Fast and Deep Deformation Approximations

STEPHEN W. BAILEY, University of California, Berkeley
DAVE OTTE, DreamWorks Animation
PAUL DILORENZO, DreamWorks Animation
JAMES F. O'BRIEN, University of California, Berkeley

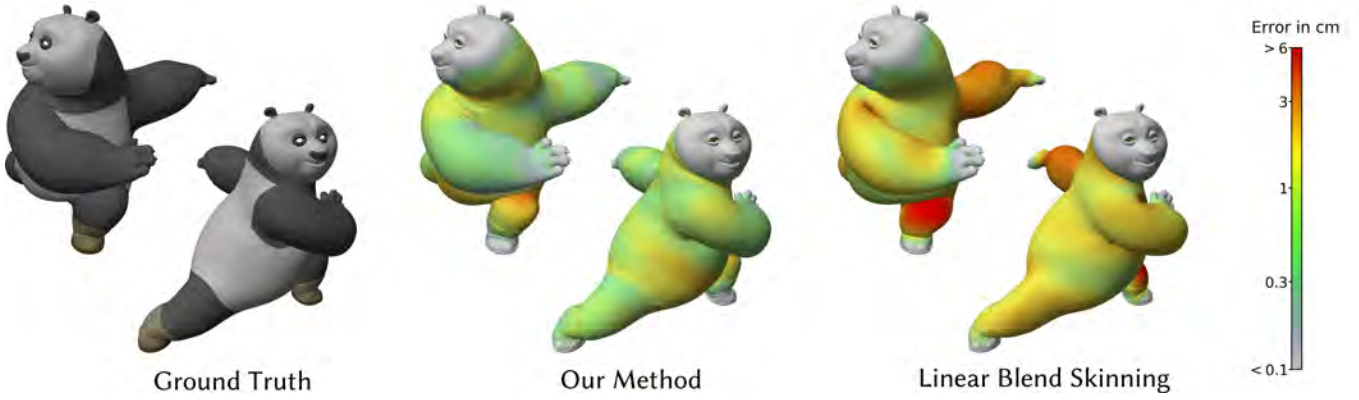


Fig. 1. Comparison of a deformed mesh using a fully evaluated rig, our fast deformation approximation, and linear blend skinning. The meshes are colored to indicate the distance error for each vertex compared with the ground truth mesh.

Character rigs are procedural systems that compute the shape of an animated character for a given pose. They can be highly complex and must account for bulges, wrinkles, and other aspects of a character's appearance. When comparing film-quality character rigs with those designed for real-time applications, there is typically a substantial and readily apparent difference in the quality of the mesh deformations. Real-time rigs are limited by a computational budget and often trade realism for performance. Rigs for film do not have this same limitation, and character riggers can make the rig as complicated as necessary to achieve realistic deformations. However, increasing the rig complexity slows rig evaluation, and the animators working with it can become less efficient and may experience frustration. In this paper, we present a method to reduce the time required to compute mesh deformations for film-quality rigs, allowing better interactivity during animation authoring and use in real-time games and applications. Our approach learns the deformations from an existing rig by splitting the mesh deformation into linear and nonlinear portions. The linear deformations are computed directly from the transformations of the rig's underlying skeleton. We use deep learning methods to approximate the remaining nonlinear portion. In the examples we show from production rigs used to animate lead characters, our approach reduces the computational time spent on evaluating deformations by a factor of 5×-10×. This significant savings allows us to run the complex, film-quality rigs in real-time even when using a CPU-only implementation on a mobile device.

CCS Concepts: • **Computing methodologies** → **Neural networks**; **Animation**;

Authors' addresses: Stephen W. Bailey, University of California, Berkeley; Dave Otte, DreamWorks Animation; Paul DiLorenzo, DreamWorks Animation; James F. O'Brien, University of California, Berkeley.

Additional Key Words and Phrases: character rigs, deep learning, mesh deformations

ACM Reference Format:
Stephen W. Bailey, Dave Otte, Paul DiLorenzo, and James F. O'Brien. 2018. Fast and Deep Deformation Approximations. *ACM Trans. Graph.* 37, 4, Article 119 (August 2018), 12 pages. https://doi.org/10.1145/3197517.3201300

1 INTRODUCTION

The level of detail that can be included in character rigs for interactive applications such as video games and virtual reality is limited by computational costs. These types of rigs need to run at interactive rates, and therefore need to be evaluated as quickly as possible. Because of this limitation, real-time character rigs often lack a high level of detail. In contrast, film-quality character rigs are not limited by hard computational constraints and their mesh deformations appear more detailed and complex. Unfortunately, film-quality character rigs are not suitable for real-time applications. A single film-quality rig might be able to run at interactive rates on a high-end machine, but typically only after tremendous effort has been spent to optimize the rig, parallelize its evaluation, and shift parts of the computation to the high-end machine's GPU. We would like to use these high quality rigs to increase the detail of characters in interactive applications, particularly those running on modest computing platforms such as mobile devices or game consoles. However, directly plugging these computationally intensive rigs into an interactive application is generally infeasible. The performance increases that come as hardware improves over time is unlikely to bring film-quality rigs to real-time, because as performance improves, the level of complexity and fidelity one expects in a film rig also tends to increase. Furthermore, many of these real-time applications need to run on modest computing hardware

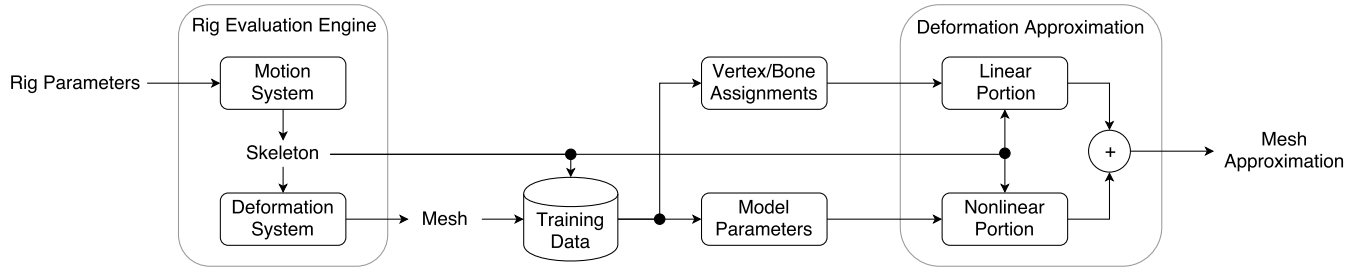


Fig. 2. Our approximation method learns the deformation system of a character rig by splitting the mesh deformation into a linear portion (Section 3.1.1) and a nonlinear portion (Section 3.1.2). The linear approximation uses rigid skinning, and the nonlinear approximation is learned from a set of training examples generated from the original rig evaluation function (Section 3.4).

such as phones or game consoles. Assuming the application is run on a fully loaded high-end machine is reasonable for users who are professional animators, but not for most other users.

To address this limitation, we present a data-driven approach to learn a computationally less expensive approximation for character rigs. Our approximation method reduces the computation enough to evaluate film-quality rigs in real-time on mobile devices. An overview of the method is outlined in Figure 2. Most character rigs are designed with two main components: a skeletal motion system and a deformation system [McLaughlin et al. 2011]. The skeletal motion system is responsible for mapping the input rig parameters that specify a pose to a configuration of the character's skeleton which is composed of bones and joints. The deformation system then maps the skeleton configuration to the final mesh geometry. The deformation system determines how the character's skin moves and includes effects such as muscle bulging and and skin wrinkles. In most character rigs, the deformation computation typically requires the most time and thus is a bottleneck.

We propose a method to approximate the deformation system faster than the original. Given an input skeleton, our system can significantly speed up the overall rig evaluation by dramatically improving the speed of the deformation system. Furthermore, our method achieves a high level of accuracy such that errors are not visually apparent.

2 RELATED WORK

A common method for rigging a character involves first defining an underlying skeleton and then deforming the character's mesh based on the positions and orientations of the skeleton's bones. One of the fastest methods to compute the deformation from the skeleton is linear blend skinning or skeleton subspace deformation as described in [Magnenat-Thalmann et al. 1988]. This method computes the deformation of a mesh from a rest pose as a weighted sum of the skeleton's bone transformations applied to each vertex. Although linear blend skinning can compute deformations quickly, these deformations can suffer from volume loss and the "candy wrapper" problem. Prior research has explored methods to solve the shortcomings of linear blend skinning. Multi-weight enveloping [Wang and Phillips 2002] addresses these problems by using using blending weights for each entry in the bone transformation matrices, and the weights are automatically learned from example poses of the rig. Quaternion-based methods [Hejl 2004], such as spherical blend skinning [Kavan and Žára 2005], are other approaches that address the limitations of linear blend skinning without significantly increasing the computational cost of the deformation.

Although linear blend skinning provides a fast method to compute mesh deformations, there are some types of deformation that are challenging to express with this approach. For example, skin slide, muscle bulges, and cloth wrinkles are difficult to achieve using only linear blend skinning. These effects, however, can be achieved using additional skinning methods at the cost of additional computation. Some of these methods include pose space deformations [Lewis et al. 2000; Sloan et al. 2001] and cage-based deformations [Joshi et al. 2007; Ju et al. 2008]. Realistic character deformations can also be computed through physics-based approaches [Capell et al. 2002], and highly realistic results can be achieved by accurately modeling the underlying anatomy of a character [Lee et al. 2009].

Skinning decomposition is the process of identifying bone transformations and bone-vertex weights to best approximate a given animation with linear blend skinning. Proposed solutions to the skinning decomposition problem provide a compressed representation of the animation as well as an efficient method to play back the animation in real-time. Prior research has explored methods to approximate arbitrary deformations [James and Twigg 2005; Kavan et al. 2007, 2010; Le and Deng 2012]. These methods seek to fit a bone structure to a series of mesh animations and optimize the bone influences for each vertex to best reconstruct the original animation. Alternatively, a volumetric approximation can be fitted to an animation using sphere-meshes [Thiery et al. 2016], and linear blend weights can be quickly computed with respect to the underlying spheres. With these methods, large animations can be efficiently played back using hardware acceleration, and the deformed meshes can be stored in a compressed format given the fitted bone structure or sphere-meshes. One drawback of these approaches is that new animations cannot be quickly fitted to the rigs because the bones are optimized for a specific set of deformations. Furthermore, an animator would need to learn to use the fitted bone structure in order to author new animations. Example-based skinning methods have been developed [Feng et al. 2008; Mohr and Gleicher 2003; Mukai and Kuriyama 2016; Wang et al. 2007] that uses the original skeleton from a character rig. These methods use training examples to learn a deformation model that can approximate mesh deformations given new skeleton poses not seen during model training.

A linearization method [Kavan et al. 2009] computes deformations by adding virtual bones to approximate nonlinear deformations in a character. This method utilizes the underlying skeleton of a rig, which allows new skeletal motion to be easily applied to the rig. However, that algorithm works only for deformations computed through a differentiable skinning technique such as dual quaternion blend skinning. Our method, on the other hand, allows for more general deformations and only assumes that the deformations can be computed as a function of the character's skeleton.

Because these skinning decomposition methods compute a compact representation of an animation with bones using linear blend skinning, the deformation evaluation is fast and efficient. However, editing the compressed animations can be difficult because the computed bones are not organized in any meaningful hierarchy. Some work [De Aguiar et al. 2008; Hasler et al. 2010; Le and Deng 2014; Schaefer and Yuksel 2007] has addressed this limitation by extracting a skeleton structure with joints while also computing bone transformations and vertex-bone weights for an example animation. By providing a hierarchical skeletal system, an animator can more easily edit an existing motion, but extra computation would be required to fit the skeleton to a new animation of the same mesh. These methods approximate mesh deformations when existing animations are provided with or without an underlying skeleton. Our method, in contrast, approximates deformations for a mesh without any example animations; however, it does require that the mesh have an underlying skeleton. We use the underlying skeleton of a character rig without modification, which lets an animator author new poses using the familiar original rig while benefiting from our fast approximate evaluation.

All of these previous skinning decomposition methods seek to find compact representation of an animation using bones with linear blend skinning. Because linear blending is fast, the compressed animations can be computed quickly, but the limitations of linear blend skinning can cause inaccuracies and undesirable artifacts. To improve the speed of linear blend skinning, a sparseness constraint must be imposed on the vertex-bone weights. In some cases where a vertex is influenced by a large number of bones, this sparseness constraint can lead to large inaccuracies in the approximation. One proposed solution [Le and Deng 2013] to this problem is to compress an animation with a two-layer linear blend skinning model, which accurately reduces the computational cost of evaluating dense skinning weights.

Our approach is also based on decomposing a deformation with linear blend skinning. To reduce the computational cost, we assign each vertex to a single bone, but to overcome the limitations of this skinning method, we also propose an extra nonlinear step, which is modeled as a function of all the bones that influence a vertex. Although our approach does require more computation than a rig using linear blend skinning, we show that the deformations can still be computed efficiently for real-time applications and that our approximation approach can reproduce deformations to a high level of accuracy on film-quality rigs as seen in our results.

3 METHOD

The rig function $\mathbf{r}(\mathbf{p})$ maps a set of artist-level rig parameters, denoted with $\mathbf{p}$, to a deformed polygonal mesh. We follow a similar notation for the rig function as described in [Hahn et al. 2012], and we assume that the rig function is a black-box. We further assume that the topology of the mesh is constant for all possible parameters $\mathbf{p}$, which allows us to express the rig function as $\mathbf{V} = \mathbf{r}(\mathbf{p})$ where $\mathbf{V}$ is a list of the vertex positions in the mesh. An intermediate step of the rig function computes the skeleton $\mathbf{S}$ of a character. The skeleton's configuration is specified by a set of linear transformations and translations for each bone in the skeleton. Specifically for a skeleton with m bones, $\mathbf{S} = [\mathbf{X}_1, \mathbf{t}_1, \mathbf{X}_2, \mathbf{t}_2, \ldots, \mathbf{X}_m, \mathbf{t}_m]$ where $\mathbf{X}_j$ is the 3×3 linear transformation matrix of bone j and $\mathbf{t}_j$ is the translation of bone j. The transformations and translations are expressed in a global coordinate frame. We further assume that the rig function can be expressed as the composition of two functions: a skeletal motion system mapping rig parameters to a skeleton and a deformation system mapping a skeleton to vertex positions. The skeletal motion system is denoted by $\mathbf{S} = \mathbf{m}(\mathbf{p})$, and the deformation system is denoted by $\mathbf{V} = \mathbf{d}(\mathbf{S})$. Composing these two systems, the rig function can be expressed as $\mathbf{r}(\mathbf{p}) = (\mathbf{d} \circ \mathbf{m})(\mathbf{p})$.

Our method provides an approach to approximate the deformation function $\mathbf{d}(\mathbf{S})$ by decomposing the function into two parts: a linear computation and a nonlinear computation. The linear portion uses rigid rotations and translations to deform the vertices in the mesh according to the bone transformations in the skeleton. This computation is fast, but the resulting mesh is visibly different from the target mesh $\mathbf{V} = \mathbf{d}(\mathbf{S})$. To correct this difference the nonlinear component utilizes a universal function approximator to estimate the remaining residual error between the mesh obtained from rigid rotations and the target mesh. The nonlinear function approximator learns from a set of randomly generated skeletons and corresponding deformed meshes that are computed offline using the rig function $\mathbf{r}(\mathbf{p})$.

3.1 Deformation Approximation

We view the rig function as a deformation applied to a mesh in some rest pose. This deformation has linear and nonlinear components, and when combined the two components fully describe the deformation applied to the mesh.

3.1.1 Linear Skinning. The linear deformation can be applied directly from the input skeleton by multiplying the vertices in the mesh with the bone transformation matrices. In our skinning method, we assign each vertex to a single bone where the vertex k is assigned to bone b_k. Starting with a mesh in a rest pose $\mathbf{V}^0$ and the corresponding skeleton $\mathbf{S}^0$, the linear deformation by a new skeleton $\mathbf{S}$ for vertex k can be computed as

$$\hat{\mathbf{d}}_k(\mathbf{S}) = \mathbf{X}_{b_k} \left(\mathbf{X}^0_{b_k}\right)^{-1} \left(\mathbf{v}^0_k - \mathbf{t}^0_{b_k}\right) + \mathbf{t}_{b_k} \tag{1}$$

where $\mathbf{X}^0_{b_k}$ and $\mathbf{t}^0_{b_k}$ are the transformation matrix and translation vector for bone b_k in the skeleton $\mathbf{S}^0$ of the rest pose, and $\mathbf{v}^0_k$ is the position of vertex k in the mesh of the rest pose.

We assume that we only have black-box access to the deformation function, and we therefore cannot rely on information about

the character rig to identify vertex-bone assignments. Instead, we assign each vertex to a single bone that best explains the vertex's deformation across a set of example poses. The bone assignment b_k is determined by selecting the bone which minimizes the least squares error of the rigid transformation of the vertex by the bone.

The linear deformation is visibly different from the target deformation where $||\mathbf{d}(\mathbf{S}) - \hat{\mathbf{d}}(\mathbf{S})||^2 \gg 0$. We view this residual error as a nonlinear function, which allows for sophisticated stretching, compression, and volume preservation. These features cannot be handled by a linear transformation alone.

3.1.2 Nonlinear Deformation. The residual $\mathbf{d}(\mathbf{S}) - \hat{\mathbf{d}}(\mathbf{S})$ expresses the error in terms of the global coordinate system and thus depends on global transformations of the skeleton. Ideally, we would like to express the residual in such a way that the error for some vertex k is only affected by a local neighborhood of bones in the skeleton. By representing the residual locally for each vertex, the error function becomes easier to learn because the residual for each vertex no longer depends on the transformations of every ancestor bone in the skeleton hierarchy.

To specify the residual locally, we define the nonlinear deformation function for some vertex k as follows

$$\mathbf{f}_k(\mathbf{S}) = \left(\mathbf{X}_{b_k}\right)^{-1}\left(\mathbf{d}_k(\mathbf{S}) - \mathbf{t}_{b_k}\right) - \left(\mathbf{X}^0_{b_k}\right)^{-1}\left(\mathbf{v}^0_k - \mathbf{t}^0_{b_k}\right) \quad (2)$$

where $\mathbf{d}_k(\mathbf{S})$ is the position of vertex k as computed from the original rig deformation function. This function removes the transformation of the rest pose from the vertex $\mathbf{v}^0_k$ and the transformation of the deformed pose from the deformed vertex $\mathbf{d}_k(\mathbf{S})$. The difference of these two positions gives us the nonlinear deformation of the vertex in the coordinate space of the bone b_k.

The deformation function can now be expressed as

$$\mathbf{d}_k(\mathbf{S}) = \mathbf{X}_{b_k}\left(\left(\mathbf{X}^0_{b_k}\right)^{-1}\left(\mathbf{v}^0_k - \mathbf{t}^0_{b_k}\right) + \mathbf{f}_k(\mathbf{S})\right) + \mathbf{t}_{b_k} \quad (3)$$

We denote our approximation with model parameters θ as $\mathbf{n}_k(\mathbf{S};\theta) \approx \mathbf{f}_k(\mathbf{S})$, and the deformation approximation $\tilde{\mathbf{d}}_k(\mathbf{S};\theta)$ can be expressed as the sum of the linear and nonlinear functions

$$\tilde{\mathbf{d}}_k(\mathbf{S};\theta) = \hat{\mathbf{d}}_k(\mathbf{S}) + \mathbf{X}_{b_k}\mathbf{n}_k(\mathbf{S};\theta) \quad (4)$$

The optimal model parameters $\hat{\theta}$ are estimated by minimizing the squared error loss over a set of n training examples

$$\hat{\theta} = \arg\min_{\theta} \sum_{i=1}^{n} \left\| \mathbf{d}_k(\mathbf{S}^i) - \tilde{\mathbf{d}}_k(\mathbf{S}^i;\theta) \right\|^2 \quad (5)$$

Instead of using one function approximator per vertex, we group the vertices into subsets and train a function approximator that outputs each vertex in the subset. In order to take advantage of the local deformation defined in $\mathbf{f}(\mathbf{S})$, we separate the vertices of the mesh into subsets P_i based on the bones that they are assigned to such that $P_i = \{k \mid b_k = i\}$. By dividing the vertices into sets this way, the nonlinear deformations for vertices in set P_i are defined in the same coordinate system, which makes the deformation function easier to learn.

3.2 Implementation

Because the function $\mathbf{f}_{P_i}(\mathbf{S})$ can be highly nonlinear, we need a model that is capable of learning arbitrary continuous functions. Feed-forward neural networks are universal function approximators [Hornik 1991] that can learn such functions. Given any continuous function, a neural network of sufficient size can approximate the function arbitrarily closely. This property thus makes neural networks good candidates for approximating the nonlinear deformation component of the rig function.

In our experiments, we trained each neural network with two fully connected hidden layers and a dense output layer. In the following section, we describe how we determine the number of layers and the number of hidden units per layer. The hidden layers used the tanh nonlinearity, and the output layer was a dense linear layer. Other activation functions such as the rectified linear unit [Glorot et al. 2011] could have been used, but we only evaluated tanh in this paper. We trained each network on inputs of the bone transformation matrices and the translation vectors given in the frame of reference of the parent bone. The transformation matrix for bone j with parent p is given as $\mathbf{X}_p^{-1}\mathbf{X}_j$, and the translation vector is given as $\mathbf{X}_p^{-1}(\mathbf{t}_j - \mathbf{t}_p)$. The root bone is not provided as an input. In total, each bone contributed 12 inputs to the neural network. The models were trained using the Adam optimization method [Kingma and Ba 2014] with the following values for the parameters: $\alpha = 0.01$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$.

3.3 Model Sparsification

The method presented so far works well to approximate the nonlinear deformation function, but similar results can be achieved with less computation per neural network. We can increase the speed of the approximation without significantly affecting the accuracy by identifying and removing extra computations in the models. We explored four approaches to reduce the size of the approximation model: reduce the size and number of the hidden layers in each neural network, reduce the dimension of the inputs, reduce the dimensions of the outputs, and reduce the total number of neural networks evaluated. Reducing the size and number of hidden layers can be done empirically, and we found that two layers each of 128 nodes in each neural network worked well for our character rigs.

Feed-forward neural networks are composed of a series of dense layers where the output $\mathbf{x}_{i+1}$ of layer i is used as the input of the next layer. The output for some layer i is computed as follows

$$\mathbf{x}_{i+1} = f(\mathbf{W}_i\mathbf{x}_i + \mathbf{b}_i) \quad (6)$$

where $\mathbf{W}_i$ and $\mathbf{b}_i$ are unknown parameters that are learned when the model is trained. The function $f(\mathbf{x})$ is a nonlinear function applied element-wise to the components of the input vector. The most time-consuming part of Equation 6 is the matrix-vector product $\mathbf{W}_i\mathbf{x}_i$. If the matrix $\mathbf{W}_i$ is $m \times n$, then the complexity of calculating the product is $O(mn)$. Therefore, to reduce the computational complexity of evaluating the neural network models, we need to reduce the sizes of the weight matrices $\mathbf{W}_i$.

3.3.1 Input Reduction. Evaluating the first layer of the network involves a large amount of computation because the dimension of

the input is large. For some set of vertices P_i, the nonlinear deformation $\mathbf{f}_{P_i}(\mathbf{S})$ is expressed locally with respect to bone i according to Equation 2. Because the vertices P_i are primarily deformed by bones near bone i, this deformation function depends only on the local bones near the vertices in this set and does not require all of the bones as input. This invariance to some of the input bones is a direct consequence of the formulation of $\mathbf{f}_{P_i}(\mathbf{S})$ in a local coordinate space of bone i.

Assuming that we have access to the original rig function $\mathbf{r}(\mathbf{p})$ and the skeleton $\mathbf{S} = \mathbf{m}(\mathbf{p})$, we can identify which bones most affect the vertices in P_i. Starting in an arbitrarily selected example pose $\mathbf{p}'$, we perturb rig controls that affect the bones one at a time and record which bones caused a change in the function $\mathbf{f}_{P_i}(\mathbf{S})$. This process is repeated with multiple example poses and with large perturbations to ensure that all bones affecting vertices in P_i are identified. We define a subset of the skeleton $\mathbf{S}_{P_i}$ as the set of all bones that influence any of the vertices in P_i. We then use this subset of bones as the input to the model approximating the nonlinear deformation function for these vertices. The rigs we tested contained between 100 and 200 bones. After reducing the number of input bones, each set of vertices tended to have around 20 bones that contributed to their deformation. By using this reduced input set, the computational cost of the first layer for each model can be significantly reduced.

3.3.2 Output Reduction. Next, we consider the size of the output layer. The output contains 3 values per vertex, and for the rigs that we tested, there were on the order of hundreds of vertex positions that each neural network approximated. Unlike the input layer, each dimension of the output needs to be predicted. However, these outputs are highly correlated with each other. With this in mind, we propose using a linear dimensionality reduction method to reduce the size of the output. Our approach is similar to the approach used by Laine et al. [2017] in which they use PCA to initialize the final layer in their network to output vertex positions.

With the data used to train each model, we run PCA on each matrix $\mathbf{V}_{P_i}^{1\cdots n}$ containing all of the vertex positions for set P_i across all n poses in the training set. The matrix $\mathbf{V}_{P_i}^{1\cdots n}$ is a $3|P_i| \times n$ matrix where there are $|P_i|$ vertices and n training examples. PCA gives us a transformation $\mathbf{T}$ that maps the set of vertex positions to a lower dimensional space. Next, we need to determine how many principal components to use in the linear transformation $\mathbf{T}$. Keeping more components will increase the accuracy of the model at the cost of more computation time. We decide the number of components to keep by finding the minimum amount that keeps the reconstruction error $\left\|\mathbf{V}_{P_i}^{1\cdots n} - \mathbf{T}^T\mathbf{T}\mathbf{V}_{P_i}^{1\cdots n}\right\|_F^2$ below some user-specified threshold.

In our experiments, we found that keeping the average per-vertex distance error below 0.03 cm was sufficient to maintain the visual accuracy of the approximation without adding too many principal components to the transformation. This threshold choice lead on average to 20-30 principal components per model, which provided a reasonable balance between speed and accuracy. Once we found the transformation $\mathbf{T}$, we appended it to the end of the neural network model as a final dense layer with a linear activation. When the model is trained, the weights of this last layer are not optimized so that the transformation is maintained.

3.3.3 Model Count Reduction. One final approach to reduce the computation of the approximation is to reduce the total number of neural networks in the approximation model. In our method as currently described so far, one model is trained per bone; however, we found that some bones had few vertices assigned to them. As a result, we were training some models to predict the deformation of a small set of vertices. These neural networks can be removed, and their vertices can be reassigned to other bones.

To remove networks approximating small subsets of vertices, we greedily removed the bone with the fewest vertices assigned to it and iteratively recomputed the vertex subsets P_i. We continued this process until the average vertex assignment error

$$e = \sum_{i=1}^{n} \left\|\mathbf{V}^i - \hat{\mathbf{d}}(\mathbf{S}^\mathbf{i})\right\|_F^2 \tag{7}$$

grew larger than some pre-defined threshold. Before removing any bone, we recorded the best average vertex assignment e_0 error given by Equation 7. Next, we removed bones one at a time. In each iteration, the bone with the fewest number of vertices was removed, and the vertices assigned to that bone were reassigned to the next best bone that minimized the error.

At each iteration, we recomputed the assignment error e_i, and we stopped this procedure when $e_i > \tau e_0$ for some scaling factor $\tau > 1$. In our experimental rigs we found that values of $\tau \in [1.1, 1.5]$ worked well. Higher values of τ will lead to fewer models that need to be trained, but fewer models could lead to larger approximation errors. If a small value of τ is chosen, then more models will be used, but the approximation errors will be smaller. Thus, the choice of τ provides a trade-off between speed and accuracy in the approximation.

3.4 Data Generation

The choice of training data is important for the rig approximator's accuracy when run on test inputs. Feed-forward neural networks do not extrapolate well from training data, and therefore, the training data needs to span the range of all possible poses that could be expected as inputs when the approximator is used. However, if the training set includes a large range of motion with improbable poses such as arms rotated into the torso or body parts stretched to twice their length, then these types of poses would represent large deformations that the approximator would need to learn. As a result, the neural network would learn these large deformations while sacrificing accuracy for smaller deformations. However, we desire a high accuracy for these smaller deformations because they are more likely to be encountered when the model is evaluated.

Here, we describe a method to create a data set that contains all of the probable poses while avoiding poses with large deformations that are unlikely to occur in an animation. First, we consider each joint in the skeleton independently. For each joint, we manually identify a reasonable range of motion for the rotation and scaling. For example, we might specify the range of the knee joint from 0 to 150 degrees. We define a range for each joint in the skeleton and generate new poses by randomly sampling independently from each

joint range. Each value is sampled from a Gaussian distribution with 1.5 standard deviation aligned with and scaled to the specified range. Specifically for some rig parameter with a range $[a, b]$, the parameter values are drawn from the following Gaussian distribution:

$$\mathcal{N}(0.5 \cdot (a + b), 1.5 \cdot (b - a)) \tag{8}$$

We re-sample values that lie outside of the range $[a, b]$. This sampling method ensures that the full range of motion for each joint is contained in the training set. Samples near the ends of the range of motion occur in the data set less frequently. If we assume that poses near the ends of the joint range create poses that an animator typically will not use, then because there are fewer of these examples in the training set, the approximator will focus on learning the deformations near the middle of the range of motion.

Our sampling method creates poses that globally appear invalid. Because our approximation method learns deformations locally, the global appearance of the character is less important than the local deformations around the joints of the character. Because each joint is sampled within the user-defined range of motion, meaningful local deformations of the mesh are contained in the samples, and our approximator can accurately learn from these example poses. Figure 3 shows several examples of poses generated by our method.

Other sampling methods could be used where appropriate. For example, a character that has a walking mode and flying mode that it switches between could use bimodal sampling. Additionally, semantic knowledge about how a character moves could be used for customized sampling. If example animations for the character are available, then supersampling methods could be used [Holden et al. 2017] to generate example poses similar to the animation.

4 RESULTS

We are interested in two key aspects of our deformation approximation: model accuracy and model speed. Model accuracy is important because we want to minimize the visual differences between the approximated mesh and the original deformed mesh. If the approximation is noticeably different than the original, then this method might not be suitable for all applications. The speed of the approximation is also important. On the rigs that we tested, our method took significantly less time to evaluate the rig compared with the original rig function. With a fast rig approximation, a highly complicated character can be evaluated at interactive rates even on low-end machines and mobile devices.

We approximated the deformation functions of four film-quality character rigs: Po, Shifu, and Tigress from *Kung Fu Panda 3*, and Astrid from *How to Train Your Dragon 2*. All of these character rigs were originally optimized to run on high-end machines. Table 1 shows the size of the models trained for all four rigs including the total memory required to evaluate the approximation models. In all of the approximators, we used two nonlinear hidden layers with the tanh activation function, and each hidden layer consisted of 128 nodes. We found that generating between 10,000 and 20,000 example poses was sufficient to train accurate approximation models for each rig.

Table 1. Statistics of the approximation models trained for the character rigs.

	Tigress	Shifu	Astrid	Po
Vertices	16,206	14,706	168,635	13,800
Character Height	182 cm	86 cm	194 cm	191 cm
Models	67	73	45	40
Avg. PCs used	22.6	19.5	24.5	29.7
Avg. input bones	26.9	22.3	14.6	57.75
Model memory size	11.5 MB	10.7 MB	67.5 MB	27.5 MB

Table 2. Mean and max approximation errors for each model tested on a walk cycle.

	Tigress	Shifu	Astrid	Po
Mean error	0.087 cm	0.016 cm	0.104 cm	0.143 cm
Max error	2.78 cm	1.10 cm	2.16 cm	4.80 cm

4.1 Model Accuracy

To evaluate the accuracy of our models, we measured the average per-vertex distance error of the approximated mesh as well as the largest single vertex error. We are interested in the largest error because even if the average error is small, a single misplaced vertex could create undesirable results. For each character, we evaluated a walk-cycle animation and computed the errors across all frames of the animation. The approximation errors are reported in Table 2. Figure 6 shows the distribution of vertex errors for each walking animation. From these plots, we observe the number of vertices falls off roughly exponentially with the distance error.

Figure 4 shows a side-by-side comparison of the mesh generated through the full rig evaluation and the approximated mesh for a frame of Astrid walking. Our method does not handle facial animation, and for each tested animation, we turned off all face controls. A region with large error in the approximation is found in the middle of Astrid's skirt. The deformation of this area is controlled by both legs, which appears to cause some inaccuracy in the approximation.

We additionally evaluated our method on more dynamic motions that contain poses near the edge or beyond the range of motion that the approximation models were trained on. We tested animations of martial arts moves for both Shifu and Tigress. The mean and max errors are presented in Table 3, and the distribution of errors is shown in Figure 7. We found that the largest errors tend to occur in the legs during kicking motions. Specifically, the legs are stretched beyond what the approximator was trained on. Because the model did not learn from example poses with a large amount of stretch, it fails on this particular pose from the animation. However, because the error is still small relative to the scale of the character and because the leg is stretched for a brief, dynamic moment, the approximation error is barely noticeable when viewing the animation. Figure 5 shows the results of our approximation on a frame of animation with the stretched leg.

4.2 Comparison

We compare our approach with linear blend skinning (LBS) and the rotational regression (RR) method of [Wang et al. 2007]. Like

Fig. 3. Example poses of Po and Shifu created by our data generation method. The poses do not look like anything an artist would create, but the local deformations of the mesh are still meaningful.

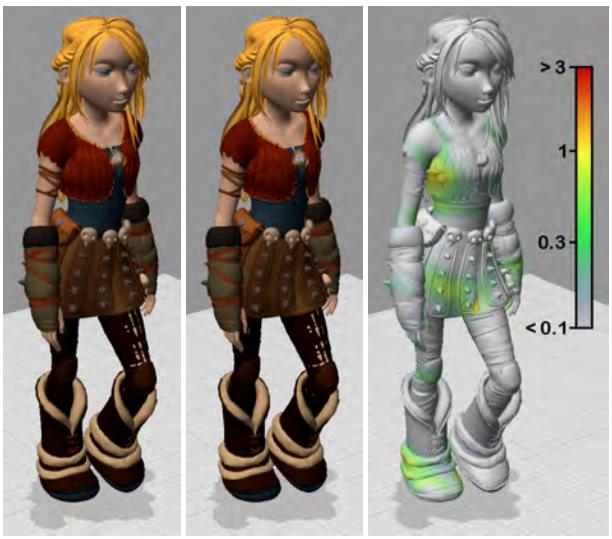


Fig. 4. Side-by-side comparison of the ground truth mesh (left) and the approximation (center). The vertices of the approximated mesh are colored to indicate the per-vertex distance error (right). Errors above and below the range of the scale are clamped to the ends of the color range. All distances are measured in centimeters.

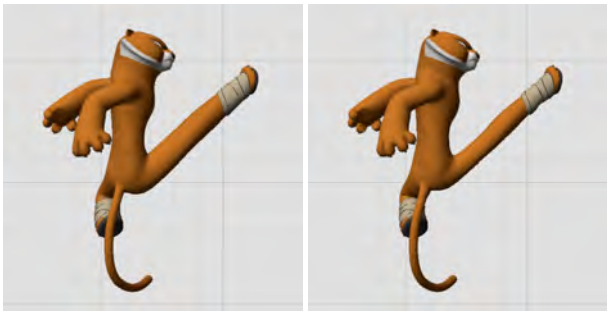
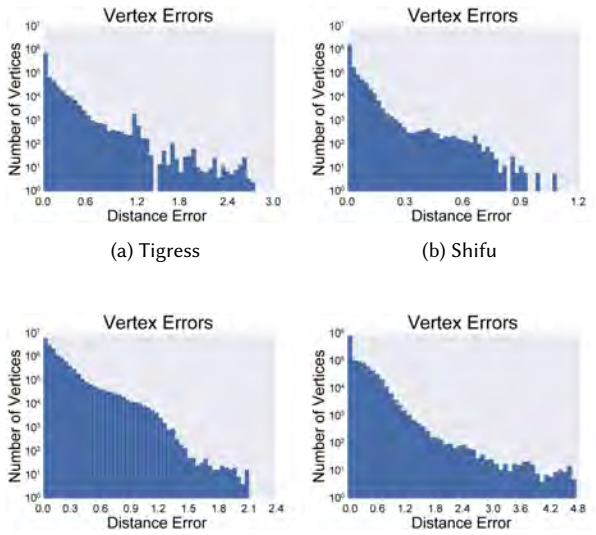


Fig. 5. Side-by-side comparison of the ground truth mesh (left) and the approximation (right) for a frame of the dynamic motion of Tigress. The most noticeable difference is the shape of the stretched leg.

Table 3. Mean and max approximation errors dynamic animations tested on the Tigress and Shifu rigs.

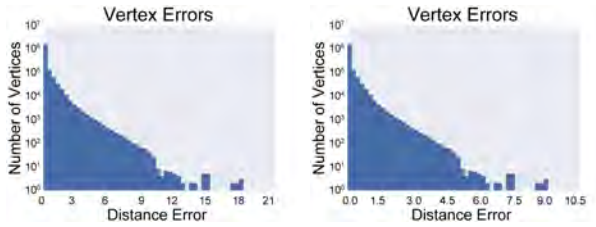
	Tigress	Shifu
Mean error	0.208 cm	0.041 cm
Max error	19.17 cm	8.14 cm



(a) Tigress (b) Shifu

(c) Astrid (d) Po

Fig. 6. Log histogram plot of the distribution of per-vertex approximation errors for the walk cycle animations. All distances are measured in centimeters.



(a) Tigress (b) Shifu

Fig. 7. Log histogram plot of the distribution of per-vertex approximation errors for the dynamic animations for Tigress and Shifu. All distances are measured in centimeters.

our algorithm, these two other methods can approximate the deformation function given any possible input pose. Thus, we can directly compare the accuracy of our approximation with these two methods.

For LBS, we estimate the bone weights for each vertex using example-based skinning decomposition [Mohr and Gleicher 2003]. We do not add any additional bones to the skeleton when we solve

for the vertex weights. We experimented with many different values of the sparseness constraint K, which determines the total number of joints that can influence each vertex.

The rotational regression method described in [Wang et al. 2007] approximates the deformation gradients of the triangles in a character's mesh. Their model uses linear regression to approximate deformation gradients from bone transformations. In their method, they only used at most two bones to approximate the gradients. However, we found that using only two bones was insufficient to approximate the deformations of our character rigs. We thus show results with the original method using $K = 2$ bones per triangle and with the method using $K = 15$ bones to approximate the gradients. To achieve the most accurate results, our implementation does not include the reduced formulation presented in their paper.

In Figure 8, we show a comparison of deformations approximated with linear blend skinning, rotational regression, and our method. We show the deformations using LBS computed with different values of K ranging from $K = 1$ to $K = m$ where m is the total number of bones in the skeleton as well as RR using the original $K = 2$ as well as $K = 15$ bones as inputs. Visually, we can see that our method outperforms LBS and RR for this example pose. For LBS with $K > 1$, the deformations suffer from significant volume loss in the legs and the arms. In our method, this volume loss problem is not apparent, and the approximated deformation is closer to the original mesh than any of the meshes generated with LBS. In the case of $K = 1$ (Figure 8a), volume loss is not seen in the deformation because no transformations matrices are blended together. However, most of the errors occur from the vertices moving tangentially to the surface of the target deformation as shown in Figure 9. This type of error in the deformation would cause undesirable stretching and distortion of any texture applied to the mesh.

We further compared our method with LBS and RR for each walking and kungfu animation. In Table 4, we present the average approximation errors for all of the animations using our method compared with the other methods. Our algorithm learns from a separate training set described in Section 3.4 while LBS and RR are trained on the same animation on which the errors are measured, a situation which benefits LBS and RR in the comparison.

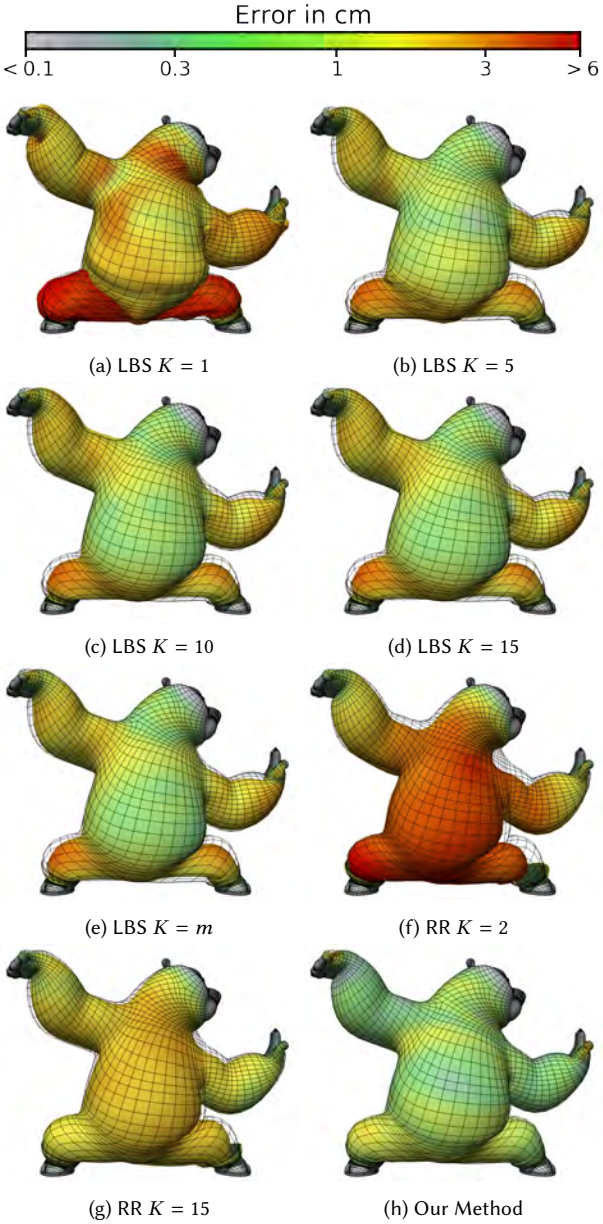
Following [Wang et al. 2007], we also compute the envoloping error (EE)

$$EE = 100 \sqrt{\frac{\sum_{i=1}^{N} \sum_{k=1}^{V} \left\| \mathbf{v}_k^i - \hat{\mathbf{v}}_k^i \right\|^2}{\sum_{i=1}^{N} \sum_{k=1}^{V} \left\| \mathbf{v}_k^i - \mathbf{c}\left(\mathbf{v}_k^0\right) \right\|^2}} \tag{9}$$

where $\hat{\mathbf{v}}_k^i$ is the approximated vertex position and

$$\mathbf{c}\left(\mathbf{v}_k^0\right) = \mathbf{X}_{b_k}^i \left(\mathbf{X}_{b_k}^0\right)^{-1} \left(\mathbf{v}_k^0 - \mathbf{t}_{b_k}^0\right) + \mathbf{t}_{b_k}^i \tag{10}$$

is the function that rigidly transforms the rest pose vertex position by the single best bone that explains its deformation. The enveloping error measures only local errors as opposed to global errors. We present the average error and the enveloping error for each animation using each approximation method in Table 4.



(a) LBS $K = 1$ (b) LBS $K = 5$ (c) LBS $K = 10$ (d) LBS $K = 15$ (e) LBS $K = m$ (f) RR $K = 2$ (g) RR $K = 15$ (h) Our Method

Fig. 8. Our method compared with linear blend skinning using at most k bone weights per vertex and rotational regression using $K = 2$ and $K = 15$ input bones per deformation gradient. The deformation errors are denoted by the vertex color. Gray indicates no error while red indicates large error. The wire-frame of the ground truth mesh is rendered on top of each image to help visualize the errors.

For each animation, our method is more accurate than LBS with both $K = 4$ and $K = m$. Furthermore, our method is more accurate than rotational regression in each tested animation with the exception of the walk cycle for Tigress when $K = 15$. The mesh deformations on the characters we tested have regions where the vertex positions depend on more than two bones such as in the

Table 4. Mean approximation errors and enveloping errors (EE) using our method compared with linear blend skinning (LBS) with $K = 4$ and $K = m$ and rotational regression (RR) using the original choice of $K = 2$ as well as $K = 15$ input bones. The comparison is shown for all of the test animations with all of the rigs. EE is defined in Equation 9.

	Our Method		LBS ($K = 4$)		LBS ($K = m$)		RR ($K = 2$)		RR ($K = 15$)	
	Mean	EE	Mean	EE	Mean	EE	Mean	EE	Mean	EE
Tigress Walk	0.085 cm	18.47	0.365 cm	85.51	0.079 cm	20.29	0.188 cm	37.07	0.063 cm	11.78
Tigress Kungfu	0.207 cm	21.35	0.788 cm	65.12	0.640 cm	52.75	1.052 cm	86.95	1.033 cm	84.85
Shifu Walk	0.015 cm	6.88	0.171 cm	57.58	0.061 cm	19.83	0.145 cm	38.62	0.110 cm	26.85
Shifu Kungfu	0.043 cm	17.55	0.331 cm	66.71	0.283 cm	59.19	0.349 cm	56.93	0.336 cm	52.85
Po Walk	0.143 cm	19.03	0.307 cm	51.02	0.155 cm	23.01	0.321 cm	41.03	0.163 cm	23.46
Astrid Walk	0.104 cm	21.53	0.270 cm	62.35	0.116 cm	30.41	0.279 cm	66.74	0.257 cm	64.44

(a) LBS $K = 1$ (b) Ground Truth

Fig. 9. Side by side comparison of LSB with $K = 1$ (left) and the target deformation (right). Although the shape of the meshes appear similar, the vertices in the LBS deformation are moved tangentially along the surface, which can cause undesirable effects when applying textures to the mes.

hands and in the torso, and our results show that rotational regression performs better when more bones are provided as input. In Astrid's mesh, there are many small, unconnected meshes such as the spikes in her skirt. No vertex on these meshes can be accurately placed using rigid skinning with a single bone, which the rotational regression method relies on. Thus, the error from the rotational regression approximation is clearly visible as seen in Figure 10.

We did not compare our deformation approximation with skinning decomposition methods that add bones to a character rig. Skinning decomposition methods solve the problem of optimally fitting bones to a mesh when an existing animation is provided. The advantages of these types of methods are that they can have arbitrarily high accuracy when reproducing the example animations [Kavan et al. 2009] and that they can play back the example animations at a fast rate. Our method, in contrast, solves the problem of approximating mesh deformations given a character rig with an existing skeleton but without any example animations. Because of the difference in the type of problem that our method solves and the type that skinning decomposition methods solve, we do not compare our approach with these algorithms.

4.3 Model Speed

As seen in Table 5, the run-time of our approximation compared to the run-time of the original rig evaluation demonstrates the computational savings that can be achieved with our method. To train our models, we used Theano [Theano Development Team 2016] in a Python environment. Once the models were trained, we

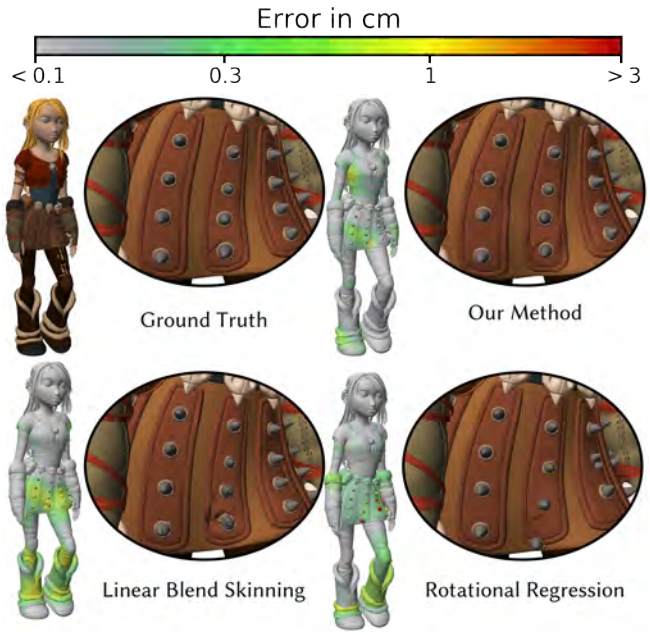


Fig. 10. Close-up of Astrid's skirt for the original mesh compared with our method, LBS, and RR. Our method more accurately approximated the vertices on the skirt. LBS produced visible errors in the middle of the skirt because those vertices cannot be placed accuratly as a linear combination of the bones. RR produced visible errors because the spikes on the skirt are separate meshes and cannot be placed accurately because RR relies on rigid skinning to fix the location of at least one vertex in each mesh.

evaluated them in our own multi-threaded C++ implementation. During training and testing, we use only the CPU to evaluate the networks. We found that running the models on the GPU was slower than the CPU. This slower performance is caused by the models having large inputs and outputs compared to the size of the hidden layers in the network. Thus, most of the time spent evaluating the network on the GPU was spent transferring data.

The training time for each approximation model took approximately 2-3 hours with the majority of the time spent evaluating the original character rig to generate the training data. Once trained, we evaluated the speed of the model by measuring the evaluation time through the model for a single input pose. Multiple input poses

can be passed into the model for a single evaluation, which would utilize matrix-matrix multiplications through the neural network as opposed to matrix-vector. Evaluating the model with multiple input poses would have a faster run-time per input compared with evaluation on poses one at a time. This speed increase comes from matrix-matrix multiplication running faster than separate matrix-vector multiplications for each input when using highly optimized linear algebra packages.

Despite the performance gains from evaluating multiple input poses simultaneously, we timed the approximation models evaluating inputs one at a time to demonstrate the applicability of our method for interactive applications. In Table 5, we compare the run-time of our method with Libee [Watt et al. 2012], a highly optimized, multi-threaded rig evaluation engine, on four different character rigs. All of the character rigs have been optimized to evaluate as fast as possible in Libee. Because our method approximates only the deformation system of the character, the times we report from Libee are measured by the difference in time between when the rig evaluation starts and finished all computations for the deformation system. We present times for running the rig evaluation in both parallel and single-threaded implementations. We ran our experiments on a machine with an Intel Xeon Processor E5-2690 v2 running at 3.0GHz with 10 cores and 92GB of RAM. In both cases, our approximation method outperforms Libee by up to a factor of 10. The largest performance gains are observed when comparing the parallel implementations.

In addition to Libee, we compare our method with the weighted variant [Kurihara and Miyata 2004] of pose space deformation [Lewis et al. 2000]. Like our method, PSD can be used to add a corrective offset to overcome the limitations of linear skinning. To compare the timing of our method with WPSD, we replace the neural networks in each nonlinear deformer and use WPSD to predict the same vertex offsets given the same input bones from the skeleton. We test WPSD using 10, 50, and 100 example poses from the test animations. The timing results using WPSD as well as the timing results evaluating the linear only skinning are shown in Table 5. Although the timing of our method is comparable to WPSD using 100 example poses, we would like to point out that the speed of WPSD depends on the number of example poses.

Table 5. Timing comparison in milliseconds for the deformation systems of the characters evaluated with Libee and our approximation using both a parallel implementation and a single-threaded implementation as well as the timing for the approximation run on a mobile device for the Astrid and Po character rigs. We also provide timing for WPSD using 100, 50, and 10 example poses and timing for the linear only skinning. The timings for the iPad, WPSD, and linear skinning are all evaluated on a parallel implementation on the CPU.

	Tigress	Shifu	Astrid	Po
Libee serial	65.2 ms	43.1 ms	142.5 ms	89.6 ms
Our approx. serial	10.6 ms	10.2 ms	62.0 ms	9.8 ms
Libee parallel	20.6 ms	8.7 ms	32.8 ms	28.2 ms
Our approx. parallel	2.7 ms	1.5 ms	7.7 ms	2.2 ms
iPad	N/A	N/A	28.6 ms	7.7 ms
WPSD 100	1.5 ms	1.5 ms	9.0 ms	1.4 ms
WPSD 50	1.0 ms	1.0 ms	7.6 ms	0.9 ms
WPSD 10	0.7 ms	0.6 ms	6.7 ms	0.6 ms
Linear only	0.5 ms	0.4 ms	3.2 ms	0.4 ms

Fig. 11. Posing example on iPad.

4.4 Applications

Our method provides a fast approximation to the deformation system, which allows a high-quality character rig to be evaluated in real-time on a low-end system or even a mobile device. To demonstrate our approach, we implemented the approximation on an iPad and evaluated both Astrid's and Po's character rigs on the device. Table 5 shows the timed results on the iPad. For both rigs, our approximation runs faster on the mobile device compared with the full evaluation of the deformation system running in parallel on a high-end machine using Libee.

We implemented a posing application for the iPad in which the user can pose the arms and legs of the character using IK controls. Figure 11 shows a screenshot of a user interacting with the application to pose Po. Because our method only approximates the deformation system, we still need to compute the input skeleton separately. The skeleton computation in Libee for both character rigs takes approximately the same amount time as the deformation. Thus, using Libee to compute the skeleton is impractical for our application. Instead, our iPad application uses a simplified skeleton system to compute only the joint angles from the IK controls, which allows our demo to run in real-time. See the supplemental video for a recording of a user interacting with our application.

5 DISCUSSION

We have presented a method that can accurately approximate mesh deformations for film-quality character rigs in real-time. Our method relies on defining the deformations in a local coordinate system to reduce the complexity of the nonlinear deformation function that we approximate. We use deep learning methods to learn these deformations and are able run the approximation in real-time.

5.1 Limitations

Our method assumes that mesh deformations are a function only of the skeleton. However, character rigs for feature films may have

additional deformations that rely on rig parameters that are not associated with bones. Currently, our approach is unable to learn these types of deformations, but the algorithm we have described can be modified if the additional rig parameters influencing these deformations are given as inputs to the approximation. Additionally, because our method only computes the approximation per pose, it cannot handle dynamics or non-deterministic behavior. Approximating these types of behaviors could make for an interesting extension to our method.

Deformations of a character's face is an example of deformations that rely both on bone transformations and additional rig parameters. In film-quality rigs, the face is animated with high-level artistic controls. Unlike the deformations on a character's body, the face deformations rely mostly on the high-level controls rather than the underlying skeleton. This difference would create a significant problem when directly applying our method to approximate the face. If our method were used to approximate the face deformation, the vertices on the face would be assigned to a small set of head bones that do not explain most of the deformation of the mesh. Furthermore each vertex might be affected by large number of rig parameters, which leads to a high-dimensional input for each vertex. Because the input dimension would be large and most of the facial deformation would need to be learned, training an approximator with our deep learning method would be challenging.

5.2 Linear Component Alternatives

From Figure 8, we can see that the approximation grows closer to the original mesh as K increases. Because LBS can be computed quickly, our method could compute the linear deformation component from Equation 1 using some $K > 1$, and this would reduce the residual error that the nonlinear function approximators need to learn. However, we found that in practice using a larger K does not have a significant visual impact on the results.

Delta Mush [Mancewicz et al. 2014] is a type of deformation that aims to preserve the volume of a deformed mesh. Additionally, Delta Mush can easily be applied to a character rig without requiring any fine-tuning. Although this method is not a linear deformation, by preserving volume, Delta Mush could bring the deformed mesh closer to the target deformation. As a result, the remaining nonlinear deformation could be easier to learn and could be approximated with smaller and faster neural networks. Using Delta Mush as an alternative for the linear component of our method could be an interesting area of exploration.

5.3 Nonlinear Component Alternatives

Although pose space deformation can approximate mesh deformations faster than our method if sufficiently few example poses are provided, the quality of the approximation depends heavily on the selected poses. We found that using poses generated from our training set described in Section 3.4 as example poses for PSD does not result in an accurate deformation approximator. Better results could be achieved by manually selecting example poses to ensure a more accurate approximation. Our method, in contrast, is able to learn an accurate approximation from this randomly generated dataset.

5.4 Potential Applications

Our method can be combined with other approaches that can provide a character skeleton in real-time to create interesting real-time experiences. For example, motion capture recordings can be used to drive a character's skeleton, and our method can use the skeleton to compute the final deformed mesh of a character. Furthermore, prior research has explored animation synthesis techniques. Motion graphs [Kovar et al. 2002] can be used to synthesize controllable animation at interactive rates. The animation generated by this approach is a sequence of skeletons, which our method can use to compute a character's mesh deformation. Other synthesis methods use generative models such as Gaussian processes [Grochow et al. 2004; Levine et al. 2012; Wang et al. 2008] or deep learning models [Holden et al. 2017, 2016]. All of these motion synthesis methods output bone positions and rotations for a character, which form the inputs to our method. Because character skeletons can be generated using many different techniques, our method can readily be applied to the outputs of these synthesis algorithms to animate a film-quality rig in real-time.

ACKNOWLEDGMENTS

We would like to thank Bret Stastny for his help with optimizing the parallel implementation of our method as well as our implementation on the iPad. We would also like to thank all of the DreamWorks character artists and animators who helped create the character rigs and animations that were used for this paper. Finally, we would like to thank the anonymous reviewers for their helpful feedback.

REFERENCES

Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. 2002. Interactive Skeleton-driven Dynamic Deformations. *ACM Trans. Graph.* 21, 3 (July 2002), 586–593. https://doi.org/10.1145/566654.566622

Edilson De Aguiar, Christian Theobalt, Sebastian Thrun, and Hans-Peter Seidel. 2008. Automatic Conversion of Mesh Animations into Skeleton-based Animations. *Computer Graphics Forum* 27, 2 (2008), 389–397. https://doi.org/10.1111/j.1467-8659.2008.01136.x

Wei-Wen Feng, Byung-Uck Kim, and Yizhou Yu. 2008. Real-time Data Driven Deformation Using Kernel Canonical Correlation Analysis. *ACM Trans. Graph.* 27, 3, Article 91 (Aug. 2008), 9 pages. https://doi.org/10.1145/1360612.1360690

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep Sparse Rectifier Neural Networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research)*, Geoffrey Gordon, David Dunson, and Miroslav Dudík (Eds.), Vol. 15. PMLR, Fort Lauderdale, FL, USA, 315–323. http://proceedings.mlr.press/v15/glorot11a.html

Keith Grochow, Steven L. Martin, Aaron Hertzmann, and Zoran Popović. 2004. Style-based Inverse Kinematics. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 522–531. https://doi.org/10.1145/1015706.1015755

Fabian Hahn, Sebastian Martin, Bernhard Thomaszewski, Robert Sumner, Stelian Coros, and Markus Gross. 2012. Rig-space Physics. *ACM Trans. Graph.* 31, 4, Article 72 (July 2012), 8 pages. https://doi.org/10.1145/2185520.2185568

Nils Hasler, Thorsten Thormählen, Bodo Rosenhahn, and Hans-Peter Seidel. 2010. Learning Skeletons for Shape and Pose. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '10)*. ACM, New York, NY, USA, 23–30. https://doi.org/10.1145/1730804.1730809

Jim Hejl. 2004. Hardware Skinning with Quaternions. In *Game Programming Gems 4*, Andrew Kirmse (Ed.). Charles River Media, 487–495.

Daniel Holden, Taku Komura, and Jun Saito. 2017. Phase-functioned Neural Networks for Character Control. *ACM Trans. Graph.* 36, 4, Article 42 (July 2017), 13 pages. https://doi.org/10.1145/3072959.3073663

Daniel Holden, Jun Saito, and Taku Komura. 2016. A Deep Learning Framework for Character Motion Synthesis and Editing. *ACM Trans. Graph.* 35, 4, Article 138 (July 2016), 11 pages. https://doi.org/10.1145/2897824.2925975

Daniel Holden, Jun Saito, and Taku Komura. 2017. Learning Inverse Rig Mappings by Nonlinear Regression. *IEEE Transactions on Visualization and Computer Graphics* 23, 3 (March 2017), 1167–1178. https://doi.org/10.1109/TVCG.2016.2628036

Kurt Hornik. 1991. Approximation Capabilities of Multilayer Feedforward Networks. *Neural Netw.* 4, 2 (March 1991), 251–257. https://doi.org/10.1016/0893-6080(91)90009-T

Doug L. James and Christopher D. Twigg. 2005. Skinning Mesh Animations. *ACM Trans. Graph.* 24, 3 (July 2005), 399–407. https://doi.org/10.1145/1073204.1073206

Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. 2007. Harmonic Coordinates for Character Articulation. *ACM Trans. Graph.* 26, 3, Article 71 (July 2007). https://doi.org/10.1145/1276377.1276466

Tao Ju, Qian-Yi Zhou, Michiel van de Panne, Daniel Cohen-Or, and Ulrich Neumann. 2008. Reusable Skinning Templates Using Cage-based Deformations. *ACM Trans. Graph.* 27, 5, Article 122 (Dec. 2008), 10 pages. https://doi.org/10.1145/1409060.1409075

Ladislav Kavan, Steven Collins, and Carol O'Sullivan. 2009. Automatic Linearization of Nonlinear Skinning. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games (I3D '09)*. ACM, New York, NY, USA, 49–56. https://doi.org/10.1145/1507149.1507157

Ladislav Kavan, Rachel McDonnell, Simon Dobbyn, Jiří Žára, and Carol O'Sullivan. 2007. Skinning Arbitrary Deformations. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games (I3D '07)*. ACM, New York, NY, USA, 53–60. https://doi.org/10.1145/1230100.1230109

L. Kavan, P.-P. Sloan, and C. O Sullivan. 2010. Fast and Efficient Skinning of Animated Meshes. *Computer Graphics Forum* (2010). https://doi.org/10.1111/j.1467-8659.2009.01602.x

Ladislav Kavan and Jiří Žára. 2005. Spherical Blend Skinning: A Real-time Deformation of Articulated Models. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games (I3D '05)*. ACM, New York, NY, USA, 9–16. https://doi.org/10.1145/1053427.1053429

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). http://arxiv.org/abs/1412.6980

Lucas Kovar, Michael Gleicher, and Frédéric Pighin. 2002. Motion Graphs. *ACM Trans. Graph.* 21, 3 (July 2002), 473–482. https://doi.org/10.1145/566654.566605

Tsuneya Kurihara and Natsuki Miyata. 2004. Modeling Deformable Human Hands from Medical Images. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '04)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 355–363. https://doi.org/10.1145/1028523.1028571

Samuli Laine, Tero Karras, Timo Aila, Antti Herva, Shunsuke Saito, Ronald Yu, Hao Li, and Jaakko Lehtinen. 2017. Production-level Facial Performance Capture Using Deep Convolutional Neural Networks. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA '17)*. ACM, New York, NY, USA, Article 10, 10 pages. https://doi.org/10.1145/3099564.3099581

Binh Huy Le and Zhigang Deng. 2012. Smooth Skinning Decomposition with Rigid Bones. *ACM Trans. Graph.* 31, 6, Article 199 (Nov. 2012), 10 pages. https://doi.org/10.1145/2366145.2366218

Binh Huy Le and Zhigang Deng. 2013. Two-layer Sparse Compression of Dense-weight Blend Skinning. *ACM Trans. Graph.* 32, 4, Article 124 (July 2013), 10 pages. https://doi.org/10.1145/2461912.2461949

Binh Huy Le and Zhigang Deng. 2014. Robust and Accurate Skeletal Rigging from Mesh Sequences. *ACM Trans. Graph.* 33, 4, Article 84 (July 2014), 10 pages. https://doi.org/10.1145/2601097.2601161

Sung-Hee Lee, Eftychios Sifakis, and Demetri Terzopoulos. 2009. Comprehensive Biomechanical Modeling and Simulation of the Upper Body. *ACM Trans. Graph.* 28, 4, Article 99 (Sept. 2009), 17 pages. https://doi.org/10.1145/1559755.1559756

Sergey Levine, Jack M. Wang, Alexis Haraux, Zoran Popović, and Vladlen Koltun. 2012. Continuous Character Control with Low-dimensional Embeddings. *ACM Trans. Graph.* 31, 4, Article 28 (July 2012), 10 pages. https://doi.org/10.1145/2185520.2185524

J. P. Lewis, Matt Cordner, and Nickson Fong. 2000. Pose Space Deformation: A Unified Approach to Shape Interpolation and Skeleton-driven Deformation. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 165–172. https://doi.org/10.1145/344779.344862

N. Magnenat-Thalmann, R. Laperrière, and D. Thalmann. 1988. Joint-dependent Local Deformations for Hand Animation and Object Grasping. In *Proceedings on Graphics Interface '88*. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 26–33. http://dl.acm.org/citation.cfm?id=102313.102317

Joe Mancewicz, Matt L. Derksen, Hans Rijpkema, and Cyrus A. Wilson. 2014. Delta Mush: Smoothing Deformations While Preserving Detail. In *Proceedings of the Fourth Symposium on Digital Production (DigiPro '14)*. ACM, New York, NY, USA, 7–11. https://doi.org/10.1145/2633374.2633376

Tim McLaughlin, Larry Cutler, and David Coleman. 2011. Character Rigging, Deformations, and Simulations in Film and Game Production. In *ACM SIGGRAPH 2011 Courses (SIGGRAPH '11)*. ACM, New York, NY, USA, Article 5, 18 pages. https://doi.org/10.1145/2037636.2037641

Alex Mohr and Michael Gleicher. 2003. Building Efficient, Accurate Character Skins from Examples. *ACM Trans. Graph.* 22, 3 (July 2003), 562–568. https://doi.org/10.1145/882262.882308

Tomohiko Mukai and Shigeru Kuriyama. 2016. Efficient Dynamic Skinning with Low-rank Helper Bone Controllers. *ACM Trans. Graph.* 35, 4, Article 36 (July 2016), 11 pages. https://doi.org/10.1145/2897824.2925905

S. Schaefer and C. Yuksel. 2007. Example-based Skeleton Extraction. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing (SGP '07)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 153–162. http://dl.acm.org/citation.cfm?id=1281991.1282013

Peter-Pike J. Sloan, Charles F. Rose, III, and Michael F. Cohen. 2001. Shape by Example. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics (I3D '01)*. ACM, New York, NY, USA, 135–143. https://doi.org/10.1145/364338.364382

Theano Development Team. 2016. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints* abs/1605.02688 (May 2016). http://arxiv.org/abs/1605.02688

Jean-Marc Thiery, Émilie Guy, Tamy Boubekeur, and Elmar Eisemann. 2016. Animated Mesh Approximation With Sphere-Meshes. *ACM Trans. Graph.* 35, 3, Article 30 (May 2016), 13 pages. https://doi.org/10.1145/2898350

Jack M. Wang, David J. Fleet, and Aaron Hertzmann. 2008. Gaussian Process Dynamical Models for Human Motion. *IEEE Trans. Pattern Anal. Mach. Intell.* 30, 2 (Feb. 2008), 283–298. https://doi.org/10.1109/TPAMI.2007.1167

Robert Y. Wang, Kari Pulli, and Jovan Popović. 2007. Real-time Enveloping with Rotational Regression. *ACM Trans. Graph.* 26, 3, Article 73 (July 2007). https://doi.org/10.1145/1276377.1276468

Xiaohuan Corina Wang and Cary Phillips. 2002. Multi-weight Enveloping: Least-squares Approximation Techniques for Skin Animation. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '02)*. ACM, New York, NY, USA, 129–138. https://doi.org/10.1145/545261.545283

Martin Watt, Lawrence D. Cutler, Alex Powell, Brendan Duncan, Michael Hutchinson, and Kevin Ochs. 2012. LibEE: A Multithreaded Dependency Graph for Character Animation. In *Proceedings of the Digital Production Symposium (DigiPro '12)*. ACM, New York, NY, USA, 59–66. https://doi.org/10.1145/2370919.2370930