

Real-Time Enveloping with Rotational Regression

by

Robert Yuanbo Wang

Submitted to the Department of Electrical Engineering and Computer
Science

in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2007

© Massachusetts Institute of Technology 2007. All rights reserved.

Author
Department of Electrical Engineering and Computer Science
February 1st, 2007

Certified by
Jovan Popovic
Associate Professor
Thesis Supervisor

Accepted by
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Real-Time Enveloping with Rotational Regression

by

Robert Yuanbo Wang

Submitted to the Department of Electrical Engineering and Computer Science
on February 1st, 2007, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

Enveloping (or skinning) is the process that relates a skeleton, which an animator controls, to a 3-D surface mesh, which the audience sees. This process is necessary in most computer graphics applications that involve animated characters. The complexity (and speed) of enveloping solutions vary from photo-realistic muscle simulations used for movie production, to artifact-ridden heuristics such as linear blend skinning used for video games and training simulations.

We propose a method for example-based enveloping of 3-D characters. We can approximate the output of muscle simulations or other high-quality enveloping tools with a model that can be evaluated at speeds comparable to the fastest enveloping techniques. Our technique introduces a rotational regression model that can accurately capture common skinning behaviors such as muscle bulging, twisting, and challenging areas such as the shoulders. Our better treatment of rotational quantities is made possible by a framework that predicts mesh deformation gradients instead of mesh vertex positions. We reconstruct the vertex positions from deformation gradients in an additional step by solving a Poisson equation.

We show that our model is significantly more expressive than linear blend skinning and capable of capturing a wider variety of effects without generalization problems. Our method is also comparable in run-time speed to linear blend skinning. All in all, we are proposing a more general, less artifact-ridden, replacement for linear blend skinning that will allow for more realistic enveloping in real-time applications.

Thesis Supervisor: Jovan Popovic

Title: Associate Professor

Acknowledgments

Thanks goes to Ilya Baran, Marco DaSilva, Paul Green, Eugene Hsu, Tom Mertens, and Sylvain Paris for conversation and discussion. Thanks goes to my office mate Jiawen Chen for providing technical advice on GPUs and software tools. And of course, special thanks go to my collaborators, Kari Pulli and Jovan Popovic.

We thank Joel Anderson, Dragomir Anguelov, and Michael Comet for their generosity in providing the datasets we used to test our model. The T-Rex example was provided by Joel Anderson. The Drago example was provided by Dragomir Anguelov. The Elbow, Shoulder, James and Gorilla examples were taken from Poser 6. The Muscle Arm and Dragon Leg were provided with the cMuscleSystem by Michael Comet.

Contents

1	Introduction	15
1.1	Related Work	16
1.2	Overview	18
2	Deformation Gradient Prediction	23
2.1	Notation	24
2.2	Rotational Regression	24
2.2.1	Model	25
2.2.2	Training	26
2.3	Scale/Shear Regression	27
3	Mesh Reconstruction	29
3.1	Poisson Mesh Reconstruction	29
3.2	Near-Rigid/SSD Vertex Constraints	30
3.3	Reduced Mesh Reconstruction	31
4	Dimensionality Reduction	35
4.1	Vertex reduction	35
4.2	Predictor Reduction	37
5	GPU Implementation	39
6	Results	41
6.1	Error Metric	41

7 Conclusion	47
A Reduced Poisson Formulation Details	53

List of Figures

1-1	Our technique accurately captures muscle deformations from a set of examples and efficiently evaluates them on graphics hardware. In this case, we approximate the anatomical model used to produce the ground truth (29,380 vertices) at speeds suitable for interactive applications. Our technique is more accurate than linear blend skinning and almost as fast.	20
1-2	Overview. Our model maps a skeletal pose \mathbf{q} to its corresponding mesh pose $\mathbf{y}(\mathbf{q})$. 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.	21
2-1	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 rotation model $R(\mathbf{q})$ and scale/shear model $S(\mathbf{q})$ combine to form our deformation gradient predictor.	24
2-2	Arm flexing. While most of the forearm rotates in the same direction and amount as the bone, parts of the bicep rotate along different axes and by different amounts.	25
2-3	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).	26

2-4	Muscle arm error histogram. 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.	27
3-1	“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).	30
4-1	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.	37
4-2	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.	37
5-1	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.	39
6-1	Our errors are not only generally lower than SSD, but their standard deviations (error-bars) are smaller as well, meaning that the errors are 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.) . .	43

6-2	Our model captures real deformations of 3-D human scan data. In this case, we are better at approximating the shoulders than SSD, despite noise in the training set.	45
6-3	The large shoulders of the gorilla are poorly approximated by SSD, but captured by our technique.	45
6-4	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.	46
6-5	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 [21] does better, but there are still differences. Our model produces a result almost indistinguishable from the ground truth.	46

List of Tables

6.1	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. The absolute speed is also sufficient for use in interactive applications.	44
-----	---	----

Chapter 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. 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 block-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.

Our rotational regression model predicts surface rotations from bone rotations. We also predict surface scaling and shearing separately, and solve a Poisson equation to reconstruct vertex positions from these predictions. Our technique makes the following contributions:

Rotational Regression Enveloping: We propose an enveloping technique that naturally models rotations. This is in contrast to techniques that linearly blend rotation matrices or use displacements to model rotational deformations. We apply a simple but general heuristic that maps bone rotations (in pose space) to triangle rotations on the surface mesh.

Accurate and GPU Ready Poisson Reconstruction: We formulate a more accurate Poisson mesh reconstruction process by integrating easy-to-predict near-rigid vertices as constraints. We also adapt our reconstruction framework for model reduction, allowing it to be computed on the GPU.

Dimensionality Reduction: Finally, we present a general dimensionality reduction scheme compatible with our reduced mesh reconstruction framework that approximates a sequence of poses with bounded error.

1.1 Related Work

Physically-based and Anatomically-based approaches: Some of the most realistic results in enveloping have been obtained by muscle simulations [23, 30, 27]. Commercial packages for muscle simulations are readily available and commonly used in movie production and special effects [5, 7]. Other approaches use physics but not anatomy to describe muscle deformations [13, 4, 3, 11]. 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* [18]. 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 for a set of examples [21, 14].

Linearly blending rotations leads to well-known artifacts such as collapsing joints and the “candy wrapper effect” [19, 21]. There have been many techniques that address these problems. Wang and Phillips [28] and Merry et al. [20] 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 [19, 24, 17, 1, 22]. 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: Weber [29], and Mohr and Gleicher [21] 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 defor-

mations given by example poses. We extend the idea of introducing additional joints by identifying precisely where they are needed (Section 4) 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 [16] 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 gradient approaches: Deformation gradients have been used by a variety of techniques for pose modeling [25, 2, 8]. We share with these techniques a common Poisson reconstruction step, but differ in how we model the deformation gradients. Der et al. [8] 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. [8] 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 [2]. 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.

1.2 Overview

Our task is to learn 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 region of the mesh with respect to a *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 2). From these predictions, we can reconstruct the mesh vertex positions by solving a Poisson equation (Section 3). 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 4) 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 5).

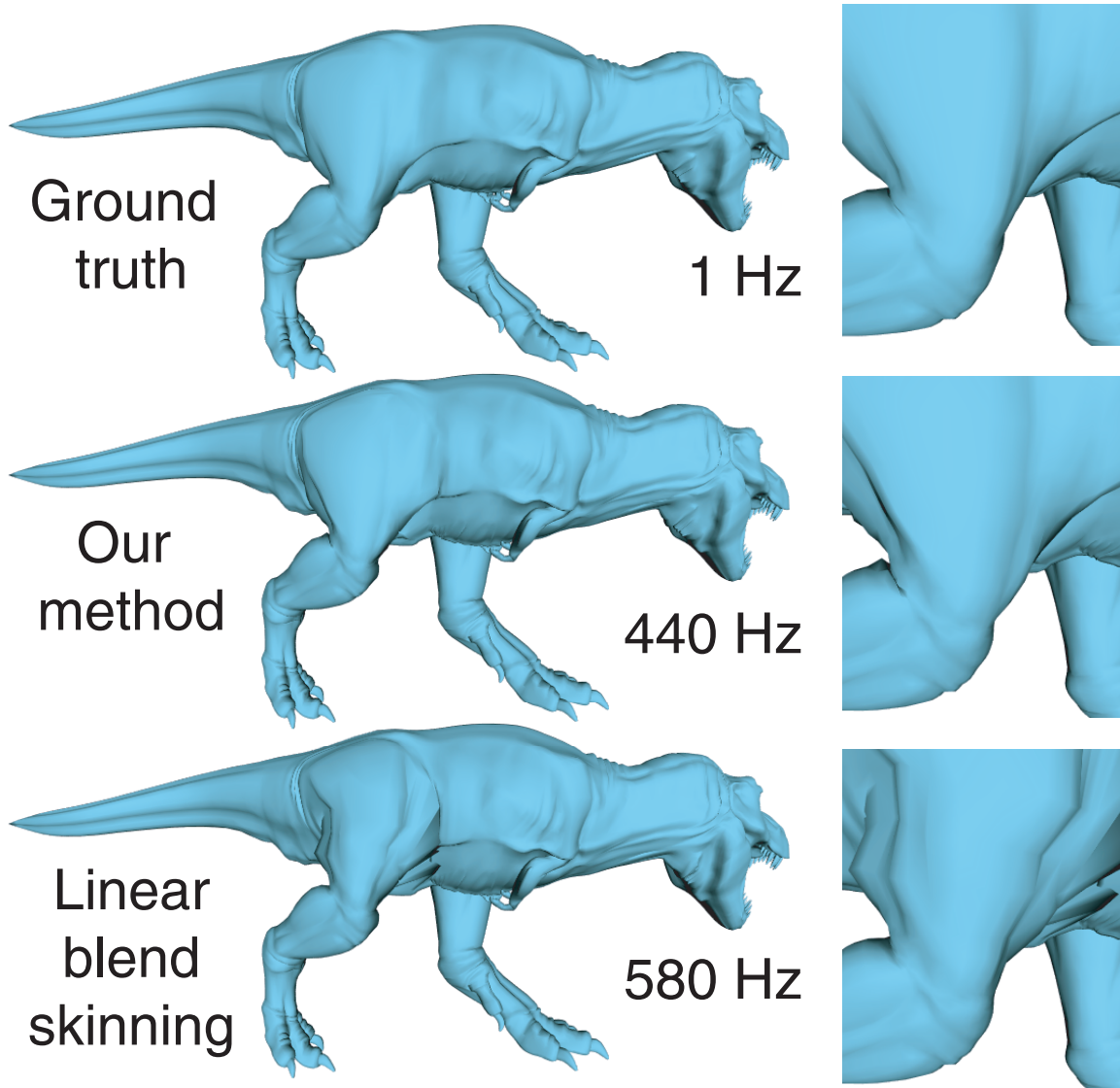


Figure 1-1: *Our technique accurately captures muscle deformations from a set of examples and efficiently evaluates them on graphics hardware. In this case, we approximate the anatomical model used to produce the ground truth (29,380 vertices) at speeds suitable for interactive applications. Our technique is more accurate than linear blend skinning and almost as fast.*

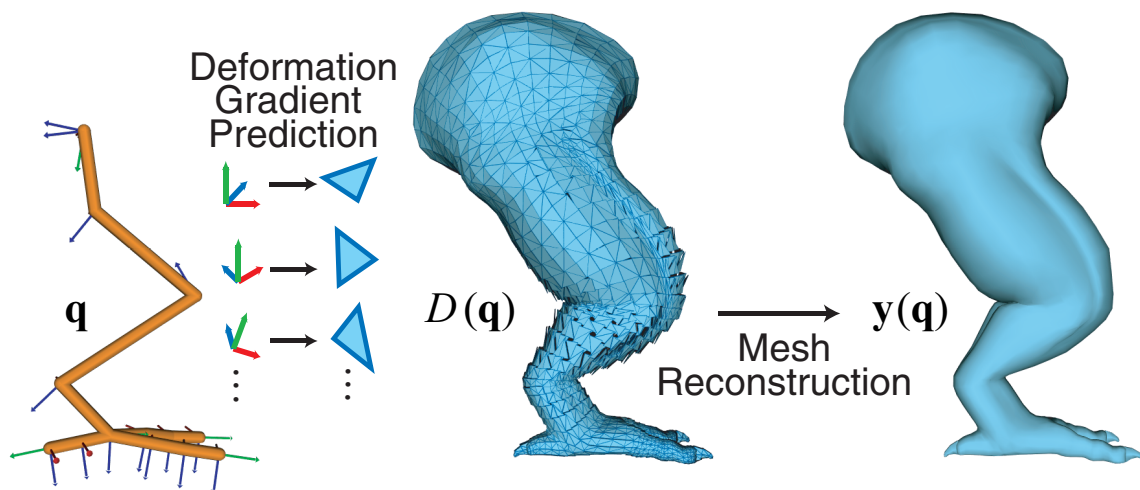


Figure 1-2: Overview. Our model maps a skeletal pose \mathbf{q} to its corresponding mesh pose $\mathbf{y}(\mathbf{q})$. The mapping has two steps. First, we predict deformation gradients of the mesh based on the skeletal pose. We then reconstruct vertex positions from these predictions by solving a Poisson equation.

Chapter 2

Deformation Gradient Prediction

The deformation gradient is a local description of orientation, scaling, and shearing of a deformed pose relative to a rest (or undeformed) pose. Let \mathbf{y} and $\hat{\mathbf{y}}$ be the coordinates of a body at a deformed pose and at the rest pose respectively. The quantity $D = \nabla_{\hat{\mathbf{y}}}\mathbf{y}$ is called the deformation gradient. If \mathbf{y} is given by an affine transformation, $\mathbf{y} = A\hat{\mathbf{y}} + \mathbf{b}$, as is the case for the three vertices of a triangle, the deformation gradient is the matrix $A \in \mathbb{R}^{3 \times 3}$. We seek to relate bone transformations \mathbf{q} to mesh deformations D . For an articulated rigid-body, this mapping is the identity between each bone and the rigid segment it affects. Linear blend skinning generalizes this mapping, allowing surface 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.

We extract deformation gradient sequences from a set of deformed mesh poses on a per-triangle basis, similarly to Sumner and Popović [25], and James and Twigg [14]. The deformation gradient D can be separated into a rotation 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 $D(r)$ (Figure 2-1). While we begin by describing deformation gradient predictors on a per-triangle basis, our model can be applied to any sequence of deformation gradients—a property we exploit for our reduced reconstruction step.

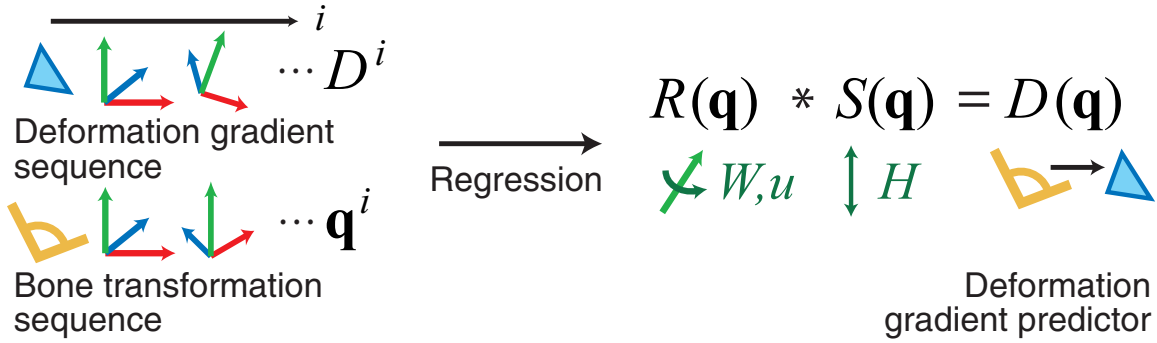


Figure 2-1: We learn a mapping from a sequence of bone transformations to a sequence of deformation gradients. We build separate regression models for rotation and scale/shear, learning parameters W and u for rotation and H for scale/shear. The rotation model $R(\mathbf{q})$ and scale/shear model $S(\mathbf{q})$ combine to form our deformation gradient predictor.

2.1 Notation

We denote each skeletal pose \mathbf{q} to be a vector of J concatenated bone transformations $[\mathbf{vec}(Q_1)^T, \mathbf{d}_1^T, \dots, \mathbf{vec}(Q_J)^T, \mathbf{d}_J^T]^T \in \mathbb{R}^{12J \times 1}$. 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.

2.2 Rotational Regression

The basic assumption of SSD is that vertices on the mesh transform with 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 2-2). We propose a more general model relating a joint rotation sequence to a triangle rotation sequence.

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.

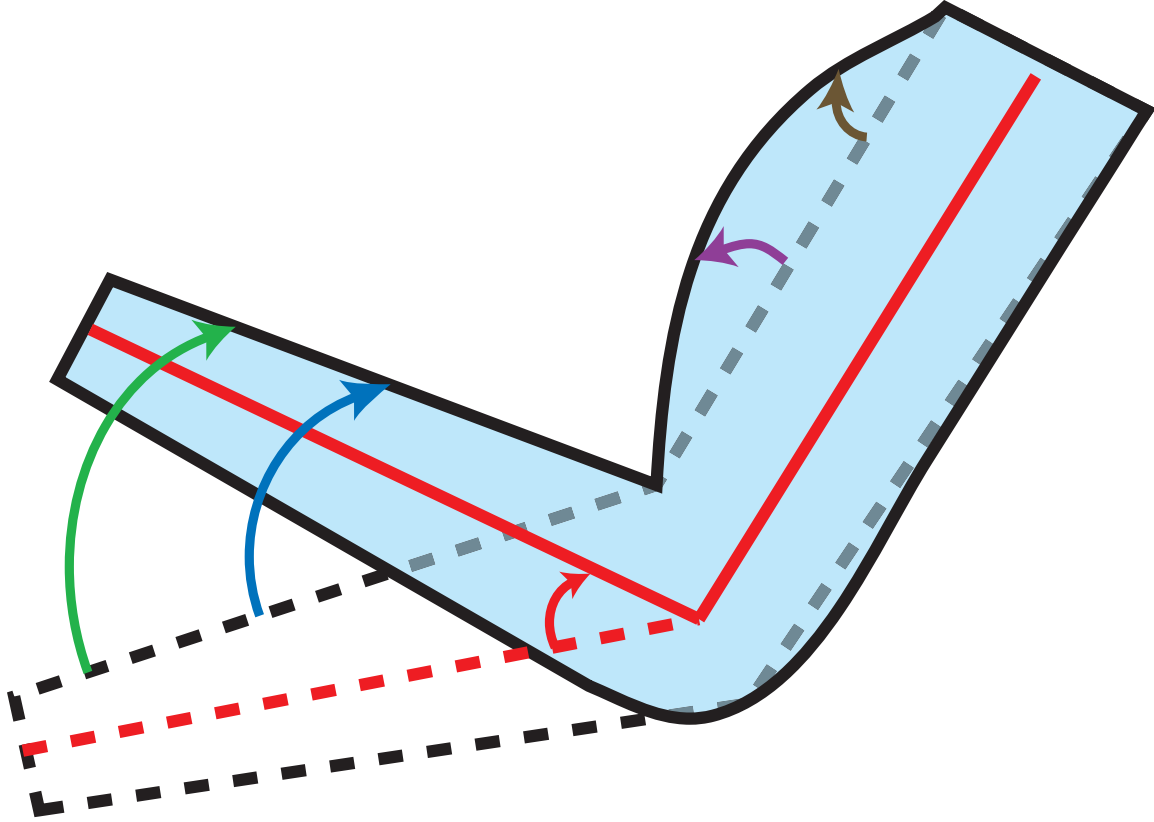


Figure 2-2: *Arm flexing. While most of the forearm rotates in the same direction and amount as the bone, parts of the bicep rotate along different axes and by different amounts.*

2.2.1 Model

To relate bone rotations to triangle rotations, we first need to express both in the same coordinate frame. Let $\tilde{\theta}$ and $\tilde{\rho}$ denote our bone rotations and triangle rotations expressed in the joint frame. Intuitively, $\tilde{\theta}$ 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 takes on a linear form:

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

where $\theta_b(\mathbf{q})$ extracts the rotation of bone b from skeletal pose \mathbf{q} (Figure 2-3).

For each triangle, we are fitting only four parameters (three for the rotation offset W and one for the scale factor u). However, this simple model works surprisingly well. The model handles twisting with the rotation scale factor u , while the rotation offset to the axis

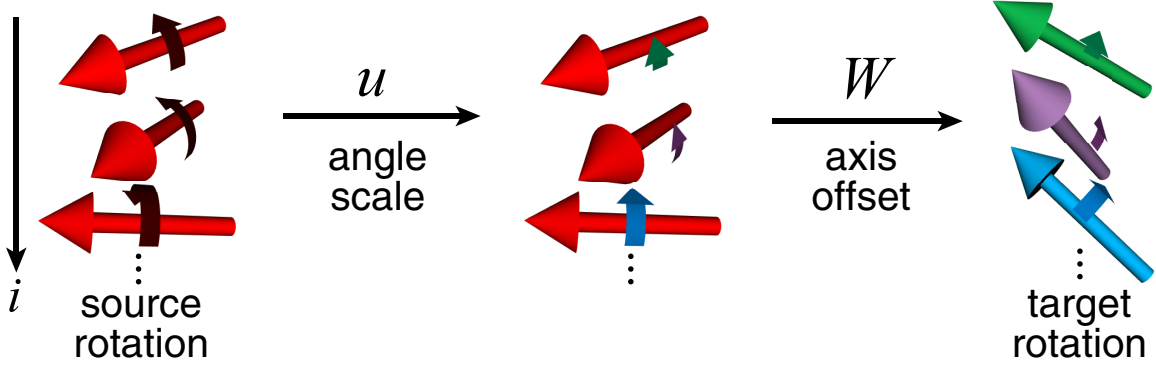


Figure 2-3: Our rotation predictors learn a constant rotational axis offset W and a constant angle scale factor u to relate a joint rotation sequence (source) to a triangle rotation sequence (target).

effectively models muscle bulges (Figure 2-4).

2.2.2 Training

For training, we are given rotation sequences for a bone \mathbf{q}^i and a triangle ρ^i . First, we transform both sets of rotations into the joint frame, forming $\tilde{\theta}_b(\mathbf{q}^i)$ and $\tilde{\rho}^i$. The optimal parameters $W \in \mathbb{SO}_3$ and $u \in \mathbb{R}$ are given by

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

which can be solved with coordinate descent, alternating between solving for u and Q separately. Given rotation offset W , 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 [10].

We initialize the scale factor u independently of W :

$$\operatorname{argmin}_u \sum_{i \in 1 \dots N} (u\|\tilde{\theta}_b(\mathbf{q}^i)\| - \|\tilde{\rho}^i\|)^2.$$

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 naively fit a model for

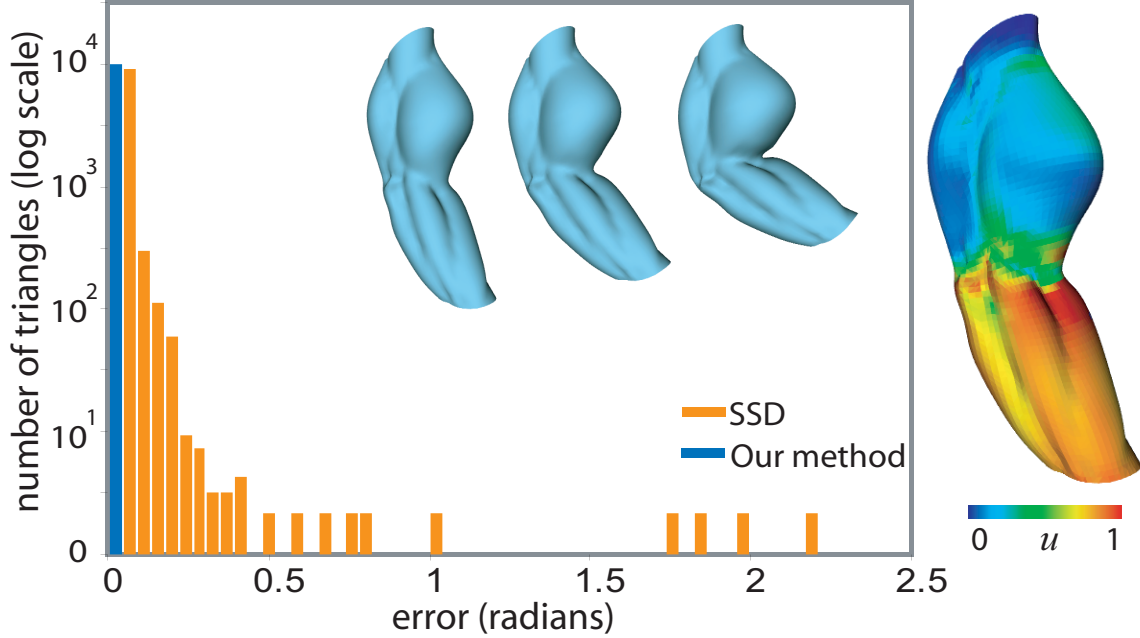


Figure 2-4: Muscle arm error histogram. We plot a histogram of the errors for our rotational regression heuristic compared to SSD for three poses of a muscle flexing. We also show the angle scale factor in false color. Note that triangles on the bicep are related to the joint rotation by a small (but non-zero) scale factor while triangles on the forearm move nearly rigidly.

each joint and select the best one that fits our triangle rotation sequence. We can also 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.

2.3 Scale/Shear Regression

We predict each component of the scale/shear matrix linearly with respect to the axis-angle representation of two joints [2],

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

The two joint rotations that we use are the joint associated with the best rotational predictor found by the rotational regression step $\boldsymbol{\theta}_{b_1}$, and its parent $\boldsymbol{\theta}_{b_2}$. We denote $\tilde{\boldsymbol{\theta}}_{b_1, b_2}(\mathbf{q})$ as $[\tilde{\boldsymbol{\theta}}_{b_1}(\mathbf{q})^T \tilde{\boldsymbol{\theta}}_{b_2}(\mathbf{q})^T 1]^T \in \mathbb{R}^{7 \times 1}$. 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:

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

Chapter 3

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.

3.1 Poisson Mesh Reconstruction

Our deformation gradient predictions describe the shape and orientation of triangles, which are then pieced together while enforcing vertex continuity constraints to obtain the final mesh. Our Poisson equation relates deformation gradients to vertices through edges [2]:

$$\operatorname{argmin}_{\mathbf{y}} \sum_{k \in 1 \dots T} \sum_{j=2,3} \|D_k(\mathbf{q}) \hat{\mathbf{v}}_{k,j} - \mathbf{v}_{k,j}\|^2, \quad (3.1)$$

where $\mathbf{v}_{k,j} = \mathbf{y}_{k,j} - \mathbf{y}_{k,1}$ denotes the j th edge of the k th triangle in the pose we are solving for and $\hat{\mathbf{v}}_{k,j}$ denotes the same edge in the rest pose. Equation 3.1 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 [26]. Given the per-triangle deformation gradient predictions for a new pose, we can obtain the vertex positions by back substitution.

3.2 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 3-1), 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 SSD predicts well and improves the quality where it does not.

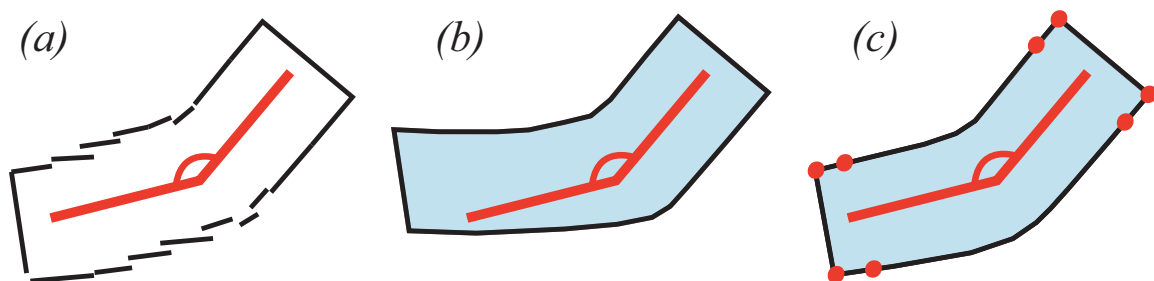


Figure 3-1: “Fixing” Poisson. We use the Poisson equation to reconstruct vertex positions from edge predictions (a). However, low-frequency errors can accumulate, and the extremities of the mesh may no longer match the joint configuration (b). Fixing a set of near-rigid vertices to their SSD prediction (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 $\sum_b^J w_{a,b} T_b(\mathbf{q}) \hat{y}_a$, the SSD prediction of vertex a at pose \mathbf{q} . We obtain our SSD weights $w_{a,b}$ by non-negative least-squares [14]. We substitute $\mathbf{y}_a = \Psi_a \mathbf{q}$ for all $\mathbf{y}_a \in F$ into

Equation 3.1:

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

where

$$\mathbf{v}_{k,j} = \begin{cases} \mathbf{y}_{k,j} - \mathbf{y}_{k,1} & \text{if } \mathbf{y}_{k,j} \notin F \text{ and } \mathbf{y}_{k,1} \notin F \\ \mathbf{y}_{k,j} - \Psi_{k,1}\mathbf{q} & \text{if only } \mathbf{y}_{k,1} \in F \\ \Psi_{k,j}\mathbf{q} - \mathbf{y}_{k,1} & \text{if only } \mathbf{y}_{k,j} \in F. \end{cases} \quad (3.2)$$

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 3.2 similarly to Equation 3.1, 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.

3.3 Reduced Mesh Reconstruction

The optimization problem in Equation 3.2 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 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 4.

Our reduced mesh reconstruction scheme is based on the idea that the triangles and vertices of an articulated model are very coordinated. We express each triangle deformation

gradient predictor as a linear combination of P key deformation gradient predictors:

$$D_k(\mathbf{q}) = \sum_{\ell \in 1 \dots P} \beta_{k,\ell} D_\ell(\mathbf{q}). \quad (3.3)$$

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

$$\mathbf{y}_a(\mathbf{t}) = \sum_{b \in 1 \dots P} \alpha_{a,b} T_b(\mathbf{t}) \hat{\mathbf{y}}_a = \Phi_a \mathbf{t} \quad (3.4)$$

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 3.3 and 3.4 into Equation 3.2 and solve for the proxy-bone transformations \mathbf{t} :

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

where

$$\mathbf{v}_{k,j} = \begin{cases} \Phi_{k,j} \mathbf{t} - \Phi_{k,1} \mathbf{t} & \text{if } \mathbf{y}_{k,j} \notin F \text{ and } \mathbf{y}_{k,1} \notin F \\ \Phi_{k,j} \mathbf{t} - \Psi_{k,1} \mathbf{q} & \text{if only } \mathbf{y}_{k,1} \in F \\ \Psi_{k,j} \mathbf{q} - \Phi_{k,1} \mathbf{t} & \text{if only } \mathbf{y}_{k,j} \in F \end{cases} \quad (3.5)$$

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

$$\mathbf{t}(\mathbf{q}) = C_1 \mathbf{d}(\mathbf{q}) + C_2 \mathbf{q}, \quad (3.6)$$

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 Appendix. To obtain the vertex positions \mathbf{y} , we substitute $\mathbf{t}(\mathbf{q})$ for \mathbf{t} in Equation 3.4.

Thus we have reduced our entire Poisson mesh reconstruction step into a matrix-vector multiplication (Equation 3.6) and matrix-palette skinning (Equation 3.4). We describe in Section 5 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 multiplies per predictor.

Chapter 4

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 for skinning mesh animations by James and Twigg [14] 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.

4.1 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 smallest number of proxy-bones P for a given maximum error threshold ϵ . This would

require us to solve

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

However, this optimization is too intractable for us to attack directly. Given fixed P and fixed transformations T_b^i , we can solve for weights $\alpha_{a,b}$ using non-negative least-squares [14]. Similarly, given fixed P and fixed $\alpha_{a,b}$, we can find the least-squares solution for 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 [6, 9].

Define the error $E_{A \rightarrow B}$ 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 4-1). 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 $A \rightarrow B$ with the smallest error $E_{A \rightarrow B}$ 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.

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 [6]. 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.

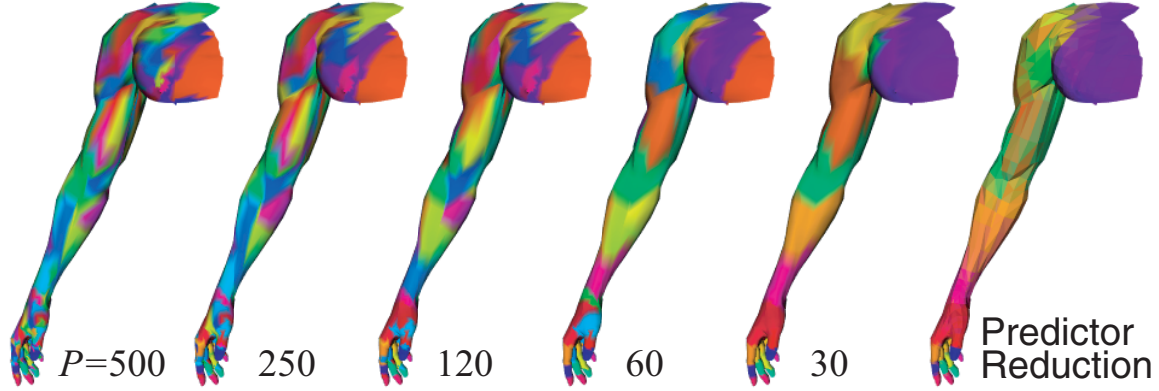


Figure 4-1: *Vertex clustering. Successive iterations merge coordinated vertices into fewer and fewer proxy-bones. The resulting transformations also form a good initial guess for predictor reduction.*

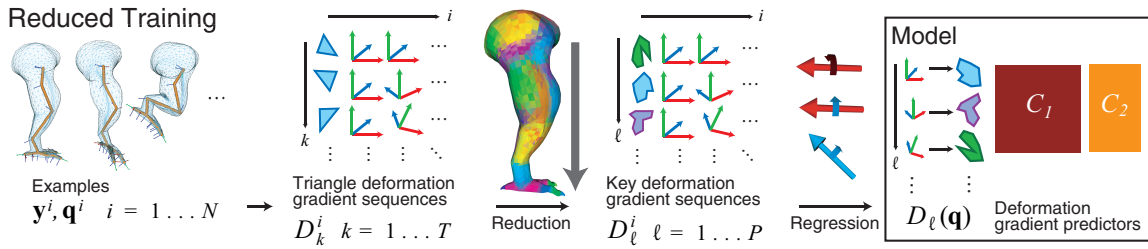


Figure 4-2: *Training a Reduced Model. Given a set of example skeleton-mesh pairs, we can extract triangle deformation sequences D_k^i . Our predictor reduction step gives us sequences of key deformation gradient sequences D_l^i , from which we train key deformation gradient predictor. These predictors, combined with the mesh reconstruction matrices C_1 and C_2 , form our model.*

4.2 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 2 (Figure 4-2). Our error metric for finding the best key sequences is the objective function from Equation 3.2 with the substitution

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

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

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

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 (Figure 4-1). 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 2.

Chapter 5

GPU Implementation

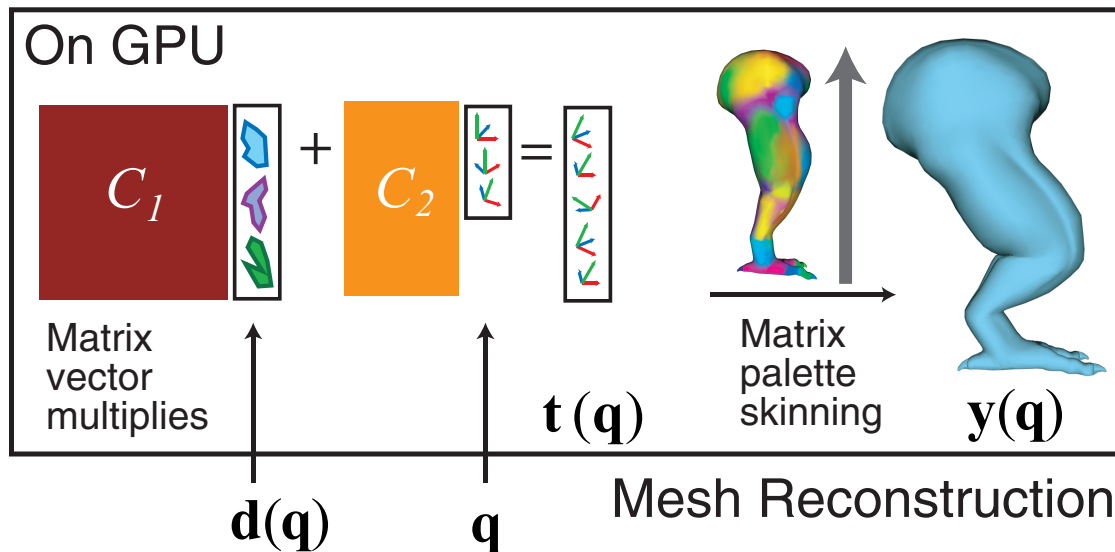


Figure 5-1: Mesh reconstruction on the GPU. We load C_1 , C_2 and the bone weights for matrix-palette skinning on the GPU beforehand. At runtime, we need only send the vectors $d(\mathbf{q})$ and \mathbf{q} per model.

There are two components of our GPU implementation: a matrix-vector multiplication and matrix-palette skinning. 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 [18] 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 [18] and deferred shading [12]. 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.

Chapter 6

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 cMuscleSystem [7], and 3-D human scan data [2].

6.1 Error Metric

We evaluate all our examples using a metric inspired by the *percent position error* (PPE) developed by Karni and Gotsman [15] 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 .

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 [14]. Like James and Twigg [14], we cap the number of non-zero weights at four. In Figure 6-1, we show that our technique is superior in terms of quality on every example we tried. Not only is our total enveloping error lower, the variance of our errors across the vertices is also lower, meaning that our errors are low-frequency and less distracting visually. We compare particular poses in Figures 6-2 and 6-3.

We evaluate the speed of our technique in Table 6.1. 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 a NVIDIA GeForce 8800 GTX. We also estimate the number of floating-point operations for each method to provide a hardware independent perspective on performance.

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 6-4. One joint twists 180 degrees with

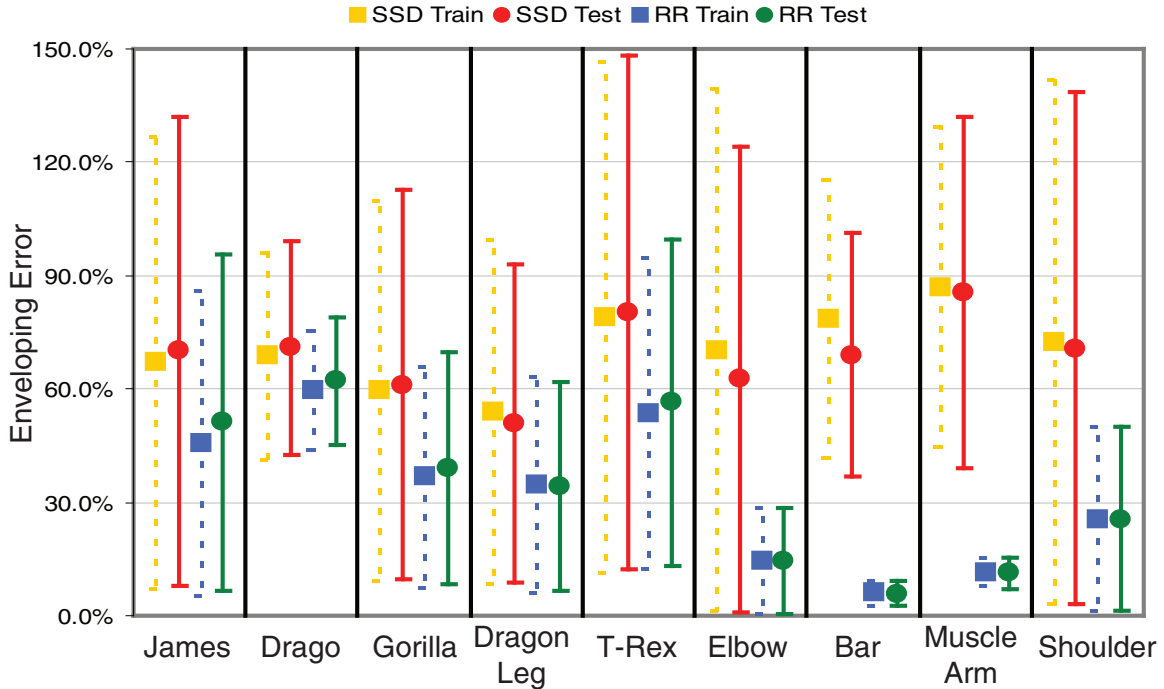


Figure 6-1: *Our errors are not only generally lower than SSD, but their standard deviations (error-bars) are smaller as well, meaning that the errors are harder to detect visually. All errors are normalized so that 100% corresponds to the best rigid-articulated body predictor for each example. Both approximation error (dotted line) and generalization error (solid line) for both SSD and our rotational regression method (RR) are shown. (Lower is better.)*

respect to the other. This example is a special case of when Eigenskin [17] and pose space deformation [19] 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 [21] can perfectly model the twisting effects in Figure 6-4. In other cases, the technique is less accurate. We highlight the limitations of Mohr and Gleicher’s technique with an anatomically rigged arm (Figure 6-5). 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 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

Example	Vertices	Joints	Proxy bones	Training Poses	Testing Poses	Our flops	SSD flops	Our fps	SSD fps
James	11106	73	80	31	LOOCV	5.1M	2.6M	595	1000
Drago	12500	16	80	49	LOOCV	5.0M	2.9M	618	1030
Gorilla	25438	61	100	46	LOOCV	9.9M	5.9M	449	673
Dragon Leg	2210	14	40	9	86	1.0M	0.5M	681	1144
T-Rex	29380	155	60	11	121	9.4M	6.8M	443	583
Elbow	2610	2	30	3	15	1.0M	0.6M	685	1164
Bar	80	2	25	3	50	0.2M	0.0M	711	1228
Muscle Arm	5256	3	30	4	40	2.0M	1.2M	692	1200
Shoulder	2610	2	40	10	100	1.0M	0.6M	690	1172

Table 6.1: *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. The absolute speed is also sufficient for use in interactive applications.*

method locates where they are most needed and adds them automatically.

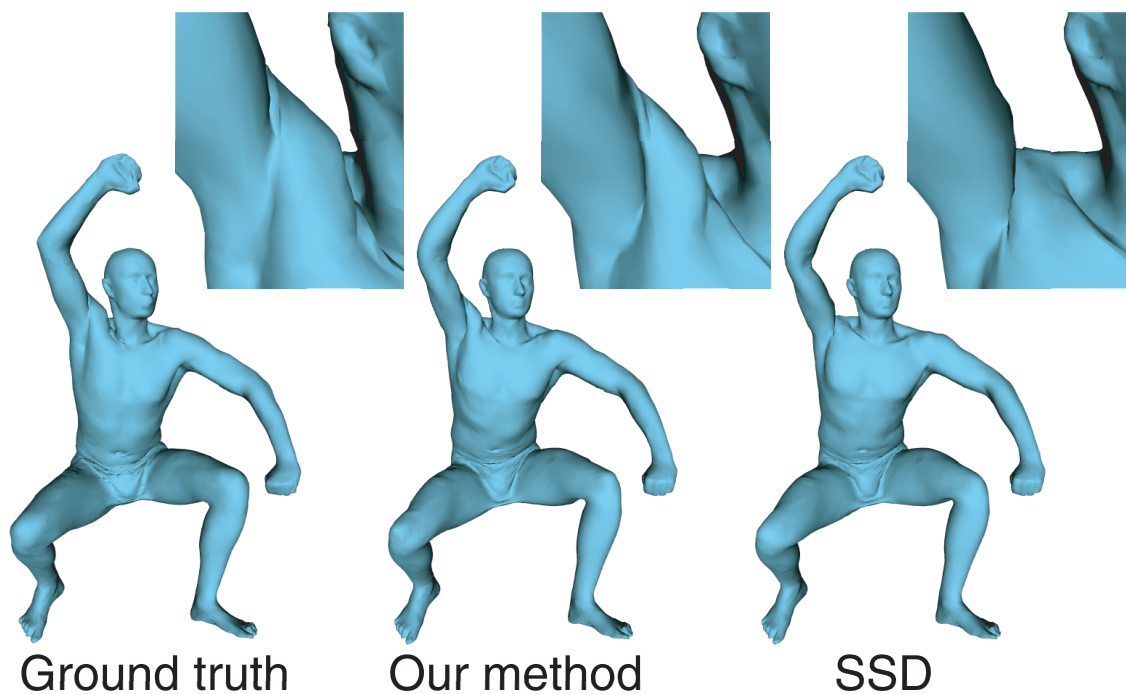


Figure 6-2: *Our model captures real deformations of 3-D human scan data. In this case, we are better at approximating the shoulders than SSD, despite noise in the training set.*

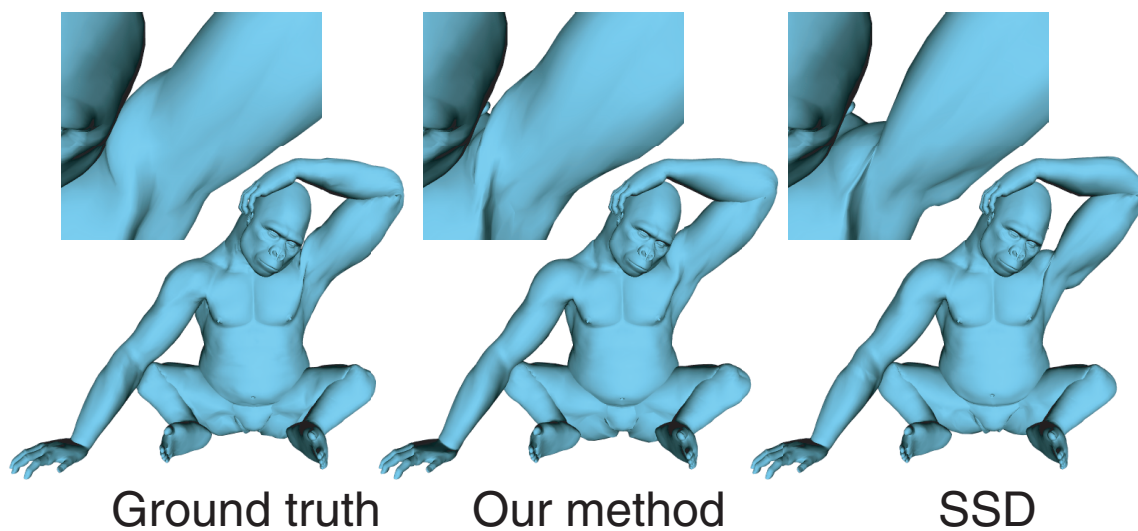


Figure 6-3: *The large shoulders of the gorilla are poorly approximated by SSD, but captured by our technique.*

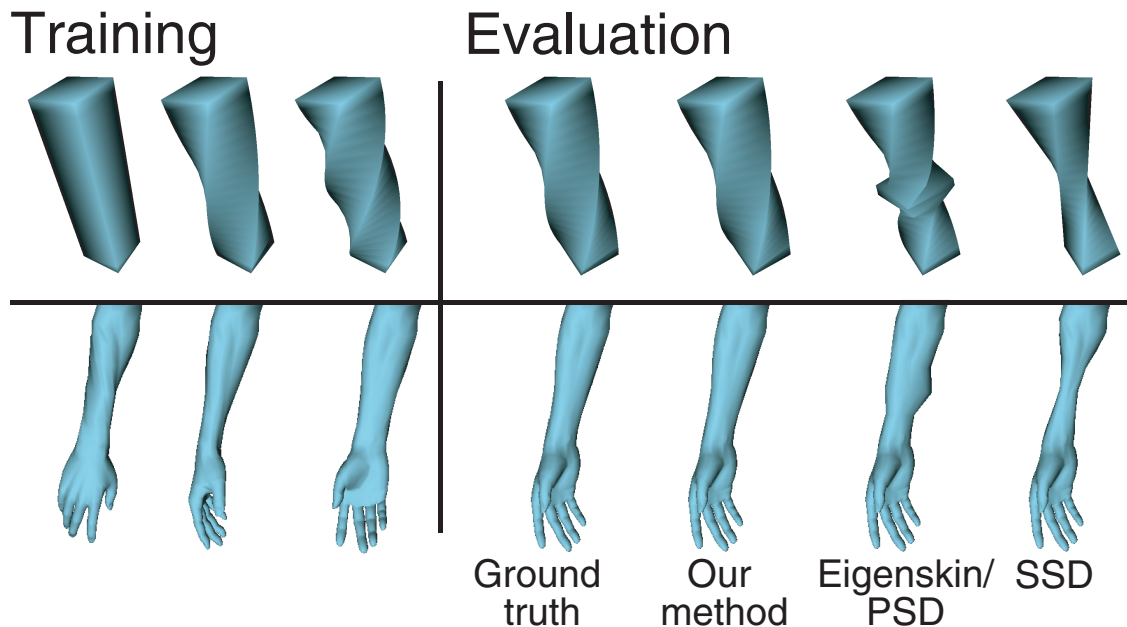


Figure 6-4: *Twisting bar and arm test.* We took three poses from an animation sequence of a twisting bar and trained an SSD model, an EigenSkin/PSD model, and our model. We evaluated each model on an unseen test pose. While SSD has difficulty even representing the twisting deformation, the EigenSkin/PSD model overtrains on the small set of poses and fails to correctly interpolate the twist.

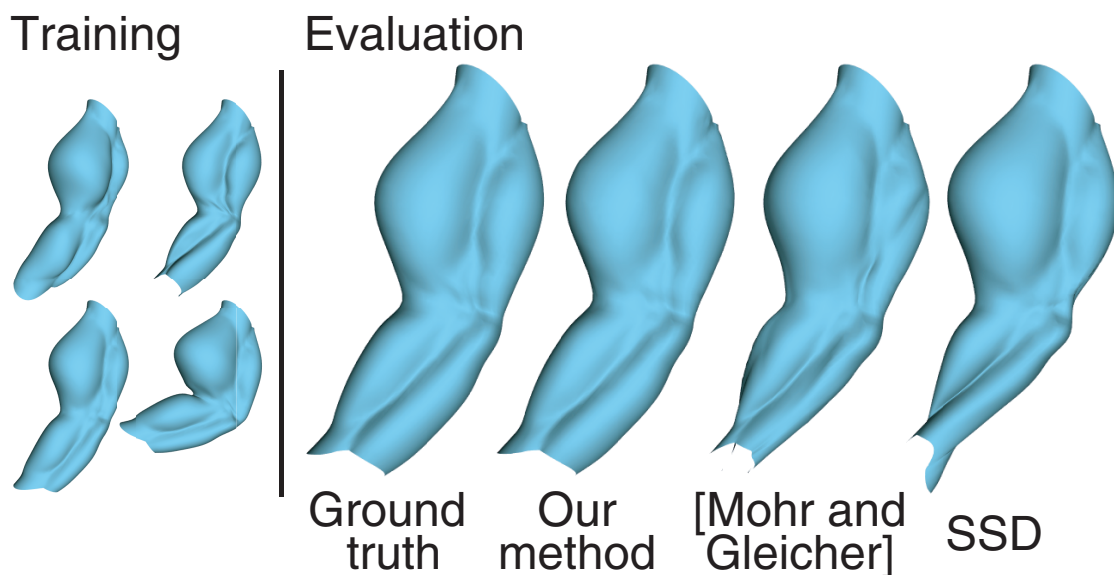


Figure 6-5: *Anatomical arm test.* We extracted a set of poses from an anatomically motivated arm rig with both bending and twisting at the elbow. The twisting and muscle bulges are enough to prevent SSD from approximating the examples well. The technique of Mohr and Gleicher [21] does better, but there are still differences. Our model produces a result almost indistinguishable from the ground truth.

Chapter 7

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. Fine wrinkles and sharp creases found in intricate skin and cloth may be better handled by a displacement interpolation technique. Extending our technique with a displacement correction model would also allow it to approximate training poses exactly, an important feature to animators. To properly handle cloth, dynamics and collisions need to be addressed. An exciting avenue of future work is to find an analog to our rotation model for dynamics.

Bibliography

- [1] Brett Allen, Brian Curless, Zoran Popović, and Aaron Hertzmann. Learning a correlated model of identity and pose-dependent body shape variation for real-time synthesis. In *2006 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 147–156, September 2006.
- [2] Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. Scape: shape completion and animation of people. *ACM Transactions on Graphics*, 24(3):408–416, August 2005.
- [3] Steve Capell, Matthew Burkhart, Brian Curless, Tom Duchamp, and Zoran Popović. Physically based rigging for deformable characters. In *2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 301–310, July 2005.
- [4] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. Interactive skeleton-driven dynamic deformations. *ACM Transactions on Graphics*, 21(3):586–593, July 2002.
- [5] cgCharacter. Absolute character tools 1.6, 2003. <http://www.cgcharacter.com/>.
- [6] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. Variational shape approximation. *ACM Transactions on Graphics*, 23(3):905–914, August 2004.
- [7] Michael Comet. Cmusclesystem 1.31, 2006. <http://www.cometdigital.com/>.
- [8] Kevin G. Der, Robert W. Sumner, and Jovan Popović. Inverse kinematics for reduced deformable models. *ACM Transactions on Graphics*, 25(3):1174–1179, July 2006.

- [9] James R. Diebel, Sebastian Thrun, and Michael Brünig. A bayesian method for probable surface reconstruction and decimation. *ACM Trans. Graph.*, 25(1):39–59, 2006.
- [10] D. W. Eggert, A. Lorusso, and R. B. Fisher. Estimating 3-d rigid body transformations: a comparison of four major algorithms. *Mach. Vision Appl.*, 9(5-6):272–290, 1997.
- [11] Zheng Guo and Kok Cheong Wong. Skinning With Deformable Chunks. *Computer Graphics Forum*, 24(3):373–381, 2005.
- [12] Shawn Hargreaves. Deferred shading. In *Proceedings of the Game Developers Conference*, March 2004.
- [13] Dae-Eun Hyun, Seung-Hyun Yoon, Jung-Woo Chang, Joon-Kyung Seong, Myung-Soo Kim, and Bert Jüttler. Sweep-based human deformation. *The Visual Computer*, 21(8-10):542–550, 2005.
- [14] Doug L. James and Christopher D. Twigg. Skinning mesh animations. *ACM Trans. Graph.*, 24(3):399–407, 2005.
- [15] Z. Karni and Craig Gotsman. Efficient compression of soft-body animation sequences. *Computer And Graphics*, 28(1):25–34, 2004.
- [16] Ladislav Kavan and Jiří Žára. Spherical blend skinning: a real-time deformation of articulated models. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 9–16, New York, NY, USA, 2005. ACM Press.
- [17] Paul G. Kry, Doug L. James, and Dinesh K. Pai. Eigenskin: real time large deformation character skinning in hardware. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 153–159, New York, NY, USA, 2002. ACM Press.
- [18] Matt Lee. Seven ways to skin a mesh: Character skinning revisited for modern gpus. In *Proceedings of GameFest, Microsoft Game Technology Conference*, August 2006.

- [19] J. P. Lewis, Matt Corder, and Nickson Fong. 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, pages 165–172, July 2000.
- [20] Bruce Merry, Patrick Marais, and James Gain. Animation space: A truly linear framework for character animation. *ACM Transactions on Graphics*, 25(4):1400–1423, October 2006.
- [21] Alex Mohr and Michael Gleicher. Building efficient, accurate character skins from examples. *ACM Trans. Graph.*, 22(3):562–568, 2003.
- [22] Taehyun Rhee, J.P. Lewis, and Ulrich Neumann. Real-time weighted pose-space deformation on the gpu. *Computer Graphics Forum*, 25(3):439–448, September 2006.
- [23] Ferdi Scheepers, Richard E. Parent, Wayne E. Carlson, and Stephen F. May. Anatomy-based modeling of the human musculature. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, pages 163–172, August 1997.
- [24] Peter-Pike J. Sloan, III Charles F. Rose, and Michael F. Cohen. Shape by example. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 135–143, New York, NY, USA, 2001. ACM Press.
- [25] Robert W. Sumner and Jovan Popović. Deformation transfer for triangle meshes. *ACM Trans. Graph.*, 23(3):399–405, 2004.
- [26] Robert W. Sumner, Matthias Zwicker, Craig Gotsman, and Jovan Popović. Mesh-based inverse kinematics. *ACM Trans. Graph.*, 24(3):488–495, 2005.
- [27] Joseph Teran, Eftychios Sifakis, Geoffrey Irving, and Ronald Fedkiw. Robust quasi-static finite elements and flesh simulation. In *2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 181–190, July 2005.

- [28] Xiaohuan Corina Wang and Cary Phillips. Multi-weight enveloping: Least-squares approximation techniques for skin animation. In *ACM SIGGRAPH Symposium on Computer Animation*, pages 129–138, July 2002.
- [29] Jason Weber. Run-time skin deformation. In *Proceedings of Game Developers Conference*, 2000.
- [30] Jane Wilhelms and Allen Van Gelder. Anatomically based modeling. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, pages 173–180, August 1997.

Appendix A

Reduced Poisson Formulation Details

Let $(k, j) \in F_0$ be the set of edges where neither $y_{k,j}$ nor $y_{k,1}$ are fixed.

Let $(k, j) \in F_1$ be the set of edges where only $y_{k,1}$ is fixed.

Let $(k, j) \in F_2$ be the set of edges where only $y_{k,j}$ is fixed. Then,

$$\begin{aligned}
 A \in \mathbb{R}^{12P \times 12P} &= \sum_{(k,j) \in F_0} (\Phi_{k,j} - \Phi_{k,1})^T (\Phi_{k,j} - \Phi_{k,1}) + \\
 &\quad \sum_{(k,j) \in F_1} \Phi_{k,j}^T \Phi_{k,j} + \sum_{(k,j) \in F_2} \Phi_{k,1}^T \Phi_{k,1} \\
 B_{1\ell} \in \mathbb{R}^{12P \times 9} &= \sum_{(k,j) \in F_0} (\Phi_{k,j} - \Phi_{k,1})^T \beta_{k\ell} (\hat{\mathbf{v}}_{k,j}^T \otimes I_{3 \times 3}) \\
 &\quad + \sum_{(k,j) \in F_1} (\Phi_{k,j})^T \beta_{k\ell} (\hat{\mathbf{v}}_{k,j}^T \otimes I_{3 \times 3}) \\
 &\quad + \sum_{(k,j) \in F_2} (-\Phi_{k,1})^T \beta_{k\ell} (\hat{\mathbf{v}}_{k,j}^T \otimes I_{3 \times 3})
 \end{aligned} \tag{A.1}$$

where \otimes denotes the Kronecker product.

$$\begin{aligned}
 B_1 \in \mathbb{R}^{12P \times 9P} &= [B_{11} \dots B_{1P}] \\
 B_2 \in \mathbb{R}^{12P \times 12J} &= \sum_{(k,j) \in F_1} (\Phi_{k,j})^T \Psi_{k,1} + \sum_{(k,j) \in F_2} (\Phi_{k,1})^T \Psi_{k,j} \\
 C_1 \in \mathbb{R}^{12P \times 9P} &= A^{-1} B_1; \quad C_2 \in \mathbb{R}^{12P \times 12J} = A^{-1} B_2
 \end{aligned}$$