational model $R(\mathbf{q})$ and scale/shear model $S(\mathbf{q})$ combine to form our deformation gradient predictor.*

3.1 Notation

We denote each skeletal pose $\mathbf{q}$ to be a vector of J concatenated bone transformations $[\mathbf{vec}(Q_1)^T, \mathbf{d}_1^T, \ldots, \mathbf{vec}(Q_J)^T, \mathbf{d}_J^T]^T \in \mathbb{R}^{12J\times 1}$. Bone transformations are defined relative to the rest pose but are not relative to the parent frame. Each mesh $\mathbf{y} \in \mathbb{R}^{3V\times 1}$ is a vector of V concatenated vertex positions. In the next section, we find it convenient to work in axis-angle representations. We use $\boldsymbol{\theta}$ and $\boldsymbol{\rho}$ to denote the axis-angle forms of bone rotations Q and mesh rotations R respectively. We represent axis-angle quantities as 3-vectors with angle encoded in the magnitude and axis encoded in the direction.

3.2 Rotational Regression

The basic assumption of SSD is that vertices on the mesh transform with some subset of the bones affecting them. However, when a muscle bulges, some parts of the mesh do not rotate by the same amount as the joint causing the bulge. Other parts may rotate in the opposite direction or along a different axis (Figure 4). We propose a more general model relating a joint rotation sequence to a triangle rotation sequence.

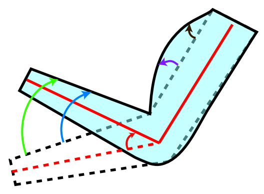


Figure 4: *Arm flexing. While most of the forearm rotates in the same direction and amount as the bone, parts of the bicep rotate about different axes and by different amounts.*

Below, we assume that we know which joint affects the triangle. In practice, we train a model for each joint in the skeleton and select the one that best approximates the triangle rotation sequence.

3.2.1 Model

To relate bone rotations to triangle rotations, we first need to express both in the same coordinate frame. Let $\tilde{\boldsymbol{\theta}}$ and $\tilde{\boldsymbol{\rho}}$ denote our bone rotations and triangle rotations expressed in the joint frame. Intuitively, $\tilde{\boldsymbol{\theta}}$ is the joint rotation. We relate the angle of the triangle rotation to the joint angle by a scale factor u and the axis of the triangle rotation to the joint axis by a rotation offset W. By using the axis-angle representation, this relationship (Figure 5) takes on a linear form:

$$\tilde{\boldsymbol{\rho}}(\mathbf{q}) = uW\tilde{\boldsymbol{\theta}}_b(\mathbf{q}),$$

where $\boldsymbol{\theta}_b(\mathbf{q})$ extracts the rotation of bone b from skeletal pose $\mathbf{q}$.

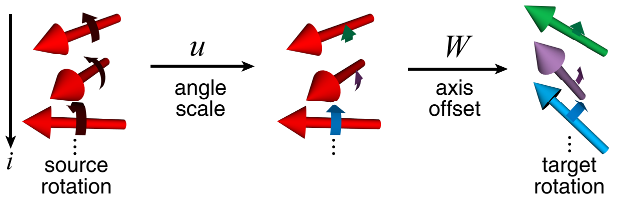


Figure 5: *Our rotation predictors learn a constant rotational axis offset W and a constant angle scale factor u to relate a joint rotation sequence (source) to a triangle rotation sequence (target).*

For each triangle, we are fitting only four parameters (three for the rotation offset W and one for the scale factor u). This simple model is surprisingly powerful and general. The model handles twisting with the rotation scale factor u, while the rotation offset to the axis effectively models muscle bulges (Figure 6).

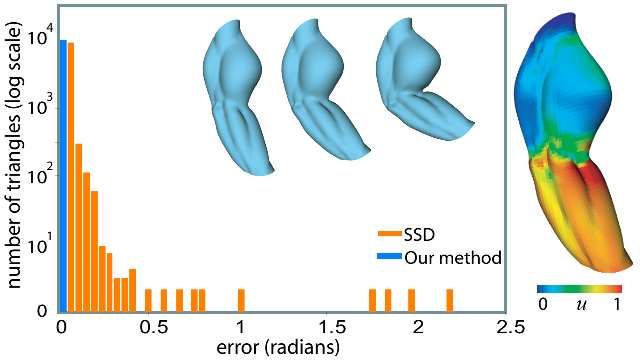


Figure 6: *Muscle arm error histogram and scale plot. We plot a histogram of the errors for our rotational regression heuristic compared to SSD for three poses of a muscle flexing. We also show the angle scale factor in false color. Note that triangles on the bicep are related to the joint rotation by a small (but non-zero) scale factor while triangles on the forearm move nearly rigidly.*

3.2.2 Training

For training, we are given rotation sequences for a bone $\mathbf{q}^i$ and a triangle $\boldsymbol{\rho}^i$. First, we transform both sets of rotations into the joint frame, forming $\tilde{\boldsymbol{\theta}}_b(\mathbf{q}^i)$ and $\tilde{\boldsymbol{\rho}}^i$. The optimal rotation offset and scale parameters, $W \in \mathbb{SO}(3)$ and $u \in \mathbb{R}$, are given by

$$\operatorname*{argmin}_{W,u} \sum_{i\in 1\ldots N} \|uW\tilde{\boldsymbol{\theta}}_b(\mathbf{q}^i) - \tilde{\boldsymbol{\rho}}^i\|^2,$$

which can be solved with coordinate descent, alternating between solving for u and W separately. Given rotation offset W, the optimal scale u has a closed-form solution. Given scale factor u, solving for W becomes an instance of the Procrustes problem, also yielding a closed form solution [Eggert et al. 1997].

We initialize the scale factor u independently of W:

$$\operatorname*{argmin}_{u} \sum_{i \in 1 \ldots N} \left(u\|\tilde{\boldsymbol{\theta}}_b(\mathbf{q}^i)\| - \|\tilde{\boldsymbol{\rho}}^i\|\right)^2.$$

If the rotational deformation is well represented by this scale-offset model, it can be shown that starting with this initial guess yields the optimal parameters in the first iteration.

Our training technique is fast enough that we can afford to fit a model for each joint and select the best one that fits our triangle rotation sequence. We do not require any prior information relating a triangle and a joint. We rely on this property in section 5.2 to relate an arbitrary surface rotation sequence to a joint rotation sequence.

Furthermore, we can fit the residual rotations to additional bones for areas with multiple joint dependencies. In practice, we found that two bones were sufficient for all of our examples.

3.3 Scale/Shear Regression

We predict each component of the scale/shear matrix linearly with respect to the axis-angle representation of two joints [Anguelov et al. 2005],

$$\mathbf{vec}(S(\mathbf{q})) = H\tilde{\boldsymbol{\theta}}_{b_1,b_2}(\mathbf{q})$$

The two joint rotations that we use are the joint associated with the best rotational predictor found by the rotational regression step $\boldsymbol{\theta}_{b_1}$ (Section 3.2), and its parent $\boldsymbol{\theta}_{b_2}$. We concatenate these two rotations and a bias term to form $\tilde{\boldsymbol{\theta}}_{b_1,b_2}(\mathbf{q}) = [\tilde{\boldsymbol{\theta}}_{b_1}(\mathbf{q})^T \; \tilde{\boldsymbol{\theta}}_{b_2}(\mathbf{q})^T \; 1]^T \in \mathbb{R}^{7\times 1}$. Given a scale/shear sequence S^i and bone rotation sequence $\mathbf{q}^i$, we can determine the parameters $H \in \mathbb{R}^{9\times 7}$ of this predictor using least-squares:

$$\operatorname*{argmin}_{H} \sum_{i \in 1 \ldots N} \|H\tilde{\boldsymbol{\theta}}_{b_1,b_2}(\mathbf{q}^i) - \mathbf{vec}(S^i)\|^2.$$

4 Mesh Reconstruction

To map deformation gradient predictions back to vertex positions, we solve a Poisson equation. First, we describe this process in its most general formulation: when we have a deformation gradient prediction at each triangle and nothing else. Next we modify the formulation to integrate the global positions of a set of near-rigid vertices. These near-rigid vertices are easy to predict with SSD, and improve accuracy by introducing global translation information into our Poisson problem. Finally, we formulate a reduced form of our mesh reconstruction optimization by exploiting coherency and coordination of triangles and vertices. This will allow us to perform the entire reconstruction step on the GPU.

4.1 Poisson Mesh Reconstruction

Poisson mesh reconstruction is the process of piecing together the deformation gradient predictions from the previous section into a single coherent mesh. Our Poisson equation formulation relates deformation gradients to vertices through edges [Anguelov et al. 2005]:

$$\operatorname*{argmin}_{\mathbf{y}} \sum_{k \in 1 \ldots T} \sum_{j=2,3} \|D_k(\mathbf{q})\hat{\mathbf{v}}_{k,j} - \mathbf{v}_{k,j}\|^2, \quad (1)$$

where $\mathbf{v}_{k,j} = \mathbf{y}_{k,j} - \mathbf{y}_{k,1}$ denotes the jth edge of the kth triangle in the pose we are solving for and $\hat{\mathbf{v}}_{k,j}$ denotes the same edge in the rest pose. Equation 1 is a least-squares problem corresponding to a linear system. We pre-factor the left-hand side of this system with the sparse Cholesky factorization [Sumner et al. 2005]. Given the per-triangle deformation gradient predictions for a new pose, we can obtain the vertex positions by back substitution.

4.2 Near-Rigid/SSD Vertex Constraints

Without information about the translational component of the skeleton, the Poisson optimization does not detect or compensate for global translational problems. Low-frequency errors can accumulate (Figure 7), and the extremities of a character may veer away from the joint configuration. We address this problem by identifying a set of near-rigid vertices. Near-rigid vertices are easy to predict, since by definition, even an articulated rigid-body predictor would suffice. In practice, we use the richer SSD model. SSD does not suffer from error accumulation problems because each vertex is dependent on the translational components of the skeleton, which contains information about the global position of each bone. Fixing a set of vertices to their SSD prediction provides our optimization with this information as well. An additional benefit of this process is that our method is exactly as fast as SSD for regions that SSD predicts well and improves the quality where it does not.

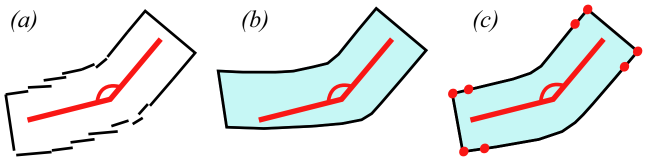


Figure 7: *"Fixing" Poisson. We use the Poisson equation to reconstruct vertex positions from edge predictions (a). However, low-frequency errors can accumulate, and the extremities of the mesh may no longer match the joint configuration (b). Fixing a set of near-rigid vertices to their SSD prediction (red dots) solves this problem (c).*

We evaluate the error of each vertex over the training set and threshold to select the set of vertices best predicted by SSD, F. We fix the vertices of this set F to their SSD predictions in our objective function. Define the linear map Ψ_a such that $\Psi_a\mathbf{q}$ is equivalent to $\sum_b^J w_{a,b}T_b(\mathbf{q})\hat{\mathbf{y}}_a$, the SSD prediction of vertex a at pose $\mathbf{q}$. We obtain our SSD weights $w_{a,b}$ by non-negative least-squares [James and Twigg 2005]. We substitute $\mathbf{y}_a = \Psi_a\mathbf{q}$ for all $\mathbf{y}_a \in F$ into Equation 1:

$$\operatorname*{argmin}_{\mathbf{y}} \sum_{k \in 1 \ldots T} \sum_{j=2,3} \|D_k(\mathbf{q})\hat{\mathbf{v}}_{k,j} - \mathbf{v}_{k,j}\|^2$$

where

$$\mathbf{v}_{k,j} = \begin{cases} \mathbf{y}_{k,j} - \mathbf{y}_{k,1} & \text{if } \mathbf{y}_{k,j} \notin F \text{ and } \mathbf{y}_{k,1} \notin F \\ \mathbf{y}_{k,j} - \Psi_{k,1}\mathbf{q} & \text{if only } \mathbf{y}_{k,1} \in F \\ \Psi_{k,j}\mathbf{q} - \mathbf{y}_{k,1} & \text{if only } \mathbf{y}_{k,j} \in F. \end{cases} \quad (2)$$

If both vertices of an edge are fixed, the error term for the edge can be dropped completely from the objective function.

We can solve Equation 2 similarly to Equation 1, by pre-factoring the left-hand side of the corresponding linear system, and evaluating new poses with back-substitution. While this formulation is sufficient for some applications, we introduce a faster formulation in the next section that is competitive with SSD in terms of speed.

4.3 Reduced Mesh Reconstruction

The optimization problem in Equation 2 solves for the coordinates of every vertex ($3V$ degrees of freedom) and requires predicting the deformation gradient of every triangle $D_k(\mathbf{q})$. In this section, we reduce this optimization to involve only the transformation matrices of a set of P *proxy-bones* ($12P$ degrees of freedom) and to require the prediction of only P deformation gradients. The size of P does

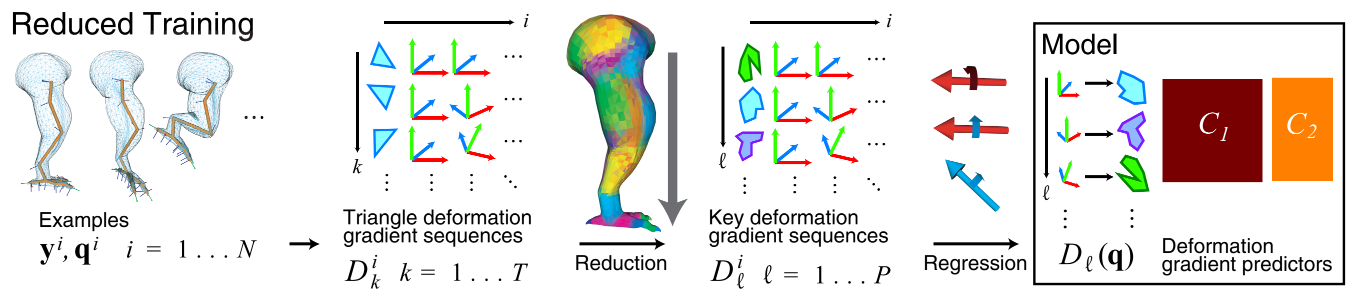

Reduced Training
Examples
$\mathbf{y}^i, \mathbf{q}^i \quad i = 1 \dots N$
Triangle deformation gradient sequences
$D_k^i \quad k = 1 \dots T$
Reduction
Key deformation gradient sequences
$D_\ell^i \quad \ell = 1 \dots P$
Regression
Model
C_1
C_2
$D_\ell(\mathbf{q})$
Deformation gradient predictors

Figure 8: *Training a Reduced Model. Given a set of example skeleton-mesh pairs, we can extract triangle deformation sequences D_k^i. Our predictor reduction step gives us sequences of key deformation gradient sequences D_ℓ^i, from which we train key deformation gradient predictor. These predictors, combined with the mesh reconstruction matrices C_1 and C_2, form our model.*

not depend on the resolution of the mesh, but rather on the complexity of the deformation. In our examples, P never exceeds 100. While we reformulate our optimization in this section, the details of selecting the key deformation gradients and proxy-bones are given in Section 5.

Our reduced mesh reconstruction scheme (Figure 8) is based on the idea that the triangles and vertices of an articulated model are very coordinated. We leverage this assumption by expressing each triangle deformation gradient predictor as a linear combination of P key deformation gradient predictors:

$$D_k(\mathbf{q}) = \sum_{\ell \in 1 \dots P} \beta_{k,\ell} D_\ell(\mathbf{q}). \tag{3}$$

We also express each vertex as the SSD-like prediction from a set of proxy-bones:

$$\mathbf{y}_a(\mathbf{t}) = \sum_{b \in 1 \dots P} \alpha_{a,b} T_b(\mathbf{t}) \hat{\mathbf{y}}_a = \Phi_a \mathbf{t} \tag{4}$$

where Φ_a is defined similarly to Ψ_a and $\mathbf{t}$ packs the proxy-bone transformations T_b similarly to $\mathbf{q}$. Our choice of an SSD-like model here is significant because the evaluation of $\mathbf{y}(\mathbf{t})$ can be performed on the GPU with matrix-palette skinning.

We substitute Equations 3 and 4 into Equation 2 and solve for the proxy-bone transformations $\mathbf{t}$:

$$\mathbf{t}(\mathbf{q}) = \operatorname*{argmin}_{\mathbf{t}} \sum_{k \in 1 \dots T} \sum_{j=2,3} \left\| \sum_{\ell \in 1 \dots P} \beta_{k,\ell} D_\ell(\mathbf{q}) \hat{\mathbf{v}}_{k,j} - \mathbf{v}_{k,j} \right\|^2$$

where

$$\mathbf{v}_{k,j} = \begin{cases} \Phi_{k,j}\mathbf{t} - \Phi_{k,1}\mathbf{t} & \text{if } \mathbf{y}_{k,j} \notin F \text{ and } \mathbf{y}_{k,1} \notin F \\ \Phi_{k,j}\mathbf{t} - \Psi_{k,1}\mathbf{q} & \text{if only } \mathbf{y}_{k,1} \in F \\ \Psi_{k,j}\mathbf{q} - \Phi_{k,1}\mathbf{t} & \text{if only } \mathbf{y}_{k,j} \in F \end{cases} \tag{5}$$

Because we chose linear models for both predictor and vertex reductions, the solution $\mathbf{t}(\mathbf{q})$ is also linear with respect to the deformation gradient predictors $D_\ell(\mathbf{q})$ and the bone rotations $\mathbf{q}$, taking the form of

$$\mathbf{t}(\mathbf{q}) = C_1 \mathbf{d}(\mathbf{q}) + C_2 \mathbf{q}, \tag{6}$$

where $\mathbf{d}(\mathbf{q}) = [\mathbf{vec}(D_1(\mathbf{q}))^T \dots \mathbf{vec}(D_P(\mathbf{q}))^T]^T$. The derivation of C_1 and C_2 are given in the supplemental materials. To obtain the vertex positions $\mathbf{y}$, we substitute $\mathbf{t}(\mathbf{q})$ for $\mathbf{t}$ in Equation 4.

Thus we have reduced our entire Poisson mesh reconstruction step into a matrix-vector multiplication (Equation 6) and matrix-palette skinning (Equation 4). We describe in Section 6 that both of these operations can be performed on the GPU. The cost of evaluating P deformation gradient predictors is negligible compared to mesh reconstruction, involving only nine 7-component dot products and a handful of quaternion multiplications per predictor.

5 Dimensionality Reduction

In the previous section, we described a reduced formulation of the mesh reconstruction step. We outlined the form of the reduction to be matrix-palette skinning for vertices and a linear blend model for deformation gradients. In this section we find the parameters required by the reduction: the SSD weights α for the vertex reduction, the blend weights β for the predictor reduction, and the key deformation gradient predictors $D_\ell(\mathbf{q})$. Given the formulation of our reduced reconstruction model, we can write objective functions for finding each of these terms. As we shall see, however, solving these optimization problems directly can be intractable, and we describe a clustering technique for finding these quantities approximately. Note that our proposed clustering is one of many that are possible. In particular, the mean-shift approach advocated in skinning mesh animations by James and Twigg [2005] could be substituted for the vertex reduction below. Mean-shift clustering is less susceptible to noise. On the other hand, our approach is faster, progressive, and clusters bones based on vertex error.

5.1 Vertex reduction

We measure the error of our vertex reduction over a set of training meshes $\mathbf{y}^i$ by the L^2 difference between the SSD-based proxy-bone prediction and the ground truth vertex position, $E(T_b^i, \alpha_{a,b}) = \sum_i^N \sum_a^V \|\mathbf{y}_a^i - \sum_b^P \alpha_{a,b} T_b^i \hat{\mathbf{y}}_a\|^2$. Ideally, we would like to find the solution with the minumum number of proxy-bones P for a given maximum error threshold ϵ. This would require us to solve

$$\min_{T_b^i, \alpha_{a,b}} P \qquad \text{subject to } E(T_b^i, \alpha_{a,b}) < \epsilon.$$

Given fixed P and fixed transformations T_b^i, we can solve for weights $\alpha_{a,b}$ using non-negative least-squares [James and Twigg 2005]. Similarly, given fixed P and fixed $\alpha_{a,b}$, we can find the least-squares solution for proxy-bone transformations T_b^i. However, we cannot solve for both simultaneously, and we do not know P beforehand. Instead, we take an approximate approach inspired by work in mesh decimation [Cohen-Steiner et al. 2004; Diebel et al. 2006].

Define the error $E_{A \to B}$ of joining proxy-bone A to proxy-bone B as $\sum_i^N \sum_{a \in G_A} \|\mathbf{y}_a^i - T_b^i \hat{\mathbf{y}}_a\|^2$. This error is an upper bound for the real approximation error of joining the vertices of group G_A

to group G_B. We add all possible joins between contiguous groups into a priority queue and iteratively perform the join with the lowest error until our error threshold is reached (Figure 9). Specifically:

1. Begin with a proxy-bone for each triangle k initialized to the transformation matrices T_k^i mapping the vertices of k from the rest pose to each pose i. Initialize the associated group G_k to contain the vertices of triangle k.
2. Greedily pick the join $A \to B$ with the smallest error $E_{A\to B}$ and add the vertices of group A to group B.
3. Solve for the weights $\alpha_{a,b}$ given the current set of transformations T_b^i
4. Solve for the transformations T_b^i given the current set of weights $\alpha_{a,b}$.
5. If $E(T_b^i, \alpha_{a,b}) < \epsilon$ go to Step 2.

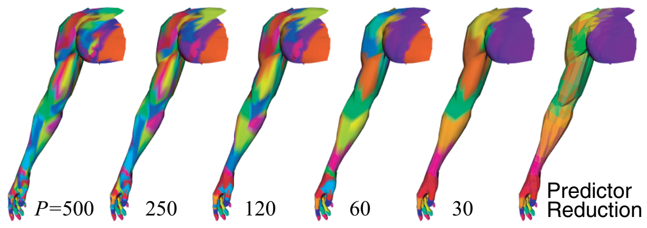


Figure 9: *Vertex clustering. Successive iterations merge coordinated vertices into fewer and fewer proxy-bones. The resulting transformations also form a good initial guess for predictor reduction.*

In practice, we need not evaluate steps 3, 4, or 5 at every iteration. Error increases smoothly, and we only need to be careful when we get close to the threshold. For efficiency reasons, we also only consider joins of contiguous proxy-bones [Cohen-Steiner et al. 2004]. We restrict each vertex to depend on only the proxy-bone transformations of the group it belongs to and the groups adjacent to that group. This reduces overfitting and also boosts performance.

5.2 Predictor Reduction

To obtain key deformation gradient predictors $D_\ell(\mathbf{q})$, we first find key deformation gradient sequences D_ℓ^i from triangle deformation gradient sequences D_k^i. We then train predictors from these sequences as in Section 3 (Figure 9). Our error metric for finding the best key sequences is the objective function from Equation 2 with the substitution

$$D_k^i = \sum_{\ell \in 1\ldots P} \beta_{k,\ell} D_\ell^i$$

where $\beta_{k,\ell}$ are the blend weights:

$$\operatorname*{argmin}_{\beta_{k,\ell}, D_\ell^i} \sum_{i\in 1\ldots N} \sum_{k\in 1\ldots T} \sum_{j=2,3} \left\| \sum_{\ell\in 1\ldots P} \beta_{k,\ell} D_\ell^i \hat{\mathbf{v}}_{k,j} - \mathbf{v}_{k,j}^i \right\|^2.$$

We can solve the optimization above using coordinate descent, alternating between solving for $\beta_{k,\ell}$ and D_ℓ^i separately. Fortunately, vertex reduction allows us to start from a particularly good initial guess for D_ℓ^i. We initialize D_ℓ^i to be the upper-left 3x3 matrix of the T_b^i matrix we found from vertex clustering. The coordinate descent converges in three iterations or less for all of our examples. Having obtained key deformation gradient sequences, D_ℓ^i, we can train deformation gradient predictors $D_\ell(\mathbf{q})$ as described in Section 3.

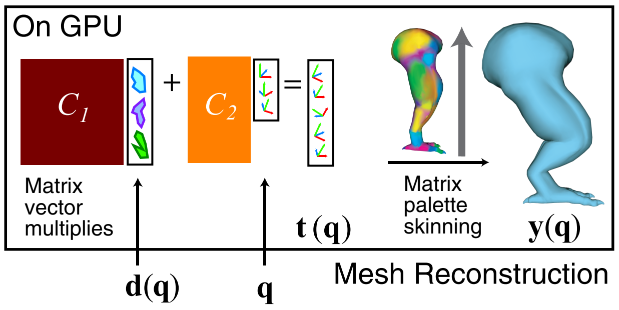


Figure 10: *Mesh reconstruction on the GPU. We load C_1, C_2 and the bone weights for matrix-palette skinning on the GPU beforehand. At runtime, we need only send the vectors $\mathbf{d}(\mathbf{q})$ and $\mathbf{q}$ per model.*

6 GPU Implementation

There are two components of our GPU implementation: a matrix-vector multiplication and matrix-palette skinning (Figure 10). Both operations are straightforward on modern graphics hardware and our implementation is one of many that are possible. We take a total of three passes to skin our character, not including the final rendering pass. The first pass performs the matrix-vector multiplication. The next pass uses matrix-palette skinning to compute the vertex positions. The third pass computes the normal vectors of the skinned character from the post-transformed geometry. The only data that we send to the GPU at runtime are the vectorized deformation gradient predictions $\mathbf{d}(\mathbf{q})$ and bone transformations $\mathbf{q}$—the remainder of the computation is performed completely on the GPU.

Matrix-vector multiplication: We precompute and upload C_1 and C_2 into video memory as a static floating-point texture. For each model, we upload textures $\mathbf{d}(\mathbf{q})$ and $\mathbf{q}$ at each frame and use a fragment program to perform the matrix-vector multiplication, one column at a time. The results are rendered on to the same $12P \times 1$ rectangle and accumulated using hardware blending. We store the final result, a vector of concatenated proxy-bone transformation matrices, as a texture.

Matrix-palette skinning: There are many ways to apply matrix-palette skinning on modern graphics hardware; see Lee [2006] for a recent survey. In light of the increase in multi-pass rendering in video games, we chose to skin only once per frame in a separate pass, storing the positions in a texture. These results can be played back for each subsequent rendering pass. This avoids redundant skinning on each rendering pass and is similar to DirectX 10 `memexport` skinning [Lee 2006] and deferred shading [Hargreaves 2004]. For each vertex, we send the proxy-bone weights and indices as texture coordinates which can be used to look up the proxy-bone transformation matrices computed in the previous pass.

Normal vectors: While traditional normal vector computation for a skinned character is usually approximated on the GPU, we perform this computation more accurately using the triangle normals of the skinned vertices. For each vertex, we precompute the indices of its 1-ring of neighbors. At runtime, these indices are passed along as texture coordinates and used to fetch the position of each neighbor computed from the skinning computation in the previous pass. We then take the appropriate cross products to compute each triangle normal, and normalize the sum of the triangle normals to obtain the vertex normal.

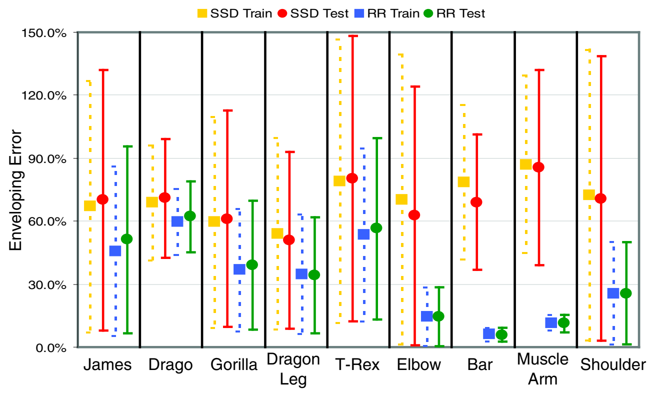


Figure 11: *Our errors are not only generally lower than SSD, but their standard deviations (error-bars) are smaller as well, meaning that the errors are usually harder to detect visually. All errors are normalized so that 100% corresponds to the best rigid-articulated body predictor for each example. Both approximation error (dotted line) and generalization error (solid line) for both SSD and our rotational regression method (RR) are shown. (Lower is better.)*

7 Results

Our technique compares favorably in quality to SSD, displacement interpolating approaches, and rotation interpolating approaches. We also compare the speed of our GPU implementation to matrix-palette skinning. Our datasets included artist-created examples from Poser, anatomically simulated examples using the cMuscle-System [2006], and 3-D human scan data [Anguelov et al. 2005].

7.1 Error Metric

We evaluate all our examples using a metric inspired by the *percent position error* (PPE) developed by Karni and Gotsman [2004] in the context of animation compression. PPE measures the total error of each predicted vertex, normalized by the best constant prediction of the animation. However, in the context of enveloping, the animation of a moving walk could be globally well preserved but locally prone to artifacts. We are only interested in these local deformations; the global motion is already given by the skeletal information at each pose. Our *enveloping error* metric normalizes the total error at each predicted vertex by the error of the best articulated rigid-body prediction of the animation:

$$EE = \sqrt{\frac{\sum_i^N \sum_a^V \|\mathbf{y}_a(\mathbf{q}^i) - \mathbf{y}_a^i\|^2}{\sum_i^N \sum_a^V \|\mathbf{r}_a(\mathbf{q}^i) - \mathbf{y}_a^i\|^2}},$$

where $\mathbf{r}_a(\mathbf{q}^i)$ is the best articulated rigid-body prediction of $\mathbf{y}_a^i$ based on the skeletal frames $\mathbf{q}^i$, computed by selecting the single best bone transformation for the vertex over all the poses.

We measure both approximation error and generalization error. Approximation error is measured by evaluating the model on the training set. We measure generalization in two ways. For the datasets where we are given an animation sequence, we sample key frames from the sequence for training and evaluate over the entire sequence. For the datasets where we are given a set of random and disjoint poses, we evaluate the leave-one-out cross validation (LOOCV) error. That is, for each example $i \in 1 \dots N$ we train on the $N-1$ poses not including i and evaluate the resulting model on pose i.

Example	Vertices	Joints	Proxy bones	Train Poses	Test Poses	Our flops	SSD flops	Our fps	SSD fps	Our Mem	SSD Mem	Train Time
James	11106	73	80	31	LOOCV	5.1M	2.6M	595	1000	5.2M	530K	6m
Drago	12500	16	80	49	LOOCV	5.0M	2.9M	618	1030	4.2M	600K	7m
Gorilla	25438	61	100	46	LOOCV	9.9M	5.9M	449	673	8.1M	1.2M	18m
Dragon Leg	2210	14	40	9	86	1.0M	0.5M	681	1144	1.1M	110K	35s
T-Rex	29380	155	60	11	121	9.4M	6.8M	443	583	5.8M	1.4M	6m
Elbow	2610	2	30	3	15	1.0M	0.6M	685	1164	880K	130K	30s
Bar	80	2	25	3	50	0.2M	0.0M	711	1228	310K	3K	5s
Muscle Arm	5256	3	30	4	40	2.0M	1.2M	692	1200	860K	250K	50s
Shoulder	2610	2	40	10	100	1.0M	0.6M	690	1172	880K	130K	30s

Table 1: *While our method is slower than SSD, we are usually within a factor of two in terms of both frame-rate and the number of floating-point operations. Our results compare most favorably for large detailed meshes, such as the T-rex, because most of the time is spent on the same matrix-palette skinning computation as SSD. Our absolute speed and memory requirements are sufficient for use in interactive applications.*

Comparison with SSD: We compared our model both in terms of quality and speed to SSD. All of our SSD models were trained using non-negative least-squares [James and Twigg 2005]. Like James and Twigg [2005], we cap the number of non-zero weights at four. In Figure 11, we show that our technique is superior in terms of quality on every example we tried. Both our total enveloping error and the variance of our errors across the vertices is lower, meaning that our errors are low-frequency and less distracting visually. We compare particular poses in Figure 12.

We evaluate the speed of our technique in Table 1. While we are slower than SSD, the performance difference is always within a factor of two. While faster GPU implementations are possible, we use the same matrix-palette skinning implementation for both our method and SSD. Both methods were benchmarked on a Pentium 4 2.8 GHz machine with an NVIDIA GeForce 8800 GTX. We also estimate the number of floating-point operations for each method to provide a hardware independent perspective on performance.

The memory usage for our technique is significantly higher than that of SSD. However, even for the largest models, the memory usage is well within the capabilities of modern graphics hardware. Training for our model is also reasonable for an offline processing step.

Overall, our technique approximates deformations significantly better than SSD, while generalizing well and being comparable enough in speed to serve as its replacement.

Comparison with Displacement Interpolating Approaches: We highlight the limitations of displacement interpolation for the case of a simple two-joint twisting model, illustrated in the Bar and Elbow examples of Figure 13. One joint twists 180 degrees with respect to the other. This example is a special case of when Eigenskin [Kry et al. 2002] and pose space deformation [Lewis et al. 2000] are equivalent. Our model can learn the twisting deformation with just three training examples, while pose space deformation, though a perfect approximator, fails to generalize correctly to the new pose.

Comparison with Rotation Interpolating Approaches: The insertion of half-way joints and expanding/contracting joints as proposed by Mohr and Gleicher [2003] can perfectly model the twisting effects in Figure 13. In other cases, the technique is less accurate. We highlight the limitations of Mohr and Gleicher's technique with an anatomically rigged arm (Figure 14). In this case, the elbow is undergoing both bending *and* twisting. Applying Mohr and Gleicher's model allows the vertices of the forearm to choose a linear

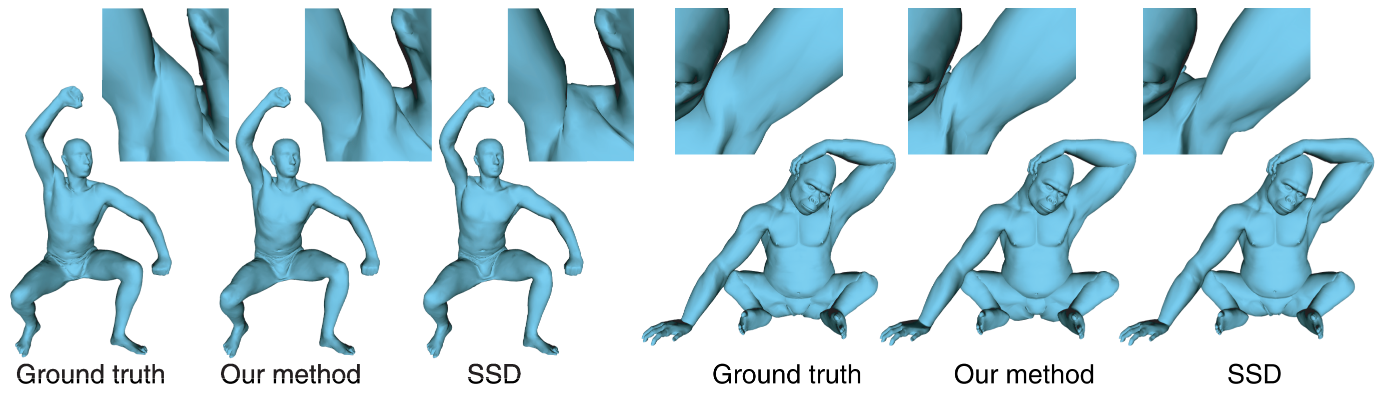


Figure 12: *Our model captures deformations of 3-D human scan data and artist created data. In both cases, we are better at approximating the shoulders than SSD.*

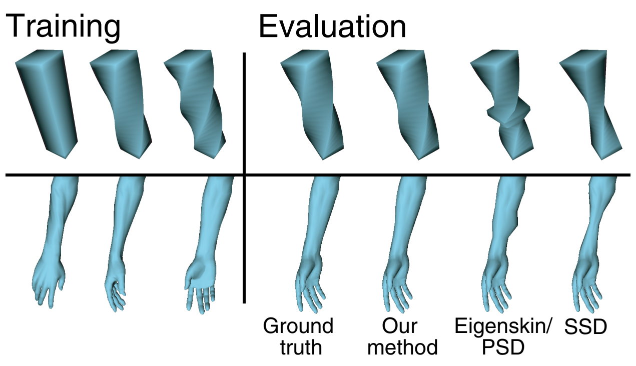


Figure 13: *Twisting bar and arm test. We took three poses from an animation sequence of a twisting bar and trained an SSD model, an EigenSkin/PSD model, and our model. We evaluated each model on an unseen test pose. While SSD has difficulty even representing the twisting deformation, the EigenSkin/PSD model overtrains on the small set of poses and fails to correctly interpolate the twist.*

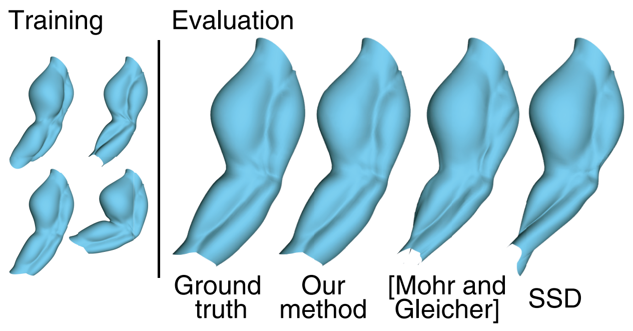


Figure 14: *Anatomical arm test. We extracted a set of poses from an anatomically motivated arm rig with both bending and twisting at the elbow. The twisting and muscle bulges are enough to prevent SSD from approximating the examples well. The technique of Mohr and Gleicher [2003] does better, but there are still differences. Our model produces a result almost indistinguishable from the ground truth.*

combination of the bending joint and the half-way twisting joint, but not the correct solution—a joint that bends *and* twists halfway. While more joints can always be added manually, our method locates where they are most needed and adds them automatically.

8 Conclusion

We have presented an example-based enveloping model suitable for use in interactive animation systems. Specifically, our experiments have shown that rotational regression is an effective way of capturing muscle bulging, twisting and areas such as the shoulder. We have tested our technique on a wide variety of examples, measuring both approximation and generalization error. Our method compares favorably to previous techniques in terms of quality and is usually within a factor of two of SSD in terms of speed.

Our model is good at approximating smooth, large-scale deformations such as muscle bulges. However, more sophisticated deformations, and in particular, deformations that cannot be linearly related to the underlying bone rotations may not be robustly captured. We strike a careful balance between simplicity and expressiveness, capturing only common deformations to avoid overfitting. If training poses need to be reproduced exactly, our technique can be augmented with a displacement correction model such as pose space deformation. While our runtime model is slower than SSD, it is still fast enough to not be a bottleneck for real-time rendering. Our memory requirements are significantly higher than that of SSD. For highly memory-sensitive applications, the ϵ parameter from Section 4.2 can be used to smoothly reduce memory usage by sacrificing accuracy. Furthermore, our model is more complex than linear blend skinning, displacement interpolating techniques, and the work of Mohr and Gleicher [2003].

We have shown that rotational regression is a surprisingly powerful model that, combined with a reduced Poisson formulation and dimensionality reduction, can be used in real-time applications. An exciting avenue of future work is to find an analog to our rotational model for dynamics.

Acknowledgments

We are grateful to Ilya Baran for help with formulating the rotational regression model, Jiawen Chen for guidance with the GPU implementation, and Sylvain Paris for his regular, insightful feedback. We thank Joel Anderson for kindly providing the T-Rex model, and Michael Comet for the Dragon Leg and Muscle Arm models. All three models were created with the cMuscleSystem software (Copyright 2005-2007 Comet Digital, LLC, All Rights Reserved). Dragomir Anguelov generously provided data and support for the Drago model. This work was supported in part by Pixar

Animation Studios, the Nokia Research Center, the National Science Foundation, and the NVIDIA Corporation.

References

ALLEN, B., CURLESS, B., POPOVIĆ, Z., AND HERTZMANN, A. 2006. Learning a correlated model of identity and pose-dependent body shape variation for real-time synthesis. In *Symposium on Computer Animation*, 147–156.

ANGUELOV, D., SRINIVASAN, P., KOLLER, D., THRUN, S., RODGERS, J., AND DAVIS, J. 2005. SCAPE: shape completion and animation of people. *ACM Trans. Graph. 24*, 3 (Aug.), 408–416.

CAPELL, S., GREEN, S., CURLESS, B., DUCHAMP, T., AND POPOVIĆ, Z. 2002. Interactive skeleton-driven dynamic deformations. *ACM Trans. Graph. 21*, 3 (July), 586–593.

CAPELL, S., BURKHART, M., CURLESS, B., DUCHAMP, T., AND POPOVIĆ, Z. 2005. Physically based rigging for deformable characters. In *Symposium on Computer Animation*, 301–310.

CGCHARACTER, 2003. Absolute character tools 1.6. www.cgcharacter.com.

COHEN-STEINER, D., ALLIEZ, P., AND DESBRUN, M. 2004. Variational shape approximation. *ACM Trans. Graph. 23*, 3 (Aug.), 905–914.

COMET, M., 2006. Cmusclesystem 1.31. www.cometdigital.com.

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

DIEBEL, J. R., THRUN, S., AND BRÜNIG, M. 2006. A bayesian method for probable surface reconstruction and decimation. *ACM Trans. Graph. 25*, 1 (Jan.), 39–59.

EGGERT, D. W., LORUSSO, A., AND FISHER, R. B. 1997. Estimating 3-d rigid body transformations: a comparison of four major algorithms. *Mach. Vision Appl. 9*, 5-6, 272–290.

GUO, Z., AND CHEONG WONG, K. 2005. Skinning With Deformable Chunks. *Computer Graphics Forum 24*, 3, 373–381.

HARGREAVES, S. 2004. Deferred shading. In *Proceedings of the Game Developers Conference*.

HYUN, D.-E., YOON, S.-H., CHANG, J.-W., SEONG, J.-K., KIM, M.-S., AND JÜTTLER, B. 2005. Sweep-based human deformation. *The Visual Computer 21*, 8-10, 542–550.

JAMES, D. L., AND TWIGG, C. D. 2005. Skinning mesh animations. *ACM Trans. Graph. 24*, 3, 399–407.

KARNI, Z., AND GOTSMAN, C. 2004. Efficient compression of soft-body animation sequences. *Computer and Graphics 28*, 1 (Feb.), 25–34.

KAVAN, L., AND ŽÁRA, J. 2005. Spherical blend skinning: a real-time deformation of articulated models. In *Symposium on Interactive 3D Graphics and Games*, 9–16.

KRY, P. G., JAMES, D. L., AND PAI, D. K. 2002. EigenSkin: real time large deformation character skinning in hardware. In *Symposium on Computer Animation*, 153–159.

LEE, M. 2006. Seven ways to skin a mesh: Character skinning revisited for modern GPUs. In *Proceedings of GameFest, Microsoft Game Technology Conference*.

LEWIS, J. P., CORDNER, M., AND FONG, N. 2000. Pose space deformations: A unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 165–172.

MERRY, B., MARAIS, P., AND GAIN, J. 2006. Animation space: A truly linear framework for character animation. *ACM Trans. Graph. 25*, 4 (Oct.), 1400–1423.

MOHR, A., AND GLEICHER, M. 2003. Building efficient, accurate character skins from examples. *ACM Trans. Graph. 22*, 3 (July), 562–568.

RHEE, T., LEWIS, J., AND NEUMANN, U. 2006. Real-time weighted pose-space deformation on the GPU. *Computer Graphics Forum 25*, 3 (Sept.), 439–448.

SCHEEPERS, F., PARENT, R. E., CARLSON, W. E., AND MAY, S. F. 1997. Anatomy-based modeling of the human musculature. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, 163–172.

SLOAN, P.-P. J., CHARLES F. ROSE, I., AND COHEN, M. F. 2001. Shape by example. In *Symposium on Interactive 3D Graphics and Games*, 135–143.

SUMNER, R. W., AND POPOVIĆ, J. 2004. Deformation transfer for triangle meshes. *ACM Trans. Graph. 23*, 3, 399–405.

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

TERAN, J., SIFAKIS, E., IRVING, G., AND FEDKIW, R. 2005. Robust quasistatic finite elements and flesh simulation. In *Symposium on Computer Animation*, 181–190.

WANG, X. C., AND PHILLIPS, C. 2002. Multi-weight enveloping: Least-squares approximation techniques for skin animation. In *Symposium on Computer Animation*, 129–138.

WEBER, J. 2000. Run-time skin deformation. In *Proceedings of Game Developers Conference*.

WILHELMS, J., AND GELDER, A. V. 1997. Anatomically based modeling. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, 173–180.