Morig: Motion-Aware Rigging of Character Meshes from Point Clouds

Zhan Xu  
UMass Amherst  
Amherst, MA, USA  
zhanxu@cs.umass.edu

Yang Zhou  
Adobe Research  
San Jose, CA, USA  
yazhou@adobe.com

Li Yi  
Tsinghua University  
Haidian, Beijing, China  
ericyi0124@gmail.com

Evangelos Kalogerakis  
UMass Amherst  
Amherst, MA, USA  
kalo@cs.umass.edu

Figure 1: Given (a) an input mesh and (b) a single-view point cloud sequence capturing a performing character, our deep learning method, called MoRig, automatically rigs and animates the mesh based on the point cloud motion. This is achieved by (c) producing motion-aware features on the mesh encoding articulated parts from the captured motion, (d) using the features to infer an appropriate skeletal rig for the mesh, and (e) re-targeting the motion from the point cloud to the rig.

ABSTRACT
We present MoRig, a method that automatically rigs character meshes driven by single-view point cloud streams capturing the motion of performing characters. Our method is also able to animate the 3D meshes according to the captured point cloud motion. MoRig’s neural network encodes motion cues from the point clouds into features that are informative about the articulated parts of the performing character. These motion-aware features guide the inference of an appropriate skeletal rig for the input mesh, which is then animated based on the point cloud motion. Our method can rig and animate diverse characters, including humanoids, quadrupeds, and toys with varying articulation. It accounts for occluded regions in the point clouds and mismatches in the part proportions between the input mesh and captured character. Compared to other rigging approaches that ignore motion cues, MoRig produces more accurate rigs, well-suited for re-targeting motion from captured characters.

KEYWORDS
character rigging, animation, neural networks, transformers

ACM Reference Format:
Zhan Xu, Yang Zhou, Li Yi, and Evangelos Kalogerakis. 2022. Morig: Motion-Aware Rigging of Character Meshes from Point Clouds. In SIGGRAPH Asia 2022 Conference Papers, December 6–9, 2022, Daegu, Republic of Korea. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3550469.3555390

1 INTRODUCTION
With the emergence of virtual worlds, mixed reality and 3D social media, the need for diverse, high-quality, animation-ready avatars is greater than ever. To animate articulated characters, it is common for artists to hand-craft character rigs for meshes. These rigs are often represented as “skeletons” with “skinning” that binds the skeletons to the meshes. Unfortunately, hand-crafting skeletons and skinning require laborious effort and extensive experience.

To accelerate the process, a few approaches have been proposed to automate steps involved in rigging. One family of approaches fits a particular skeletal template to the input character [Baran and Popović 2007; Li et al. 2021a], which can then be animated according to motion capture data for that template. These approaches are well-suited for motion re-targeting, yet require different templates to be hand-crafted for characters with varying articulation structure. A different family of approaches [Liu et al. 2019; Xu et al. 2020] is able to infer rigs for diverse characters e.g., animals, toys, robots, and other man-made objects. However, such methods are motion-agnostic: they process a single static mesh, and may end up creating a skeleton which is far from the users’ expectations for animation. Our method, called MoRig, attempts to build a common ground...
between the above two families of methods: it is able to rig diverse characters while also considering motion cues to drive the rigging process. Specifically, given a point cloud sequence capturing the motion of a performing character, MoRig infers a skeletal rig for a mesh of a target character. The rig matches the articulating parts of the captured character. In contrast to motion-agnostic approaches [Liu et al. 2019; Xu et al. 2020], MoRig is also able to animate the skeletal rig of the target character to match the captured motion.

Inferring character rigs and motion from point cloud guidance has several challenges. First, captured point clouds, in particular ones captured from a single view, suffer from occlusions allowing only partial observations of the moving parts in the performing character. Second, the captured character may not match well the input target mesh due to geometric differences, such as mismatches in part proportions. Third, transferring the motion from the point cloud requires controlling the produced rig to closely match this motion, while also being robust to noise and outliers. MoRig addresses the challenges through a novel mesh deformation network that attempts to closely align the target character with each point cloud frame, while factoring out mismatches in part proportions and noise. To infer the skeletal rig, our method incorporates a neural network that considers both the geometry of the target mesh as well as motion-aware features produced by a motion encoder module based on transformers [Vaswani et al. 2017]. Our method improves the rigging and skinning of diverse characters compared to state-of-the-art methods, as demonstrated in our experiments. To summarize, our contributions are the following:

- A deformation module that aligns target meshes of articulated characters with point clouds under noise and occlusion.
- A transformer-based motion encoder that encodes motion trajectories of mesh vertices into features revealing articulated parts.
- A unified pipeline of rigging, skinning, and motion transfer from point cloud sequences to meshes of diverse articulated characters.

2 RELATED WORK

We briefly review here relevant lines of work in automatic rigging and structure understanding from point cloud motion.

Template-based rigging. In the pioneering work of Pinocchio, Baran et al. [2007] proposed fitting input skeleton templates to a given 3D mesh through an optimization procedure. Each template has specific topology, fixed number of joints, and is suited for a specific class of characters. Instead of hand-tuned optimization, Li et al. [2021a] proposed a neural network approach to estimate joint positions for a given skeletal template. They also improve skeleton-based deformations through a neural blend shape technique. An advantage of such template-based approaches is that motion capture data for a particular template can easily be applied. However, they are only able to rig characters whose articulation structure is compatible with the given template. A different template and network has to be trained for characters with different articulation structure (e.g., different number of parts or bone connectivity).

Template-free rigging. Another family of approaches attempts to discover the underlying articulation structure of a given shape without relying on templates. Geometric methods create skeletons based on the medial axis or centerline of shapes [Au et al. 2008; Huang et al. 2013; Tagliasacchi et al. 2012]. However, these methods often fail to discover underlying moving parts since they rely entirely on shape geometry. Neural networks approaches have also been proposed to infer the articulation structure from the input geometry of a shape [Xu et al. 2020, 2019b]. These networks are trained on large databases of rigged characters, thus they tend to extract skeletons that often correspond well to articulated parts. Despite their advantage of handling characters with diverse articulation structures, these methods operate on single static shapes without considering any motion cues. As a result, their resulting skeletons might be incompatible with desired animations. Our approach is inspired by such learning-based methods. It also re-uses some of the mesh neural networks proposed in [Xu et al. 2020] for joint and skinning prediction. Yet, the main difference is that our method considers both geometry and motion cues as input for rigging. Furthermore, our method is able to animate the resulting rigs automatically given the motion captured in an input point cloud sequence.

Skinning. Another aspect of rigging is skinning prediction, which defines the influence from each bone to each vertex. Various geometric-based methods have been proposed for skinning [Dionne and de Lasa 2013; Jacobson et al. 2011; Kavan et al. 2007; Le and Hodgins 2016]. Yet these often require hand-tuning. Liu et al. [2019] takes as input a skeleton template and predicts skinning based on a learned graph neural network. Our method also learns to predict skinning, yet the predictions rely not only on geometric, but also motion cues.

Rigging from mesh sequences. Several methods have been proposed to predict skeletons, including skinning weights, from sequences of meshes [De Aguiar et al. 2008; James and Twigg 2005; Kavan et al. 2010; Le and Deng 2012, 2014]. However, these mesh sequences must be manually created through modeling operations. They must be complete, noiseless, and have one-to-one correspondences between all their vertices. Instead of relying on manual effort to create mesh sequences, our method performs rigging driven by single-view performance capture of characters through commodity sensors.

Structure understanding from point clouds. Several methods have been proposed to discover articulation structure from point clouds [Hayden et al. 2020; Li et al. 2016, 2007; Wang et al. 2019; Xu et al. 2019a; Yan et al. 2019; Yi et al. 2018; Yuan et al. 2016]. Given multiple observations of a captured object, they segment each point cloud into underlying moving parts. They trend not to produce consistent parts across the point cloud sequence, thus often require some form of synchronization [Huang et al. 2021]. Our goal is different in the sense that we aim to extract an animation skeleton that is consistent with both the input point cloud sequence and a target mesh. Another key technical difference is our use of a motion transformer to encode vertex trajectories. A more similar line of work to ours attempts to fit skeletons or deformation graphs to point cloud sequences [Božič et al. 2021; Lu et al. 2018, 2019; Tzionas and Gall 2016; Zhang et al. 2013]. However, their outputs are specific to the input point clouds, and cannot readily be transferred to meshes or to other characters with different part proportions. Our method instead takes into account both the geometry of a target mesh and the motion in the point cloud sequence for character rigging.
3 METHOD

Overview. Given an input target mesh \( M \) representing an articulated character, the goal of our method is to rig it such that it can be animated according to the motion captured in a sequence of input point clouds \( \mathcal{P} = \{\mathcal{P}^{(t)}\}_{t=1}^{T} \) where \( T \) is the number of frames in the sequence. We assume that these point cloud frames capture different poses of a reference character whose moving parts correspond to the underlying articulated parts in the target mesh \( M \). For example, if the animator expects the target skeletal rig to contain bones for articulated parts, such as arms, legs, head, ears, and so on, these parts should be (a) present in the captured reference character, (b) captured under different poses in the input point clouds. Our method is designed to handle both partial observations, e.g., single-view point clouds with occlusion, and also mismatches in part proportions e.g., different limb length and diameter between the reference and target characters. We also note that we do not assume any prior knowledge about the number or type of parts in the target character – our goal is to discover them driven by the point cloud motion of the reference character. After rigging, our method can use the rig to produce an animated mesh sequence \( \hat{M} = \{\hat{M}^{(t)}\}_{t=1}^{T} \) that matches the motion of the input point clouds.

The pipeline of our method is highlighted in Figure 2. The first stage of our pipeline is a correspondence module. Its goal is to (a) assign the target mesh vertices with a soft correspondence mask representing the probability for them to have a corresponding point per frame, and (b) a matching score between each vertex and each point per frame. Its outputs drive a deformation module such that (a) vertices assigned with high probability in the correspondence mask are deformed towards matching points (b) the deformation is also propagated to vertices that have low probability in the mask due to occlusions or any small structural differences between the target and reference character. As a result, the deformation module closely aligns the target mesh with each point cloud frame.

The deformed meshes help establish per-vertex motion trajectories, which are subsequently processed by a motion encoder module. The module has the form of a transformer encoder that aims to convert the complete motion trajectories across all frames into motion-aware features per each vertex on the target mesh. The features are highly correlated with underlying parts i.e., vertices belonging to different articulated part tend to have distinct features. They are therefore particularly informative for locating skeletal joints and assessing the skinning weights on the target mesh (Figure 3). The skeleton and skinning predictions are performed through a rigging module, which follows RigNet [Xu et al. 2020], yet with the difference that this module utilizes as input our motion-aware features. The output rig is used to produce an improved mesh deformation sequence which is consistent with both the inferred skeleton and the point cloud sequence. In the rest of this section, we describe the above modules at test time. Training is discussed in Section 4.

3.1 Correspondence Module

This module computes partial correspondences between the vertices in the target mesh \( M \) and the points per point cloud frame \( \mathcal{P}^{(t)} \).

Point feature extraction. To perform this task, we process the point cloud in each frame \( \mathcal{P}^{(t)} \) through a PointNet++ network [Qi et al. 2017] to output a feature descriptor \( f_p^{(t)} \) for each point \( p^{(t)} \) at frame \( t \). The output feature is normalized to unit length (details on our PointNet++ architecture variant are included in the supplement).

Vertex feature extraction. In addition, for each target mesh vertex \( v \in M \), we extract a feature descriptor \( f_v \) through a GMEdgeNet network performing graph convolution on the target mesh [Xu et al. 2020]. The network relies on a message passing procedure, in which each mesh vertex accumulates messages from both its topological (one-ring) and geodesic neighbors on the mesh. By stacking several layers of message passing, each vertex encodes information from a larger context around it. The output per-vertex feature descriptor \( f_v \) from the last layer is also normalized to unit length (architecture details can be found in the supplement).

Point-Vertex matching scores. The matching score between each mesh vertex and input point is computed through cosine similarity: \( \gamma^{(t)}_{p,v} = \mathbf{f}_p^{(t)} \cdot \mathbf{f}_v \). The networks for feature extraction are trained with supervisory signal that encourages corresponding mesh vertices and points to have high similarity, as explained in Section 4.

Correspondence mask. Some vertices may not have any corresponding points due to structural differences between the target and reference character, and also due to occlusions in the reference point clouds. We employ another network, in the form of a two-layer MLP, to assess the probability \( q^{(t)}_{p,v} \) for a mesh vertex \( v \) to have correspondence with any point in the point cloud frame \( \mathcal{P}^{(t)} \). More specifically, for each mesh vertex, the MLP takes as input a vector concatenating its own feature descriptor \( \mathbf{f}_v \), the feature descriptor \( \mathbf{f}_p^{(t)} \) of the most similar point \( p(v) \) at frame \( t \), where \( p(v) = \arg \max_p \gamma^{(t)}_{p,v} \) as well as their matching score \( \gamma^{(t)}_{p,v} \). The MLP outputs mask probabilities with higher values for vertices with correspondences in the captured point cloud. We found that using both these vertex and nearest point feature vectors helped assess the above probability (see our supplement for mask visualizations).

3.2 Deformation Module

Given the vertex-point matching scores and the probabilistic correspondence mask, the deformation module deforms the target mesh \( M \) towards each frame \( \mathcal{P}^{(t)} \). One possibility to perform such deformation is to move each mesh vertex towards its most similar point in the frame i.e., given a vertex position \( \mathbf{y}_v \) and its closest matching point position \( \mathbf{y}^{(t)}_{p(v)} \) at frame \( t \), the vertex displacement could be computed as \( \mathbf{d}_v^{(t)} = \mathbf{y}^{(t)}_{p(v)} - \mathbf{y}_v \). This strategy is far from optimal since the displacements can easily be affected by noise, occlusions, and overall uncertainty. We found that a better strategy is to displace each vertex towards a weighted average of points using the matching scores as weights, i.e., \( d_{v}^{(t)} = \frac{\sum_p \gamma^{(t)}_{p,v} \cdot (\mathbf{y}^{(t)}_p - \mathbf{y}_v)}{\sum_p \gamma^{(t)}_{p,v}} \) where \( \tau \) is learned.

Deformation propagation. Still, the above displacements become unreliable for vertices with low probability \( q^{(t)}_{p,v} < 50\% \) (the threshold is set
We can further improve the above displacements by processing while being robust to occlusions. (c) The motion encoder converts the resulting per-vertex trajectories to motion-aware features that are correlated with underlying articulated parts. (d) The rigging module outputs a character rig by utilizing these features. The input mesh is animated according to the rig and the point cloud motion.

For these vertices, we replace their displacements with the ones coming from their closest and reliable geodesic neighbor (i.e., with $q_v^{(t)} \geq 50\%$). This encourages these vertices with no correspondences to deform more coherently with the rest of the shape. However, the deformation tends to remain discontinuous in some areas since the displacement is separately calculated for each vertex and also due to the sensitivity of the above probability threshold. We can further improve the above displacements by processing them through another GMEdgeNet. The network here takes as input concatenated per-vertex positions, displacements, and correspondence probabilities i.e., $x_v = [y_v, d_v^{(t)}, q_v^{(t)}]$. The network performs feature transformations and message passing such that the input features are diffused and propagated over the mesh. The network outputs updated per-vertex displacements $e_v^{(t)}$ representing more coherent deformations aligning the target mesh closely to each point cloud frame. We observed that all the above features are useful for estimating accurate deformations i.e., vertices with low correspondence probability have larger changes since their initial displacement is not reliable.

**Iterative deformation.** The deformation proceeds iteratively. After computing the displacements for the first frame, the mesh is deformed by applying the per-vertex displacements: $y_v^{(1)} = y_v + e_v^{(1)}$. At the next frame, we compute correspondences and matching scores between the deformed mesh and the next point cloud frame $p^{(2)}$. We then compute displacements to deform this mesh towards the next point cloud frame, resulting in updated vertices $y_v^{(2)} = y_v^{(1)} + e_v^{(2)}$. This procedure iterates until the last frame.

### 3.3 Motion encoder

The motion encoder aims to transform the per-vertex estimated displacements into motion-aware features, which help to estimate the rig. The input to the motion encoder is a per-vertex sequence of displacements $\{e_v^{(t)}\}_{t=2}^T$. It is important to note here that the encoding considers displacements computed after the initial deformation. The initial displacements $e_v^{(1)}$ are used to align the target mesh with the first point cloud frame, and accounts for both pose and shape differences between the target and reference character. The rest of the displacements represent motion trajectories relative to the initially deformed mesh. We extract motion-aware features representing the motion of this deformed mesh, which also drive the rigging process. Since the features are computed with respect to this deformed mesh after initial displacement, the skeletal extraction and skinning (described in Sec. 3.4) are performed in this deformed mesh, and then transferred to the original target mesh.

**Transformer encoder.** We tried a number of strategies to encode the per-vertex trajectories. We found that the best performance was achieved through a transformer-based encoder module. Specifically, for each frame, the encoder first embeds each vertex displacement to a $D$-dimensional feature vector through a GMEdgeNet ($D = 32$ in our implementation). This results in a sequence of features $\{h_v^{(t)}\}_{t=2}^T$. We insert an extra token with a learned embedding $g_v^{(1)}$ at the beginning of the sequence via another MLP, similar to the [CLS] token in vision transformers [Dosovitskiy et al. 2021]. This token plays the role of representing the whole sequence (and not the first frame). The transformer has multi-attention heads, each of which transforms the above features into query, key, value representations following [Vaswani et al. 2017]:

$$q_v^{(t)} = Q_v g_v^{(t)}, \quad k_v^{(t)} = K_v g_v^{(t)}, \quad v_v^{(t)} = V_v g_v^{(t)},$$

where $n$ denotes the attention head index, and $Q_v, K_v, V_v$ are learnable parameters for query, key, and value transformations respectively. These parameters are shared across all mesh vertices. We then compute attention matrices: $A_{v,i}^{(t)} = \text{softmax}(q_{v,i}^{(t)} \cdot k_{v,i}^{(t)}/\sqrt{D})$. The
The per-vertex motion features encoding the whole sequence: \( B_{v:n}^{(t)} = \sum_{i=1}^{T} A_{v:n}^{(t)} \). The features from all attention heads are concatenated, are projected back to a \( D \)-dimensional feature vector with a linear transformation \( U \) shared across all vertices: \( h_v = [B_{v:1}^{(1)}, ..., B_{v:N}^{(1)}]U \), where \( N \) is the number of attention heads. The feature vector \( h_v \) represents the per-vertex motion-aware features encoding its trajectory. Details about the motion encoder architecture are provided in the supplement.

### 3.4 Rigging

The per-vertex motion features \( h_v \) are used by RigNet modules [Xu et al. 2020] for joint extraction and skinning. Both joint and skinning are estimated on the mesh after the initial deformation step. Below, we discuss these modules along with our changes.

**Joint Extraction.** The joint extraction in RigNet is performed through a combination of regression and clustering. In the regression step, the mesh vertices are displaced towards nearest candidate joint locations. Since not all vertices are equally useful to determine joint locations, another GMEdgeNet assesses a per-vertex scalar value representing the confidence of localizing a joint from that vertex. After displacement, the vertices are clustered into joints. Our motion-aware features are particularly useful to better assess the importance of vertices: by computing the difference between the motion features of a vertex with respect to each of its neighbors, then aggregating these differences through pooling (as done in GMEdgeNet), we can localize boundaries between articulated parts that are close to joint areas (Figure 3, middle). Vertex displacements can also benefit from motion-aware features since they are highly similar within parts, making the displacements and resulting clustering into joints more coherent (Figure 3, right). Thus, for both GMEdgeNets used for joint displacement and the vertex relevance map, we take vertex positions with their motion-aware features as input. The resulting joint locations are significantly improved compared to the original RigNet that uses only vertex positions as input (Figure 4). After joint discovery, we use RigNet’s minimum spanning tree procedure to form the animation skeleton. If the target mesh has bilateral symmetry, we reflect discovered joints according to this mesh symmetry plane to enforce symmetric skeletons.

**Skinning.** For skinning prediction, RigNet uses a GMEdgeNet processing vertex positions as well as volumetric distances to nearest bones. Similarly, we append the motion-aware features as additional input to the network. As a result, the skinning weights become more coherent within articulated parts since the motion-aware features tend to be similar across their vertices (Figure 5).

**Rig transfer.** Our next step is to transfer the skeleton and skinning weights from the deformed mesh, whose vertex positions are \( y^{(1)} = \{y_v^{(1)}\}_{v=1}^V \), to the original target mesh with vertex positions \( y = \{y_v\}_{v=1}^V \), where \( V \) is the number of vertices. These two meshes have explicit correspondences between their vertices, yet they may differ in pose and part proportions. There are various strategies to transfer skeletons across meshes driven by vertex correspondences [Allen et al. 2003; Avril et al. 2016], however they use hand-engineered energy optimization approaches or are sensitive to select particular vertices as markers. In our case, we resorted to fit a joint regressor that maps the 3D positions of vertices to joint locations: \( j = f(y) \), where \( j = \{j_i\}_{i=1}^J \) are joint positions, \( J \) is the number of joints extracted from the rigging step, and \( f \) is a neural network interpolating function in the form of a MLP with a single hidden layer. Fitting is performed with gradient descent using as training data variants of the mesh \( y^{(1)} \). The variants are created by applying random rotations and anisotropic scaling transformations to the joints of that mesh. We found that sampling about 300 variants promotes an accurate fitting of joint positions under different mesh configurations i.e., the average Chamfer distance between predicted joints and ground-truth ones in neutral-pose meshes (unseen during training) was less than 0.1% of the largest bounding box axis of test meshes in the “ModelsResource” dataset (discussed in our experiments). The fitting is fast, taking 3 sec per mesh (measured on a NVidia 2080Ti). After fitting, we estimate the joint positions with the MLP. The skinning weights are copied to corresponding vertices from the deformed mesh to the target one.

**Motion transfer.** Given the motion trajectories of the deformed mesh vertices \( y^{(1)} \) and the skeleton, we apply full-body IK [Parent 2012] to compute joint angles such that the deformed mesh follows the point cloud motion. The joint angles are transferred to the original mesh through IK-based retargeting [Lee and Shin 1999].

### 4 TRAINING

We now describe the training of all the modules in our pipeline.

**Datasets.** We rely on two datasets for training our modules. First, we make use of the training split of the ModelsResources dataset [Xu et al. 2020], which contains 2, 163 rigged meshes. Since there are no animations associated with these characters, we synthesize animation sequences for each mesh by adding random rotations to the joints represented in keyframes, then interpolate the motion resulting in a 100-frame sequence per character (total 21, 630 frames).

As a second dataset, we use the DeformingThing4D dataset [Li et al. 2021b]. The dataset has the advantage of containing realistic motion data for humanoid characters based on motion capture, as well as animations for non-humanoid characters created by artists. For training, we use 83 characters from this dataset. Each model has multiple animation sequences (on average, 15.44 sequences per
To evaluate skinning alone, we use the animator’s skeleton (a) the animator’s skinning weights, (b,c) MoRig’s skinning vertices and their closest points in the point cloud per frame. We use binary cross entropy to supervise correspondence probability learning:

\[ L = \sum_{t, v} \log \frac{\exp(f_v(t)) \cdot f_\theta / \tau}{\sum_{p' \in \mathcal{R}(v)} \exp(f_{p'}(t)) \cdot f_\theta / \tau} \]

where \( \tau \) represents a learnable temperature [Radford et al. 2021]. Similarly, we use the same loss to match the representation of each point with its corresponding vertex.

**Deformation module training.** To train the GMEdgeNet used in the deformation module, we provide supervisory signal in the form of ground-truth displacements \( \Delta_v^{(t)} \) for each mesh vertex \( v \) at frame \( t \) in the training animation sequences. We use the L1 loss to penalize deviation of predicted displacements from the ground-truth ones:

\[ L_{\text{flow}} = \sum_{v, t} |\Delta_v^{(t)} - \Delta_v^{(t)}|_1. \]

We pre-train the correspondence and the deformation modules on the synthetic sequences of the ModelsResource dataset, then fine-tune them on the sequences provided in the DeformingThing4D dataset.

**Motion encoder training.** To train the transformer encoder, we provide supervisory signal such that the output motion features \( h_v \) are similar for vertices with similar rigid motion. The training skinning weights can help us assess such vertices. Specifically, if two vertices of the mesh have almost identical skinning weights, they are expected to have similar motion features. Given two vertices \( v \) and \( u \), we examine if the L1 difference of the ground-truth skinning weights \( |s_v - s_u| \) is small (we use a threshold 0.1). We use the InfoNCE loss to promote similar motion features for such vertices:

\[ L_{\text{motion}} = - \sum_{v, u} \log \frac{\exp(h_v \cdot h_u / \tau^{'})}{\sum_{v' \in \mathcal{R}(v)} \exp(h_v \cdot h_{v'}/ \tau^{'})} \]

where \( \mathcal{R}(v) \) is the set of vertices whose skinning weights differ from vertex \( v \) based on the above threshold, and \( \tau^{'} \) is a learnable parameter. We note that we use this loss on the ModelsResources training split only, which provides ground-truth skinning weights.

**Rigging module training.** To train the rigging module, we follow the same losses under the same training procedure described in [Xu et al. 2020] on the ModelsResources training split.
Figure 6: Rigging and deformation results with real-world point cloud sequences from DFaust (top) and KillingFusion [Slavcheva et al. 2017] (bottom). For each example, the top row shows the original target mesh, and three of the input point cloud frames. In the bottom row, we show the predicted motion features in the red rectangle, our rigging results, and then deformed meshes corresponding to the point cloud frames (see also video for animated results).

Table 1: Comparison with other methods and MoRig variants.

| Joint evaluation | Skinning evaluation | Anim.    |
|------------------|--------------------|----------|
|                  | Chamfer           | IoU      | Precision | Recall | Avg L1 | L2 dist |
| SkeRig           | 7.5%              | 26.5%    | 33.4%     | 23.2%  | N/A    | N/A     |
| RigNet           | 3.9%              | 61.6%    | 67.6%     | 58.9%  | 82.3%  | 80.8%   | 0.39    | 3.1%    |
| MoRig            | 3.5%              | 64.7%    | 72.2%     | 61.2%  | 83.7%  | 85.5%   | 0.32    | 2.4%    |
| avg pool         | 3.7%              | 61.8%    | 67.2%     | 56.0%  | 77.5%  | 78.6%   | 0.33    | 2.9%    |
| max pool         | 3.6%              | 63.2%    | 70.5%     | 58.8%  | 81.4%  | 83.2%   | 0.34    | 2.8%    |

5 RESULTS

In this section, we describe our experiments. Our code and data are provided here: https://github.com/zhan-xu/MoRig. Please also see our supplementary video for animations and more results.

Comparisons. Since our method does not rely on particular specific skeleton templates, we compare with other template-free rigging methods. We first compare with RigNet [Xu et al. 2020]. RigNet does not consider any motion cues as input. The main point of this comparison is to show that motion cues are useful to achieve better rigging. We note that RigNet is trained on the same split as our method, uses the same training losses for its rigging module, and its hyperparameters are tuned in the same validation split as ours. Second, we compare with [Le and Deng 2014] (abbreviated as “SkeRig”), an optimization-based method for rigging based on input mesh sequences. This method does not process input point clouds. It instead processes mesh sequences to produce both skeletons and skinning for an input target mesh. We provide this method with the target mesh and the corresponding deformed mesh sequence produced by our deformation module. The goal of this comparison is to test whether the proposed deep motion features and rigging modules are better than an optimization method that fits joints and skinning to the mesh sequence produced by our deformation stage.

Qualitative comparisons. Figure 4 shows comparisons of our method with RigNet and SkeRig on test animation sequences from the test split of the ModelsResource dataset. In all examples, the number and placement of joints is more accurate for MoRig. Figure 5 shows comparisons on skinning weight prediction. As shown in the error maps, our method produces more accurate skinning weights.

Quantitative comparisons. We use the evaluation measures proposed in RigNet [Xu et al. 2020] to measure the quality of the predicted skeletons and skinning. The evaluation is performed on the test split of ModelsResource, which contains 270 characters along with artist-made rigs and our 270 synthetically generated animation sequences (one per test character). As seen in Table 1, our method outperforms both RigNet and SkeRig for joint prediction. For example, in terms of Chamfer distance between artist-specified and predicted joints, our method offers a relative reduction of 53% compared to SkeRig (7.5% → 3.5% distance error relative to the character’s height), and by 10% compared to RigNet (3.9% → 3.5%). Our supplement includes more evaluation wrt real scans – same trends were observed there.

Regarding evaluation wrt. skinning, we use the same animators’ skeleton as input to competing methods such that differences in skeleton prediction are factored out. We here compare only with...
We presented MoRig, a method for rigging and animating a character, although we attempted to make our method robust to occlusions and mismatches between part proportions of the target and reference character, large changes in geometry and poses may result in inaccurate rigs and motion transfers. In particular, if the initial point cloud frame is corrupt, then the initial deformation becomes unreliable, and our method will fail. Our method is limited to linear blend skinning, and cannot deal with topological changes. Finally, it requires guidance in the form of point clouds, which may not always be easy to capture.

ACKNOWLEDGMENTS

We thank Kentaro Ko, Takuya Narihira, Tamaki Kojima for helpful discussion. We are grateful for the support from Sony Corporation.

REFERENCES

1. Brett Allen, Brian Curless, and Zoran Popović. 2003. The Space of Human Body Shapes: Reconstruction and Parameterization from Range Scans. ACM TOG 22, 3 (2003).
2. Oscar Xin-Chung Au, Chiare-Lan Tai, Hung-Kuo Chu, Daniel Cohen-Or, and Tong-Yee Lee. 2008. Skeleton Extraction by Mesh Contracture. ACM TOG 27, 3 (2008).
3. Quentin Avril, Donya Ghafourzadeh, Srinivasan Ramachandran, Sabel Fahladoust, Sarah Ribet, Olivier Dinnon, Martin de lasa, and Eric Paquette. 2016. Animation Setup Transfer for 3D Characters. Comput. Graph. Forum 35, 2 (2016).
4. Ilya Baran and Jovan Popović. 2007. Automatic Rigging and Animation of 3D Characters. ACM TOG 26, 3 (2007).
5. Federica Bogo, Javier Romero, Gerard Pons-Moll, and Michael J. Black. 2017. Dynamic FAUST: Registering Human Bodies in Motion. In Proc. CVPR.
6. Aljaz Božič, Pablo Palafoux, Michael Zollhöfer, Justus Thies, Angela Dai, and Matthias Nießner. 2021. Neural Deformation Graphs for Globally-consistent Non-rigid Reconstruction. In Proc. CVPR.
7. Edison De Aguiar, Christian Theobalt, Sebastian Thrun, and Hans-Peter Seidel. 2008. Automatic Conversion of Mesh Animations into Skeleton-based Animations. Comput. Graph. Forum 27, 2 (2008).
8. Olivier Dinnon and Martin de lasa. 2013. Geodesic Voxel Binding for Production Character Meshes. In Proc. SCA.
9. Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In Proc. ICLR.
10. David S Hayden, Jason Pacheco, and John W Fisher. 2020. Nonparametric object and parts modeling with lie group dynamics. In Proc. CVPR.
11. Hui Huang, Shihao Wu, Daniel Cohen-Or, Mingjun Gong, Hao Zhang, Guojin Li, and Baosong Chen. 2013. L1-Medial Skeleton of Point Cloud. ACM TOG 32, 4 (2013).
12. Jiahui Huang, He Wang, Tolga Birdal, Minhyuk Sung, Federica Arrigoni, Shin-Mu Hun, and Leonidas Guibas. 2021. MultiBodySync: Multi-Body Segmentation and Motion Estimation via 3D Scan Synchronization. In Proc. CVPR.
13. Alex Jacobson, Ilya Baran, Jovan Popovic, and Olga Sorkine. 2011. Bounded Biharmonic Weights for Real-Time Deformation. ACM TOG 30, 4 (2011).
14. Doug L. James and Christopher D. Twigg. 2005. Skinning Mesh Animations. ACM TOG 24, 3 (2005).
15. Ladislav Kavan, Steven Collins, Jiří Zára, and Carol O’Sullivan. 2007. Skinning with Dual Quaternions. In Proc. 3DID.
16. L. Kavan, P.-P. Sloan, and C. O’Sullivan. 2010. Fast and Efficient Skinning of Animated Meshes. Comput. Graph. Forum 29, 2 (2010), 327–336.
17. Binh Huy Le and Zhigang Deng. 2012. Smooth Skinning Decomposition with Rigid Bones. ACM TOG 31, 6 (2012).
18. Binh Huy Le and Zhigang Deng. 2014. Robust and accurate skeletal rigging from mesh sequences. ACM TOG 33, 4 (2014).
19. Binh Huy Le and Jessica K. Hodgins. 2016. Real-Time Skeletal Skinning with Optimized Centers of Rotation. ACM TOG 35, 4 (2016).
20. Jehee Lee and Sung Yong Shin. 1999. A Hierarchical Approach to Interactive Motion Editing for Human-like Figures. In Proc. ACM SIGGRAPH.
21. Hao Li, Guowei Wan, Honghua Li, Andriy Shafir, Kai Xu, and Baoquan Chen. 2016. Mobility fitting using 4d ransac. Comput. Graph. Forum 35, 5 (2016).
22. Peizhuo Li, Kfir Aberman, Rana Hancock, Lbin Lin, Olga Sorkine-Hornung, and Hao Li. 2021a. Learning Skeletal Articulations with Neural Blend Shapes. ACM TOG 40, 4 (2021).
23. Ting Li, Vinutha Kallem, Dheeraj Singaraju, and René Vidal. 2007. Projective factorization of multiple rigid-body motions. In Proc. CVPR.
24. Yang Li, Hikari Takehara, Takafumi Taketomi, Ino Zheng, and Matthias Nießner. 2021b. Mcomplete: Non-rigid motion estimation beyond the observable surface. In Proc. ICCV.
25. Lijuan Liu, Youyi Zheng, Di Tang, Yi Yuan, Changjie Fan, and Kun Zhou. 2019. NeudNeck: Automatic Skin Binding for Production Characters with Deep Graph Networks. ACM TOG 38, 4 (2019).
26. Xuequan Lu, Honghua Chen, Sai-Kit Yeung, Zigang Deng, and Wenzhi Chen. 2018. Unsupervised articulated skeleton extraction from point set sequences captured by a single depth camera. In Proc. AAAI.
27. Xuequan Lu, Zhigang Deng, Jun Luo, Wenzhi Chen, Sai-Kit Yeung, and Ying He. 2019. 3D articulated skeleton extraction using a single consumer-grade depth camera. Comput. Vis. Image Underst. 188 (2019).
28. Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. arXiv preprint arXiv:1807.03748 (2018).
29. Rick Parent. 2012. Computer animation: algorithms and techniques. Newnes.
30. Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. 2017. Pointnet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In Proc. NeurIPS.
31. Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In Proc. ICLR.
32. M. Slavcheva, M. Baust, D. Cremers, and S. Ilic. 2017. KillingFusion: Non-rigid 3D Reconstruction without Correspondences. In Proc. CVPR.
Andrea Tagliasacchi, Ibraheem Alhashim, Matt Olson, and Hao Zhang. 2012. Mean Curvature Skeletons. *Comput. Graph. Forum* 31, 5 (2012).

Dimitrios Tzionas and Juergen Gall. 2016. Reconstructing Articulated Rigged Models from RGB-D Videos. In *ECCV Workshops on Recovering 6D Object Pose*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proc. NeurIPS*.

Xiaogang Wang, Bin Zhou, Yahao Shi, Xiaowu Chen, Qinping Zhao, and Kai Xu. 2019. Shape2Motion: Joint Analysis of Motion Parts and Attributes From 3D Shapes. In *Proc. CVPR*.

Xun Xu, Loong-Fah Cheong, and Zhuwen Li. 2019a. 3d rigid motion segmentation with mixed and unknown number of models. *IEEE TPAMI* 43, 1 (2019).

Zhan Xu, Yang Zhou, Evangelos Kalogerakis, Chris Landreth, and Karan Singh. 2020. RgNet: Neural Rigging for Articulated Characters. *ACM TOG* 39, 4 (2020).

Zhan Xu, Yang Zhou, Evangelos Kalogerakis, and Karan Singh. 2019b. Predicting Animation Skeletons for 3D Articulated Models via Volumetric Nets. In *Proc. 3DV*.

Li Yi, Haibin Huang, Difan Liu, Evangelos Kalogerakis, Hao Su, and Leonidas Guibas. 2018. Deep Part Induction from Articulated Object Pairs. *ACM TOG* 37, 6 (2018).

Quanshi Zhang, Xuan Song, Xiaowei Shao, Ryosuke Shibasaki, and Huijing Zhao. 2013. Unsupervised Skeleton Extraction and Motion Capture from 3D Deformable Matching. *Neurocomputing* 100 (2013).