HeterSkinNet: A Heterogeneous Network for Skin Weights Prediction

XIAOYU PAN, State Key Lab of CAD&CG, Zhejiang University; ZJU-Tencent Game and Intelligent Graphics Innovation Technology Joint Lab, China

JIANCONG HUANG, Tencent Games Lightspeed & Quantum Studios, China

JIAMING MAI, State Key Lab of CAD&CG, Zhejiang University; ZJU-Tencent Game and Intelligent Graphics Innovation Technology Joint Lab, China

HE WANG, School of Computing, University of Leeds, United Kindom

HONGLIN LI, Quanzhou Medical College, China

TONGKUI SU, Tencent Games Lightspeed & Quantum Studios, China

WENJUN WANG, Tencent Institute of Games, China

XIAOGANG JIN*, Corresponding author, State Key Lab of CAD&CG, Zhejiang University; ZJU-Tencent Game and Intelligent Graphics Innovation Technology Joint Lab, China
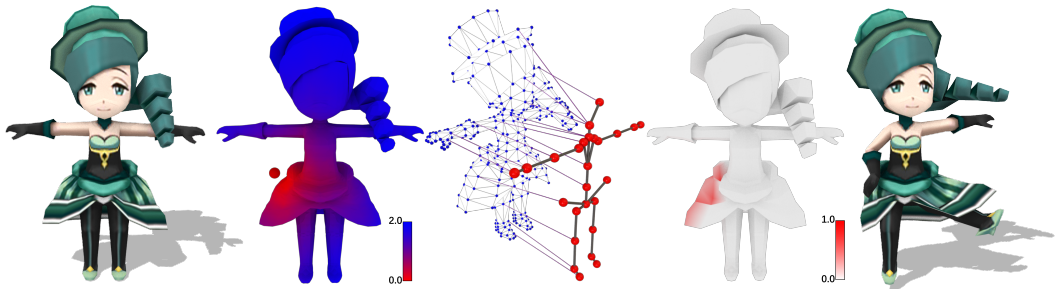
Fig. 1. Given a character mesh and skeleton, HeterSkinNet builds a heterogeneous graph network to estimate skin weights. From left to right: example character model, per-vertex HollowDist to the dress bone (red sphere), the heterogeneous graph our network operates on, predicted skin weights of the dress bone, a pose with our estimated skin weights.

Authors' addresses: Xiaoyu Pan, State Key Lab of CAD&CG, Zhejiang University; ZJU-Tencent Game and Intelligent Graphics Innovation Technology Joint Lab, Hangzhou, China, panxiaoyu6@gmail.com; Jiancong Huang, Tencent Games Lightspeed & Quantum Studios, Shenzhen, China, bbbhuang@tencent.com; Jiaming Mai, State Key Lab of CAD&CG, Zhejiang University; ZJU-Tencent Game and Intelligent Graphics Innovation Technology Joint Lab, Hangzhou, China, maijmwq@126.com; He Wang, School of Computing, University of Leeds, Leeds, United Kindom, H.E.Wang@leeds.ac.uk; Honglin Li, Quanzhou Medical College, Quanzhou, China, lihonglin79@qq.com; Tongkui Su, Tencent Games Lightspeed & Quantum Studios, Shenzhen, China, tongkuisu@tencent.com; Wenjun Wang, Tencent Institute of Games, Shenzhen, China, jamesonwang@tencent.com; Xiaogang Jin*, Corresponding author, State Key Lab of CAD&CG, Zhejiang University; ZJU-Tencent Game and Intelligent Graphics Innovation Technology Joint Lab, HangZhou, China, jin@cad.zju.edu.cn.

Character rigging is universally needed in computer graphics but notoriously laborious. We present a new method, HeterSkinNet, aiming to fully automate such processes and significantly boost productivity. Given a character mesh and skeleton as input, our method builds a heterogeneous graph that treats the mesh vertices and the skeletal bones as nodes of different types and uses graph convolutions to learn their relationships. To tackle the graph heterogeneity, we propose a new graph network convolution operator that transfers information between heterogeneous nodes. The convolution is based on a new distance *HollowDist* that quantifies the relations between mesh vertices and bones. We show that HeterSkinNet is robust for production characters by providing the ability to incorporate meshes and skeletons with arbitrary topologies and morphologies (e.g., out-of-body bones, disconnected mesh components, etc.). Through exhaustive comparisons, we show that HeterSkinNet outperforms state-of-the-art methods by large margins in terms of rigging accuracy and naturalness. HeterSkinNet provides a solution for effective and robust character rigging.

1 INTRODUCTION

There is a surging growth of need for high-quality character animation in film and video game productions. To animate a character mesh, rigging is a popular solution that is widely adopted in industry, where animators first create a skeleton hierarchy and then bind the mesh to the skeleton with the mesh vertices associated with the bones under some weighting schemes, i.e., skin weights. Although methods have been developed to automatically compute skin weights, they are still often painted by hand to enable fine control of the mesh deformation caused by bone transformations. Consequently, when the models and motions become complex, this step becomes time-consuming even for skilled animators.

Existing efforts have been made to automatically compute the skin weights by hand-crafted functions [Baran and Popović 2007; Jacobson et al. 2011; Wareham and Lasenby 2008] and learned functions [Liu et al. 2019; Xu et al. 2020] of vertex features with respect to bones. However, different bones influence areas of various sizes to different extents. While the spine of a character can influence vertices on the torso which are far from it, finger bones only influence close-by vertices. Further, the bone-influence relation changes with the skeleton morphology of characters, shown by Figure 2 which contains characters of different species with both in-body and out-of-body bones. In extreme cases, the skeleton structure might be totally arbitrary, e.g., out-of-body bones used to deform cloths. The majority of existing methods often simplify the heterogeneous relations between mesh vertices and bones, by homogeneously considering them in one model (e.g., one function or network) to capture different bones influencing different areas, downplaying the importance of the intrinsic shape features of these areas and the connections of the bones. A few attempts [Liu et al. 2019] have been made to apply different functions to different bones by semantically labeling the bones. However, they are designed for one type of skeletal morphology and therefore unable to generalize to characters with different skeletal morphologies.

In this paper, we present a deep-learning-based method, HeterSkinNet, to automatically estimate skin weights of character models with diverse meshes and arbitrary skeletal morphologies. Our network extracts both vertex features and bone features and estimates skin weights according to their relations. It starts by recognizing the different roles of the mesh and the skeleton in rigging, representing them in two graphs, Mesh Graph and Skeleton Graph respectively. Graph convolution operations are used to extract node features. To model the interaction between the mesh and
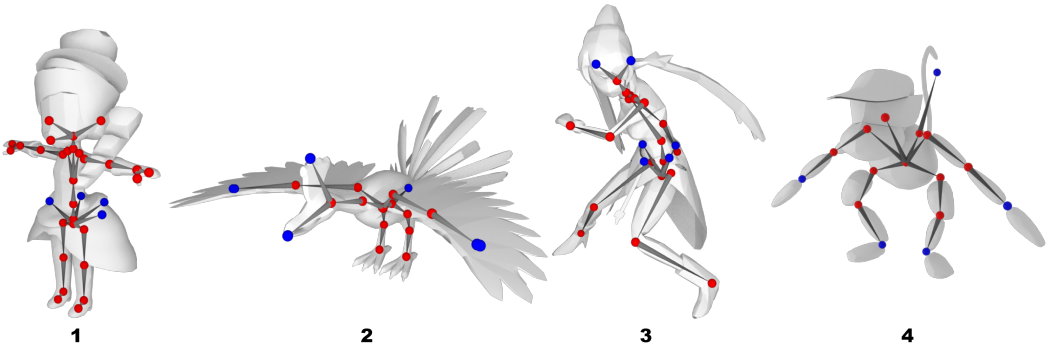
Fig. 2. Example characters with their skeletons. The out-of-body bones controlling thin features are colored blue. Most of them are ending bones, which start and end at the same leaf joints.

the skeleton, we propose a new intrinsic distance named "HollowDist", which is robust for ill-conditioned situations in production models. Finally, to enable information passing between the two graphs, we propose a new convolution operator, which convolutes between the two kinds of nodes and builds interactions between them. We evaluate HeterSkinNet on a large dataset with a wide variety of meshes and skeletons. The qualitative and quantitative results show that our method can handle the mesh and skeleton diversity well, even for out-of-body bones and disconnected mesh components. We also extensively compare our method with state-of-the-art methods where the quantitative results are improved by large margins.

Formally, our technical contributions include:

- The first heterogeneous graph neural network that estimates skin weights for character models with diverse skeletal morphologies and body meshes.
- A new graph convolution operator that handles heterogeneous nodes in a graph.
- A new distance to quantify vertex-bone relations which can handle ill-conditioned situations including out-of-body bones and disconnected meshes.

2 RELATED WORK

Skin Deformations. Various skinning techniques have been proposed for character rigging. Among them, Linear Blend Skinning (LBS) [Alexa 2002] and Dual Quaternion Skinning (DQS) [Kavan et al. 2007] are widely used in real-time applications due to their simplicity and computational efficiency. In these methods, vertices on the mesh are deformed by weighted transformations of bones. The weights associating vertices with bones determine the quality of deformation. Several attempts have been proposed to automatically estimate high-quality skin weights, which can be categorized into geometry-based and data-driven methods. Geometry-based methods estimate skin weights based on hand-crafted functions of vertex features with respect to bones, such as methods utilizing heat diffusion [Baran and Popović 2007], illumination model [Wareham and Lasenby 2008], Laplacian energy [Jacobson et al. 2011] and volumetric geodesic distance [Dionne and de Lasa 2013; Dionne and De Lasa 2014]. The hand-crafted functions deployed in these methods make assumptions for the weight distributions. Therefore, it is hard for these methods to capture the anatomic information intrinsic in the input meshes, such as skin flexibility and spine disparity.

Data-driven methods can successfully capture anatomic information of meshes by learning from a set of character models with artist-painted skin weights. NeuroSkinning [Liu et al. 2019] is the first work predicting skin weights of production characters with complex garments. Their network

performs graph convolution on graphs represented by meshes to extract high-level features of vertices and predicts the vertex weights as outputs. However, their network assumes that the bones should be labeled semantically as human bones or cloth bones, which restricts its generalization to models with different skeletal morphologies. RigNet [Xu et al. 2020] provides an end-to-end method of rigging models of arbitrary categories, which predicts the model skeleton and corresponding skin weights jointly. To this end, in the skinning stage of their network, they take intrinsic vertex-bone distances as inputs, which catch the implicit mesh shape information. However, their method does not differentiate the skin weights functions applied for different bones. Different from the two methods mentioned above, our method can be applied to models with arbitrary skeletal morphologies while considering the different situations of bones. Our approach also takes intrinsic vertex-bone distances as network inputs and estimates skin weights using vertex features and bone features extracted by our heterogeneous network.

Graph Neural Networks. Graph Neural Networks (GNNs) have shown superior power on deep learning tasks processing data represented by graphs [Bronstein et al. 2017; Wu et al. 2020; Zhang et al. 2020; Zhou et al. 2018]. There is an increasing interest in applying GNNs on geometric data of different representations [Xiao et al. 2020], especially for meshes. These GNNs can be generally categorized into spectral approaches [Boscaini et al. 2016; Monti et al. 2017; Yi et al. 2017] and spatial approaches [Hanocka et al. 2019; Liu et al. 2019; Masci et al. 2015; Verma et al. 2018; Xu et al. 2020]. Spectral approaches operate on a graph's spectral domain by performing eigen decomposition of the graph Laplacian. They fail to generalize to meshes with different topologies, which have different graph Laplacians. Spatial methods operate on spatially neighboring graph nodes, so that they can generalize to unseen graphs. Our network performs convolution on spatially connected nodes and thus can process character models with different meshes and skeletal morphologies.

In contrast to the above networks operating on homogeneous graphs, other networks focus on handling heterogeneous graphs. Wang et al. [Wang et al. 2019a] proposed a graph neural network based on a hierarchical attention mechanism for general graph analysis tasks. Similarly, Hu et al. [Linmei et al. 2019] deployed a dual-level attention mechanism on heterogeneous graphs for short text classification. However, our method applies a heterogeneous graph neural network on a different task using different convolution operations, which builds heterogeneous graphs constituted of vertex and bone nodes and learns their high-level features by transferring features with their homogeneous neighbors and heterogeneous neighbors alternatively.

Deep-Learning-based Deformation. Several deep learning methods have been proposed for mesh deformation. Some methods process mesh animation sequences. Tan et al. proposed Mesh VAE [Tan et al. 2018] to extract spatially localized deformation components of mesh deformation, which can be used for shape analysis and synthesis tasks. Qiao et al. introduced [Qiao et al. 2018] a long-short-term-memory (LSTM) based network to synthesize new mesh animation sequences based on the input sequence.

Some researchers adopt neural networks as non-linear modules to enhance the linear deformations of meshes. Luo et al. [Luo et al. 2020] proposed a network mapping linear displacements to corresponding nonlinear displacements for simulation of elastic materials. Similarly, Bailey et al. [Bailey et al. 2018] used a fully-connected neural network to approximate the nonlinear deformations of a specific character. Li et al. leveraged a densely connected graph attention network [Li et al. 2020] for generalization, which can be adopted to human character models with unseen meshes and the same skeleton as those in the training set. Our network estimates the skin weights as the linear portion of deformations, which can be complementary to the above methods.
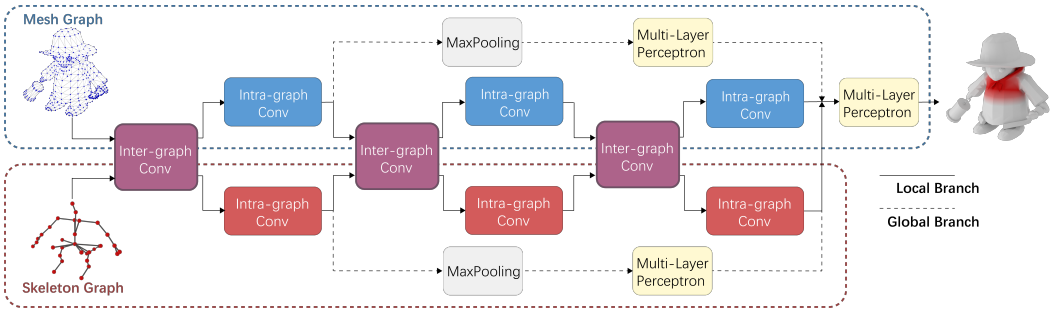
Fig. 3. The structure of our network. The network operates on a heterogeneous graph composed of Mesh Graph and Skeleton Graph. Intra-graph convolution components operate within the two subgraphs, and inter-graph convolution components transfer features between them. The detail of inter-graph convolution components is shown in Figure 5.

3 METHODOLOGY

Given a character mesh and skeleton, our method aims to obtain the skin weights associating the mesh vertices with the skeleton, which represent the influences of bone transformations on the vertices. First, we present a new distance named 'HollowDist', which quantifies vertex-bone relations. Its calculation is based on surface voxelization and the breadth-first search algorithm. Next, we construct the graph that our network operates on. The graph is constituted of two subgraphs: Mesh Graph, whose nodes and edges represent vertices and mesh edges respectively, and Skeleton Graph, whose nodes and edges indicate bones and joints respectively. The connections between nodes on the two subgraphs are built by HollowDist between corresponding vertices and bones.

Our network is shown in Figure 3 and is composed of intra-graph and inter-graph convolution components: the former operate within subgraphs, i.e., Mesh Graph and Skeleton Graph, and the latter transfer information between the two subgraphs. The network first transfers the node attributes between the two subgraphs via an inter-graph convolution component and then processes the transferred features separately within the two subgraphs by intra-graph components. In each subgraph, the learned features are fed into two branches: a global branch and a local branch. The global branch of each subgraph consists of a max-pooling layer and an MLP layer, which extracts the overall feature of the corresponding subgraph. The local branch is composed of alternatively stacked intra-graph and inter-graph components, which extracts the features of each vertex and bone nodes. At the end of our network, the bone nodes' features are scattered to their neighboring vertex nodes with the concatenation of the two subgraphs' global features, and the resulting features are sent into a stack of MLPs to get the skin weights. The loss function of our network not only minimizes the difference between the predicted skin weights and the ground truth, but also encourages the skin weights to distribute smoothly on the mesh.

In the following subsections, we will first describe the calculation of HollowDist, and then we will introduce the construction of our graph, the individual components of our network and the loss function.

3.1 HollowDist

To model vertex-bone relationships, NeuroSkinning [Liu et al. 2019] uses the Euclidean distance which ignores the shape of the mesh and therefore can represent paths that may penetrate the mesh. Thus a restriction needs to be imposed which dictates that the path cannot penetrate the mesh, resulting in a restricted distance. Several distances [Rustamov et al. 2009; Solomon et al. 2014]

have been proposed in calculating such distances for manifold meshes. But they cannot handle meshes containing non-manifold geometries and intersected triangles. To tackle these meshes, we adopt the idea introduced in [Dionne and de Lasa 2013], which voxelizes the mesh and calculate the distances between bones and vertices via paths inside the mesh. However, their method cannot handle out-of-body bones and meshes with disconnected parts. Different from their method, our approach does not assume that the bones are inside the mesh. We divide the space into mesh cells, bone cells and hollow cells and find paths across hollow cells, which can be either inside the mesh or outside the mesh. Moreover, to tackle disconnected mesh parts, our method restarts the search for unreachable mesh cells.

We first identify the mesh cells by voxelizing the mesh via the GPU-based surface voxelization method proposed in [Schwarz and Seidel 2010], which fills cells intersecting with mesh triangles and keeps the remaining cells hollow. Compared with other rasterization-based methods [Eisemann and Décoret 2008], this approach does not miss important thin structures in production models, such as garments and hair. Then, we find the path from bone cells to mesh cells via the breadth-first search algorithm, which iteratively traverses from seed cells to their neighboring untraversed cells and marks them as new seeds for the next iteration, extending the path to the new seed cells. Our algorithm is illustrated in Algorithm 1 and is visualized in Figure 4. We denote cs as cells, with side length of the cell s_c. For each bone b_i, we identify the bone cells as cells intersecting with it. The distances of bone cells are set to zero,

Fig. 4. Calculation of the HollowDist. (a) A Path from a bone cell to a mesh cell. The cells are colored to indicate their HollowDist to the bone, in which bone cells are colored white. Mesh surfaces are represented by blue lines, and cells intersecting with it are mesh cells, which have bold boundaries. (b) Restart the search for unconnected mesh cells.

and those of the remaining cells are set to infinity (lines 2-9). The path starts from the bone cells and will not penetrate the mesh boundaries which are represented by mesh cells (lines 11-19). For mesh cells that cannot be reached, we find the nearest mesh cell neighboring to any untraversed hollow cell and restart the search from one of its neighboring untraversed hollow cells (lines 20-24). The length of such a path is named HollowDist, as it is mostly on the hollow cells.

The HollowDist for each vertex is calculated based on the length of the path found above. For vertex v_i in cell c, its HollowDist to bone b_j is computed as follows:

$$d_{v_i}^{b_j} = d_{c_{prev}}^{b_j} + ||p_{c_{prev}} - p_{v_i}||_2, \tag{1}$$

where c_{prev} is the cell neighboring to c, which is the previous cell on the path, and $p_{c_{prev}}$ is its center position.
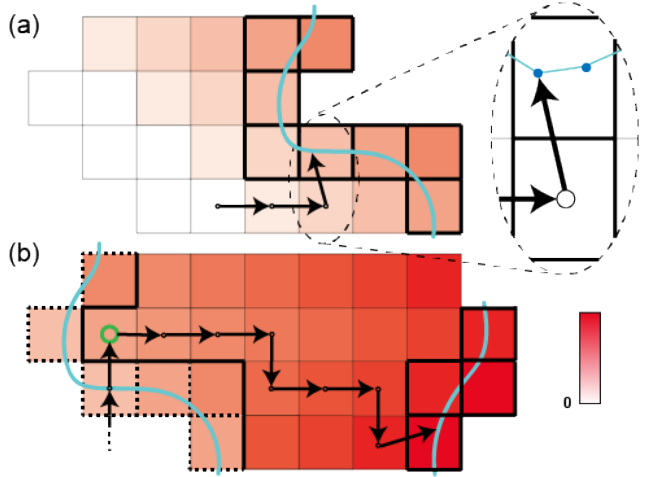

Algorithm 1: HollowDist Computation

   Input: Cell grid $\mathbb{G}$, side length of cell s_c and Character skeleton $\mathcal{S}$

   Output: Distance between each bone and mesh cells **d**

1 **foreach** *bone b_i in $\mathcal{S}$* **do**

     `// Initialize cell distances`

2   **foreach** *cell c_i of $\mathbb{G}$* **do**

3        $d_{c_i}^{b_i} = \infty$;

4   **end**

5   Create empty cell queue Q;

6   **foreach** *cell c_i intersecting with bone b_i* **do**

7        $d_{c_i}^{b_i} = 0$;

8   Push c_i to Q;

9   **end**

     `// Compute distance`

10   **while** *exists untraversed mesh cell* **do**

11   **while** *Q is not empty* **do**

12   Pop c_i from Q;

13   **foreach** *untraversed cell c_j neighboring to c_i* **do**

14   **if** *not (c_i is mesh cell and c_j is not a mesh cell)* **then**

15            $d_{c_j}^{b_i} = d_{c_i}^{b_i} + s_c$;

16   Push c_j to Q;

17   **end**

18   **end**

19   **end**

      `// Restart the search`

20   Find the nearest mesh cell c_i neighboring to any untraversed hollow cell;

21   **foreach** *Untraversed hollow cell c_j neighboring to c_i* **do**

22        $d_{c_j}^{b_i} = d_{c_i}^{b_i} + s_c$;

23   Push c_j to Q;

24   **end**

25   **end**

26 **end**

3.2 Graph Construction

Given a mesh $\mathcal{M}$ with N vertices and its associated skeleton $\mathcal{S}$ with B bones, we construct a heterogeneous graph $\mathcal{G} = (\mathcal{G}_m, \mathcal{G}_s, \mathcal{A}_{ms})$, which is comprised of Mesh Graph $\mathcal{G}_m$, Skeleton Graph $\mathcal{G}_s$ and the adjacency matrix $\mathcal{A}_{ms}$ representing edges between the two subgraphs. The Mesh Graph is denoted as $\mathcal{G}_m = (\mathcal{V}_m, \mathcal{E}_m)$, where $\mathcal{V}_m$ is the set of graph nodes indicating vertices on the mesh, and $\mathcal{E}_m \in \mathcal{V} \times \mathcal{V}$ represents its edges. We also denote v_i as the i-th node in $\mathcal{V}_m$. To construct the Skeleton Graph $\mathcal{G}_s = (\mathcal{V}_s, \mathcal{E}_s)$, we first add zero-length helper bones at leaf joints for skinning at the end of limbs, then treat each bone as a node and each joint as an edge, and thus $\mathcal{V}_s$ and $\mathcal{E}_s$ have similar meanings as those in the Mesh Graph $\mathcal{G}_m$. $\mathcal{A}_{ms} \in [0, 1]^{N \times B}$ is the connectivity matrix between the Mesh Graph and the Skeleton Graph, $\mathcal{A}_{ms}[i, j] = 1$ if vertex v_i is influenced by bone b_j. We assume a vertex is influenced by its nearby bones and thus set $\mathcal{A}_{ms}[i, j] = 1$ if bone b_j is in

the Top-K closest bones to vertex v_i according to HollowDist. K is the max number of influential bones to a vertex, which is set manually.

The inputs of our network are attributes of nodes. The attribute of a vertex node also captures its relationship with the bones. For vertex v_i, its calculated HollowDist to all bones are denoted as $\{D_{i,j}\}_{j=1,...,B}$, where $D_{i,j}$ is the HollowDist from v_i to the bone b_j. We select the Top-K nearest bones in the ascending order: $\{n_{i,j}\}_{j=1,...,K}$, in which $n_{i,j}$ indicates the index of the j-th nearest bone to vertex v_i. The vertice's attribute is the concatenation of its position and the inverse of HollowDists to the Top-K closest bones: $f_{v_i} = [p_{v_i}^T, 1/D_{i,n_{i,1}}, ..., 1/D_{i,n_{i,K}}] \in R^{K+3}$. For bone nodes, we use their starting and ending joint positions as attributes: $f_{b_j} = [p_{b_{j,start}}^T, p_{b_{j,end}}^T] \in R^6$.

3.3 Individual Components

Intra-graph Convolution Components. The intra-graph convolution components are built based on EdgeConv [Wang et al. 2019b], which aggregates nodes' local features of their homogeneous neighbors. The operations on the Mesh Graph $\mathcal{G}_m$ and the Skeleton Graph $\mathcal{G}_s$ are slightly different. For the Skeleton Graph, we simply use EdgeConv, which is shown as follows:

$$
\begin{aligned}
f_{b_i}' &= EdgeConv(b_i, \mathcal{N}(b_i)) \\
&= \max_{b_j \in \mathcal{N}(b_i)} MLP(f_{b_i} || (f_{b_i} - f_{b_j}); W_s).
\end{aligned}
\tag{2}
$$

Here, $||$ is the concatenation operator. $f_{b_i} \in R^{F_b}$ and $f_{b_i}' \in R^{F_b'}$ are the input feature and the output feature of b_i, whose dimensions are F_b and F_b', respectively. $\mathcal{N}(b_i)$ is the set of bones connecting to bone b_i, and $W_s \in R^{2F_b \times F_b'}$ is the trainable weight matrix of the component.

When applying convolution operations on the mesh graph, a node's receptive field is determined by the distances to its one-ring neighbors, which are influenced by the mesh tessellation. To make the network less sensitive to the mesh tessellation, we adopt the operation introduced in [Xu et al. 2020], which separately conducts EdgeConv within nodes' geodesic neighbors and one-ring neighbors and gets the result by processing the concatenation of the results through an MLP. The geodesic neighbors of a vertex are vertices whose geodesic distances from it are within a threshold (0.06). The operation is shown as follows:

$$
f_{v_i}' = MLP(EdgeConv(v_i, \mathcal{N}_m(v_i)) || EdgeConv(v_i, \mathcal{N}_g(v_i)); W_v),
\tag{3}
$$

where $\mathcal{N}_m(v_i)$ and $\mathcal{N}_g(v_i)$ are the one-ring neighbors and geodesic neighbors of the vertex node v_i, $W_v \in R^{2F_v' \times F_v'}$ is the trainable weight matrix of the component, and F_v and F_v' denote the dimensions of input features and output features, respectively.

Inter-graph Convolution Component. Within this component, the vertex and bone nodes aggregate features from each other mutually. Due to the unbalanced property of $\mathcal{G}$'s two subgraphs, whose numbers of nodes differ greatly, i.e., $\mathcal{G}_m$ contains thousands of vertex nodes while $\mathcal{G}_s$ have dozens of bone nodes, we design the vertex-to-bone and bone-to-vertex convolution operations differently.

For the vertex-to-bone convolution, the bone nodes gather information from their influenced vertex nodes. A lossless way to gather features is to directly concatenate their features. However, the numbers of vertices influenced by bones vary greatly, which makes the dimensions of resulting features indeterminate and thus is unfeasible in a neural network. Inspired by [Wen et al. 2019], we instead concatenate the maximum, mean and variance pooled from all their influenced vertex nodes' features to make the resulting features' dimensions fixed. This convolution operation also
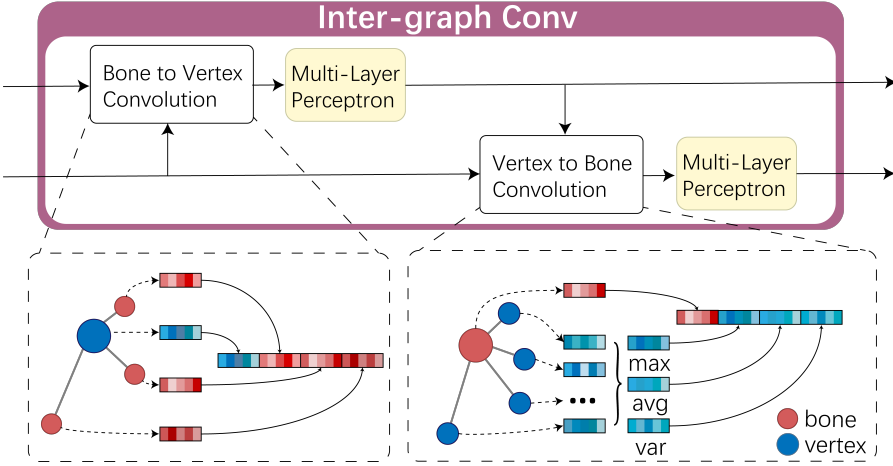
Fig. 5. Illustration of the inter-graph convolution operator. Within this operator, the bone-to-vertex convolution and the vertex-to-bone convolution are performed sequentially. In the bone-to-vertex convolution, the vertex nodes in the Mesh Graph concatenate their features with the features of their Top-K nearest bone nodes. In the vertex-to-bone convolution, the bone nodes in the Skeleton Graph concatenate their features with the maximum, mean and variance of their influenced vertex nodes' features. Each convolution is followed by an MLP layer for the convolution operator's nonlinearity.

encourages the network to learn information from the correlations among these vertex nodes' features besides learn from the nodes' individual features.

For the bone-to-vertex convolution, the vertex nodes gather features from their corresponding bone nodes. Different from the vertex-to-bone convolution, a vertex is set to be influenced by its nearest K bones, directly concatenating their features can result in fixed-sized features. For a vertex node v_i, we first sort the bone nodes according to their HollowDists to it and then concatenate their features with the vertex node's feature and process the concatenated feature via an MLP.

The aforementioned two convolution operations are shown as follows:

$$f'_{b_j} = MLP((f_{b_j} || \max_{v_i \in \mathcal{I}(b_j)} (f_{v_i}) || \operatorname*{mean}_{v_i \in \mathcal{I}(b_j)} (f_{v_i}) || \operatorname*{var}_{v_i \in \mathcal{I}(b_j)} (f_{v_i})); W_{v2b}), \qquad (4)$$

$$f'_{v_i} = MLP(f_{v_i} || f_{b_{n_{i,1}}} || ... || f_{b_{n_{i,K}}}; W_{b2v}). \qquad (5)$$

Here, $\mathcal{I}(b_j) = \{v_i | \mathcal{A}_{ms}[i, j] = 1\}$ represents the set of vertices that bone b_j influences and $W_{v2b} \in R^{F_b \times (F_b + 3F_v)}$ and $W_{v2b} \in R^{F_v \times (F_v + KF_b)}$ are learnable weight matrices to make the feature dimensions unchanged after operations.

3.4 Loss Function

To make the animation satisfactory, the skin weights should satisfy several constraints. Primarily, they are required to be convex, i.e., $w_{ij} \geq 0$ and $\sum_j w_{ij} = 1$, where w_{ij} indicates the skin weight of v_i regarding to bone b_j. Rather than directly imposing the constraints on the resulting weight matrix, we add a softmax layer at the end of our network to ensure its non-negativity and affinity. Apart from being convex, the weight matrix is required to be sparse to ensure computation efficiency. In our network, this constraint is automatically satisfied by only predicting the weights regarding to the nearest K bones for each vertex.

Our loss function is shown as follows:

$$\mathcal{L} = \mathcal{L}_D + \lambda_S \mathcal{L}_S, \tag{6}$$

where $\mathcal{L}_D$ is the data fitting term, $\mathcal{L}_S$ is the smoothing term, and λ_S is a predefined smooth factor.

The data fitting term $\mathcal{L}_D$ encourages the predicted weights to be close to the ground truth. We adopt the strategy in [Liu et al. 2019], which treats the skin weights as label distributions. For a vertex v_i, its predicted weight w_{ij} can be considered as its possibility for selecting bone b_j. Therefore, our data fitting object is minimizing the distance between the predicted weight distribution and the ground truth distribution. To this end, we use the Kullback-Leibler divergence between the two distributions, $\mathcal{L}_D = \sum w_{ij} log(\frac{w_{ij}}{\hat{w}_{ij}})$, as the data fitting term, where $\hat{w}_{ij}$ is the ground truth.

To encourage the skin weights to distribute smoothly over the mesh, we add a smoothing term $\mathcal{L}_S$ to the loss function, which is calculated using the discrete Laplacian matrix of the mesh: $\mathcal{L}_S = \sum_{j=1}^{B} w_j^T L w_j$, in which $L \in R^{N \times N}$ is the mesh Laplacian matrix, and $w_j \in R^N$ is the weight column vector of bone b_j.

4 EXPERIMENTS

4.1 Dataset

We use the "ModelsResource-RigNetv1" dataset from [Xu et al. 2019] for our experiments. This dataset contains manually rigged models with various meshes and different skeletons, covering a wide range of classes, such as humanoids, birds, fish, etc. Most characters' meshes are non-manifold and contain disjoint parts, and some of their bones are placed outside of the meshes. The models are all oriented consistently and rescaled to 1 meter in height. Some models with multiple parts may have duplicate vertices in the same position at the connection of two parts, which have the same skin weights but different connections. To ensure that they deform in the same way, we merge the vertices in the same position when building the mesh graph and assign the same prediction results to the vertices in that position. Moreover, to ensure that the models with different numbers of vertices have similar impacts on our network during training, for models with less than 1K vertices, we subdivide their meshes so that their vertex numbers range from 1K to 5K. All the above pre-processing steps are performed via Houdini [SideFX 2020]. Finally, 3026 models are randomly selected for training, and the remaining 100 models are used for testing. We present 4 character models in our test set in Figure 2, whose statistics are shown in Table 2.

Table 1. The statistics of our example characters (Figure 2)

Character	Vertices	Triangles	Bones	Parts
1	4520	4455	34	7
2	4421	4341	23	15
3	3324	3084	25	13
4	3674	3564	18	17

4.2 Implementation Details

For the calculation of HollowDist, we voxelize all meshes using grid size $88 \times 88 \times 88$. We implement HeterSkinNet using Pytorch Geometric [Fey and Lenssen 2019], and train it on an NVIDIA GeForce RTX 2080Ti GPU, using Adam optimizer [Kingma and Ba 2014] with learning rate $1e-4$ and weight decay $1e-4$. The dimensions of local and global features of vertex nodes are 256 and 512, and those

of bone nodes' features are 128 and 512, respectively. The final layer of the network is a stack of 3 fully connected layers with feature dimensions of $[1024, 512, K]$. In the inter-graph convolution component, the variances of vertex nodes' features gathered by bone nodes are multiplied by 0.1 for training stability. We use Leaky ReLU with $\alpha = 0.2$ as the activation function in the linear layers of our network. Before training, they are initialized using Kaiming normal [He et al. 2015]. During training, we randomly drop out 85% edges in the Mesh Graph and 50% parameters in the final layers to enhance the generalization ability of our network and save GPU memory.

For models in the ModelsResource-RigNetv1 dataset [Xu et al. 2019], we calculate the statistical ratio of vertices' skin weights sum of their Top-K nearest bones to that of all bones (the blue line in Figure 6) as $\sum_{i=1}^{K} w_i / \sum w_i$, where w_i is a vertex's skin weight to its i-th nearest bone, and the statistical ratio of influential bones among vertices' Top-K nearest bones to that of all influential bones (the orange line in Figure 6) as $= |\mathcal{I}_K|/|\mathcal{I}|$, where $\mathcal{I}_K$ is the set of influential bones in the Top-K nearest bones $\mathcal{I}_K = \{b_i | i <= K$ and $w_i > 10^{-4}\}$ and $\mathcal{I}$ is the set of all influential bones $\mathcal{I} = \{b_i | w_i > 10^{-4}\}$, respectively. Here, we assume that bone b_i is influential if $w_i > 10^{-4}$. We found that the vertices are bound to their Top-3 nearest bones with more than 90% of total skin weights, and the influential bones among these bones account for 90% total influential bones. To achieve a balance between prediction accuracy and computation efficiency, we choose $K = 3$. The smooth factor in the loss function is empirically set to 0.4 according to grid search.
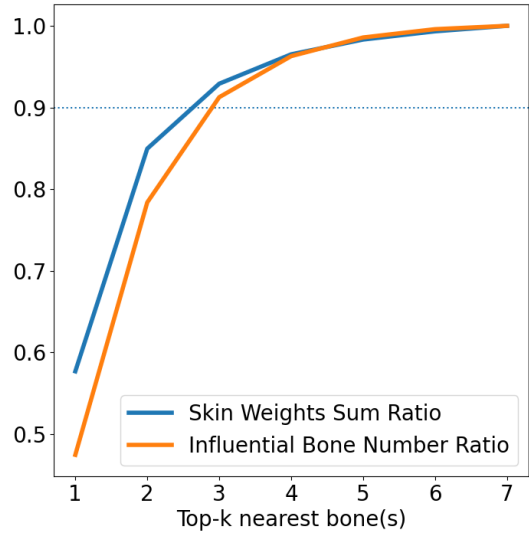

Fig. 6. For models in the ModelsResource-RigNetv1 dataset [Xu et al. 2019], we calculate the statistical ratio of the skin weights sum of Top-K nearest bones to that of all bones, and the statistical ratio of influential bones in the Top-K nearest bones to all influential bones.

4.3 Evaluation Metrics

To evaluate the network's ability to predict skin weights, we leverage the following quantitative metrics:

Precision and Recall. The two metrics indicate the ability to find bones influential to vertices, which have similar meanings to those in pattern recognition. Precision is the fraction of correctly predicted influential bones in all predicted influential bones, and recall is the fraction of those in all influential bones in the ground truth.

L1-norm. This metric is used to measure the numerical accuracy of predicted weights, i.e., the distance between predicted skin weights and the ground truth. A character model's L1-norm is the average of L1-norms over its vertices.

Distance error. Besides directly measuring the similarity between the predicted results and the ground truth, we evaluate the deformation quality of models with our predicted skin weights. This

metric represents the distance between deformed meshes, skinned with predicted skin weights and the ground truth. To simulate possible poses in real motions, we generate new poses by rotating 30% of bones of a model by angles sampled independently from normal distributions $\mathcal{N}(0, 25°)$. For each character model, we generate 10 new poses, some of which are shown in Figure 8. The generated poses may appear globally invalid, but the local deformations are meaningful and possibly seen in real motions. The distance error is the average of the Euclidean distances between the corresponding vertices of the two meshes.

4.4 Model Accuracy

Table 2. Evaluation of the predicted skin weights for our example characters.

Character	Precision	Recall	L1-norm	Dist. Err.
1	79.15%	91.53%	0.2715	0.002127
2	64.86%	83.67%	0.5661	0.007667
3	83.48%	83.02%	0.2795	0.004147
4	88.91%	89.17%	0.1645	0.001231

The overall quantitative metrics over the test set is shown in Table 3. The metrics for our example characters are listed in Table 2. The results show that our network can effectively predict the skin weights for new characters with low errors in general. We then investigate the worst cases. Figure 7 visualizes the per-vertex L1-norm errors on example character meshes. High errors are found on the large curly hair of character 1, the neck and tail of character 2 and the belly of character 4. A further investigation shows that there is a scarcity in the training data which contains similar geometries as the ones with high predicted errors shown in Figure 7, which can be improved by incorporating more data. Besides, errors tend to appear on regions near joints, such as shoulders, ankles and torsos, where vertices are influenced by multiple connected bones. Different artists tend to have different understanding of the anatomies of those areas and thus paint different skin weights on similar areas, which introduces noises to the training data.

Figure 8 shows a side-by-side comparison of 4 randomly generated poses of character 1 with our predicted skin weights and the ground truth, as well as the error maps between the corresponding meshes. The largest errors are mainly on the end of the dress due to the lack of similar structures in the training set. Even with those errors, the deformation results are visually plausible and similar to those generated by the ground truth, which demonstrates that our network can predict ready-to-use skin weights for high-quality deformations.

Table 3. Comparisons with other skinning methods

Method	Precision ↑	Recall ↑	L1-norm ↓	Dist. Err. ↓
GeoVoxel	73.29%	72.21%	0.6057	0.010342
NeuroSkinning	75.25%	76.07%	0.5460	0.008487
RigNet	79.94%	79.02%	0.4075	0.006823
Ours	**83.04%**	**81.11%**	**0.3269**	**0.005682**

Fig. 7. Per-vertex L1-norm errors on example models.

Fig. 8. Randomly generated poses for a character. The meshes on the top row are deformed by the ground truth, meshes on the middle row are deformed by our predicted skin weights, and the color maps on the bottom row indicate per-vertex distance errors.

4.5   Comparisons

We compare our method with Geodesic Voxel binding (GVB) [Dionne and de Lasa 2013], NeuroSkinning [Liu et al. 2019] and RigNet [Xu et al. 2020]. For GVB, we use the implementation in Maya [Autodesk 2020] with 3 influential bones, 0.5 falloff and 128 resolution. For NeuroSkinning, since the models in the dataset have different skeletal morphologies and cannot fit into a super-skeleton, we only use their network architecture and their way in selecting the nearest bones, i.e., based on the Euclidean distance. For the distance used in RigNet, we implement the volumetric geodesic distance instead of the approximation used in their public implementation to robustly compute the distance of vertices on disconnected components and intersected triangles. We implement the above two networks using PyTorch Geometric and train and evaluate on the same train/test split as ours with the same hyperparameters (3 influential bones, 0.25 weight pruning threshold).
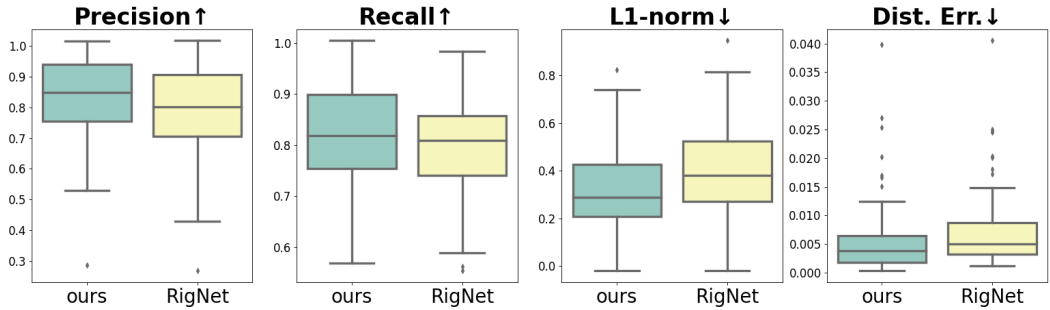

Fig. 9. Statistical comparisons between our method and RigNet [Xu et al. 2020].

We first show the quantitative results in Table 3, where the metrics are averaged over the whole test set. Our numerical results outperform the competing methods by large margins according to all quantitative metrics. We further provide statistical analysis between our method and RigNet [Xu et al. 2020] in Figure 9. One-way ANOVA tests are performed for all the four evaluation metrics, and they are all significant: for precision $F = 4.5259, p = 0.0346 < 0.05$, for recall $F = 3.6959, p = 0.0495 < 0.05$, for L1-norm $F = 12.0506, p = 0.0006 < 0.05$, for distance error $F = 4.0373, p = 0.0458 < 0.05$.

Next, we qualitatively compare our method and competing methods. Figure 10 shows a side-by-side comparison of per-vertex L1-norm errors of skin weights produced by our method and the competing methods. Our method tends to predict skin weights with lower errors. On the top example, errors tend to appear in thin parts, such as the end of the dress and the pigtails. The competing methods ignores the difference between these areas and the body parts and applies the same functions on the whole mesh. Our method extracts both bone features and vertex features and applies functions on these areas according to their bone features, hence estimates the skin weights more close to the ground truth. In the bottom example, errors tend to appear on joint areas such as shoulders, the crotch, and ankles, where vertices are influenced by multiple connected bones. Our method operates on the skeleton graphs and learns the relations between connected bones and thus produces lower errors on these areas than other methods.

We provide a closer comparison of deformation results with skin weights predicted by RigNet [Xu et al. 2020] and our method in Figure 11. In the left column, the ending of the braid and the bottom of the dress are controlled by out-of-body bones, RigNet fails to tackle these bones, making these areas clip with the body. The main reason is that the volumetric geodesic distance they used requires that the bones are inside the body mesh. Our method uses HollowDist, which have no

assumption for bone-mesh relationships and thus is robust for out-of-body bones. In the right column, the ball is not connected with character's head but still controlled by the head bone. RigNet does not assign any bone for the ball, as it cannot calculate volumetric geodesic distances for vertices on it. Our method uses HollowDist, which handles the disconnected ball and assigns it to the head bone.
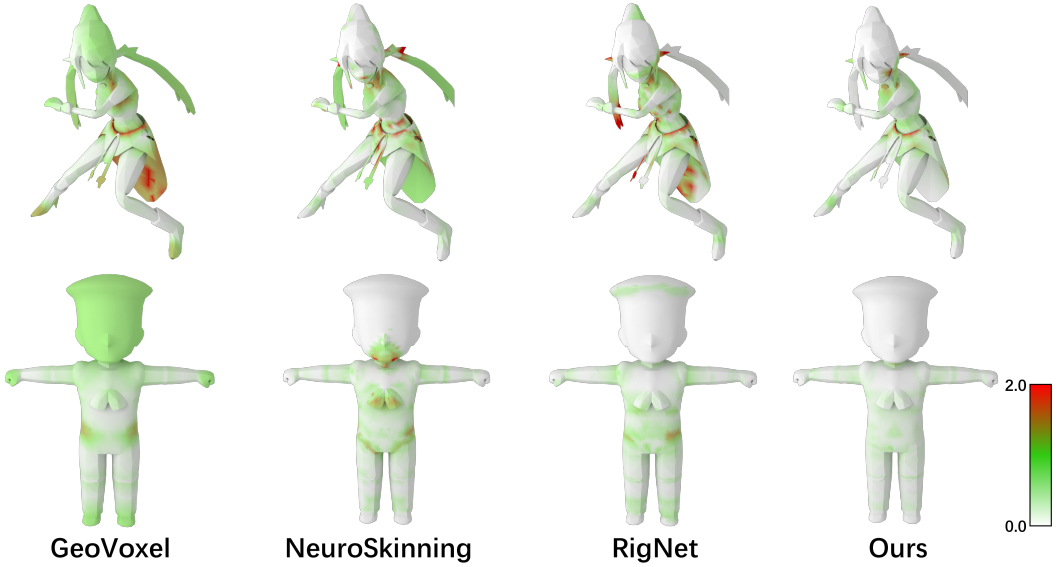

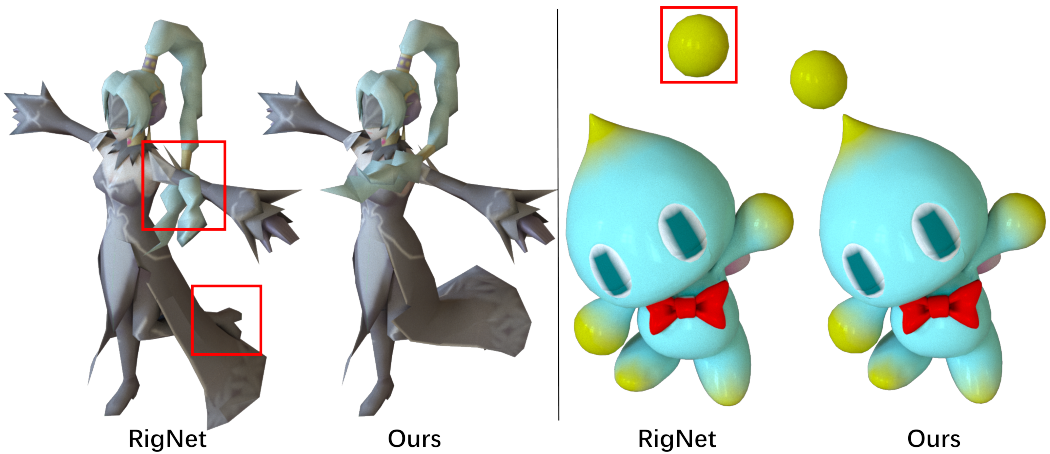Fig. 10. Comparisons of per-vertex L1-norm errors.

Fig. 11. Comparisons of deformation results using the skin weights predicted by RigNet [Xu et al. 2020] and our method.

4.6 Ablation Study

We conduct ablation experiments to analyze the importance of each part in our network. Their quantitative results are shown in Table 4. First, we use the Euclidean distance as an alternative to our proposed HollowDist. The quantitative results show that the network using HollowDist significantly outperforms that using the Euclidean distance, as HollowDist captures the intrinsic properties of the meshes. Next, we remove the inter-graph convolution component, which makes the network extract the vertex and bone features separately. The quantitative result shows that this component helps the network to better extract the node features and thus obtain better prediction results.

The prediction results of our network with and without the smoothing term in the loss function are shown in Figure 12. The smoothing term in the loss function improves the smoothness of predicted skin weights, which makes the deformations more satisfactory.

Table 4. Ablation study of our network

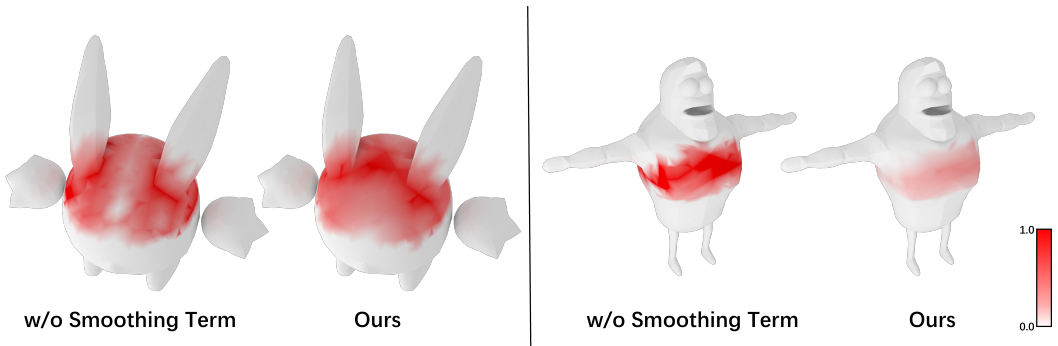| Method	Precision ↑	Recall ↑	L1-norm ↓	Dist. Err. ↓
Ours	**82.11%**	**83.69%**	**0.3269**	0.005682
Euclidean distance	76.23%	76.52%	0.5253	0.008002
w/o HeterConv	80.21%	80.80%	0.3889	0.006243
w/o Smoothing term	81.40%	83.68%	0.3298	**0.005681**

Fig. 12. Comparisons of the skin weights estimated by our network with and without the smoothing term in the loss function.

5 CONCLUSION AND FUTURE WORK

We have presented a heterogeneous graph neural network to automatically estimate skin weights for character rigging. Our inter-graph convolution operation allows feature aggregation between heterogeneous nodes, thus our network can extract both vertex and bone features. Our method can cope with models containing multiple disjoint parts or outside bones with a new distance, HollowDist. Experimental results demonstrate that the skin weights predicted by our network can be used to produce high-quality ready-for-production animations, which greatly reduces the time needed for manual labor.

HeterSkinNet has several limitations. First, models in production are increasingly delicate and sophisticated, which may contain over 10K vertices and over 200 bones [Liu et al. 2019]. Since no such data are available, we only test our method on a dataset whose models have fewer numbers of vertices (1K-5K) and bones (10-150). Evaluating its performance on more complex models is an interesting direction for expanding its applications. Secondly, the calculation of HollowDist takes most of the time in the whole procedure (~1 min). Accelerating this step would be beneficial when the number of models we process is large. Using sparse voxelization and exploring other GPU-based distances may be feasible directions. Thirdly, it assumes that a vertex is influenced by its nearest K bones, where K

Fig. 13. L1-norm errors of some failure cases.

is a user-defined integer. However, the numbers of influential bones of vertices on a mesh vary according to the vertices' semantic information and the skeleton structure. It will be an interesting direction to further expand our network to predict the numbers of vertices' influential bones as well as their skin weights. Fourthly, in the vertex-to-bone convolution, we have only used the maximum, mean and variance of the vertices' features, exploring more statistics such as covariance matrix may be a valuable direction for improving the inter-graph convolution operation. Finally, for regions too far away from their control bones, such as the belly of the lion (top row of Figure 13) and the head of the mushroom (bottom row of Figure 13), our method may fail as these regions have similar HollowDists to their control bones. Exploring vertex-bone features more distinctively on these regions may boost the prediction performance.

ACKNOWLEDGMENTS

REFERENCES

Marc Alexa. 2002. Linear Combination of Transformations. *ACM Transactions on Graphics (TOG)* 21, 3 (2002), 380–387.

Autodesk. 2020. *Maya.* https://www.autodesk.com/products/maya/overview

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), 1–12.

Ilya Baran and Jovan Popović. 2007. Automatic Rigging and Animation of 3D Characters. *ACM Transactions on Graphics (TOG)* 26, 3 (2007), 72–es.

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 (NIPS)*, Vol. 29. 3189–3197.

Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric Deep Learning: Going beyond Euclidean Data. *IEEE Signal Processing Magazine (SPM)* 34, 4 (2017), 18–42.

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 (SCA)*. 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 (TVCG)* 20, 10 (2014), 1367–1378.

Elmar Eisemann and Xavier Décoret. 2008. Single-Pass GPU Solid Voxelization for Real-Time Applications. In *Proceedings of Graphics Interface (GI)*. 73–80.

Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. 2019. MeshCNN: A Network with an Edge. *ACM Transactions on Graphics (TOG)* 38, 4 (July 2019), 1–12.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on Imagenet Classification. In *Proceedings of the IEEE International Conference on Computer Vision(ICCV)*. 1026–1034.

Alec Jacobson, Ilya Baran, Jovan Popovic, and Olga Sorkine. 2011. Bounded Biharmonic Weights for Real-Time Deformation. *ACM Transactions on Graphics (TOG)* 30, 4 (2011), 78.

Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. 2007. Skinning with Dual Quaternions. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D)*. 39–46.

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

Tianxing Li, Rui Shi, and Takashi Kanai. 2020. DenseGATs: A Graph-Attention-Based Network for Nonlinear Character Deformation. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D)*. 1–9.

Hu Linmei, Tianchi Yang, Chuan Shi, Houye Ji, and Xiaoli Li. 2019. Heterogeneous Graph Attention Networks for Semi-Supervised Short Text Classification. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 4820–4829.

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 Transactions on Graphics (TOG)* 38, 4 (July 2019), 1–12.

Ran Luo, Tianjia Shao, Huamin Wang, Weiwei Xu, Xiang Chen, Kun Zhou, and Yin Yang. 2020. NNWarp: Neural Network-Based Nonlinear Deformation. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 26 (April 2020), 1745–1759.

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 (ICCVW)*. 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 (CVPR)*. 5425–5434.

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).

R.M. Rustamov, Y. Lipman, and T. Funkhouser. 2009. Interior Distance Using Barycentric Coordinates. *Computer Graphics Forum (CGF)* 28, 5 (July 2009), 1279–1288.

Michael Schwarz and Hans-Peter Seidel. 2010. Fast Parallel Surface and Solid Voxelization on GPUs. *ACM Transactions on Graphics (TOG)* 29, 6, Article 179 (2010), 10 pages.

SideFX. 2020. *Houdini*. https://www.sidefx.com/

Justin Solomon, Raif Rustamov, Leonidas Guibas, and Adrian Butscher. 2014. Earth Mover's Distances on Discrete Surfaces. *ACM Transactions on Graphics (TOG)* 33, 4 (July 2014), 1–12.

Qingyang Tan, Lin Gao, Y. Lai, Jie Yang, and S. Xia. 2018. Mesh-based Autoencoders for Localized Deformation Component Analysis. *CoRR* abs/1709.04304 (2018).

Nitika Verma, Edmond Boyer, and Jakob Verbeek. 2018. FeaStNet: Feature-Steered Graph Convolutions for 3D Shape Analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2598–2606.

Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S. Yu. 2019a. Heterogeneous Graph Attention Network. In *The World Wide Web Conference (WWW)*. 2022–2032.

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. 2019b. Dynamic Graph CNN for Learning on Point Clouds. *ACM Transactions on Graphics (TOG)* 38, 5 (2019), 1–12.

Rich Wareham and Joan Lasenby. 2008. Bone Glow: An Improved Method for the Assignment of Weights for Mesh Deformation. In *Articulated Motion and Deformable Objects (AMDO)*, Francisco J. Perales and Robert B. Fisher (Eds.). 63–71.

Chao Wen, Yinda Zhang, Zhuwen Li, and Yanwei Fu. 2019. Pixel2Mesh++: Multi-View 3D Mesh Generation via Deformation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 1042–1051.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S. Yu, Philip. 2020. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)* (2020).

Yun-Peng Xiao, Yu-Kun Lai, Fang-Lue Zhang, Chunpeng Li, and Lin Gao. 2020. A Survey on Deep Geometry Learning: From a Representation Perspective. *Computational Visual Media (CVM)* 6, 2 (2020), 113–133.

Zhan Xu, Yang Zhou, Evangelos Kalogerakis, Chris Landreth, and Karan Singh. 2020. RigNet: Neural Rigging for Articulated Characters. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 1–12.

Zhan Xu, Yang Zhou, Evangelos Kalogerakis, and Karan Singh. 2019. Predicting Animation Skeletons for 3D Articulated Models via Volumetric Nets. In *International Conference on 3D Vision (3DV)*. 298–307.

Li Yi, Hao Su, Xingwen Guo, and Leonidas J. Guibas. 2017. SyncSpecCNN: Synchronized spectral CNN for 3D Shape Segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2282–2290.

Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2020. Deep Learning on Graphs: A Survey. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* (2020).

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and M. Sun. 2018. Graph Neural Networks: A Review of Methods and Applications. *CoRR* abs/1812.08434 (2018).