NeuroSkinning: Automatic Skin Binding for Production Characters with Deep Graph Networks

LIJUAN LIU, NetEase Fuxi AI Lab
YOUYI ZHENG, State Key Lab of CAD&CG, Zhejiang University
DI TANG, NetEase Fuxi AI Lab
YI YUAN, NetEase Fuxi AI Lab
CHANGJIE FAN, NetEase Fuxi AI Lab
KUN ZHOU, State Key Lab of CAD&CG, Zhejiang University

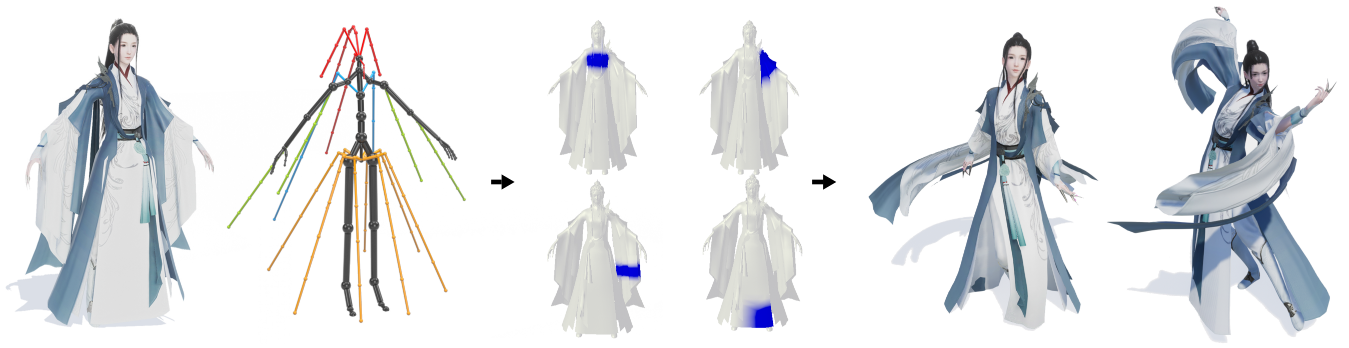

Fig. 1. Given a character mesh and its associated skeleton hierarchy in rest pose (left), our method automatically predicts skin weights (middle), which can produce high-quality deformations comparable to those generated by artist-painted weights (right).

We present a deep-learning-based method to automatically compute skin weights for skeleton-based deformation of production characters. Given a character mesh and its associated skeleton hierarchy in rest pose, our method constructs a graph for the mesh, each node of which encodes the mesh-skeleton attributes of a vertex. An end-to-end deep graph convolution network is then introduced to learn the mesh-skeleton binding patterns from a set of character models with skin weights painted by artists. The network can be used to predict the skin weight map for a new character model, which describes how the skeleton hierarchy influences the mesh vertices during deformation. Our method is designed to work for non-manifold meshes with multiple disjoint or intersected components, which are common in game production and require complex skeleton hierarchies for animation control. We tested our method on the datasets of two commercial games. Experiments show that the predicted skin weight maps can be readily applied to characters in the production pipeline to generate high-quality deformations.

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

Additional Key Words and Phrases: Animation, Skinning, Deep Learning

ACM Reference Format:
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. Graph.* 38, 4, Article 114 (July 2019), 12 pages. https://doi.org/10.1145/3306346.3322969

1 INTRODUCTION

Skinning techniques, such as Linear Blend Skinning (LBS) [Kavan and Žára 2005] and Dual Quaternion Skinning (DQS) [Kavan et al. 2007], are widely used in game production and supported by standard modeling software. In a typical skinning pipeline, an artist first creates a mesh model and specifies a skeleton hierarchy for the model. Skin weights are then painted onto the mesh to indicate how the mesh vertices deform with the skeleton. For complex character models in production, it is a time-consuming process to paint skin weights to produce satisfactory deformations even for professionals. For example, it took more than eight hours for an artist to paint the skin weights for the character shown in Fig. 1.

A few techniques have been proposed to automatically compute skinning weights using heat diffusion [Bang et al. 2015; Baran and Popović 2007; Wareham and Lasenby 2008], elastic deformer [Kavan and Sorkine 2012], or geodesic voxel binding [Dionne and de Lasa 2013]. They either work only for manifold meshes or assume that a skeleton bone has high influence weights on mesh vertices near to it.

[*]Corresponding authors: Youyi Zheng (youyizheng@zju.edu.cn) and Kun Zhou (kunzhou@acm.org).
Authors' addresses: Lijuan Liu, NetEase Fuxi AI Lab; Youyi Zheng, State Key Lab of CAD&CG, Zhejiang University, Hangzhou, 310058; Di Tang, NetEase Fuxi AI Lab; Yi Yuan, NetEase Fuxi AI Lab; Changjie Fan, NetEase Fuxi AI Lab; Kun Zhou, State Key Lab of CAD&CG, Zhejiang University, Hangzhou, 310058.

0730-0301/2019/7-ART114 $15.00
https://doi.org/10.1145/3306346.3322969

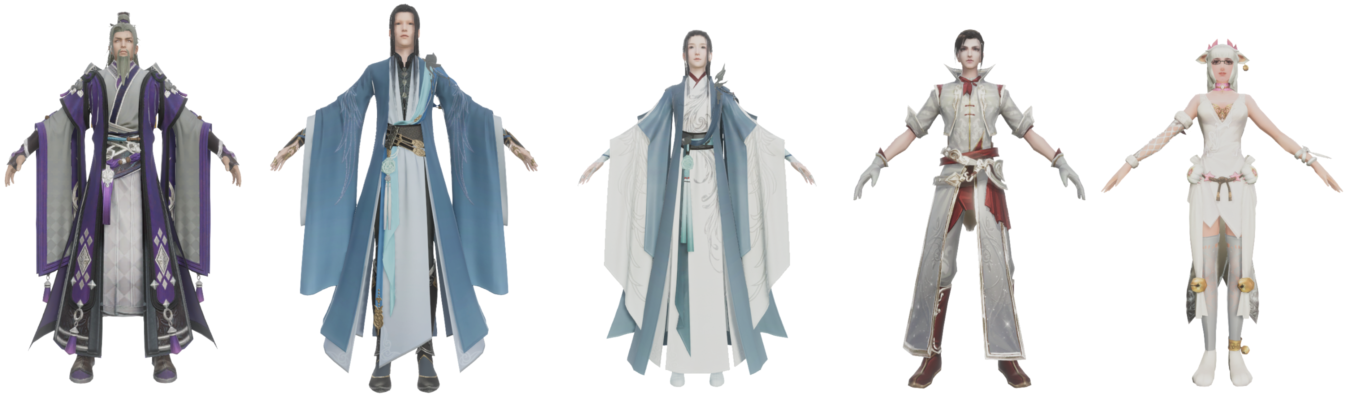

Fig. 2. Example production characters. These characters contain sleeve swings, dresses, cloaks, ribbons and so on. As an example, it is hard for previous binding solutions to separate sleeves from dresses because of the intersection between them. The left three are from Game A and the right two are from Game B.

Character models in game production, however, are often equipped with many pieces of accessories such as garments and armors, which could be non-manifold meshes and contain multiple disjoint or intersected components. In order to animate such characters through skinning, artists usually design complex skeleton structures consisting of a standard bipedal rig for the torso and auxiliary bones for accessories. For example, the character shown in Fig. 1 contains long sleeve swings, skirts, and ribbons, and some are non-manifold. The sleeves intersect with the skirts, and the skirts may intersect with legs. Many skeleton bones of this character are designed to control the deformation of the garment. It is difficult for previous techniques to properly deal with such models. In production, it largely relies on artists to resolve these difficult situations and manually paint the skin weights to generate satisfactory deformations.

In this paper, we seek a data-driven approach to automatically compute skin weights for game production characters. We assume that a set of character meshes have been rigged with skeletons and the skin weights have also been painted by artists. From this dataset, we aim to learn a model that can precisely depict the binding patterns between the meshes and skeletons, and thus can automatically infer the skin weights for a new character mesh with its associated skeleton hierarchy, which may not appear in the training dataset. The key challenge in developing such an inference model comes from two ends. First, as aforementioned, production meshes have complex, varying shapes and topologies. Characters, even in the same game, could be dressed with dramatically different garments (see Fig. 2). Second, although the basic bipedal rig for different characters is the same or very similar, the auxiliary bones could vary significantly as different characters have different accessories, leading to skeletons with different structures and different numbers of bones (see Fig. 14). It is thus difficult to formulate these two varying factors into a learning-based framework to find the binding patterns between them.

To this end, we introduce a deep-learning-based method to automatically compute skin weights for production characters. Given a character mesh and its associated skeleton hierarchy in rest pose, we construct a graph for the mesh, each node of which encodes the mesh-skeleton attributes of a vertex. We design an end-to-end deep graph convolution network to predict the skin weights for a character model, which is trained from a set of models with skin weights painted by artists. To account for non-manifold meshes with various topologies, our method leverages a graph-based convolutional unit that mimics the convolution and pooling operations in images for effective feature learning. The graph convolution unit, combined with multi-layer perceptron, enables our network to effectively learn the mesh-skeleton binding patterns. It also makes the network insensitive to the graph size and topology. To account for the varying skeleton structures in the training dataset, we propose the super skeleton and bone masks as a unified representation for all skeleton structures in the training set. Specifically, each bone of a skeleton in the training set is assigned a label, indicating its semantic usage (e.g., 'Spine' of the bip). The super skeleton is defined as a virtual configuration of the semantic union of all bones in the training set. The structure of an individual skeleton is then represented by a mask, indicating its valid bones in the super skeleton configuration. This representation enables us to handle various skeleton structures that may or may not be contained in the training dataset, as long as they are a subset of the super skeleton.

We have qualitatively and quantitatively evaluated our method on character models from two commercial games. Experimental results show that the skin weight maps predicted by our method can yield satisfactory deformations, and compare favorably with those generated by alternative solutions. Our method has also been integrated into commercial software (3Ds Max) and deployed in a game production pipeline, greatly reducing the time and efforts spent on skin painting.

In the following, we first review related work (Sec. 2), then overview the pipeline of our method (Sec. 3). The method and implementation details are explained in Sec. 4. Experiments are described in Sec. 5. Finally, Sec. 6 concludes the paper.

2 RELATED WORK

Skin Deformation. Skinning techniques can be roughly divided into physics-based [Kim et al. 2017; Mukai and Kuriyama 2016; Si et al. 2014], example-based [Le and Deng 2014; Loper et al. 2015], and geometry-based methods [Kavan et al. 2007; Kavan and Žára 2005].

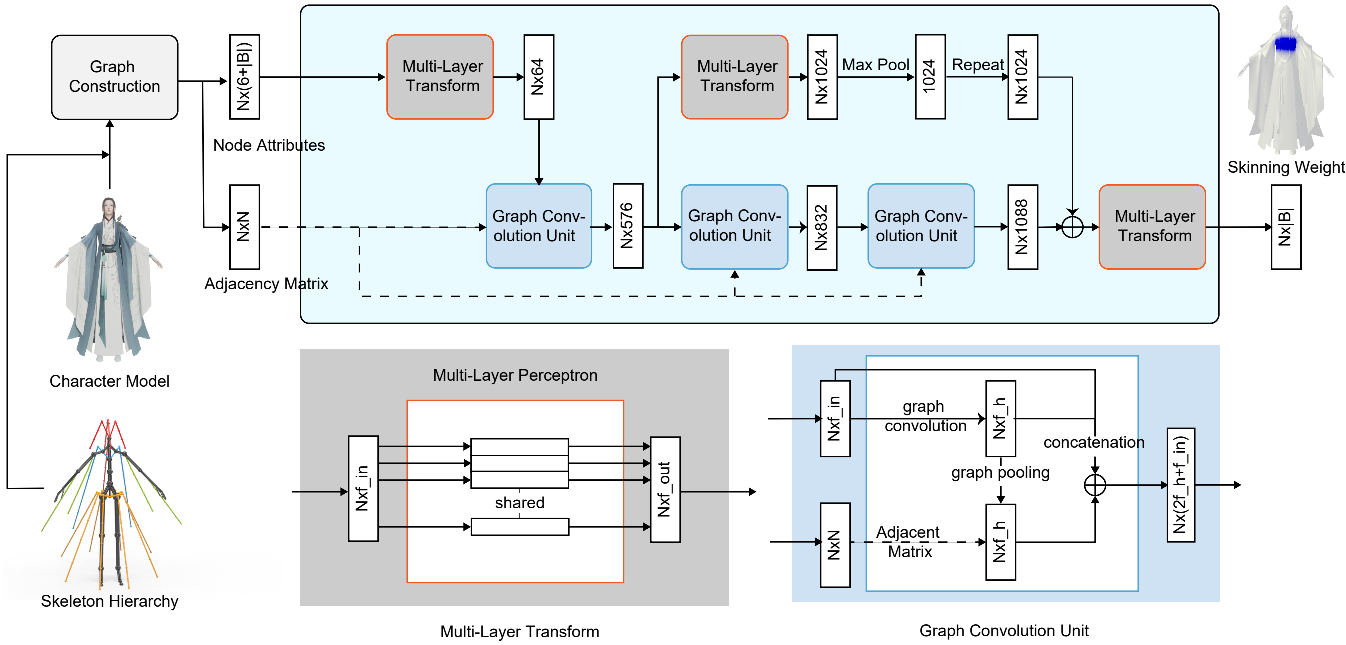

Fig. 3. The pipeline of our method. We first construct a graph for the input character mesh with its associate skeleton hierarchy. Each graph node encodes the mesh-skeleton attributes. The graph and node attributes are fed into our graph convolution net to predict the skin weight map.

Because of the computational efficiency and simplicity, geometry-based techniques, such as LBS and DQS, are widely used and supported in real-time applications such as video games.

In geometry-based techniques, the deformation quality heavily depends on the skin weights, and a few techniques have been proposed to compute weights automatically. Earlier methods exploit heat diffusion [Baran and Popović 2007], illumination models [Wareham and Lasenby 2008], Laplacian energy [Jacobson et al. 2011], and elastic energy function [Kavan and Sorkine 2012] to compute the skinning weights. Later methods utilize geodesic voxel binding to compute weights, which are capable of handling non-manifold and degenerated meshes [Dionne and de Lasa 2013, 2014]. Tomohiko and Shigeru [2016] propose a method for dynamic skinning with a helper bone rig, which is built with nuclear norm optimization and Binh [2016] reduces the joint-bulging artifact of DQS by estimating an optimized center of rotation (CoR) for each vertex. Bang et al. [2018] provide a spline-based skinning interface which integrates with an interpolation-based method and a diffusion-based model to compute skin weights. One common problem of these techniques is that they assume that a skeleton bone has high influence weights on mesh vertices close to it. The assumption, however, dose not hold well for production characters that may contain multiple intersected components, such as those shown in Fig. 1 and Fig. 2.

Example-based skinning methods (e.g., [Le and Deng 2014]) typically require multiple poses/meshes of a character as input and compute the skin weights for this specific character only. In this paper, we learn the mesh-skeleton binding patterns from a set of character models with skin weights painted by artists. We automatically predict skin weights for a new character, taking the character mesh and its associated skeleton hierarchy in rest pose as input.

Deep-learning-based Deformation. A few attempts have been made to extend deep neural networks to work with mesh deformations. In [Tan et al. 2018], a mesh-based autoencoder is proposed to extract localized deformation components with sparsity constraints. Bailey et al. [2018] propose a deep learning method to approximate the nonlinear portion of a character deformation with fully-connected networks. Luo et al. [2018] describe a re-usable deep neural network to add displacements to the results simulated from the linear elasticity so that the final results approximate nonlinear elastic deformations well. Qiao et al. [2018] introduce a GCNN-LSTM framework to cope with mesh animation sequences that the trained model improves the generation of mesh sequences. Different from these techniques, our method automatically computes skin weights for skin deformations using deep graph convolution networks.

Deep Graph Convolution. There is an increasing interest in generalizing convolutional networks for graph inputs in recent years. Attempts in this direction can be roughly categorized as spectral approaches [Bruna et al. 2013; Defferrard et al. 2016; Henaff et al. 2015; Kipf and Welling 2016] and spatial approaches [Boscaini et al. 2016; Hamilton et al. 2017; Masci et al. 2015; Monti et al. 2017; Veličković et al. 2018]. Our method falls into the second class. Spatial approaches define convolutions directly on the graph, operating convolution groups of spatially close neighbors. For example, Geodesic-CNN proposed in [Masci et al. 2015] is an early attempt at applying convolution directly to manifolds by representing manifolds with a set of patches represented in geodesic polar coordinates. AnisotropicCNN proposed in [Boscaini et al. 2016] introduces an alternative way to extract patches on manifolds by using anisotropic heat kernels. Monti et al. [2017] provide a mixture model network which allows designing convolution operations on both manifolds and graphs. GraphSAGE [Hamilton et al. 2017] is provided to compute node representation in a graph with a sampled fixed-size neighborhood. [Veličković et al. 2018] propose graph attention networks to aggregate an arbitrary number of neighbors to represent the current node with an attention mechanism. In our work, the skin meshes are converted into graphs and we take graph attention networks for reference to construct our graph convolution network.

3 OVERVIEW

Fig. 3 illustrates the pipeline of our method. The input is a character mesh with its associated skeleton hierarchy, all in rest pose. The output is the the skin weights for the character so that the mesh can be deformed along with the skeleton as desired. We first construct a graph for the mesh, with each graph node representing a vertex and encoding the mesh-skeleton attributes. The graph then is fed into a graph convolution network. The network first transforms the node attributes into features through a Multi-Layer Perceptron (MLP) layer and a graph convolution unit. Next, the learned features are fed into two network branches: a global branch which consists of multiple MLP layers and a Max Pooling layer, and a local branch which consists of multiple graph convolution layers. The transformed features are concatenated and fed into another MLP layer to finally get the skin weights. We consider the skin weight map as a label distribution – for each mesh vertex its weight vector is a selection of different bones with different probabilities. The loss function of our network is defined to minimize the difference between the prediction distribution and the ground truth distribution.

Our network is designed to consider both the locality and globality of the learned feature for mesh-skeleton attributes. It contains a global branch with MLP and Max Pooling to aggregate information across all the nodes to learn global features, and a local branch with attention-driven graph convolution and graph pooling for learning local spatial features. The global pooling operation provides two advantages. First, the learned feature can distinguish itself from others even though the local geometric structures are similar. Second, the global features of different character models are different, which can handle the problem of diversity of skin weights across different shapes. We have evaluated various network components (the convolution and pooling operations) and compare with alternatives (see Sec. 5.3).

4 THE METHOD

We first describe how to construct the graph for the input mesh and its skeleton. Then we explain the individual components, loss function and implementation of our network in details.

4.1 Graph Construction

Let $\mathbf{M}$ be a mesh with N vertices, and its skeleton hierarchy $\mathbf{B}$ has $|B|$ bones. We construct a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{A})$, in which $\mathcal{V}$ indicates a set of nodes that correspond to vertices in the mesh, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ indicates the edges, $\mathcal{A}$ is a $(0, 1)$ adjacency matrix of size $N \times N$ that indicates the connectivity of vertex pairs in the mesh with $a(i, j) \in \{0, 1\}$ for each undirected edge $(i, j) \in \mathcal{E}$. For a node $v_i \in V$ its neighborhood set is denoted by $\mathcal{N}(i)$.

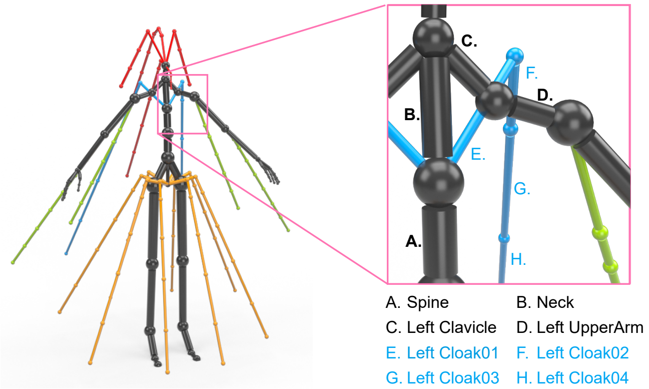

Fig. 4. The skeleton hierarchy used in Game A. *Black*: bip bones for body control; *yellow*: support bones for dresses; *green*: support bones for sleeves; *blue*: support bones for cloaks; *red*: support bones for hair-style.

Node attributes. We assign a mesh-skeleton attribute vector to each graph node to encode the geometry information of the corresponding mesh vertex and its relative position to the skeleton. Specifically, for a node $v_i \in \mathcal{V}$, its attribute vector is defined as $v_i = [p_i^T, n_i^T, d_i^T]$, in which $p_i \in \mathbb{R}^3$ is the vertex position in the Cartesian coordinate system, $n_i \in \mathbb{R}^3$ is the normal of the vertex, and $d_i \in \mathbb{R}^{|B|}$ is the distance vector to the skeleton bones, i.e., $d_{i,j}$ is the shortest distance from p_i to the line segment of the j-th bone. The dimension of v_i is $6 + |B|$.

Super skeleton and skeleton mask. The structure and the number of bones could vary among skeletons in the training set. As the input and output of our network all assume a fixed bone number $|B|$, we construct a super skeleton $\check{s}$ as the union of all skeleton structures in the training set, and fix $|B|$ to be the size of $\check{s}$. Note that $\check{s}$ is a virtual configuration and does not have actual geometry layout. Any individual skeleton structure in the training set can be represented by $\check{s}$ and a mask vector indicating the validity of each bone in $\check{s}$. Specifically, for each bone in individual skeletons, we manually assign a semantic label of five classes, including bip, dress, sleeves, cloaks, and hair-style. Each bone can be further assigned a label indicating its actual usage in its class, such as 'Spine' and 'Neck' in the bip class and 'Left Cloak01' in the cloaks class (see Fig. 4). We then group and align bones of the same usage label together to get the super skeleton. For an individual skeleton, a $(0, 1)$ mask vector M of size $|B|$ is generated to indicate its valid bones.

Before constructing graphs, all models need to be normalized. We guarantee that all models are located in the same Cartesian coordinate system by moving a common bone (i.e., the spine) to the origin of the coordinate system, and rotating the models to a fixed orientation. Then we scale all the models along the height direction into the range of $[0, 1]$.

4.2 Multi-Layer Transform

In our network, we transform the input node attributes into higher-level features and preserve the spatial proximity using Multi-Layer Perceptron (MLP), which is called multi-layer transform in Fig. 3. The MLP contains a group of fully-connected layers whereas the number of hidden layers is set through experiments (see Sec. 4.5).

The MLP is known to allow approximate solutions for extremely complex problems. As in [Qi et al. 2017], we use shared multi-layer perceptron to handle unordered input, i.e., all nodes share a single copy of MLP.

4.3 Graph Convolution Unit

The graph convolution unit takes node features and the adjacency matrix as input and outputs transformed node features. In our proposed unit, the graph convolution operation is used to learn the node features by weighting the neighborhood features, and the graph pooling operation is used to enlarge the receptive field for every node. We describe graph convolution and graph pooling in details here.

Graph Convolution. Graph convolution is defined to translate input node features within the graph to latent features so that, for every node feature, the output feature aggregates the local features in the neighborhood weighted by trainable filter coefficients. A well-defined graph convolution operator should handle the unordered and size-changed neighborhoods of nodes. Inspired by [Veličković et al. 2018], we define the graph convolution operator as a masked self-attention function on nodes.

With the input node features, $\mathbf{f} = \{\vec{f_1}, \ldots, \vec{f_i}, \ldots, \vec{f_n}\}$, $\vec{f_i} \in \mathbb{R}^F$, where n is the number of input features and F is the input feature dimension, the output of the graph convolution layer is denoted as $\mathbf{f}^{'} = \{\vec{f}_1', \ldots, \vec{f}_i', \ldots, \vec{f}_n'\}$, $\vec{f}_i' \in \mathbb{R}^{F'}$ and F' is the output feature dimension. The graph convolution operation is separated into three steps:

- First, for every input node feature, a linear transformation parameterized by a weight matrix $\mathbf{C} \in \mathbb{R}^{F' \times F}$ is applied to get higher-level output,

$$\vec{h}_i = C\vec{f_i}, \tag{1}$$

  where the weight matrix $\mathbf{C}$ is trainable and can be shared over input features.
- Then a masked attention mechanism is introduced to weight the importance of different nodes in the neighborhood. Starting from an attention function $\phi : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \longrightarrow \mathbb{R}$, the importance of node j to node i is computed as $e_{ij} = \phi(\vec{h}_i, \vec{h}_j)$. Instead of using concatenation as in [Veličković et al. 2018], we set the attention function ϕ as the dot product, which demonstrates faster convergence and stable training in practice [Thekumparampil et al. 2018]. Then the graph adjacency matrix mask $\mathcal{A}$ is applied to the score map to keep the neighborhood locality such that, for each node i, only the nodes $j \in \mathcal{N}(i)$ is computed:

$$e_{ij} = \phi(\vec{h}_i, \vec{h}_j) * a_{ij}, \tag{2}$$

  where $*$ means element-wise production and $a_{ij} \in (0, 1)$ indicates the connection between vertex i and j. The attention coefficients are then normalized with the softmax function:

$$\alpha_{ij} = softmax_j(e_{ij}) = \frac{exp(e_{ij})}{\sum_{k \in \mathcal{N}(i)} exp(e_{ik})}. \tag{3}$$

- The final output feature for every node is formulated as a linear combination of the corresponding features in the neighborhood with the attention coefficient (with nonlinearity σ applied):

$$\vec{f}_i' = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} \vec{h}_j\right). \tag{4}$$

  The multi-head attention introduced in [Vaswani et al. 2017] is applied here to extend the output feature to stabilize the learning process. The final output feature is the concatenation of K independent attention mapped linear transformations:

$$\vec{f}_i' = \|_{k=1}^{K} \, \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}^k C^k \vec{f}_i\right). \tag{5}$$

Graph pooling. We have two pooling operations, i.e., the global Max pooling (c.f. [Qi et al. 2017]) and the local graph pooling. The later is used to aggregate node features from the neighborhoods. Specifically, for the node feature $\vec{f}_i'$, we seek J closest nodes across the graph based on geodesic distances computed with the input features and adjacency matrix (in case of graph disconnection, the geodesic distance is set to infinity), and then apply average pooling on the neighbor features set to get the local neighborhood information $AvgPool(\vec{f}_j' | j \in J)$. Finally, we concatenate the pooled feature to the node feature to enhance the locality for every node, the output feature after applying local pooling is

$$\vec{f}_{out}' = Concat(\vec{f}_i', AvgPool(\vec{f}_j' | j \in J)). \tag{6}$$

The size of the local neighbors J indicates the receptive field of this local pooling operation and the concatenation operation enhances the locality information to the output features.

We concatenate the output of the proposed unit with the input feature. As demonstrated in Fig. 3, for the input feature dimension F, the convolution output dimension F' and the number of heads for attention K, the output feature dimension of the graph convolution unit is $2(KF') + F$.

4.4 Loss Function

Depending on different applications, the skin weight map W output by our network merits different constraints. Basically, it is required to be convex, i.e. $w_{ij} \geq 0$ and $\sum_j^{|B|} w_{ij} = 1$, where $w_{ij} \in [0, 1]$ indicates how much the j-th bone's rotation and translation influence vertex p_i during deformation. In addition, the weight map is always sparse for the computation efficiency of deformation. Instead of enforcing these constraints in the loss function, we apply a softmax layer in the network to scale the weight to $[0, 1]$ before computing the loss. The final weights are further thresholded and normalized to enforce sparsity and convexity, i.e., we only keep the top κ most influential bones for each vertex, where κ is a dataset-dependent parameter (see details in Sec. 5).

For the loss function, a simple choice is the regression loss such as the Euclidean distance between the predicted weight map and the ground truth. However, it is known that such an $L2$ loss function tends to blur the prediction results [Isola et al. 2017]. In our situation, we consider the weight map as a label distribution so that, for every vertex v_i with the whole skeleton hierarchy $\mathbf{B}_{|B|}$, the weight vector $\{w_{ij} | j \in |B|\}$ is a selection of different bones with different probabilities. The learning target here is converted to minimize the difference between the prediction distribution and the ground truth distribution. f-divergence, specifically, the Kullback-Leibler divergence is widely used to measure the distance between two distribution. In our case, based on the nature of our problem, we use the Kullback-Leibler divergence loss, $\mathcal{L} = \sum_{i=1}^{N} \sum_{j=1}^{|B|} w_{ij}(\log \frac{w_{ij}}{w'_{ij}})$ to minimize the distance between the predicted distribution and the ground truth distribution.

4.5 Implementation Details

In this section, we provide some implementation details of the proposed network.

As shown in Fig. 3, the network is divided into two stages. In the first stage, we feed the node attributes to a multi-layer perceptron with hidden feature sizes of 128 and 64, then apply the first graph convolution unit to the resulting features with the hidden feature size of 64, the heads number of 4, and graph pooling receptive size of 16. In the global featuring branch of the second stage, the features are transformed with a multi-layer perceptron with the hidden size of [512, 1024]. The following max pooling layer operates over all input features, leading to the feature size of [1, 1024]. Subsequently, this feature is duplicated N times to get the feature size of [N, 1024] (the top branch in Fig. 3). In the local branch, the input features are processed by two graph convolution units with the hidden feature size of [64, 64], the heads number of [2, 2], and pooling receptive size of [24, 64]. The local features, together with the replicated global feature, are taken as input to the classifier unit represented by three fully connected layers with the hidden feature size of [1024, 512, $|B|$].

During the training phase, we set the batch size as 12 and choose the Adam solver for optimization, with the initial learning rate of $lr = 10^{-3}$. The learning rate decays once every 200 epochs with a decay rate of 0.99.

It is noteworthy that, similar to PointNet, our network architecture is not affected by the number of graph nodes or vertices (N). During training, we randomly select N vertices from each character model as a training sample for stable training and also saving of GPU memories. The adjacency matrix for each sample is updated accordingly. Once trained, the network is directly applied to the original mesh of a new character model, without sub-sampling vertices. In our experiments, we found that $N = 2048$ can suffice to capture the mesh connectivity (which affects the graph convolution operation) and produce satisfactory results. Training with more vertices does not provide obvious improvement. Note that the MLP operation is agnostic to the number of input vertices.

5 EXPERIMENTS

We have conducted a group of experiments to evaluate the proposed method with two datasets from commercial games. We name the two datasets as Game A and Game B, respectively. We train our network separately on the two datasets. All characters in the datasets have been skinned by professional artists and can deform with high quality even under extreme poses. The artists constructed some complex skeleton hierarchies, including bipedal bones to animate the torso, and other bones to animate the dresses, sleeves, streamers and so on (see Fig. 4). The characters in the datasets can be deformed with a group of skeleton motions, such as dancing and walking. Fig. 5 and Fig. 6 show some example motions. More examples are shown in the supplementary material. We also make public the datasets to facilitate further academic research in this field [1]. The details of datasets are listed below.

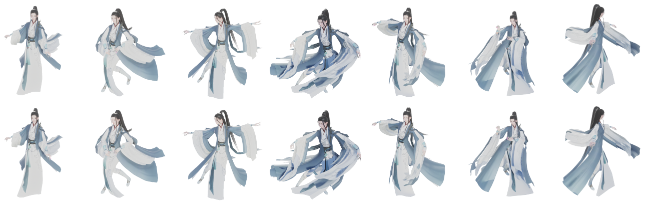

Fig. 5. Example poses of a character in Game A. The top row is the ground truth meshes deformed by artist-painted weights, and the bottom row is the meshes deformed by our predicted weights.

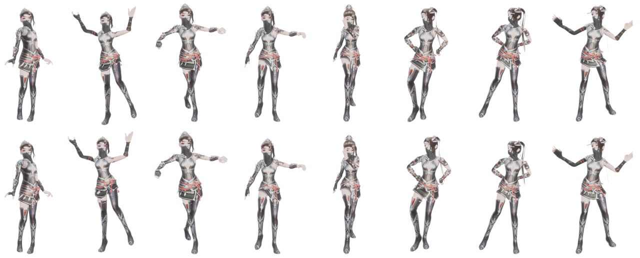

Fig. 6. Example poses of a character in Game B. The top row is the ground truth meshes deformed by artist-painted weights, and the bottom row is the meshes deformed by our predicted weights.

Game A. There are about 500 characters in this dataset. Fig. 2 (the left three) shows a few examples. As shown, all the models wear complex, ancient costumes. The number of vertices of a mesh ranges from 5,000 to 40,000, and most meshes are non-manifold and composed of multiple disjoint components. There are also intersections between sleeves and dresses. Table 1 provides the statistics of five characters shown in Fig. 13. All characters share the same skeleton hierarchy, which makes the super skeleton unnecessary. The skeleton consists of 240 bones (see Fig. 4). For computational efficiency, a vertex is set to be influenced by no more than 4 bones (i.e., $\kappa = 4$). The dataset is randomly divided into a training set of 400 models and a testing set of about 100 models.

Game B. This dataset has 26 different skeleton structures, each of which is shared by 20-50 characters, resulting in 1171 characters in total. The number of vertices of a mesh ranges from 1,000 to 5,000, and the number of bones ranges from 80 to 230. Fig. 14 shows several example characters, and the statistics are listed in Table 2. The number of bones influencing a vertex is no more than 3. The super skeleton created for this dataset contains 416 bones. We divide the dataset into training and testing sets randomly and ensure that for each skeleton structure, 80% of the characters are in the training set and others are in the testing set.

[1] http://fuxi.163.com/en/thesis/NeuroSkinning.html

Table 1. Statistics for five testing models in Game A (shown in Fig. 13).

Character	#Vertices	#Parts	#Height (cm)	#Bones
I	13170	192	162	240
II	10447	190	159	240
III	19342	379	165	240
IV	21104	416	161	240
V	25413	330	155	240

Table 2. Statistics for five testing models in Game B (shown in Fig. 14).

Character	#Vertices	#Parts	#Height (cm)	#Bones
I	3348	34	172	111
II	2120	24	170	111
III	3556	82	148	189
IV	2374	33	186	140
V	2839	69	179	133

Table 3. Evaluation of the predicted weight maps for five characters in Game A. *precision* and *recall* are the precision and recall scores of bone selection. *mean error* is the per-vertex skin weight prediction error on average, and *max error* is the max error.

Character	precision	recall	mean error	max error
I	0.813	0.982	0.0914	1.32
II	0.847	0.986	0.0427	1.78
III	0.836	0.987	0.0583	1.69
IV	0.707	0.986	0.0726	1.83
V	0.648	0.991	0.0890	1.49

Table 4. Evaluation of the deformation quality for five characters in Game A. *mean dist.1* (*max dist.1*) is the per-vertex average (max) distance error of the 1st extreme pose (Fig. 13). *mean dist.2* / *max dist.2* is the average (max) error of the 2nd extreme pose. The distance errors are normalized by the average height of all testing characters.

Character	mean dist.1	max dist.1	mean dist.2	max dist.2
I	0.0019	0.0583	0.0021	0.0507
II	0.0011	0.0896	0.0012	0.0482
III	0.0016	0.1499	0.0009	0.2161
IV	0.0014	0.0961	0.0009	0.0596
V	0.0012	0.0921	0.0010	0.0596

All results shown and the statistics reported in the following are on the test characters.

5.1 Quality Metrics

Several metrics are used to evaluate the quality of the predicted weight maps as well as the resulting deformations. For weight maps, one important metric is the precision and recall of the selection of influencing bones for every vertex, as the wrong selection would cause unexpected deformations. We set the bone selection results I_{pred} as

$$I_{pred}(i,j) = \begin{cases} 1, & w_{ij} \geq thres \\ 0, & w_{ij} < thres \end{cases} \tag{7}$$

where $I_{pred}(i,j) = 1$ means the j-th bone influences the i-th vertex, and $thres = 0.0001$ is a threshold. For the ground truth weight maps, the bone selection I is also computed according to Eqn. (7) with $thres = 0$. Then the precision and recall scores are computed between I_{pred} and I over all testing data. Another direct metric for evaluating skin weights is the difference between the predicted weight map and ground truth, i.e., the $L1$ distance between the weight vectors for every vertex, which can be visualized as a heat map over the mesh.

To evaluate the deformation quality, we measure the average distance error and the max distance error between the deformed models with the predicted weights and the ground truth weights. The characters are deformed with a set of action poses first, then two poses with the largest average errors in the set are selected to evaluate the deformation quality.

5.2 Results

Results on Game A. For the bone selection, the overall precision and recall scores of the bone selection are 0.743 and 0.982 respectively. Table 3 lists detailed scores for the five characters shown in Fig. 13. The results demonstrate that our method can effectively learn the mesh-skeleton binding patterns from the training data.

Table 3 also shows the errors of the predicted weight maps from the ground truth. The error distribution over the mesh is visualized in Fig. 7. It can be observed that regions with large errors are always found in the shoulder, pelvis, and the bottom of the dress. The deformation of the shoulder and pelvis areas is influenced by more than two bones. For example, the vertices around the shoulder are controlled by the spines, clavicle, and upper arm (Fig. 4). In these areas, different artists could paint different weights for different characters, which may introduce noises in the training data. The large errors near the bottom of the dress are caused by the large variation of geometry in these areas across the training data. We expect that an argumentation of the training data can alleviate both problems.

For the deformation quality, the per-vertex distance error on the testing set is 0.00066 on average, and the max error is 0.21612, normalized by the average height of all testing characters. Table 4 shows the detailed numbers for five characters. As demonstrated in Fig. 5 and the supplementary video, our predicted weight maps can produce high-quality deformations comparable to those generated by artist-painted weight maps.

Results on Game B. In this dataset, the overall precision and recall scores of the bone selection are 0.803 and 0.945 respectively. The higher recall scores in both Game A and Game B may be due to

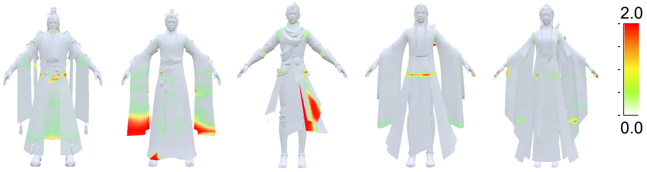

Fig. 7. Per-vertex skin weight prediction errors on five models in Game A.

the fact that our prediction does not enforce the sparsity of the weights (the ground truth weights are usually sparse). Table 5 lists the detailed numbers for several testing characters, as well as the mean and max errors of the predicted weight maps. Fig. 14 visualizes the error distribution over the mesh. The results demonstrate the efficacy and robustness of our method with varying skeleton hierarchies.

Fig. 6 and Fig. 14 show several deformation results using the predicted weight maps. The per-vertex distance errors of the deformed meshes are reasonable (Table 6). As demonstrated in the figures and supplementary video, there are no noticeable errors. According to the feedback of our artists, many of the predicted models (e.g., the first and the third characters in Fig. 14) can be directly used in production without any adjustment, while some may require slight modifications in local regions.

Generalization. Our method is designed to work with characters having new skeleton structures not contained in the training set. Fig. 9 shows two such examples (predicted using the network trained on Game B). As described in Sec. 4.1, we first align the skeleton to the super skeleton of the dataset and get the skeleton mask. Our network can then be applied to this character to predict the skin weight map. We provide side-by-side comparisons of the deformed meshes in Fig. 9. As shown here and in the supplementary video, the characters can be deformed well.

To further evaluate the generalization ability of the trained network, we also tested it with several humanoid characters downloaded from the Internet. We use the network trained on the dataset of Game A to predict their skin weights. The results are shown in Fig. 10 and the corresponding animation can be found in the accompanying video. As shown, our trained network works well on humanoid characters whose shapes do not significantly differ from the training examples. On the other hand, if the geometry varies dramatically, such as the monster model with long and fat arms shown in Fig. 8, artifacts will be introduced. The model is deformed slightly away from the rest pose. The hand region is incorrectly bound to the bones of sleeves. This verifies the fact that the performance of our trained network could decrease if the topology and skeleton layout are quite different from the training set.

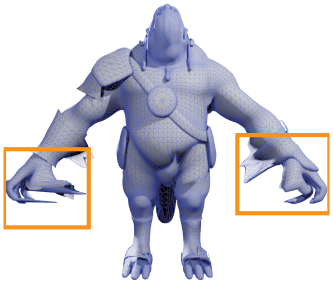

Fig. 8. A failure example.

As our method is learning-based, it requires sufficient training data to achieve satisfactory results. In dataset A, we alternatively reduce our training set size to be 25% and 50% of the original dataset (while keeping the testing set unchanged) to see how the training size will influence the performance. We collect the precision and recall scores for both tests. We get (precision = 0.4967, recall = 0.8488) for the training size of 25% and (precision = 0.6211, recall = 0.9038) for the training size of 50%. Indeed, the increasing of training data will boost the performance.

Table 5. Evaluation of the predicted weight maps for five characters in Game B.

Character	precision	recall	mean error	max error
I	0.903	0.996	0.01135	1.203
II	0.856	0.875	0.1647	1.541
III	0.712	0.992	0.06936	1.674
IV	0.823	0.925	0.1760	1.481
V	0.792	0.936	0.1393	1.548

Table 6. Evaluation of the deformation quality for characters in Game B.

Character	mean dist.1	max dist.1	mean dist.2	max dist.2
I	0.0003	0.0748	0.0004	0.0774
II	0.0019	0.0775	0.0023	0.0762
III	0.0011	0.0423	0.0018	0.0436
IV	0.0025	0.0441	0.0027	0.0902
V	0.0022	0.0856	0.0027	0.0902

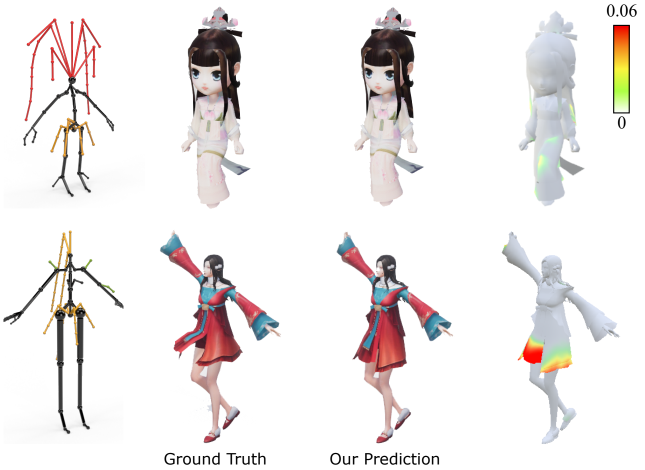

Fig. 9. Skin binding for characters that have skeleton structures not covered in the training set. Notice that the character in the bottom row shows larger prediction errors, mainly due to the incorrect binding of the skirt to the leg bone.

Deployment in game production. Our method is being actively used in game production. Both games mentioned in the paper require to frequently update characters with new costumes (often weekly), which is common for popular online games. The network trained on existing characters is used to compute skin weights for updated characters, which greatly reduces the manual labors in skin weight painting. In the future, for a new game which may has characters of dramatically different skeleton structures or geometries, we can let artists paint skin weights for a small portion of characters, from which we train a network. This network can then be applied to other characters to automatically compute skin weights.

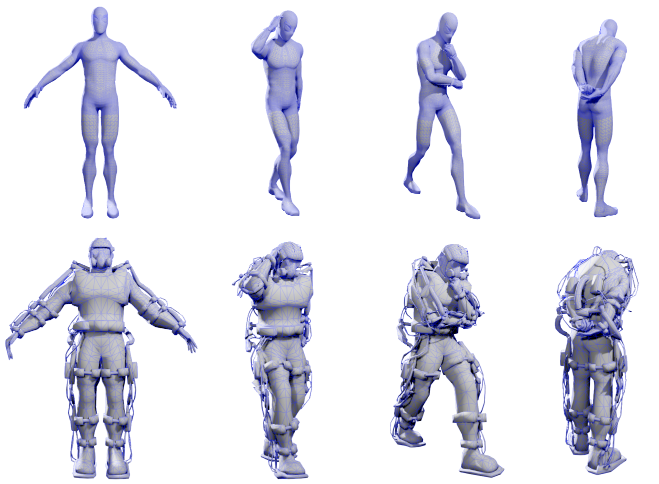

Fig. 10. Two humanoid characters with predicted skinning weights in rest pose (left). The right three columns show the deformation results. The character in the bottom row has a very different geometry style from the training set. Top row: original model courtesy of Pawas Saxena.

Table 7. Architecture of the fully-connected network used in our comparison.

Operation	BatchNorm	Activation	Dropout
FC(1024)	Yes	LeakyReLU(0.1)	No
FC(512)	Yes	LeakyReLU(0.1)	Yes(0.6)
FC(512)	Yes	LeakyReLU(0.1)	No
FC(256)	No	LeakyReLU(0.1)	No
FC(240)	No	No	No

Table 8. Evaluation of the predicted weight maps on Game A using different network structures.

Character	precision	recall	mean	max
			error	error
Fully-Connected	0.483	0.851	0.131	1.86
PoinNet ([Qi et al. 2017])	0.659	0.902	0.182	**1.67**
Our network (no pooling)	0.723	0.957	0.117	1.71
Our network	**0.743**	**0.982**	**0.071**	1.83

5.3 Comparisons

We compare our method with the GVB method introduced in [Dionne and de Lasa 2013]). Fig. 11 shows some deformation results using the skin weights computed by the GVB. As shown, the method fails to separate the sleeves and dress automatically, and in many cases, the bindings are too loose to make the character deform correctly (e.g., human body part always undergoes soft deformation). The main reason is the lack of the prior knowledge of the dataset. Our learning-based method makes full use of prior knowledge, and the resulting character can perform well during deformation. Please see the accompanying video for animation comparison.

We also conduct experiments to compare the performance of different network structures used in our method. The first structure only contains fully-connected layers (i.e., without graph convolution), with the detailed architecture listed in Table 7. The second structure is the same as PointNet [Qi et al. 2017]. The third structure is a modification of our proposed network with graph pooling removed. The experiments are conducted on Game A. The quantitative results on the predicted weight maps are listed in Table 8, and we show some deformation results in Fig. 12. As observed, the fully-connected network fails to separate the dress and sleeves (Fig. 12 left). The PointNet can separate individual parts of the character to a certain degree, but is still problematic (Fig. 12 middle left). Our network without graph pooling can already generate better results than PointNet (Fig. 12 middle). With the graph pooling operation integrated into the graph convolution network, the receptive field of the vertices of different layers in the resulting network is controllable, resulting in the best result (Fig. 12 middle right). Table 8 further shows that the precision and recall scores of the bone selection with different network structures.

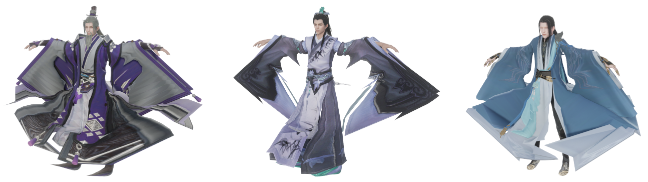

Fig. 11. Deformation results using the skin maps computed by [Dionne and de Lasa 2013].

6 CONCLUSION

We have introduced the first deep-learning-based method for automatic skin binding using graph convolution networks. Our method is designed to work with production meshes which are often composed of multiple, disjoint or intersected components and require complex skeleton hierarchies for animation control. We have demonstrated the efficacy and robustness of our method on datasets of commercial games. The skin weight maps predicted by our method can produce high-quality deformations and are readily applied to characters in game production.

Our work still has some limitations which need to be explored in the future. First, our method requires artists to manually specify the skeleton hierarchy and positions for the input character mesh in rest pose. It would be interesting to investigate methods for inferring the skeleton positions together with the skinning weights in a joint manner, which could further reduce manual labors in the skinning pipeline. Second, the weight maps predicted by our method are not guaranteed to be smooth everywhere, which may lead to uneven deformation in places with complex substructure. Many attempts have been done to generate smooth weight maps, such as [Bang and Lee 2018]. However, most of them are introduced as a post-process in the skinning pipeline. Developing an intuitive end-to-end method to predict smooth weight maps remains an interesting problem. Finally, as a deep-learning-based approach, our method relies on a large set of training samples, and its generalization ability to drastically different skeleton hierarchies is weak. We would like to explore

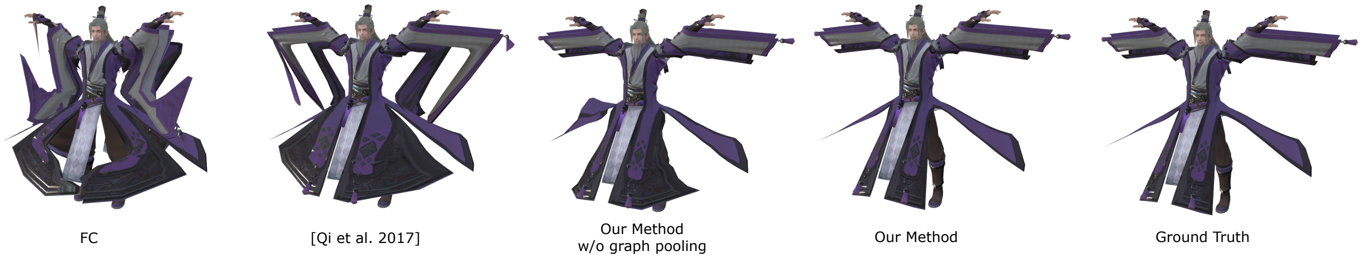

Fig. 12. Comparisons between different network structures. From left to right: fully-connected, PointNet, our network (no pooling), our network, ground truth.

different representations of the mesh-skeleton attributes as well as network architectures to achieve better generalization ability.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their constructive suggestions, Jie Deng and Zhipeng Hu for helpful discussions, and Pawas Saxena for letting us use their work under the Creative Commons License. This work is partially supported by the National Key Research & Development Program of China (2018YFE0100900), NSF China (No. U1609215) and the Fundamental Research Funds for the Central Universities.

REFERENCES

Stephen W Bailey, Dave Otte, Paul Dilorenzo, and James F O'Brien. 2018. Fast and deep deformation approximations. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 119:1–119:12.

Seungbae Bang, Byungkuk Choi, Roger Blanco i Ribera, Meekyoung Kim, Sung-Hee Lee, and Junyong Noh. 2015. Interactive rigging with intuitive tools. In *Computer Graphics Forum*, Vol. 34. 123–132.

Seungbae Bang and Sung-Hee Lee. 2018. Spline Interface for Intuitive Skinning Weight Editing. *ACM Transactions on Graphics (TOG)* 37, 5 (2018), 174:1–174:14.

Ilya Baran and Jovan Popović. 2007. Automatic rigging and animation of 3d characters. *ACM Transactions on graphics (TOG)* 26, 3 (2007), 72:1–72:8.

Davide Boscaini, Jonathan Masci, Emanuele Rodolà, and Michael Bronstein. 2016. Learning shape correspondence with anisotropic convolutional neural networks. In *Advances in Neural Information Processing Systems*. 3189–3197.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral Networks and Locally Connected Networks on Graphs. *CoRR* abs/1312.6203 (2013).

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*. 3844–3852.

Olivier Dionne and Martin de Lasa. 2013. Geodesic voxel binding for production character meshes. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 173–180.

Olivier Dionne and Martin de Lasa. 2014. Geodesic Binding for Degenerate Character Geometry Using Sparse Voxelization. *IEEE Transactions on Visualization and Computer Graphics* 20 (2014), 1367–1378.

Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*. 1024–1034.

Mikael Henaff, Joan Bruna, and Yann LeCun. 2015. Deep Convolutional Networks on Graph-Structured Data. *CoRR* abs/1506.05163 (2015).

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. 2017. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Alec Jacobson, Ilya Baran, Jovan Popovic, and Olga Sorkine. 2011. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph.* 30, 4 (2011), 78–1.

Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. 2007. Skinning with dual quaternions. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*. 39–46.

Ladislav Kavan and Olga Sorkine. 2012. Elasticity-inspired deformers for character articulation. *ACM Transactions on Graphics (TOG)* 31, 6 (2012), 196:1–196:8.

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*. 9–16.

Meekyoung Kim, Gerard Pons-Moll, Sergi Pujades, Seungbae Bang, Jinwook Kim, Michael J Black, and Sung-Hee Lee. 2017. Data-driven physics for human soft tissue animation. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 54:1–54:12.

Thomas N. Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. *CoRR* abs/1609.02907 (2016).

Binh Huy Le and Zhigang Deng. 2014. Robust and accurate skeletal rigging from mesh sequences. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 84:1–84:10.

Binh Huy Le and Jessica K Hodgins. 2016. Real-time skeletal skinning with optimized centers of rotation. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 37.

Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J Black. 2015. SMPL: A skinned multi-person linear model. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 248:1–248:16.

Ran Luo, Tianjia Shao, Huamin Wang, Weiwei Xu, Kun Zhou, and Yin Yang. 2018. NNWarp: Neural Network-based Nonlinear Deformation. *IEEE Transactions on Visualization and Computer Graphics* 24, 11 (2018).

Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. 2015. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 37–45.

Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. 2017. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Vol. 1. 3.

Tomohiko Mukai and Shigeru Kuriyama. 2016. Efficient dynamic skinning with low-rank helper bone controllers. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 36:1–36:11.

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Vol. 1. 77–85.

Yi-Ling Qiao, Lin Gao, Yu-Kun Lai, and Shihong Xia. 2018. Learning Bidirectional LSTM Networks for Synthesizing 3D Mesh Animation Sequences. *CoRR* abs/1810.02042 (2018).

Weiguang Si, Sung-Hee Lee, Eftychios Sifakis, and Demetri Terzopoulos. 2014. Realistic biomechanical simulation and control of human swimming. *ACM Transactions on Graphics (TOG)* 34, 1 (2014), 10:1–10:15.

Qingyang Tan, Lin Gao, Yu-Kun Lai, Jie Yang, and Shihong Xia. 2018. Mesh-Based Autoencoders for Localized Deformation Component Analysis. In *AAAI*.

Kiran Koshy Thekumparampil, Chong Wang, Sewoong Oh, and Li-Jia Li. 2018. Attention-based Graph Neural Network for Semi-supervised Learning. *CoRR* abs/1803.03735 (2018).

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*. 5998–6008.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. *International Conference on Learning Representations* (2018).

Rich Wareham and Joan Lasenby. 2008. Bone glow: An improved method for the assignment of weights for mesh deformation. In *International Conference on Articulated Motion and Deformable Objects*. 63–71.

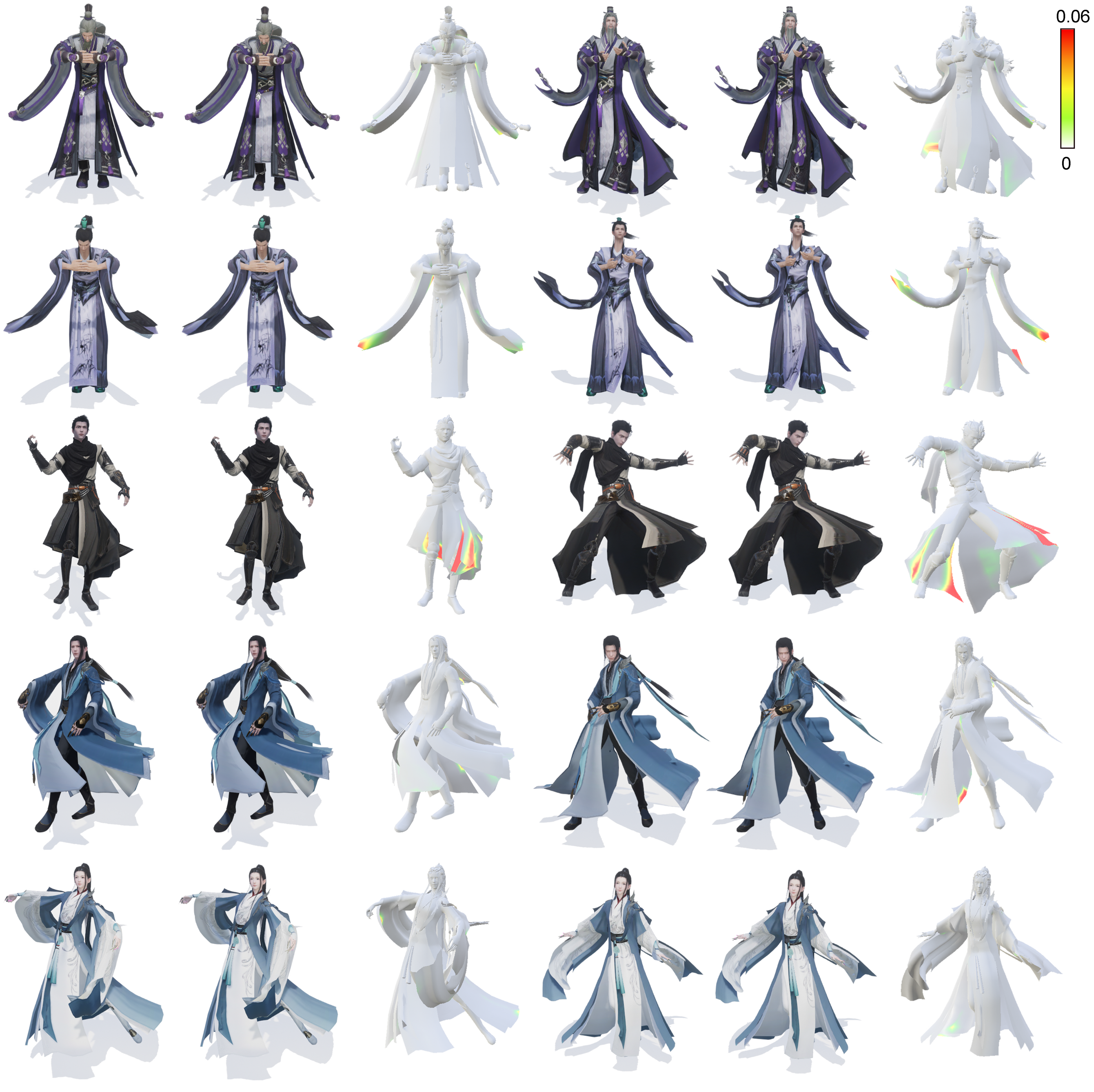

Fig. 13. **Deformations of five characters in Game A.** Side-by-Side comparisons of the ground truth deformation and our predicted deformation in extreme poses. Each row shows a character. The first and fourth columns are the ground truth deformation. The second and fifth columns are the predicted deformation. The third and sixth columns are the visualization of the per-vertex prediction errors.

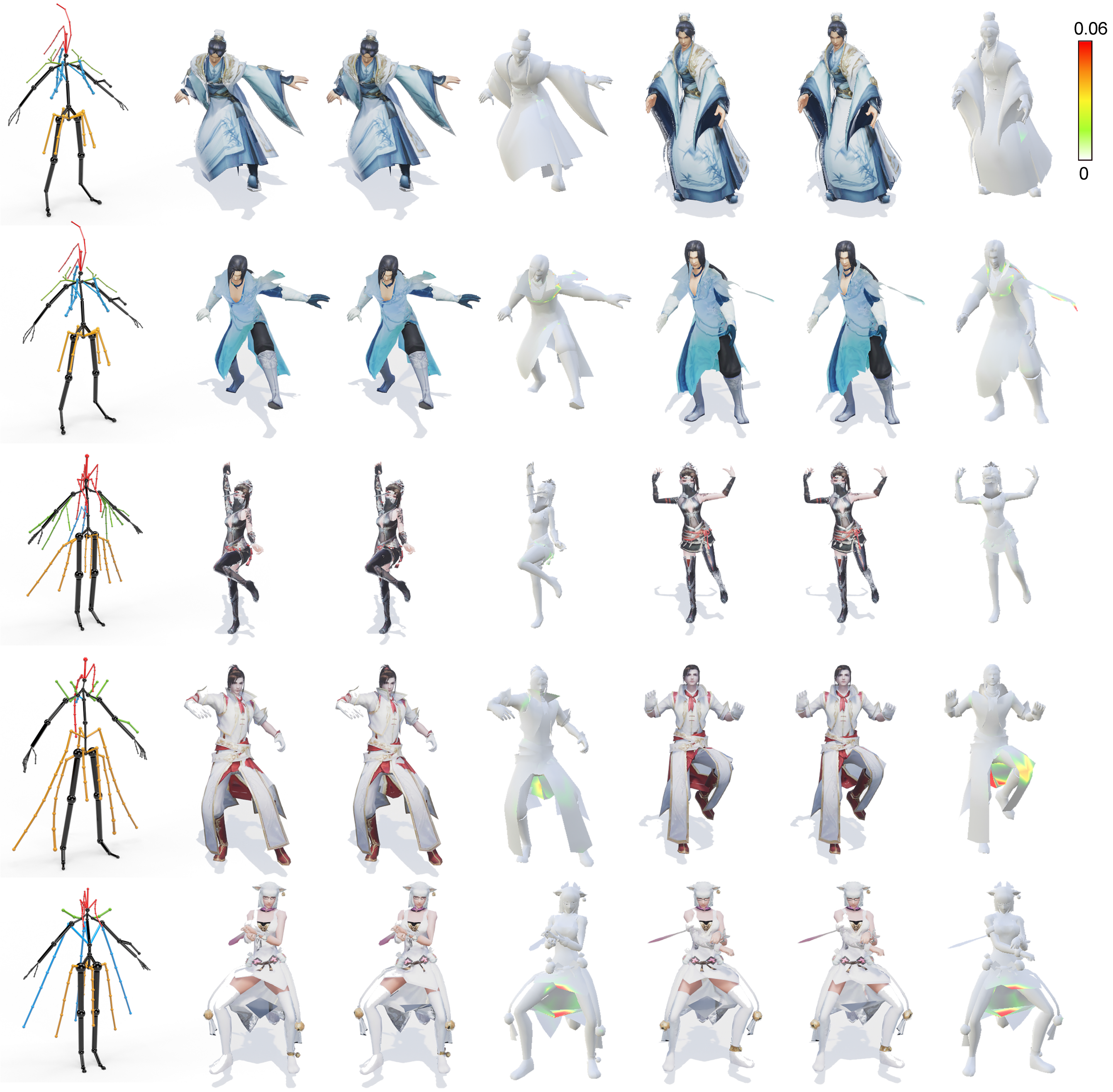

Fig. 14. **Deformations of five characters in Game B.** Side-by-Side comparisons of the ground truth deformation and our predicted deformation in extreme poses. Each row shows a character. The first column is the skeleton hierarchy. The second and fifth columns are the ground truth deformation. The third and sixth columns are the predicted deformation. The fourth and the seventh columns are the visualization of the per-vertex prediction errors.