GANimator: Neural Motion Synthesis from a Single Sequence

PEIZHUO LI, ETH Zurich, Switzerland
KFIR ABERMAN, Google Research, USA
ZIHAN ZHANG, The University of Chicago, USA
RANA HANOCKA, The University of Chicago, USA
OLGA SORKINE-HORNUNG, ETH Zurich, Switzerland

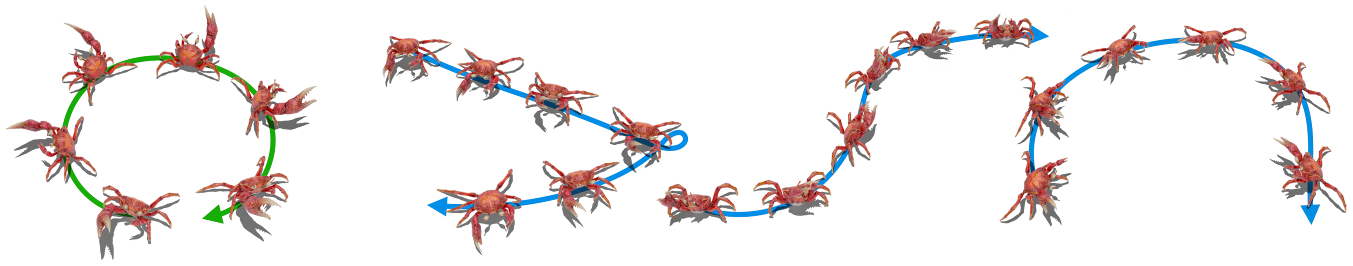

Fig. 1. Given **a single training motion sequence** of skeleton with arbitrary topology, our method learns to **synthesize novel motion sequences**.

We present GANimator, a generative model that learns to synthesize novel motions from a single, short motion sequence. GANimator generates motions that resemble the core *elements* of the original motion, while simultaneously synthesizing novel and diverse movements. Existing data-driven techniques for motion synthesis require a large motion dataset which contains the desired and specific skeletal structure. By contrast, GANimator only requires training on a *single* motion sequence, enabling novel motion synthesis for a variety of skeletal structures *e.g.,* bipeds, quadropeds, hexapeds, and more. Our framework contains a series of generative and adversarial neural networks, each responsible for generating motions in a specific frame rate. The framework progressively learns to synthesize motion from random noise, enabling hierarchical control over the generated motion content across varying levels of detail. We show a number of applications, including crowd simulation, key-frame editing, style transfer, and interactive control, which all learn from a single input sequence. Code and data for this paper are at https://peizhuoli.github.io/ganimator.

CCS Concepts: • **Computing methodologies** → **Motion processing**; *Machine learning approaches.*

Additional Key Words and Phrases: neural motion processing, motion synthesis

Authors' addresses: Peizhuo Li, ETH Zurich, Switzerland, peizli@ethz.ch; Kfir Aberman, Google Research, USA, kfiraberman@gmail.com; Zihan Zhang, The University of Chicago, USA, zzhang18@uchicago.edu; Rana Hanocka, The University of Chicago, USA, ranahanocka@gmail.com; Olga Sorkine-Hornung, ETH Zurich, Switzerland, sorkine@inf.ethz.ch.

0730-0301/2022/7-ART138
https://doi.org/10.1145/3528223.3530157

ACM Reference Format:
Peizhuo Li, Kfir Aberman, Zihan Zhang, Rana Hanocka, and Olga Sorkine-Hornung. 2022. GANimator: Neural Motion Synthesis from a Single Sequence. *ACM Trans. Graph.* 41, 4, Article 138 (July 2022), 12 pages. https://doi.org/10.1145/3528223.3530157

1 INTRODUCTION

Generating realistic and diverse human motion is a long-standing objective in computer graphics. Motion modeling and synthesis commonly uses a probabilistic model to capture limited local variations [Li et al. 2002] or utilizes a large motion dataset obtained by motion capture (mocap) [Holden et al. 2016]. Capturing data with a mocap system is costly both during stage-setting and in post-process (e.g., involving manual data clean-up). Motion datasets are often limited, i.e., they lack the desired skeletal structure, body proportions, or styles. Therefore, utilizing motion datasets often requires non-trivial processing such as retargeting, which may potentially degrade or introduce errors in the original captured motion. Moreover, there are no extensive datasets which contain imaginary creatures or non-standard animals (such as the hexapedal crab in Figure 1), which limits existing data-driven techniques.

In this work, we develop a framework that is capable of generating diverse and realistic motions using only a *single* training sequence. Our strategy greatly simplifies the data collection process while still allowing the framework to create realistic variations and faithfully capture the details of the individual motion sequence. Inspired by the SinGAN model for image synthesis [Shaham et al. 2019], our key idea is to leverage the information within a single motion sequence in order to exploit the rich data over multiple temporal and spatial scales. The generation process is divided into several levels, similar to progressive training schemes for images [Karras et al. 2018; Shaham et al. 2019].

Our framework is an effective tool for generating novel motion which is not exactly present in the single given training sequence. GANimator can synthesize long, diverse, and high-quality motion sequences using only a short motion sequence as input. We show that for various types of motions and characters, we generate sequences that still resemble the core *elements* of the original motion, while simultaneously synthesizing novel and diverse movements. It should be noted that the synthesized motion is not copied verbatim from the original sequence. Namely, patches from the original motion do not appear in the synthesized motion.

We demonstrate the utility of the GANimator framework to generate both *conditional* and *unconditional* motion sequences that are *reminiscent* of the input motion. GANimator can unconditionally (*i.e.,* driven by noise) generate motions for simulating crowds and motion editing/mixing. In addition, GANimator is able to produce *controllable* motion sequences, which are conditioned on a user-given input (*e.g.,* trajectory position). We are able to inject global position and rotation of the root joint as an input for interactively controlling the trajectory of the synthesized motion. GANimator is also able to perform key-frame editing, which generates high-quality interpolations between user-modified poses. Lastly, GANimator enables motion style transfer – synthesizing the style of the input motion onto a different motion.

Our system relies on skeleton-aware operators [Aberman et al. 2020a] as a backbone for our neural motion synthesis framework. The skeleton-aware layers provides the platform for applying convolutions over a fixed skeleton topology. Since our network trains on a single motion sequence, we automatically adjust the operators to adhere to the structure of the input skeleton. Therefore, our system is able to train on a wide variety of skeletal structures, *e.g.,* bipeds, quadropeds, hexapeds, and more. Further, for ground-inhabiting creatures, we incorporate a self-supervised foot contact label. This ensures proper placement of the feet on the ground plane and avoids notorious foot sliding artifacts. We demonstrate the effectiveness of GANimator in handling a wide variety of skeletal structures and motions, and its applicability in various motion editing and synthesis tasks.

Achieving desirable results in this constrained scenario is highly challenging, with existing techniques commonly producing undesirable results that fall into one of two extremes. In the first extreme, generated results span the breadth of poses contained in the original motion sequence, but are jittery and incoherent. In the second extreme, results are smooth, but lack variety and coverage of the motion elements contained in the original sequence. Our proposed technique strikes a favorable balance between these extremes, synthesizing high-quality, novel, and varied motion sequences. Our framework produces desirable motion sequences that contain all the original motion elements, while still achieving diverse and smooth motion sequences.

2 RELATED WORK

We review various relevant works on motion generation, focusing on human and other character animation. For an in-depth survey of this extensive body of literature we refer the readers to the surveys in [Geijtenbeek et al. 2011; Mourot et al. 2021].

Statistical modeling of visual and motion data. Representing and generating complex visual objects and phenomena of stochastic nature, such as stone and wood textures or breathing and walking cycles with natural variations, is one of the fundamental tasks in computer graphics. In one of the pioneer works in this area, Perlin [1985] introduced the famous Perlin noise function that can be used for image synthesis with variations. This technique was later applied to procedural animation synthesis [Perlin and Goldberg 1996] to achieve random variability. Instead of using additive noise, statistical approaches [Bowden 2000; Brand and Hertzmann 2000; Chai and Hodgins 2007; Pullen and Bregler 2000] propose using probabilistic models such as kernel-based distribution and hidden Markov models, enabling synthesis with variations via direct sampling of the model. To capture both local and global structure, MotionTexture [Li et al. 2002] clusters a large training dataset into textons, a linear dynamics model capable of capturing local variation, and models the transition probability between textons. Though the goal resembles ours, this data-hungry statistical model does not fit in our single-motion setting. We perform an in-depth comparison to MotionTexture regarding the ability to model both global and local variations when data is limited in Sec. 4.2. A similar idea of using two-level statistical models is employed in multiple works [Lee et al. 2002; Tanco and Hilton 2000] for novel motion synthesis. Other statistical models based on Gaussian processes learn a compact latent space [Grochow et al. 2004; Levine et al. 2012; Wang et al. 2007] or facilitate an interactive authoring process [Ikemoto et al. 2009] from a small set of examples for multiple applications, including inverse kinematics, motion editing and task-guided character control. Lau et al. [2009] use a similar Bayesian approach to model the natural variation in human motion from several examples. However, probabilistic approaches tend to capture limited variations in local frames and require a large dataset for a better understanding of global structure variations.

Interpolation and blending of examples. Another approach for example-based motion generation is explicit blending and concatenation of existing examples. Early works [Mizuguchi et al. 2001; Park et al. 2002; Rose et al. 1998, 1996; Wiley and Hahn 1997] use various interpolation and warping techniques for this purpose. Matching-based works [Arikan and Forsyth 2002; Kovar and Gleicher 2004; Pullen and Bregler 2002] generate motion by matching user-specified constraints in the existing dataset and interpolating the retrieved clips. Kovar et al. [2002] explicitly model the structure of a corpus of motion capture data using an automatically constructed directed graph (the *motion graph*). This approach enables flexible and controllable motion generation and is widely used in interactive applications like computer games. Min et al. [2012] propose a mixture of statistical and graph-based models to decouple variations in the global composition of motions or actions, and the local movement variations. One significant constraint of blending-based methods is that the diversity of the generated results is limited to the dataset, since it is composed of interpolations and combinations of the dataset. Further works [Heck and Gleicher 2007; Safonova and Hodgins 2007; Zhao et al. 2009] focus on the efficiency and robustness of the algorithm but are still inherently limited in the variety of the generated results. Motion matching [Büttner and Clavet

2015] directly finds the best match in the mocap dataset given the user-provided constraints, bypassing the construction of a graph to achieve better realism. However, it generally requires task-specific tuning and is unable to generate novel motion that are unseen in the dataset.

Physics-based motion synthesis. Another loosely related field is physically-based motion generation, where the generation process runs a *controller* in a physics simulator, unlike kinematics approaches that directly deal with joints' transformations. Agrawal et al. [2013] optimize a procedural controller given a motion example to achieve physical plausibility while ensuring diversity in generated results. However, such controllers are hand-crafted for specific tasks, such as jumping and walking, and are thus difficult to generalize to different motion data. Results in [Wei et al. 2011; Ye and Liu 2010] show that combining physical constraints and statistical priors helps generate physically realistic animations and reactions to external forces, but the richness of motion is still restricted to the learned prior. With the evolution of deep reinforcement learning, control policies on different bodies including biped locomotion [Heess et al. 2017; Peng et al. 2017] and quadurpeds [Luo et al. 2020] can be learned from scratch without reference. It is also possible to achieve high quality motion by learning from reference animation [Peng et al. 2018]. Lee et al. [2021] propose to learn a parameterized family of control policies from a single clip of a single movement, e.g., jumping, kicking, backflipping, and are able to generate novel motions for a different environment, target task, and character parameterization. As in other works, the learned policy is limited to several predefined tasks.

Neural motion generation. Taylor and Hinton [2009] made initial attempts to model motion style with neural networks by restricted Boltzmann machines. Holden et al. [2016; 2015] applied convolutional neural networks (CNN) to motion data for learning a motion manifold and motion editing. Concurrently, Fragkiadaki et al. [2015] chose to use recurrent neural networks (RNN) for motion modeling. RNN based works also succeed in short-term motion prediction [Fragkiadaki et al. 2015; Pavllo et al. 2018], interactive motion generation [Lee et al. 2018] and music-driven motion synthesis [Aristidou et al. 2021]. Zhou et al. [2018] tackle the problem of error accumulation in long-term random generation by alternating the network's output and ground truth as the input of RNN during training. This method, called acRNN, is able to generate long and stable motion similar to the training set. However, like many other deep learning based methods, it struggles when only a short training sequence is provided, whereas we are able to address the limited data problem. We compare our method to acRNN in Sec. 4.2. Holden et al. [2017] propose phase-functioned neural networks (PFNN) for locomotion generation and introduce *phase* to neural networks. Similar ideas are used in quadruped motion generation by Zhang et al. [2018]. Starke et al. [2020] extend phase to local joints to cope with more complex motion generation. Henter et al. [2020] propose another generative model for motion based on normalizing flow. Neural networks succeed in a variety of motion generation tasks: motion retargeting [Aberman et al. 2020a, 2019; Villegas et al. 2018], motion style transfer [Aberman et al. 2020b; Mason et al. 2022], key-frame based motion generation [Harvey et al. 2020], motion matching [Holden et al. 2020] and animation layering [Starke et al. 2021]. It is worth noting that the success of deep learning methods hinges upon large and comprehensive mocap datasets. However, acquiring a dataset involves costly capturing steps and nontrivial post-processing, as discussed in Section 1. In contrast, our method achieves comparable results by learning from a single input motion sequence.

3 METHOD

We propose a generative model that can learn from a single motion sequence. Our approach is inspired by recent works in the image domain that use progressive generation [Karras et al. 2018] and works that propose to train deep networks on a single example [Shaham et al. 2019; Shocher et al. 2019]. We next describe the main building blocks of our hierarchical framework, motion representation, and the training procedure.

3.1 Motion representation

We represent a motion sequence by a temporal set of T poses that consists of root joint displacements $\mathbf{O} \in \mathbb{R}^{T\times 3}$ and joint rotations $\mathbf{R} \in \mathbb{R}^{T\times JQ}$, where J is the number of joints and Q is the number of rotation features. The rotations are defined in the coordinate frame of their parent in the kinematic chain, and represented by the 6D rotation features ($Q = 6$) proposed by Zhou et al. [2019], which yields the best result among other representations for our task (see ablation study in Section 4.4).

To mitigate common foot sliding artifacts, we incorporate foot *contact labels* in our representation. In particular, we concatenate to the feature axis $C \cdot T$ binary values $\mathbf{L} \in \{0, 1\}^{T\times C}$, which correspond to the contact labels of the foot joints, $\mathcal{F}$, of the specific creature. For example, for humanoids, we use $\mathcal{F}$ = {left heel, left toe, right heel, right toe}. For each joint $j \in \mathcal{F}$ and frame $t \in \{1, \dots, T\}$, the tj-th label is calculated via

$$\mathbf{L}^{tj} = \mathbb{1}[\|\mathrm{FK}_{\mathbf{S}}([\mathbf{R}, \mathbf{O}])^{tj}\|_2 < \epsilon],$$

where $\|\mathrm{FK}_{\mathbf{S}}([\mathbf{R}, \mathbf{O}])^{tj}\|_2$ denotes the magnitude of the velocity of joint j in frame t retrieved by a forward kinematics (FK) operator. FK applies the rotation and root joint displacements on the skeleton $\mathbf{S}$, and $\mathbb{1}[V]$ is an indicator function that returns 1 if V is true and 0 otherwise.

To simplify the notation, we denote the metric space of the concatenated features by $\mathcal{M}_T \equiv \mathbb{R}^{T\times(JQ+C+3)}$. In addition, we denote the input motion features by $\mathbf{T} \equiv [\mathbf{R}, \mathbf{O}, \mathbf{L}] \in \mathcal{M}_T$, and its correponding downsampled versions by $\mathbf{T}_i \in \mathcal{M}_{T_i}$.

3.2 Progressive motion synthesis architecture

Our motion generation framework is illustrated in Fig 2. It consists of S coarse-to-fine generative adversarial networks (GANs) [Goodfellow et al. 2014], each of which is responsible to generate motion sequences with a specific number of frames $\{T_i\}_{i=1}^S$. We denote the generators and discriminators by $\{G_i\}_{i=1}^S$ and $\{D_i\}_{i=1}^S$, respectively.

The first level is purely generative, namely, G_1 maps a random noise $z_1 \in \mathcal{M}_{T_1}$ into a coarse motion sequence

$$\mathbf{Q}_1 = G_1(z_1), \tag{1}$$

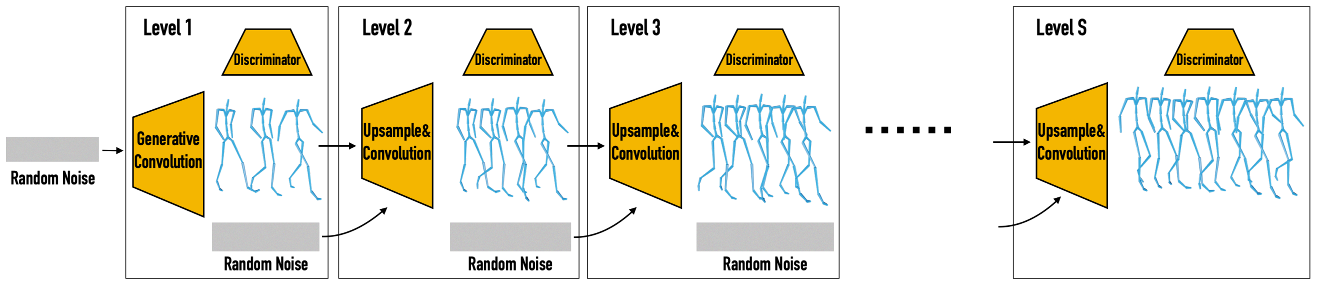

Fig. 2. Our progressive motion synthesis architecture. Starting with random noise that synthesize a coarse motion sequence through a generative network, our framework progressively upsamples the motion until it reaches to the finest temporal resolution. Each level receives the output of the previous level (except for the first level) and a random noise as output an upsampled version of the input sequence. We use adversarial training, such that each generated result is fed into a discriminator at a corresponding level.

where $\mathbf{Q}_1 \in \mathcal{M}_{T_1}$. Then, the generators in the finer levels G_i ($2 \leq i \leq S$) progressively upsample $\mathbf{Q}_1$ via

$$\mathbf{Q}_i = G_i(\mathbf{Q}_{i-1}, z_i), \tag{2}$$

where in each level the sequence is upsampled by a fixed scaling factor $F > 1$. The process is repeated until the finest output sequence $\mathbf{Q}_S \in \mathcal{M}_{T_S}$ is generated by G_S.

Note that $z_i \in \mathcal{M}_{T_i}$ has an i.i.d normal distribution $\sim \mathcal{N}(0, \sigma_i)$ along the temporal axis while being shared along the channel axis, and we found that σ_i is highly correlated with the magnitude of the high-frequency details generated by G_i, thus, we select

$$\sigma_i = \frac{1}{Z_i} \|\uparrow \mathbf{T}_{i-1} - \mathbf{T}_i\|_2^2, \tag{3}$$

where $Z_i = T_i(QJ + C + 3)$ is the number of entries in $\mathbf{T}_i$, and $\uparrow$ is a linear upsampler with a scaling factor $F > 1$. In all of our experiments we select $F = 4/3$ and $S = 7$.

3.2.1 *Network components.*

Generator. Our generator G_i contains a fully convolutional neural network $g_i(\cdot)$ that has a few skeleton-aware convolution layers [Aberman et al. 2020a] followed by non-linear layers (see Appendix A). Since the main role of the network is to add missing high-frequency details, we use a residual structure [He et al. 2016], hence for $2 \leq i \leq S$, we get

$$G_i(\mathbf{Q}_{i-1}, z_i) = g_i(\uparrow \mathbf{Q}_{i-1} + z_i) + \uparrow \mathbf{Q}_{i-1}. \tag{4}$$

Discriminator. While discriminators in classic GAN architectures output a single scalar indicating whether the input is classified as "real" or "fake", such a structure in the case of a single sequence in the training data leads to mode collapse, since our generator trivially overfits the sequence. In order to prevent overfitting we limit the receptive field of the discriminator by employing a PatchGAN [Isola et al. 2017; Li and Wand 2016] classifier, which calculates a confidence value to each input patch. The final output of our discriminator is the average value of all the per-patch confidence values, predicted by the network.

Skeleton-aware operators. We employ skeleton-aware convolutions [Aberman et al. 2020a] as a fundamental building block in our framework. Skeleton-aware operators require a fixed skeleton topology that is defined by a set of joints (vertices) and an adjacency list (edges). Since our network operates on a single sequence we adapt the topology to adhere the input sequence. This enables operation on any skeleton topology, and does not require retargeting the input motion to a specific skeletal structure. To incorporate the foot contact labels into the skeleton-aware representation, we treat each label as a virtual joint connected to its corresponding joint rotations vertex. In addition, due to the high correlation between end-effectors and global positions, we add a connection between the contact labels vertices to the vertex of the global displacements $\mathbf{O}$, where we treat the latter as another virtual joint that is connected to the neighbors of the root joint in the kinematic chain.

3.2.2 *Loss functions.*

Adversarial Loss. We train level i with the WGAN-GP [2017] loss:

$$\begin{aligned}\mathcal{L}_{\text{adv}} &= \mathbb{E}_{\mathbf{Q}_i \sim \mathbb{P}_{g_i}}\left[D_i(\mathbf{Q}_i)\right] - D_i(\mathbf{T}_i) \\ &+ \lambda_{\text{gp}} \mathbb{E}_{\hat{\mathbf{Q}} \sim \mathbb{P}_{\hat{g}_i}}\left[\left(\|\nabla D_i(\hat{\mathbf{Q}}_i)\|_2 - 1\right)^2\right],\end{aligned} \tag{5}$$

where $\mathbb{P}_{g_i}$ is defined as the distributions of our generated samples in the ith level, ∇ is the gradient operator and $\mathbb{P}_{\hat{g}_i}$ is the distribution of the linear interpolations $\hat{\mathbf{Q}}_i = \lambda \mathbf{Q}_i + (1 - \lambda)\mathbf{T}_i$ with λ as a uniformly distributed variable in $[0, 1]$. The gradient penalty term in (6) enforces Lipschitz continuity so the Wasserstein distance between generated and training distribution can be well approximated [Gulrajani et al. 2017].

Reconstruction Loss. To ensure that the network generates variations of all the different temporal patches, and does not collapse to generation of a specific subset of movements, we require the network to reconstruct the input motion from a set of pre-defined noise signals $\{z_i^*\}_{i=1}^S$, namely, $G_i(\uparrow \mathbf{T}_{i-1}, z_i^*)$ should approximate the single training example $\mathbf{T}_i$ at level i. To encourage the system to do so, we define a reconstruction loss

$$\mathcal{L}_{\text{rec}} = \|G_i(\uparrow \mathbf{T}_{i-1}, z_i^*) - \mathbf{T}_i\|_1. \tag{6}$$

Note that the noise models the variation of generated results, but during reconstruction, we do not expect any variation. To this end, we fix z_1^* as a pre-generated noise for the first level and set $z_i^* = 0$ for the other levels.

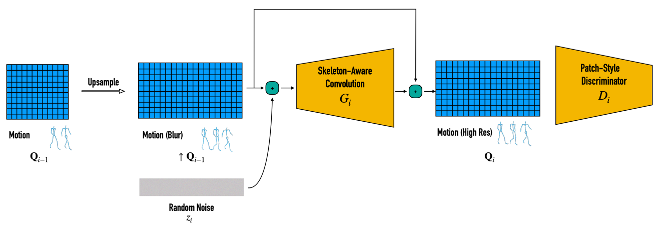

Fig. 3. Generator structure. Our residual generator receives the sum of a noise vector and the upsampled result from previous level (in the first level it receives only noise). It predicts the missing high-frequency details, which is added to the input animation via skip connections.

Contact Consistency Loss. As accurate foot contact is one of the major factors of motion quality, we predict the foot contact labels in our framework and use an IK post-process to ensure the contact. Since the joint contact labels $\mathbf{L}$ are integrated into our motion representation $\mathcal{M}$, the skeleton-aware networks can directly operate on $\mathcal{M}$ and learn to predict the contact label as part of the motion.

We noticed that the implicit learning of contact labels can cause artifacts in the transition between activated and non-activated contact labels. Thus, we propose a new loss that encourages a consistency between contact label and feet velocity. We require that in every frame either the contact label or the foot velocity will be minimized, via

$$\mathcal{L}_{\text{con}} = \frac{1}{T|\mathcal{F}|} \sum_{j\in\mathcal{F}} \sum_{t=1}^{T} \|\text{FK}_{\text{S}}(\mathbf{R},\mathbf{O})^{tj}\|_2^2 \cdot s(\mathbf{L}^{tj}), \tag{7}$$

where $s(x) = 1/[1+\exp(5-10x)]$ is the transformed sigmoid function. We demonstrate the effectiveness of this loss in the ablation study (Section 4.4).

3.2.3 *Training.*

Our full loss used for training summarizes as:

$$\mathcal{L} = \lambda_{\text{adv}}\mathcal{L}_{\text{adv}} + \lambda_{\text{rec}}\mathcal{L}_{\text{rec}} + \lambda_{\text{con}}\mathcal{L}_{\text{con}}. \tag{8}$$

Although each level can be trained separately, the generated samples of generators in the low levels may be over-blurred due to low temporal resolution and the smoothing effect applied by convolutional kernel. To improve the robustness and quality of the results, we combine every 2 consecutive levels as a block and train the framework block by block. A similar technique is also used by Hinz et al. [2021].

For a detailed description of the layers in each component and the specific values of the hyper-parameters, we refer the reader to Appendix A.

4 EXPERIMENTS AND EVALUATION

We evaluate our results, compare them to other motion generation techniques and demonstrate the effectiveness of various components in our framework through an ablation study. Please refer to the supplementary video for the qualitative evaluation.

4.1 Implementation details

Our GANimator framework is implemented in PyTorch [Paszke et al. 2019], and the experiments are performed on NVIDIA GeForce RTX 2080 Ti GPU. We optimize the parameters of our network with the loss term in Eq. (8) using the Adam optimizer [Kingma and Ba 2014]. We used different training sequences, whose length ranges from 140 frames to 800 frames at 30fps. It contains both artist-created animation and motion capture data from Mixamo [2021] and Truebones [2022]. The training time is proportional to the training sequence length, e.g., it takes about 4 hours to train our network on a human animation sequence with around 600 frames (15,000 iterations per level).

4.2 Novel motion synthesis

We demonstrate the ability of motion sequence extrapolation in Fig. 4 and compare our method to the recent acRNN work [Zhou et al. 2018] and the classical statistical model MotionTexture [Li et al. 2002]. We quantitatively compare these methods with metrics dedicated to a single training example in Section 4.3.

Since our network is fully convolutional, we can generate high-quality motion sequences of arbitrary length given a single training example. As a simple application, we can easily generate a crowd animation, as shown in Fig. 5 and the accompanying video.

When only a single training sequence is provided, acRNN can only generate a limited number of frames before converging to a constant pose, because the lack of data leads to an overfitted model that is not robust to perturbation and error accumulation in RNNs, while our fully convolutional framework does not suffer from this issue.

MotionTexture [Li et al. 2002] automatically clusters the patches of the training dataset into several textons and models the transition probability between textons, where each texton represents the variation of a small segment of similar motions. MotionTexture relies on similar but not identical patches in the dataset for modeling local variations. It constructs the global transition probability between textons based on the frequencies of consecutive relationships of corresponding patches in the dataset. However, when only a single training sequence is provided, we need to manually divide the sequence into patches and specify the transition between textons. When splitting the training sequences into several textons and manually permuting them to create global structure variations, the transitions between textons can be unnatural (see Fig. 4). It is possible to learn a single texton for the whole training sequence to achieve better quality, but the model creates almost zero local variations due to limited data.

4.3 Evaluation

We discuss quantitative metrics for novel motion synthesis from a single training sequence and compare our method and existing motion generation techniques.

Coverage. An important quantitative measurement for the quality of our model is the *coverage* of the training example. Since there is only one training example, we measure the coverage on all possible temporal windows $\mathcal{W}(\mathbf{T}, L) = \left\{\mathbf{T}^{i:i+L-1}\right\}_{i=1}^{L_T-L+1}$ of a given length L, where $\mathbf{T}^{i:j}$ denotes the sequence of frames i to j of the training example $\mathbf{T}$, and L_T is the total length of $\mathbf{T}$. Given a generated result $\mathbf{Q}$, we label a temporal window $\mathbf{T}_w \in \mathcal{W}(\mathbf{T}, L_c)$ as covered if its distance measure to the nearest neighbor in $\mathbf{Q}$ is smaller than an empirically chosen threshold ϵ. The coverage of animation $\mathbf{Q}$ on $\mathbf{T}$

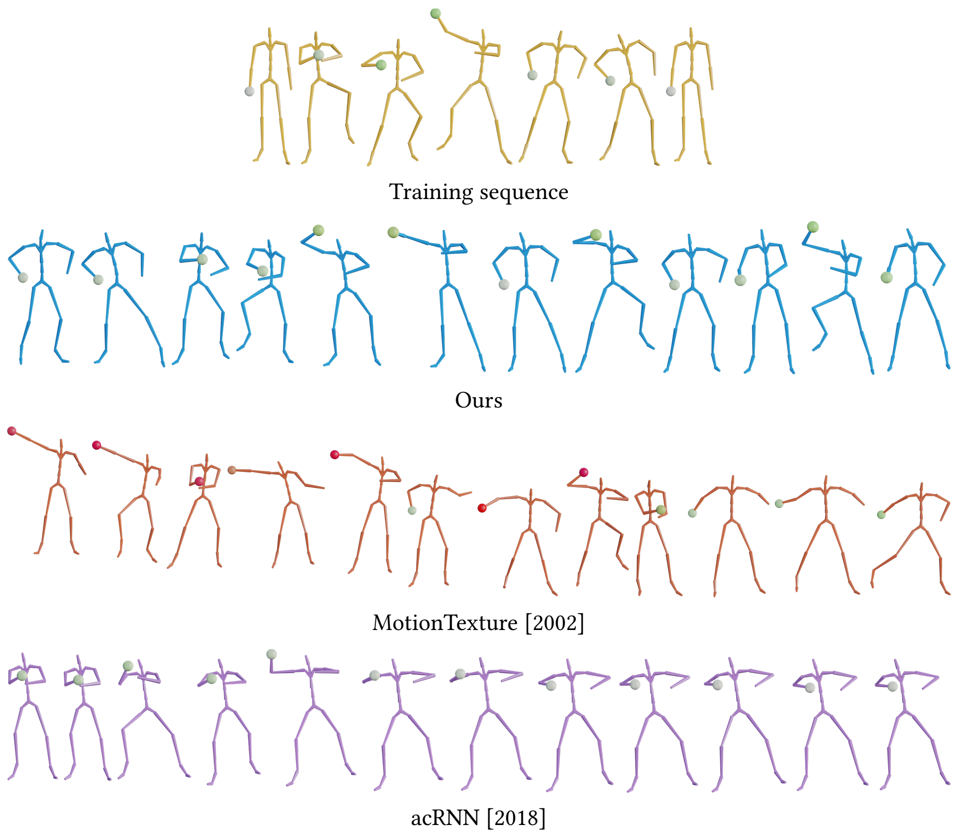

Fig. 4. We train our framework, MotionTexture [2002] and acRNN [2018] on the Gangnam style dancing sequence with 371 frames and use them to synthesize a new sequence with 600 frames. The magnitude of velocity of the right hand is visualized with a heatmap (white - low, green - average, red - high). It can be seen that our method generates global structure variation, the poses and transitions look natural (see supplementary video) and visually similar to the training sequence. For MotionTexture [2002], we manually pick a path between all the trained textons to generate results with similar structure. However, it can be seen that such a process result in unnatural transitions visualized by the large hand velocity and bad foot contact. The result of acRNN [2018] converges to a static pose very quickly due to insufficient data and thus the velocity of the hand gradually vanishes.

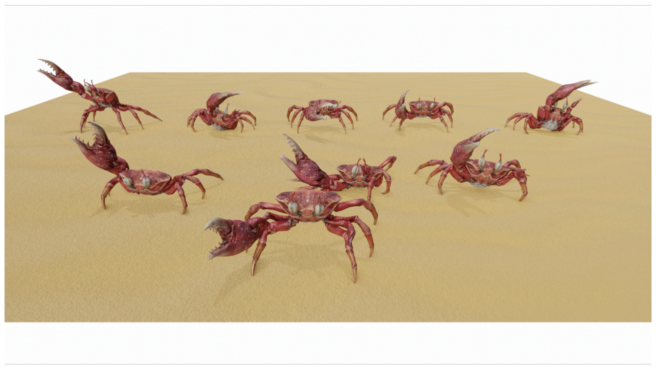

Fig. 5. Crowd animation. Our framework trained on a single crab dancing sequence can synthesize various novel motions that can be used to simulate crowd and augment data for various purposes.

is defined as

$$\mathrm{Cov}(\mathbf{Q},\mathbf{T}) = \frac{1}{|\mathcal{W}(\mathbf{T},L_c)|}\sum_{\mathbf{T}_w\in\mathcal{W}(\mathbf{T},L_c)} \mathbb{1}\left[\mathrm{NN}(\mathbf{T}_w,\mathbf{Q}) < \epsilon\right]. \quad (9)$$

$\mathrm{NN}(\mathbf{Q}_1,\mathbf{Q}_2)$ denotes the distance of the nearest neighbor of the animation sequence $\mathbf{Q}_1$ in $\mathbf{Q}_2$. It is crucial to use an appropriate distance measure here. In our setting, we choose the Frobenius norm on the local joint rotation matrices:

$$\mathrm{NN}(\mathbf{Q}_1,\mathbf{Q}_2) = \frac{1}{L_1}\min_{\mathbf{Q}_w\in\mathcal{W}(\mathbf{Q}_2,L_1)} \|\mathbf{Q}_1-\mathbf{Q}_w\|_F^2, \quad (10)$$

where L_1 is the length of $\mathbf{Q}_1$. We use joint rotations and not positions since our model creates local variations so that location deviations accumulate along the kinematics chain, which would cause location-based high distance measures on visually similar patches.

We choose $L_c = 30$, capturing local patch length of 1 second. Similarly, the coverage of a model $\mathcal{G}(\cdot)$ on $\mathbf{T}$ is defined by

$$\mathrm{Cov}(\mathcal{G},\mathbf{T}) = \mathbb{E}_z\mathrm{Cov}(\mathcal{G}(z),\mathbf{T}). \quad (11)$$

Global diversity. To quantitatively measure the global structure diversity against a single training example, we propose to measure the distance between patched nearest neighbors (PNN). The idea is to divide the generated animation into several segments, where each

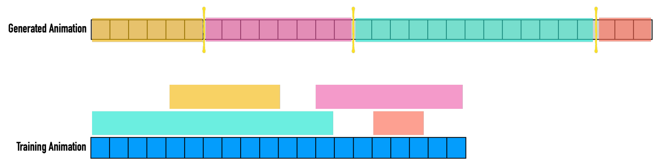

Fig. 6. Patched Nearest Neighbor metric. The generated animation (top row) is divided into several segments, where the length of each segment is at least $T_{\min}$. Then each segment is assigned to its nearest neighbor in the training animation (bottom row) as visualised by the color bars.

segment is no shorter than a threshold $T_{\min}$, and find a segmentation that minimizes the average per-frame nearest neighbor cost as illustrated in Figure 6. For each frame i in the generated animation $\mathbf{Q}$ we match it with frame l_i in the training animation $\mathbf{T}$, such that between every neighboring points in the set of discontinuous points $\{i \mid l_i \neq l_{i-1} + 1\}$ is at least $T_{\min}$. This is because every discontinuous point corresponds to the starting point of a new segment. We call such an assignment $\{l_i\}_{i=1}^{L}$ a *segmentation* on $\mathbf{Q}$. When large global structure variation is present, it is difficult to find a close nearest neighbor for large $T_{\min}$. The patched nearest neighbor is defined by, minimizing over all possible segmentations,

$$\mathcal{L}_{\text{PNN}} = \min_{\{l_i\}} \frac{1}{L} \sum_{i=1}^{L} \|\mathbf{Q}^i - \mathbf{T}^{l_i}\|_F^2, \tag{12}$$

where $\mathbf{Q}^i$ denotes frame i in Q and $\mathbf{T}^{l_i}$ denotes frame l_i in $\mathbf{T}$. The PNN can be solved efficiently with dynamic programming, similar to MotionTexture [2002]; we refer the reader to Appendix B for more details. In all our experiments, we choose $T_{\min} = 30$, corresponding to one second of the animation.

Local diversity. Our framework synthesizes animations carrying similar but *diverse* visual content compared to the training sequence. We measure the local frame diversity by comparing every local window $\mathbf{Q}_w \in \mathcal{W}(\mathbf{Q}, L_d)$ of length L_d to its nearest neighbor in the training sequence $\mathbf{T}$:

$$\mathcal{L}_{\text{local}} = \frac{1}{|\mathcal{W}(\mathbf{Q}, L_d)|} \sum_{\mathbf{Q}_w \in \mathcal{W}(\mathbf{Q}, L_d)} \text{NN}(\mathbf{Q}_w, \mathbf{T}). \tag{13}$$

Similar to the definition of coverage, we use the Frobenius norm over local joint rotation matrices. We choose $L_d = 15$ to capture *local* differences between the generated result and the training example.

We quantitatively compare our results to MotionTexture [Li et al. 2002] and acRNN [Zhou et al. 2018] using the metrics above as reported in Table 1. Since we use the nearest neighbor cost against the training sequence to measure diversity, high diversity score does not necessarily imply plausible results: MotionTexture creates unnatural transitions that are not part of the training sequence, and acRNN converges to a pose that does not exist in the training sequence also leading to a high diversity score. It can be seen that acRNN has limited coverage due to its convergence to a static pose, while our method generates motions that cover the training sequence well. MotionTexture trained as a single texton overfits to the training sequence, creating little variation on both local and global scale.

Table 1. Quantitative comparison to existing motion generation techniques.

	Coverage ↑	Global Diversity	Local Diversity
MotionTexture [2002]	84.6%	1.03	1.04
MotionTexture (Single)	100%	0.21	0.33
acRNN [2018]	11.6%	5.63	6.69
Ours	97.2%	1.29	1.19

Table 2. Quantitative comparison for ablation study.

	Rec. Loss ↓
W/o contact consist. loss	4.04
Euler angle	23.6
Quaternion	9.25
Full appraoch	2.85

Meanwhile, our model strikes a good balance between generating plausible motions and maintaining diversity.

4.4 Ablation study

We evaluate the impact of motion representation, temporal receptive field, and the foot contact consistency loss on our performance. The results are reported in Table 2 and demonstrated in the supplementary video.

Reconstruction loss. In this experiment, we discard the reconstruction loss and retrain our model. The supplementary video shows that the quality of the motion is degraded as a result. The reconstruction loss ensures that an anchor point in the latent space can be used to reconstruct the training sequence perfectly. Since our framework generates novel variations of the training sequence, this anchor point helps to stabilize the generated results. For the same reason, the reconstruction loss reflects the quality of generated results of our framework and we report the impact of different components on the quality of results in Table 2.

Contact consistency loss. In this experiment we discard the contact consistency loss $\mathcal{L}_{\text{con}}$ and retrain our model. Our framework predicts the foot contact label that can be used to fix sliding artifacts in a post-process. Although it can be learned implicitly, i.e., without the contact consistency loss, the generated result contains inconsistent global positions and contact labels, leading to unnatural leaning and transitions of contact status after the post-process. The accompanying video shows that the contact consistency loss promotes consistency between predicted animation and contact labels, providing a robust fix to sliding artifacts.

Rotation representation. In this experiment we use three different representations of joint rotations to train our networks: Euler angles, quaternions, and 6D representation [Zhou et al. 2019]. The results are shown in the accompanying video. It can be seen that the network struggles to generate reasonable results with Euler angles because of the extreme non-linearity. Quaternions yield more stable results compared to Euler angles, but the double-cover problem and

Table 3. Coverage measurement of motion mixing.

Training Set	Coverage of Seq. 1	Coverage of Seq. 2
Only Seq. 1	100.0%	1.5%
Only Seq. 2	2.4%	100%
All Seq.	100.0%	97.8%

the non-linearity still cause sudden undesired changes and degrade the realism of the generated motion.

Temporal receptive field. A key component of generating diverse and realistic motion with global variation is the choice of the temporal receptive field, which we explore in this experiment. We demonstrate the impact by training our framework on a clip of the YMCA dance in the accompanying video. We observe that when a large temporal receptive field is chosen, the network memorizes the global structure, creating a static pose in the middle of generation without being able to permute the dance sequence. When the temporal receptive field is small, the network observes only limited information about the context and generates jittering results. The network can generate smooth and plausible permutations of the dance when a suitable receptive field is picked.

5 APPLICATIONS

In this section we showcase the utility of pure generation and user-guided generation based on a single training example in different applications, including motion mixing, style transfer, key-frame editing, and interactive trajectory control.

Motion mixing. In addition to novel motion variation synthesis based on a single example animation, our framework can be also trained on several animations. Given N training animations $\{\mathbf{T}^k\}_{k=1}^N$, we can train a single generative network on all N examples by replacing the reconstruction loss in Eq. (6) with

$$\mathcal{L}'_{\text{rec}} = \frac{1}{N}\sum_{k=1}^{N} \|G_i(\uparrow \mathbf{T}^k_{i-1}, z_i^{k*}) - \mathbf{T}^k_i\|_1. \quad (14)$$

We train our model on two animation sequences of an elephant, shown in Fig. 7 and the video. We quantitatively measure the coverage of novel motion synthesis on two training animations in Table 3. It can be seen that the generated result of mixed training covers all the training examples well, and the synthesized output naturally fuses the two training sequences.

Style transfer. Our framework can perform motion style transfer using an input animation sequence whose style is applied to the content of another input animation. We exploit the hierarchical control of the generated content at different levels of detail. Since style is often expressed in relatively high frequencies, we use the downsampling of content input $\mathbf{T}^C$ to control the generation of the neural network trained on style input $\mathbf{T}^S$ for the style transfer task. Namely, we downsample the content animation $\mathbf{T}^C$ to the corresponding coarsest resolution $\mathbf{T}^C_1$ and use it to replace the output of the first level of the network trained on $\mathbf{T}^S$. We demonstrate our result in Fig. 8 and the video. It can be seen that, with a single

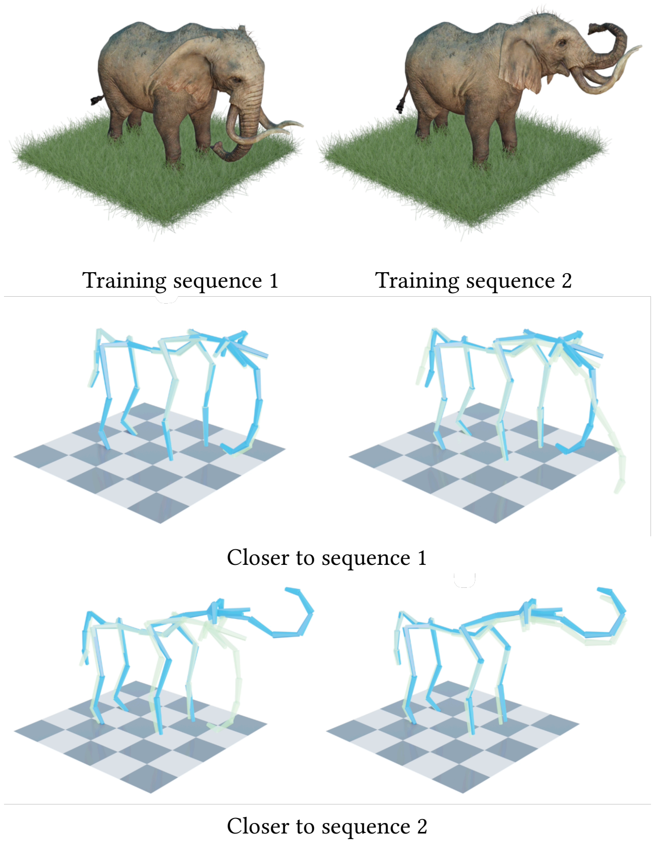

Fig. 7. The model is trained with two sequences. The first sequence (left) contains relative static motion, the second sequence (right) contains larger movement. We visualize the skeletal animation of our generated result (blue) and its patched nearest neighbor (green) in corresponding sequences. It can be seen that our result contains the content from both training sequence.

example, we successfully transfer the proud style, yielding a proud walk sharing the same pace as the content input. Due to the fact that the output is generated by multi-scale patches learned from the style input $\mathbf{T}^S$, the content of $\mathbf{T}^C$ is required to be similar to the content of $\mathbf{T}^S$, in order to generate high-quality results (e.g., both of the inputs are walking).

Key-frame editing. Our framework can also be applied to key-frame editing (Fig. 9). We train our network on the input animation $\mathbf{T}$. The user can then manually edit some frames by changing their poses at the coarsest level $\mathbf{T}_1$, and the network produces smooth, realistic, and highly-detailed transitions between the modified key-frames, including unseen but plausible content.

Conditional generation. Our framework can generate motion while accounting for user-specified constraints on the motion of selected joints. Formally, given the constrained joints C, the user-specified constraints $\mathbf{C}$ are given by $\mathbf{Q}^C = \{\mathbf{Q}^j,\ j \in C$, where $\mathbf{Q}^j$ denotes the motion of joint j. In the case of motion generation with trajectory control, we set $C = \{$root position, root orientation$\}$.

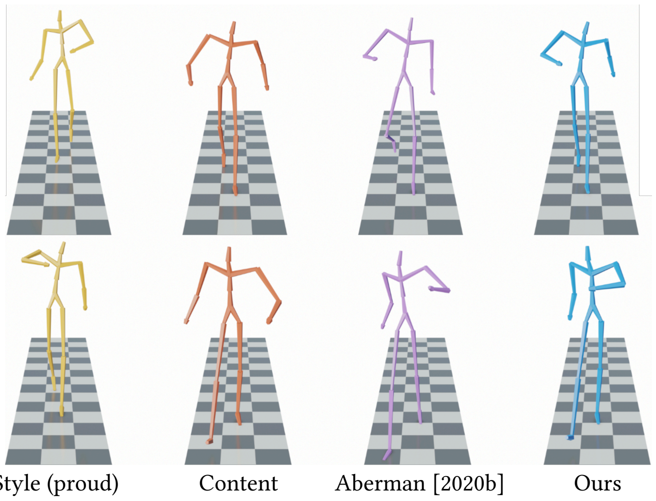

Fig. 8. We synthesize the proud style from the style input onto the content input. It can be seen that our result contains the same content as the content input while express the proud style, *e.g.* higher elbow position on walking. We also show the result from [Aberman et al. 2020b] for comparison.

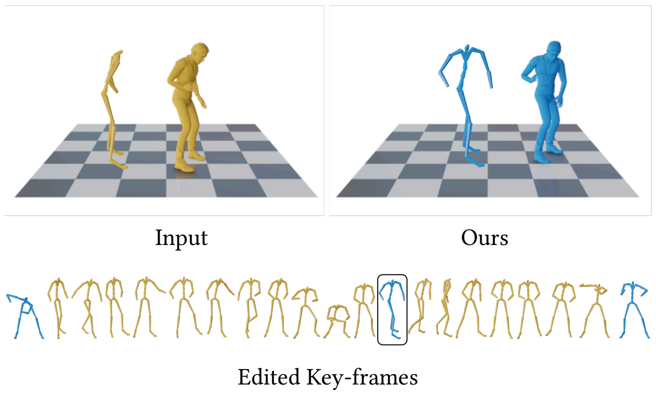

Fig. 9. We manually edit three key-frames (blue) in the input sequence by changing the poses. We visualize the result of one editing, where the character is made to face the audience. It can be seen that our model follows the editing and generates a plausible result. Please refer to the accompanying video for a complete result.

The key component of enabling conditional generation is enforcing correlation between the generated motion and the constraints. The training process of the conditional generation is described in Fig. 10. As constraints are defined as part of the motion, we use the *concatenation trick*, denoted by $concat(\mathbf{Q}, \mathbf{C})$, during the training and inference time, where $concat(\mathbf{Q}, \mathbf{C})$ is the result of replacing the motion of constrained joints $\mathbf{Q}^C$ in motion $\mathbf{Q}$ by the constraints $\mathbf{C}$.

For the evaluation of the i-th level, given the constraints at corresponding frame rate $\mathbf{C}_i$, we use $concat(\uparrow \mathbf{Q}_{i-1}, \mathbf{C}_i)$ as the input of the generator instead of the upsampled result $\uparrow \mathbf{Q}_{i-1}$ from the previous level. For training, we randomly generate $\mathbf{C}_i$ by sampling from a pre-trained generator mentioned above, taking the corresponding part of the motion and downsampling it to the corresponding frame rate. We employ the concatenation trick again on the output of the generator and let $\mathbf{Q}_i = concat(G_i(concat(\uparrow \mathbf{Q}_{i-1}, \mathbf{C}_i), z_i), \mathbf{C}_i)$ as the output of this stage and feed it into the discriminator. The discriminator rules out motions that do not belong to the plausible motion distribution and forces the network to generate motions complying with the given constraints.

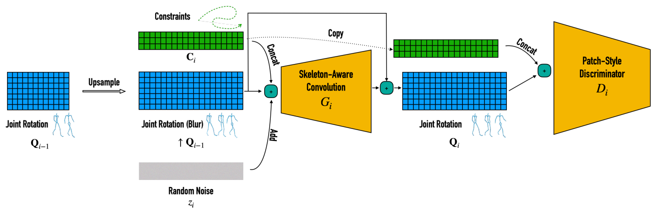

Fig. 10. Training of conditional generation. To construct the input for the conditional generator, we concatenate the constraints (e.g., root joint movement) sampled from a pre-trained generator to the upsampled result generated with the same condition from the previous level. We again concatenate the constraints to the result produced by the generator and feed it to the discriminator. The discriminator ensures the correlation between constraints and the generated result.

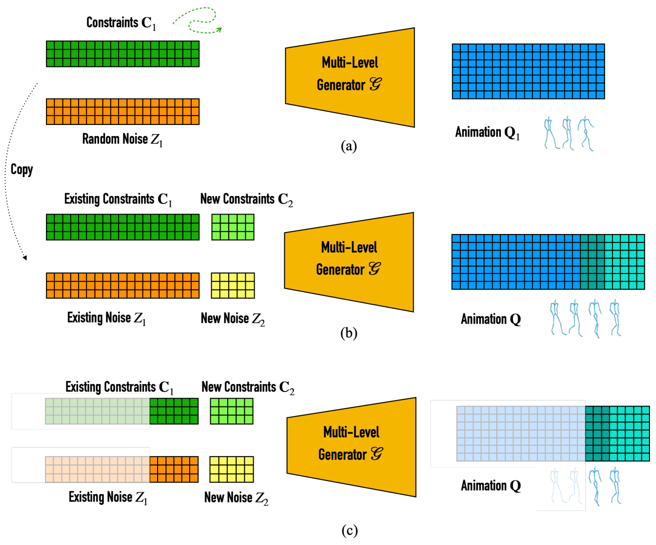
Constraints $\mathbf{C}_1$
Random Noise Z_1
Multi-Level Generator $\mathcal{G}$
Animation $\mathbf{Q}_1$
(a)
Copy
Existing Constraints $\mathbf{C}_1$
New Constraints $\mathbf{C}_2$
Existing Noise Z_1
New Noise Z_2
Multi-Level Generator $\mathcal{G}$
Animation $\mathbf{Q}$
(b)
Existing Constraints $\mathbf{C}_1$
New Constraints $\mathbf{C}_2$
Existing Noise Z_1
New Noise Z_2
Multi-Level Generator $\mathcal{G}$
Animation $\mathbf{Q}$
(c)

Fig. 11. Interactive Generation. (a) Our conditional generation framework can be conceptually simplified as a multi-layer *convolutional* generator that takes user-specified constraints e.g., root joint movement, and random noise as input to generate an animation. (b) When new constraints are given, we concatenate them with the existing constraints and noise as the input for the generator. In the generated result, the frames that are outside of the receptive field of new constraints remain the same (blue area). The frames within the receptive field of new constraints are changed and are used to create a smooth transition between existing and new constraints (dark cyan). The frames complying with new constraints are generated (cyan). (c) During a generation, we only need to keep the frames within the receptive field of the dark cyan area, denoted by activated colors.

Interactive generation. Conditional generation is particularly interesting for interactive applications, such as video games, where the motion of a character needs to be generated online in accordance to joystick controller inputs, for example. We demonstrate the process for exploiting a pre-trained conditional generation framework for such interactive application in Fig. 11. Unlike RNNs that are straightforward to use for interactive or real-time generation, convolutional neural networks generally require modifications, such as causal convolution [Oord et al. 2016]. We take a different approach by exploiting the limited receptive field of convolution.

Let $\mathcal{G}(\mathbf{C}, Z)$ be the multi-level conditional generation result with constraints $\mathbf{C}$, where Z is the set of random noise $\{z_i\}_{i=1}^{S}$ used during generation. In the interactive setting, we assume there are existing constraints $\mathbf{C}_1$ and the corresponding random noise Z_1, which generates the result $\mathbf{Q}_1 = \mathcal{G}(\mathbf{C}_1, Z_1)$, as demonstrated in Fig. 11(a). Given the new constraints $\mathbf{C}_2$ and corresponding noise Z_2, we concatenate them with $\mathbf{C}_1$ and Z_1 along the temporal axis, denoted by $\mathbf{C} = ext(\mathbf{C}_1, \mathbf{C}_2)$ and $Z = ext(Z_1, Z_2)$, and generate a new result $\mathbf{Q} = \mathcal{G}(\mathbf{C}, Z)$. Denote the temporal receptive field of $\mathcal{G}(\cdot)$ by R and the lengths of constraints $\mathbf{C}_1$ and $\mathbf{C}_2$ by L_1 and L_2, respectively. Note that $\mathbf{Q}^{1:L_1-r}$ and $\mathbf{Q}_1^{1:L_1-r}$ are equal (blue area in Fig. 11(b)), where $\mathbf{Q}^{i:j}$ denotes frame i to j of motion $\mathbf{Q}$ and $r = \lceil R/2 \rceil$ is the halved receptive field. The sequence $\mathbf{Q}^{L_1-r+1:L_1}$ is different from $\mathbf{Q}_1^{L_1-r+1:L_1}$ and creates a smooth transition to the new constraints (dark cyan area in Fig. 11(b)). The remaining part $\mathbf{Q}^{L_1+1:L_1+L_2}$ is the newly generated result complying with the new constraints $\mathbf{C}_2$ (cyan area in Fig. 11(b)).

Therefore, given the new constraints, we only need to run the convolutional network on the concatenation of the last $2r$ frames of the existing constraints and the new constraints, as demonstrated by the activated area in Fig. 11(c), and withhold the last r frames from displaying on the screen. The full method for interactive generation is summarized in Algorithm 1. We demonstrate interactive trajectory control in Fig. 12 and in the accompanying video.

Algorithm 1 Interactive Generation

```
$\mathbf{C}_1 \leftarrow$ initial constraints
$Z_1 \leftarrow$ inital generated noise
$r \leftarrow$ halved receptive field of $\mathcal{G}(\cdot)$
$\mathbf{Q} \leftarrow \mathcal{G}(\mathbf{C}_1, Z_1)$
*Display* $\mathbf{Q}^{1:\mathrm{end}-r}$
**while** $\mathbf{C}_2 \leftarrow$ new constraints **do**
    $Z_2 \leftarrow$ generated noise
    $\mathbf{C} \leftarrow ext(\mathbf{C}_1, \mathbf{C}_2)$
    $Z \leftarrow ext(Z_1, Z_2)$
    $\mathbf{Q} \leftarrow \mathcal{G}(\mathbf{C}, Z)$
    *Display* $\mathbf{Q}^{r+1:\mathrm{end}-r}$
    $\mathbf{C}_1 \leftarrow \mathbf{C}^{\mathrm{end}-2r+1:\mathrm{end}}$
    $Z_1 \leftarrow Z^{\mathrm{end}-2r+1:\mathrm{end}}$
**end while**
```

6 DISCUSSION AND CONCLUSION

In this work we presented a neural motion synthesis approach that leverages the power of neural networks to exploit the information available within a *single* motion sequence. We demonstrated the utility of our framework on a variety of applications, including synthesizing longer motion sequences of the same essence but with plausible variations, controlled motion synthesis, key-frame editing and interpolation, and style transfer. Despite the fact that motion data is irregular, we presented a neural representation and system for effectively learning to synthesize motion. Key to our technique is a combination of skeleton-aware convolutional operators, which serve as a backbone for a progressive motion synthesis framework.

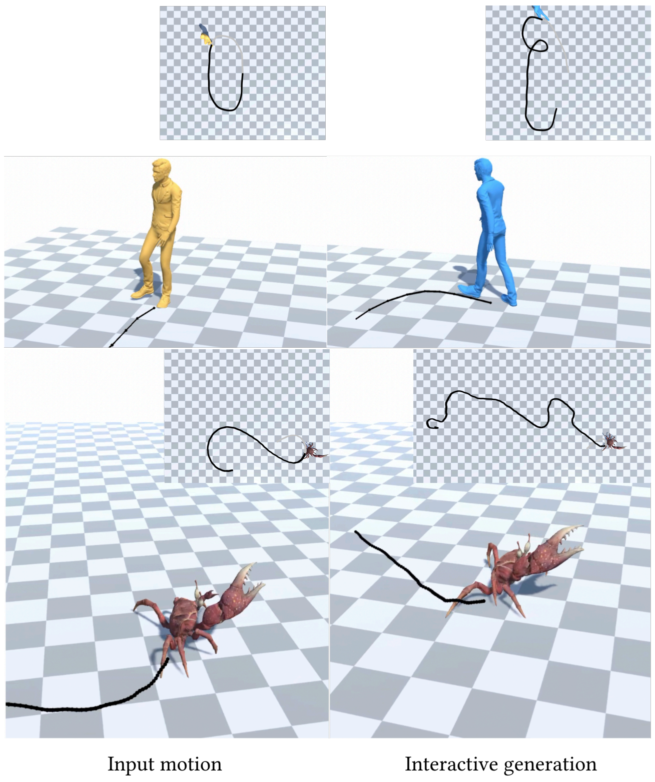

Fig. 12. We demonstrate the interactive trajectory control on humanoid and a hexapod crab. It can be seen that our model can cope with diverse input trajectories despite the single training sequence.

While our current framework enables interactive control of the synthesized motion trajectory, the motion essence itself has to be learned offline, in advance. In addition, our current system's physical plausibility is limited to simple skeletal kinematics and foot contact handling. It would be interesting to incorporate higher level physics to enable synthesis of interactions and motions in complex environments. An interesting future direction is to explore online motion learning through sparse demonstrations, for example using continual learning. It is conceivable that our method could be used for data augmentation to benefit training of elaborate frameworks that require large datasets. It is also possible to explore applications of our ideas in non-skeletal settings, such as facial and other non-skeletal rigs. Our current approach does not involve the interplay of motion and shape deformation, although the latter is also an essential part of realistic character animation (e.g., geometric ground contact for creatures like snake demonstrated in the supplementary video), and we are keen on exploring this intersection in future research.

ACKNOWLEDGMENTS

We thank Andreas Aristidou and Sigal Raab for their valuable inputs that helped to improve this work. This work is supported in part by the European Research Council (ERC) Consolidator grant no. 101003104.

REFERENCES

Kfir Aberman, Peizhuo Li, Dani Lischinski, Olga Sorkine-Hornung, Daniel Cohen-Or, and Baoquan Chen. 2020a. Skeleton-aware networks for deep motion retargeting. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 62–1.

Kfir Aberman, Yijia Weng, Dani Lischinski, Daniel Cohen-Or, and Baoquan Chen. 2020b. Unpaired motion style transfer from video to animation. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 64–1.

Kfir Aberman, Rundi Wu, Dani Lischinski, Baoquan Chen, and Daniel Cohen-Or. 2019. Learning Character-Agnostic Motion for Motion Retargeting in 2D. *ACM Trans. Graph.* 38, 4 (2019), 75.

Adobe Systems Inc. 2021. Mixamo. https://www.mixamo.com Accessed: 2021-12-25.

Shailen Agrawal, Shuo Shen, and Michiel van de Panne. 2013. Diverse motion variations for physics-based character animation. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* 37–44.

Okan Arikan and David A Forsyth. 2002. Interactive motion generation from examples. *ACM Transactions on Graphics (TOG)* 21, 3 (2002), 483–490.

Andreas Aristidou, Anastasios Yiannakidis, Kfir Aberman, Daniel Cohen-Or, Ariel Shamir, and Yiorgos Chrysanthou. 2021. Rhythm is a Dancer: Music-Driven Motion Synthesis with Global Structure. *arXiv preprint arXiv:2111.12159* (2021).

Richard Bowden. 2000. Learning statistical models of human motion. In *IEEE Workshop on Human Modeling, Analysis and Synthesis, CVPR*, Vol. 2000. Citeseer.

Matthew Brand and Aaron Hertzmann. 2000. Style machines. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques.* 183–192.

Michael Büttner and Simon Clavet. 2015. Motion Matching-The Road to Next Gen Animation. *Proc. of Nucl. ai* 2015 (2015). https://www.youtube.com/watch?v=z_wpgHFSWss&t=658s

Jinxiang Chai and Jessica K Hodgins. 2007. Constraint-based motion optimization using a statistical dynamic model. In *ACM SIGGRAPH 2007 papers.* 8–es.

Katerina Fragkiadaki, Sergey Levine, Panna Felsen, and Jitendra Malik. 2015. Recurrent network models for human dynamics. In *Proceedings of the IEEE International Conference on Computer Vision.* 4346–4354.

Thomas Geijtenbeek, Nicolas Pronost, Arjan Egges, and Mark H. Overmars. 2011. Interactive Character Animation using Simulated Physics. In *Eurographics 2011 - State of the Art Reports*, N. John and B. Wyvill (Eds.). The Eurographics Association. https://doi.org/10.2312/EG2011/stars/127-149

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. *Advances in neural information processing systems* 27 (2014).

Keith Grochow, Steven L Martin, Aaron Hertzmann, and Zoran Popović. 2004. Style-based inverse kinematics. In *ACM SIGGRAPH 2004 Papers.* 522–531.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. 2017. Improved training of wasserstein GANs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems.* 5769–5779.

Félix G Harvey, Mike Yurick, Derek Nowrouzezahrai, and Christopher Pal. 2020. Robust motion in-betweening. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 60–1.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 770–778.

Rachel Heck and Michael Gleicher. 2007. Parametric motion graphs. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games.* 129–136.

Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, et al. 2017. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286* (2017).

Gustav Eje Henter, Simon Alexanderson, and Jonas Beskow. 2020. Moglow: Probabilistic and controllable motion synthesis using normalising flows. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–14.

Tobias Hinz, Matthew Fisher, Oliver Wang, and Stefan Wermter. 2021. Improved techniques for training single-image gans. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision.* 1300–1309.

Daniel Holden, Oussama Kanoun, Maksym Perepichka, and Tiberiu Popa. 2020. Learned motion matching. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 53–1.

Daniel Holden, Taku Komura, and Jun Saito. 2017. Phase-functioned neural networks for character control. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–13.

Daniel Holden, Jun Saito, and Taku Komura. 2016. A deep learning framework for character motion synthesis and editing. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–11.

Daniel Holden, Jun Saito, Taku Komura, and Thomas Joyce. 2015. Learning motion manifolds with convolutional autoencoders. In *SIGGRAPH Asia 2015 Technical Briefs.* 1–4.

Leslie Ikemoto, Okan Arikan, and David Forsyth. 2009. Generalizing motion edits with gaussian processes. *ACM Transactions on Graphics (TOG)* 28, 1 (2009), 1–12.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. 2017. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 1125–1134.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. 2018. Progressive Growing of GANs for Improved Quality, Stability, and Variation. In *International Conference on Learning Representations*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

Lucas Kovar and Michael Gleicher. 2004. Automated extraction and parameterization of motions in large data sets. *ACM Transactions on Graphics (ToG)* 23, 3 (2004), 559–568.

Lucas Kovar, Michael Gleicher, and Frédéric Pighin. 2002. Motion Graphs. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '02).* Association for Computing Machinery, New York, NY, USA, 473–482. https://doi.org/10.1145/566570.566605

Manfred Lau, Ziv Bar-Joseph, and James Kuffner. 2009. Modeling spatial and temporal variation in motion data. *ACM Transactions on Graphics (TOG)* 28, 5 (2009), 1–10.

Jehee Lee, Jinxiang Chai, Paul SA Reitsma, Jessica K Hodgins, and Nancy S Pollard. 2002. Interactive control of avatars animated with human motion data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques.* 491–500.

Kyungho Lee, Seyoung Lee, and Jehee Lee. 2018. Interactive character animation by learning multi-objective control. *ACM Transactions on Graphics (TOG)* 37, 6 (2018), 1–10.

Seyoung Lee, Sunmin Lee, Yongwoo Lee, and Jehee Lee. 2021. Learning a family of motor skills from a single motion clip. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–13.

Sergey Levine, Jack M Wang, Alexis Haraux, Zoran Popović, and Vladlen Koltun. 2012. Continuous character control with low-dimensional embeddings. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 1–10.

Chuan Li and Michael Wand. 2016. Precomputed real-time texture synthesis with markovian generative adversarial networks. In *European conference on computer vision.* Springer, 702–716.

Yan Li, Tianshu Wang, and Heung-Yeung Shum. 2002. Motion texture: a two-level statistical model for character motion synthesis. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques.* 465–472.

Ying-Sheng Luo, Jonathan Hans Soeseno, Trista Pei-Chun Chen, and Wei-Chao Chen. 2020. Carl: Controllable agent with reinforcement learning for quadruped locomotion. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 38–1.

Ian Mason, Sebastian Starke, and Taku Komura. 2022. Real-Time Style Modelling of Human Locomotion via Feature-Wise Transformations and Local Motion Phases. *arXiv preprint arXiv:2201.04439* (2022).

Jianyuan Min and Jinxiang Chai. 2012. Motion graphs++ a compact generative model for semantic motion analysis and synthesis. *ACM Transactions on Graphics (TOG)* 31, 6 (2012), 1–12.

Mark Mizuguchi, John Buchanan, and Tom Calvert. 2001. Data driven motion transitions for interactive games.. In *Eurographics (Short Presentations).*

Lucas Mourot, Ludovic Hoyet, François Le Clerc, François Schnitzler, and Pierre Hellier. 2021. A Survey on Deep Learning for Skeleton-Based Human Animation. *Computer Graphics Forum* (2021).

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499* (2016).

Sang Il Park, Hyun Joon Shin, and Sung Yong Shin. 2002. On-line locomotion generation based on motion blending. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation.* 105–111.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Dario Pavllo, David Grangier, and Michael Auli. 2018. Quaternet: A quaternion-based recurrent model for human motion. *arXiv preprint arXiv:1805.06485* (2018).

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. 2018. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–14.
Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. 2017. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–13.
Ken Perlin. 1985. An image synthesizer. *ACM Siggraph Computer Graphics* 19, 3 (1985), 287–296.
Ken Perlin and Athomas Goldberg. 1996. Improv: A system for scripting interactive actors in virtual worlds. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques.* 205–216.
Katherine Pullen and Christoph Bregler. 2000. Animating by multi-level sampling. In *Proceedings Computer Animation 2000.* IEEE, 36–42.
Katherine Pullen and Christoph Bregler. 2002. Motion capture assisted animation: Texturing and synthesis. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques.* 501–508.
Charles Rose, Michael F Cohen, and Bobby Bodenheimer. 1998. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications* 18, 5 (1998), 32–40.
Charles Rose, Brian Guenter, Bobby Bodenheimer, and Michael F Cohen. 1996. Efficient generation of motion transitions using spacetime constraints. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques.* 147–154.
Alla Safonova and Jessica K Hodgins. 2007. Construction and optimal search of interpolated motion graphs. In *ACM SIGGRAPH 2007 papers.* 106–es.
Tamar Rott Shaham, Tali Dekel, and Tomer Michaeli. 2019. SinGAN: Learning a generative model from a single natural image. In *Proceedings of the IEEE/CVF International Conference on Computer Vision.* 4570–4580.
Assaf Shocher, Shai Bagon, Phillip Isola, and Michal Irani. 2019. Ingan: Capturing and retargeting the" dna" of a natural image. In *Proceedings of the IEEE/CVF International Conference on Computer Vision.* 4492–4501.
Sebastian Starke, Yiwei Zhao, Taku Komura, and Kazi Zaman. 2020. Local motion phases for learning multi-contact character movements. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 54–1.
Sebastian Starke, Yiwei Zhao, Fabio Zinno, and Taku Komura. 2021. Neural animation layering for synthesizing martial arts movements. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–16.
Luis Molina Tanco and Adrian Hilton. 2000. Realistic synthesis of novel human movements from a database of motion capture examples. In *Proceedings Workshop on Human Motion.* IEEE, 137–142.
Graham W Taylor and Geoffrey E Hinton. 2009. Factored conditional restricted Boltzmann machines for modeling motion style. In *Proceedings of the 26th annual international conference on machine learning.* 1025–1032.
Truebones Motions Animation Studios. 2022. Truebones. https://truebones.gumroad.com/ Accessed: 2022-1-15.
Ruben Villegas, Jimei Yang, Duygu Ceylan, and Honglak Lee. 2018. Neural kinematic networks for unsupervised motion retargetting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 8639–8648.
Jack M Wang, David J Fleet, and Aaron Hertzmann. 2007. Gaussian process dynamical models for human motion. *IEEE transactions on pattern analysis and machine intelligence* 30, 2 (2007), 283–298.
Xiaolin Wei, Jianyuan Min, and Jinxiang Chai. 2011. Physically valid statistical models for human motion generation. *ACM Transactions on Graphics (TOG)* 30, 3 (2011), 1–10.
Douglas J Wiley and James K Hahn. 1997. Interpolation synthesis of articulated figure motion. *IEEE Computer Graphics and Applications* 17, 6 (1997), 39–45.
Yuting Ye and C Karen Liu. 2010. Synthesis of responsive motion using a dynamic model. In *Computer Graphics Forum*, Vol. 29. Wiley Online Library, 555–562.
He Zhang, Sebastian Starke, Taku Komura, and Jun Saito. 2018. Mode-adaptive neural networks for quadruped motion control. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–11.
Liming Zhao, Aline Normoyle, Sanjeev Khanna, and Alla Safonova. 2009. Automatic construction of a minimum size motion graph. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics symposium on Computer animation.* 27–35.
Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. 2019. On the continuity of rotation representations in neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 5745–5753.
Yi Zhou, Zimo Li, Shuangjiu Xiao, Chong He, Zeng Huang, and Hao Li. 2018. Auto-Conditioned Recurrent Networks for Extended Complex Human Motion Synthesis. In *International Conference on Learning Representations.*

A NETWORK ARCHITECTURES

In this section, we describe the details for the network architectures.

Table 4 describes the architecture for our generator and discriminator network for a single level, where `Conv` and `LReLU` denote skeleton-aware convolution [Aberman et al. 2020a] and leaky ReLU activation respectively. All the convolution layers use reflected padding, kernel size 5 and neighbor distance 2. For simplicity, we denote the number of input animation features $JQ + 3 + |\mathcal{F}|$ by F_0.

In our experiments, we use $\lambda_{\text{adv}} = 1, \lambda_{\text{rec}} = 50, \lambda_{\text{con}} = 5$ and $\lambda_{\text{gp}} = 1$.

Name	Layers	in/out channels
Generator	`Conv + LReLU`	F_0/F_0
	`Conv + LReLU`	$F_0/2F_0$
	`Conv + LReLU`	$2F_0/2F_0$
	`Conv`	$2F_0/F_0$
Discriminator	`Conv + LReLU`	F_0/F_0
	`Conv + LReLU`	$F_0/2F_0$
	`Conv + LReLU`	$2F_0/2F_0$
	`Conv`	$2F_0/1$

Table 4. Network Architectures

B SOLVING PATCHED NEAREST NEIGHBOR

In this section, we describe how to solve the patched nearest neighbor (PNN). Given the generated animation $\mathbf{Q}$ of length L_Q and the training animation $\mathbf{T}$ of length L_T, we search for the corresponding segmentation $\{l_i\}_{i=1}^{L_Q}$ for each frame in $\mathbf{Q}$, such that the minimum distance of any two points in the discontinuous point set $\{i \mid l_i \neq l_{i-1} + 1\}$ is no less than $T_{\min}$.

Let $D(i)$ denotes the PNN for first i frames, with boundary condition $D(0) = 0$. We can solve other $D(i)$ by

$$D(i) = \min_{0 \le j \le i - T_{\min}, 1 \le k \le L_T - i + j} D(j) + \text{cost}(j, i, k) \quad (15)$$

$$J(i), K(i) = \operatorname*{arg\,min}_{0 \le j \le i - T_{\min}, 1 \le k \le L_T - i + j} D(j) + \text{cost}(j, i, k) \quad (16)$$

$$\text{cost}(j, i, k) = \sum_{m=1}^{i-j} \|\mathbf{Q}^{j+m} - \mathbf{T}^{k+m}\|_2^2, \quad (17)$$

where $\text{cost}(j, i, k)$ denotes the distance between $\mathbf{Q}^{j+1:i}$ and $\mathbf{T}^{k+1:k+i-j}$. After solving $D(i), J(i), K(i)$, the label $\{l_i\}$ can be backtraced by Algorithm 2.

Algorithm 2 Backtracing Label

```
$p \leftarrow L_Q$
while $p > 0$ do
    $l_{J(p)+1:p} \leftarrow K(p) + 1, K(p) + 2, \cdots, K(p) + p - J(p)$
    $p \leftarrow J(p)$
end while
```

The $\text{cost}(j, i, k)$ can be precomputed with time complexity $O(L_Q^2 L_T)$. The dynamic programming for solving $D(i)$ also runs with time complexity $O(L_Q^2 L_T)$. The PNN cost $\mathcal{L}_{\text{PNN}}$ is given by $D(L_Q)/L_Q$.