Stretchable and Twistable Bones for Skeletal Shape Deformation

Alec Jacobson Olga Sorkine
New York University & ETH Zurich

Figure 1: *Left to right: the* Beast *model is rigged to a skeleton in its rest pose. The neck is stretched and the arms are twisted and stretched using linear blend skinning. LBS relies solely on per-bone scalar weight functions, resulting in the explosion of the head and hands. The candy-wrapper artifact of LBS is also noticeable at the elbows. The dual quaternion skinning (DQS) solution [Kavan et al. 2008] correctly blends rotations, avoiding the candy-wrapper artifact, but reliance on bone weights alone unnaturally concentrates the twisting near the elbows. DQS also does not alleviate the stretching artifacts. Our solution, stretchable, twistable bones skinning (STBS), uses an extra set of weights per bone, allowing stretching without explosions and smooth twisting along the entire length of each arm.*

Abstract

Skeleton-based linear blend skinning (LBS) remains the most popular method for real-time character deformation and animation. The key to its success is its simple implementation and fast execution. However, in addition to the well-studied elbow-collapse and candy-wrapper artifacts, the space of deformations possible with LBS is inherently limited. In particular, blending with only a scalar weight function per bone prohibits properly handling stretching, where bones change length, and twisting, where the shape rotates along the length of the bone. We present a simple modification of the LBS formulation that enables stretching and twisting without changing the existing skeleton rig or bone weights. Our method needs only an extra scalar weight function per bone, which can be painted manually or computed automatically. The resulting formulation significantly enriches the space of possible deformations while only increasing storage and computation costs by constant factors.

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

Keywords: shape deformation, articulated character animation, linear blend skinning

Links: DL PDF WEB VIDEO

1 Introduction

Skinning and skeletal deformation remain standard for character animation because the associated deformation metaphor is directly intuitive for many situations: most characters are creatures or humans who ought to behave as if a skeleton was moving underneath their skin. The motion capture pipeline, for example, explicitly relies on this metaphor to build a subspace representation of human motion [Anguelov et al. 2005]. At the cost of performance, some applications demand physical accuracy, ensuring preservation of volume or simulating muscles [Teran et al. 2005]. Other applications, such as video games, crowd simulation and interactive animation editing, cannot afford to compromise real-time performance, so they trade accuracy for speed and often adopt the simplest and most efficient implementation of skeletal deformation.

The long-standing standard real-time skeletal deformation method is linear blend skinning (LBS), also known as skeletal subspace deformation or enveloping [Magnenat-Thalmann et al. 1988; Lewis et al. 2000]. In a typical workflow, a trained rigging artist manually constructs and fits a skeleton of rigid bones within the target shape. The skeleton is bound to the shape by assigning a set of correspondence weights for each bone, a process which can be tedious and labor-intensive. To deform the shape, animators assign transformations to each skeleton bone, either directly or with the assistance of an inverse kinematics engine or motion capture data. These transformations are propagated to the shape by blending them linearly as matrix operations according to the bone weights.

Linearly blending matrix transformations with scalar weight functions has a number of limitations. Many improvements of LBS focus on the problems arising from linearly blending rotations as matrices, which results in shape collapses near joints. Multi-weight enveloping (MWE) [Wang and Phillips 2002; Merry et al. 2006] and Dual Quaternions [Kavan et al. 2008] have been proposed as alternative rotation blending methods. However, a different set of limitations arises from the fact that using only a single scalar weight function per bone limits the space of possible deformations. We show that neither LBS nor its improvements properly handle stretching, where bones change length, nor twisting, where the skin

twists along the length of a bone, as in the human forearm (see Fig. 1).

Our goal is to expand the space of deformations possible with skinning to include stretching and twisting. We achieve this by using an additional set of weights for each bone. We call them *endpoint weights* as they reveal correspondences between the shape and the endpoints of each bone. These additional weights allow us to modify standard skinning formulas — in particular, LBS and dual quaternion skinning (DQS) — to explicitly expose stretching and twisting of bones. We hence term our method "stretchable, twistable bones skinning" (STBS). Deformation computation remains embarrassingly parallel, and the involved extra storage and computation costs are minimal. Unlike other methods that employ additional weights, our endpoint weights have a clear and intuitive geometric meaning and thus may be painted manually. Alternatively, capitalizing on recent methods like [Baran and Popović 2007] that build skeletons and compute bone weights automatically, our endpoint weights may similarly be computed automatically, keeping pipelines fully automatic if desired.

Many artists will be hesitant to change to a new skinning scheme. Our solution complements the typical skinning environment without changing the rigid skeleton metaphor or interfering with existing controls: if bones are not stretched or twisted, the deformation remains the same as defined by the underlying skinning method. We also take advantage of existing skinning rigs, allowing users to opt in without modifying their existing skeletons or bone weights.

Problem context. Skeletal skinning with bone weights works well because bones as deformation handles properly capture the natural rigidity of body parts. Linearly blending bone transformations via bone weight functions efficiently expresses rigidity along bones during bending, packing smooth transitions near joints, where the weights briefly overlap [Magnenat-Thalmann et al. 1988]. Unfortunately, bone weights that produce natural bending are insufficient for producing plausible stretching and twisting, because they are poor at controlling the subspace *along* the bone.

Works like [Wang and Phillips 2002; Merry et al. 2006] improve LBS by supplying additional weights per bone. These extra weights are additional degrees of freedom which can alleviate joint collapse, and perform twisting better. However, these additional weights do not retain an immediate or intuitive geometric meaning, since they essentially correspond to individual transformation matrix entries. As a result, they cannot be easily painted or adjusted manually, but only computed automatically via fitting example poses of the target shape. Often such example poses do not exist or are difficult to design, making these improvements challenging to apply in practice.

Even with the reliance on example poses aside, the deformation formulations of multi-weight enveloping methods do not sufficiently expand the space of deformations to capture stretching. Consider for example a single bone within a cigar-like shape, as in Fig. 2. In order to maintain good properties such as reproduction of the identity transformation and translational and rotational invariance, the bone's LBS weights and any of the extra weights of [Wang and Phillips 2002; Merry et al. 2006] must equal 1 everywhere on the shape. This means that if the bone changes its length, the only choice is to scale the whole shape by the same transformation, resulting in "explosion" past either endpoint.

To overcome LBS artifacts, riggers often manually subdivide bones or add special anatomically incorrect bones. Painting weights for these special bones is difficult because their intuitive meaning is less clear. The method of [Mohr and Gleicher 2003] uses example poses to determine such extra bones and their weights automatically. With enough extra bones with proper weights, twisting and stretching can be achieved, but at the sacrifice of an anatomically meaningful skeleton.

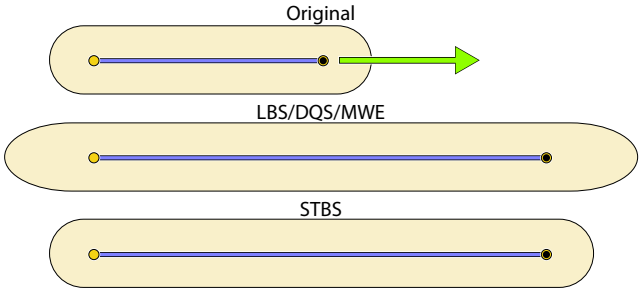

Figure 2: *A single bone controls a cigar shape (top). With only one bone, the weights of LBS and [Wang and Phillips 2002; Merry et al. 2006] must equal 1 everywhere. When the bone is stretched, these methods must scale the entire shape, resulting in explosion past the bone's endpoints (center). While the bone weights in our method must also be 1 everywhere, the endpoint weights are allowed to vary, such that proper stretching is achievable (bottom).*

Other works such as [Forstmann and Ohya 2006; Yang et al. 2006; Forstmann et al. 2007] use curve or spline skeletons to cope with LBS artifacts. These methods share a similar foundation as our method, but focus on fixing joint collapse and do not explicitly treat stretching. To define correspondences between their curved skeleton "bones" and points on the shape, they rely on inverse Euclidean distance schemes which ignore the geometry of the shape being animated. The new rigging tools and controls needed for curved skeletons are inconsistent with the existing rigging pipeline [Kavan et al. 2008].

Instead of relying on additional weights or alternative rigging metaphors, Kavan et al. [2008] directly solve the joint collapse and candy-wrapper artifacts by blending rigid bone transformations as dual quaternions rather than matrices, leaving the skeleton and bone weights metaphor untouched. However, DQS still relies on a single set of bone weights, so stretching remains unsolved and twisting must still be concentrated near joints (see Fig. 3).

Recent works in 2D and 3D have successfully demonstrated the flexibility of point handles with associated weight functions (see e.g. [Langer and Seidel 2008; Jacobson et al. 2011]). Point weights by construction vary over the shape more than bone weights, and typically, much of the shape is significantly affected by two or more points. This means they are unsuitable for bending limbs rigidly, but properly capture stretching (see Fig. 4). Many works on shape editing and deformation advocate the point handle metaphor (see e.g. [Igarashi et al. 2005] or the survey in [Botsch and Sorkine 2008]), albeit at much higher computational costs of surface deformation due to the involved global optimizations.

Contributions. Our contribution is to combine the notions of point weights and bone weights, allocating each to the tasks they do well, while maintaining the standard skeleton skinning framework. In our technique, bone weights continue to enforce rigidity and control smooth bending. In addition, we introduce point weights at bone endpoints to enable proper stretching and twisting. Since our goal is orthogonal to fixing the rotation blending artifacts of LBS, our method complements techniques like DQS and works with any underlying skinning method. We show that our method extends the space of deformations available for skinning in a useful way, demonstrate 2D and 3D examples of bending, stretching and twisting, and discuss several options for obtaining the required endpoint weight functions.

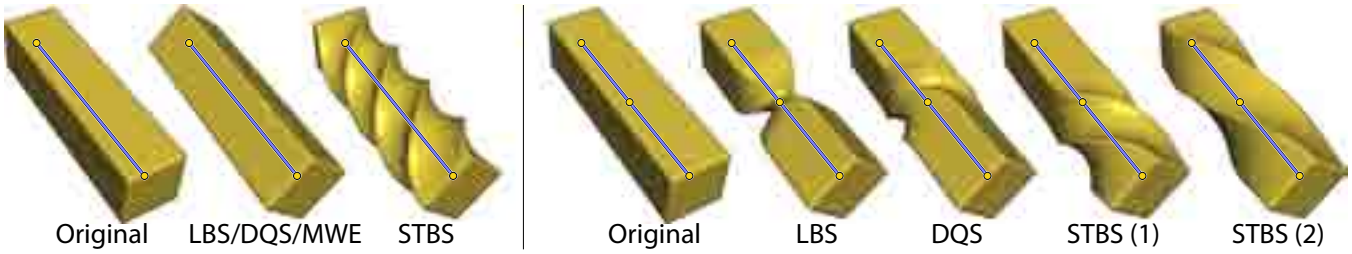

Figure 3: *Left: A single bone controls a box in 3D. The bone's weights in LBS, DQS, and MWE must equal 1 everywhere. If the bone twists about its axis by 315 degrees, these methods must twist the entire shape uniformly, i.e. simply rigidly rotate it. The bone weights in our method must also be 1 everywhere, but the endpoint weights are allowed to vary, so interesting twisting may be spread over the entire shape. Right: Two bones control a box in 3D. In order to bend properly, the bone weights in an LBS or DQS rig must only overlap close to the joints. As a result these methods must pack interesting twisting near joints. LBS linearly blends rotation matrices, resulting in the candy-wrapper effect. DQS corrects this artifact by blending rotations as quaternions, but twisting is still concentrated near the joint. Our method keeps the same bone weights, but our extra endpoint weights enable twisting to be spread across the length of one bone or both bones.*

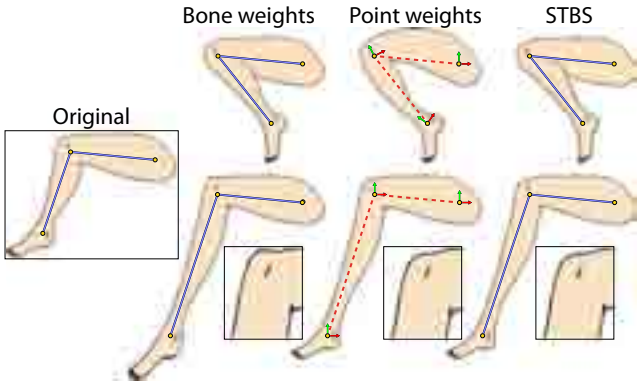

Figure 4: *Bones properly capture rigidity necessary to bend a leg (upper left), but stretching with bone weights explodes the foot and knee unintuitively (lower left). Point-handles properly treat stretching as blended translations (lower center), but blending rotations causes limbs to lose their rigidity (upper center). Our stretchable bones solution uses bone weights and point weights, allocating each to the tasks they do well. Our bones bend smoothly at joints (upper right) and stretch intuitively (lower right).*

2 Stretchable, twistable bones

Our goal is to derive a simple skinning equation, capable of deforming 2D and 3D shapes by a skeleton whose bones may stretch and (in 3D) twist. Let $\mathcal{S} \subset \mathbb{R}^d$ denote our target shape in dimension $d = 2, 3$. We denote the set of (possibly disjoint and unordered) bones in the skeleton by the line segments $B_i = \{(1-t)\mathbf{a}_i + t\mathbf{b}_i \mid t \in [0,1]\}, i = 1, ..., m$. We derive our skinning equation by first decomposing the basic linear blend skinning equation. Here, the user defines affine transformations T_i for each bone B_i. The new positions for all points $\mathbf{p} \in \mathcal{S}$ are computed as the weighted combinations:

$$\mathbf{p}' = \sum_{i=1}^{m} w_i(\mathbf{p})\, T_i \mathbf{p}, \tag{1}$$

where $w_i : \mathcal{S} \to \mathbb{R}$ is the scalar *bone weight function* associated with bone B_i. If the bone transformations T_i are rigid, they can be intuitively decomposed into translation and rotation parts, yielding:

$$\mathbf{p}' = \sum_{i=1}^{m} w_i(\mathbf{p}) \left\{\mathbf{a}_i' + R_i\left(-\mathbf{a}_i + \mathbf{p}\right)\right\}, \tag{2}$$

where R_i is the user-defined rotation that takes the bone B_i's rest vector $(\mathbf{b}_i - \mathbf{a}_i)$ to its pose vector $(\mathbf{b}_i' - \mathbf{a}_i')$. If the bones are allowed to change length then a scaling term is needed to make sure that the bone endpoints reach their pose positions. The decomposition becomes:

$$\mathbf{p}' = \sum_{i=1}^{m} w_i(\mathbf{p}) \left\{\mathbf{a}_i' + R_i\left(S_i\left(-\mathbf{a}_i + \mathbf{p}\right)\right)\right\}, \tag{3}$$

where S_i performs anisotropic scaling in the reference frame of B_i, namely, $S_i = X_i^{-1} A_i X_i$, where X_i rotates $(\mathbf{b}_i - \mathbf{a}_i)$ to the x-axis and A_i scales anisotropically along the x-axis by a factor of $\|\mathbf{b}_i' - \mathbf{a}_i'\| / \|\mathbf{b}_i - \mathbf{a}_i\|$.

Notice that for each bone B_i, S_i and R_i are constant over $\mathcal{S}$, so that there is no choice for each bone but to rotate and stretch all points uniformly. For stretching this means that if $\mathbf{p}$ lies *beyond* an endpoint of a bone, it will get overly stretched, as shown in Fig. 2. This effect is not removed even when multiple bones deform an area (e.g. around a joint), resulting in unwanted bulging (see Fig. 4). As for twisting, a bone twists all attached points around its axis rigidly (see Fig. 3), relying on the weighted average to blend twists from different bones. This effectively packs any interesting twisting near the joints where bone weights overlap.

The above problems are due to missing information: a point $\mathbf{p} \in \mathcal{S}$ does not "know" where on each bone B_i it is attached, i.e., it does not know its position relative to either of the bone's endpoints. This causes the excessive stretching (instead of localizing it to the bone area) and prevents gradual twisting along the bone.

We now insert this missing information in the form of *endpoint weight functions* $e_i(\mathbf{p})$ for each bone. These functions vary from 0 to 1 as $\mathbf{p}$'s correspondence shifts from endpoint $\mathbf{a}_i$ to $\mathbf{b}_i$. With these extra weight functions we have enough information to fix Eq. 3 to handle stretching and twisting correctly. First, we replace the scaling term S_i with a weighted translation along the bone direction:

$$\mathbf{p}' = \sum_{i=1}^{m} w_i(\mathbf{p}) \left\{\mathbf{a}_i' + R_i\left(e_i(\mathbf{p})\, \mathbf{s}_i + \left(-\mathbf{a}_i + \mathbf{p}\right)\right)\right\}, \tag{4}$$

where $\mathbf{s}_i = (\frac{\|\mathbf{b}_i' - \mathbf{a}_i'\|}{\|\mathbf{b}_i - \mathbf{a}_i\|} - 1)(\mathbf{b}_i - \mathbf{a}_i)$, the full stretch vector at $\mathbf{b}_i$.

To allow twisting along bones, we insert an additional rotation term:

$$\mathbf{p}' = \sum_{i=1}^{m} w_i(\mathbf{p}) \{\mathbf{a}_i' + R_i\, K_i\left(e_i(\mathbf{p})\right)\left(e_i(\mathbf{p})\, \mathbf{s}_i + \left(-\mathbf{a}_i + \mathbf{p}\right)\right)\}, \tag{5}$$

where $K_i(t)$ is the twisting rotation about the axis $(\mathbf{b}_i - \mathbf{a}_i)$ by angle $(1-t)\theta_{\mathbf{a}_i} + t\theta_{\mathbf{b}_i}$. The angles $\theta_{\mathbf{a}_i}$ and $\theta_{\mathbf{b}_i}$ are the user-defined twists at the endpoints $\mathbf{a}_i$ and $\mathbf{b}_i$, respectively. The new

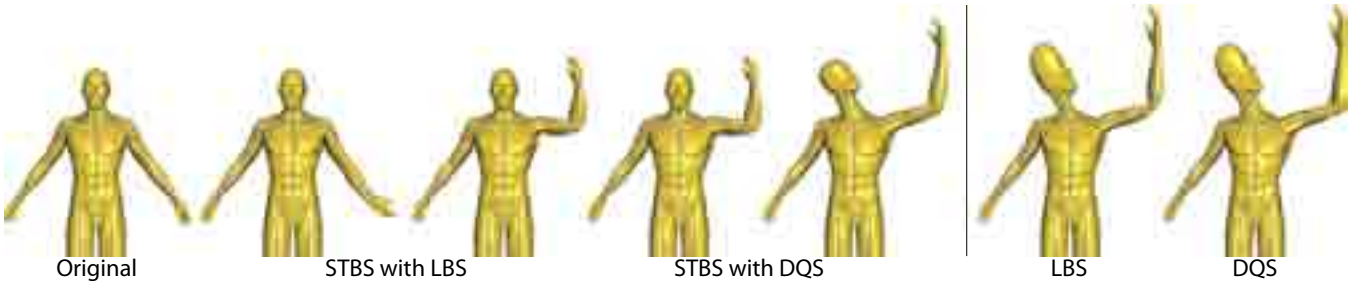

Figure 5: *Left to right: A human model is rigged to a skeleton using bounded biharmonic bone and endpoint weights. Its arm is twisted by 180 degrees, spreading the twist along the length of the upper and lower arm. In its twisted state, the arm is bent at the elbow. Joint collapse artifacts are corrected by switching to DQS as the underlying skinning formulation. Finally, the neck and arm are stretched; notice that the head and hand do not explode. In contrast, using LBS results in joint collapse, isolated twisting and shape explosion. DQS prevents joint collapse, but twisting is still packed near joints and proper stretching is not achieved.*

rotation $K_i(e_i(\mathbf{p}))$ is a function of $\mathbf{p}$ and thus is not constant over $\mathcal{S}$, enabling interesting twisting along each bone. Notice that if bone B_i is not stretched or twisted at its endpoints, its contribution is the same as in the original skinning equation (1).

2.1 Dual-quaternion skinning

Since we are blending rigid transformations, by applying the distributive property, Eq. 5 may be simplified into a deformation equation that consists of a single rotation and translation per bone:

$$\mathbf{p}' = \sum_{i=1}^{m} w_i(\mathbf{p}) \left\{ \mathcal{T}_i\left(e_i(\mathbf{p})\right) + \mathcal{R}_i\left(e_i(\mathbf{p})\right) \mathbf{p} \right\}. \tag{6}$$

Both the translations $\mathcal{T}_i$ and the rotations $\mathcal{R}_i$ are functions of the endpoint weight functions e_i, evaluated at $\mathbf{p}$, but they are constant w.r.t. the bone weight functions w_i. Thus far our skinning equation, like LBS, treats these translations and rotations as matrix operators and linearly combines them across bones as a sum of each matrix element weighted by the respective bone weights w_i.

Instead, we may blend $\mathcal{T}_i$ and $\mathcal{R}_i$ in their dual quaternion forms [Kavan et al. 2008]. As expected, DQS eliminates collapses near joints when bones are rotated. Combining DQS with our stretchable, twistable bones makes for a powerful and expressive skinning equation, as can be seen in Fig. 5.

2.2 Properties of good endpoint weights

For the stretchable, twistable bones deformation equation (5) to produce visually good deformations, care must be taken in defining both the bone weight functions w_i and the endpoint weight functions e_i. The desirable properties for a bone weight w_i are the same as in standard skinning (see e.g. [Jacobson et al. 2011] for a detailed discussion): the weight function should be shape-aware (i.e., dependent on the distance measured in the shape, as opposed to the ambient Euclidean space), should equal 1 on the rigid region corresponding to the bone and smoothly fall off toward 0, reaching exactly 0 on rigid parts corresponding to other bones; the weights should be bounded between 0 and 1, since negative weights lead to unintuitive "opposite" deformation effects and weights greater than 1 exaggerate the prescribed transformations; the bone weights should partition unity at all points on the shape.

The list of desirable properties for endpoint weights is similar to that of bone weights, namely they should vary smoothly on $\mathcal{S}$, be shape-aware, and bounded between 0 and 1. In addition, the Lagrange (interpolation) property must be fulfilled, to ensure that user-defined positions and twists applied at endpoints are met: $e_i(\mathbf{a}_i) = 0$ and $e_i(\mathbf{b}_i) = 1$. Furthermore, in 2D, since bones lie directly on the shape they control, the endpoint weight functions should provide linear interpolation along their bone segments, i.e., $e_i(\mathbf{p}(t)) = t$ for $\mathbf{p}(t) = (1-t)\mathbf{a}_i + t\mathbf{b}_i, t \in [0,1]$. In 3D, bones are typically inside the volume enclosed by the shape, so this requirement becomes a "convergence" requirement: endpoint weight functions should approach linear interpolation as the shape approaches the bones. Notice that (in 2D) linear interpolation combined with the boundedness property contradicts smoothness exactly at the bone endpoints, where e_i will only be C^0. The importance of linear interpolation over smoothness depends on the application. Note also in (5) that e_i is completely independent of the other endpoint and bone weights. Therefore, endpoint weights do not need to partition unity between themselves or with any of the bone weights.

2.3 Defining endpoint weights

Endpoint weights, like bone weights, may be painted manually. This is feasible because, unlike the extra weights of [Wang and Phillips 2002] and [Merry et al. 2006], our extra weights have a clear geometric meaning: for each bone they tell each point in the domain how much it should stretch and twist. The desired weight properties are intuitive and conceivably specifiable by a user assisted with sufficient 2D and 3D weight painting tools (e.g. Maya). As with LBS, both the bone weights w_i and the point weights e_i may be edited interactively, so a user can make changes to the weights and see the results immediately.

Strictly speaking, the endpoint weights of different bones are unrelated. However, we have found that it is sufficient to supply traditional point-based weight functions for each *joint* in the skeleton. Then the endpoint weights for each bone become a combination of that bone's incident joint weights:

$$e_i = \frac{1}{2}\left(\left(1 - j_{\mathbf{a}_i}\right) + j_{\mathbf{b}_i}\right), \tag{7}$$

where $j_{\mathbf{a}_i}$ and $j_{\mathbf{b}_i}$ are the weight functions at the joints incident on bone B_i at its respective endpoints $\mathbf{a}_i$ and $\mathbf{b}_i$.

Still, painting weights manually can be tedious and time-consuming. In the case that bone weights were assigned automatically, manually painting endpoint weights would disrupt a fully automatic rigging pipeline. Thus, we would like the option to define endpoint weights automatically.

There are many existing automatic methods for computing such weights. One simple method is to project each point $\mathbf{p}$ onto the nearest point of each bone, taking the fraction of where it falls between the bone's endpoints as its weight:

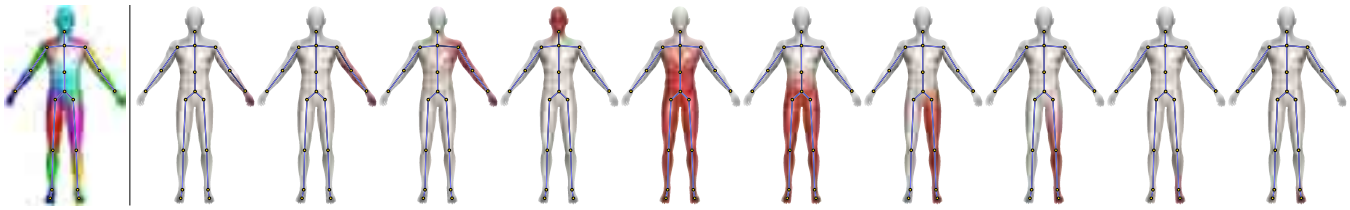

Figure 6: *The left image visualizes bone weights for a skeleton in a human model. Each bone is assigned a color. The color at a point on the shape is the weighted average of each bone's color according to the bone weights. The series of images on the right visualizes the endpoint weights of each bone. Red and white correspond to weights of 1 and 0 respectively. Both the bone weights and endpoint here were automatically computed using BBW.*

$$e_{\text{proj}\,i}(\mathbf{p}) = \frac{\|\text{proj}_i(\mathbf{p}) - \mathbf{a}_i\|}{\|\mathbf{b}_i - \mathbf{a}_i\|}, \tag{8}$$

where $\text{proj}_i(\mathbf{p})$ is the projection of point $\mathbf{p}$ onto B_i. These weights satisfy the boundedness and linear interpolation properties, but they largely ignore the shape of $\mathcal{S}$ and are only C^0 where they reach the values of 1 and 0. Variants of this type of weight function were used for the curved skeletons in [Forstmann and Ohya 2006; Yang et al. 2006; Forstmann et al. 2007] to associate a frame with each point $\mathbf{p}$ that corresponds to the curve's local frame.

Another immediate method would be to use inverse Euclidean distance weighting to define weights for the set of joint points:

$$j_{\text{IEDW}\,i}(\mathbf{p}) = \frac{1}{d_i(\mathbf{p})^\alpha}, \tag{9}$$

where $d_i(\mathbf{p})$ is the Euclidean distance from $\mathbf{p}$ to the rest position of joint i. While endpoint weights derived from these joint weights may be smooth (for $\alpha \geq 2$) and bounded, they again ignore the shape of $\mathcal{S}$, and their locality diminishes beyond the endpoints as the ratio of distances to the two endpoints regresses to 0.5.

The above methods are easy to implement and computationally lightweight. They are unsatisfactory to be used directly but serve well as initial guesses for manually painting the endpoint weights.

The automatic bone weights method presented in Section 4 of [Baran and Popović 2007], also known as Bone Heat (BH), may be trivially adapted to define joint weights (by considering a bone shrunk to a point). These weights require visibility computation and solving of large sparse linear systems of equations, but we have found that the quality of these weights in many cases justifies the computation costs. The methods of [Weber et al. 2007] and [Wang et al. 2007] may similarly be adapted to define joint weights. However, they only consider the surface of the shape rather than its volume, so the shape-awareness property is not fully achieved. Furthermore, these methods also require example poses, which often do not exist.

The bounded biharmonic weights w_{BBW} of [Jacobson et al. 2011] (BBW) minimize the Laplacian energy subject to the constraint $w_{\text{BBW}} \in [0, 1]$ (and additional interpolation constraints, as required). This method may be used to compute joint weights or endpoint weights directly. The weights have been shown to be smooth, localized and shape-aware. The Lagrange and linear interpolation properties are satisfied by imposing appropriate boundary conditions. These weights are the solution to a quadratic programming problem, which means they are somewhat slower to compute than the previous methods. Another limitation of this method is that it requires a volume discretization, which is often difficult to obtain for surfaces in 3D. The auxiliary interior vertices of the volume discretization increase the complexity of the precomputation. Nevertheless, we have found that when a volume discretization is available, these weights produce the highest quality results.

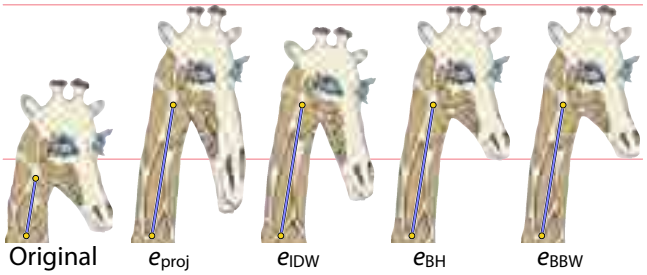

Figure 7: *Left to right: A single bone controls the* Giraffe*'s head and neck in 2D. Projecting onto the bone to obtain endpoint weights e_{proj} results in a nonsmooth and shape-*unaware *deformation. Endpoint weights e_{IDW} derived from inverse Euclidean distance joint weights are smooth, but shape-unaware and suffer from the fall-off effect (the top of the head sags too low). In this 2D example, BH endpoint weights e_{BH} and BBW endpoint weights e_{BBW} are virtually indistinguishable, producing appealing deformations.*

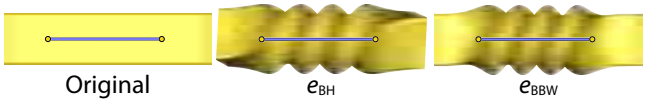

Figure 8: *A single bone controls a box in 3D (left). A 360 degree twist is applied to an endpoint using BH endpoint weights e_{BH} (middle). Notice the fall-off effect in the regions beyond each handle: the full twist is not reached. The same twist is applied using BBW endpoint weights e_{BBW} (right). These weights have proper locality, so the full twist is achieved.*

When computing both bone weights and endpoint weights from scratch, BBW may be preferred since the same implementation produces high quality weights for both bones and endpoints (see Fig. 6). When bone weights already exist, the additional endpoint weights obtained by BBW and by extending BH are both high quality. In many cases deformations using the endpoint weights obtained with these methods and the same bone weights are virtually identical (see Fig. 7 and Fig. 9). However, the extension of BH may suffer from the fall-off effect in regions beyond endpoints (see Fig. 8).

3 Implementation and results

We implemented stretchable, twistable bones skinning (STBS) using LBS and DQS as the underlying transformation blending mechanisms. Our implementation supports manually painting bone and endpoint weights or automatically computes them using [Baran and Popović 2007] (BH) or [Jacobson et al. 2011] (BBW). Our timings for computing endpoint weight functions are similar or faster than the bone weight computation times originally reported in those works. When using BH our precomputation time per endpoint

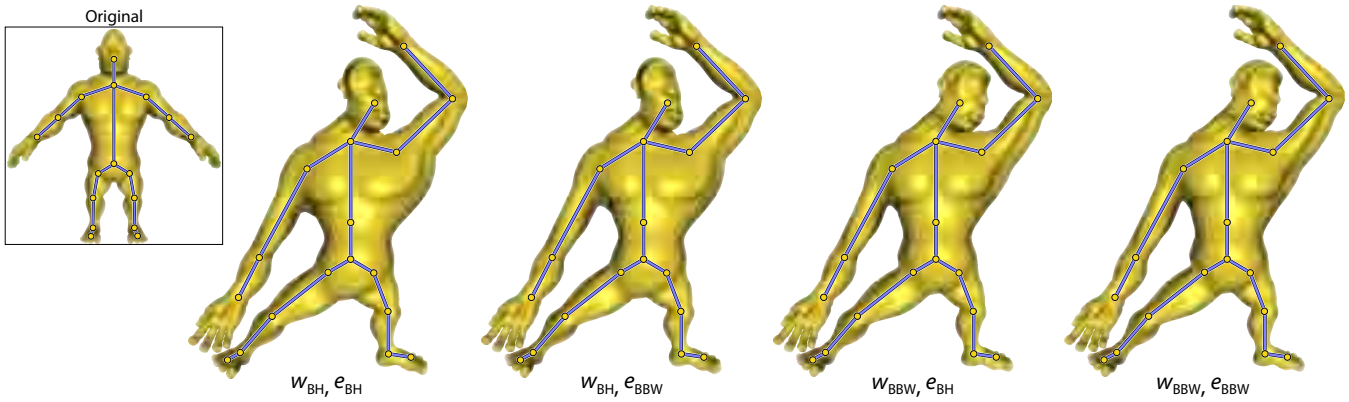

Figure 9: *Left to right: The* Ogre *(inset) is deformed with BH bone weights w_{BH} and BH endpoint weights e_{BH}. Switching to BBW endpoint weights e_{BBW} improves the deformation only slightly (e.g. see the shoulder). Changing endpoint weights cannot fix problems arising from insufficient bone weights. Instead, switching to BBW bone weights w_{BBW} noticeably improves the deformation around the head, shoulder and belly. With same bone weights, the endpoint weights of these methods produce visually similar results (comparing the first and final pairs).*

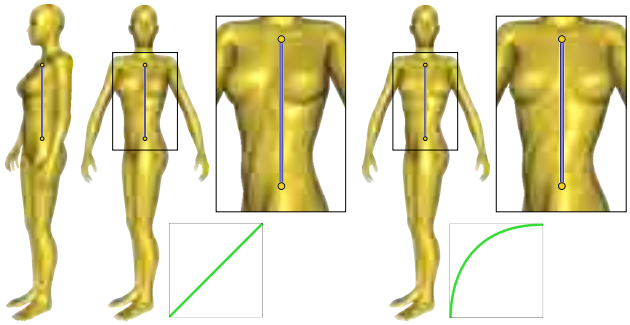

Figure 10: *A typical skinning rig must break the spine into many smaller bones in order to capture twisting. Here, a human model is twisted along the spine by applying twists at the top endpoint of a* single *bone. The endpoint weights smoothly blend the twist along the torso. They may be filtered through a spline curve to adjust their effect interactively. Filtered endpoint weights concentrate the twist in the abdomen, keeping the chest more rigid (right). Insets visualize the identity and user-defined spline filters, respectively.*

Figure 11: *A dog's yawn is exaggerated with stretchable bones. Refer to the accompanying video for an animation of this figure.*

weight function is on the order of milliseconds in 2D and seconds in 3D. When using BBW our precomputation time per endpoint weight function is less than a second in 2D and on the order of tens of seconds in 3D, where we use heavily graded tetrahedral meshes for the volume discretization. Endpoint weights can be computed faster than bone weights by taking advantage of the fact that we only care about a bone's endpoint weights in regions of the domain where the bone's bone weight function is greater than zero. In our examples, we limit the optimization to the portion of the domain with corresponding bone weights greater than 1e-7.

A vertex shader implementation of STBS does not differ much from that of LBS or DQS. It requires loading the additional endpoint weights into constant memory once per deformation session, and in 3D transferring the new per-endpoint twist parameters along with the usual bone transformations. Finally, the skinning equation is replaced with either (5) or the dual quaternion form of (6).

The automatic weights are precomputed before the user begins applying transformations to the bones. In terms of efficiency at deformation time, our method requires twice as many weights as LBS or DQS: it stores the same bone weights and the extra endpoint weights per bone. The original LBS or DQS require respective 8 or 12 shader operations per shape vertex per bone [Kavan et al. 2008]. Because the transformations in (6) are not constant over the shape, using STBS with LBS or DQS as the underlying blending method requires the extra operations need to build the transformation at each shape vertex. In our implementation we count this as an extra 12 operations per shape vertex per bone.

Twisting and stretching bones richly expands the space of possible deformations without forfeiting the rigid skeleton metaphor or modifying existing bones and bone weights. In Fig. 5, a skeleton rigged to a human properly twists, stretches and bends its arm using STBS. Blending transformations between bones as dual quaternions corrects rotational artifacts when bending at joints.

In 2D, the ability to stretch bones without worrying about "explosions" near endpoints enables real-time creation and playback of animated of images. Fig. 12 shows stretchable bones deforming a single image into a life-like series of poses.

Endpoint weights allow interesting twisting in 3D to occur over large regions of a shape controlled by a single bone. In Fig. 10, a human's entire torso is twisted smoothly using a single twistable bone. The endpoint weights in our system may be filtered interactively to alter their effect. The original weights, automatically computed using BBW, twist the human's chest too much. Using a spline filter, the endpoint weights are interactively adjusted so that the twist is concentrated in the abdomen.

Exaggeration is a cornerstone principle in animation. Stretchable bones facilitate stylized and exaggerated actions. In Fig. 11, stretching the bones in a dog's mouth emphasizes a yawning action, avoiding distracting shape explosion artifacts.

Figure 12: *With our method, bone joints may be freely dragged about by the user without worry that changes in bone length will cause explosion artifacts. Here an old photograph of Max Schmeling is brought to life.*

4 Conclusion

We have shown that with only slight modifications to existing skinning equations, we are able to expand the space of deformations possible with standard rigid skeleton rigs. Our method does not change the rigid skeleton metaphor, nor does it modify existing skeletons or bone weights. The additional endpoint weights required by our technique are feasibly painted by the user or computed using automatic point weight methods.

Though our skinning formula and extra weights expand the space of possible deformations, our method is still inherently limited by its simplicity. Like all skinning methods, STBS cannot prevent self-collisions, maintain global properties (e.g. total volume), minimize non-linear deformation energies, or respond to physically based forces. An obvious extension would be to use STBS as a reduced deformable model for more complicated methods.

In future work, we would like to explore further the role of extra weight functions. For example, the weights used to control stretching and twisting do not have to be the same. It would be interesting to expose these as separate parameters. We would also like to consider the orthogonal problem of specifying stretchable, twistable bone transformations with inverse kinematics or procedural animation. Our endpoint weights have a clear geometric meaning, such that their computation does not require example poses, but fitting to example poses could allow more accurate weights and enable skinning animations with stretchable, twisting bones as in, e.g. [James and Twigg 2005; Kavan et al. 2010]. Finally, it would be interesting to control advanced effects such as muscle bulging by applying simple filters to existing endpoint weights.

Acknowledgements

We are grateful to Ofir Weber and Ilya Baran for illuminating discussions, to the United States Library of Congress for its collection of public domain photographs including the half-portrait of Max Schmeling and to Felix Hornung for beautifying the teaser image. This work was supported by an SNF award 200021_137879.

References

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

BARAN, I., AND POPOVIĆ, J. 2007. Automatic rigging and animation of 3D characters. *ACM Trans. Graph. 26*, 3, 72:1–72:8.

BOTSCH, M., AND SORKINE, O. 2008. On linear variational surface deformation methods. *IEEE TVCG 14*, 1, 213–230.

FORSTMANN, S., AND OHYA, J. 2006. Fast skeletal animation by skinned arc-spline based deformation. In *Proc. Eurographics, short papers volume*.

FORSTMANN, S., OHYA, J., KROHN-GRIMBERGHE, A., AND MCDOUGALL, R. 2007. Deformation styles for spline-based skeletal animation. In *Proc. SCA*, 141–150.

IGARASHI, T., MOSCOVICH, T., AND HUGHES, J. F. 2005. As-rigid-as-possible shape manipulation. *ACM Trans. Graph. 24*.

JACOBSON, A., BARAN, I., POPOVIĆ, J., AND SORKINE, O. 2011. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph. 30*, 4.

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

KAVAN, L., COLLINS, S., ZARA, J., AND O'SULLIVAN, C. 2008. Geometric skinning with approximate dual quaternion blending. *ACM Trans. Graph. 27*, 4, 105:1–105:23.

KAVAN, L., SLOAN, P.-P., AND O'SULLIVAN, C. 2010. Fast and efficient skinning of animated meshes. *Comput. Graph. Forum 29*, 2, 327–336.

LANGER, T., AND SEIDEL, H.-P. 2008. Higher order barycentric coordinates. *Comput. Graph. Forum 27*, 2, 459–466.

LEWIS, J. P., CORDNER, M., AND FONG, N. 2000. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *Proc. SIGGRAPH*, 165–172.

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

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

MOHR, A., AND GLEICHER, M. 2003. Building efficient, accurate character skins from examples. *ACM Trans. Graph. 22*, 3.

TERAN, J., SIFAKIS, E., BLEMKER, S. S., NG-THOW-HING, V., LAU, C., AND FEDKIW, R. 2005. Creating and simulating skeletal muscle from the visible human data set. *IEEE TVCG 11*, 3, 317–328.

WANG, X. C., AND PHILLIPS, C. 2002. Multi-weight enveloping: least-squares approximation techniques for skin animation. In *Proc. SCA*, 129–138.

WANG, R. Y., PULLI, K., AND POPOVIĆ, J. 2007. Real-time enveloping with rotational regression. *ACM Trans. Graph. 26*, 3.

WEBER, O., SORKINE, O., LIPMAN, Y., AND GOTSMAN, C. 2007. Context-aware skeletal shape deformation. *Comput. Graph. Forum 26*, 3, 265–274.

YANG, X., SOMASEKHARAN, A., AND ZHANG, J. J. 2006. Curve skeleton skinning for human and creature characters. *Comput. Animat. Virtual Worlds 17*, 3-4, 281–292.