Multi-Resolution Real-Time Deep Pose-Space Deformation

MIANLUN ZHENG, University of Southern California, USA
JERNEJ BARBIČ, University of Southern California, USA

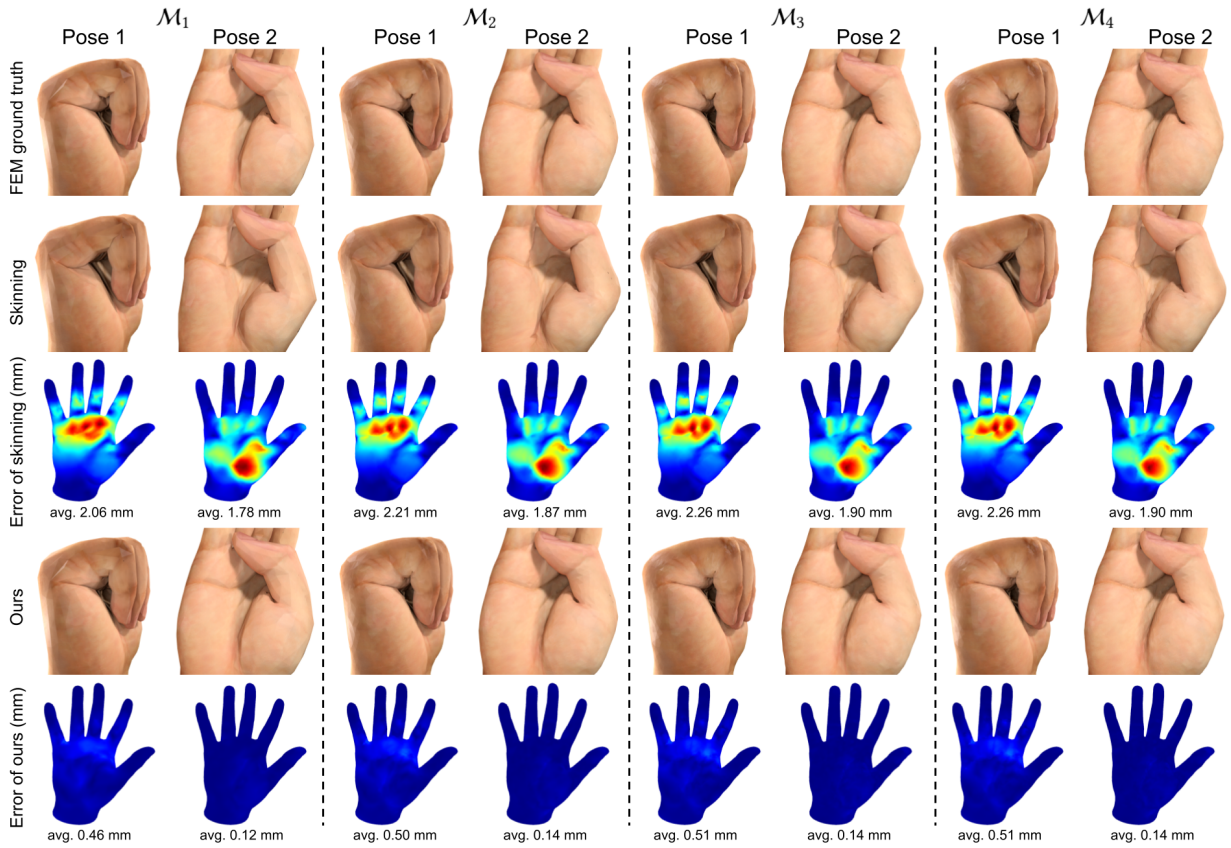

Fig. 1. Our method permits one to place the hand into any pose in the hand's ROM and rapidly compute its quality shape, and do so at any (or several) desirable discrete mesh resolution levels. The training set consists of 3,607 frames of FEM musculoskeletal simulation exercising the ROM of the hand. We show the ground truth (computed using FEM musculoskeletal simulation; 48 seconds per frame), the output of linear blend skinning (visible artefacts), and our result. We show two representative hand poses that are **not** a part of the training set. The four mesh Levels of Detail $\mathcal{M}_0$, $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$ have 1,133, 4,528, 18,105, 72,414 vertices, respectively. Skinning running times for the four levels are 162, 210, 446, 1,156 microseconds per frame, respectively. Our method computes the skinning correctives at the four levels (61, 144, 299, 548 microseconds per frame, respectively), plus performs the same skinning. For a small additional computational cost on top of skinning, we greatly improve the output quality compared to skinning (see images and the error plots), and speedup the computation by 100,000× compared to FEM simulation. Note that the error does not decrease down to zero on progressive LODs because at each LOD, the FEM training dataset contains new deformation detail only introduced and resolved by the mesh at this LOD.

Authors' addresses: Mianlun Zheng, University of Southern California, Los Angeles, USA, mianlunz@usc.edu; Jernej Barbič, University of Southern California, Los Angeles, USA, jnb@usc.edu.

0730-0301/2024/12-ART216
https://doi.org/10.1145/3687985

We present a hard-real-time multi-resolution mesh shape deformation technique for skeleton-driven soft-body characters. Producing mesh deformations at multiple levels of detail is very important in many applications in computer graphics. Our work targets applications where the multi-resolution shapes must be generated at fast speeds ("hard-real-time", e.g., a few milliseconds at most and preferably under 1 millisecond), as commonly needed in computer games, virtual reality and Metaverse applications. We assume that the character mesh is driven by a skeleton, and that high-quality character shapes are available in a set of training poses originating from a high-quality (slow) rig such as volumetric FEM simulation. Our method combines multi-resolution analysis, mesh partition of unity, and neural networks, to learn

the pre-skinning shape deformations in an arbitrary character pose. Combined with linear blend skinning, this makes it possible to reconstruct the training shapes, as well as interpolate and extrapolate them. Crucially, we simultaneously achieve this at hard real-time rates and at multiple mesh resolution levels. Our technique makes it possible to trade deformation quality for memory and computation speed, to accommodate the strict requirements of modern real-time systems. Furthermore, we propose memory layout and code improvements to boost computation speeds. Previous methods for real-time approximations of quality shape deformations did not focus on hard real-time, or did not investigate the multi-resolution aspect of the problem. Compared to a "naive" approach of separately processing each hierarchical level of detail, our method offers a substantial memory reduction as well as computational speedups. It also makes it possible to construct the shape progressively level by level and interrupt the computation at any time, enabling graceful degradation of the deformation detail. We demonstrate our technique on several examples, including a stylized human character, human hands, and an inverse-kinematics-driven quadruped animal.

CCS Concepts: • **Computing methodologies** → **Physical simulation**.

Additional Key Words and Phrases: real-time, multi-resolution, shape approximation, FEM, neural networks, skinning

ACM Reference Format:
Mianlun Zheng and Jernej Barbič. 2024. Multi-Resolution Real-Time Deep Pose-Space Deformation. *ACM Trans. Graph.* 43, 6, Article 216 (December 2024), 11 pages. https://doi.org/10.1145/3687985

1 INTRODUCTION

There are many techniques to produce high-quality animated shapes of digital characters, such as production character rigs, FEM soft-tissue simulation, 4D scanning and manual shot-sculpting. However, these techniques are time-consuming and often cannot be performed in real time. In real-time systems, one often animates a 3D mesh based on the motion of the underlying joint hierarchy. This can be done to animate humans, animals, plants and even inanimate objects, and is pervasive in real-time systems such as computer games, virtual reality, virtual production and the Metaverse. The standard algorithm for this task (linear blend skinning (LBS) [Jacobson et al. 2014]) is fast and easy to use, but it also produces well-known artifacts. Those can be corrected with several real-time techniques such as dual quaternions [Kavan et al. 2008], "delta mush" [Le and Lewis 2019], helper joints [Kavan et al. 2009] or pose-space deformation [Lewis et al. 2000].

These techniques, however, do not address a critically important component of real-time systems, namely the need for multiple levels of detail. In a typical interactive application, important assets are represented with multiple meshes, with a progressive number of vertices and triangles ("Levels of Detail", LOD). LODs are very common, for example, in popular game engines such as Unreal and Unity. They are necessary to keep the real-time performance sufficiently high, so that a character that is far from the camera can be displayed at a lower resolution. Similarly, if multiple characters are in the foreground, it is useful to display them at a decreased resolution to maintain the required update frame rates. In our work, we give a multi-resolution shape deformation technique that, given any character pose, produces the output shape at any (or several) desired level(s) of detail (Figure 1), based on any suitable shape deformation training data; and does so at hard real-time rates.

Our technique is grounded in multi-resolution methods extensively investigated in computer graphics [Boier-Martin et al. 2005]. It is similar in spirit to Pose-Space Deformation [Lewis et al. 2000] and Fast Deep Deformations [Bailey et al. 2018] in that we also generate pose-dependent pre-skinning correctives to the neutral shape using neural networks, to which linear blend skinning is applied to produce the output shape. However, our key distinction is that we give an algorithm to do so at multiple LODs, in an interruptible manner, and in a manner whereby output accuracy can be controled. Our first contribution is to define neural networks so that they predict the shape deformation at some LOD and in some spatially localized region, *relative to the shape deformation already determined on coarser levels*. We observe that if this is combined with the mesh prolongation operator, the neural networks at some LOD then only need to resolve the shape deformation detail arising at this resolution level, as opposed to re-creating the entire shape deformation from scratch; this leads to large memory savings. Next, we contribute the observation that the spatially localized regions can be defined automatically using the prolongation operator [Liu et al. 2021], in a manner intertwined with the mesh simplification algorithm, avoiding tedious manual selection of regions. The different mesh LODs do not need to be subdivisions of each other. We use quadric error mesh simplification [Garland and Heckbert 1997] to create our mesh LODs, but our technique is suitable for other mesh hierarchies, as long as they permit a definition of prolongation and restriction operators [Liu et al. 2021], e.g., progressive meshes [Hoppe 1996], or mesh subdivision [James and Pai 2003].

The LOD hierarchy and all other necessary datastructures are created during pre-processing. Analogous to how the prolongation operator defines the influence of a coarse vertex to the finer level, we use the prolongation operator to define the spatial influence weights for each neural network. Then at runtime, one can select any particular LOD (or several, or even all), and our technique produces quality pre-skinning deformations at those LODs, at hard real-time rates. An example of such an application are computer games, where different characters may be rendered at different LODs only known at runtime, based on character importance, distance to camera and other metrics [Funkhouser and Séquin 1993]. They can also be rendered at several LODs at the same frame in case of a multi-player game where multiple cameras observe the same character from different distances; or at two LODs when blending two LODs to avoid popping. Other potential applications include progressive transmission of character deformation over a network, or multi-resolution contact handling with graceful degradation [Otaduy 2004].

Our technique is fast; producing the complete LOD hierarchy of correctives for meshes with over 70,000 vertices (Figures 1, 2) in approximately 1 millisecond when using multiple cores (with cold caches), and in a few milliseconds when using a single core. We are not aware of any other work that has investigated how to approximate given character shape training data in hard real time at multiple levels of detail. A naive solution would be to simply treat each LOD as a separate problem and train pre-skinning correctives separately for each mesh LOD, e.g., using Bailey's method [Bailey et al. 2018]. We compare to such a method and demonstrate that our method greatly reduces the memory requirement to store the

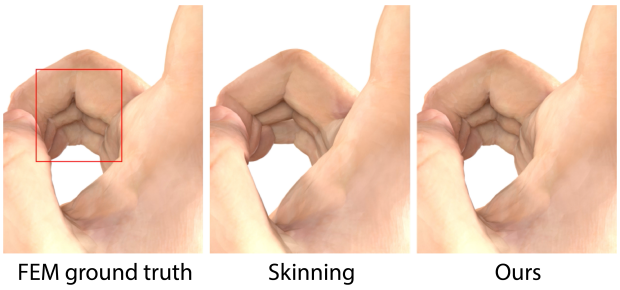

Fig. 2. **Mesh detail on the hand.** Our method better captures the ground truth than skinning.

neural networks and other datastructures needed for runtime evaluation. Note that a small memory footprint is critically important for interactive applications such as games, whereby memory needs to be shared with the other game assets. Our training times are also substantially reduced, thanks to a lesser number of required neural networks. Our method maintains comparable or faster runtime speeds to non-hierarchical methods, depending on how many LODs are required at a particular frame at runtime. These results are possible thanks to the incremental nature of our multi-resolution neural networks that only need to add incremental deformation detail at each LOD. The "world" of real-time computing in the microsecond regime is very different to the more typical real-time computer animation applications with running times in tens of milliseconds or more. For example, for a mesh with 70,000 vertices, it takes 50 microseconds just to write a single already computed mesh shape to memory, let alone do any deformation computation. Therefore, although our core contribution is algorithmic, we also discuss various code and memory optimizations suitable for such a fast computational regime.

2 RELATED WORK

Real-time shape approximation. Starting from a set of rig poses and matching shapes as input, there are several learning-based methods to reproduce and interpolate high-quality static shape deformations in *real time*. One can learn pre-skinning residuals using Radial Basis Functions (RBFs) [Lewis et al. 2000], or further process the residuals using PCA [Kry et al. 2002]. The latter was applied to hand deformation with training obtained from FEM, similar to our hand example but using a soft-tissue FEM pipeline [McAdams et al. 2011] as opposed to musculoskeletal anatomy-based simulation. Bailey et al. [Bailey 2020; Bailey et al. 2018] used both PCA and deep neural networks to learn real-time film-quality character mesh deformations originating from production rigs. Similarly for facial animation, Song et al. [Song et al. 2020] used joint transforms as input and learned localized character shapes in differential coordinates. These methods did not pursue multi-resolution shape deformation. In follow-up work, Bailey et al. [Bailey et al. 2020] presented a method that uses convolutional neural networks for approximating facial mesh deformations. While this method employed a 2-level coarse/fine deformation structure, their construction uses texture mapping, and is specific to facial deformation (e.g., to accommodate wrinkles). Our method is designed for general meshes and generic multi-level mesh simplifications schemes; we use 4 LODs in our examples. Multi-resolution hierarchies are commonly used in computer graphics, see, e.g., the survey of Boier-Martin et al. [Boier-Martin et al. 2005]. They can be combined with FEM simulation [Capell et al. 2002; Debunne et al. 2001; Grinspun et al. 2002; Zhang et al. 2022], and form the core of the multigrid method [Otaduy et al. 2007; Zhu et al. 2010]. These methods are typically not designed for hard real-time systems, however. For fast evaluation in games, progressive upscaling has been previously explored for cloth simulation [Kavan et al. 2011]; in our work, we drive character skins using a skeleton hierarchy.

Learning physics. Numerous studies have demonstrated that learning-based approaches are effective in addressing the computational challenges of physics-based simulation methods, particularly in subspace learning and modeling. Fulton et al. [Fulton et al. 2019] utilized an autoencoder neural network to generate nonlinear reduced spaces for deformation dynamics, and Holder et al. [Holden et al. 2019] used a neural network to learn the motion and interaction of deformable objects in a subspace. Badias et al. [Badías et al. 2020] applied model reduction to contact solutions in a reduced space, enabling real-time interaction between virtual and physical objects. To achieve fast and detailed contact-driven deformation, Casas et al. [Casas et al. 2021] integrated nonlinear learning-based corrections with existing linear handle-based subspace formulations, whereas Romero et al. [Romero et al. 2022] machine-learned contact deformations in a contact-centric manner and integrated them with subspace dynamics. Unlike the above methods that benefit from the subspace, the light-weight NNWarp [Luo et al. 2018] corrects a linear displacement by predicting a per-vertex nonlinear incremental displacement, allowing real-time simulation of large models. Statistical and learning approaches have been used for a long time to build approximations of real-world scanned deformations, for example [Loper et al. 2015] and its many follow-up works. These methods have however generally not been designed with real-time or multi-resolution performance in mind. There are also researchers working on producing character soft-tissue dynamics using neural networks [Habermann et al. 2021; Romero et al. 2020; Santesteban et al. 2020; Zheng et al. 2021]. Simulating clothing using deep learning is another well-studied field [Bertiche et al. 2021; Chentanez et al. 2020; Santesteban et al. 2019]. As discussed above, the application of machine learning to predict shape deformation has been extensively researched. Few methods have discussed real-time performance, however. Moreover, previous methods did not address the need to simultaneously support multiple LODs and hard real-time deformation. In our work, we give a method that computes the shape at any level(s) of detail at hard real-time rates, and does so intertwined with standard widely used mesh level-of-detail algorithms, e.g., quadric error surfaces [Garland and Heckbert 1997].

Learning the rig. Researchers have also explored learning-based methods to produce a LBS rig from mesh animations [Le and Deng 2014], or avoid the standard LBS shortcomings. Liu et al. [Liu et al. 2019] presented an end-to-end deep graph convolution network to automatically compute skin weights for skeleton-based deformation

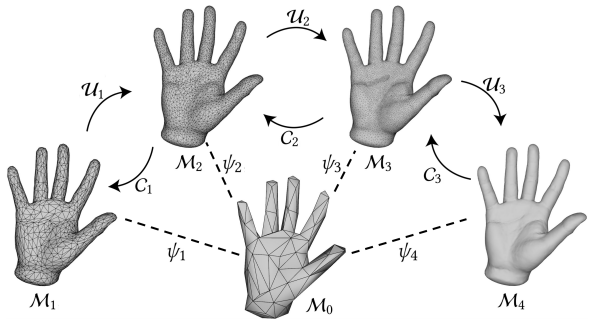

Fig. 3. **Multi-resolution mesh hierarchy and the upsampling and coarsening operators.** Level 0 is only used for defining our "basis functions" for shape deformation (Section 5).

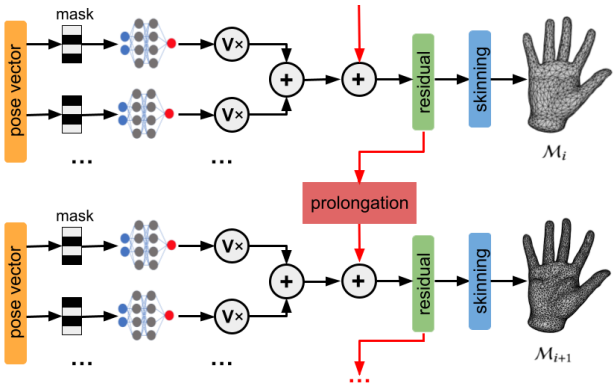

Fig. 4. **Multi-resolution pose-space deformation.** Pre-skinning displacements are constructed progressively, level by level, using a set of spatially localized neural networks at each LOD. The mask exists because only a part of the pose enters the neural network, determined using perturbation analysis on each joint as explained in [Bailey et al. 2018].

of production characters. Deng et al. [2020] computed neural indicator functions to efficiently represent articulated deformable objects, and RigNet [Xu et al. 2020] utilized a deep architecture to predict skeletons and surface skin weights from input 3D character meshes. Similarly, the work [Li et al. 2021] developed a neural network capable of rigging an input character mesh, along with neural blend shapes to improve the deformation quality. The above methods specifically focused on rigging and skinning weights, whereas we investigate LOD mesh deformation at hard real-time rates. Our training data comes from physically based simulation and incorporates volume preservation and contact resolution.

3 OVERVIEW

We assume that we are given a neutral-pose mesh at multiple LODs $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_r$, where each $\mathcal{M}_i$ is a triangle mesh, and level 1 is the coarsest and r is the finest. Let n_i be the number of vertices in $\mathcal{M}_i$. We assume that the LODs are generated using a mesh simplification algorithm that can define the prolongation and restriction operators between adjacent levels, for example, as described in [Liu et al. 2021]. Thus, we first create our restriction (coarsening) $C_i \in \mathbb{R}^{3n_{i-1} \times 3n_i}$ and prolongation (upsampling) $\mathcal{U}_i \in \mathbb{R}^{3n_{i+1} \times 3n_i}$ operators (Figure 3), following [Liu et al. 2021]. Here, C_i coarsens any scalar field from $\mathcal{M}_i$ to $\mathcal{M}_{i-1}$, and $\mathcal{U}_i$ upsamples any scalar field from $\mathcal{M}_i$ to $\mathcal{M}_{i+1}$; we have $C_{i+1}\mathcal{U}_i = 1_{\mathcal{M}_i}$. We also assume that we are provided with a joint hierarchy and skinning weights on the finest level $\mathcal{M}_r$. We use the coarsening operators to coarsen the skinning weights from $\mathcal{M}_r$ to all coarser LODs.

We assume that we are given training data (p^j, u_r^j) consisting of N frames, $j = 1, \ldots, N$, whereby $p^j \in \mathbb{R}^P$ is a pose, and $u_r^j \in \mathbb{R}^{3n_r}$ gives the displacements of all vertices away from the neutral-pose mesh $\mathcal{M}_r$. In our examples, the training shapes come from two types of volumetric FEM rigs, namely soft-tissues surrounding volumetric bones [McAdams et al. 2011] and musculoskeletal FEM rigs [Zheng et al. 2022]. However, our method makes no explicit assumption on the source of quality mesh deformations, and could in principle be applied, e.g., even to 4D optical scans followed by mesh registration. The pose vector p^j contains the joint angles of the joint hierarchy, represented using Euler angles.

We transform the training shapes into the pre-skinning space using inverse skinning, and then use these shapes to train level-specific and spatially localized neural networks (Section 4). This is done incrementally so that the neural networks at each level only add the deformation detail introduced at that level (Figure 4). At runtime, given a pose p, the networks then produce progressive pre-skinning displacements on meshes $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_r$ until either the computation is terminated, or the last level has been computed. Finally, by applying skinning, we compute the output shape for pose p. Our output shapes reasonably re-create the training data in the training poses, and interpolate and extrapolate it to unseen poses. Of course, outside of the training ROM, our method under-performs (Figure 12).

4 MULTI-RESOLUTION POSE-SPACE DEFORMATION

As is commonly done [Bailey et al. 2018; Kry et al. 2002; Lewis et al. 2000], our method operates completely on pre-skinning displacements. At runtime, it constructs pre-skinning displacements in a given pose; and skinning then only transforms those displacements into the world-space. Given training poses p^j and vertex deformations u_r^j at the finest mesh level of detail $\mathcal{M}_r$, for $j = 1, \ldots, N$, we first invert skinning to convert them to the pre-skinning space, producing pre-skinning displacements X_r^j. This is done by computing the skinning transformation (consisting of a 3x3 linear matrix and a translation) at each vertex, and X_r^j is then obtained by applying its inverse to the world-coordinate training vertex position, followed by subtraction of the undeformed vertex position. We then coarsen each X_r^j to each level of our hierarchy, using the coarsening operators C_i, producing pre-skinning displacements X_i^j, for levels $i = r, \ldots, 1$. At each level i, our goal is to generate a deformation function F_i that maps pose p to a pre-skinning displacement X on $\mathcal{M}_i$. This is *not* done directly by using X_i^j as the training data, because this would not leverage the work already done at coarser LODs. Instead, we train our F_i by leveraging the already trained F_{i-1}, i.e., we train deformation functions in the order $F_1, F_2, \ldots, F_r$, using "induction".

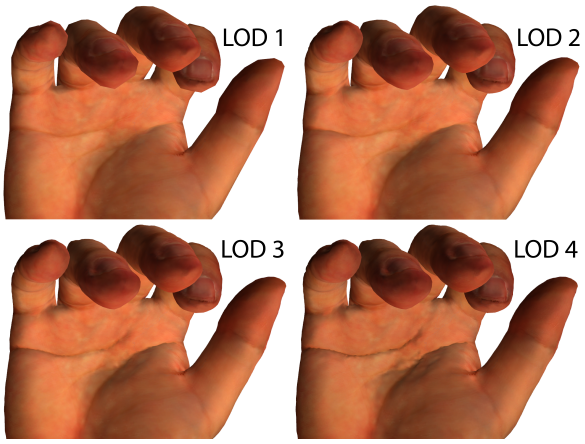

Fig. 5. **Hand shape computation and real-time rendering at multiple LODs.** Phong shading + texture mapping, dynamic normals. Observe the increasing geometric detail seen in shading as LOD is increased. At lower LODs, geometric detail is lost and only texture mapping remains.

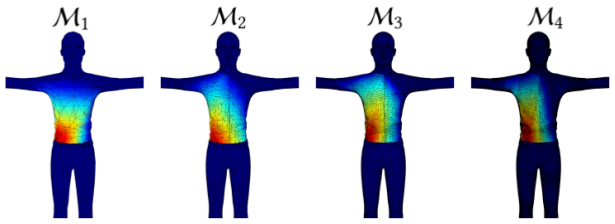

Fig. 6. **Multi-resolution shape function.** We show a representative shape function at several LOD levels.

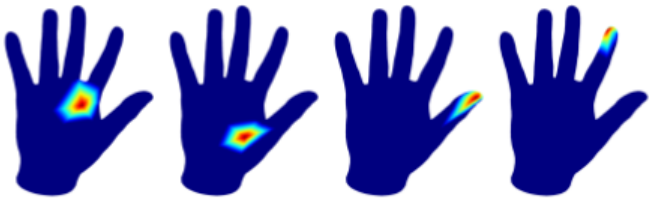

Fig. 7. **A few basis functions** on $\mathcal{M}_4$ (not the complete set).

Suppose F_{i-1} is already trained and we now need to train F_i. Given a training pose p^j, we first evaluate $F_{i-1}(p^j)$, upsample it to level i using the upsampling operator, and subtract it from X_i^j,

$$Y_i^j = X_i^j - \mathcal{U}_i F_{i-1}(p^j). \tag{1}$$

Note that $\mathcal{U}_i F_{i-1}(p^j)$ is the predicted pre-skinning displacement, on the mesh of level i, as predicted by the coarser levels $1, \ldots, i-1$, and Y_i^j is the "residual" deformation detail added on level i. It is this residual deformation detail that is approximated by the neural networks at level i. Specifically, we use neural networks to construct a deformation function N_i that maps an arbitrary pose p to a residual Y_i. We then compute our pre-skinning displacements at level i as

$$F_i(p) = \mathcal{U}_i F_{i-1}(p) + N_i(p). \tag{2}$$

The "induction" starts ($i = 1$) without any previous level; i.e., $N_1(p)$ is trained directly using the training shapes X_1^j.

At runtime, given a new pose p, we then first evaluate $F_1(p) = N_1(p)$, and then use Equation 2 to compute $F_2(p), F_3(p), \ldots$. This progressively reconstructs the pre-skinning shape at finer and finer LODs. This process is interruptible, permitting us to trade the output mesh quality for speed, and continues until the desired LOD is reached. Finally, we apply skinning (at any required level) to compute the output character shape. Figure 5 demonstrates this runtime process, combined with real-time rendering.

5 NEURAL DEFORMATION FUNCTIONS

We now describe how we create our deformation functions N_i at each LOD level i. We do this by training a set of spatially-localized neural networks, using the concept of a mesh partition of unity. A partition of unity on $\mathcal{M}_i$ is a set of smooth non-negative scalar functions $\psi_i^k : \mathcal{M}_i \to \mathbb{R}$ that add to 1 at each vertex, $\sum_k \psi_i^k = 1_{\mathcal{M}_i} \equiv [1, \ldots, 1]^T$ (1 repeated n_i times). Partitions of unity are of course not new, see, e.g. [Babuška et al. 1994; Ohtake et al. 2003]; our contribution is to combine them with neural networks and a multi-resolution scheme. We first create a partition of unity on the coarsest resolution level, as follows. First, observe that column k of the upsampling operator $\mathcal{U}_i$ gives the "influence" of vertex k of $\mathcal{M}_i$ to vertices on $\mathcal{M}_{i+1}$. This column can be seen as a non-negative scalar field defined on vertices of $\mathcal{M}_{i+1}$; we will denote it by $\mathcal{U}_i^{(k)}$. Observe that the field is localized because the influence of each vertex of $\mathcal{M}_i$ onto mesh $\mathcal{M}_{i+1}$ is spatially limited. To partition the unity on $\mathcal{M}_1$, we execute the mesh simplification algorithm on $\mathcal{M}_1$, producing an even coarser LOD $\mathcal{M}_0$. This LOD will not actually be constructed at runtime, and will not have any neural networks; we will only use it to define our partition of unity. Namely, our partition of unity on $\mathcal{M}_1$ consists of n_0 scalar fields $\psi_1^k = \mathcal{U}_0^{(k)} \in \mathbb{R}^{3n_1}$, where $k = 1, \ldots n_0$. We then prolong each scalar function ψ_1^k to all the resolution levels $\mathcal{M}_2, \ldots, \mathcal{M}_r$, using upsampling operators $\mathcal{U}_1, \ldots, \mathcal{U}_{r-1}$ (Figure 6)

$$\psi_i^k = \mathcal{U}_{i-1}\psi_{i-1}^k \in \mathbb{R}^{3n_i}, \quad \text{for } k = 1, \ldots, n_0, \text{ and } i = 2, \ldots, r. \tag{3}$$

Partition of unity proof. We now prove that the resulting functions ψ_i^k are still a partition of unity at each level; we call them "basis functions" (of each level) because they define a linearly independent basis of a subspace of scalar function on that level (Figure 7). We proceed by induction. Suppose ψ_i^k are a partition of unity of $\mathcal{M}_i$, i.e., $\sum_{k=1}^{n_0} \psi_i^k = [1 \ldots 1]^T$ (n_i 1s). Then,

$$\sum_{k=1}^{n_0} \psi_{i+1}^k = \sum_{k=1}^{n_0} \mathcal{U}_i \psi_i^k = \mathcal{U}_i \sum_{k=1}^{n_0} \psi_i^k = \tag{4}$$

$$= \mathcal{U}_i [1 \ldots 1]^T \ (n_i \text{ 1s}) = [1 \ldots 1]^T \ (n_{i+1} \text{ 1s}). \tag{5}$$

Last equality holds because each row of $\mathcal{U}_i$ sums to 1, for all i; this is because the sum of influences of vertices of $\mathcal{M}_i$ to any particular vertex of $\mathcal{M}_{i+1}$ must be 1. By the same argument, ψ_1^k (columns of $\mathcal{U}_0$) are also a partition of unity (base induction case). ■

Spatially-localized neural networks. At each LOD level $i = 1, \ldots r$, we use the basis functions ψ_i^k, $k = 1, \ldots, n_0$, to define n_0 spatially-localized neural networks of level i; each neural network predicts the residual in the spatially localized region of the support of ψ_i^k. The construction below is repeated separately for each $k = 1, \ldots, n_0$. We start with the residuals $Y_i^j \in \mathbb{R}^{3n_i}$ (Equation 1), for $j = 1, \ldots, N$. We then form the training shapes $\hat{Y}_i^{j,k} = \psi_i^k Y_i^j \in \mathbb{R}^{3n_i}$, where we have multiplied the residual of every vertex with the value of ψ_i^k at that vertex; this localizes the residuals to the support of ψ_i^k. We then perform dimensional reduction on $\hat{Y}_i^{j,k}$ using PCA along the index j [Bailey et al. 2018; Kry et al. 2002], expressing

$$\left[\hat{Y}_i^{1,k} \quad \hat{Y}_i^{2,k} \quad \ldots \quad \hat{Y}_i^{N,k}\right] \approx V_i^k \left[z_i^{1,k} \quad z_i^{2,k} \quad \ldots \quad z_i^{N,k}\right], \quad (6)$$

where $V_i^k \in \mathbb{R}^{3n_i \times d_i}$ is the PCA basis matrix, $z_i^{j,k} \in \mathbb{R}^{d_i^k}$, and $d_i^k \geq 0$ is the retained dimension of region k on level i (determined as explained in the next paragraph). Here, in notation, we indicated that the number of rows of V_i^k is $3n_i$, i.e., three DOFs for each vertex of $\mathcal{M}_i$, but of course due to the finite support of ψ_i^k, many (most) rows of V_i^k are zero, and this is exploited in our work to reduce memory storage. Next, we learn a neural network $\mathcal{N}_i^k$ that approximates the function $p \mapsto z_i^k$. We use a fully connected neural network with two hidden layers, with the tanh activation function. We experimented with various sizes of the two hidden layers and settled on $5 \cdot \max(P, d_i^k)$, where P is the pose dimension. As explained in [Bailey et al. 2018], we also perform per-neural network culling of input dimensions in p using perturbation analysis on each joint. Finally, we construct our deformation function N_i (Equation 2) as

$$N_i(p) = \sum_{k=1}^{n_0} V_i^k \mathcal{N}_i^k(p). \quad (7)$$

Selecting the number of retained dimensions. The value d_i^k is determined automatically by specifying a target average per-vertex per-training-frame PCA reconstruction error ϵ,

$$\frac{1}{n_i N} \left\| \left[\hat{Y}_i^{1,k} \quad \ldots \quad \hat{Y}_i^{N,k}\right] - V_i^k \left[z_i^{1,k} \quad \ldots \quad z_i^{N,k}\right] \right\|_F^2 = \quad (8)$$

$$= \frac{1}{n_i N} \sum_{\ell=d_i^k+1}^{\min(3n_i, N)} \sigma_\ell^2 < \epsilon^2, \quad (9)$$

where σ_ℓ are the singular values of the LHS matrix in Equation 6. We use a uniform global threshold ϵ for all regions and all levels; this means that the number of retained dimensions will vary across the model based on the deformation complexity; this ensures that all regions on the model resolve deformations to approximately the same quality level. By adjusting the parameter ϵ, one can trade the reconstruction accuracy for runtime reconstruction speed and required memory (Figure 8). Note that, depending on ϵ, some d_i^k may be zero; i.e., the L_2-norm of training residuals in a region is very small relative to the requested accuracy ϵ. In such a case, we do not train a neural network, but instead discard (cull) this region altogether. Culling enables us to localize computation to areas with the largest displacements.

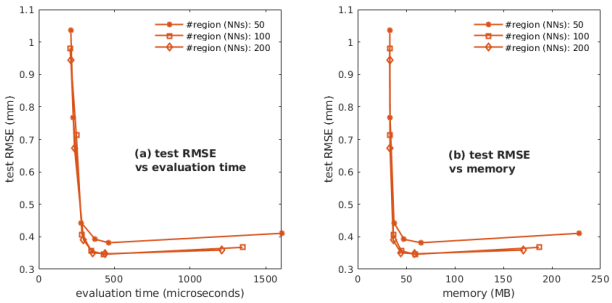

Fig. 8. **Trading approximation accuracy for speed and memory,** by adjusting the ϵ threshold. Note that the y-axis shows the testing error, i.e., difference to the ground truth on motions *unseen* during training. All the datapoints on the same curve were obtained by keeping the number of basis functions (n_0) constant and varying ϵ. Hand example (Figure 1). Timings correspond to evaluating all four resolution levels.

Table 1. **Computation time** to reconstruct one shape at runtime. Megan example (Figure 9). All times are in **micro**seconds, and averaged over the entire testing animation (521 frames). Intel Xeon(R) W-3275 CPU, 56 cores @ 2.5 GHz (only 28 effectively utilized), with 196 GB RAM. Each cell reports two times in the format A/B, to compare our work (time "A") to simply using prior work [Bailey et al. 2018] to separately process each level (time "B"). Our times correspond to the incremental time at each level; the total time to construct all levels is 697 microseconds. "NN time" is the time to evaluate neural networks. #NNs is the number of non-culled neural networks at each level, out of the $n_0 = 100$ available networks. Observe that, due to the incremental construction of deformation detail, our method has significantly fewer NNs at deeper levels than if separately processing each level.

LOD	#vtx	#NNs	NN **time**	$u = Uq$	upsampling	total
1	1,296	70 / 70	50 / 50	16 / 16	18 / 18	84 / 84
2	4,815	65 / 75	48 / 101	35 / 44	34 / 31	117 / 176
3	18,840	45 / 73	34 / 120	60 / 126	64 / 65	158 / 311
4	74,742	49 / 78	40 / 114	142 / 824	156 / 125	338 / 1063

Table 2. **Memory** to reconstruct one shape at runtime. Megan example (Figure 9). Same experiment and same A/B comparison as in Table 1. "NN mem" is the memory to store all the trained neural network coefficients. All values are in MegaBytes. For our method, we give the incremental memory needed at each level; our total memory for all levels is 79.7 MB. Our memory consumption is much smaller than for the compared method that separately processes each level.

LOD	#vtx	#NNs	NN **mem**	$u = Uq$	upsampling	total
1	1,296	70 / 70	3.4 / 3.4	1.1 / 1.1	0 / 0	4.5 / 4.5
2	4,815	65 / 75	3.5 / 6.1	6.2 / 8.8	0.3 / 0	10.0 / 14.9
3	18,840	45 / 73	1.7 / 6.8	13.8 / 35.0	1.1 / 0	16.6 / 41.8
4	74,742	49 / 78	0.6 / 7.0	43.5 / 201	4.5 / 0	48.6 / 208

6 RUN-TIME CODE AND MEMORY CONSIDERATIONS

Our work aims at hard-real-time animation suitable for computer games and VR systems, whereby the total computational budget to deform a character with 70,000 vertices is at most a few milliseconds. Under such constraints, it is necessary to employ certain code optimizations, as described next. We are not aware of any prior work

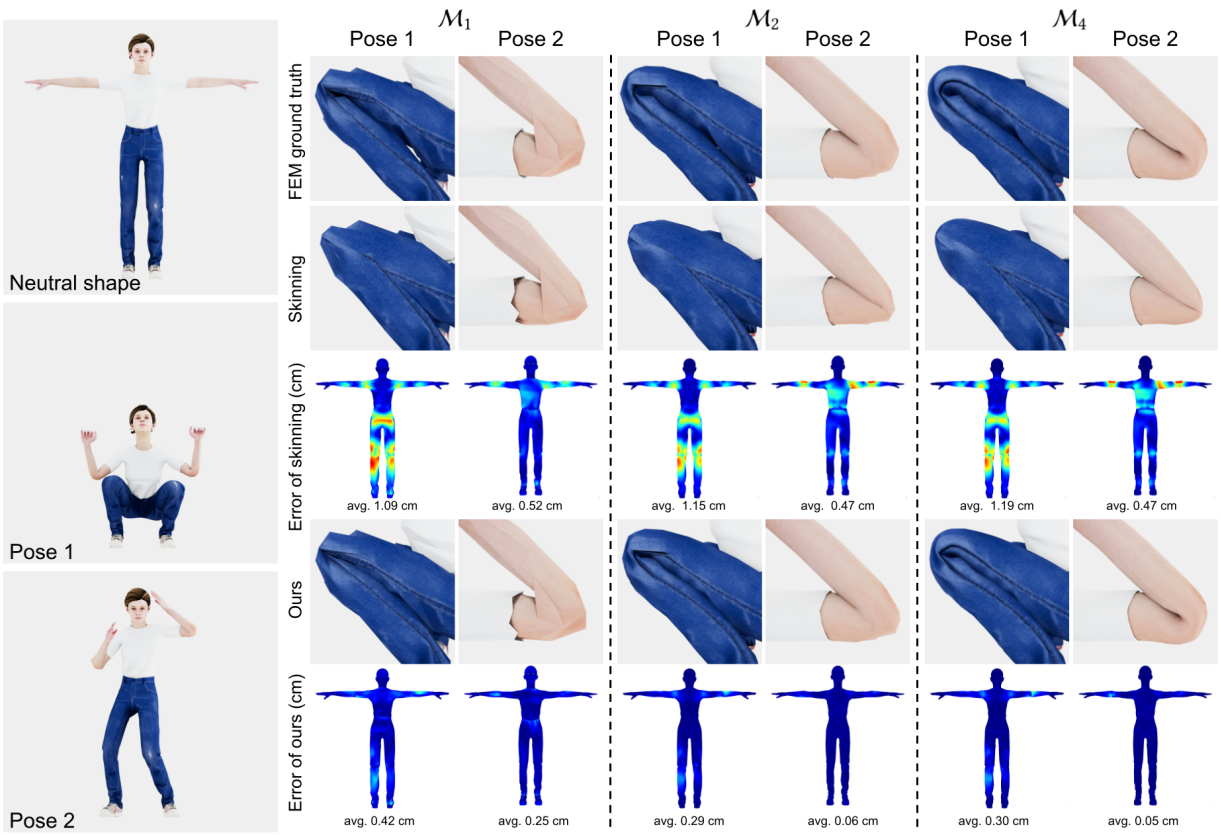

Fig. 9. **Hard real-time LOD character deformation.** The ground truth of this character was computed using FEM deformable simulation, including self-collision handling. Our real-time system is able to regenerate high-quality shapes, as well as interpolate them to new poses. Observe good-quality shape deformation near the characters' joints.

that has discussed such speed optimizations in the context of mesh LOD deformation computations for hard-real-time systems.

At runtime, we evaluate our neural networks using our own code; we do not use standard libraries such as PyTorch [Paszke et al. 2019]. We first used the C/C++ distribution of libtorch at runtime, but found it approximately 5-10x slower than our code. This is not surprising as such code is designed for general neural network applications as opposed to real-time systems. In time-critical sections, our code prefers C and avoids C++, and in particular, we do not use the algorithms from the Standard Template Library (STL). We avoid all dynamic memory allocations (no malloc), as these would otherwise drastically slow down performance. We allocate all memory for our runtime datastructures as one contiguous block. All the runtime data for each spatial region at each hierarchical level (trained neural network weights, modal matrices) is stored contiguously in memory, to improve read cache coherence.

As is commonly the case in high-performance applications, our runtime code is memory-bound, i.e., it takes significantly more time to read neural network and upsampling weights from memory, than to actually use them for computation. The largest computational cost is incurred in upsampling, and therefore we optimize memory reads of upsampling weights by storing them in 16-bit precision. Specifically, each vertex (on any LOD level i) has three non-negative weights w_1, w_2, w_3 that sum to 1; they are weights of vertices of a level $i-1$ triangle "driving" this vertex. We only store and read w_1 and w_2 from memory; they are represented as 16-bit unsigned integers i_1 and i_2, packed into the lowest and highest 16 bits of an 32-bit unsigned integer, respectively. At runtime, after reading i_1 and i_2 from memory, we can access w_1, w_2, w_3 by "hacking" the IEEE 754 single-precision floating point format. Namely, we exploit the fact that a single-precision FP number in the format 1.binarydigits (1 and binary digits after the decimal point) conveniently stores the binarydigits into the 23-bit mantissa. Note that ALU and floating point operations are very cheap compared to memory reads.

```
// Read 32-bits from memory, containing two 16-bit unsigned ints.
unsigned int packedWeights = memoryArray[...]; // read 32 bits
// Create FP representations of the two 16-bit unsigned integers,
// by interpreting the low and high 16-bits as FP mantissas.
unsigned int w1i = ((packedWeights & 65535) << 7) | (127 << 23);
unsigned int w2i =
  (((packedWeights >> 16) & 65535) << 7) | (127 << 23);
// View as float. Only writing to local variables w1 and w2
// stored in registers. No memory write occurs here.
float w1 = *(float*) &w1i; float w2 = *(float*) &w2i;
// Now, use w1 - 1.0f and w2 - 1.0f, and (3.0f - w1 - w2) to
// access barycentric weights w1, w2, w3.
```

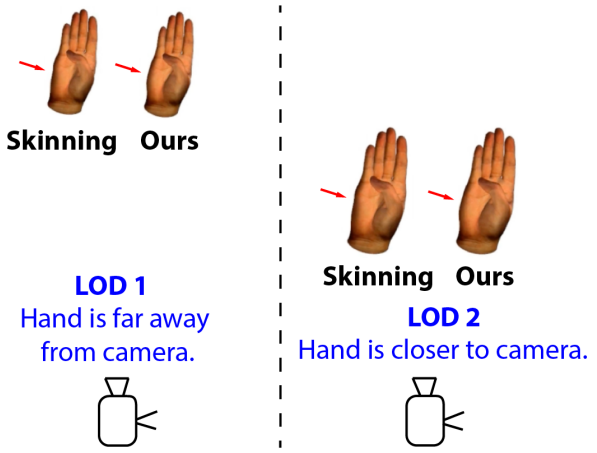

Fig. 10. **Shape changes are visible even under coarse LODs.** The shown sizes of hands at LOD 1 (coarsest LOD) and LOD 2 (second coarsest LOD) are correctly proportional to their differences to the camera. The coarsest LOD 1 is activated when the hand is far away from camera. Even in this case, there are silhouette differences between skinning and our method.

If a weight is 1.0, we re-order the weights so that $w_3 = 1$, and therefore $w_1 = w_2 = 0$. This is necessary because the above representation cannot store 1.0, but only values slightly smaller than 1: representable values range from 0 to $1 - 2^{16}$. Our approach accelerated upsampling by approximately 35% in our examples, at a negligible loss of runtime vertex position precision. Our upsampling speedup is applicable generally to any method that stores vertex quantities and needs to quickly upsample them using a triangle mesh LOD such as [Liu et al. 2021]. For fairness, we apply our code and memory optimizations to all the compared methods.

7 RESULTS

We demonstrate our method on several examples from computer animation practice, including human hands (Figure 1), a full-body cartoon character (Figure 9), and a four-legged animal (an elephant) whose skeleton is driven using IK (Figure 11).

Hands. Our training data was obtained from the project [Zheng et al. 2022]; it uses anatomically based FEM simulation with fat, muscles and bones. The linear blend skinning skeleton and weights were modeled and computed in Maya. Because our goal is real-time shape deformation in interactive applications (games, VR, Metaverse), we incorporated our method into an OpenGL real-time application (GLUT window manager), together with real-time skinning, dynamic vertex normals and real-time rendering (Phong shading) (Figure 5). Real-time dynamic normals are required in such applications to properly visualize the deformed shape, both under skinning only, or skinning plus our correctives. Therefore we include the time to compute dynamic normals in our "skinning" timings. Furthermore, hard-real-time application are particularly challenging when running times drop below 1 millisecond because cache misses become extremely expensive. Namely, one crucial aspect is whether the "runtime datastructures" (skinning weights, PCA basis matrices, neural network coefficients, upsampling weights)

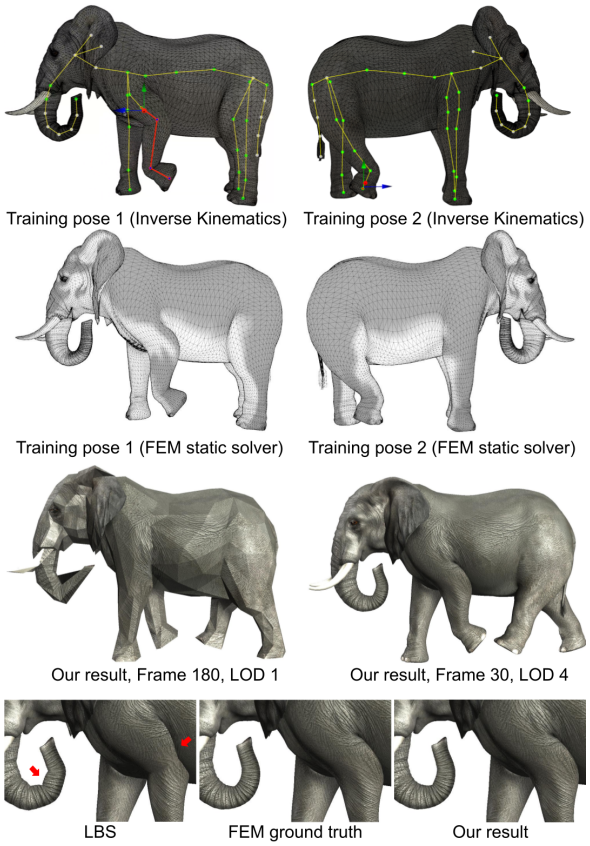

Fig. 11. **Hard-real-time LOD quadruped.** Top row: Two representative training poses generated using IK, with the skinned shape. Next row: the training shapes for the above two training poses (FEM deformable simulation with self-collision handling). Next row: Representative frame of our output, shown at LODs 1 and 4. Our real-time method produces high-quality shapes for animations that have not been seen during training, such as this walk cycle. Bottom row: zoom of the skinned shape (625 microsec per frame), FEM ground truth (29.7 seconds per frame), and our shape (1,142 microseconds per frame including skinning; **26,000**× faster than FEM). All performance numbers are at LOD 4 (33,346 vertices). Observe that, unlike skinning, our method resolves self-collisions near the elephant's hips.

are present in the memory caches ("hot cache"), or not/were evicted ("cold cache"). The typical situation in a real-time rendering system is a cold cache because the real-time renderer keeps evicting the runtime datastructures from the caches at each rendering frame; this effect can cause even a 2x slowdown of performance when in the hard-real-time regime. Cold-cache performance drop relative to the otherwise reported hot cache performance in our paper, is visible in our real-time video screen captures. Memory storage and consequently runtime performance (due to better caching) could be improved using techniques typically done in game industry such as by using lower-level APIs (e.g., Vulkan or Metal), or directly accessing the graphics drivers and window managers of each platform; we consider such optimizations beyond the scope of our academic work. In Figure 10, we demonstrate that our method brings benefits

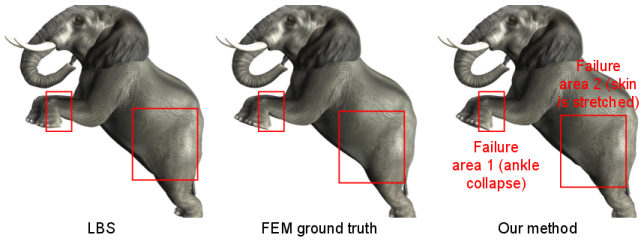

Fig. 12. **Limitation of our method.** Our training data does not contain poses where the elephant stands on its last feet, or where the front ankle is bent severely. Our method fails in such poses.

even when objects are far away from the camera, and not just when they are right in front of the camera (see also our Main Video). This establishes that it is useful to compute the correctives even at coarse LODs, as opposed to merely use skinning.

Cartoon character. Our cartoon character (Megan) comes from the Mixamo project [Mixamo 2024]. Mixamo provides an animated skeleton and skinning weights for the rendering mesh; we remeshed it in Maya to increase the resolution, and resampled the Mixamo skinning weights onto this new mesh, which then serves as our finest LOD4 mesh (the "skin"). Megan is simulated using FEM constrained dynamics similarly to [McAdams et al. 2011]. Specifically, we first create a "shrunken skin", by computing the signed distance field D against the character polygon soup geometry [Xu and Barbič 2014]. We then use isosurface meshing [Boissonnat and Oudot 2005] to create a quality surface triangle mesh of the skin ($D = 0$), and the shrunken skin ($D = -d$, where $d > 0$ is the "fat" thickness). Finally, we tet-mesh the volume between the shrunken skin and the skin [Hang Si 2011], obtaining FEM mesh for the body "fat". The "shrunken skin" is then equipped with skinning weights using Maya, and animated with the Mixamo animations. Our "fat" is constrained to this animated "shrunken skin", but is otherwise free to deform, subject to collisions and self-collisions. This gives us good quality skin deformations and contact at the character joints (Figure 9). The largest pre-skinning displacements occur near the joints, and are captured by our multi-resolution neural deformation functions F_i.

IK example. In our third example, we equipped a quadruped's (an elephant) skeleton with inverse kinematics. We drag (in real-time) IK handles to generate representative training skeleton poses; we record every mouse move and this resulted in 3646 training poses. In this stage, the elephant's shape is deformed using LBS, producing the expected visual artefacts, i.e., severe self-collisions at the hip joints. We created the skinning weights in Maya using Maya's "HeatMap" method, followed by manual improvements and smoothing using the popular "brSmoothWeights" Maya plugin. Next, we use FEM simulation (as described above with the "Megan" example [McAdams et al. 2011]) to compute correct FEM training shapes for the 3646 training poses, incorporating volume preservation and skin self-collision handling. Finally, we train our system using these training poses and FEM shapes. The end result is that we can then apply our trained method onto new motions (unseen during training) in hard real-time, such as a complete elephant walk cycle (Figure 11).

In Figure 13, we compare our method to using a single-level method of [Bailey et al. 2018] separately at each LOD level. Our method offers a large memory reduction. The goal of our work is to rapidly construct the shapes at any chosen hierarchical level (or levels) at runtime. For prior work to offer such a capability, it needs to separately train and store neural networks at each hierarchical level, essentially reconstructing the shape from scratch at each level; this significantly increases the memory footprint. Note that prior work cannot "naively" achieve such a capability by only training on the finest level and then downsampling the result to any chosen level, because this would always incur the computational cost of the finest level, even if only, say, the coarsest level is needed. The average PCA basis dimensions (for equal PCA reconstruction error) are also smaller in our method, enabling smaller neural networks. In the hand example (Figure 1) for our method, they are 13.1, 9.0, 5.6, 1.1, for LOD 1,2,3,4, respectively. They are 13.1, 13.7, 13.1, 13.1 when using the single-level method separately on each level; the difference is particularly salient at the finest LODs, where our method benefits from the work already done on coarser LODs. The same is observed in other examples. In terms of speed, our method is clearly better when all hierarchical levels need to be computed, and offers comparable speed when only a single level needs to be computed.

We use PyTorch [Paszke et al. 2019] to perform the training, using Adam's optimizer [Kingma and Ba 2015]. We use a learning rate of 0.01 at the start of training, and decay it by a factor of 0.999999 after each epoch, for 5000 epochs. The different regions are independent and can be trained in parallel. In Figure 14, we measure the training time of our method. As expected, the time depends on the accuracy parameter ϵ and the number of basis functions n_0. Figure 14 also compares to prior work [Bailey et al. 2018]: it can be seen that our training times are substantially faster, thanks to the lower dimensions of the per-level spatially localized regions made possible by the incremental nature of our shape construction. Faster training times permit artists to explore various training parameters to arrive at the optimal speed vs memory tradeoff.

In Figure 15, we compare our technique to Dual Quaternion Skinning [Kavan et al. 2008]. Because we learn from a quality rig, our method produces visually better shapes that resolve self-contact and preserve volume. Tables 1 and 2 give our performance statistics, Figure 12 shows failure, and Figure 16 analyzes multicore performance. Complete training data is shown in Supplementary Material.

8 CONCLUSION

We gave a technique to approximate and generalize arbitrary training mesh deformations using a set of hierarchically defined neural networks. We do so by employing a partition of unity at multiple resolution levels, and defining the shape functions using the mesh prolongation (i.e., upsampling) operator of a standard and widely used mesh simplification algorithm. These functions then guide our neural networks support and construction. The resulting technique permits a progressive computation of mesh deformations, enabling one to trade computational accuracy for speed. Our technique greatly decreases the required memory in applications involving meshes at multiple resolution levels; such applications are

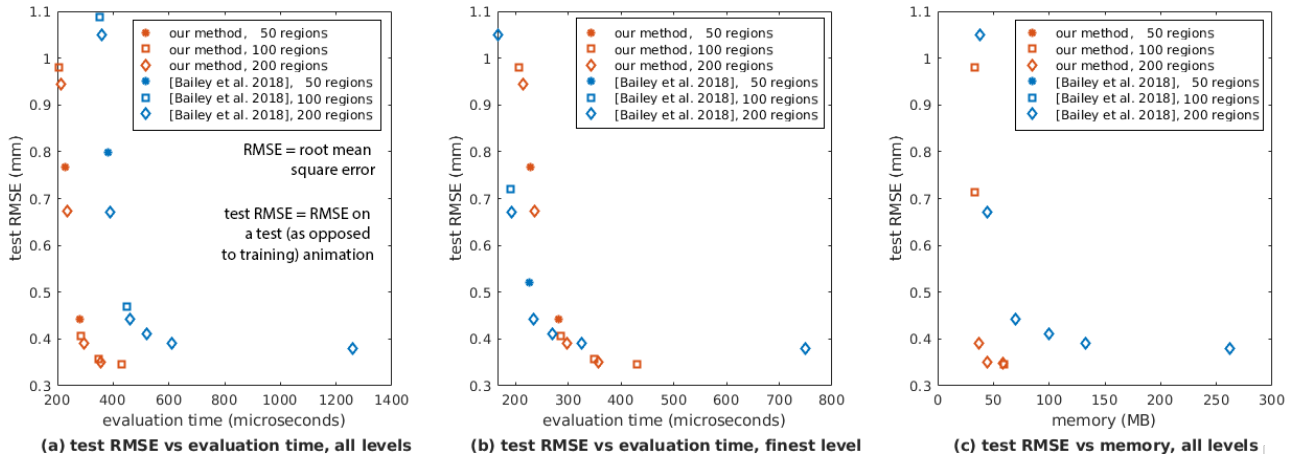

Fig. 13. **Comparison to a single-level method.** We train our LOD method and the single-level method [Bailey et al. 2018], under the same training dataset. Hand example (Figure 1). In these plots, we only vary the number of regions and the PCA approximation threshold. The purpose of varying the PCA approximation threshold is to control the speed VS accuracy (and memory VS accuracy) tradeoff. We repeat the experiment for several #regions (50, 100, 200), essentially performing a parameter sweep to discover the number of regions producing best results. The purpose of varying #regions is to ensure comparison fairness, as different methods may be optimal under different #regions. For plot clarity, we do not show datapoints where there is another datapoint that is better *both* in test RMSE *and* the running time (for the computational speed subplot) and in test RMSE *and* the memory (for the memory subplot). We do not compare the memory to compute a single level, because such a method would not be able to construct the entire LOD hierarchy (the goal of our paper).

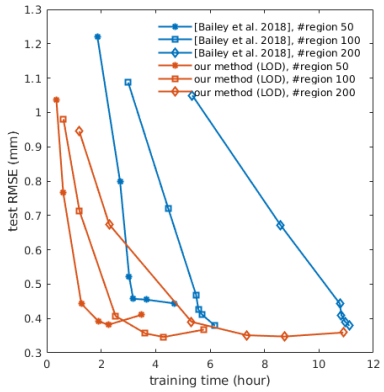

Fig. 14. **Training time of our multi-resolution method,** corresponding to Figure 13. Thanks to the incremental nature of our neural networks, the reduced dimensions of our regions are significantly lower than in prior work [Bailey et al. 2018], resulting in faster training times of our method.

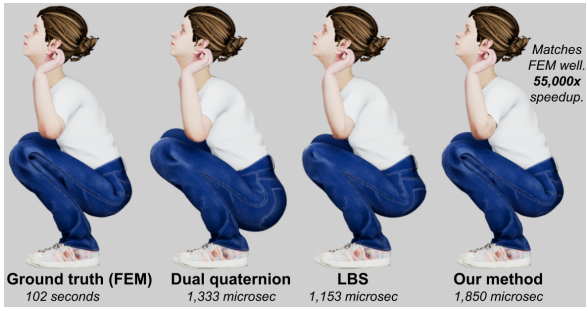

Fig. 15. **Comparison to dual quaternions.** Our method produces visually better shapes that closely match the ground truth (FEM simulation with self-contact handling and volume preservation), at a modest increase of computing time. This pose was **unseen during training**.

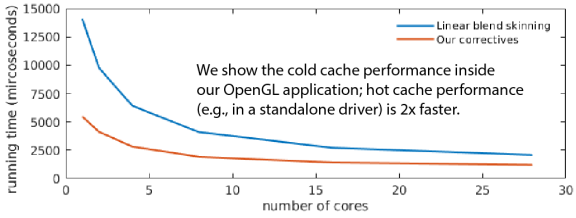

Fig. 16. **Multicore performance.** CPU performance vs number of cores on a complex example (72,414 vertices). Hand example, finest LOD (LOD 4).

common in computer games and other interactive systems. Our basis functions at all levels are prolongations of a partition of unity on $\mathcal{M}_0$. We considered an alternative scheme whereby the basis functions on $\mathcal{M}_{i+1}$ are obtained directly from the upsampling operator $\mathcal{U}_i$, without using coarser levels. This produces basis functions that become progressively narrower on progressive levels; but did not improve performance in our examples because our basis functions are occupying relatively narrow regions as is (Figure 7). Our technique requires good-quality training shapes, which is a standard, but time-consuming process. As common with learning methods, quality outside of the training dataset diminishes. We only use the CPU to compute skinning and correctives, but both of those could be offloaded to the GPU, either using vertex or compute shaders, or CUDA. In the future, we would like to employ our multi-resolution construction for other applications in computer animation, e.g., to model or approximate other signals defined on 3D meshes.

ACKNOWLEDGMENTS

This research was sponsored in part by NSF (IIS-1911224), USC Provost Fellowship to Mianlun Zheng, Bosch and Adobe Research.

REFERENCES

Ivo Babuška, Gabriel Caloz, and John E. Osborn. 1994. Special Finite Element Methods for a Class of Second Order Elliptic Problems with Rough Coefficients. *SIAM J. Numer. Anal.* 31, 4 (1994), 945–981.

Alberto Badías, David González, Icíar Alfaro, Francisco Chinesta, and Elías Cueto. 2020. Real-time interaction of virtual and physical objects in mixed reality applications. *Internat. J. Numer. Methods Engrg.* 121, 17 (2020), 3849–3868.

Stephen Wells Bailey. 2020. *Applications of Machine Learning for Character Animation.* University of California, Berkeley.

Stephen W. Bailey, Dalton Omens, Paul Dilorenzo, and James F. O'Brien. 2020. Fast and Deep Facial Deformations. *ACM Transactions on Graphics (SIGGRAPH 2020)* 39, 4, Article 94 (2020).

Stephen W. Bailey, Dave Otte, Paul Dilorenzo, and James F. O'Brien. 2018. Fast and Deep Deformation Approximations. *ACM Transactions on Graphics (SIGGRAPH 2018)* 37, 4, Article 119 (2018).

Hugo Bertiche, Meysam Madadi, and Sergio Escalera. 2021. PBNS: Physically based neural simulator for unsupervised garment pose-space deformation. *ACM Trans. on Graphics (SIGGRAPH Asia 2021)* 40, 6 (2021).

Ioana Boier-Martin, Denis Zorin, and Fausto Bernardini. 2005. A survey of subdivision-based tools for surface modeling. *DIMACS Series in Discrete Math and Theoretical CS* 67 (2005), 1.

Jean-Daniel Boissonnat and Steve Oudot. 2005. Provably good sampling and meshing of surfaces. *Graphical Models* 67, 5 (2005), 405–451.

Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. 2002. A Multiresolution Framework for Dynamic Deformations. In *Proc. of the Symp. on Comp. Animation 2002.* 41–48.

Dan Casas, Jesús Pérez, Miguel A Otaduy, Cristian Romero, et al. 2021. Learning Contact Corrections for Handle-Based Subspace Dynamics. (2021).

Nuttapong Chentanez, Miles Macklin, Matthias Müller, Stefan Jeschke, and Tae-Yong Kim. 2020. Cloth and skin deformation with a triangle mesh based convolutional neural network. In *Computer Graphics Forum*, Vol. 39. 123–134.

Gilles Debunne, Mathieu Desbrun, Marie-Paule Cani, and Alan H. Barr. 2001. Dynamic Real-Time Deformations Using Space & Time Adaptive Sampling. In *Proc. of ACM SIGGRAPH 2001.* 31–36.

Boyang Deng, John P Lewis, Timothy Jeruzalski, Gerard Pons-Moll, Geoffrey Hinton, Mohammad Norouzi, and Andrea Tagliasacchi. 2020. NASA: Neural articulated shape approximation. In *Proc. of Euro. Conf. on Comp. Vision (ECCV).* 612–628.

Lawson Fulton, Vismay Modi, David Duvenaud, David IW Levin, and Alec Jacobson. 2019. Latent-space dynamics for reduced deformable simulation. In *Computer Graphics Forum*, Vol. 38. 379–391.

T. A. Funkhouser and C. H. Séquin. 1993. Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments. In *Proc. of ACM SIGGRAPH 93.* 247–254.

Michael Garland and Paul S. Heckbert. 1997. Surface simplification using quadric error metrics. In *Proc. of ACM SIGGRAPH 97.* 209–216.

Eitan Grinspun, Petr Krysl, and Peter Schröder. 2002. CHARMS: A Simple Framework for Adaptive Simulation. *ACM Trans. on Graphics* 21, 3 (July 2002), 281–290.

Marc Habermann, Lingjie Liu, Weipeng Xu, Michael Zollhoefer, Gerard Pons-Moll, and Christian Theobalt. 2021. Real-time deep dynamic characters. *ACM Trans. on Graphics (SIGGRAPH 2021)* 40, 4 (2021), 1–16.

Hang Si. 2011. TetGen: A Quality Tetrahedral Mesh Generator and a 3D Delaunay Triangulator.

Daniel Holden, Bang Chi Duong, Sayantan Datta, and Derek Nowrouzezahrai. 2019. Subspace neural physics: Fast data-driven interactive simulation. In *Symp. on Computer Animation (SCA).* 1–12.

Hugues Hoppe. 1996. Progressive meshes. In *Proc. of ACM SIGGRAPH 96.* 99–108.

Alec Jacobson, Zhigang Deng, Ladislav Kavan, and John P. Lewis. 2014. Skinning: Real-time Shape Deformation. In *ACM SIGGRAPH 2014 Courses.*

Doug L. James and Dinesh K. Pai. 2003. Multiresolution Green's Function Methods for Interactive Simulation of Large-scale Elastostatic Objects. *ACM Trans. on Graphics* 22, 1 (2003), 47–82.

Ladislav Kavan, Steven Collins, and Carol O'Sullivan. 2009. Automatic linearization of nonlinear skinning. In *Proc. of Symp. on Interactive 3D Graphics and Games.* 49–56.

L. Kavan, S. Collins, J. Zara, and C. O'Sullivan. 2008. Geometric Skinning with Approximate Dual Quaternion Blending. *ACM Trans. on Graphics* 27, 4 (2008).

Ladislav Kavan, Dan Gerszewski, Adam W. Bargteil, and Peter-Pike Sloan. 2011. Physics-Inspired Upsampling for Cloth Simulation in Games. *ACM Trans. Graph.* 30, 4, Article 93 (2011), 10 pages.

Diederik Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *Int. Conf. on Learning Representations (ICLR).*

P. G. Kry, D. L. James, and D. K. Pai. 2002. EigenSkin: Real Time Large Deformation Character Skinning in Hardware. In *In Symp. on Computer Animation (SCA).*

Binh Huy Le and Zhigang Deng. 2014. Robust and accurate skeletal rigging from mesh sequences. *ACM Trans. on Graphics (SIGGRAPH 2014)* 33, 4 (2014), 1–10.

Binh Huy Le and John-Peter Lewis. 2019. Direct delta mush skinning and variants. *ACM Trans. on Graphics (SIGGRAPH 2019)* 38, 4 (2019), 113–1.

J. P. Lewis, Matt Cordner, and Nickson Fong. 2000. Pose Space Deformation: A Unified Approach to Shape Interpolation and Skeleton-Driven Deformation. In *Proc. of ACM SIGGRAPH 2000.* 165–172.

Peizhuo Li, Kfir Aberman, Rana Hanocka, Libin Liu, Olga Sorkine-Hornung, and Baoquan Chen. 2021. Learning skeletal articulations with neural blend shapes. *ACM Trans. on Graphics (SIGGRAPH 2021)* 40, 4 (2021), 1–15.

Hsueh-Ti Derek Liu, Jiayi Eris Zhang, Mirela Ben-Chen, and Alec Jacobson. 2021. Surface multigrid via intrinsic prolongation. *ACM Transactions on Graphics (SIGGRAPH 2021)* 40, 4, Article 80 (2021), 13 pages.

Lijuan Liu, Youyi Zheng, Di Tang, Yi Yuan, Changjie Fan, and Kun Zhou. 2019. Neuroskinning: Automatic skin binding for production characters with deep graph networks. *ACM Trans. on Graphics (SIGGRAPH 2019)* 38, 4 (2019), 1–12.

M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black. 2015. SMPL: A Skinned Multi-Person Linear Model. *ACM Trans. on Graphics (SIGGRAPH Asia 2015)* 34, 6 (2015), 248:1–248:16.

Ran Luo, Tianjia Shao, Huamin Wang, Weiwei Xu, Xiang Chen, Kun Zhou, and Yin Yang. 2018. NNWarp: Neural network-based nonlinear deformation. *IEEE Trans. on Visualization and Computer Graphics (TVCG)* 26, 4 (2018), 1745–1759.

A. McAdams, Y. Zhu, A. Selle, M. Empey, R. Tamstorf, J. Teran, and E. Sifakis. 2011. Efficient elasticity for character skinning with contact and collisions. *ACM Trans. on Graphics (SIGGRAPH 2011)* 30, 4 (2011).

Mixamo. 2024. Adobe Mixamo project, https://www.mixamo.com.

Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel. 2003. Multi-level partition of unity implicits. *ACM Trans. on Graphics (SIGGRAPH 2003)* 22, 3 (2003), 463–470.

Miguel A. Otaduy. 2004. *6-DoF Haptic Rendering Using Contact Levels of Detail and Haptic Textures.* Ph.D. Dissertation. Department of Comp. Science, University of North Carolina at Chapel Hill.

Miguel A. Otaduy, Daniel Germann, Stephane Redon, and Markus Gross. 2007. Adaptive Deformations with Fast Tight Bounds. In *Symp. on Computer Animation (SCA).* 181–190.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *arXiv preprint arXiv:1912.01703* (2019).

Cristian Romero, Dan Casas, Maurizio M Chiaramonte, and Miguel A Otaduy. 2022. Contact-centric deformation learning. *ACM Trans. on Graphics (SIGGRAPH 2022)* 41, 4 (2022), 1–11.

Cristian Romero, Miguel A Otaduy, Dan Casas, and Jesus Perez. 2020. Modeling and estimation of nonlinear skin mechanics for animated avatars. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 77–88.

Igor Santesteban, Elena Garces, Miguel A Otaduy, and Dan Casas. 2020. SoftSMPL: Data-driven Modeling of Nonlinear Soft-tissue Dynamics for Parametric Humans. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 65–75.

Igor Santesteban, Miguel A Otaduy, and Dan Casas. 2019. Learning-based animation of clothing for virtual try-on. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 355–366.

Steven L Song, Weiqi Shi, and Michael Reed. 2020. Accurate face rig approximation with deep differential subspace reconstruction. *ACM Trans. on Graphics (SIGGRAPH 2020)* 39, 4 (2020), 34–1.

Hongyi Xu and Jernej Barbič. 2014. Signed Distance Fields for Polygon Soup Meshes. *Graphics Interface 2014* (2014).

Zhan Xu, Yang Zhou, Evangelos Kalogerakis, Chris Landreth, and Karan Singh. 2020. Rignet: Neural rigging for articulated characters. *ACM Trans. on Graphics (SIGGRAPH 2020)* 39, 4 (2020), 14.

Jiayi Eris Zhang, Jérémie Dumas, Yun Fei, Alec Jacobson, Doug L James, and Danny M Kaufman. 2022. Progressive simulation for cloth quasistatics. *ACM Transactions on Graphics (SIGGRAPH Asia 2022)* 41, 6 (2022), 1–16.

Mianlun Zheng, Bohan Wang, Jingtao Huang, and Jernej Barbič. 2022. Simulation of Hand Anatomy Using Medical Imaging. *ACM Trans. on Graphics (SIGGRAPH Asia 2022)* 41, 6 (2022).

Mianlun Zheng, Yi Zhou, Duygu Ceylan, and Jernej Barbič. 2021. A deep emulator for secondary motion of 3d characters. In *Proceedings of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition.* 5932–5940.

Yongning Zhu, Eftychios Sifakis, Joseph Teran, and Achi Brandt. 2010. An efficient multigrid method for the simulation of high-resolution elastic solids. *ACM Trans. on Graphics (TOG)* 29, 2 (2010), 16.