Two-Layer Sparse Compression of Dense-Weight Blend Skinning

Binh Huy Le* Zhigang Deng†
University of Houston

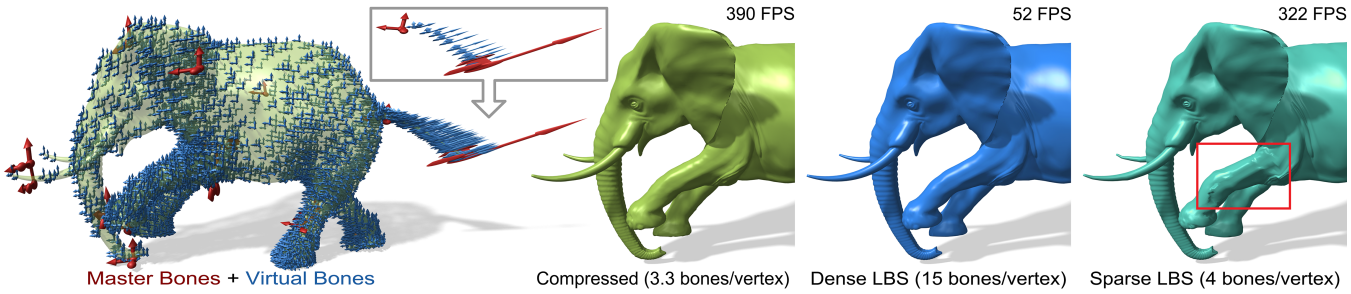

Figure 1: *Our compression model blends master bone transformations and caches them as virtual bone transformations (left most). It can compress Linear Blend Skinning (LBS) model with dense weights and generate a fast and compact model without sacrificing the quality of skinning, compared with dense-weight LBS model.*

Abstract

Weighted linear interpolation has been widely used in many skinning techniques including linear blend skinning, dual quaternion blend skinning, and cage based deformation. To speed up performance, these skinning models typically employ a sparseness constraint, in which each 3D model vertex has a small fixed number of non-zero weights. However, the sparseness constraint also imposes certain limitations to skinning models and their various applications. This paper introduces an efficient two-layer sparse compression technique to substantially reduce the computational cost of a dense-weight skinning model, with insignificant loss of its visual quality. It can directly work on dense skinning weights or use example-based skinning decomposition to further improve its accuracy. Experiments and comparisons demonstrate that the introduced sparse compression model can significantly outperform state of the art weight reduction algorithms, as well as skinning decomposition algorithms with a sparseness constraint.

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

Keywords: skinning, linear blend skinning, skinning from examples, sparse coding, dictionary learning

Links: DL PDF WEB VIDEO

*e-mail: bhle2@cs.uh.edu
†e-mail:zdeng@cs.uh.edu

1 Introduction

Blend skinning (i.e., smoothly interpolating deformation along the surface of 3D models) is probably the most widely employed skinning technique to date due to its simplicity, efficiency and flexibility. The blend skinning idea has been used in most popular skinning approaches including skeleton based skinning with linear blending [Merry et al. 2006], dual-quaternion blending [Kavan et al. 2008], cage based skinning with mean value coordinates [Ju et al. 2005], harmonic coordinates [Joshi et al. 2007], and combination model with bounded biharmonic weights [Jacobson and Sorkine 2011]. Blend skinning is typically linear; it is controlled by a weight matrix, where each element defines the contribution of a bone (in skeleton based skinning) or a control point (in cage-based skinning) to interpolation of a mesh vertex. To speed up skinning performance, a sparseness constraint is often imposed on the weight matrix, that is, the weight matrix contains only a small proportion of non-zero elements. In practice, for the sake of effective parallel implementation on GPUs or multi-core CPUs, a more strict sparseness constraint is typically imposed on the weight matrix, which requires every vertex to be associated with no more than k bones or control points. On the one hand, the sparseness constraint has the advantages of saving computation and balancing workload between different processing cores. On the other hand, this setup has the following intrinsic limitations.

Limitation #1: It is difficult to handle *exceptional vertices* that are naturally associated with more than k bones or control points. The exceptional vertices typically appear on smooth and highly deformable regions of 3D models. As a specific example shown in Fig. 2(b), more than 23 percent of the vertices in a cheb model (illustrated in red), rigged by [Baran and Popović 2007] with 17 bones, are influenced by all the bones. Also, exceptional vertices might be required by design; as an example, vertices on the palm region of a hand model (Fig. 2(c)) are influenced by several proximal phalanges and all five metacarpal bones. Indeed, due to existence of such exceptional vertices, a difficult trade-off often needs to be delicately handled in sparse-weight skinning models.

Limitation #2: Conventional weighted blending is inefficient from a computational perspective, despite the fact that performance has always been one of most important concerns for skinning models [Kavan et al. 2010]. Specifically, the weights of two neighboring

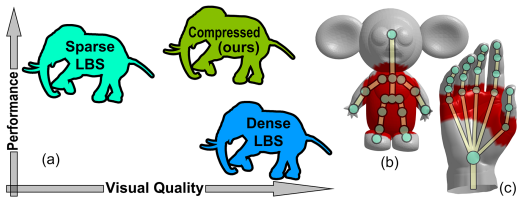

Figure 2: *(a): Our skinning compression model can achieve a high performance with insignificant loss of visual quality. (b) and (c): Examples of exceptional vertices (illustrated in red color): a cheb model (b) is rigged by [Baran and Popović 2007], and a hand model (c) is rigged manually.*

vertices are typically similar if skinning is smooth. These similar linear combinations are calculated multiple times; if they can be properly cached (or compressed), overall computational cost can be measurably reduced.

Limitation #3: Imposing a sparseness constraint makes a skinning problem a selection of discrete variables that does not have any yet known optimal polynomial solutions [Zou and Hastie 2005]. Putting it together with convex (non-negativity and affinity) constraints on weights would make the skinning problem even more challenging. Any non-optimal solution might lead to a non-smooth skinning model or even a bad approximation [Landreneau and Schaefer 2010]. Solutions to many skinning applications can be significantly simplified if a sparseness constraint does not need to be handled.

In this paper, we present a lossy weight matrix compression approach to free skinning models from the sparseness constraint and thus overcome the above limitations (Fig. 2(a)). Based on the weight compression, we construct an effective two-layer blend skinning model to reduce computation of linear blend skinning (LBS) with dense weights (Fig. 3). Specifically, its master bone blending layer blends the transformations of the original control bones (called *master bones*) and caches the results as virtual bone transformations. Then, its virtual bone blending layer blends the virtual bone transformations in a similar way to produce vertex transformations. In the virtual bone blending layer, each vertex transformation is blended by no more than two virtual bones.

Contributions. The main contributions of this work are:

- We introduce a two-layer blend skinning model to effectively compress weights of the widely used LBS model with insignificant loss of accuracy (§3.1). Our model is friendly to parallel implementation on multi-core processors.
- Besides formulating compression of skinning weights as a sparse dictionary learning problem (§3.2 to §3.4), our model also provides flexible control to users by setting either an error threshold or a desired skinning performance. It can directly work on an input bone-vertex weight map or utilize example poses to achieve a better accuracy (§3.5).

Compared with existing weight reduction techniques [James and Twigg 2005; Landreneau and Schaefer 2010], our model allows more flexible control on the trade-off between accuracy (skinning error) and performance (a desired number of blending operations). Through our experiments and direct comparisons, we show that our model achieves substantially smaller approximation errors than state of the art weight reduction techniques, given the same total number of bones (§4). We also analyze the performance and memory overhead of our approach on graphics hardware and its potential applications for other skinning models (§5).

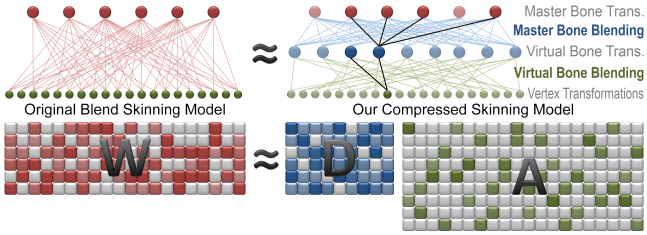

Figure 3: *A conventional blend skinning model (top left) with a dense weight matrix W (bottom left) is approximated as a two-layer blending with virtual bones (top right). This is equivalent to factorizing W into a sparse dictionary D and a matrix of sparse coefficients A (bottom right). D has at most c non-zero elements, while A has at most 2 non-zero elements. See §3.1 for details.*

2 Related Work

Blend Skinning. A straightforward idea of skinning a 3D model with smooth surface deformation is to blend a set of sparse deformation controls along its surface. The blending is typically a linear interpolation controlled by skinning weights. To date, two popular skinning approaches have been developed: *skeleton based skinning* and *cage based skinning*.

In skeleton based skinning, deformation of a 3D model is controlled by manipulating the transformations of a set of bones [Merry et al. 2006]. Through skinning weight based blending, the transformations are propagated to all the vertices of the model. The most popular way to blend bones is to linearly blend transformation matrices [Kavan et al. 2010; Hasler et al. 2010; Jacobson et al. 2012a]. Orthogonal bone transformations can be non-linearly transformed to a dual-quaternion form before dual-quaternion linear blending is applied [Kavan et al. 2007; Kavan et al. 2008]. Skinning weights are typically represented as a matrix form. A weight matrix can be constructed manually through weight painting, automatically extracted from static 3D models [Baran and Popović 2007; Jacobson et al. 2012b; Kavan and Sorkine 2012; Borosán et al. 2012], or created by skinning example poses [Wang and Phillips 2002; Mohr and Gleicher 2003; James and Twigg 2005; Schaefer and Yuksel 2007; Le and Deng 2012].

Cage based skinning [Ju et al. 2005] can be conceptually regarded as an extension of the well-known free form deformation method [Sederberg and Parry 1986], where a 3D model is embedded into a low resolution cage; thus, each vertex of the model can be represented relative to the vertices of the cage. By controlling positions of the cage vertices, the shape of the 3D model is deformed in accordance to the cage. Specifically, vertex coordinates of the 3D model are computed as weighted linear combinations of the cage vertices. Weights of the linear combinations can be computed by various methods, including barycentric coordinates [Warren et al. 2004], mean value coordinates [Ju et al. 2005], and harmonic coordinates [Joshi et al. 2007]. Recently, Jacobson et al. [2011] introduced bounded biharmonic weights to combine skeleton based and cage based into one skinning model.

Skinning Weight Reduction. One major issue of the above blend skinning methods is their scalability with respect to high resolution models. When a model has a large number of vertices, blending of many control parameters per vertex is very costly. Generally, this problem is handled by constraining the number of non-zero weights per vertex. In most cases, this threshold number is set to 4 for the sake of an efficient implementation on GPU vertex shader [Nguyen 2007]. This constraint can be simply enforced by select-

ing the four largest weights by consulting example poses [James and Twigg 2005; Schaefer and Yuksel 2007; Le and Deng 2012] or considering mesh fairness [Landreneau and Schaefer 2010].

Although these weight reduction methods help efficient skinning implementation, they have a common drawback as mentioned in §1, that is, they may significantly affect certain exceptional vertices that are naturally associated with more than 4 bones. This problem can be alleviated, to a certain extent, by employing ℓ^1-norm minimization for the sparseness constraint [Hasler et al. 2010], which helps to adapt the number of non-zero weights in accordance to the approximation power required for each vertex. However, output of different numbers of non-zero weights per vertex is unfriendly in terms of parallel implementation; thus, this will significantly degrade skinning performance.

Virtual Bones for Skinning. With a sparseness constraint enforced on the LBS model, skinning quality on certain parts of a 3D model can be enhanced by adding virtual (auxiliary) bones [Mohr and Gleicher 2003], where their transformations precisely fit the vertices on those parts of the 3D model. Virtual bone transformations give additional degrees of freedom to skinning models. However, there is no clear way to model the relation between virtual bone transformations and master bone transformations. Due to the requirement of additional input parameters, adding virtual bones has certain drawbacks, e.g., increasing the complexity of animation control and increasing data transfer between CPU and GPU.

Virtual bones are also used to approximate non-linear skinning models. Kavan and colleagues [2009] model the relation between virtual bone transformations and master bone transformations using a dual-quaternion blending technique. Also, their virtual bones require extra computational cost for data conversion between transformations and dual quaternions. Nevertheless, their work assumes the original skinning model is already sparse, and it actually increases the number of blending operations. Thus, the performance of their model [Kavan et al. 2009] is bounded by the sparse LBS model.

3 Compression of Blend Skinning Models

3.1 Problem Formulation

Let $W \in \mathbb{R}^{k\times n}$ be the weight matrix of an input skinning model with n vertices and k bones, as illustrated at the top left of Fig. 3. We denote the i-th column of W (or the original weights of the i-th vertex) as w_i. We compress this original skinning model using a two-layer blending scheme with virtual bones (the top right of Fig. 3). At the first layer, a.k.a. *master bone blending*, we calculate and cache the transformations of m virtual bones by blending the transformations of k original bones (called *master bones*). At the second layer, a.k.a. *virtual bone blending*, we calculate the position of each vertex by blending the transformations of the virtual bones and applying the resultant transformation to the vertex. We also impose a sparseness constraint on each blending layer to make the model friendly to parallel implementation on graphics hardware. Specifically, at the master bone blending layer, we allow at most c blending operations for each virtual bone; at the virtual bone blending layer, we allow at most 2 blending operations for each vertex.

Letting $d_j \in \mathbb{R}^k$ be the blending weights of the j-th virtual bone, we can represent all the master bone blending weights as a sparse matrix $D = [d_1, \ldots, d_m] \in \mathbb{R}^{k\times m}$. Similarly, letting $\alpha_i \in \mathbb{R}^m$ be the blending weights of the i-th vertex, we can represent all the virtual bone blending weights as another sparse matrix $A = [\alpha_1, \ldots, \alpha_n] \in \mathbb{R}^{m\times n}$, where each column α_i has at most two non-zero elements. With the above matrix representations, the blend skinning compression problem can be viewed as a sparse coding problem in which the original matrix W needs to be factorized into D (i.e., dictionary) and A (i.e., coefficients), as illustrated at the bottom of Fig. 3. Each column d_j is called an *atom* of the dictionary. This sparse coding problem is then formulated as minimizing the following quadratic error function:

$$\min_{D,A} {\Delta_W}^2 = \min_{D,A} \frac{1}{kn} \|DA - W\|_F^2 \tag{1a}$$

$$\text{Subject to: } \mathbf{card}(\alpha_i) \le 2, \forall i \tag{1b}$$

$$\mathbf{card}(d_i) \le c, \forall i \tag{1c}$$

Parameter selection. We denote $\mathbf{card}(x)$ as the cardinality (number) of non-zero elements in vector x. The constraint in Eq. (1b) enforces that at most two virtual bones can influence any particular vertex. This number (two) is empirically chosen for best performance. Since the number of vertices is typically large, minimizing the number of blending operations at the virtual bone blending layer would be the most effective way to reduce computational cost. In Eq. (1c), we also constrain the sparseness of D to reduce computational cost and thus improve performance. In order to maintain the approximation power of this model, we need to keep the sparseness of dictionary atoms to be no less than that of the original weights or vertices. However, we empirically found that slightly increasing the sparseness of atoms could expand its approximation power. For this reason, we choose $c = \max_{i=1\ldots n} \mathbf{card}(w_i) + 1$ in our experiments. Note that users can perform fine tuning on c to further optimize performance with different input data.

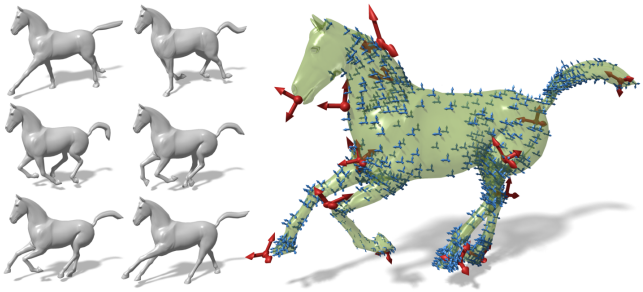

Figure 4: *Some example poses of an animated mesh sequence (left) and its corresponding compressed blend skinning model (right). Master bone transformations are illustrated in red, and virtual bone transformations are illustrated in blue. We place each virtual bone at a vertex with the largest sparse coefficient. Our model distributes virtual bones adaptively so that more virtual bones are employed for highly deformed regions (e.g., the legs or the tail).*

Approximation power and effectiveness of our model. Compared with directly imposing a sparseness constraint on skinning weights, our two-layer sparse blending model essentially expands the linear blending space from *c-1* bone blending operations per vertex to $2c$ bone blending operations per vertex. Furthermore, virtual bones in our model can cache similar blending of master bones and save significant computational cost. The virtual bones can also be adaptively distributed in accordance with deformation complexity, e.g., the vertices on highly deformed regions could employ more virtual bones than those on other regions, as illustrated in Fig. 4.

In terms of parallel implementation, our compression model is friendly to stream processing architectures, such as GPUs, since blending operations for each node (virtual bone or vertex) are independent of each other, and they have the same workload (i.e.,

the same number of blending operations per node). Thus, the two blending layers in our model can be effectively implemented on GPUs as a process of two passes.

3.2 Algorithm Overview

The overview of our sparse weight matrix factorization for compressing blend skinning models is presented in **Algorithm** 1. Users can terminate this algorithm two different ways: specifying a maximum number of virtual bones (or dictionary size) Σ, or specifying an error threshold ϵ. In the former case, we can control the computational cost of the compressed model (i.e., number of blending operations). In the latter case, we can control the accuracy of the compressed model (i.e., the approximation error, Δ_W).

Algorithm 1 Sparse Weight Matrix Factorization

Input: a weight matrix $W = [w_1, \ldots, w_n] \in \mathbb{R}^{k \times n}$, an error threshold ε OR a maximum number of virtual bones Σ.
Output: $D = [d_1, \ldots, d_m] \in \mathbb{R}^{k \times m}$ and $A = [\alpha_1, \ldots, \alpha_n] \in \mathbb{R}^{m \times n}$ s.t. Eq. (1)

1: Initialize a dictionary with $m = 2$ atoms: $D = \{d_1, d_2\}$
2: Initialize coefficients A according to D
3: **repeat**
4: **for** $t = 1 \rightarrow \kappa(\Delta_W, m)$ **do**
5: Find vertex p with the largest approximation error
6: Add w_p to the dictionary
7: Update coefficients A from vertex p
8: **end for**
9: **repeat**
10: Update dictionary D
11: **for** $i = 1 \rightarrow n$ **do**
12: Update coefficients A from vertex i
13: **end for**
14: **until** Convergence
15: **until** $\Delta_W < \varepsilon$ OR $m = \Sigma$
16: **return** D and A

The algorithm 1 can be briefly summarized as follows: First, we initialize a minimum dictionary with 2 atoms and its corresponding coefficients (line 1 and line 2). Then we sequentially add atoms to the dictionary (line 4 to line 8) along with jointly optimizing the dictionary and coefficients (line 9 to line 14) until error $\Delta_W < \varepsilon$ or the size of the dictionary $m = \Sigma$.

Initialization. We initialize the first atom of the dictionary using the weights of a vertex with the largest ℓ^2-norm (i.e. $d_1 = \arg\max_{w_i} \|w_i\|_2$). The second atom is initialized as the weights of another vertex with the smallest dot product to d_1 (i.e., $d_2 = \arg\min_{w_i} \{w_i \cdot d_1\}$). The coefficients A can be solved per-vertex (column by column) by linear least squares with two unknowns (corresponding to two atoms) as follows: $\alpha_i = \arg\min_x \|Dx - w_i\|_2^2$.

Adding atoms. In order to minimize Δ_W, we always add the weights of a vertex p that has the largest error to the dictionary as follows (line 5):

$$D \leftarrow [D, w_p] \text{ s.t. } p = \arg\max_i \|D\alpha_i - w_i\|_2^2 \tag{2}$$

Instead of adding atoms to the dictionary one by one, followed by a joint dictionary-coefficients optimization, we can improve performance by adding $\kappa(\Delta_W, m)$ atoms (line 4) before each joint optimization. $\kappa(\Delta_W, m)$ is computed based on the current approximation error $\Delta_W(D, A)$ (refer to Eq. (1a)) and the current dictionary size m (Eq. (3)). Note that when adding each atom without dictionary optimization, we still need to update coefficients according to the added atom (line 7).

$$\kappa(\Delta_W, m) = \min\{\left(\frac{\Delta_W}{\varepsilon} - 1\right) m + 1, \Sigma - m\} \tag{3}$$

Dictionary-coefficients optimization. We solve the joint dictionary-coefficients optimization problem by a block coordinate descent approach with warm restarts [Nocedal and Wright 2000], that is, alternatively updating the dictionary (line 10) and coefficients (line 11 to line 13). With the warm restarts, we found that its alternative update process can converge within a small number of iterations. In our experiments, we found that it typically converged within 3 iterations.

In the following sections, we describe details of two major steps in the algorithm 1, i.e., dictionary update (§3.3) and coefficients update (§3.4). Note that two coefficients update steps (line 7 and line 12) use the same procedure. To speed up performance, we only perform updates on a subset of vertices with potential coefficient changes. This is implemented as an ordered update process that starts from one vertex, i.e., vertex p in line 7 and vertex i at line 12.

3.3 Dictionary Update

At the dictionary update step, we employ an online dictionary update algorithm with warm restarts [Mairal et al. 2010], since it is simpler and faster than other methods (e.g., K-SVD [Aharon et al. 2006] or Newton's method [Lee et al. 2007]). Since our dataset is relatively small, we can use it at each update rather than sampling it. Specifically, we precompute Φ and Γ as follows:

$$\Phi = \sum_{i=1}^{n} \alpha_i \alpha_i^{\mathsf{T}} = [\phi_1, \ldots, \phi_m] \in \mathbb{R}^{m \times m} \tag{4a}$$

$$\Gamma = \sum_{i=1}^{n} w_i \alpha_i^{\mathsf{T}} = [\gamma_1, \ldots, \gamma_m] \in \mathbb{R}^{k \times m} \tag{4b}$$

Since α_i is sparse, the complexity of computing Φ and Γ is $\mathcal{O}(n)$. Then we update each atom (column) d_j of the dictionary as follows:

$$d_j \leftarrow \frac{1}{A_{j,j}} (\gamma_j - D\phi_j) + d_j \tag{5}$$

After each atom update, we enforce the sparseness constraint in Eq. (1c) by keeping the c largest elements of vector d_j, while setting the others to be 0. Finally, we normalize the result so that d_j satisfies the affinity constraint (i.e., the sum of all the elements equals to 1) by keeping it within the effective range of a floating point number on the machine. Although other normalization methods such as dividing d_j by the maximum element would work in theory, we found that dividing d_j by the sum of all the elements (normalization with the affinity constraint) can improve the rate of convergence since the weights $\{w_i\}$ are also constrained to be affinity.

3.4 Coefficients Update

The challenge of coefficients update for each vertex is to find two optimum dictionary atoms (i.e., virtual bones) that contribute to the vertex most. If the two optimum atoms are determined, the coefficients update becomes a trivial least squares problem with two unknowns. Although the two optimum atoms can be determined

by a greedy algorithm such as matching pursuit [Mallat and Zhang 1993], it requires $\mathcal{O}(m)$ operations per vertex, or $\mathcal{O}(mn)$ operations for updating all the coefficients. To this end, we propose a fast coefficients update method that requires approximately $\mathcal{O}(n)$ operations by assuming skinning weights are typically smooth across neighboring vertices (the fairness assumption on the manifold).

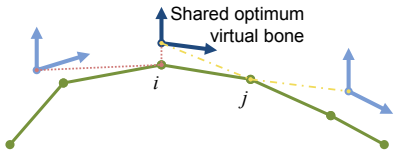

Figure 5: *On a manifold, two neighboring vertices typically have similar skinning weights and coefficients; thus, they most likely share the same optimum virtual bones (dictionary atoms). This assumption can be used to accelerate the update of coefficients.*

With this assumption, two neighboring vertices on a mesh would have similar skinning weights and coefficients. Furthermore, the two vertices might share the same optimum virtual bones, as illustrated in Fig. 5. If the two optimum atoms for vertex i are updated, we expect the two atoms might be the optimal ones for its neighboring vertices. Thus, we can perform the update as a preorder graph traversal, that is, we start coefficient updating at a vertex with two known optimum dictionary atoms and use these atoms as candidates to update the coefficients of its neighboring vertices; then we repeat the same process for these neighboring vertices. Specifically, we implement the coefficients update recursively as a depth-first search on a mesh edge-based graph (Algorithm 2).

Algorithm 2 CoefficientsUpdate(vertex i, candidate atoms d_r, d_s)

Input: vertex i, candidate atoms d_r, d_s
1: **if** the linear combination of d_r and d_s improves error E_i **then**
2: Update α_i and E_i by linear least squares
3: **for all** $j \in \mathcal{N}(i)$ **do**
4: Find $\{r', s'\} \subset \{r, s, \rho_j, \sigma_j\}$ s.t. the linear combination of $d_{r'}$ and $d_{s'}$ best approximates w_j
5: CoefficientsUpdate(j, $d_{r'}$, $d_{s'}$)
6: **end for**
7: **end if**

In Algorithm 2, assuming vertex i has two candidate atoms with indices r and s, we update coefficients α_i and corresponding approximation error E_i (line 2) if and only if the linear combination of the two candidate atoms d_r and d_s can improve E_i (line 1). Here we define the approximation error E_i of weight w_i as ${E_i}^2 = \|D\alpha_i - w_i\|_2^2$. Since the two candidate atoms are known, we solve the optimum α_i by least squares with two unknowns, i.e., $(\alpha_i)_r$ and $(\alpha_i)_s$. If w_i is constrained to be affinity, we also impose the affinity constraint to α_i to improve the rate of convergence. Specifically, we solve:

$$\min_{\substack{(\alpha_i)_r \\ (\alpha_i)_s}} \|d_r(\alpha_i)_r + d_s(\alpha_i)_s - w_i\|_2^2 \text{ s.t. } (\alpha_i)_r + (\alpha_i)_s = 1 \quad (6)$$

Once candidate atoms d_r and d_s are used to update coefficients α_i of vertex i, we then recursively update coefficients of its neighboring vertices $\mathcal{N}(i)$ (line 3 to line 6). Let j be a vertex in $\mathcal{N}(i)$, we first find candidate atoms r' and s' of j so that the linear combination of $d_{r'}$ and $d_{s'}$ best approximates w_j (line 4). We only select r' and s' within the set of current optimum atoms of vertex j and the candidate atoms of vertex i. Specifically, let ρ_j and σ_j be indices of current optimum atoms of vertex j, i.e. $(\alpha_j)_{\rho_j}$ and $(\alpha_j)_{\sigma_j}$ are nonzero coefficients, we only select r' and s' in the set of $\{r, s, \rho_j, \sigma_j\}$. In the end, we consider six combinations of two atom indices and solve linear least squares for each of them, similar to Eq. (6), to find the best combination.

Employing coefficients update in the main algorithm. We employ the above recursive algorithm 2 in the main algorithm 1 for two purposes. First, after weights w_p are added to the dictionary (line 6 in Algorithm 1), the new dictionary atom w_p is the optimum atom for vertex p. We start a recursive coefficients update from vertex p (line 7) with the latest added dictionary atom w_p. The other candidate atom can be selected arbitrarily. Compared to a conventional solution of updating coefficients for all the vertices, our algorithm can save a significant amount of unnecessary update cost for vertices far away from p on the mesh that are not potentially affected by the latest added atom (virtual bone).

Second, in full coefficients update step, we start the recursive coefficients update one by one from every vertex (line 12 in Algorithm 1). For each vertex i, we keep its current optimum atoms and use them as candidate atoms. In other words, candidate atoms d_r and d_s for vertex i will be the atoms such that $(\alpha_i)_r \neq 0$ and $(\alpha_i)_s \neq 0$. If the least squares solution using these candidate atoms improves the approximation error, the update will be propagated through its neighboring vertices on the mesh. In general, this recursive update process might visit each vertex several times, which makes the amortized complexity of the full coefficient update step $\mathcal{O}(n)$ in practice. We will further analyze performance of our algorithm through experiments in §4.

3.5 Factorization from Example Poses

Our model can also utilize example poses for a better approximation. Although we can potentially extend our weight factorization algorithm for various skinning models such as cage based deformation [Ju et al. 2005] or dual-quaternion blending [Kavan et al. 2008], without loss of generality, in this work we only demonstrate our model for the most popular LBS approach.

In order to utilize the example poses, we need to find a compression model that best approximates given example poses instead of a compression model that best approximates skinning weights. Similar to previous works [James and Twigg 2005; Kavan et al. 2010; Le and Deng 2012], we also minimize a quadratic error function on all the example poses given corresponding bone transformations. Fortunately, we can still adapt our sparse weight matrix factorization (Algorithm 1) by replacing the approximation error on skinning weights with the approximation error on example poses.

Assume we have f example poses. Let $v_i^t \in \mathbb{R}^3$ be the position of vertex i in example pose t, and $u_i \in \mathbb{R}^3$ be the rest pose position of vertex i. For each example pose t, we represent its bone transformations as matrices, and use $T_j^t \in \mathbb{R}^{3\times4}$ to denote the transformation of bone j relative to its position at the rest pose. Given the example poses and bone transformations, we can solve the optimized LBS weights $W^* = [w_1^*, \ldots, w_n^*] \in \mathbb{R}^{k\times n}$ using linear least squares with equality constraint ($\sum_{j=1}^{k}(w_i^*)_j = 1$) and inequality constraint ($w_i^* \geq 0$) [Le and Deng 2012]. Note that the optimized weights W^* are not constrained to be sparse. We utilize example poses by modifying certain steps in Algorithm 1 as follows.

Approximation error. We replace the approximation error on weight matrix ${\Delta_W}^2$ (Eq. (1)) with the approximation error on example poses ${\Delta_E}^2$ (Eq. (7a)). The approximation error for each vertex ${E_i}^2$ (Eq. (7b)) is used to find new atoms for the dictionary (line 5 in Algorithm 1). Here, we normalize the approximation error

by subtracting the lower bound E_i^{*2} (i.e., the approximation error with the optimized LBS weights w_i^*, refer to Eq. (7c)) from it.

$$\Delta_E{}^2 = \frac{1}{3fn}\sum_{i=1}^{n} E_i{}^2 \tag{7a}$$

$$\text{Where: } E_i{}^2 = \sum_{t=1}^{f}\left\|\sum_{j=1}^{k}(D\alpha_i)_j T_j^t \begin{bmatrix} u_i^t \\ 1 \end{bmatrix} - v_i\right\|_2^2 - E_i^{*2} \tag{7b}$$

$$E_i^{*2} = \sum_{t=1}^{f}\left\|\sum_{j=1}^{k}(w_i^*)_j T_j^t \begin{bmatrix} u_i^t \\ 1 \end{bmatrix} - v_i\right\|_2^2 \tag{7c}$$

Coefficients update. We also use the approximation error $E_i{}^2$ on example poses (Eq. (7b)) to solve coefficients for the linear combination of atoms (line 1 and line 4 in Algorithm 2).

Dictionary update. We use the optimized weights W^* to precompute matrix Γ instead of W (refer to Eq. (4b)). Then we perform the same column update using Eq. (5). It should be noted that this dictionary update neither gives the best error reduction nor guarantees convergence in theory. However, this simple update is faster and more effective than other complicated methods, e.g., employing divergence protections such as Shift-cutting [Nocedal and Wright 2000] or Marquardt parameter [Marquardt 1963].

4 Results

To evaluate our approach, we first performed skinning decomposition on a number of animated mesh sequences provided by [Sumner and Popović 2004; Vlasic et al. 2008] (reported in Table 1) to extract their optimized linear blend skinning models. For fair comparisons between different sequences, we first rescaled all the datasets so that their rest poses are tightly fit in a unit sphere. Then, for each sequence, we implemented the smooth skinning decomposition algorithm [Le and Deng 2012] to extract its flexible bone transformations and convex weight map.

Name	n	f	Name	n	f
samba	9971	175	horse-gallop	8431	48
camel-gallop	21887	48	camel-collapse	21887	53
elephant-gallop	42321	48	horse-collapse	8431	53

Table 1: *The test datasets used in this work. n denotes the number of vertices, and f denotes the number of example poses.*

We also considered different levels of the sparseness constraint on the weight map by setting different maximum numbers of bones per vertex (namely, 4 bones per vertex, 8 bones per vertex, and dense weights without the sparseness constraint). In our compressed skinning model, we calculate *the number of bones per vertex* as the average number of blending operations per vertex (including the blending operations on both layers). Specifically, the number of bones per vertex in our model is $\frac{mc}{n} + 2$. To compare the approximation powers of different approaches on the example poses, we use a fitting error measure E proposed by Kavan et al. [2010]. In this section, we report all errors multiplied by 1000 for the sake of convenience.

In Fig. 6, we show several example results of our skinning compression model with different thresholds of the objective function. We can hardly observe visual distortions if the error threshold drops below 0.001. If example poses are not utilized in our approach,

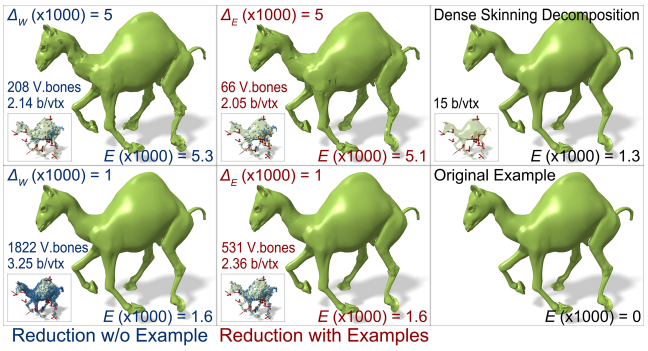

Figure 6: *Example results of our skinning compression model with different thresholds of the objective function (Δ_W and Δ_E). We can hardly observe visual distortions if the error threshold drops below 0.001. The input skinning model (camel gallop) has 15 bones with a dense weight matrix. For each compression example, its bone distribution is illustrated at the bottom-left corner. Master bone transformations are illustrated in red; virtual bone transformations are illustrated in blue. The fitting error on the example poses, E, is also reported. "V. Bones" denotes virtual bones; "b/vtx" denotes bones per vertex.*

the value of the objective function Δ_W represents the average error in original bone-vertex weights, since the original bone-vertex weights satisfy the affinity constraint. Δ_W can also represent the *relative* distortion of the compressed model, compared to the original blend skinning model. Otherwise, if example poses are used, the value of the objective function Δ_E represents the *absolute* distortion, compared to the original skinning model.

Since all the datasets are rescaled to tightly fit in a unit sphere, the amounts of distortion caused by the both compression methods (with/without utilizing example poses) are highly similar if the same error threshold is used. This relation is illustrated in Fig. 7. In this figure, at each step we incrementally add 10 virtual bones to our skinning compression approach and then perform joint dictionary-coefficients optimization. For each step, the fitting errors on the example poses are plotted along with the objective function.

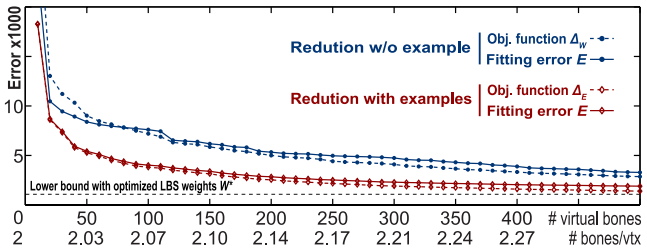

Figure 7: *Relation of the objective function and the fitting error on the example poses. The input skinning model has 15 bones with a dense weight matrix extracted from a camel gallop sequence. The relation between value of the objective function and the fitting error is quite similar for the both cases (i.e., with/without utilizing example poses). Compression utilizing the example poses produces smaller fitting errors than that without utilizing the example poses.*

Figs. 6 and 7 also illustrate that, to achieve the same approximation accuracy, utilizing the example poses can produce a compressed skinning model with a smaller number of virtual bones than with-

out utilizing the example poses. In addition, when the example poses are provided, we have a more accurate way to control the fitting error. Specifically, we can calculate the fitting error using the following equation: $E^2 = {\Delta_E}^2 + {E^*}^2$, where ${E^*}^2$ is the lower bound of the fitting error (i.e., the fitting error with the optimized LBS weights W^*).

Comparison without utilizing example poses. In Table 2, we compare the approximation errors on weight matrix (i.e., Δ_W) among our method, k-largest weight reduction, and smooth weight reduction [Landreneau and Schaefer 2010]. We can see that, for all the cases, our approach achieves substantially smaller approximation errors than the other two approaches. In this comparison, all the three methods only perform reduction on the weight matrix without utilizing example poses. We use two sets of input skinning models generated by skinning decomposition [Le and Deng 2012]: skinning models with 8 bones per vertex and with dense bone-vertex weights. Using the three different methods, the skinning models with 8 bones per vertex are reduced to 4 bones per vertex, and the skinning models with dense weights are reduced to 4 bones per vertex and 8 bones per vertex, respectively. Given a reduced number of bones per vertex, k_r, we calculate the number of virtual bones (or the size of the dictionary) for our method as $\Sigma = \frac{(k_r-2)n}{c}$. In the k-largest weight reduction method, we keep only the k_r largest weights per vertex and then normalize the sum of these weights to be 1. In the smooth weight reduction method [Landreneau and Schaefer 2010], at the normalization step, it also takes mesh fairness into account by matching the Laplacian of reduced weights with that of the original weight map. Incorporating smoothness to the reduction process would increase the approximation error on the weight matrix.

Name$_{[k]}$	8 to 4 bones/vertex			Dense to 4 bones/vertex			Dense to 8 bones/vertex		
	k-Largest	Smooth	Ours	k-Largest	Smooth	Ours	k-Largest	Smooth	Ours
samba$_{10}$	19.6	24.5	**1.2**	21.9	26.6	**1.1**	1.8	2.4	**0.2**
samba$_{20}$	14	18	**1.7**	18.1	22.2	**2.4**	5	6.5	**0.9**
camel-gallop$_{15}$	12.3	18.8	**0.5**	19.5	27.6	**0.8**	5	8.9	**0.2**
camel-gallop$_{30}$	8.8	14	**0.6**	19.7	26.2	**1.3**	7.9	12.6	**0.5**
elephant-gallop$_{15}$	16.4	24.5	**0.5**	30.3	42	**0.8**	8.1	12.1	**0.3**
elephant-gallop$_{30}$	12.8	20.1	**0.6**	25.9	34.4	**1.3**	10	14.6	**0.5**
horse-gallop$_{15}$	13	17.3	**0.9**	157.7	161.5	**1.3**	4.9	6.6	**0.3**
horse-gallop$_{30}$	7.3	9.6	**0.8**	23.1	29.6	**2.4**	9.1	12.4	**0.9**
camel-collapse$_{20}$	14.7	23.2	**1.3**	22	30.1	**1.8**	7.5	12.4	**0.6**
camel-collapse$_{40}$	9.9	15.9	**1.6**	18.8	24.5	**2.5**	8.3	12.1	**1.1**
horse-collapse$_{20}$	15.9	22.4	**2.8**	28.8	37	**4**	10.1	14.2	**1.5**
horse-collapse$_{40}$	10.8	14.7	**3.3**	23.6	29.4	**5.1**	11	14.6	**2.7**

Table 2: *Comparison of the approximation errors on weight matrix (Δ_W): k-largest weight reduction, smooth weight reduction [Landreneau and Schaefer 2010], and our method. In this comparison, example poses are not used. The number of bones k is shown as a subscript of input model name. The errors are multiplied by 1000 for convenience.*

Comparison with utilizing example poses. If example poses are used, we also compare our method with geometry weight reduction [James and Twigg 2005], poisson weight reduction [Landreneau and Schaefer 2010], and smooth skinning decomposition [Le and Deng 2012]. From an original skinning model with k bones per vertex, the geometry weight reduction method selects k_r bones per vertex with best approximation, normalizes the weights of the k_r bones, and then minimizes the fitting error on the example poses using linear least squares. Similar to the smooth weight reduction [Landreneau and Schaefer 2010], the Poisson weight reduction method takes mesh fairness into account by matching Laplacians on the example poses.

Qualitative comparisons among the weight reduction methods are presented in Fig. 8. All the methods reduce the skinning models with dense weights to 4 bones per vertex. From this figure, we can see that the weight reduction methods utilizing the example poses always give better approximations than the same methods without utilizing the example poses. In general, the results by the smooth and Poisson weight reduction methods are smoother and more pleasing than those by the k-largest and geometry weight reduction methods. Meanwhile, without noticeable visual distortions, our skinning compression method approximates the original skinning models significantly better than the four weight reduction methods (i.e., smooth, Poisson, k-largest, and geometry). In particular, when the example poses are utilized, our method, with only 4 bone-blending operations per vertex, can compress and approximate the original models as good as dense-weight skinning models. This approximation is even better than the employed original skinning decomposition approach [Le and Deng 2012] with 4 bones per vertex, e.g. in the case of the elephant-gallop model.

In Table 3, we also quantitatively compare the fitting errors on example poses among all the methods. The results show that our model can substantially outperform the four chosen weight reduction methods. In most cases, our results are even better than the employed original skinning decomposition approach [Le and Deng 2012] with the same number of bones. Note that this comparison gives certain bias to the employed skinning decomposition approach [Le and Deng 2012], since it is allowed to optimize bone transformations and the rest pose to fit with the reduced weights.

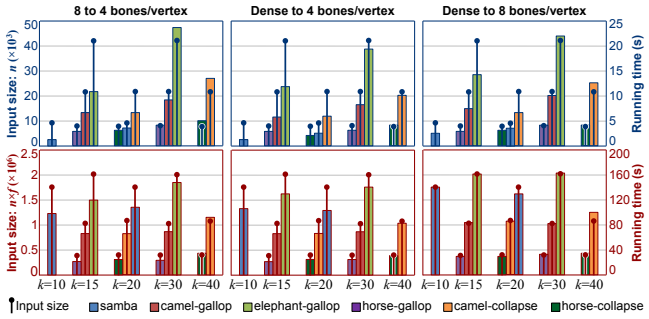

Figure 9: *Running time (seconds) of our skinning compression approach on CPU. The first row shows running time of our compression without using example poses and the bottom row shows the running time with using example poses. k denotes the number of master bones, n denotes the number of vertices of the input 3D model, and f is the frame number of example poses.*

Performance. We also illustrate the running time of our approach for some test cases in Fig. 9. We implemented our approach in C++ and ran all the experiments on an off-the-shelf computer with a single 2.0 GHz CPU core. As illustrated in this figure, when the example poses are utilized, our approach will need more than an order of magnitude running time than the case without utilizing example poses. For the same number of master bones, we observe the running time of our compression increases linearly with respect to the size of the input data. Approximately, the running time without example poses increases linearly with respect to the number of vertices n, and the running time with the example poses increases linearly with respect to the size of all the example poses (i.e., $n \times f$). This observation shows that our algorithm, especially the coefficients update (§3.4), has approximately a linear complexity in practice.

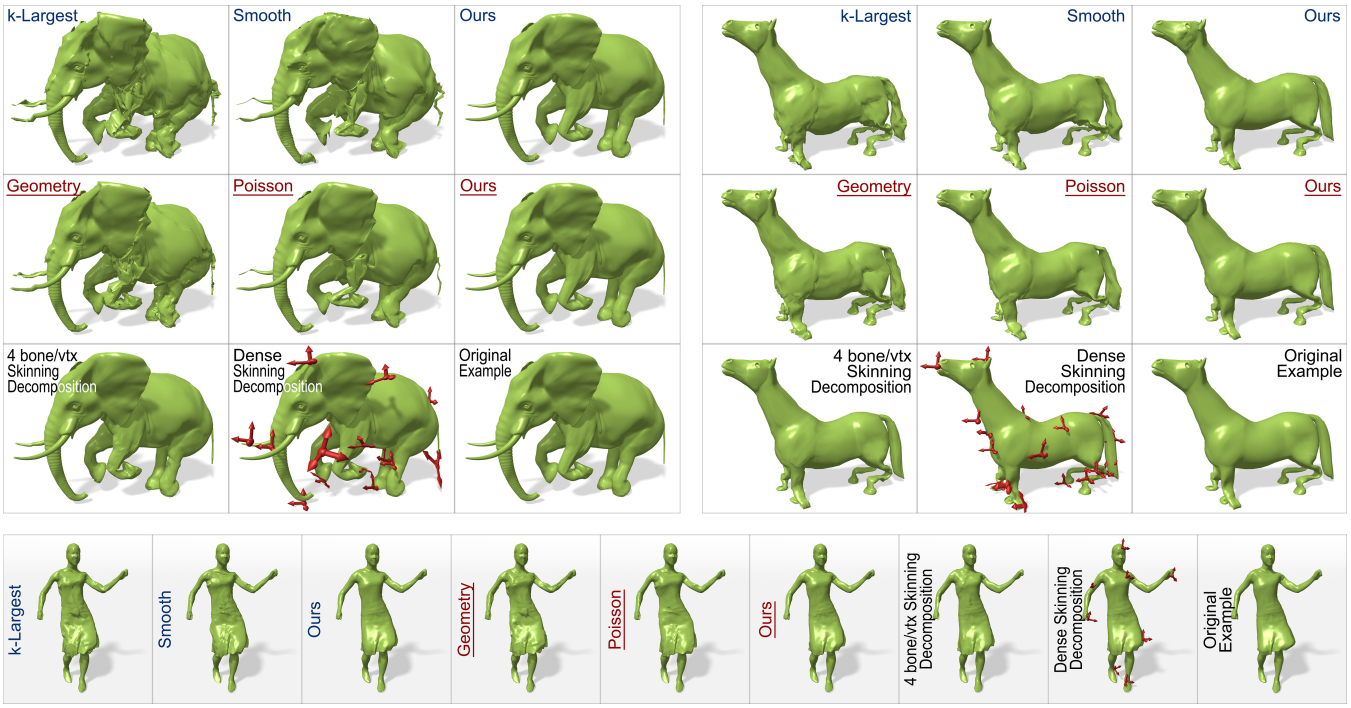

Figure 8: *Comparisons between our method and several selected skinning weight reduction methods (i.e., k-largest weight reduction, smooth weight reduction [Landreneau and Schaefer 2010], geometry weight reduction [James and Twigg 2005], and Poisson weight reduction [Landreneau and Schaefer 2010]). The three used models are elephant-gallop with 15 bones (top-left), horse-collapse with 20 bones (top-right), and samba with 10 bones (bottom). All the methods reduce the skinning models with dense weights (obtained via the smooth skinning decomposition method [Le and Deng 2012]) to 4 bones per vertex. The methods noted with blue do not utilize example poses, while the methods noted with red utilize example poses. In addition, "4 bones/vtx skinning decomposition" denotes the skinning model with 4 bones per vertex that is directly computed by the smooth skinning decomposition approach [Le and Deng 2012].*

5 Discussion

Besides the LBS model (as demonstrated above), our two-layer blend skinning compression approach can also be potentially extended to handle and compress other skinning models. For instance, we could reduce non-linear skinning models with dense weights such as dual-quaternion blend skinning [Kavan et al. 2008] by replacing the linear coefficients update in §3.4 with a non-linear least squares solver (e.g., the Levenberg-Marquardt algorithm [Marquardt 1963]). Our compression method could also work for cage-based deformation models by using an objective function similar to the one used in [Landreneau and Schaefer 2010]. Note that additional non-trivial analysis would be required to polish the above thoughts.

Our current approach has certain limitations.

- First, our two-layer blending approach imposes some computational overhead. In Fig. 10, we show GPU performance comparison between our approach and the LBS model. Our two-layer model performs much faster than the LBS model with dense weights. Our model and the LBS with sparse weights have similar performance in some scenarios that require intensive computation, i.e., models with 8 bones per vertex or models with a high resolution (e.g., the camel model (21887 vertices) and the elephant model (42321 vertices)). In other scenarios that require light computation, our model performs slower due to extra I/O operations on the buffers and cost for synchronization between two passes on GPUs. Fortunately, since the overhead is insignificant for complex models, our approach is suitable well for skinning applications that require both high performance and visual quality.

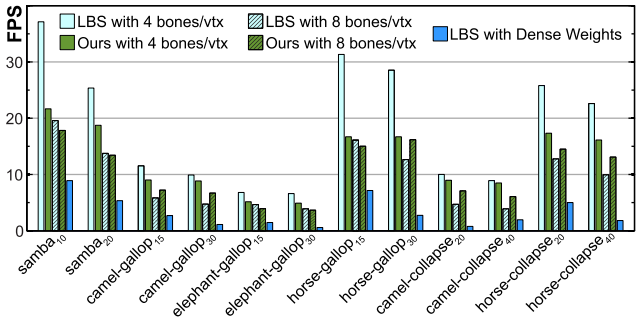

Figure 10: *GPU Performance comparison between LBS and our approach. Both methods were implemented with Microsoft Direct3D 11 running on NVIDIA Geforce GT 540M. Normal vectors were transformed and then normalized to unit length. The performance was measured by animating 150 instances and reporting FPS for the entire pipeline including rendering.*

- Second, our model requires additional storage space for caching virtual bone transformations. Fig. 11 visualizes the memory overhead required by our approach, compared with linear blend skinning with the same number of bones per ver-

Name$_{[k]}$	8 to 4 bones/vertex						Dense to 4 bones/vertex						4 b/v SD	Dense to 8 bones/vertex						8 b/v SD	Dense SD
	k-Largest	Smooth	Ours w/o Ex	Geometry	Poisson	Ours w/ Ex	k-Largest	Smooth	Ours w/o Ex	Geometry	Poisson	Ours w/ Ex		k-Largest	Smooth	Ours w/o Ex	Geometry	Poisson	Ours w/ Ex		
samba$_{10}$	10.6	11.5	**5**	9.4	10.5	**4.8**	11.8	12.7	**4.8**	10.5	11.6	**4.7**	5.7	5.1	5.3	**4.8**	4.8	5	**4.8**	4.9	4.8
samba$_{20}$	5.7	6.5	**2.3**	5	5.8	**2.1**	7.6	8.3	**2.4**	6.8	7.6	**2**	2.7	3.8	4.4	**2**	3.1	3.7	**1.9**	2.2	1.9
camel-gallop$_{15}$	8.3	9.7	**1.6**	6	8	**1.4**	13.1	14.9	**1.5**	9.5	11.9	**1.3**	2.3	5	6.5	**1.3**	3.2	4.2	**1.3**	1.6	1.3
camel-gallop$_{30}$	8.1	9.4	**1**	4.9	7.6	**0.8**	13.8	15.5	**2**	8.3	10.6	**0.8**	1.3	7.5	9.2	**0.7**	3.7	5.4	**0.4**	0.8	0.3
elephant-gallop$_{15}$	11.5	13.8	**2.1**	9.1	12.4	**1.9**	18.7	21.2	**1.8**	13.7	16.9	**1.6**	3.2	9	10.9	**1.7**	5.2	7.1	**1.6**	2	1.6
elephant-gallop$_{30}$	4.5	5.4	**1**	3.3	4.8	**0.8**	13	15.2	**1.5**	9.1	11.3	**0.7**	1.5	6.6	8.3	**0.7**	4.2	5.8	**0.5**	0.9	0.4
horse-gallop$_{15}$	9.5	10.3	**2.6**	7	8.2	**2.3**	20.7	23.5	**2.4**	10.1	11.5	**2**	3.2	5.9	6.9	**1.9**	3.9	4.7	**1.9**	2.5	1.9
horse-gallop$_{30}$	5.4	5.7	**1.2**	3.8	4.5	**1**	18.1	20.8	**3.4**	12.3	14.7	**1.6**	1.8	9.6	11.6	**1.6**	5.3	6.7	**0.8**	0.9	0.5
camel-collapse$_{20}$	10.6	12.4	**1.8**	6.2	8.4	**1.5**	15.5	17.4	**1.9**	10.3	13.4	**1.4**	2.8	7.2	9.5	**1.2**	3.6	5.4	**1.1**	1.5	1.1
camel-collapse$_{40}$	6.9	7.6	**1.6**	4	5.8	**1.2**	15.1	16.4	**3.1**	9.8	12.5	**1.5**	2.1	9.2	11	**1.5**	4.5	6.5	**0.8**	1.1	0.5
horse-collapse$_{20}$	12	13.4	**3**	7.2	9	**2.1**	17.9	20.3	**3.5**	11	13.5	**2.1**	3.2	8.4	10.5	**1.9**	4.2	5.5	**1.4**	1.9	1.3
horse-collapse$_{40}$	8.2	8.4	**2.8**	4.4	5.6	**2**	17.8	20	**4.8**	10.1	12.6	**2.5**	2.3	10.9	12.8	**2.9**	4.9	6.4	**1.3**	1.3	0.6

Table 3: *Comparison of the fitting errors on example poses (E) among our skinning compression method and several selected weight reduction methods: k-largest weight reduction, geometry weight reduction method [James and Twigg 2005], the smooth weight reduction method [Landreneau and Schaefer 2010], and the Poisson weight reduction method [Landreneau and Schaefer 2010]. "SD" denotes the employed original skinning decomposition method [Le and Deng 2012]. "b/v" denotes bones per vertex. "w/ Ex" (or "w/o Ex") denotes with (or without) utilizing example poses in our approach. The numbers in blue denote results without utilizing example poses, while the numbers in red with underline denote results with utilizing example poses. The number of bones k is shown as the subscript of the input model name. For the sake of convenience, all the errors in this table are multiplied by 1000.*

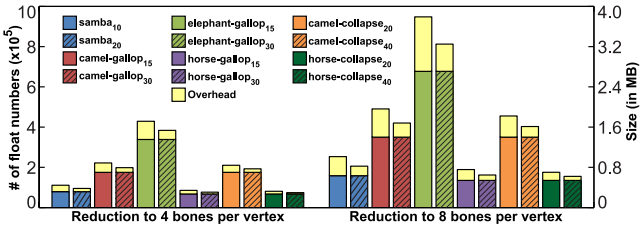

Figure 11: *The memory overhead required by our approach to cache virtual bone transformations, compared with the linear blend skinning model with the same number of bones per vertex. Here we assume each vertex is stored as four 32-bit floating point numbers, a transformation matrix is stored as sixteen floating point numbers, and a blending weight with index is stored as two floating point numbers.*

tex. Note that the total memory required by our approach to store blending weights is equal to the memory for storing the linear blending skinning weights. The latter is proportional to the number of vertices of a 3D model.

- Finally, transformation blending in our approach cannot go beyond certain intrinsic limitations of the LBS model, among which sophisticated deformation effects such as muscle bulges or skin wrinkles cannot be captured well. Several sound solutions have been proposed to alleviate this problem including multi-weight blending [Wang and Phillips 2002] and skinning correction [Lewis et al. 2000; Kry et al. 2002].

Within specific contexts, we can modify or extend our compression model for better performance. For example, the number of virtual bones per vertex in the virtual bone blending layer can be increased for smoother skinning. However, this will measurably affect performance, since the number of blending operations will increase by n (the number of vertices) when the number of virtual bones per vertex increases by one. In addition, we can also increase the number of blending layers, which will increase the number of combinations of master bone blending operations at an exponential rate. Nevertheless, increasing the number of layers would put additional overhead on skinning models, especially with common multi-pass implementation on GPUs. We believe that analyzing and extending our compression model in these directions will be useful for certain skinning applications.

6 Conclusions

We introduce a two-layer sparse compression approach to effectively compress the weights of the widely used linear blend skinning model. By employing virtual bones to cache transformation blending of master bones, our approach can significantly reduce computational cost, with insignificant loss of the accuracy of the original skinning model. The virtual bones allow users to optimally distribute and control the approximation errors on different regions of 3D models, while keeping our compressed skinning model friendly to parallel implementation on multi-core processors.

By formulating this weight compression problem as a sparse dictionary learning problem, we design a fast and simple solution to take advantage of the fairness property on the manifold. Our approach can be flexibly controlled by setting either an error threshold or a desired skinning performance (i.e., a maximum number of blending operations). It can directly work on an input bone-vertex weight map or utilize example poses to achieve a smaller approximation error (i.e., a better accuracy).

Through many qualitative and quantitative comparisons between our approach and selected state-of-the-art skinning weight reduction algorithms, we demonstrate the effectiveness and efficiency of our skinning compression and reduction approach. Compared with conventional linear blend skinning models with the same number of bones per vertex, our approach can approximate an input model with a much smaller error. Finally, our approach can be potentially extended to handle other skinning models including dual-quaternion blend skinning and cage-based skinning.

Acknowledgements

This research work is supported in part by NSF IIS-0914965. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the agencies. We would like to thank Xiao Cheng

for making the GPU implementation, and thank J.P. Lewis, Olin Johnson, and Chang H. Yun for their careful proofreading. We also thank Daniel Vlasic, Robert Sumner, and Jovan Popovic for providing the mesh sequences used in this work. Finally, we would like to thank the anonymous SIGGRAPH reviewers for their useful and constructive comments.

References

AHARON, M., ELAD, M., AND BRUCKSTEIN, A. 2006. K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation. *IEEE Transactions on Signal Processing 54*, 11 (Nov.), 4311–4322.

BARAN, I., AND POPOVIĆ, J. 2007. Automatic rigging and animation of 3d characters. *ACM Trans. Graph. 26* (July).

BOROSÁN, P., JIN, M., DECARLO, D., GINGOLD, Y., AND NEALEN, A. 2012. Rigmesh: automatic rigging for part-based shape modeling and deformation. *ACM Trans. Graph. 31*, 6 (Nov.), 198:1–198:9.

HASLER, N., THORMÄHLEN, T., ROSENHAHN, B., AND SEIDEL, H.-P. 2010. Learning skeletons for shape and pose. In *I3D'10*, 23–30.

JACOBSON, A., AND SORKINE, O. 2011. Stretchable and twistable bones for skeletal shape deformation. *ACM Trans. Graph. 30* (Dec.), 165:1–165:8.

JACOBSON, A., BARAN, I., POPOVIĆ, J., AND SORKINE, O. 2011. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph. 30*, 4 (July), 78:1–78:8.

JACOBSON, A., BARAN, I., KAVAN, L., POPOVIĆ, J., AND SORKINE, O. 2012. Fast automatic skinning transformations. *ACM Trans. Graph. 31*, 4 (July), 77:1–77:10.

JACOBSON, A., WEINKAUF, T., AND SORKINE, O. 2012. Smooth shape-aware functions with controlled extrema. *Comp. Graph. Forum 31*, 5 (Aug.), 1577–1586.

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

JOSHI, P., MEYER, M., DEROSE, T., GREEN, B., AND SANOCKI, T. 2007. Harmonic coordinates for character articulation. *ACM Trans. Graph. 26*, 3 (July).

JU, T., SCHAEFER, S., AND WARREN, J. 2005. Mean value coordinates for closed triangular meshes. *ACM Trans. Graph. 24*, 3 (July), 561–566.

KAVAN, L., AND SORKINE, O. 2012. Elasticity-inspired deformers for character articulation. *ACM Trans. Graph. 31*, 6 (Nov.), 196:1–196:8.

KAVAN, L., MCDONNELL, R., DOBBYN, S., ŽÁRA, J., AND O'SULLIVAN, C. 2007. Skinning arbitrary deformations. In *I3D'07*, 53–60.

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

KAVAN, L., COLLINS, S., AND O'SULLIVAN, C. 2009. Automatic linearization of nonlinear skinning. In *I3D'09*, 49–56.

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

KRY, P. G., JAMES, D. L., AND PAI, D. K. 2002. Eigenskin: real time large deformation character skinning in hardware. In *SCA'02*, 153–159.

LANDRENEAU, E., AND SCHAEFER, S. 2010. Poisson-based weight reduction of animated meshes. *Comput. Graph. Forum 29*, 6, 1945–1954.

LE, B. H., AND DENG, Z. 2012. Smooth skinning decomposition with rigid bones. *ACM Trans. Graph. 31*, 6 (Nov.), 199:1–199:10.

LEE, H., BATTLE, A., RAINA, R., AND NG, A. Y. 2007. Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems 19*. MIT Press, 801–808.

LEWIS, J. P., CORDNER, M., AND FONG, N. 2000. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *Proc. of ACM SIGGRAPH'00*, 165–172.

MAIRAL, J., BACH, F., PONCE, J., AND SAPIRO, G. 2010. Online learning for matrix factorization and sparse coding. *J. Mach. Learn. Res. 11* (Mar.), 19–60.

MALLAT, S., AND ZHANG, Z. 1993. Matching pursuits with time-frequency dictionaries. *Trans. Sig. Proc. 41*, 12 (Dec.), 3397–3415.

MARQUARDT, D. W. 1963. An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *SIAM Journal on Applied Mathematics 11*, 2, 431–441.

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

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

NGUYEN, H. 2007. *GPU gems 3 (first edition), chapter 4.4*. Addison-Wesley Professional.

NOCEDAL, J., AND WRIGHT, S. 2000. *Numerical Optimization*. Springer.

SCHAEFER, S., AND YUKSEL, C. 2007. Example-based skeleton extraction. In *SGP'07*, 153–162.

SEDERBERG, T. W., AND PARRY, S. R. 1986. Free-form deformation of solid geometric models. *SIGGRAPH Comput. Graph. 20*, 4 (Aug.), 151–160.

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

VLASIC, D., BARAN, I., MATUSIK, W., AND POPOVIĆ, J. 2008. Articulated mesh animation from multi-view silhouettes. *ACM Trans. Graph. 27* (August), 97:1–97:9.

WANG, X. C., AND PHILLIPS, C. 2002. Multi-weight enveloping: least-squares approximation techniques for skin animation. In *SCA'02*, 129–138.

WARREN, J., SCHAEFER, S., HIRANI, A. N., AND DESBRUN, M. 2004. Barycentric coordinates for convex sets. Tech. rep., Advances in Computational and Applied Mathematics.

ZOU, H., AND HASTIE, T. 2005. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society, Series B 67*, 301–320.