Synthesizing Mesh Deformation Sequences with Bidirectional LSTM

Yi-Ling Qiao, Yu-Kun Lai, Hongbo Fu, and Lin Gao*

Abstract—Synthesizing realistic 3D mesh deformation sequences is a challenging but important task in computer animation. To achieve this, researchers have long been focusing on shape analysis to develop new interpolation and extrapolation techniques. However, such techniques have limited learning capabilities and therefore often produce unrealistic deformation. Although there are already networks defined on individual meshes, deep architectures that operate directly on mesh sequences with temporal information remain unexplored due to the following major barriers: irregular mesh connectivity, rich temporal information, and varied deformation. To address these issues, we utilize convolutional neural networks defined on triangular meshes along with a shape deformation representation to extract useful features, followed by long short-term memory (LSTM) that iteratively processes the features. To fully respect the bidirectional nature of actions, we propose a new share-weight bidirectional scheme to better synthesize deformations. An extensive evaluation shows that our approach outperforms existing methods in sequence generation, both qualitatively and quantitatively.

Index Terms—Mesh deformation, mesh sequences, LSTM, deep learning, shape generation

1 Introduction

In recent years, many techniques [1], [2], [3] have been developed to capture 3D shape animations, which are often represented as sequences of triangular meshes with detailed geometry. Analyzing such animation sequences for synthesizing new realistic 3D mesh sequences is very useful in practice for the film and game industries. Although deep learning has achieved significant success in synthesizing a variety of media types, directly utilizing the temporal information for synthesizing mesh deformation sequences by deep learning methods remains unexplored.

A major challenge to achieve this is to go beyond individual meshes and understand the *temporal* relationships among them. There are several works like [4], [5], which can generate mesh sequences but their generation is based on interpolation or extrapolation in the feature space without using temporal information. Since the feature space is not infinite, such methods cannot extrapolate long sequences or capture temporal properties like periodicity; see Fig. 5 for comparisons that demonstrate the differences between extrapolation-based methods and ours. Skeleton constraints [6], [7] can also help with the generation of meshes. Dyna [8] analyzes the dynamics of human shapes. It operates on a skeleton representation and thus is not applicable to general models such as flags and dresses. Alternatively, [9] and [10] study the relationships among all models in an *unordered* dataset by performing clustering and shape analysis on the *whole* datasets, which do not take sequences into account. Thanks to the development of deep learning methods, in particular, the recurrent neural network (RNN) and its variants such as long short-term memory (LSTM) [11] and gated recurrent unit (GRU) [12], one can more easily learn from sequences. Based on RNNs, impressive results have been achieved in tasks including movie prediction [13], [14], music composition [15], text generation [16] and completion [17]. Compared to the standard RNN, LSTM can mitigate the problem of vanishing gradients and thus better deal with long-range dependencies [18]. This motivates us to use LSTM to learn to synthesize mesh deformation sequences.

However, unlike audio or text, mesh frames cannot serve as direct input to LSTM. The input to LSTM cells is usually well-organized low-dimensional vectors, but raw mesh representations are unordered vertices and faces in high dimensions. Simply feeding a mesh into the LSTM would lead to huge time and memory cost.

To make the mesh data tractable for LSTM, we apply CNNs for feature extraction and dimensionality reduction before putting the mesh data into LSTM cells. Recent efforts have been made for lifting 2D Convolutional Neural Networks (CNNs) to 3D data [19], including multi-view [20] or 3D voxel [21], [22] representations. Those networks, however, are not suitable for high-resolution mesh data. In this paper, we utilize mesh-based CNNs [23] working on a shape deformation representation [24]. Share-weight convolutional kernels introduced by [23] can significantly reduce the number of network parameters, allowing meshes with many vertices to be handled. Meanwhile, the deformation representation works well with large-scale deformation,

* *Corresponding author is Lin Gao.*

- *Y.-L Qiao is with the Beijing Key Laboratory of Mobile Computing and Pervasive Device, Institute of Computing Technology, Chinese Academy of Sciences and also with with Department of Computer Science, University of Maryland, College Park, US. E-mail: yilingq97@gmail.com*
- *L. Gao is with the Beijing Key Laboratory of Mobile Computing and Pervasive Device, Institute of Computing Technology, Chinese Academy of Sciences and also with the University of Chinese Academy of Sciences, Beijing, China. E-mail: gaolin@ict.ac.cn*
- *Y.-K Lai is with the School of Computer Science and Informatics, Cardiff University, Wales, UK. E-mail: LaiY4@cardiff.ac.uk*
- *H. Fu is with the School of Creative Media, the City University of Hong Kong. E-mail: hongbofu@cityu.edu.hk*

suitable for describing rich deformation information in an animation sequence. We construct an auto-encoder with a mesh-based CNN to map a mesh to a low-dimensional latent vector, suitable to be processed by LSTM.

Note that mesh sequences have a very different property from audio or text, where the motion of mesh sequences is usually bi-directional, i.e. both the forward and backward actions can be reasonably present in an animation. Therefore, we design a share-weight bi-directional LSTM to learn the two directions simultaneously. In some sense, this design doubles the number of training samples and thus stabilizes training. Moreover, since the learned LSTM can generate motion in both directions, sequence completion becomes easier. Based on the assumption that both directions of the motion should be reasonable, our method can be used on reversible motions. On the other hand, we admit that some sequences might challenge our assumption if their processes along the time axis are not reversible. For example, motions like a bouncing ball with significant damping might not be suitable for our model. Our method is largely limited to processes dominated by conservative forces, like many humans' or animals' actions. To apply our method, we also expect similarities between the training and test data, as other learning based methods. We will later demonstrate the effectiveness of our novel designs.

In summary, to analyze 3D mesh deformation sequences, we propose a novel bidirectional LSTM architecture combined with mesh-based convolutions. This pipeline is able to deal with complex mesh sequence data and flexible for multiple applications. Our main contributions are:

1) We propose a deep learning method to cope with mesh deformation sequences by exploiting the underlying temporal information. Our method is able to generate sequences with realism and diversity conditioned on different initial frames.
2) We design a share-weight bidirectional LSTM specifically for learning from deformation sequences. Our experimental results show that bidirectional generation helps to complete a sequence in a more natural way.

Next, we first review relevant work and then present our feature representation, network architecture, and loss functions. In the Experiments section, we show extensive experimental results to justify our design. Since no existing work is designed to synthesize/complete mesh animation sequences, we compare our work with the state-of-the-art works that can be extended or applied to this problem, both qualitatively and quantitatively. Finally, we draw conclusions from our work.

2 Related Work

RNN and LSTM. The recurrent neural network and its variants, such as LSTM [11] and GRU [12], have been widely used in dealing with sequential data, including text [25], [26], video [13], [14] and audio [27], [28]. [29] learns representations of video by LSTM in an unsupervised manner. But none of the above methods have attempted to analyze mesh sequences. Researchers have utilized various methods for a better quality of generation. For example, PredNet [30] learns to predict future frames by comparing errors between prediction and observation, similar to the loss adopted in our network. [31] incorporates policy gradients into generative adversarial networks (GANs) [32] and LSTM to generate sequences. Attempts have also been made to predict video frames using CNNs [33]. To avoid predicting videos directly in the high-dimensional pixel space, some work uses high-level abstraction such as human poses [34], [35] to assist with generation. This motivated us to use mesh-based CNNs for extracting low-dimensional features, which are then fed into LSTM.

Many variants of RNNs have been developed to predict or generate realistic motion sequences. For example, [36] proposes an encoder-recurrent-decoder (ERD) to learn spatial embeddings and temporal sequences of videos and motion capture data. [37] generates image sequences with a sequential variational auto-encoder, where two RNN chains are used to encode and decode sampled sequences accordingly. However, such approaches that iteratively take the output as input to the next stage could cause error accumulation and make the sequence freeze or diverge. To address this problem, [38] presents Auto-Conditioned RNNs (acRNNs), whose inputs are previous output frames interleaved with ground truth. With ground truth frames at the beginning of a sequence, acRNN can generate output sequences conditioned on given input sequences. Our approach adopts similar spatial encoding blocks. But unlike their LSTMs, which have a single direction only, our bidirectional LSTM allows easier generation of mesh sequences.

[39] builds a sequence-to-sequence architecture, which is able to predict multiple actions, but the method does not have spatial encoding modules. Using an encoder-decoder structure, [40] extracts feature representations of human motion for prediction and classification. Wang et al. [41] propose to learn a latent space of garment meshes for interactive animation synthesis. Their method focuses on generating clothing meshes based on given body motions, while our method exploits the temporal information and can generate mesh sequences for general shape types. The spatial-temporal relationship is exploited in [42] to generate human motion skeleton sequences. [35] uses a GAN and LSTM to generate actions or complete sequences by optimizing the input vector of the GAN. Compared to their tasks, our goal focuses on general mesh sequences, which have higher dimensions and are more general than skeleton data. More research works on the 3D shape generation could be referred to [43];

3D Shape Interpolation. Compared with image generation, synthesizing 3D shapes is more challenging due to the high dimensionality and irregular connectivity of mesh data. Previous works mostly generate 3D shapes via interpolation in parameterized representations. For example, [44] proposes to interpolate shapes in a Riemannian shell space. Based on existing shapes, data-driven methods (e.g. [45]) can generate realistic samples. However, such traditional methods focusing on shape representations and shape analysis have limited learning capabilities. More recently, [4] proposes to use Variational Autoencoders (VAEs) to map mesh models into a latent space and generate new models by decoding latent vectors. Locally deformed shapes can also be generated by a combination of deep learning and

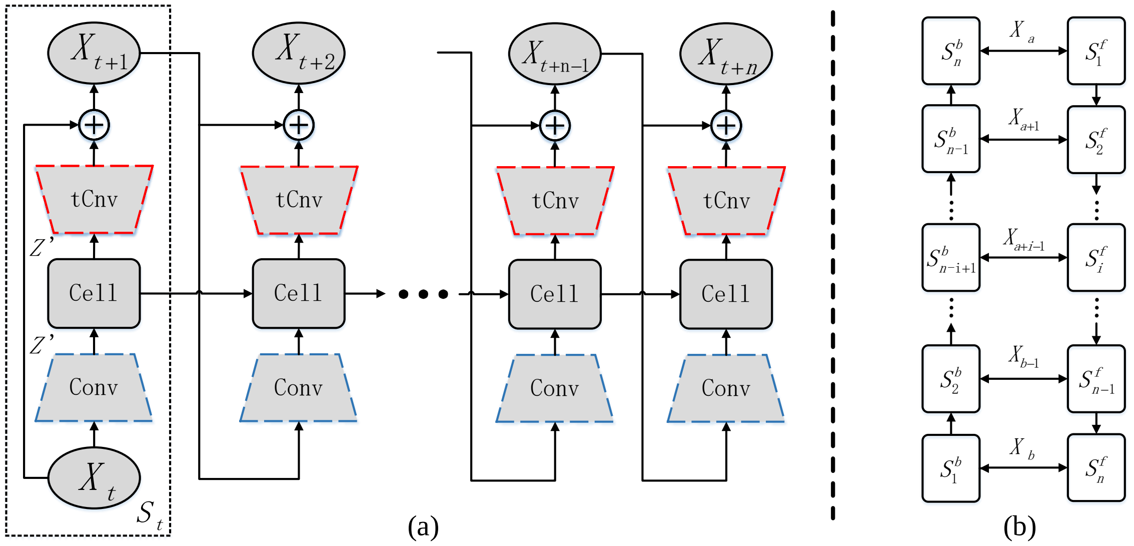


Fig. 1. Architecture of our network. (a) shows that our network is composed of LSTM module *cell* and mesh convolution modules $Conv, tCnv$. Taking the network S_t at time step t as an example, the input to $Conv$ is the deformation representation X_t. $tCnv$ is a stack of transpose convolution layers mirroring $Conv$. The output of $tCnv$ is the feature change δX_t. $\delta X_t + X_t$ gives the predicted feature for time step $t+1$, and is fed into S_{t+1} iteratively. (b) is our bidirectional LSTM. Both chains have the same architecture as in (a), and the only difference is their opposite directions. The forward chain takes the first model as input and the backward chain takes the last. They share weights and their predictions are constrained to match each other.

sparse regularization [46]. While these learning-based methods can produce new shapes that are diverse and realistic, the temporal information of mesh animation sequences has not been fully explored, which we will address in this paper.

3 Methodology

3.1 Mesh Sequence Representation

Mesh animation sequences are typically represented as a set of meshes with the same vertex connectivity but different vertex positions. Such meshes can be obtained by consistent remeshing or template fitting and become very common nowadays due to the improved scanning and modeling techniques. These animated mesh sequences usually involve large-scale and complex deformations.

In this work, we represent shapes using the shape deformation representation [24], a state-of-the-art representation that works well for large-scale deformation and suitable for deep learning methods. Compared to the plain mesh representation, this representation is rotation-invariant and keeps the local structure of the manifold. Assume the motion sequence dataset M contains n meshes and each mesh is denoted as m_t $(t = 1, 2, \dots, n)$. We denote $\boldsymbol{p}_{t,i} \in \mathbb{R}^3$ as the i^{th} vertex of the t^{th} model. $\boldsymbol{D}_{t,i}$ represents the deformation gradient defined in each 1-ring vertex neighborhood, and is computed as

$$\boldsymbol{D}_{t,i} = \arg\min_{\boldsymbol{D}_{t,i}} \sum_{j \in N_i} c_{ij} \left\| (\boldsymbol{p}_{t,i} - \boldsymbol{p}_{t,j}) - \boldsymbol{D}_{t,i}(\boldsymbol{p}_{1,i} - \boldsymbol{p}_{1,j}) \right\|_2^2 , \tag{1}$$

where N_i is the 1-ring neighborhood of the i^{th} vertex, and c_{ij} is the cotangent weight to avoid discretization bias [47]. Note that Equation 1 computes the deformation gradients w.r.t. the first mesh, which is treated as the template of the motion sequence. During reconstruction, we recover each mesh by deforming the first mesh using its deformation gradients. The deformation gradient matrix $\boldsymbol{D}_{t,i}$ is decomposed into a rotation matrix $\boldsymbol{R}_{t,i}$ and scaling matrix $\boldsymbol{S}_{t,i}$, i.e., $\boldsymbol{D}_{t,i} = \boldsymbol{R}_{t,i}\boldsymbol{S}_{t,i}$. The difficulty for representing large-scale deformations is that the same rotation matrix $\boldsymbol{R}_{t,i}$ can be mapped to two rotation axes with opposite directions, and the associated rotation angles might include different numbers of cycles. To solve this rotation ambiguity problem, a global integer programming based method [24] is applied to obtain as-consistent-as-possible (ACAP) assignment of angles and rotation axes. After removing trivial entries such as zeros in the rotation and scale components at the i^{th} vertex of the t^{th} model, the algorithm outputs feature vectors $\boldsymbol{q}_{t,i} \in \mathbb{R}^9$ that represent local deformations. The mesh representation X_t is eventually produced by linearly normalizing each dimension of $\boldsymbol{q}_{t,i}$ into $[-0.95, 0.95]$ to make it suitable for applying the *tanh* activation function, following [4]. The reverse linear mapping needs to be applied to synthesized feature vectors, before their use for shape reconstruction [24].

3.2 Generative Model

The overall architecture of our network is illustrated in Fig. 1. In this illustration, we denote LSTM cells as *cell*. $Conv$ refers to the mesh convolutional operations [23], [46] and $tCnv$ represents transpose convolutions. For each convolutional filter, the output at a vertex is computed by a weighted sum of its 1-ring neighbors along with a bias:

$$y_i = W_1 x_i + W_2 \frac{\sum_{j=1}^{d_i} x_{n_{ij}}}{d_i} + b, \tag{2}$$

where x_i and y_i are input and output at the i^{th} vertex, W_1, W_2 and b are the filter's weights and bias, d_i is the degree

TABLE 1
Comparison between two variants of our method and previous works on per-vertex position error (average distance between vertex positions of ground truth and prediction on the n^{th} frame). On four sequences from the Dyna dataset [8], we compare our method variants with [4] and [24]. The existing methods generate future mesh sequences by extrapolating in the feature space given two initial models. Also, we feed our network with one or three initial frames (IFs) and observe that more initial frames generally improve the performance. On average, our method outperforms extrapolation-based methods, which generate reasonable results in a short period but suffer from error accumulation later.

Methods	Punching			ShakeArm			Handstand			Horse		
	5th	10th	15th	5th	10th	15th	5th	10th	15th	5th	10th	15th
Ours+1 IF ($\times 10^{-4}$)	175	156	285	335	381	319	323	527	516	603	869	1328
Ours+3 IF ($\times 10^{-4}$)	**95**	**84**	**107**	301	**226**	290	212	**379**	**428**	451	**329**	**671**
[4] ($\times 10^{-4}$)	132	240	457	**291**	433	688	**93**	489	797	**286**	713	1032
[24] ($\times 10^{-4}$)	294	361	413	391	472	**110**	487	401	1589	334	1051	1568

TABLE 2
Average vertex position errors ($\times 10^{-4}$) between ground truth and prediction on Punching in Dyna [8] with different network architectures. We can see that our share-weight BD-LSTM outperforms standard BD-LSTM and unidirectional LSTM. We also observe a decrease in accuracy when the L_{KL} term is omitted.

Ours	Std. BD-LSTM	Unidir. LSTM	w/o L_{KL}
88	123	114	103

of the i^{th} vertex, and n_{ij} is the j^{th} 1-ring neighbor of the i^{th} vertex.

Given the LSTM state s_t and model X_t, we first describe how to generate the next model X_{t+1}. First, X_t is put into the mesh convolutional sub-network $Conv$, which outputs a low-dimensional latent vector $z = Conv(X)$. After that, z is sent to LSTM $cell$, outputting $(z', s_{t+1}) = cell(z, s_t)$, where s_{t+1} represents the updated state and z' is the updated latent vector. z' is then passed to transpose mesh convolution $tCnv$. Similar to many sequence generation algorithms, the output of $tCnv(z')$ is the difference between the next and current models $\delta X_t = X_{t+1} - X_t$, instead of X_{t+1} itself, to alleviate error accumulation. In the end, the generated model from X_t is simply worked out as $X_{t+1} = X_t + \delta X_t$. Consecutive models are generated iteratively in the same way. Fig. 1 (a) shows the structure of an LSTM chain, and (b) illustrates the whole process of bidirectional training. Suppose that we already have a set of models $S_{i,j} = \{X_i, X_{i+1}, ..., X_j\}, (i \leq j)$. To extend the sequence, we would like to predict its n future models $S_{j+1,j+n|i,j} = \{X_{j+1}, X_{j+2}, ..., X_{j+n} | X_i, X_{i+1}, ..., X_j\}$. Our method first puts the existing models into the network in their order, and let the LSTM cell update its state to s_j from an initial state s_0. When it comes to the j^{th} model, the network outputs X_{j+1}, which is afterwards treated as the $(j+1)^{\mathrm{st}}$ input, and this process repeats for n times, leading to the follow-up sequence $S_{j+1,j+n|i,j}$.

3.3 Bidirectional Generation

Sequence generation is a challenging problem in various data forms, not only for the potentially tricky ways to exploit temporal information but also about how to obtain enough training data. When the data is scarce for a specific application, which is often the case for 3D model datasets, training can be problematic.

However, unlike text, movie, and audio, 3D model sequences can be more flexible. On the one hand, the order of 3D shape sequences is less strict, i.e., the inverse of motion is often reasonable. On the other hand, there are usually multiple plausible paths between two meshes. Based on these two observations, we propose a bidirectional generation scheme, which allows generating more possible deformation sequences than the dataset has and avoids restricting results to specific deformation paths, as illustrated in Fig. 1. From a 3D model deformation sequence, we arbitrarily choose two models X_a and X_b as endpoints of two inverse n-length sequences S^f, S^b such that $S_1^f = S_n^b = X_a$ and $S_n^f = S_1^b = X_b$. Let X_a and X_b have opposite initial states s_0^f and s_0^b, respectively. We expect them to generate similar models, satisfying $\forall 1 \leq i \leq n, S_i^f \approx S_{n+1-i}^b$. In summary, there are weight-sharing structures at different levels. Take the network diagram in Fig. 1 as an example. First, inside $Conv$ and $tCnv$, there are convolutional kernels that share weights among vertices. Second, the weights for the encoder-decoder structure and the LSTM $cell$ for each time step are shared. Third, unlike standard bi-directional LSTMs (Std. BD-LSTM in Table 2), our backward and forward chains share weights. Weight-sharing between the forward and backward chains halves the trainable parameters, which mitigate overfitting.

3.4 Loss Function

Our loss function is composed of three terms as follows

$$L = L_{recon} + \alpha_1 L_{bidir} + \alpha_2 L_{reg}. \tag{3}$$

To illustrate this, let the ground truth models be $\{X_1, X_2, ..., X_n\}$, which are expected to be the results of forward sequence S^f and backward sequence S^b. The reconstruction loss

$$L_{recon} = \sum_{i=1}^{n} (||S_i^f - X_i|| + ||S_i^b - X_{n+1-i}||) \tag{4}$$

forces both sequences to resemble samples from the dataset. Meanwhile, as described before, bidirectional sequences S^f and S^b share weights and have similar outputs, which is ensured by the bidirectional loss

$$L_{bidir} = \sum_{i=1}^{n} \left\| S_i^f - S_{n+1-i}^b \right\|. \tag{5}$$

Furthermore, $L_{reg} = L_{KL} + L_2$ contains the KL divergence term and the L_2 penalty on network weights to regularize the network. Assume that $\{w_i\}$ is the set of trainable parameters in the neural network. The L_2 penalty is computed by $L_2 = \sum w_i^2$. The KL divergence helps find a good

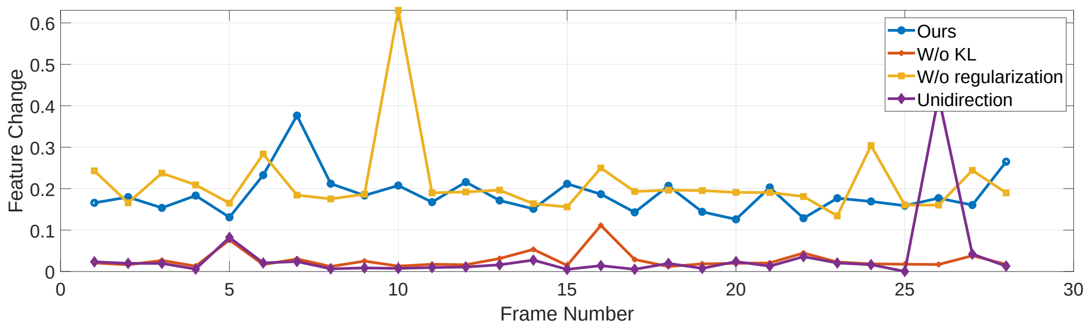


Fig. 2. Shape feature changes between subsequent frames of different methods. This line graph depicts the amount of feature changes $mean(||X_t - X_{t+1}||/X_t)$ between consecutive frames. The proposed method (blue) has stable and visible changes. Networks without KL loss or bidirectional-training suffer from frozen sequences, and the one without L_2 regularization has significant jerk.

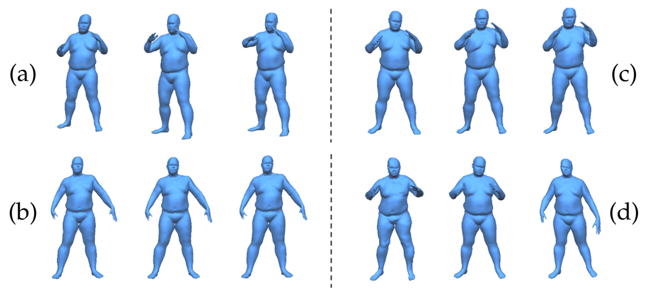


Fig. 3. Visualization of the results in the ablation study corresponding to the line graph in Fig. 2. (a) is generated by our method. (b) is from the network trained without L_{KL}, which nearly freezes at one frame. (c) is generated from a standard unidirectional LSTM, and encounters a problem similar to (b). (d) is generated from the network trained without L_2 regularization. The models do move but have sudden changes. In summary, our method generates smooth and lively animation, indicating that all the loss terms contribute to good performance.

TABLE 3
The numbers of frames and training/test sequences for all datasets: Dyna (Figs. 3, 4, 7, 9, and 11), Dress and Paper (Fig. 4), Handstand (Fig. 5), Horse (Fig. 5), Camel (Fig. 6), Pant (Fig. 8), Finger (Fig. 8), and Crane (Fig. 12).

Datasets	Dyna	Dress	Handstand	Horse	Camel
Frames	2610	601	523	574	574
Train/test seq.	1840/398	418/89	356/73	397/83	397/83
Datasets	Pant	Paper	Finger	Crane	SMPL
Frames	4009	873	571	522	501
Train/test seq.	3145/770	636/143	83/394	73/355	N/A

latent feature for the high dimensional representation [48]. We expect that the encoded latent vector z conforms to a Gaussian distribution, such that the feature is normalized and variables in this vector are decoupled. Finding such a feature embedding, in the sense of maximum likelihood estimation, is equivalent to minimizing reconstruction loss and KL divergence with a Gaussian distribution [48]. Therefore, we have $L_{KL} = D_{KL}(q(z|X)|p(z))$ where $q(z|X)$ is the posterior distribution and $p(z)$ is the Gaussian prior distribution. Since the KL constraint enforces the feature learned to have a similar distribution to Gaussian, the space for optimization can be simpler. The encoder-decoder structure and the KL loss are used to reduce dimensions and better learn the distribution of shapes. We wish the network to learn a low-dimensional random distribution such that the decoder can generate reasonable shapes even from the probably noisy and uncertain LSTM output. As we show in Table 2, training with the KL loss helps to get a lower overall loss. Moreover, we find that the KL term can stabilize the training process by mitigating sudden jerks. Without this, the manifold of the embedded feature may contain steep regions, where small changes in the latent space could map to big differences in the generated model (see Fig. 2). In all our experiments, we set $\alpha_1 = 0.5$ and $\alpha_2 = 0.1$.

4 Experiments

4.1 Implementation Details

We use Tensorflow as the framework of our implementation. Experiments are performed on a PC with an Intel Core i7-2600 CPU and an NVIDIA Tesla K40c GPU. We use Adam optimizer [49] to update weights, with default Adam parameters $\beta_1 = 0.9$, and $\beta_2 = 0.999$ as in [49]. For each dataset, we randomly exclude a subsequence, which takes up 20% of the dataset, as a test set, and use the remaining frames as the training data. A training process takes 7,000 iterations, lasting for around 8 hours. We list the numbers of frames and training/test sequences for all the datasets in Table 3.

In each iteration, we generate 8 sequences, each of them containing a deforming mesh at 32 frames. For the datasets where motion is slow [8], we sample every other model in sequences, by doing so the difference between adjacent frames can be more remarkable. For all experiments, the LSTM *cell* has 3 layers and 128 hidden dimensions, and we set initial states as $s_0^f = -s_0^b = [0.1]^{128}$. The mesh convolution module $Conv$ is composed of 3 layers with $tanh$ as the activation function.

4.2 Framework Evaluation

Bidirectional Generation. Our proposed share-weight bidirectional LSTM (BD-LSTM for short) better utilizes temporal information to facilitate sequence completion. As we will later show, it leads to results with greater diversity. In addition, the L_{bidir} and L_{KL} terms impose stronger constraints during training and consequently help predict more accurate sequences. According to the numerical results

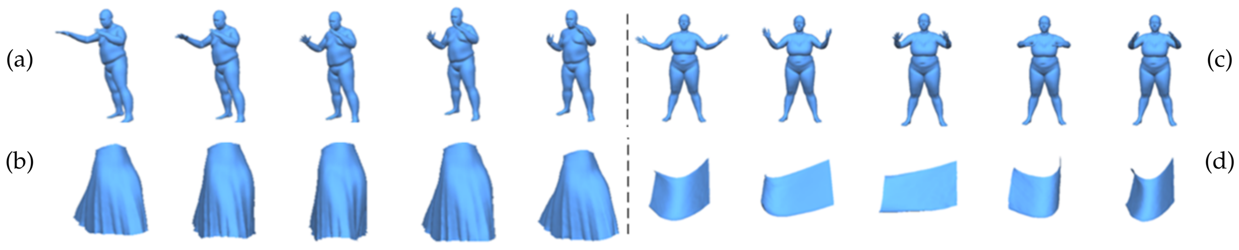


Fig. 4. Qualitative results of our method on (a,c) Dyna [8], (b) dress, and (d) paper. The first model in each sequence is fed as input, and the network automatically generates the following four meshes. This is the first approach that is able to generate an animation sequence from a single initial mesh. Our method is applicable to general types of models that cannot be represented by skeletons, such as the dress and paper sequences.

in Table 1, our method is more effective than existing methods [4], [24], which are state-of-the-art methods for mesh interpolation and extrapolation, though they are not originally designed for mesh sequence generation or completion.

Ablation Study of Loss Terms. Error accumulation is a common problem in sequence generation tasks [37], [38], [39]. Generated meshes often freeze because results tend to stay at an average shape, or even diverge to random results. To address this problem, we use three mechanisms: 1) L_{KL} divergence to regularize the internal distribution; 2) L_2 regularization loss to mitigate overfitting; and 3) bidirectional generation to impose an additional constraint. To verify the impact of these terms, we train models with one of the three terms turned off. For unidirectional sequences, we use only one direction of LSTM. The line graph in Fig. 2 shows representation changes $mean(||X_t - X_{t+1}||/X_t)$ between adjacent frames X_t and X_{t+1}. The four networks are trained on Dyna [8] for 7,000 iterations, and tested on 32 randomly chosen sequences. It can be seen from the results that without KL or BD-LSTM, the generated sequences tend to freeze. Meanwhile, the use of L_2 regularization helps reduce jerk. In Fig. 3, we show sampled frames of the generated sequences corresponding to the lines in Fig. 2. We can easily observe the freezing frames in (b) and (c) as well as sudden changes in (d). Please refer to the accompanying video, which provides more visual comparisons of generated sequences.

We also perform a quantitative comparison of the average per-vertex error between prediction and ground truth on the Dyna dataset [8]. As shown in Table 2, our BD-LSTM leads to the lowest average per-vertex error among all the compared methods.

Number of Initial Frames. Generating a sequence based on initial frames is a useful but challenging application. In theory, the more bootstrap frames the network has, the more knowledge it obtains about the sequence, therefore making a more accurate prediction. Previous mesh generation approaches, however, are based on extrapolation, which require the availability of at least two existing models (endpoints). Our method can take advantage of all input frames by feeding them into the LSTM. A previous human motion prediction method uses $u = 4$ frames to start the recurrent network [38]. We test $u = 1$ and $u = 3$ in Table 1 (Ours+1 IF and Ours+3 IF), which shows that more initial frames can reduce the distance between prediction and ground truth.

4.3 Sequence Generation

We now evaluate the sequence generation capability of the proposed method. Starting from some initial frames, sequence generation predicts future frames. In the first part of this section, we show both qualitative and quantitative results of the generating models. In the second part, we further show that a single network can generate meshes in different shapes and motions, only conditioned on the initial frames.

Generating Sequences. As far as we are aware, this is the first work to learn and generate long mesh sequences given one or multiple initial meshes. Given two initial frames, previous methods like [46] often use extrapolation to generate new following meshes. However, simply extrapolating meshes from a limited number of input frames easily fails to capture long-term temporal information, e.g., the periodicity of a sequence. In contrast, since the transformations between frames of LSTM are nonlinear, LSTM has a better ability to generate long sequences. Previous NLP research [25] has also shown LSTM can generate words and sentences endlessly. With the help of LSTM, our model can model the history information and generate realistic mesh sequences iteratively, even if the number of models in the dataset is very limited. In our experiment, we feed the first test mesh model to our trained framework and let it generate the subsequent frames. Some qualitative and quantitative results are shown respectively in Fig. 4 and Table 1. We compare our model with ground truth as well as previous extrapolation-based methods [4], [24]. Fig. 5 plots the predictions on the 5^{th}, 10^{th} and 15^{th} future frames. We can see that both extrapolation methods fail on the 15^{th} frame because linearly extending the motion path eventually exceeds the plausible deformation space. In contrast, our method is aware of the periodicity of the sequence and able to return back once reaching the extreme point, producing natural motion cycles. More examples can be found in the supplementary video.

In addition, in Figs. 6 and 8 we show long sequences generated by our method. Since in our implementation the network is constructed to have a fixed length of 32 frames, we run the LSTM for multiple times, with the initial frames of a run set to the last frames of the previous run, except for the first run. In the end, all the sub-sequences are combined together to form a long sequence. With only one given initial model, the camel still moves realistically even after 220 frames. We also generate 300 frames of a walking finger and moving pants in Fig. 8. We will also show in Sec. 4.4 that

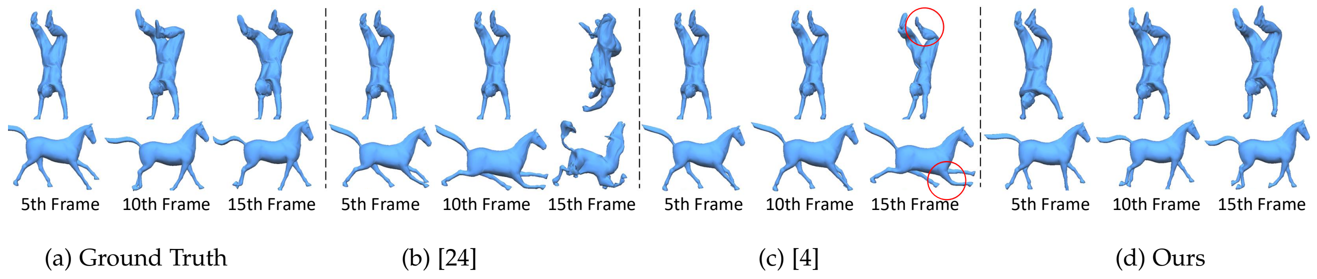


Fig. 5. Comparison with other works on sequence generation. In this experiment, two consecutive frames are sent into the network, and we aim to predict the future $5^{\text{th}}, 10^{\text{th}}, 15^{\text{th}}$ meshes. (a) shows the ground truth of the meshes; (b) is obtained by using linear extrapolation in the feature space [24]; (c) is the result by extrapolation on a feature from deep learning [4]; (d) is our result. We can see that extrapolation-based generation fails when predicting frames further away. (b) totally fails on the 15^{th} frame, and (c) also produces abnormal deformation, as highlighted in the red circles. In contrast, our method forms a natural cycle and avoids exceeding the limits (following the horse's stride).

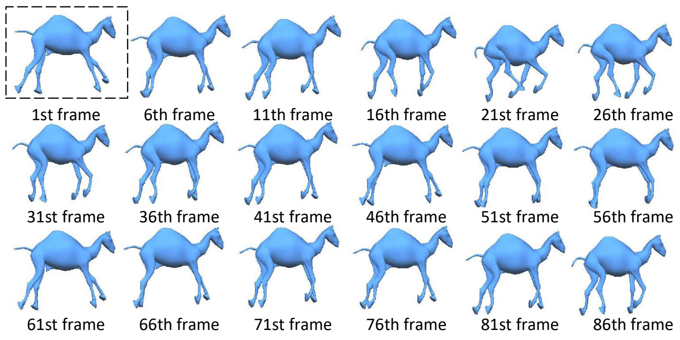


Fig. 6. Sampled frames of a long sequence with 220 deforming meshes generated by our network from a single initial frame (i.e., the first frame, highlighted in a black box). The camel steps forward periodically in this animation sequence.

a small set of keyframes can be given as input to flexibly control the generation of long sequences.

To test how our method can generalize to new data, we feed the network with a single initial frame and test its performance with random initial states. Figure 10 shows that our method can generate diversified sequences from a same initial frame and random initial states. Moreover, we train and test on completely disjoint sequences as shown in Fig. 7. On the left side, we train our network on (a) chicken wings from the Dyna [8] dataset, and test on (b) jumping jack. Our method generates a sequence in (c), which is different from both (a) and (b). On the right side, we use the sequences from SMPL [50]. We train on (d) a dancing man while test on (e) a jumping man. Our method generates (f) a novel motion but more similar to dancing poses. These examples show that the generated sequences do not always match the motions in the training, yet they can still look plausible. This also indicates that a single initial frame is not enough to fully specify the whole animation, and our aim is to produce plausible animations. In fact, the results are different from both training and test data, demonstrating that our method does not simply memorize motions from the training set. Nevertheless, our model can generate reasonable motions because the training and test data still share some similarities since they are both human motions.

Conditional Generation. Another promising application of our method is to generate sequences of various meshes conditioned not only on the action but also other characteristics like shape. Previous approaches achieve conditional human motion generation on videos [29] and skeletons [35], but not on 3D shape sequences. To illustrate the effectiveness of our method, we take Dyna [8] as an example. In this collection of datasets, there are female/male models of different subjects and actions. All meshes in different datasets have the same number of vertices and share connectivity, so we train our model on a mixture of those datasets. Unlike previous conditional generation methods, which take condition labels as input, the information of conditions in our method is implicitly contained in initial frames. To test this, we feed $u = 2$ bootstrap models with a certain body mass index (BMI)/gender/motion as input and obtain the following $n = 16$ frames as output. Our method does *not* require a label to be specified and automatically generates meshes consistent with the input frames. We show our results in Fig. 9. It can be observed that our method can generate human meshes of different subjects and genders. Furthermore, even if the first frames of different inputs are the same, the network can produce different action sequences, according to the second frame.

4.4 Sequence Completion

We now consider another important application, namely sequence completion, which produces in-between meshes given two endpoint frames.

Completion Given Keyframes. Completing a mesh sequence based on given anchors is an important application in animation. In our approach, we clip the target sequence by keyframes. For each segment, we run our bidirectional network by treating two keyframes as endpoints. We set a threshold (e.g. 5%), and if the difference is lower than the threshold between two models in the forward and backward sequences, we merge the two sequences at this pair. We try multiple runs until it can be merged together. Since the computation is identical for each segment, for illustration we show an example of completing one segment constrained on two keyframes. Fig. 11 shows an example on the Dyna dataset [8]. It can be seen from (f), (g) and (h) that when the source and target models are highly similar, interpolation-based approaches lead to results with almost identical meshes since they generate meshes along the shortest path between the source and target models (or their representations in feature space).

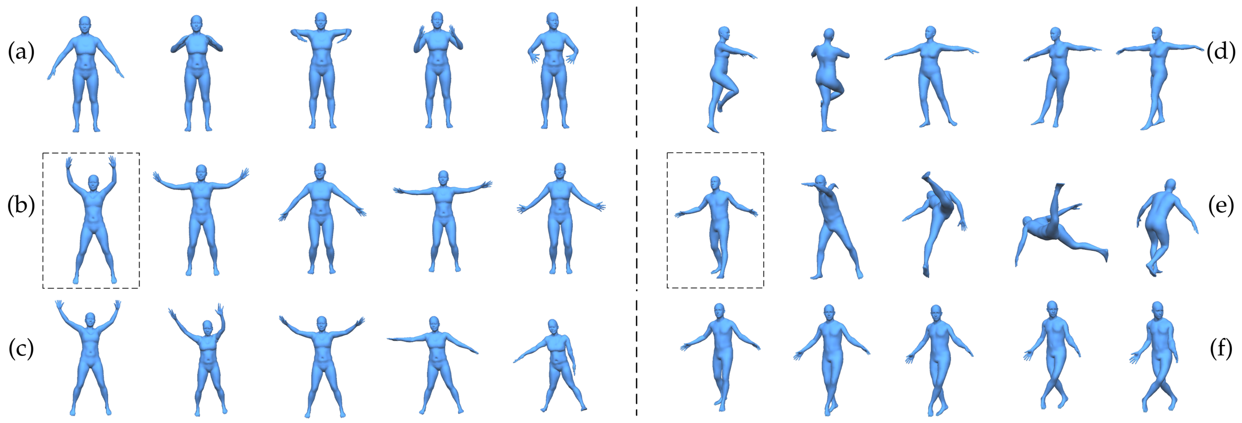


Fig. 7. Test on novel data. We train and test our network on disjoint sequences. The left side is on the Dyna [8] dataset. The network is first trained on (a) chicken wings and then tested on jumping jack. (b) are the samples of original meshes in the test dataset. (c) is the generated results given 3 frames from the test dataset. The first input mesh is highlighted in a black box in (b) and the consecutive second and third frames of input are similar to the first frame with slight changes. The right side is on SMPL animation sequences [50]. We train on (d) a dancing sequence and test on (e) a jumping man. Results (f) are generated by our method given 3 frames from the test dataset. The figure inside the black box in (e) is the first input mesh. (f) are novel poses of a man crossing his legs. The two examples indicate that our method can generate sequences on unseen data and synthesize motions that are different from both training and testing data.

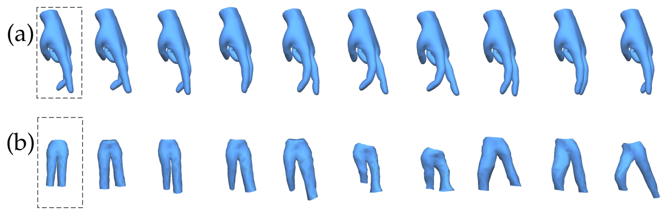


Fig. 8. Two examples of long sequence generation (for 300 frames): (a) walking fingers and (b) moving pants. Inside the black boxes are the inputs.

With the help of bidirectional LTSM, our method can generate animation given keyframes. This allows us to generate a long sequence easily, with flexible control of animation. Fig. 12 shows two examples of such results. Here, BD-LSTM performs sequence completion for gaps between keyframes. Bidirectional generation enables our network to accomplish sequence completion and generation controlled by keyframes.

We also perform a quantitative experiment of sequence completion using a similar setting as in Fig. 11 with meshes from Dyna f_50004_jumpingJacks dataset, and the length of the sequence to be completed is set as 9 frames. We run the experiment on 4 sequences, and compare our method with the state-of-the-art interpolation methods [4], [45]. Our method achieves the best performance with error 69.2% lower than the state-of-the-art methods, in terms of the average per-vertex position error between completion results and ground truth (0.868 for [45], 0.775 for [4] and 0.239 for ours). This is because our method can learn motion patterns (in this case cyclic) rather than simply interpolating between given endpoints.

Diversified Sequence Completion. Previous interpolation-based methods usually adopt a deterministic strategy to complete sequences and thus result in a fixed sequence. Our work, however, is able to produce diversified sequence completion results. We run multiple trials with randomized initial states until the sequences from both sides can merge together at a certain frame (e.g., the difference between two frames is less that 5%). Since random trials can produce diversified results, the network is able to generate different sequences as shown in Figures 11 (b-d). In the real world, there is often more than one possible motion between two static poses and our method can therefore better describe such characteristics than other generation methods.

Alternative Strategy. To test alternative completion strategies, we also implement an optimization+unidirectional strategy [35] previously used for human action completion. Given the source model X_0 and target model X_n, we first find the optimal LSTM initial state $\hat{s}_0 = \arg\min_{s_0} ||\hat{X}_n - X_n||$, where $\hat{X}_*$ is the vector representing a synthesized mesh, and

$$\begin{cases} (\hat{X}_i, s_i) = G(X_0, s_0) & i = 1 \\ (\hat{X}_{i+1}, s_{i+1}) = G(\hat{X}_i, s_i) & i > 1, \end{cases} \tag{6}$$

where the LSTM forward iteration is denoted as $(X_{t+1}, s_{t+1}) = G(X_t, s_t)$. After solving the optimization problem with [51], we then compute $\{\hat{X}_t\}$ through Eq. 6. Fig. 11 (e) shows a completion result with the source and target models set to the same as the compared methods. Compared to interpolation-based strategies, the optimization+unidirectional strategy can achieve more realistic morphing, but it only finds one optimal initial state, not providing diverse possible choices as our BD-approach.

5 Conclusion

In this paper, we presented the first deep architecture to generate mesh animation sequences. Our technique can not only predict future frames given initial frames, but also complete mesh sequences based on keyframes and generate sequences conditioned on given meshes.

Fig. 13 shows two failure cases of our method. We use the same experimental setting as in Fig 7. For (a), the network is trained on a dancing motion sequence from SMPL and tested on a jumping motion. Since the distributions of

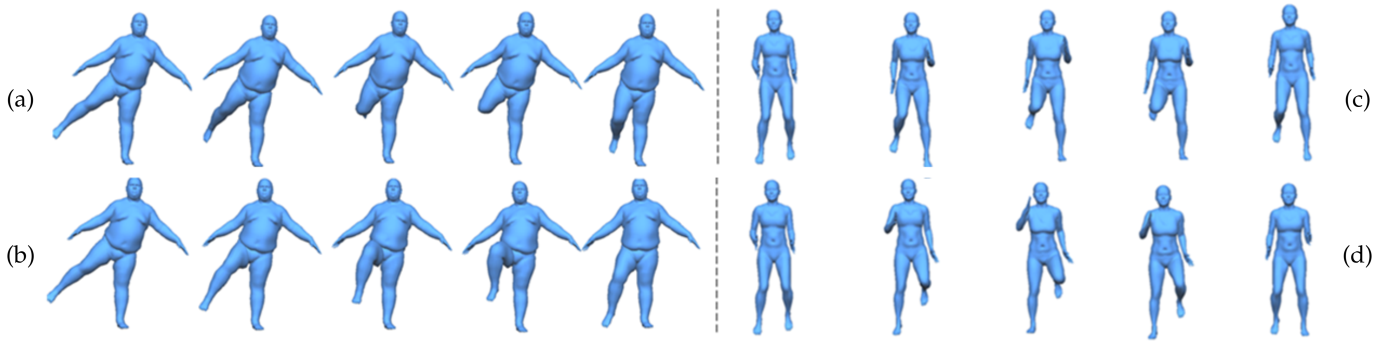


Fig. 9. Conditional generation. Trained with a mixture of different Dyna datasets, our network can output sequences conditioned on the input meshes. In this figure, the first two meshes of each sequence are the input. For example, on the right of the figure, we feed two fit female shapes into the network. The second frame in (c) lifts her right leg while (d) lifts the left leg. Our model can perceive their differences and predict subsequent motions according to the conditions. Similar results can be observed in the fat male example on the left.

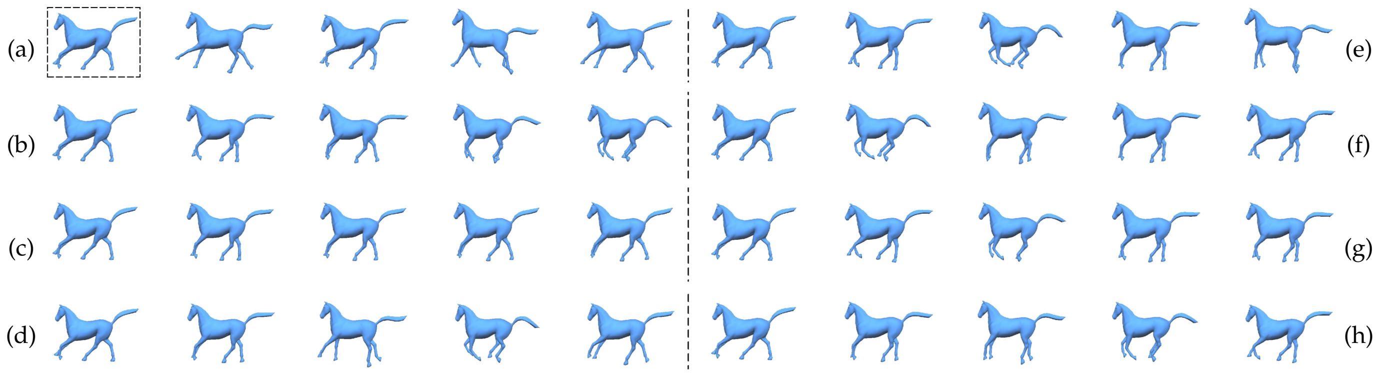


Fig. 10. Generated results by given an initial frame and random states. The frame in the black box is the initial frame. (a) is the ground truth sequence. (b)-(h) are sequences generated by our model on given random initial states in a normal distribution. The results indicate that our method can generate diversified sequences with different input states.

shapes in training and test data differ a lot, meshes generated on the test data might not always be realistic and could involve weird poses with self-intersections. For (b), we train the network on chicken wings and test it on jumping jack from Dyna. The sequence freezes on this shape. The freezing problem also appears in other LSTM papers. In our case, this problem can be resolved by additional user interactions. When the sequence is freezing, we can provide a new keyframe to continue generation.

Several future extensions could be explored. First, our current work mainly focuses on the generation of mesh sequences, while this network structure can also potentially benefit other applications like classification. Second, since the auto-encoder and LSTM are trained together, we have not fully studied how spatial and temporal information individually influenced the encoding process. Finally, the current network only learns the deformation. In the future, people may build more powerful algorithms that take dynamics into consideration.

Acknowledgments

This work was supported by Beijing Program for International S&T Cooperation Project (No. Z191100001619003), Beijing Municipal Natural Science Foundation (No. L182016), National Natural Science Foundation of China (No. 61872440 and No. 61828204), Royal Society Newton Advanced Fellowship (No. NAF\R2\192151), Youth Innovation Promotion Association CAS, Tencent AI Lab Rhino-Bird Focused Research Program (No.JR202024).

References

[1] F. Bogo, J. Romero, M. Loper, and M. J. Black, "FAUST: Dataset and evaluation for 3D mesh registration," in *CVPR*, 2014.
[2] M. Dou, S. Khamis, Y. Degtyarev, P. Davidson, S. R. Fanello, A. Kowdle, S. O. Escolano, C. Rhemann, D. Kim, J. Taylor, P. Kohli, V. Tankovich, and S. Izadi, "Fusion4D: Real-time performance capture of challenging scenes," *ACM Trans. Graph.*, vol. 35, no. 4, pp. 114:1–114:13, 2016.
[3] C. Stoll, J. Gall, E. de Aguiar, S. Thrun, and C. Theobalt, "Video-based reconstruction of animatable human characters," *ACM Trans. Graph.*, vol. 29, no. 6, pp. 139:1–139:10, 2010.
[4] Q. Tan, L. Gao, Y.-K. Lai, and S. Xia, "Variational autoencoders for deforming 3D mesh models," in *CVPR*, 2018.
[5] L. Gao, Y.-K. Lai, D. Liang, S.-Y. Chen, and S. Xia, "Efficient and flexible deformation representation for data-driven surface modeling," *ACM Trans. Graph,*, vol. 35, no. 5, p. 158, 2016.
[6] E. De Aguiar, C. Theobalt, S. Thrun, and H.-P. Seidel, "Automatic conversion of mesh animations into skeleton-based animations," in *Computer Graphics Forum*, vol. 27, no. 2. Wiley Online Library, 2008, pp. 389–397.
[7] H.-B. Yan, S. Hu, R. R. Martin, and Y.-L. Yang, "Shape deformation using a skeleton to drive simplex transformations," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 3, pp. 693–706, 2008.
[8] G. Pons-Moll, J. Romero, N. Mahmood, and M. J. Black, "Dyna: A model of dynamic human shape in motion," *ACM Trans. Graph.*, vol. 34, no. 4, p. 120, 2015.

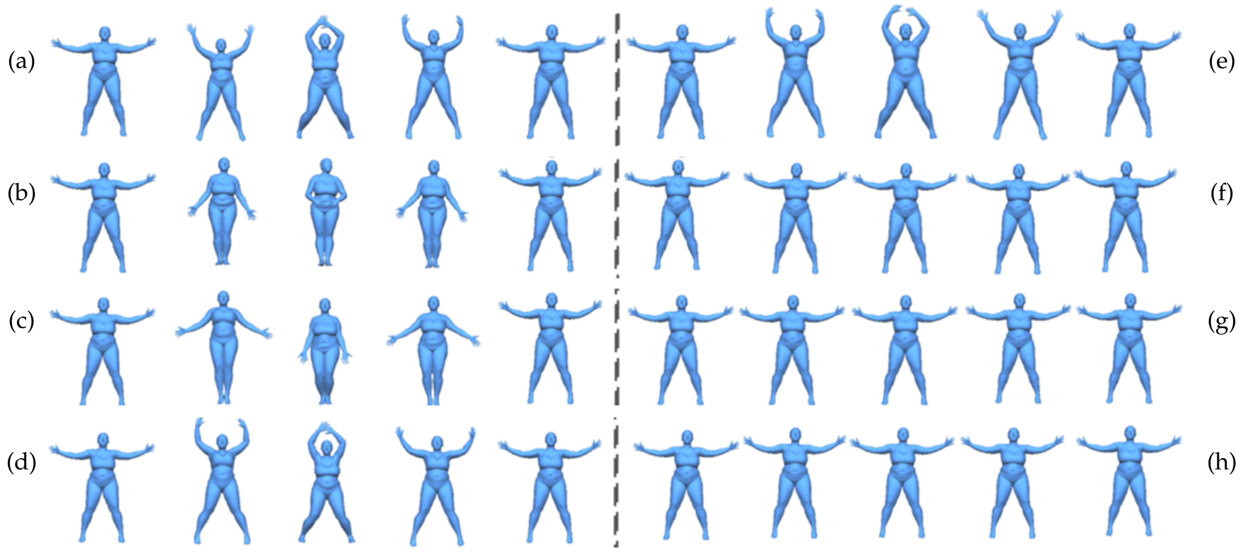


Fig. 11. Diversified sequence completion. We show the completion results produced by different methods. Source (first) and target (last) meshes are shared among all the sequences. (a) is the ground truth from f_ 50004_jumpingJacks dataset [8]; (b), (c) and (d) are the results by our BD-LSTM; (e) is the optimization+unidirectional baseline strategy described in the paper; (f), (g) and (h) are interpolation results using [4], [45] and [24] accordingly. We can see from (f), (g) and (h) that interpolation-based approaches generate almost identical meshes, since interpolation often follows the shortest path between source and target. Compared to (e), our method can generate diverse, plausible results for users to choose.

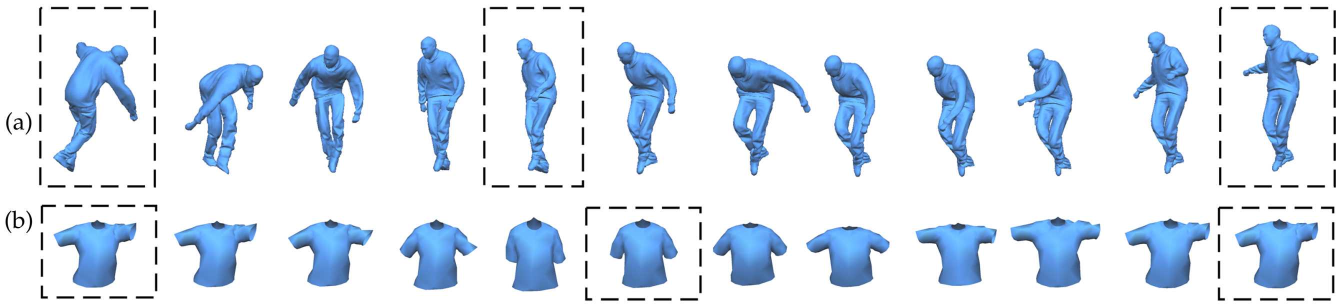


Fig. 12. Keyframe control. Our method can generate a sequence given a small set of keyframes (highlighted in black boxes). The BD-LSTM generates in both forward and backward directions and thus completes the gaps between consecutive keyrames. This application enables more convenient control of sequence synthesis and editing.

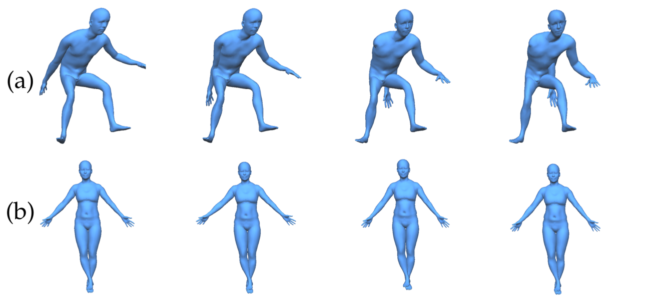


Fig. 13. Failure cases. (a) is obtained by training the network on dancing while testing it on jumping motion from SMPL [50]. (b) is obtained by training the network on chicken wings and testing it on jumping jack from Dyna [8].

[9] O. Sidi, O. van Kaick, Y. Kleiman, H. Zhang, and D. Cohen-Or, "Unsupervised co-segmentation of a set of shapes via descriptor-space spectral clustering," *ACM Trans. Graph.*, vol. 30, no. 6, 2011.

[10] H. Huang, E. Kalogerakis, and B. Marlin, "Analysis and synthesis of 3D shape families via deep-learned generative models of surfaces," *CGF*, vol. 34, no. 5, pp. 25–38, 2015.

[11] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[12] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv:1406.1078*, 2014.

[13] M. Mathieu, C. Couprie, and Y. LeCun, "Deep multi-scale video prediction beyond mean square error," *arXiv:1511.05440*, 2015.

[14] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh, "Action-conditional video prediction using deep networks in atari games," in *NIPS*, 2015.

[15] Q. Lyu, Z. Wu, J. Zhu, and H. Meng, "Modelling high-dimensional sequences with LSTM-RTRBM: Application to polyphonic music generation," in *IJCAI*, 2015.

[16] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *CVPR*, 2015.

[17] O. Melamud, J. Goldberger, and I. Dagan, "Context2vec: Learning generic context embedding with bidirectional LSTM," in *SIGNLL*, 2016.

[18] F. M. Bianchi, L. Livi, and C. Alippi, "Investigating echo-state networks dynamics by means of recurrence analysis," *IEEE TNNLS*, vol. 29, no. 2, pp. 427–439, 2018.

[19] E. Kalogerakis, M. Averkiou, S. Maji, and S. Chaudhuri, "3D shape segmentation with projective convolutional networks," in *CVPR*, 2017.

[20] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3D shape recognition," in *ICCV*, 2015.

[21] G. Riegler, A. O. Ulusoy, and A. Geiger, "Octnet: Learning deep 3D representations at high resolutions," in *CVPR*, 2017.

[22] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum, "Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling," in *NIPS*, 2016.

[23] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *NIPS*, 2015.

[24] L. Gao, Y.-K. Lai, J. Yang, L.-X. Zhang, L. Kobbelt, and S. Xia, "Sparse data driven mesh deformation," *IEEE TVCG*, 2019.

[25] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio, "Generating sentences from a continuous space," *arXiv:1511.06349*, 2015.

[26] T. Mikolov, S. Kombrink, L. Burget, J. Černockỳ, and S. Khudanpur, "Extensions of recurrent neural network language model," in *ICASSP*, 2011.

[27] J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio, "A recurrent latent variable model for sequential data," in *NIPS*, 2015.

[28] E. Marchi, G. Ferroni, F. Eyben, L. Gabrielli, S. Squartini, and B. Schuller, "Multi-resolution linear prediction based features for audio onset detection with bidirectional LSTM neural networks," in *ICASSP*, 2014.

[29] N. Srivastava, E. Mansimov, and R. Salakhudinov, "Unsupervised learning of video representations using LSTMs," in *ICML*, 2015.

[30] W. Lotter, G. Kreiman, and D. Cox, "Deep predictive coding networks for video prediction and unsupervised learning," *arXiv:1605.08104*, 2016.

[31] L. Yu, W. Zhang, J. Wang, and Y. Yu, "SeqGAN: Sequence generative adversarial nets with policy gradient," in *AAAI*, 2017.

[32] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *NIPS*, 2014.

[33] C. Vondrick, H. Pirsiavash, and A. Torralba, "Generating videos with scene dynamics," in *NIPS*, 2016.

[34] J. Walker, K. Marino, A. Gupta, and M. Hebert, "The pose knows: Video forecasting by generating pose futures," in *ICCV*, 2017.

[35] H. Cai, C. Bai, Y.-W. Tai, and C.-K. Tang, "Deep video generation, prediction and completion of human action sequences," *arXiv:1711.08682*, 2017.

[36] K. Fragkiadaki, S. Levine, P. Felsen, and J. Malik, "Recurrent network models for human dynamics," in *ICCV*, 2015.

[37] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra, "DRAW: A recurrent neural network for image generation," *arXiv:1502.04623*, 2015.

[38] Z. Li, Y. Zhou, S. Xiao, C. He, and H. Li, "Auto-conditioned LSTM network for extended complex human motion synthesis," *arXiv:1707.05363*, 2017.

[39] J. Martinez, M. J. Black, and J. Romero, "On human motion prediction using recurrent neural networks," in *CVPR*, 2017.

[40] J. Bütepage, M. J. Black, D. Kragic, and H. Kjellström, "Deep representation learning for human motion prediction and classification," in *CVPR*, 2017.

[41] T. Y. Wang, T. Shao, K. Fu, and N. J. Mitra, "Learning an intrinsic garment space for interactive authoring of garment animation," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 6, pp. 1–12, 2019.

[42] H. Wang, E. S. Ho, H. P. Shum, and Z. Zhu, "Spatio-temporal manifold learning for human motions via long-horizon modeling," *IEEE transactions on visualization and computer graphics*, 2019.

[43] Y. Xiao, Y. Lai, F. Zhang, C. Li, and L. Gao, "A survey on deep geometry learning: From a representation perspective," *Comput. Vis. Media*, vol. 6, no. 2, pp. 113–133, 2020. [Online]. Available: https://doi.org/10.1007/s41095-020-0174-8

[44] P. Huber, R. Perl, and M. Rumpf, "Smooth interpolation of key frames in a Riemannian shell space," *Computer Aided Geometric Design*, vol. 52, pp. 313–328, 2017.

[45] L. Gao, S.-Y. Chen, Y.-K. Lai, and S. Xia, "Data-driven shape interpolation and morphing editing," *CGF*, vol. 36, no. 8, pp. 19–31, 2017.

[46] Q. Tan, L. Gao, Y.-K. Lai, J. Yang, and S. Xia, "Mesh-based autoencoders for localized deformation component analysis," in *AAAI*, 2018.

[47] Z. Levi and C. Gotsman, "Smooth rotation enhanced as-rigid-as-possible mesh animation," *IEEE TVCG*, vol. 21, no. 2, pp. 264–277, 2015.

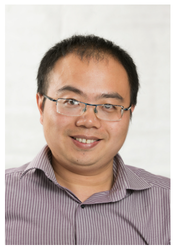
[48] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," *arXiv:1312.6114*, 2013.

[49] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv:1412.6980*, 2014.

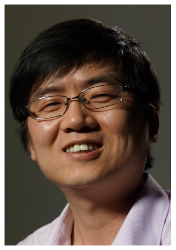
[50] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black, "SMPL: A skinned multi-person linear model," *ACM Trans. Graphics (Proc. SIGGRAPH Asia)*, vol. 34, no. 6, pp. 248:1–248:16, Oct. 2015.

[51] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary computation*, vol. 9, no. 2, pp. 159–195, 2001.

Yi-Ling Qiao received a bachelor's degree in computer science and technology from the University of Chinese Academy of Sciences in 2019. He is currently a PhD student in computer science at University of Maryland, College Park. His research interests include computer graphics and geometric processing.



Yu-Kun Lai received his bachelor's degree and PhD degree in computer science from Tsinghua University in 2003 and 2008, respectively. He is currently a Professor in the School of Computer Science & Informatics, Cardiff University. His research interests include computer graphics, geometry processing, image processing and computer vision. He is on the editorial boards of *Computer Graphics Forum* and *The Visual Computer*.



Hongbo Fu received the BS degree in information sciences from Peking University, China, in 2002 and the PhD degree in computer science from the Hong Kong University of Science and Technology, in 2007. He is a Professor in the School of Creative Media, City University of Hong Kong. His primary research interests fall in the fields of computer graphics and human computer interaction. He has served as an associate editor of The Visual Computer, Computers & Graphics, and Computer Graphics Forum.

Lin Gao received the bachelor's degree in mathematics from Sichuan University and the PhD degree in computer science from Tsinghua University. He is currently an Associate Professor at the Institute of Computing Technology, Chinese Academy of Sciences. He has been awarded the Newton Advanced Fellowship from the Royal Society and the AG young researcher award. His research interests include computer graphics and geometric processing.