Building Efficient, Accurate Character Skins from Examples

Alex Mohr * Michael Gleicher *

University of Wisconsin, Madison

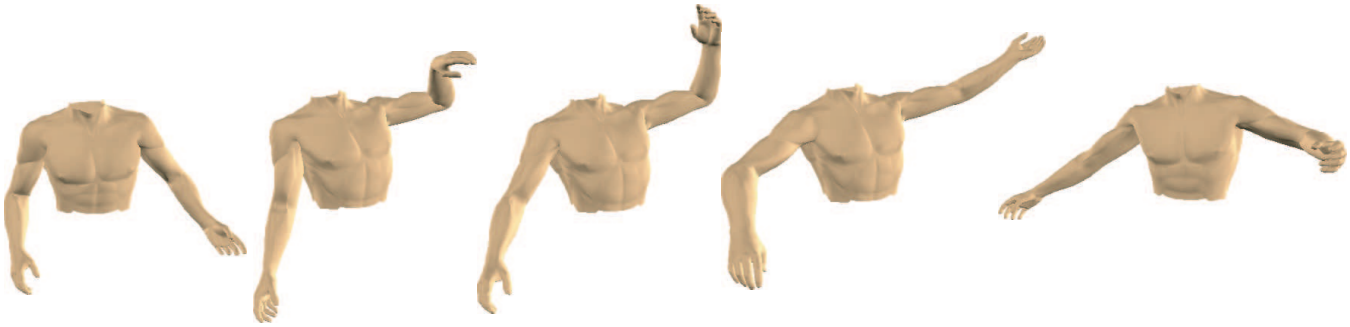

Figure 1: Several still frames of our character from Figure 9 in a new animation sequence.

Abstract

Good character animation requires convincing skin deformations including subtleties and details like muscle bulges. Such effects are typically created in commercial animation packages which provide very general and powerful tools. While these systems are convenient and flexible for artists, the generality often leads to characters that are slow to compute or that require a substantial amount of memory and thus cannot be used in interactive systems. Instead, interactive systems restrict artists to a specific character deformation model which is fast and memory efficient but is notoriously difficult to author and can suffer from many deformation artifacts. This paper presents an automated framework that allows character artists to use the full complement of tools in high-end systems to create characters for interactive systems. Our method starts with an arbitrarily rigged character in an animation system. A set of examples is exported, consisting of skeleton configurations paired with the deformed geometry as static meshes. Using these examples, we fit the parameters of a deformation model that best approximates the original data yet remains fast to compute and compact in memory.

Keywords: Interactive, Skin, Approximation

1 Introduction

To be believable, animated characters must deform in plausible ways as they move. It is possible to accomplish this by having an artist sculpt an entire character mesh by hand for every frame of an animation sequence, but this is impractical. Instead, animators typically manipulate an underlying hierarchical skeleton. The character mesh geometry must then be attached to the underlying skeleton so that as the skeleton deforms, the mesh also deforms appropriately. This attachment of model geometry to an underlying skeleton is called a "skin" and can be viewed as a function that maps from the skeletal parameters to a deformation field.

There are two fundamental aspects of skin creation—*authoring* and *computation*. Skin authoring refers to how artists use tool sets to describe the behavior of skin geometry as the skeleton moves. Skin computation refers to the method by which the deformed mesh geometry is evaluated for display at some skeleton configuration. For high-end applications, the authoring methods drive skin creation while for interactive systems, computation methods dominate.

For high-end applications such as film, the visual fidelity of characters is paramount, so artists require flexibility and control in skin authoring. Hence, there are many different ways to create characters using commercial tools. One technique involves modeling skin substructure such as muscles and tendons to drive the skin geometry [Wilhelms and Gelder 1997; Scheepers et al. 1997]. Many deformers which drive skins by linking their control points to the skeletal parameters with custom expressions or scripts are also available. Some examples include FFD lattices [Sederberg and Parry 1986] or Wires [Singh and Fiume 1998]. High-end characters often use a combination of these techniques—different tools are appropriate for different parts of the character. This generality and control means that the computation aspect of high-end characters is highly customizable, tightly coupled to authoring, and potentially unbounded. In fact, high-end tools allow authors to continually develop new skin computation models through custom scripts, expressions and complex deformers.

In contrast, interactive systems require fast computation and small memory size for characters. Thus, the character computation model is fixed and artists must restrict their tool set to author characters in direct support of it. The most common skin computation model in games and interactive systems goes by many names including SSD, enveloping, smooth skinning, and linear blend skinning. This technique assigns a set of influencing joints and blending weights to each vertex in the character. The skin is computed by transforming each vertex by a weighted combination of the joints' local coordinate frames. This technique is discussed in more detail in Section 3. While fast to evaluate and compact in memory, this method is notorious not only for its authoring difficulty, but

* {amohr, gleicher}@cs.wisc.edu, http://www.cs.wisc.edu/graphics

also for its undesirable deformation artifacts. However, this method is widely used since these characters can be used with arbitrary amounts of animation data and can be posed at runtime.

A different character computation mechanism previously used in interactive systems is called mesh animation. Mesh animation works by storing a large number of deformed models as static meshes—one for each frame of animation. These static models are then either displayed directly or are linearly interpolated at runtime. Mesh animation is interesting since it decouples skin authoring from runtime skin computation, allowing artists to use any tools they want to author characters. Unfortunately mesh animation is only appropriate when the required animation sequences are short and are known *a priori*. As games and interactive applications use larger amounts of animation, storing every frame becomes prohibitive. This technique is also incapable of generating new poses at runtime; for example, to place the character's hand exactly on a door knob or to make footfalls land precisely on stairs. Due to these limitations, mesh animation is losing popularity.

In this paper, we present an automated method to build character skins that are fast to compute and compactly represented from a set of examples. This technique allows artists to use any skin authoring tools they like while producing characters that meet the performance demands and work with the computation models used in interactive systems. We present a framework for extending linear blend skinning that allows us to capture these detailed skin deformations. We show how we can fit the parameters of our skinning model using a sampling of an arbitrarily rigged character's deformations.

The rest of this paper is organized as follows. After a review of related work, we describe the simple linear blend skinning model and its strengths and weaknesses. The limitations of this approach lead us to a discussion of our framework for extending this skinning model. Next we describe how we fit the parameters of our skinning model using a sampling of the original character deformations. Finally we present results and applications of our technique.

1.1 System Overview

Building a skin with our system involves two major steps. We begin with a character rigged in an animation package such as Maya. We then sample this character's skin deformations by exporting the character's geometry in several poses. Next we fit the parameters of our underlying skinning model using this sampled data.

We wish to obtain a good sampling of the character's skin deformations to fit our underlying model with. To do this, we pose the character to exercise all the joints fully and include its extreme poses. This step does not require a trained animator since these poses are only intended to exercise the degrees of freedom of the character and need not correspond to a realistic motion. Once this is done, the poses are sampled regularly at k times. This sampling can be very simple to obtain from the user's perspective—in our case, users must simply invoke a script we have implemented in Maya. Each sample consists of the skeleton configuration and the corresponding deformed skin geometry as a static mesh. We call a paired skeleton configuration and static mesh an *example*.

Using this set of examples, our system first determines the set of joints that should influence each vertex, and then solves a bilinear least-squares problem to fit the parameters of the underlying skinning model. As mentioned earlier, the skinning model we use is an extension of the standard linear blend skinning model. Our extension adds extra joints to the character that are simply related to the existing joints. These new joints are designed in such a way to capture richer deformations than the standard linear blend skinning model. Our system is configured to add these extra joints automatically to characters, but we allow users to fine tune the specific set of extra joints if they wish.

2 Related Work

Character skin deformations are fundamental to character animation and have been addressed for some time in the literature. Catmull [1972] introduced one of the first skeleton-driven techniques—rigid skinning to a hierarchically structured articulated figure. A 2D skeletal bilinear deformation method was presented by Burtnyk and Wein [1976]. An early 3D skeleton-driven technique that went beyond rigid skinning was presented by Magnenat-Thalmann, et al. [1988]. Their technique used custom programmed algorithms to deform character meshes based on the nature of particular joints.

More recently, novel skinning methods that start with a simple skin and use sparse data interpolation to correct errors between it and a set of examples have been introduced. Three examples, Pose Space Deformation, Shape by Example, and EigenSkin [Lewis et al. 2000; Sloan et al. 2001; Kry et al. 2002] use radial basis interpolation of corrections to linear blend skins. Another recent work applies these techniques to range scan data [Allen et al. 2002]. These techniques are similar to ours in that they take examples as input. The results of these approaches are quite good, and unlike our technique, they can handle skin deformations that depend on abstract parameters rather than only skeleton configurations. However, these methods are not appropriate for interactive characters since they require storing potentially large amounts of example data for runtime interpolation. In contrast, our method discards all example data after the fitting process so the size of our runtime structures does not scale with the number of inputs.

Other authors have used physical simulation for interactive deformations, especially secondary animation [James and Pai 2002; Capell et al. 2002]. Our method cannot capture these secondary deformations directly; however, a technique such as DyRT [James and Pai 2002] can be applied to the characters we generate to add secondary animation.

There has been some recent work on fitting skinning models. One method solves for joint centers and vertex weights for a scanned arm [Nebel and Sibiryakov 2002] but the Multi-Weight Enveloping technique [Wang and Phillips 2002], or MWE, is most similar to our approach. MWE extends linear blend skinning by giving each vertex one weight to each coefficient of each influencing joint's transformation matrix instead of one weight per influencing joint. They then find these weights by solving a linear least-squares problem using a set of examples as input. While on the surface Multi-Weight Enveloping and our technique seem very similar, they are in fact different in a fundamental way. Both MWE and our technique use an extension of linear blend skinning as an underlying deformation model. However, MWE extends linear blend skinning by adding more *vertex weights* to the model while in contrast, our method adds more *joints*.

MWE uses a large number of weights per vertex (12 per influencing joint). This introduces the possibility of rank deficient matrices in the least-squares solutions [Wang and Phillips 2002], especially since the matrix coefficients are usually highly correlated. This can lead to overfitting, which MWE must take measures to avoid. In contrast, since the number of weights per vertex in one of our skins remains relatively small (1 per influencing joint) and our extra joints are explicitly designed to be very different from existing joints, our technique requires no special provisions to avoid overfitting. Even so, our method can detect and handle small amounts of overfitting if it occurs as explained in Section 5.2. Another consequence of having one weight per entry in the joint transformation matrices is that MWE skins are not as easily accelerated by graphics hardware as skins created using our method. Finally, since our skins are computed in the same manner as linear blend skins, existing software infrastructure can make use of them with little or no changes.

3 Linear Blend Skinning

The traditional interactive skinning model goes by many names. Lewis et. al call it Skeleton Subspace Deformation or SSD, Maya calls it "smooth skinning" and we call it linear blend skinning. This technique is widely used for interactive applications. An excellent description of this method is found in Lewis et al. [2000].

The linear blend skinning algorithm works by first placing a hierarchical skeleton inside a static model of a character, typically in some neutral pose. This initial character pose is referred to as "dress pose". Then, each vertex is assigned a set of influencing joints and a blending weight for each influence. Computing the deformation in some pose involves rigidly transforming each dress pose vertex by all of its influencing joints. Then the blending weights are used to combine these rigidly transformed positions. The deformed vertex position at some skeletal configuration c, $\bar{\mathbf{v}}_c$ is computed as

$$\bar{\mathbf{v}}_c = \sum_{i=1}^{n} w_i M_{i,c} M_{i,d}^{-1} \mathbf{v}_d \qquad (1)$$

where w_i are the weights (usually affine or convex), $\mathbf{v}_d$ is the dress-pose location of some vertex $\mathbf{v}$, $M_{i,c}$ is the transformation matrix associated with the ith joint in configuration c and $M_{i,d}^{-1}$ is the inverse of the dress-pose matrix associated with the ith influence. (Taken together, $M_{i,d}^{-1}\mathbf{v}_d$ represents the location of $\mathbf{v}_d$ in the local coordinate frame of the ith influence.) Note that a deformed vertex position in the dress pose configuration $c = d$ is the same as the provided dress pose vertex ($\bar{\mathbf{v}}_d = \mathbf{v}_d$) if the weights are affine.

This skinning algorithm is notorious for its failings. It cannot represent complex deformations and suffers from characteristic artifacts such as the "candy-wrapper" collapse effect on wrists and collapsing around bending joints as shown in Figure 2. The artifacts occur because vertices are transformed by linearly interpolated matrices. If the interpolated matrices are dissimilar as in a rotation of nearly 180 degrees, the interpolated transformation is degenerate, so the geometry must collapse. In addition to these deformation problems, linear blend skins are very difficult to author [Lewis et al. 2000].

Despite its failings, this skinning algorithm is very fast and widely supported by commercial applications so it remains popular especially in games and virtual environments.

4 Extending Linear Blend Skinning

The linear blend skinning model is not sufficient to capture deformations well as shown in Figure 3. The problem in this particular case is that as the twist approaches 180 degrees, the linearly blended matrix becomes degenerate and collapses the skin geometry. Linearly blended transformations tend to collapse the more different they are. The resulting loss of volume can also be observed around hinge joints such as the knee and elbow as shown in Figure 2.

We observe that we can help avoid the collapse problem by avoiding blending transformations that are so dissimilar. We can accomplish this by adding extra transformations that properly interpolates without collapsing. In the case of the twisting wrists, we can add an extra joint that interpolates the rotation angle correctly and does not collapse. In fact, artists sometimes do this by hand to help avoid wrist collapses.

More generally, we observe that *any* deformation effect could be obtained by adding joints that deform appropriately to capture that deformation effect. For example, to capture muscle bulges, we can add joints that scale up when the muscle should bulge, and scale down when the muscle relaxes. For wrinkles, we could add several joints that move and scale in concert to capture the wrinkles. In fact, in the limit we could add as many transformations as vertices and capture all deformations exactly.

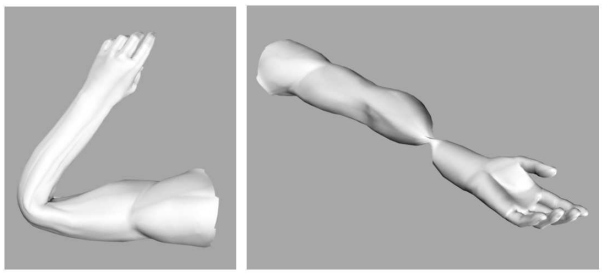

Figure 2: Common problems with linear blend skinning: the bent arm on the left demonstrates shrinkage around bent joints such as the elbow and knee while the twisted wrist on the right demonstrates the "candy-wrapper" collapse effect. These artifacts are caused by blending dissimilar transformations.

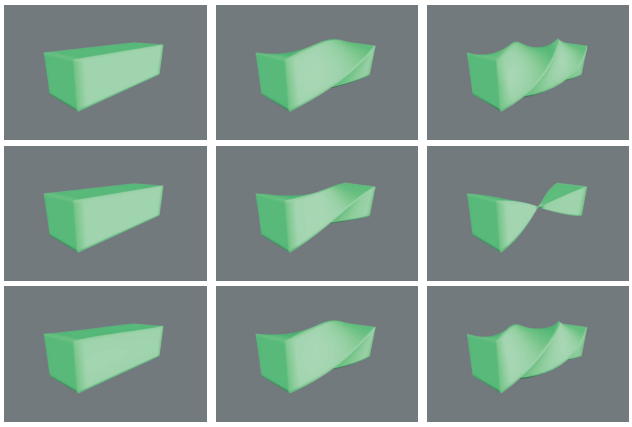

Figure 3: Top Row: Three examples of a twisting box driven by a nonlinear deformer. Middle Row: Solved linear blend skin using one joint. Bottom Row: Our result with just one additional joint that half interpolates the twist rotation.

Unfortunately, adding so many extra joints is impractical. First, adding such a large number of joints would severely impact the performance of our resulting skins. Worse, even if we could find these transformations for the input examples, it is unclear how to determine the general relationships of these transformations to the skeletal parameters in all poses. Without knowledge of this relationship, our scheme would only be able to reproduce the input frames and would not work well in new poses.

Instead, we extend the traditional linear blend skinning model by adding a relatively small number of joints that are simply related to the original skeletal parameters and fit using them. We choose these extra joints by both examining the places where the standard linear blend model fails and by examining extra character deformations that we would like to capture. We then add joints that we believe will help resolve these artifacts. Finally, we fit the parameters of our skinning model using this extended skeleton. The key to our success is that since vertices choose weighted sums of transformations, if *any linear scaling* of an added joint is beneficial it may be used. Thus the additional joints need not be exact.

We emphasize that this is a framework for obtaining better deformations and the joints we choose to add are based on our observations of characters. Different characters with different deformations may require a different set of additional joints. However, once some set of these joints is determined, the skin may be solved using our fitting algorithm without change.

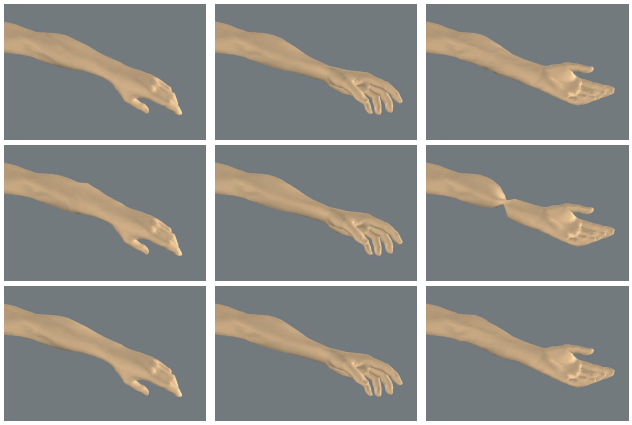

Figure 4: Top Row: Original examples of a twisting wrist. Middle Row: Linear blend skin approximation. Bottom Row: Our result using one additional joint.

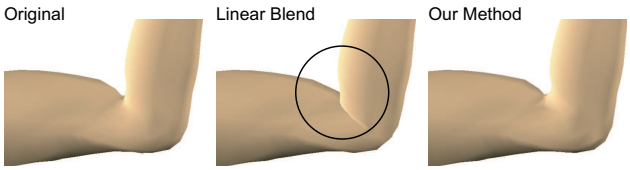

Figure 5: Linear blend skinning alone is incapable of capturing correct creasing around elbows. At the left is an example of a bent elbow. In the middle is the linear blend skin approximation. notice the interpenetration. In contrast, our method avoids the interpenetration.

4.1 Additional Joints

To help solve the collapsing geometry problem, our system can automatically add joints that properly interpolate rotations without collapsing. This is done by examining the rotation of a joint relative to the dress pose and computing the new joint as the halfway spherical linear interpolation [Shoemake 1985] of this rotation, located at the same position in space. More joints with evenly distributed interpolation parameters could be added to sample this rotation space even better; however, in our experience just a single interpolated rotation is sufficient.

Figure 4 demonstrates the improvements gained by simply adding a single interpolated rotation joint in the twisting case. Figure 5 shows the improvements for the bent elbow case.

Another type of effect not easily captured by the simple linear blend model is bulging and denting of skins caused by muscles, tendons, or other substructure. These particular effects cannot be captured since the joints employed in animating a character do not typically scale up and down as would be necessary to approximate these effects.

We have observed that for many characters, the substructure deformation effects from muscles and tendons are often simply related to the angles between joints. For example, a bicep bulge is small when the elbow is near full extension while the bugle is large when the elbow is near full flexion. The effect is similar for other muscles in the body. To capture these effects, our system can add several joints that scale up and down based on the angle between particular joints.

We add these scaling joints as follows. First we choose a joint in the original skeleton that will drive the scaling parameters of the new joints. Once this driver is chosen, there are two sets of joints that we add. The first set is "upstream" of the driver and lies in the middle of the bone connecting the driver to its parent, the second set is "downstream" and lies in the middle of the bones connecting the driver to its children.

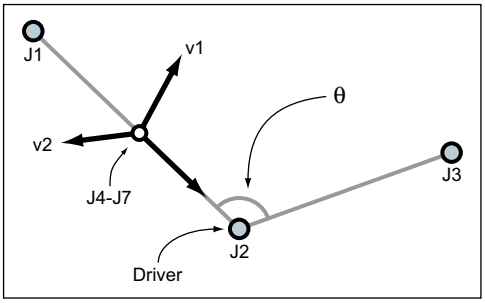

Figure 6: Our method adds extra joints to characters to help better approximate deformations. Here $J4$ through $J7$ are automatically added upstream joints that scale depending on the angle θ. As θ decreases, $J4$ scales up in the direction $v1$ which is orthogonal to the bone connecting $J1$ and $J2$. Meanwhile, $J5$ scales up in the direction $v2$, orthogonal to both the bone and $v1$. $J6$ and $J7$ operate similarly, but scale down as θ increases rather than up. Downstream joints are very similar except that these joints are positioned on the bone from $J2$ to $J3$.

All upstream joints are oriented in the same way, with one axis aligned with the bone as shown in Figure 6. We use four upstream joints. Two of them scale up about two axes orthogonal to the bone and a corresponding pair scale down about the two axes orthogonal to the bone. The scale parameters of these joints are set based on the angle of the bone connecting the driver to its parent and the bone connecting the driver to its child. If the driver has multiple children, a vector that is the sum of the bones connecting the driver to its children is used to measure the angle. Downstream joints are similar. We use four downstream joints on each bone connecting the driver to its children that scale just as the upstream joints do.

The scale parameters are computed as follows. For joints that scale up, the scale parameter s is

$$s = 1 + \frac{k}{2}\left(\frac{\mathbf{b}_1 \cdot \mathbf{b}_2}{\|\mathbf{b}_1\|\,\|\mathbf{b}_2\|} + 1\right)$$

where $\mathbf{b}_1$ and $\mathbf{b}_2$ are the bone vectors used to measure the angle at the driver joint and k is the maximum scale factor when the angle between $\mathbf{b}_1$ and $\mathbf{b}_2$ is zero. For joints that scale down, the scale parameter is simply s^{-1}. The value for k may be chosen by the user but in our experience, we have found that 8 works well for our examples. Again, since vertices may take any scaling of these new joints, a conservative large value is fine. For example, if a vertex in fact needed a joint that scaled by 2 instead of 8, it could be assigned a weight of $\frac{1}{4}$.

5 Fitting the Skinning Model

Once our system has augmented the input skeleton, we use a fitting procedure to set the parameters of the underlying skinning model to match the example data well.

As mentioned earlier, the input to the fitting process is a set of examples. An example is simply a static character mesh paired with a skeleton. This static mesh is deformed according to the skeleton configuration, but it is not attached to the skeleton in any way. For our results, our examples were generated by exporting rigged objects from Maya, but they could have been sculpted by hand or come from another program.

A linear blend skin computes a deformed vertex as described earlier in Equation 1. Examining this skinning model, only the M_i are predetermined. These are the coordinate frames associated with all the joints in the character. That means for each vertex, we are able to choose the set of influencing joints, influence weights (w_i) and the dress pose vertex position ($\mathbf{v}_d$). We would like to choose the influence sets, weights and dress pose vertex positions that best approximate the examples and generalize well to new poses.

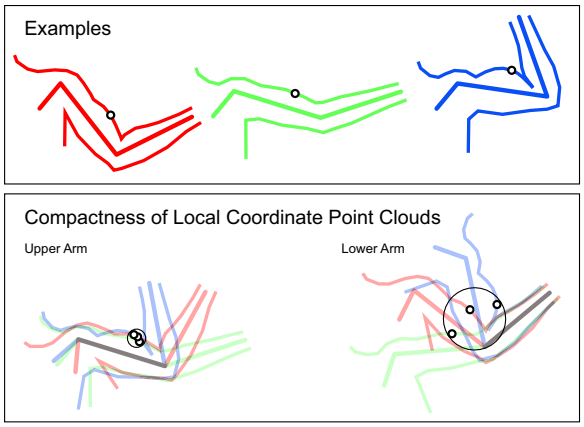

Figure 7: Top: A set of three examples of a deforming arm mesh with a bulging bicep. A particular point on the arm mesh is highlighted in each example. Bottom Left: Each example rotated and aligned so that the upper arm bones coincide. The highlighted points form a cloud in the local coordinate frame of the upper arm. Even though the bicep bulges significantly, this cloud is compact. Lower Right: A similar point cloud but relative to the forearm. This cloud is far less compact than the former, making the forearm a poorer choice for an influence.

5.1 Finding Influence Sets

We determine influence sets first for several reasons. Ideally, the influence sets would fall out naturally from the weight solving procedure (irrelevant joints would have a weight of zero) but this does not happen in practice because our samplings are necessarily not exhaustive. Also, the more joints that a vertex depends on, the slower the skin can be to compute and current hardware only supports a limited number of influences per vertex. Thus, we would like to select a small set of good influences. Also, choosing the influence sets appropriately lets us bound the size of the problems we must solve to determine the weights as discussed in Section 5.2. This makes the solving process faster.

In most recent research, influence set determination has been left to users [Lewis et al. 2000; Wang and Phillips 2002; Sloan et al. 2001]. The task is typically accomplished by "painting" the regions of influence for each joint over the mesh. While less difficult than painting the weights themselves [Lewis et al. 2000], it is a tedious process. In contrast, our system automatically determines the influence sets for each vertex using a heuristic algorithm.

We observe that vertices in a character skin typically transform nearly rigidly with respect to some joint. For instance, vertices on the forearm roughly follow the forearm. We believe that for most characters, their skin is most heavily influenced by those joints that they are bound to. Even though a point on the bicep is not truly rigid as an arm moves (due to muscle bulge), we believe that these points remain mostly rigidly attached to the upper arm, and therefore should be influenced by it. Using this observation, we measure how rigidly a vertex transforms with every joint over all examples and use the most rigidly transforming joints for the influence set.

For a single vertex, a rigidity score for a joint is computed as follows. For each example, the local coordinate position of the vertex is computed as $M_{i,e}^{-1}\mathbf{v}_e$ where $M_{i,e}$ is the coordinate frame associated with the ith joint in the eth example and $\mathbf{v}_e$ is the global coordinate position of the vertex on the eth example. The collection of these local coordinate positions over all examples forms a point cloud as shown in Figure 7. The more compact this point cloud, the more rigid we believe the vertex-joint relationship to be. We measure the compactness of this point cloud by taking its diameter (the maximum distance between any two points in the cloud). We have found that the simple $O(n^2)$ algorithm that compares each point to every other to be fast enough for our purposes but this diameter may be computed more quickly. An $O(n \log n)$ time algorithm is possible. See [Malandain and Boissonnat 2002] for faster methods.

Once the compactness measures for all joints are computed for a vertex, the smallest k are chosen as the influence set for that vertex. It may be tempting to use a threshold scheme to choose influence sets but we have found this problematic. It is unclear how to pick a good threshold because as the rigidity scores get larger, they become less meaningful. For instance, it may happen as an artifact of the particular input examples that points on the left shoulder move much more rigidly relative to the right leg rather than the left leg but both choices make no sense for influences. Since larger rigidity scores are not particularly meaningful, it is nearly impossible to pick a meaningful threshold value.

As in other linear blend skinning systems, influence sets need only be determined conservatively [Wang and Phillips 2002] so we allow users to choose k if desired. In our experience, we have found that between three and eight influences works well, depending on the complexity of the character.

5.2 Solving for Weights and Vertices

Once the influence sets have been determined, only the weights and dress pose vertex positions remain (w_i and $\mathbf{v}_d$). We would like to find the best vertices and weights that minimize the least-squares difference between the skin and the examples at all the example skeleton configurations. That is

$$\min \left\| \sum_{i=1}^{n} \bar{\mathbf{v}}_{e_i} - \mathbf{v}_{e_i} \right\|^2$$

for all examples where $\mathbf{v}_{e_i}$ is the input vertex position from the ith example and $\bar{\mathbf{v}}_{e_i}$ is the deformed vertex computed by the skinning model at the ith example configuration.

$$\bar{\mathbf{v}}_e = \sum_{i=1}^{n} w_i M_{i,e} M_{i,d}^{-1} \mathbf{v}_d$$

This problem is bilinear in the weights and vertices. We use an alternation technique to solve the optimization. This works by first fixing the first variable and solving a linear least-squares problem to find the second, then fixing the second and solving a linear least-squares problem for the first. This process is then repeated until it converges. This technique is commonly used and is described in [Freeman and Tenenbaum 1997]. We start by solving for weights since we have no good guess for them but we know that the initial dress pose vertices are ideal. Next we hold the weights fixed and solve for vertex positions. This process typically converges after one or two iterations.

As mentioned in Section 2, we have found that since we are solving for a small numbers of weights using large numbers of examples, our systems are often well conditioned and do not suffer from overfitting if the input data is well sampled. Thus we do not have to take special precautions to avoid overfitting as in [Wang and Phillips 2002], although we include tests for robustness.

For clarity, we present the matrices we solve via least-squares in block form. First we introduce some notation: $T_{i,e} = M_{i,e} M_{i,d}^{-1}$. In order to ensure that the resulting weights are affine, we set $w_1 = 1 - \sum_{i=2}^{n} w_i$, and solve for w_2 through w_n.

$$\begin{bmatrix} (T_{2,e_1} - T_{1,e_1})\mathbf{v}_d & \cdots & (T_{n,e_1} - T_{1,e_1})\mathbf{v}_d \\ \vdots & \ddots & \vdots \\ (T_{2,e_k} - T_{1,e_k})\mathbf{v}_d & \cdots & (T_{n,e_k} - T_{1,e_k})\mathbf{v}_d \end{bmatrix} \begin{bmatrix} w_2 \\ w_3 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} \mathbf{v}_{e_1} - T_{1,e_1}\mathbf{v}_d \\ \vdots \\ \mathbf{v}_{e_k} - T_{1,e_k}\mathbf{v}_d \end{bmatrix}$$

The matrix used to solve for vertex positions is as follows.

$$\begin{bmatrix} \sum_{i=1}^{n} w_i T_{i,e_1} \\ \vdots \\ \sum_{i=1}^{n} w_i T_{i,e_k} \end{bmatrix} \begin{bmatrix} \mathbf{v}_d \end{bmatrix} = \begin{bmatrix} \mathbf{v}_{e_1} \\ \vdots \\ \mathbf{v}_{e_k} \end{bmatrix}$$

To handle homogeneous coordinates, the translation parts of the $\sum_{i=1}^{n} w_i T_{i,e_k}$ matrices are subtracted from the $\mathbf{v}_e$ on the right hand side.

We solve these least-squares problems using the singular value decomposition. This lets us detect when our matrices are rank deficient, leading to overfitting. We detect this by comparing the ratio of the largest singular value to the smallest, and issuing a warning if there are any singular values below some fraction of this ratio. To recover, we zero these singular values and continue with the fitting process. If overfitting is a problem, provisions such as those taken in [Wang and Phillips 2002] could also be used. However, in all the examples in this paper, no singular values were zeroed.

5.3 Handling Normals

It is not only important for the geometry in a skin approximation to be accurate, but also important for normals to be well approximated. If they are not, lighting calculations will not produce good results. We assume that normals are specified per vertex. It may seem that just transforming a dress pose normal by the inverse transpose of the corresponding vertex's transformation matrix would be correct. To be more explicit,

$$\bar{\mathbf{n}}_c = \frac{(\sum w_i M_{i,c} M_{i,d}^{-1})^{-T} \mathbf{n}_d}{\left\| (\sum w_i M_{i,c} M_{i,d}^{-1})^{-T} \mathbf{n}_d \right\|}$$

While this is technically valid for local neighborhoods of smooth surfaces [Turkowski 1990], we do not have a smooth surface. Instead we have single points that are computed independently. Computing the normals in this manner can give undesirable results when the blended transformations are not pure rotations.

Interactive systems typically approximate normal calculations as

$$\bar{\mathbf{n}}_c = \frac{\sum w_i M_{i,c}^{-T} M_{i,d}^{T} \mathbf{n}_d}{\left\| \sum w_i M_{i,c}^{-T} M_{i,d}^{T} \mathbf{n}_d \right\|} \tag{2}$$

since it is often faster to calculate the joint matrices and their inverses incrementally by composition because at each step, rotations and scales have special inverse forms. This alleviates the need for a general inversion operation. In EigenSkin [Kry et al. 2002], normals are treated as second skinning problem and are computed independently. In our system, we take the model used in existing systems as in Equation 2 and include normals in our optimization process. To do this, we simply add more terms to the objective function to include the differences between normal vectors. We allow users to scale normals if they wish to change their relative influence on the least-squares solution.

6 Results

The simple linear blend skinning model commonly used in video games and other interactive applications is very fast and compact but cannot capture the high quality deformations that make convincing characters. Our framework for extending the linear blend model allows us to capture much more interesting deformations while retaining its efficiency.

The most egregious deformation problems of linear blend skinning are solved by our approach. Figures 3 and 4 show how our system can fix collapsing twists by adding just a single extra joint. Collapsing and interpenetrations around hinge joints are also fixed using our method as shown in Figure 5.

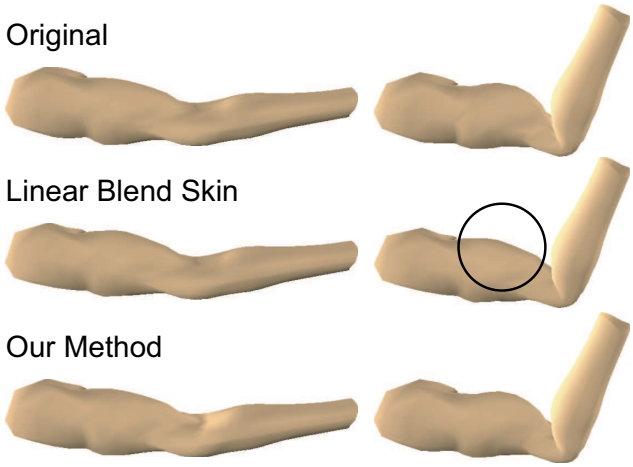

Figure 8: Top: Examples of a muscular arm flexing. Middle: Linear blend skin approximation. Note the lack of bicep bulge. Bottom: Results using our method.

In addition to solving these problems with linear blend skinning, our extension framework can capture other more subtle and detailed deformations required for convincing characters. Figure 8 demonstrates how our method can capture the bulges in the biceps and triceps of a character's arm. While the particular extra joints we have chosen to add to our characters may not be capable of capturing the full deformation for any character, different extra joints that do capture the desired deformations may be added and solved using our technique.

To demonstrate that our technique can be used on more than just simple arms and legs, Figure 9 shows a rigged upper body and its approximation by our system. This figure also shows this character in new poses from an animation sequence, demonstrating that our resulting skins generalize well to new poses.

Our solution procedure is generally very fast. None of the examples shown here took more than five minutes to solve on a modern personal computer. The slowest was the upper body model which has more than 6000 vertices, 50 examples, and 5 influences per vertex. The computation time for each vertex depends on the number of influences and the number of examples. Also, since each vertex is solved independently, our algorithm is trivial to parallelize.

6.1 Applications

The ability to generate compactly represented, fast to evaluate, high quality skin approximations from a set of examples is very useful. Applications range from building characters for video games and virtual environments to high-end animation previewing.

Many current interactive systems such as video games only support linear blend skinned characters. Aside from the deformation problems associated with using this model, authoring these skins is notoriously difficult. Determining the blending weights and influence sets is left to the skin author to set directly. None of the more intuitive or useful deformer primitives provided by animation systems may be used.

Using our method, character authors may use any tools they like to author characters. All our system requires is a set of examples which is used to compute the appropriate influence sets and blending weights automatically. This frees the author from setting them manually. It is important to note that since our characters are a straightforward extension to linear blend skinning, many existing interactive systems already have the software infrastructure to sup-

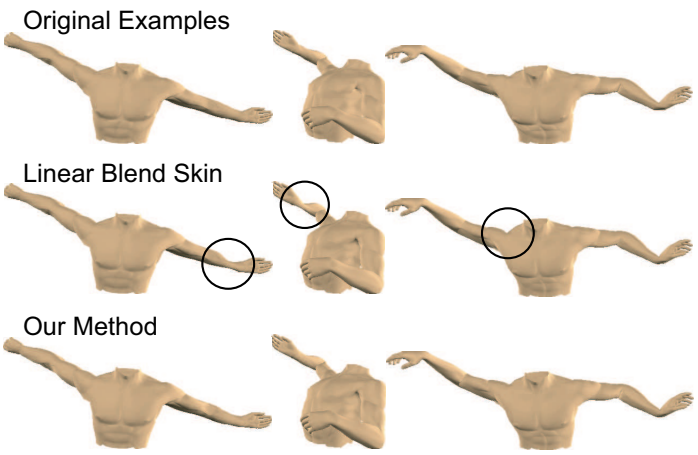

Figure 9: Top Row: Examples of an upper body rigged in Maya. Middle Row: Best linear blend skin. Note the circled problem areas. Bottom Row: Our results.

port them. In addition, since our skins are computed in the same manner as existing linear blend skins, they are already accelerated by current graphics hardware.

Another application of our system is to map a character originally attached to one skeleton onto a different underlying skeleton. We call this process *skin retargeting*. Skin retargeting is useful if a particular interactive system requires characters to have a specific skeleton. For instance, a video game may have an optimized engine for characters with a particular skeleton topology. Ordinarily, if a character was created for a different skeleton, the character would have to be re-rigged manually to work on the new skeleton topology. However, this can be accomplished much more easily with our system. One just exports a set of example meshes deformed by the original skeleton but paired with corresponding poses of the new skeleton. Our system sees this as any other set of data and solves for the proper influence sets and blending weights.

Another application of our technique is targeted at high-end animation. High-end characters often have such complex deformations that they cannot be computed interactively. Thus, animators typically work with low fidelity versions that only roughly suggest the actual shape of the character. Using our method, interactive characters could be built that allow animators to interact with much better approximations of the deformed characters.

6.2 Discussion

In this paper, we have presented a method for building fast to evaluate, compact representations that produce accurate approximations of deforming characters. The characters may be rigged using any available tool since our system only requires static deformed meshes paired with skeletal configurations as input.

While our technique works well for a wide variety of character skins, it has limitations. For instance, character deformations in our model are only driven by the skeleton's joint parameters. Our method cannot capture deformations that are driven by abstract parameters such as "happiness" as in [Lewis et al. 2000; Sloan et al. 2001]. Our system also cannot accurately reproduce deformations that are not representable as linear combinations of the transformations expressed in our skeletons. For instance, the scaling joints presented in this paper can only fully capture deformations that are well approximated by a scaling that is linearly related to the cosine of the angle between two bones. This assumption may be violated by a character whose muscle bulges only when its arm is fully bent. The scaling joints also assume that only the angle between joints is important, so bending the shoulder forward is treated the same as bending it up. Even though not all deformations can be captured using the extra joints presented here, new joints may be added to capture any important deformation, and our influence set and vertex weight solving framework may be applied without change.

Despite these limitations, our method produces high-quality yet fast and compact skinned characters that work with existing game engines, graphics hardware and other runtime systems.

Acknowledgments

We would like to thank Luke Tokheim for help with rigging and J. P. Lewis and Karan Singh for their generosity with characters. We would also like to thank the UW Graphics Group, especially Andrew Selle and Hyun Joon Shin for their help with video production. This research is supported by NSF grants CCR-9984506 and CCR-0204372, and equipment donations from Intel.

References

ALLEN, B., CURLESS, B., AND POPOVIĆ, Z. 2002. Articulated body deformation from range scan data. *ACM Transactions on Graphics 21*, 3 (July), 612–619.

BURTNYK, N., AND WEIN, M. 1976. Interactive skeleton techniques for enhancing motion dynamics in key frame animation. *CACM 19*, 10, 564–569.

CAPELL, S., GREEN, S., CURLESS, B., DUCHAMP, T., AND POPOVIĆ, Z. 2002. Interactive skeleton-driven dynamic deformations. *ACM Transactions on Graphics 21*, 3, 586–593.

CATMULL, E. E. 1972. A system for computer generated movies. In *Proc. ACM Annual Conf.* August, 422–431.

FREEMAN, W. T., AND TENENBAUM, J. B. 1997. Learning bilinear models for two-factor problems in vision. In *IEEE Computer Vision and Pattern Recognition*.

JAMES, D. L., AND PAI, D. K. 2002. DyRT: Dynamic response textures for real time deformation simulation with graphics hardware. *ACM Transactions on Graphics 21*, 3, 582–585.

KRY, P. G., JAMES, D. L., AND PAI, D. K. 2002. Eigenskin: Real time large deformation character skinning in hardware. In *ACM SIGGRAPH Symposium on Computer Animation*, 153–160.

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*, Annual Conference Series, ACM SIGGRAPH.

MAGNENAT-THALMANN, N., LAPERRIRE, R., AND THALMANN, D. 1988. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings of Graphics Interface '88*, 26–33.

MALANDAIN, G., AND BOISSONNAT, J.-D. 2002. Computing the diameter of a point set. In *Discrete Geometry for Computer Imagery (DGCI 2002)*, A. Braquelaire, J.-O. Lachaud, and A. Vialard, Eds., vol. 2301.

NEBEL, J.-C., AND SIBIRYAKOV, A. 2002. Range flow from stereo-temporal matching: application to skinning. In *IASTED International Conference on Visualization, Imaging, and Image Processing*.

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*, Annual Conference Series, ACM SIGGRAPH, 163–172.

SEDERBERG, T. W., AND PARRY, S. R. 1986. Free-form deformation of solid geometric models. In *Proceedings of SIGGRAPH 86*, Annual Conference Series, ACM SIGGRAPH, 151–160.

SHOEMAKE, K. 1985. Animating rotation with quaternion curves. In *Proceedings of SIGGRAPH 85*, Annual Conference Series, ACM SIGGRAPH, 245–254.

SINGH, K., AND FIUME, E. L. 1998. Wires: A geometric deformation technique. In *Proceedings of SIGGRAPH 98*, Annual Conference Series, ACM SIGGRAPH, 405–414.

SLOAN, P.-P. J., CHARLES F. ROSE, I., AND COHEN, M. F. 2001. Shape by example. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, 135–143.

TURKOWSKI, K. 1990. Transformations of surface normal vectors. Tech. Rep. 22, Apple Computer, July.

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

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