Interactive Character Animation by Learning Multi-Objective Control

KYUNGHO LEE, Seoul National University, South Korea
SEYOUNG LEE, Seoul National University, South Korea
JEHEE LEE*, Seoul National University, South Korea

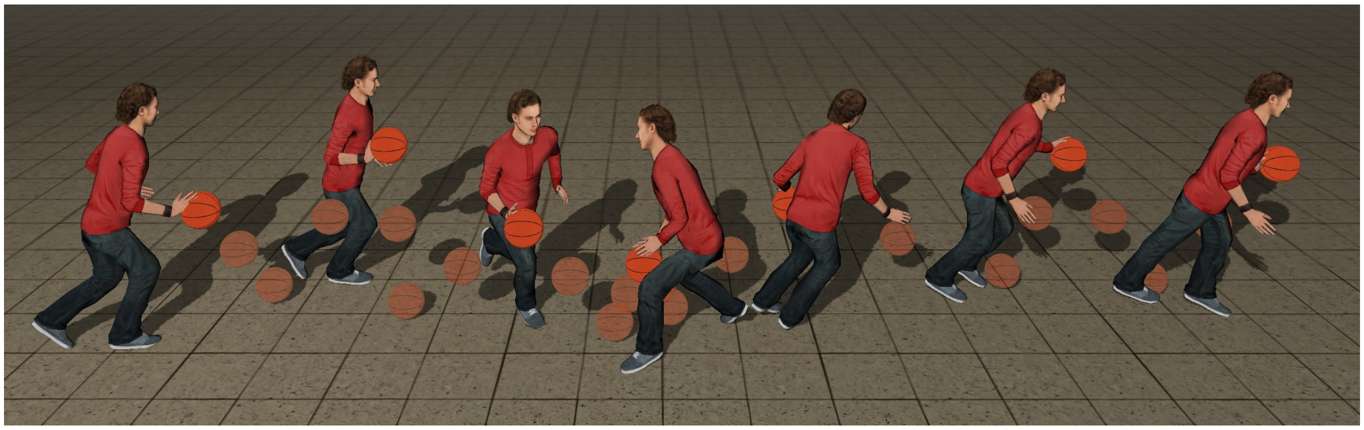

Fig. 1. Recurrent neural network learned basketball rules and skills for interactive character animation.

We present an approach that learns to act from raw motion data for interactive character animation. Our motion generator takes a continuous stream of control inputs and generates the character's motion in an online manner. The key insight is modeling rich connections between a multitude of control objectives and a large repertoire of actions. The model is trained using Recurrent Neural Network conditioned to deal with spatiotemporal constraints and structural variabilities in human motion. We also present a new data augmentation method that allows the model to be learned even from a small to moderate amount of training data. The learning process is fully automatic if it learns the motion of a single character, and requires minimal user intervention if it deals with props and interaction between multiple characters.

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

Additional Key Words and Phrases: Character animation, interactive motion control, motion grammar, deep learning, recurrent neural network, multi-objective control

*Corresponding author

Authors' addresses: Kyungho Lee, Seoul National University, 1 Gwanak-ro, Gwanak-gu, Seoul, 08826, South Korea, khlee@mrl.snu.ac.kr; Seyoung Lee, Seoul National University, 1 Gwanak-ro, Gwanak-gu, Seoul, 08826, South Korea, seyounglee@mrl.snu.ac.kr; Jehee Lee, Seoul National University, 1 Gwanak-ro, Gwanak-gu, Seoul, 08826, South Korea, jehee@cse.snu.ac.kr.

ACM Reference Format:
Kyungho Lee, Seyoung Lee, and Jehee Lee. 2018. Interactive Character Animation by Learning Multi-Objective Control. *ACM Trans. Graph.* 37, 6, Article 180 (November 2018), 10 pages. https://doi.org/10.1145/3272127.3275071

1 INTRODUCTION

Interactive character animation is an important issue in computer graphics. Among the many technical aspects of character animation, we are particularly interested in two key challenges: how to make virtual characters behave like humans, and how to control them as we want. Humans move in diverse, but highly structured ways. Various factors such as habitual patterns, social customs, sports rules and dance choreography contribute to the formation of behavioral structures. Capturing, modeling and reproducing such a structure for virtual characters is a challenging problem. For example, the movement of our virtual basketball player is governed by basketball regulations, which describe how many steps the player can take while holding a ball, when pivoting is allowed, and so on.

Interaction with virtual characters includes specifying spatiotemporal conditions such as "where to go", "when to arrive", "what to do", and "how to move". The virtual player may be controlled, for example, to go to the corner of the basketball court, pass the ball to another player, receive the ball back, and shoot the ball to the rim. Given the sequence of conditions, the virtual player has to plan its actions in compliance with basketball regulations. The problem becomes even more challenging if multiple players have to collaborate and synchronize their movements while interacting with each other. This problem has previously been addressed using either state space search [Kovar et al. 2002; Lee et al. 2002; Safonova and Hodgins 2007] or mixed discrete-continuous optimization [Hyun et al. 2016; Kim et al. 2012; Won et al. 2014]. Both approaches are computationally demanding and thus cannot achieve interactive performance.

Recently, deep learning and neural networks have received a great deal of attention as a means of modeling, predicting, recognizing, and synthesizing human movements [Fragkiadaki et al. 2015; Holden et al. 2017; Martinez et al. 2017; Taylor and Hinton 2009; Zhang et al. 2018]. Learning to act by interacting with the environment in the framework of interactive character animation has not been fully addressed yet. Issues of technical interest include dealing with a diversity of action repertoires, a multitude of dynamically specified goals, and continuous control over space and time.

In this paper, we present a new supervised learning approach that takes a continuous stream of control inputs to generate character animation interactively. The common practice of supervised learning requires a collection of example input-output pairs. In our case, the input corresponds to the intention (or objective) of action while the output is the actual motion the character takes. Although large motion databases are readily available, the motion data only record the actions taken by an actor but his/her intention is not explicitly represented. Our learning model infers the intention of action from raw motion data to build rich connections between a multitude of control objectives and a large repertoire of actions.

It is well-known that Recurrent Neural Network (RNN) is remarkably adept at modeling sequential data. Our motion generator is a multi-layered RNN conditioned to deal with spatiotemporal constraints and rich structural variabilities in human motion. Our experiments demonstrate that spatiotemporally conditioned RNN can not only capture underlying structures of human motion but also provide precise control over interactive characters. Our approach shares a number of advantages with existing data-driven, supervised learning approaches [Holden et al. 2017; Zhang et al. 2018]. Our motion generator is suitable for real-time interactive applications since it generates motion frames one-by-one in an online manner without delay. The motion generator is compact since the training data can be discarded after learning. Secondary motion effects and props can also be learned together with the character's full-body motion.

We also present a new data augmentation method that allows our model to be learned even from a small amount of training data. The augmentation method systematically enriches spatial, temporal, and combinatorial variabilities in the training data. The augmentation process is fully automatic with the motion of a single character, while the process requires minimal user intervention to deal with props and interaction between multiple characters.

2 RELATED WORK

In this section, we will review data-driven approaches studied in computer graphics. The key idea of data-driven approaches is to utilize a collection of high-quality motion data to generate new motions potentially with new target characters and new target environments. The early work focused on building data structures or computational models of human movements to support efficient motion generation, search, and manipulation. The popular idea based on *motion graphs* allows transitioning between motion frames to generate a novel sequence of motions [Kovar et al. 2002; Lee et al. 2002]. Temporally aligning and blending a collection of similar motions can generate a family of parametrized motions [Kovar and Gleicher 2004; Shin and Oh 2006]. Alternatively, a collection of similar motions can be used to build a statistical model based on principle component analysis [Min and Chai 2012] and Gaussian Process Latent Variable Models (GPLVMs) [Grochow et al. 2004; Levine et al. 2012]. The motion graph is equivalent to a finite state machine, if we consider each individual motion frame (or motion segment) as a symbol. The expressive power of a finite state machine is limited and thus can describe only very simple structures in diverse human behaviors. Hyun et al [2016] studied *motion grammars*, which are essentially context-free grammars augmented with spatiotemporal conditions, to model complex behavioral structures. Motion grammars and the syntax of human activities have also been studied in the context of computer vision, linguistics, and robotics [Dantam and Stilman 2013; Guerra-Filho and Aloimonos 2012].

Directing a character to move to a goal position or along a target trajectory requires appropriate control and/or planning algorithms together with the computational model of human movements. Character animation has often been formulated as state space search in high-dimensional configuration space and addressed using the $A^\star$ algorithm or its variants [Kovar et al. 2002; Lee et al. 2002; Safonova and Hodgins 2007]. State space search based on branch-and-bound is notorious for its heavy computational burden, which can be mitigated by various types of precomputation on state space, action space, and environment configuration space. Lee and Lee [2004] introduced reinforcement learning methods into computer graphics community. The key idea of reinforcement learning is to learn state value functions during the learning phase such that the optimal action at each individual state can be determined immediately at runtime without state space search. Treuille *et al.* [2007] employed linear function approximators to compactly represent state value functions. The revolutionary development of deep learning replaced linear function approximators with deep neural networks to achieve dramatically improved representation power and learning capacity [Mnih and Kavukcuoglu 2015]. The problem can alternatively be formulated from the viewpoint of path planning. Popular path planning algorithms using probabilistic roadmaps and rapidly-growing random trees have been exploited for full-body character animation in virtual environments with obstacles and dynamic environment features [Choi et al. 2011; Levine et al. 2011].

Spatiotemporal conditions play a key role of simulating multi-character interactions. Various approaches have been studied to spatially align and temporally synchronize the motion of multiple characters interacting with each other. *Motion patches* [Lee et al. 2006] were originally invented to simulate character-object interactions. A motion patch is a fragment of character animation that captures how a character interact with an environment object. Given a collection of motion patches, character animation is simply a spatiotemporal tiling of motion patches. The concept of motion patches have evolved to deal with interpersonal interactions as well [Kim et al. 2012; Shum et al. 2008]. Multi-character interactions can be specified through various forms of user interfaces. Won *et al.* [2014] translated a drama script into an *event graph* and used a generate-and-rank approach to synthesize animated scenes in which multiple characters interact with each other. Hyun *et al.* [2016] generated the animation of basketball plays from simple drawings on a tactic board. The previous studies in this category using a collection of canned motion segments and/or motion patches commonly formulated the problem as mixed optimization of combinatorial planning and continuous optimization. Deciding the sequence of motion segments is a discrete, combinatorial problem, while precisely aligning spatial locations and timings is a continuous optimization problem. The mixed optimization has previously been addressed using stochastic methods, such as simulated annealing and MCMC sampling, which are computationally expensive. The previous studies reported that animating dozens of characters for 10 to 20 seconds takes several minutes to several hours of computation on a typical PC.

A new class of machine learning algorithms exploiting the expressive power of deep neural networks have successfully been applied to character animation. Holden *et al.* [2016] learned biped locomotion along target root trajectories using Convolutional Neural Networks (CNNs). In the next year, they utilized a fully-connected neural network with a latent phase variable to interactively generate biped locomotion on uneven terrain. The latent phase variable is useful to align cyclic movements in the training data and thus accelerate the progress of learning. Zhang *et al.* [2018] generalized the phase-functioned architecture further to deal with multiple modes, which allow the model to learn various gaits in quadrupled locomotion. Deep reinforcement learning that incorporates deep neural networks into the framework of reinforcement learning as function approximators has been particularly successful for physically based simulation and control of biped locomotion [Liu and Hodgins 2017; Peng et al. 2018, 2017; Yu et al. 2018] and flapping flight [Won et al. 2017]. Tan *et al.* [2014] used feedforward neural networks to physically simulate bicycle stunts.

Recurrent Neural Networks (RNNs) are popular models for processing sequential, time-series data. The RNN takes inputs, update its internal state through recurrent connection, and generates outputs at every time-step iteratively. Therefore, the history of inputs affects the generation of outputs. The online and history-dependent nature of the RNN makes it particularly suitable for natural language understanding, speech recognition and machine translation [Graves and Jaitly 2014; Sutskever et al. 2011]. It was reported that a variant of RNNs, Long Short-Term Memory (LSTM) networks, are capable of learning simple context-free and context-sensitive languages [Gers and Schmidhuber 2001]. Recently, the RNNs have also been used for image processing [Donahue et al. 2015; Suwajanakorn et al. 2017] and character animation [Mordatch et al. 2015]. The RNNs learned from full-body human motion data demonstrated their effectiveness successfully in action recognition [Du et al. 2016] and action prediction [Fragkiadaki et al. 2015; Ghosh et al. 2017; Jain et al. 2016; Martinez et al. 2017; Zhou et al. 2018]. We are interested in motion synthesis for interactive graphics applications, which poses new challenges for RNN-based approaches.

3 MULTI-OBJECTIVE CONTROL MODEL

Consider a character that interacts with the environment over discrete time steps. At each time step t, the character receives an observation and makes a move to update its full-body pose and location for the next time step. The observation in three-dimensional

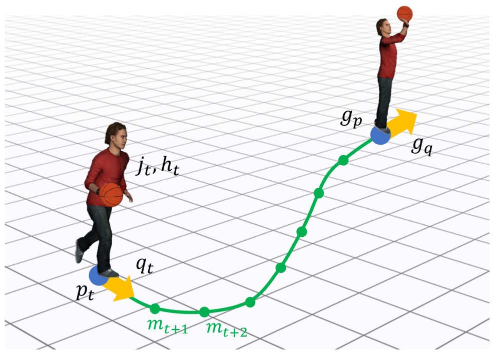

Fig. 2. Interactive character animation using user control inputs.

simulations is rich including the full-body pose of the character, the configuration of the environment, and the configuration of its collaborative/adversarial counterparts. Any desired future observation can serve as a control parameter. The user may specify where and when the character will execute a target action, and can also specify the quality and style of the generated motion.

The character may have a set of actions it can choose from. For example, the basketball player can perform actions such as *Dribble*, *Shoot*, *Pass*, and *Catch*. Each action is parameterized by a different set of control parameters $\mathbf{f}_a$, which is a subset of the full observation. For multi-objective control, we use integrated control parameters F as input of the system.

$$F = \bigcup_a f_a,$$

At each time step t, the user provides control parameters $\mathbf{f}_a$ specific to the action currently performing. We want to leave the parameters irrelevant to the current action unspecified. However, it is difficult to implement unspecified values in the network-based architecture. Instead, we predict those irrelevant parameters from the current states using the control network to feed them back into the network. The prediction is the future observation the system will likely to achieve assuming that there is no intention (or control) to regulate the future states.

The control task is to decide a next move at every time step t that best achieves the goal

$$G_t = \mathrm{diag}(S)\, F_t + \mathrm{diag}(1 - S)\, P_t,$$

where S is a binary vector with its entries either 0 or 1. The selection vector is defined for each individual action type to select entries relevant to the particular action from F_t. Our multi-objective control model takes a dynamically specified target G_t and current state $\mathbf{m}_t$ as input and generates a next state $\mathbf{m}_{t+1}$ and the prediction of control parameters P_{t+1} as output. Interactive control is a discrete-time process of generating motion $M = \{\mathbf{m}_t\}$ starting from initial configuration $\mathbf{m}_0$.

Full-body Configuration. The animated character in motion capture data is an articulate figure of rigid segments connected by ball-and-socket joints. The motion of an articulated character is represented by its time-varying position and orientation of the root body segment in the reference coordinate system together with its

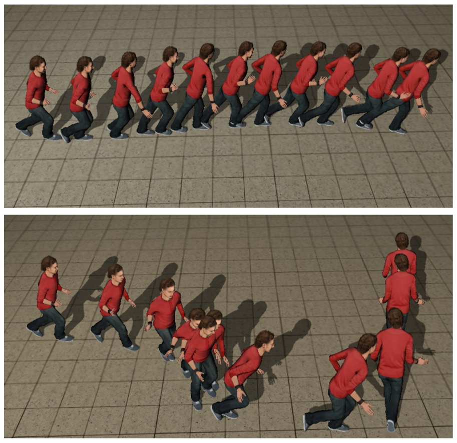
Fig. 3. Example of Quality Attributes. The control task requires the agent to move from one position to the other in time T with *Straightness* attribute S, and T is longer than it normally takes for the agent to complete the task. (Top) If S is close to one, the agent will walk slowly along the straight line between the two positions. (Bottom) Otherwise, if S is close to zero, the agent will take a curved path at normal speed.

joint angles also varying over time. It is often beneficial to maintain both joint angles and joint positions together in training networks even though the representation is redundant. Let $\mathbf{m} = (p, q, \mathbf{j}, \mathbf{h}, \mathbf{c})$ be the full-body configuration of a character, where $p \in R^2$ is the position of the character, $q \in R^2$ is its facing direction, $\mathbf{j}$ is a long vector concatenating joint angles, and $\mathbf{h}$ is another long vector concatenating joint positions. Physical contact (e.g., foot-ground and hand-ball contact) is yet another important information to understand the physical state of human poses. $\mathbf{c}$ is a binary vector that encodes whether individual body parts, such as forefeet and heels, are in contact with the ground surface or any props.

User Control. One way of controlling a character is to specify target position $g_p \in R^2$ and facing direction $g_q \in R^2$ interactively and command the character to track the target (see Figure 2). Additionally, we can specify which action g_a to perform at the target location and how long it takes to get there. Let $U = (g_p, g_q, g_a, g_t)$ be a user control input, where g_a is a one-hot vector denoting an action label and g_t is the timing of action. various control parameters, such as moving direction and velocity, can be added to the target list if necessary. The flexibility of user control is important, because our model represents various types of actions and each type may require a different form of user control. Here, all spatial coordinates are represented with respect to the body local coordinate system of the character at time t and all temporal values are relative to the current time t.

Quality Attributes. The quality attribute describes a certain aspect of movements that can be measured from motion data. For example, *Straightness* describes how straight the moving path is, and *Uprightness* describes the posture of the character in action (see Figure 3). The individual attribute at time t is a scalar value of $[0, 1]$ and evaluated from a window of neighborhood $(\mathbf{m}_{t-\tau}, \cdots, \mathbf{m}_{t+\tau})$. The quality attribute also serves as a control parameter to specify the style of the generated motion.

Bounded parameters. The control vector includes many parameters, and each parameter will be normalized to have a mean of zero and a standard deviation of one for network training. Therefore, in this network-based formulation, there is no notion of weighing the significance of one parameter with respect to the others. Instead, we use a different approach to modulate the significance of individual parameters. For example, the range of an orientation parameter is bounded within a 2π-interval (or [0,1] if unit quaternions are used), while the range of a position parameter is unbounded. Consider the normalized value of a target position g_p, which is discriminative when the target is far away (large value). As the character approaches the target, the magnitude of the parameter becomes small and not discriminative anymore. To cope with this range resolution problem, we may use an auxiliary bounded parameter

$$\bar{g}_p = \min(g_{\max}, \|g_p\|) \frac{g_p}{\|g_p\|},$$

where $g_{\max}$ is the bound of $\bar{g}_p$. $\bar{g}_p$ is discriminative only when the target is close within the bound. g_p and its bounded value $\bar{g}_p$ operate together as a coordinated pair of long-range and short-range sensors. Auxiliary bounded parameters can be exploited for any type of parameters to supplement their resolution power.

4 NETWORK TRAINING

RNN is a kind of neural networks particularly useful for modeling sequential data. Unlike many feedforward neural networks, a RNN maintains hidden internal states that is not only dependent on the current observation, but also relies on the previous hidden state and hence the previous observations. RNN Training is similar to feedforward network training in the sense that network parameters are updated incrementally via backpropagation. Since parameters are shared by all time steps in RNN, gradients at the current time step would affect gradient computation at the previous time steps. This process is called Backpropagation Through Time (BPTT). We built our RNNs using Long Short-Term Memory units (LSTMs), which preserves gradients well while backpropagating through time/layers and thus can deal with long-term dependencies.

Our network model consists of multiple LSTM layers with encoder and decoder units (see Figure 4). The input to the network is the current full-body configuration $\mathbf{m}_t$ and a goal G_t. The output is the prediction of control parameters $\hat{P}_{t+1}$ and the full-body configuration $\mathbf{m}_{t+1}$, which recursively feeds back into the network at the next time step.

4.1 Preprocessing Training Data

Our network is trained using raw motion capture data, which are labeled with minimal manual effort. The training data are often unstructured consisting of a number of short motion clips. Manual labeling is needed when there is a prop or an interaction between multiple characters. We selected and labeled key frames in which a key action occurs. Any frame without explicit action label has default action, which is Move-To in our experiments. The key action usually involves the change of contact states, such as a ball leaving

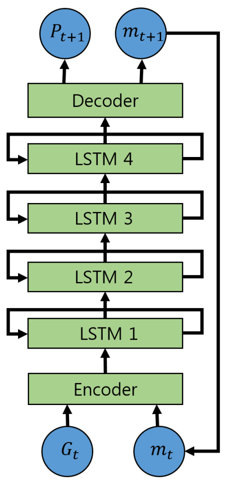

Fig. 4. Network architecture.

the hand, a racket hitting a ball, and a dancer putting his/her hand on the shoulder of the other. Foot-ground contact can be identified automatically if the foot is close to the ground surface and its velocity is below a certain threshold. Ball-Hand and Ball-Racket contacts are not easy to identify since the ball is not included in motion capture data. We labeled Ball-Hand and Ball-Racket contacts semi-automatically throughout the training data.

4.2 Control Objectives

The common practice of supervised learning requires a collection of input-output pairs and searches for network parameters that minimize the desired and simulated outputs given input data. Unlike this common practice, our training data do not have such input-output pairs, but only have a collection of motion sequences that correspond to the outputs of the neural network. Fortunately, task descriptions can be inferred from training data to build input-output correspondences. This inference step allows RNN to be learned even if we do not know how to create desired outputs given arbitrary inputs.

RNN training is a randomized process. A motion clip is selected randomly from a pool of training data. If the motion clip is very long, a random subsequence of fixed length (in our experiments, the subsequences are 6 seconds long) is selected from the clip. Let $\hat{M} = \{\hat{\mathbf{m}}_t\}$ be a sequence of motions. We use a hat above a symbol to indicate the measurements in the training data. The task $\hat{G}_t$ at every time step t should be inferred from the data. We assume that the agent has a multitude of intentions (or objectives) at every moment. Specifically, given a set of temporal offsets $\tau_1, \cdots, \tau_n$, the agent at time step t has multiple objectives of performing the action labeled at frame $\hat{\mathbf{m}}_{t+\tau_i}$ in time τ_i for any $1 < i < n$. This approach generates a dense set of control tasks to be learned by the network. The rich multi-objective learning allows precision and flexibility of control with a variety of actions in the unstructured training data.

The total amount of information RNN has to learn depends on the time horizon it wants to predict. RNN should be bigger and deeper to look ahead further, since many possibilities that can happen in the lookahead interval have to be learned and encoded. In our implementation, the time horizon $t_{\max}$ is limited to 4 seconds.

4.3 Loss Function

All consecutive pairs $(\hat{\mathbf{m}}_t, \hat{\mathbf{m}}_{t+1})$ are used to evaluate the loss function, which has three terms:

$$E = E_{\text{motion}} + E_{\text{contact}} + E_{\text{rigid}}. \tag{1}$$

E_{motion} penalizes the discrepancy between training data and network outputs.

$$E_{\text{motion}} = \sum_t \|\mathbf{m}_t - \hat{\mathbf{m}}_t\|^2, \tag{2}$$

where $\hat{G}_{i-1}$ is the input to the network, $\hat{\mathbf{m}}_t$ is the expected output observed in the training data, and $\mathbf{m}_t$ is the actual output from the network. E_{contact} penalizes foot sliding.

$$E_{\text{contact}} = \sum_t \sum_{k \in \text{Feet}} c_t^k c_{t+1}^k \|\mathbf{h}_t^k - T(\mathbf{h}_{t+1}^k)\|^2, \tag{3}$$

where k is the index of a joint position. Here, c_t^k is 1 if joint k is in contact with the ground surface at time step t. Otherwise, $c_t^k = 0$. $T(\mathbf{h}_{t+1}^k)$ transforms the coordinates of $\mathbf{h}_{t+1}^k$ to compare with $\mathbf{h}_t^k$ while taking body translation and rotation at $[t, t+1]$ into consideration. Condition $c_t^k = c_{t+1}^k = 1$ indicates that the contact remains for the time step. E_{rigid} penalizes the lengthening and shortening of body links.

$$E_{\text{rigid}} = \sum_t \sum_{(k,m) \in \text{Links}} (\|\mathbf{h}_t^k - \mathbf{h}_t^m\| - l_{km})^2, \tag{4}$$

where joint k and joint m are adjacent to each other connected by a rigid link and l_{km} is the length of the link. The distance between the adjacent pair of joints should be preserved.

5 LEARNING BALL TRAJECTORY

Interactive character animation often includes props, such as balls and rackets. Characters and props move together in a synchronized manner. Dribbling the basket ball is of particular importance, because the ball goes through interesting physical stages: ball-in-hand, ballistic-flight, and bounce-on-ground. The ball in hand moves synchronized with the body. When the ball is thrown, the ballistic flight is governed by a physics law and its moving trajectory is independent of the character's motion. Bouncing on the ground yields a rapid change of the moving direction. We consider the ball as an extra joint of the articulated figure and learn dribble actions including the ball.

Our basketball training data do not include balls, so we synthesized ball trajectories via physics simulation. Since ball-hand contacts are annotated at frames, the initial position and velocity of the ballistic flight, and its bouncing point can be estimated from motion data. We augmented training data with simulated ball trajectories. The ball position b_t is described with respect to the body local coordinates and included as part of the character's pose.

Even though RNN is adept at learning ballistic trajectories, extra loss terms are required to prevent visual artifacts. E_{ball} is added to the loss function of the full-body motion in Equation (1).

$$E_{\text{ball}} = E_{\text{hand}} + E_{\text{bounce}}, \tag{5}$$

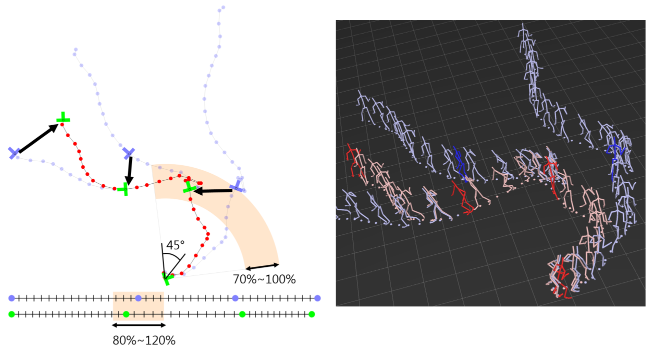

Fig. 5. Spatial and temporal perturbation of motion data. The spatial trajectory depicted in blue is perturbed to generate spatial variations. A varied spatial trajectory is depicted in pink. Temporal variations are generated by varying the timing of actions (left, bottom).

where E_{hand} keeps the ball in hand when the contact flag $c_t^k = 1$.

$$E_{\text{hand}} = \sum_t \sum_{k \in \text{Hands}} c_t^k \| |\mathbf{h}_t^k - b_t| - \text{radius}\|^2. \tag{6}$$

The distance between joint $\mathbf{h}_t^k$ and ball b_t is maintained to be equal to the radius of the ball. E_{bounce} enforces that the ball bounces at the ground height. Let c_t be the contact flag that goes on if the ball touches either hands. The contact flag indicates that the ball is pushed off when $c_t = 1$ and $c_{t+1} = 0$. The ball travels and bounces on the ground while $c_t = c_{t+1} = 0$. The ball comes back to a receiving hand when $c_t = 0$ and $c_{t+1} = 1$. Assuming that the trajectory is ballistic, the minimal height of the ball between push-off and receive-back is its height when it bounces on the ground. E_{bounce} prevents the ball from bouncing before it reaches the ground height.

$$E_{\text{bounce}} = \sum_t (1 - c_t) c_{t+1} \|y_t - \text{radius}\|^2, \tag{7}$$

where y_t is the minimal height of the ball since the last push-off. y_t is updated incrementally at every frame.

$$y_{t+1} = c_t b_{t+1}^y + (1 - c_{t+1}) \min(y_t, b_{t+1}^y), \tag{8}$$

where b_t^y is the y-coordinate (height from the ground surface) of the ball at time t. Note that bouncing happens in the middle of ballistic flight and the loss is reflected later when the ball comes back to the hand. Backpropagation through time allows such a delayed loss to be reflected in network training.

6 DATA AUGMENTATION

The performance and accuracy of our model depends mainly on the size and quality of training data. The amount of high-quality motion data available in each individual application domain is insufficient. In our examples, the size (playing time) of raw motion data for each domain ranges from small (only a few seconds) to moderate (20 minutes). We augment the raw data to increase both combinatorial and spatiotemporal variations. The whole process is fully automatic if the underlying behavior is simple in the class of *regular language*, and semi-automatic if syntactic structures in the class of *context-free language* are required.

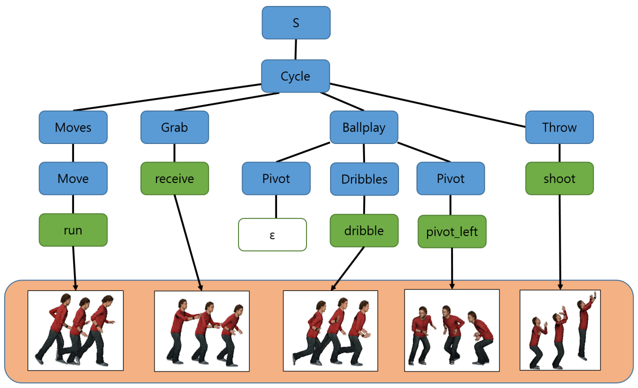

Fig. 6. Example of grammar expansion

6.1 Simple regular behavior

We begin with simple behaviors that can be captured by a finite state machine. The learning process is episodic. The goal of data augmentation is to create a large number of episodic motion sequences, exhibiting a diversity of action sequences, moving directions, and speeds. Given a collection of raw motion data, we generate episodes through two steps. The first step is to generate random combinatorial variations by resequencing motion frames. To do so, we constructed a motion graph from the input data [Lee et al. 2002]. The motion graph is a directed graph that takes motion frames as its nodes. The edge of the graph represents a transition probability from one frame to the other. The transition happens more likely if the start and end frames are similar to each other. Random walk through the graph generates a novel sequence of motions by string visited frames together. The random walk is discrete and combinatorial since only discrete choices (transitions) are provided at every node. The length of random walk depends on the complexity of behavioral structures we want to capture from the data set.

The second step is to enrich spatial and temporal variations by perturbing motion sequences from random walk. Simply adding Gaussian noise could introduce visual artifacts and corrupt the quality of the output motion. Perturbation should be conducted while preserving the integrity and coherence of human motion. To do so, we employ Laplacian motion editing [Kim et al. 2009], which can deform the spatial trajectory and timing of motion data in an as-rigid-as-possible manner subject to user-provided positional, directional, and timing constraints (see Figure 5). Given a motion sequence from random walk, we first divide it into segments of random lengths, then deform the spatial trajectory of each segment randomly in the range of its length from 70% to 100% and its angle from -45°to 45°, and finally connect those segments back together to have the spatial trajectory of the whole sequence perturbed smoothly. We also add timing constraints at random time instances to accelerate or decelerate randomly in the range of 80% to 120% of the original speed.

6.2 Context-free behavior

While motion graphs capture simple structures that can be inferred immediately from observations, more complex behavior patterns are required for various character animation applications. Ideally, we wish to learn a formal grammar from training data via grammar induction. However, motion databases currently available for individual domains are insufficient for automatic grammar induction. Instead, we provide with a manually-crafted formal grammar (see Appendix).

The use of a motion grammar requires extra efforts for motion segmentation and labeling. The basketball regulation requires detailed action classification. We defined motion grammars similarly to the work of Hyun *et al.* [2016]. The motion grammar has a finite number of terminal and non-terminal symbols, and includes a set of production rules. Each terminal symbol (a.k.a. action symbol) represents an action and may corresponds to multiple motion segments. For example, we have many Jump-And-Shoot motions in the training data. All of the motions match to the same action symbol. One of the nonterminal symbols serves as a starting symbol and fully expanding non-terminal symbols produces a string of action symbols. One non-terminal symbol may have multiple production rules. Randomly applying production rules generates a family of random parse trees representing diverse structural variabilities. Given a string of action symbols, selecting motion segments corresponding to individual action symbols and concatenating them will generate a new episodic motion sequence, which goes through the process of spatial and temporal perturbation in Section 6.1.

7 EXPERIMENTS

We implemented our algorithm in Python using TensorFlow [TensorFlow 2015] for the learning and evaluation of recurrent neural networks. The computing power of GPUs (NVIDIA GeForce GTX 1080) were utilized to accelerate neural network operations. We use multi-layer RNNs with four LSTM layers of size 512. The time step for truncated BPTT is 48. Our implementation uses tanh activation function and Adam optimization algorithm. The network architecture and the network output format are the same in all examples, but the network input depends on the application domain and the design of control parameters. The largest network model takes 32 Mbytes in main memory. Learning takes about 12 to 24 hours on a single GPU. Each dimension of input and output vectors is normalized to have zero mean and one standard deviation for network training, and denormalized to compute loss E_{contact} and E_{rigid} in Cartesian space.

We utilized a collection of motion data sampled at the rate of 30 fps. The total playtime of the basketball, tennis, and locomotion data are 10 to 12 minutes long. We built a motion graph for each data set, and data augmentation generated 4 hours of motion data for each domain. The motion grammar was constructed only for the basketball example (see Appendix for the motion grammar). The learning process is fully automatic except for the tennis and basketball examples. The tennis data require minimal manual processing to label the motion frames in which the racket hits the ball. The basketball data set require manual efforts for motion segmentation and labeling.

Locomotion. The locomotion example demonstrates an interactive character that can be controlled with only a few control parameters. The training data include unstructured motion clips exhibiting walking, turning, side stepping, and backward stepping. The user can specify either target position or target direction, or both to control the character interactively. Being able to enable/disable control options is an important aspect of multi-objective control that allows many, unorganized action repertoires to be learned in a single network model. The network model learned rich and temporally dense supervision that enables multi-objective control. We use quality attributes to select particular gaits or locomotion styles. Each attribute can be either binary or continuous scale in $[0, 1]$ depending on the composition of training data. If training data include distinctive gaits and styles without any transitioning between them, binary attributes select a particular one among them. If there are a lot of transitioning between gaits and styles in training data, continuous-scale attributes generate inbetween styles as if motion blends do.

Basketball. In the construction of the basketball model, we define six key actions (*Shoot*, *Pass*, *Receive*, *Catch*, *Dribble*, *Walk*) and four control parameters (position, direction, action type, timing). Each action is parameterized by a combination of the control parameters. For example, *Pass* takes two control parameters (position and direction), which describe the location where the action will occur and the direction it throws the ball, respectively. *Receive* takes a different combination of control parameters (position and timing), which describe the location where the player receive the ball and the timing of the ball arriving the location, respectively. The user can control the character interactively by dragging and triggering key actions. The target position, orientation, and timing are determined appropriately when an action is triggered. The ballistic trajectory of the ball after shooting is physically simulated. *Catch* requires the character to predict the ballistic trajectory and decide where and when it will catch the ball. In the multi-character setting, the user controls one character with the ball at a time while the other characters move autonomously based on stochastic play patterns.

Tactics Board. Hyun *et al.* [2016] demonstrated a tactics board interface, which allows a user to draw tactical plans on the board and generates the animation of 3D full-body basketball players acting out the plans. We reproduced their results using our network model (see Figure 7). The motion grammar and the MCMC algorithm by Hyun *et al.* took 4 and 6 minutes to generate animations, respectively. We took their motion data and motion grammar to learn our model. Single character motion generation with RNN takes only 5 millisecond per frame. The tactics board examples take 2.76 and 3.84 seconds, respectively. Our algorithm is about 45 times faster than the MCMC algorithm.

Tennis. Two actions, *Stroke* and *Return*, are defined for the tennis example. The character strokes the ball and returns to the default location to prepare for the next stroke repeatedly. Control parameters can be defined flexibly depending on the design of a scenario and a user interface for character animation. Since the ball travels across the court between two players, the ranges of the ball positions and timings (20 to 30 meters in space and 1 to 2 seconds in time) are quite broader than the expected precision of hitting the ball at the middle of the racket at an exact timing (several centimeters in space and sub-second time scale). The normalized coordinates and timing of the ball are not discriminative when the player strokes the ball.

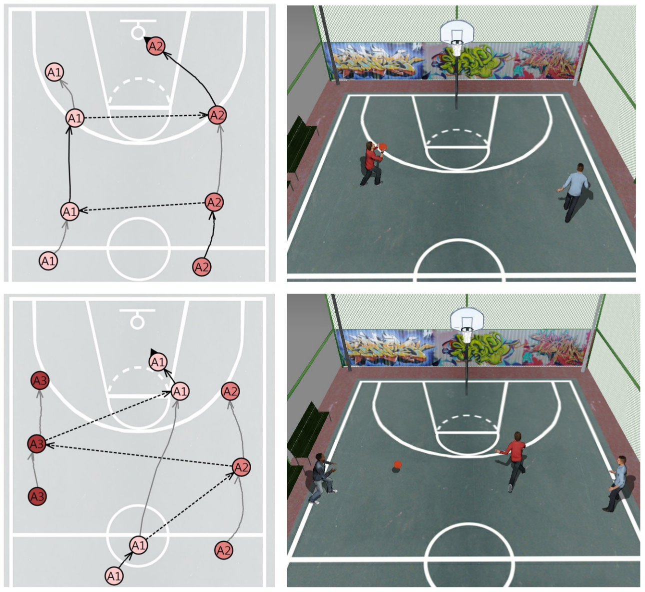

Fig. 7. Animation authoring from drawings on the tactics board.

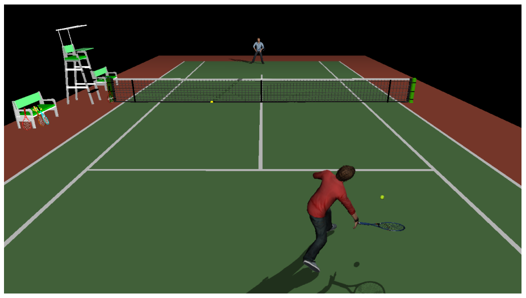

Fig. 8. A snapshot of interactive tennis play.

We found that the use of auxiliary bounded parameters significantly improves the precision of control.

Data Augmentation. The dribbling character was created using one minutes of dribbling motion data taken from the basketball data set. The character demonstrates a number of dribbling skills in response to the user's control. To evaluate its agility and responsiveness, we measured the average time to get to random targets 6 meters away from the character.

We performed a series of experiments by successively increasing the amount of data augmentation to demonstrate the effectiveness of our data augmentation method (see Figure 9). Control accuracy improves as the amount of synthetic data increases up to a certain threshold. After the threshold, additional data make no difference. The size of the required data augmentation depends on the content of the original mocap data. In our examples, we generated 4 hours of data, which is large enough, regardless of the data domain. It is just that we want to make the process automatic without the need of parameter tuning. We would also like to note that the relation between the learning speed and the size of training data is not linear. The relation is proportional when the training set is small, but the learning speed converges at a certain threshold, implying that excessive data expansion is less harmful than one might fear.

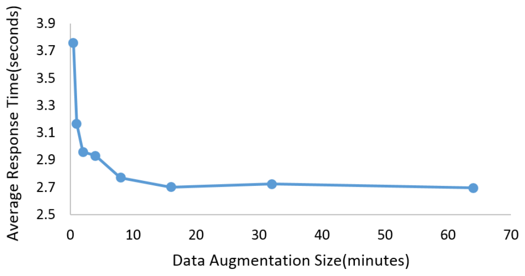

Fig. 9. Agility by Data Augmentation.

Learning Locomotion from Small Datasets. Our network model can be trained using a small amount of training data. A few short motion clips can learn a fully functional motion generator, though it is not as responsive as one may expect. The character can be more agile, responsive, and realistic with bigger and more diverse training data.

Teacher Forcing. Teacher forcing is a technique that uses the ground truth of the past frame as input rather than the output of the network [Williams and Zipser 1989]. It allows the output to stay in the ground truth, and makes the learning faster and more stable in the training phase. We tested this technique, which helps to reduce the convergence time. However, the network thus obtained often exhibited artifacts with a complex example, such as basketball. It seems there is a trade-off between learning speed and robustness. All demos in this paper were learned without teacher forcing for high quality output.

8 DISCUSSION

Recent video games provide rich details of characters' motion using large, high-quality motion databases. Many game developers have designed finite state machines and control mechanisms carefully to create interactively controllable game characters. Our model can be trained exploiting those readily-available motion databases, data structures, and control algorithms, which can generate episodic motion sequences randomly. The model learned from the synthesized training data can simulate the functionality of the existing systems while offering all the benefits of RNN-based online control, such as compact memory usage, computational efficiency at runtime, and flexibility in controller design.

The quality of the generated output mainly depends on the size and variability of the training data. Even though our data augmentation algorithm circumvents this issue to some degree, the diversity and responsiveness of character's motion are limited by the scope of the training data. Visual artifacts may be present if the animated character is controlled beyond its mobility and agility. We found that our model handles extreme control rather gracefully, avoiding undesirable visual artifacts such as jerky twitches and discontinuous jumps.

Recently, deep learning approaches have been quite successful in physics-based and data-driven animation. This trend does not imply that the conventional physics-based and data-driven methods will be obsolete. Existing animation methods will not be replaced by deep neural networks, which in fact supplement existing animation pipelines. As demonstrated in our work, many existing methods, such as motion graphs, motion grammars, and Laplacian motion editing, are essential components of learning deep neural networks. We found that deep learning can be an effective glue to put heterogeneous components together.

RNN vs Feedforward Neural Networks. Previously, feedforward neural networks (FNNs) have been successfully applied to interactive character animation. RNNs have a lot of advantages over FNNs in modeling sequential, time-series data. FNN decides its action based on immediate observation at the current state, while RNN accounts for the trace of previous actions. In the work of Holden *et al.* [2017], carefully designed user interfaces were used to provide structured control inputs, which include both past and future trajectories of biped locomotion. The structured input makes it possible to use FNN for sequential data synthesis. A latent phase variable also played an important role of aligning cyclic patterns of locomotion. As demonstrated by Zhang *et al.* [2018] recently, the phase model can generalize to deal with a range of periodic and non-periodic actions using a Mixture of Experts model. They reported that a number of modes (or experts) should be chosen carefully to warrant stable convergence and avoid expert imbalance. Our RNN-based model requires neither structured inputs nor phase variables. We can design control inputs flexibly as an arbitrary combination of target position, target orientation, moving direction, moving velocity, and the timing and type of actions. Our model can learn many structurally-different actions all together without explicit representation of phases or modes, because the underlying dynamics and the connectivity of motions are learned in the recurrent network.

Future work. There are many exciting directions to explore. We wish to explore the simulation and control of multiple characters physically interacting with each other at close distance. As the distance decreases, synchronizing and aligning inter-personal interactions become exponentially challenging. An interesting direction would be to learn collaborative and adversarial behaviors between multiple characters. It would also be interesting to learn secondary effects, such as cloth flicking, hair waving, and muscle bulging.

ACKNOWLEDGMENTS

This research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the SW STARLab support program (IITP-2017-0536-20170040) supervised by the IITP(Institute for Information & communications Technology Promotion.

REFERENCES

Myung Geol Choi, Manmyung Kim, Kyung Lyul Hyun, and Jehee Lee. 2011. Deformable Motion: Squeezing into cluttered environments. *Computer Graphics Forum* 30, 2 (2011), 445–453.

Neil Dantam and Mike Stilman. 2013. The Motion Grammar: Analysis of a linguistic method for robot control. *IEEE Transactions on Robotics* 29, 3 (2013), 704–718.

Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. 2015. Long-term recurrent convolutional networks for visual recognition and description. In *the IEEE Conference on Computer Vision and Pattern Recognition.* 2625–2634.

Yong Du, Yun Fu, and Liang Wang. 2016. Representation learning of temporal dynamics for skeleton-based action recognition. *IEEE Transactions on Image Processing* 25, 7 (2016), 3010–3022.

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

Felix A Gers and E Schmidhuber. 2001. LSTM recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks* 12, 6 (2001), 1333–1340.

Partha Ghosh, Jie Song, Emre Aksan, and Otmar Hilliges. 2017. Learning Human Motion Models for Long-term Predictions. *arXiv preprint arXiv:1704.02827* (2017).

Alex Graves and Navdeep Jaitly. 2014. Towards end-to-end speech recognition with recurrent neural networks. In *the 31st International Conference on Machine Learning.* 1764–1772.

Keith Grochow, Steven L Martin, Aaron Hertzmann, and Zoran Popović. 2004. Style-based inverse kinematics. *ACM Transactions on Graphics* 23, 3 (2004), 522–531.

Gutemberg Guerra-Filho and Yiannis Aloimonos. 2012. The syntax of human actions and interactions. *Journal of Neurolinguistics* 25, 5 (2012), 500–514.

Daniel Holden, Taku Komura, and Jun Saito. 2017. Phase-functioned neural networks for character control. *ACM Transactions on Graphics* 36, 4, Article 42 (2017).

Daniel Holden, Jun Saito, and Taku Komura. 2016. A deep learning framework for character motion synthesis and editing. *ACM Transactions on Graphics* 35, 4, Article 138 (2016).

Kyunglyul Hyun, Kyungho Lee, and Jehee Lee. 2016. Motion grammars for character animation. *Computer Graphics Forum* 35, 2 (2016), 103–113.

Ashesh Jain, Amir R Zamir, Silvio Savarese, and Ashutosh Saxena. 2016. Structural-RNN: Deep learning on spatio-temporal graphs. In *the IEEE Conference on Computer Vision and Pattern Recognition.* 5308–5317.

Manmyung Kim, Youngseok Hwang, Kyunglyul Hyun, and Jehee Lee. 2012. Tiling motion patches. In *the 11th ACM SIGGRAPH/Eurographics Conference on Computer Animation.* 117–126.

Manmyung Kim, Kyunglyul Hyun, Jongmin Kim, and Jehee Lee. 2009. Synchronized multi-character motion editing. *ACM Transactions on Graphics* 28, 3, Article 79 (2009).

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

Lucas Kovar, Michael Gleicher, and Frédéric Pighin. 2002. Motion graphs. *ACM Transactions on Graphics* 21, 3 (2002), 473–482.

Jehee Lee, Jinxiang Chai, Paul SA Reitsma, Jessica K Hodgins, and Nancy S Pollard. 2002. Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics* 21, 3 (2002), 491–500.

Jehee Lee and Kang Hoon Lee. 2004. Precomputing Avatar Behavior from Human Motion Data. In *the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* 79–87.

Kang Hoon Lee, Myung Geol Choi, and Jehee Lee. 2006. Motion Patches: Building blocks for virtual environments annotated with motion data. *ACM Transactions on Graphics* 25, 3 (2006), 898–906.

Sergey Levine, Yongjoon Lee, Vladlen Koltun, and Zoran Popović. 2011. Space-time planning with parameterized locomotion controllers. *ACM Transactions on Graphics* 30, 3, Article 23 (2011).

Sergey Levine, Jack M Wang, Alexis Haraux, Zoran Popović, and Vladlen Koltun. 2012. Continuous character control with low-dimensional embeddings. *ACM Transactions on Graphics* 31, 4, Article 28 (2012).

Libin Liu and Jessica K. Hodgins. 2017. Learning to Schedule Control Fragments for Physics-Based Characters Using Deep Q-Learning. *ACM Transactions on Graphics* 36, 3 (2017).

Julieta Martinez, Michael J Black, and Javier Romero. 2017. On human motion prediction using recurrent neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* IEEE, 4674–4683.

Jianyuan Min and Jinxiang Chai. 2012. Motion Graphs++: A compact generative model for semantic motion analysis and synthesis. *ACM Transactions on Graphics* 31, 6, Article 153 (2012).

Volodymyr Mnih and Koray et al. Kavukcuoglu. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

Igor Mordatch, Kendall Lowrey, Galen Andrew, Zoran Popovic, and Emanuel V Todorov. 2015. Interactive control of diverse complex characters with neural networks. In *Neural Information Processing Systems.* 3132–3140.

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. 2018. DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills Paper Abstract Author Preprint Paper Video. *ACM Transactions on Graphics* 37, 4 (2018).

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* 36, 4, Article 41 (2017).

Alla Safonova and Jessica K Hodgins. 2007. Construction and optimal search of interpolated motion graphs. *ACM Transactions on Graphics* 26, 3, Article 106 (2007).

Hyun Joon Shin and Hyun Seok Oh. 2006. Fat Graphs: Constructing an interactive character with continuous controls. In *the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 291–298.

Hubert PH Shum, Taku Komura, Masashi Shiraishi, and Shuntaro Yamazaki. 2008. Interaction patches for multi-character animation. *ACM Transactions on Graphics* 27, 5, Article 114 (2008).

Ilya Sutskever, James Martens, and Geoffrey E Hinton. 2011. Generating text with recurrent neural networks. In *the 28th International Conference on Machine Learning*. 1017–1024.

Supasorn Suwajanakorn, Steven M Seitz, and Ira Kemelmacher-Shlizerman. 2017. Synthesizing Obama: Learning lip sync from audio. *ACM Transactions on Graphics* 36, 4, Article 95 (2017).

Jie Tan, Yuting Gu, C Karen Liu, and Greg Turk. 2014. Learning bicycle stunts. *ACM Transactions on Graphics* 33, 4, Article 50 (2014).

Graham W Taylor and Geoffrey E Hinton. 2009. Factored conditional restricted Boltzmann machines for modeling motion style. In *the 26th Annual International Conference on Machine Learning*. 1025–1032.

TensorFlow. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. (2015). Software available from tensorflow.org.

Adrien Treuille, Yongjoon Lee, and Zoran Popović. 2007. Near-optimal character animation with continuous control. *ACM Transactions on Graphics* 26, 3, Article 7 (2007).

Ronald J Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural computation* 1, 2 (1989), 270–280.

Jungdam Won, Kyungho Lee, Carol O'Sullivan, Jessica K Hodgins, and Jehee Lee. 2014. Generating and ranking diverse multi-character interactions. *ACM Transactions on Graphics* 33, 6, Article 219 (2014).

Jungdam Won, Jongho Park, Kwanyu Kim, and Jehee Lee. 2017. How to Train Your Dragon: Example-guided control of flapping flight. *ACM Transactions on Graphics* 36, 6, Article 198 (2017).

Wenhao Yu, Greg Turk, and C. Karen Liu. 2018. Learning Symmetry and Low-energy Locomotion Paper Abstract Author Preprint Paper Video. *ACM Transactions on Graphics* 37, 4 (2018).

He Zhang, Sebastian Starke, Taku Komura, and Jun Saito. 2018. Mode-Adaptive Neural Networks for Quadruped Motion Control. *ACM Transactions on Graphics* 37, 4 (2018).

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*. https://openreview.net/forum?id=r11Q2SlRW

APPENDIX

Motion grammar for basketball players

Starting symbols : $\{S\}$

Non-terminals : $\{S, Cycle, Moves, Move, Grab, Ballplay, Pivot, PivotL, PivotR, Dribbles, Throw\}$

Terminals : $\{run, walk, idle, receive, catch, pivot\_left, pivot\_right, dribble\ pass, shoot, layup\}$

Production Rules :

S	$\rightarrow$	$Cycle \mid Cycle\ S$
$Cycle$	$\rightarrow$	$Moves\ Grab\ Ballplay\ Throw$
$Moves$	$\rightarrow$	$\epsilon \mid Move \mid Move\ Moves$
$Move$	$\rightarrow$	$run \mid walk \mid idle$
$Grab$	$\rightarrow$	$receive \mid catch$
$Ballplay$	$\rightarrow$	$Pivot\ Dribbles\ Pivot$
$Pivot$	$\rightarrow$	$\epsilon \mid PivotL \mid PivotR$
$PivotL$	$\rightarrow$	$pivot\_left \mid pivot\_left\ PivotL$
$PivotR$	$\rightarrow$	$pivot\_right \mid pivot\_right\ PivotR$
$Dribbles$	$\rightarrow$	$\epsilon \mid dribble \mid dribble\ Dribbles$
$Throw$	$\rightarrow$	$pass \mid shoot \mid layup$