# Learning Virtual Chimeras by Dynamic Motion Reassembly

SEYOUNG LEE, Seoul National University, South Korea
JIYE LEE, Seoul National University, South Korea
JEHEE LEE, NCsoft, South Korea and Seoul National University, South Korea

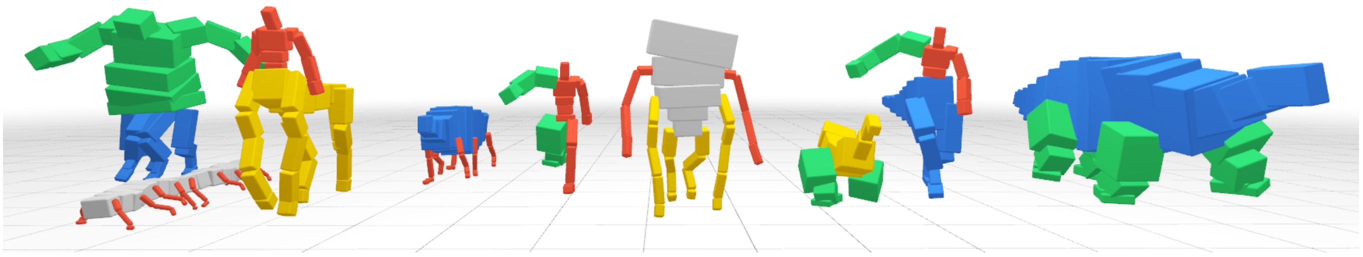
Fig. 1. Physically based movement of various chimeras.

The *Chimera* is a mythological hybrid creature composed of different animal parts. The chimera's movements are highly dependent on the spatial and temporal alignments of its composing parts. In this paper, we present a novel algorithm that creates and animates chimeras by dynamically reassembling source characters and their movements. Our algorithm exploits a two-network architecture: part assembler and dynamic controller. The part assembler is a supervised learning layer that searches for the spatial alignment among body parts, assuming that the temporal alignment is provided. The dynamic controller is a reinforcement learning layer that learns robust control policy for a wide variety of potential temporal alignments. These two layers are tightly intertwined and learned simultaneously. The chimera animation generated by our algorithm is energy efficient and expressive in terms of describing weight shifting, balancing, and full-body coordination. We demonstrate the versatility of our algorithm by generating the motor skills of a large variety of chimeras from limited source characters.

CCS Concepts: • **Computing methodologies → Physical simulation**; **Motion processing**.

Additional Key Words and Phrases: Physics-Based Simulation, Data-Driven Animation, Deep Reinforcement Learning

**ACM Reference Format:**
Seyoung Lee, Jiye Lee, and Jehee Lee. 2022. Learning Virtual Chimeras by Dynamic Motion Reassembly. *ACM Trans. Graph.* 41, 6, Article 182 (December 2022), 13 pages. https://doi.org/10.1145/3550454.3555489

Authors' addresses: Seyoung Lee, Department of Computer Science and Engineering, Seoul National University, South Korea, seyounglee@mrl.snu.ac.kr; Jiye Lee, Department of Computer Science and Engineering, Seoul National University, South Korea, jiye.lee@mrl.snu.ac.kr; Jehee Lee, NCsoft, Kyungkido, South Korea and Seoul National University, Department of Computer Science and Engineering, Seoul, South Korea, jeheel@gmail.com.

## 1 INTRODUCTION

The *Chimera* is a mythological hybrid creature composed of different animal parts. Such imaginary creatures frequently appear in many video games and movies. Although motion capture is not eligible for imaginary creatures, we have good knowledge of how each animal part moves in animal motion capture. We are interested in learning the motions of chimeras, given that the motions of individual parts are provided. The body and motion of a creature are closely related to each other. Even a small change in body proportion can have a substantial impact on the way it moves. Composing different animal parts can have a greater impact on the dynamics of individual part motion. We found that the dynamics of a chimera's movement are highly dependent on the spatial and temporal alignments of its composing parts.

In this paper, we present a novel algorithm that creates and animates highly-varied characters, which we call *chimeras*, from a collection of source animal/human characters and their motion data. Our algorithm based on deep reinforcement learning (DRL) learns the best spatial and temporal alignments of body parts while preserving the style and/or semantics of the source motions as much as possible. The ANN (Artificial Neural Network) policy thus learns to control the chimera in physics-based simulation. The source data can be either motion captured or keyframed by artists. We will demonstrate the efficacy of our algorithm with a broad range of dynamic movements, including multi-legged locomotion, jump, kick, and punch in a variety of hybrid forms.

Our algorithm exploits a two-level network architecture: part assembler and dynamic controller. The part assembler is a supervised learning layer that searches for the spatial alignment among body parts on a frame-by-frame basis, assuming that the temporal alignment is provided. The dynamic controller is a reinforcement learning layer that learns robust control policy for a wide variety of potential temporal alignments. These two layers are tightly intertwined and learned simultaneously in the learning phase. The key technical component is multi-dimensional timewarp functions that allow the timeline of individual parts to be scaled, shifted, and warped, separately. Our algorithm is more flexible and stable than alternative methods. The chimera animation generated by our algorithm is energy efficient and expressive in terms of describing weight shifting, balancing, and full-body coordination. We will also demonstrate that our algorithm can discover natural gait in biped, quadruped, and insect forms.

## 2 RELATED WORK

Generating the motion of an imaginary creature has been a challenging problem in computer graphics. Many approaches have focused on creating life-like motions based on optimality principles. Graphics researchers have shown great interest in the legged locomotion of articulated characters with human-like morphologies. Geijtenbeek et al. [2013] generated the muscle-actuated locomotion of bipedal characters. Yu et al. [2018] and Kry et al. [2009] simulated energy-efficient locomotion for various morphologies. The locomotion of soft bodies [Bern et al. 2017; Min et al. 2019; Tan et al. 2012] and elastic models [Coros et al. 2012; Liu et al. 2013; Schulz et al. 2014] have also been studied. There have been studies that focused on finding optimal body structure or proportion for locomotion tasks using genetic algorithms or sampling based methods [Sims 1994; Wampler and Popović 2009; Zhao et al. 2020]. Synthesizing the physics-based movement of a creature is computationally expensive. Graph neural networks have been exploited to control agents of various morphologies with a single policy [Huang et al. 2020; Pathak et al. 2019; Wang et al. 2018].

Data-driven approaches exploiting motion capture data have also been explored extensively in computer graphics. Motion retargeting methods transfer the motion of one character to another while maintaining the semantics of the original motion as much as possible [Gleicher 1998; Ho et al. 2010; Hodgins and Pollard 1997; Jin et al. 2018; Lee and Shin 1999; Shin et al. 2001]. Contact and/or interaction events between characters are commonly selected as key aspects to be preserved for motion retargeting [Gleicher 1998; Ho et al. 2010; Jin et al. 2018; Kim et al. 2009; Shin et al. 2001]. Most retargeting algorithms assumed that the structures of the source and target characters are similar. On the other hand, Hecker et al. [2008] encoded motion data in a morphology-independent form and had it reproduced in various characters with different skeleton structures. The motion puppetry system by Seol et al. [2013] retargets the motion of human actors to imaginary creatures by learning the direct mapping between them. Wampler et al. [2014] synthesized the gaits of a wide range of new creatures by learning coherent patterns from motion databases. Aberman et al. [2019] generated character-agnostic motions by decomposing the motions into latent components.

Motion splicing techniques synthesize the motion of articulated characters by implanting different source motions for each body part. The key challenge of kinematic splicing/transplanting approaches is the decision mechanism that tells us if the new combination of body parts is visually acceptable or not. Rule-based [Heck et al. 2006; Ikemoto and Forsyth 2004], analogy-based [Jang et al. 2008], and neural network-based [Starke et al. 2021] methods have been studied. The motion puzzle system recently developed by Jang et al. [2022] locally transfers different styles for each body part through an attention-based neural network model. Our system decides the plausibility of the new motion by learning a physics-based controller and evaluating the motion in forward dynamics simulation environments. The use of physics simulation makes big improvements in motion quality and the scope of motions the system can generate.

Physics-based character simulation and control have many advantages over kinematic/data-driven approaches [Lee et al. 2010; Liu et al. 2016; Muico et al. 2011; Sok et al. 2007; Yin et al. 2007]. The advent of deep reinforcement learning (DRL) have had a big impact on physics-based character simulation and made it possible to reproduce high-quality, highly dynamics motor skills by learning their control policies [Peng et al. 2018]. The DRL-based methods have further been explored to deal with diverse motor skills [Bergamin et al. 2019; Park et al. 2019; Peng et al. 2021; Won et al. 2020], diverse body shapes/proportions [Mordatch et al. 2015; Won and Lee 2019], anatomical body modeling [Lee et al. 2019; Park et al. 2022], visiomotor control [Merel et al. 2020], improving controllability [Lee et al. 2021b; Ling et al. 2020] and animal locomotion [Luo et al. 2020; Min et al. 2019; Reda et al. 2022; Won et al. 2017; Zhang et al. 2018]. DRL has also shown its potential in learning parameterized motor skills [Lee et al. 2021a] and discovering new motor skills from scratch without reference data [Tao et al. 2022; Yin et al. 2021].

Understanding the timing and phase of motion has always been an important aspect of learning human and animal motion [Holden et al. 2017; Starke et al. 2022]. Lee et al. [2021a] demonstrated that the flexibility of a single motion clip can be improved substantially by learning timewarp functions and employing variable time-step reinforcement learning. Park et al. [2022] reported that timewarp functions are an integral part of large-scale, massively-conditioned network models for learning the relation between human anatomy and gait. Starke et al. [2022] demonstrated that the multi-dimensional phase manifold learned by a periodic autoencoder could effectively represent spatial and temporal alignments. In this work, major improvements over the previous studies include its ability to modulate the timing of individual part animations using part-wise timewarp functions. The temporal alignments learned by our system vary depending on the physical properties of the body parts and part animations.

## 3 BUILDING CHIMERAS

The body of the animated character is represented by a tree-type articulation of rigid links connected by either revolute or ball-and-socket joints. A segment of character animation is given as a tuple $A = \{B, T, M\}$, where $B$ is a list of body links, $T$ is a list of rigid transformations between parent-child links, and $M$ is a sequence of joint angles and root translations/rotations varying over time. Without loss of generality, we assume that $M(t)$ is a continuous function of time. The motion data including samples at discrete times are considered a continuous, piecewise-linear function.

The part animation $\hat{A} = \{\hat{B}, \hat{T}, \hat{M}\}$ is a subset of $A$, where $\hat{B} \subset B$ is a connected sub-tree of the body links, $\hat{T} \subset T$ is a collection of transformations between body links in $\hat{B}$, and $\hat{M}$ is a time series of joint angles of $\hat{B}$. The chimera is constructed by taking part animations from a collection of source animations and composing these

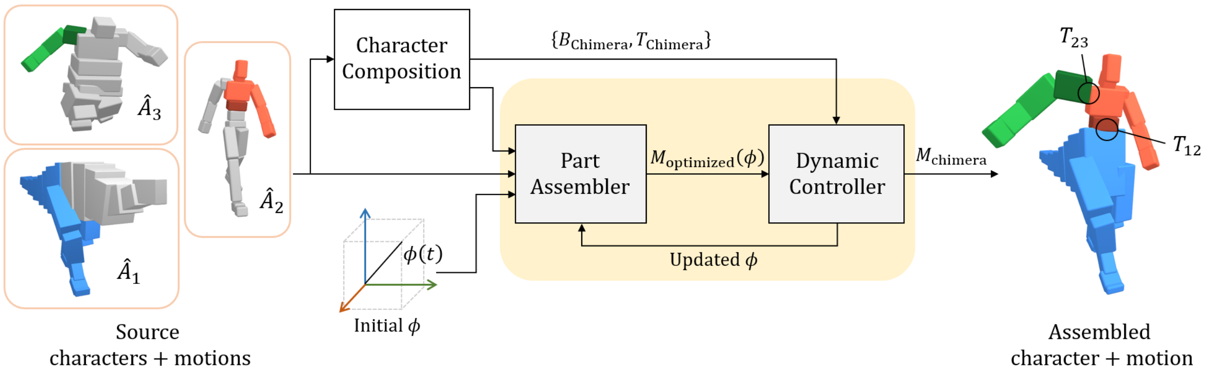

Fig. 2. System overview

part animations. In Figure 2, we took the two legs and the tail from the T-rex ($\hat{A}_1$), the trunk, the head, and the left arm from the human ($\hat{A}_2$), the right arm from the bear ($\hat{A}_3$). Simply specifying the transformations at attachment points generates an initial composition of the part animations.

$$A_{\text{initial}} = (\hat{B}_1 \cup \hat{B}_2 \cup \hat{B}_3, \hat{T}_1 \cup \hat{T}_2 \cup \hat{T}_3 \cup \{T_{12}, T_{23}\}, \hat{M}_1 \cup \hat{M}_2 \cup \hat{M}_3),$$

where $T_{12}$ and $T_{23}$ are transformations at attachment points. Any structural variation is feasible as long as the chimera's skeleton maintains a tree-structure with a unique root node and no cyclic paths. This brute-force composition is hardly ideal because of spatial and temporal mismatches. The body-body and body-environment contact in the source animations may not be preserved, unexpected interpenetration may happen, and the source animations may have different timings and cyclic patterns. The initial composition should be further refined to achieve better alignments in space and time among part animations. Through physics-based processing and learning, chimera animations are expected to satisfy the following requirements.

- Chimera animations should exhibit appropriate physical effects, such as shifting weight, maintaining balance, and conserving momentum.
- Part animations should be coordinated and synchronized in time for the chimera to be energy efficient and effective in terms of accomplishing tasks.
- Chimera animations should preserve the spatial/geometric context of the source animations, which include contact timing and interpenetration prevention.

We present a new DRL algorithm that learns a control policy mimicking the composition of part animations. The policy controls the chimera in physics-based simulation to generate physically valid motions. Specifically, our algorithm is equipped with two technical components, part-wise timewarping and optimization-based part assembly, for achieving better spatial and temporal alignments. The timewarp $\phi_i(t)$ is a monotonically increasing function that maps the timeline $t \in R$ of the chimera animation to the timeline $\phi_i \in R$ of its $i$-th part animation (see Figure 3). Our algorithm learns the policy conditioned by multi-dimensional phase alignments $\Phi = (\phi_i) \in R^N$,

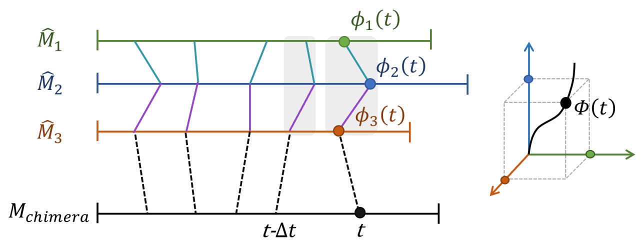

Fig. 3. Temporal alignments of the motions. In the timeline on the left, the sky blue and purple lines temporally map the frames in each timeline of the source motion. The connected colored lines are mapped to the time of $M_{\text{chimera}}$ by the black dashed lines. When the interval between $t - \Delta t$ and $t$ is widened, the transition from the first gray area to the second gray area proceeds slowly. Conversely, when the interval is narrowed, the transition proceeds quickly.

where N is the number of part animations and eventually determines the best time alignments among part animations.

Given multi-dimensional phase $\Phi(t)$ at any time frame $t$, enhancing spatial integrity between body parts can be formulated as a non-linear optimization problem similar to motion retargeting. The joint angles in individual parts should be retargeted to the chimera's body while maintaining the key aspects of the source animations. Although the computational cost of this per-frame optimization is moderate, the optimization is invoked iteratively to generate assembled pose for varying phase alignments in policy learning. To reduce the computational burden of policy learning, we learn a regression network that mimics this optimization procedure in a supervised manner. The use of the regression network achieves a ten-thousandfold increase in computational efficiency.

## 4 PART ASSEMBLY

The part assembler takes phase assignment $\Phi = (\phi_i)$ at a frame as input and generates a pose of the chimera satisfying spatial constraints. Let $M_{\text{initial}}(\Phi)$ be an initial full-body pose constructed by composing parts $\hat{M}_i(\phi_i)$ for $i$ in a brute-force manner. Let $M_{\text{optimized}}(\Phi)$ be the optimized pose, which will be feedback to the policy learner as a reference pose to track. The optimization problem has three objectives.

$$\mathbf{M}_{\text{optimized}}(\Phi) = \text{argmin}\, E_{\text{contact}} + E_{\text{global}} + E_{\text{reg}}. \tag{1}$$

We use a derivative-free optimization method, CMA-ES, to solve this optimization problem [Hansen and Ostermeier 1996].

Contact is an important visual cue that also has a significant impact on the dynamics of articulated systems. Any type of contact events in part animations should be reproduced in the chimera animation as well. The contact objective includes two terms.

$$E_{\text{contact}} = \sum_{p \in \text{points}} \min(h(p), 0)^2 + \sum_{c \in \text{contacts}} (\text{dist}(p_c) - k_c)^2. \tag{2}$$

The first term prevents any point on the body from penetrating the ground, where $h(p)$ is the height from the ground. The second term continuously modulates the distance between the body and the contact point around a contact event using an importance-based approach [Shin et al. 2001]. Here, $p_c$ is the point on the body in contact with the ground surface or any object when the contact occurs at time $\phi_c$. The target distance $k_c$ is modulated based on the importance $w_c$ at the current time $\phi$ such that

$$\begin{aligned} k_c &= (1 - w_c)\text{dist}(p_c) + w_c \text{dist}(\hat{p}_c), \\ w_c &= \max(1 - \frac{|\phi - \phi_c|}{\sigma}, 0), \end{aligned} \tag{3}$$

where $\text{dist}(p_j)$ is the distance to the estimated contact point in the optimized pose, $\text{dist}(\hat{p}_j)$ is the actual distance to the contact point in the source animation. $\sigma$ is a constant that controls the distribution of importance. With high importance $w_c = 1$, the contact constraint is strictly enforced in the optimized pose. With lower values, the contact constraints are gently guided. This importance-based approach allows the frame-by-frame optimization framework to continuously deal with the discrete nature of contact constraints.

In the initial composition $M_{\text{initial}}(\Phi)$, the root of one body part is attached to the other through a parent-child relationship. Therefore, the global trajectory of the child body part in the composition can be quite different from its global trajectory in the source animation. We sometimes want to preserve the global trajectories of the part as well as its internal joint angles (see Figure 4). $E_{\text{global}}$ serves this purpose.

$$E_{\text{global}} = \sum_{i \in N} \| \log(\bar{q}_i^{-1} q_i) \|^2, \tag{4}$$

where $q_i \in S^3$ is the root orientation of the body part in the initial composition, $\hat{q}_i \in S^3$ is the root orientation in the source animation, $\bar{q}_i = \text{slerp}(R_i \hat{q}_i, q_i, a_i)$ is the calibrated root orientation. $R_i \in S^3$ is a calibration transformation that matches the facing direction of the source animation and the composite animation. $a_i$ is a user-specified parameter. slerp stands for spherical linear interpolation [Lee 2008]. If $a_i = 1$, the part animation is defined with respect to the local, body-attached coordinate system of its parent. If $a_i = 0$, the part animation is represented and transferred in the global, reference coordinate system.

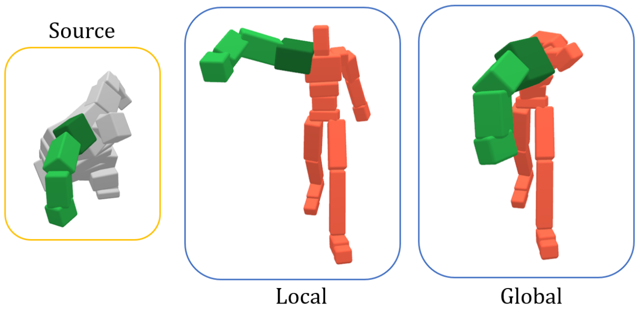

Fig. 4. Arm transplanting with respect to the local coordinate system vs the global coordinate system. The dark green colored body is the root of the transplanted arm.

We use two types of regularization terms to avoid excessive deviation of the source animations and maintain temporal coherence.

$$E_{\text{reg}} = \| \mathbf{p} - \hat{\mathbf{p}} \|^2 + w_{\text{reg}} \sum_{\Phi_k \in \text{neighbor}(\Phi)} \| \mathbf{p} - \mathbf{p}_{\Phi_k} \|^2, \tag{5}$$

where $\mathbf{p}$ is the generalized coordinate of the optimized pose, $\hat{\mathbf{p}}$ is the generalized coordinate of the initial composition $M_{\text{initial}}(\Phi)$. Our system maintains the results of part assembly during policy learning. The second term exploits the previous record to maintain frame-by-frame coherence for the per-frame optimization. The second term is evaluated for the neighborhood of $\Phi$ within a user-specified threshold $\Delta\Phi$.

## 4.1 Training

Policy learning we will discuss in the next section generates numerous phase $\Phi$ tuples that need to be evaluated through part assembly. Repeated computation can be circumvented by storing the computation results in the training datasets and incrementally learning a regression network that imitates the function of the optimization-based part assembler. We use kernel density estimation to collect phase samples uniformly over the phase space. Given a phase alignment $\Phi$, the optimization is performed and added to the training dataset only when the density around $\Phi$ is lower than a threshold. The density is estimated by

$$\text{density}(\Phi) \propto \sum_{\Phi_k \in \text{neighbor}(\Phi)} \exp\left(-\frac{1}{N}(\Phi_k - \Phi)^T(\Phi_k - \Phi)\right). \tag{6}$$

The training dataset is stored as a $k$-d tree such that it allows efficient neighborhood search.

The regression network $P_\theta$ is a feedforward network with fully-connected layers.

$$P_\theta(\Phi) = \mathbf{d}_\Phi, \tag{7}$$

The regression network learns a mapping between phase alignment $\Phi$ and optimized pose $M_{\text{optimized}}(\Phi)$ such that $M_{\text{optimized}}(\Phi) = M_{\text{initial}}(\Phi) + \mathbf{d}_\Phi$. Note that learning a pose displacement is easier and more accurate than learning an optimized pose directly. The network is trained to minimize the loss

$$\theta = \underset{\theta}{\text{argmin}} \sum_{\Phi} \|\mathbf{d}_\Phi - \mathbf{d}^*_\Phi\|^2, \tag{8}$$

where $\mathbf{d}_\Phi$ and $\mathbf{d}^*_\Phi$, respectively, are pose displacements generated by the regression network and by part assembly optimization.

## 5 POLICY LEARNING

For simplicity of explanation, we assume that individual part animations are either periodic (e.g., a cycle of locomotion) or a single execution of aperiodic motion (e.g., jump, punch, and kick). Both can be denoted by a periodic function

$$\hat{M}_i(\phi + L_i) = \hat{M}_i(\phi), \tag{9}$$

$L_i$ is either the period for periodic motions or the time duration for aperiodic motions.

The multi-dimensional timewarp functions $\Phi = (\phi_1, \cdots, \phi_N)$ align part animations in the composition. Each timewarp function maps the timeline of the chimera animation to the timeline of a part animation. In other words, the timeline of the part animation is scaled, shifted, and timewarped by $\phi_i^{-1}$ and thus aligned with the other part animations. The timewarp functions have requirements.

- The timewarp functions increase monotonically.
- All timewarped animations have the same period.

The first requirement prevents time from flowing backward. The second requirement guarantees that all part animations align precisely to form a periodic animation for the chimera. Note that we do not explicitly construct timewarp functions $\phi_i(t)$ or their inverses. We instead learn a control policy $\pi$ conditioned on phases. The control policy performs two roles. First, it informs the proper combinations of joint actuation to track the target pose, which is obtained through part assembly. Second, it generates phase increments for the next time instance. Repeating this until the end of the episode completes the timewarp of part animations. The key technical challenge is the design of states, actions, and rewards for reinforcement learning that satisfy the timewarp requirements.

Our learning algorithm is episodic. Each simulation episode includes about ten cycles of physically-simulated movements driven by the policy. Our learning algorithm collects experience tuples from the episodes to update the value and policy networks. Algorithm 1 shows the whole process of learning.

### 5.1 State and Action

As reported by Park et al. [2019], DRL with imitation rewards requires rich, descriptive state representations to uniquely identify individual states. In our system, the state consists of the body configuration (including generalized coordinates, generalized velocities, joint positions, the body up-vector, and the root height) at the current time $t$, the current phase $\Phi = (\phi_1, \cdots, \phi_N)$, and joint positions at the predicted next pose $M(\Phi + \Delta\Phi)$ computed in the local, body-attached coordinate frame, where $\Delta\Phi$ is a user-provided constant.

The action specifies the spatial displacement $D$ and phase increments $\Psi = (\psi_1, \cdots, \psi_N)$.

$$(D_t, \psi_t) = \pi(s_t). \tag{10}$$

The action updates the phases at the next time step by $\Phi(t + \Delta t) = \Phi(t) + (\exp(\psi_1), \cdots, \exp(\psi_N))$. This update rule ensures that the timewarp functions increase monotonically because $\exp(\psi_i)$ is always positive. The reference pose is $M_{\text{ref}}(\Phi(t)) = M_{\text{optimized}}(\Phi(t)) + D_t$, which is fed into PD controllers to actuate joints.

**Algorithm 1** Learning chimera animation

```
    $P(\Phi)$ : regression network
    $\pi(s|\Phi)$ : policy function
    $V(s|\Phi)$ : value function
    $B_P$ : Training data for regression network
    $B_\pi$ : Memory for update of policy and value network
    $Q_A, Q_B$ : Memory for visit-based state initialization
 1: repeat
 2:     /* Generate training tuples */
 3:     repeat
 4:         Sample initial state $s$ and $\Phi$ from $Q_B$
 5:         while not terminated do
 6:             Get target pose from $P(\Phi)$
 7:             Get action from $\pi(s|\Phi)$
 8:             Step and update $s$ and $\Phi$
 9:             Add tuples for PPO update to $B_\pi$
10:             if $Density(\Phi, Q_A) > \alpha$ then
11:                 Add $\Phi$ to $Q_A$ and pop the oldest if $Q_A$ is full
12:             if $Density(\Phi, B_P) <$ threshold then
13:                 Do part assembly optimization for $\Phi$
14:                 Add $\Phi$ and displacements to $B_P$
15:     until $N$ tuples are generated
16:     /* Update network */
17:     Update $\pi$, $V$ with $B_\pi$
18:     Update $P$ with $B_P$
19:     Clear $B_\pi$
20:     /* Update state Initialization */
21:     Clear $Q_B$
22:     for $a \in Q_A$ do
23:         if $Density(a, Q_B) < \beta$ then
24:             Add $a$ to $Q_B$
25: until no improvement
```

### 5.2 Reward

The objective of reinforcement learning is to learn the optimal policy that maximizes the discounted cumulative reward. We use two types of rewards: Continuous and spike. Continuous rewards are given over a period of integrable time duration, while spike rewards are given at discrete time instances. This classification is necessary to handle the variable time-step RL formulation [Lee et al. 2021a]. Our reward function consists of four reward terms.

$$r = w_{\text{tracking}} r_{\text{tracking}} + w_{\text{energy}} r_{\text{energy}} + w_{\text{align}} r_{\text{align}} + w_{\text{task}} r_{\text{task}}. \tag{11}$$

The tracking reward is for imitating a sequence of optimized poses.

$$\begin{aligned} r_{\text{tracking}} &= r_{\text{q}} \cdot r_{\text{ee}}, \quad \text{where} \\ r_{\text{q}} &= \exp\Big(-\frac{1}{\sigma_{\text{q}}^2} \sum_{j \in \text{joints}} \|\log(q_j^{-1}\hat{q}_j)\|^2\Big), \\ r_{\text{ee}} &= \exp\Big(-\frac{1}{\sigma_{\text{ee}}^2} \sum_{j \in \text{ee}} \|\hat{p}_j - p_j\|^2\Big). \end{aligned} \tag{12}$$

$r_{\text{q}}$ and $r_{\text{ee}}$ penalize the differences in joint angles and end-effector positions, respectively. The hat symbol stands for measures in the optimized poses.

The energy reward is for favoring energy-efficient movements.

$$\begin{aligned} r_{\text{energy}} &= -r_{\text{effort}} - r_{\text{CoT}}, \quad \text{where} \\ r_{\text{effort}} &= \sum_{j \in \text{dof}} |\ddot{q}_j|^2, \\ r_{\text{CoT}} &= \frac{\sum_{j \in \text{dof}} |\tau_j \cdot \dot{q}_j|}{m\|v\|}. \end{aligned} \tag{13}$$

$\dot{q}_j$ and $\ddot{q}_j$, respectively, are the angular velocity and acceleration of each joint. $\|v\|$ is the distance the chimera travels in $\Delta t$. $m$ is the body mass and $\tau_i$ is joint torque. The cost of transport (CoT) is a dimensionless measure that evaluates how efficiently a dynamical system is moving from one place to another. Biological studies have shown that animals tend to minimize CoT when walking and running [Taylor et al. 1970; Tucker 1970]. Both $r_{\text{tracking}}$ and $r_{\text{energy}}$ are continuous rewards.

Let $\omega$ be the estimated period of the episode.

$$\omega = \frac{1}{N} \sum_i \frac{\phi_i}{L_i}. \tag{14}$$

The align rewards are received instantaneously whenever the simulation episode completes a cyclic period (e.i., $\omega$ reaches an integer number).

$$r_{\text{align}} = \exp\Big(-\frac{1}{\sigma^2_{\text{align}}} \sum_i (\cos(\phi^*_i) - \cos(\phi_i))^2 + (\sin(\phi^*_i) - \sin(\phi_i))^2\Big), \tag{15}$$

where $\phi^*_i$ are values from the previous period. The align rewards are maximized when the timewarp functions are precisely synchronized with each other.

The user can specify any motion features to control, such as moving direction, velocity, and punch/kick impact, and design reward functions accordingly. In our experiments, two types of task rewards are used.

$$r_{\text{task}} = \exp\Big(-\frac{\|v - (v \cdot \hat{d})\hat{d}\|^2}{\sigma^2_{\text{dir}}}\Big) + \exp\Big(-\frac{\|max(\hat{f} - f, 0)\|^2}{\sigma^2_{\text{force}}}\Big) \tag{16}$$

where $\hat{d}$ is the target moving direction and $\hat{f}$ is the target contact force at punch impact. The task rewards can be either continuous or spike, depending on how we design reward functions.

## 5.3 Policy Update

We use the variable time-step PPO (Proximal Policy Optimization) algorithm to update the value and policy networks [Lee et al. 2021a]. This algorithm is stable and invariant under the choice of time steps. Let $R_c$ and $R_s$, respectively, be the sum of continuous and spike rewards. Generalized Advantage Estimation (GAE) with variable time step $\Delta\omega$ is

$$\delta(\Phi) = \int_0^{\Delta\omega(\Phi)} \gamma^v R_c(\Phi) dv + R_s(\Phi) + \gamma^{\Delta\omega(\Phi)} V(\Phi + \Delta\Phi) - V(\Phi),$$

where $\gamma$ is the discount factor and $\Delta\Phi = (\exp(\psi_1), \cdots, \exp(\psi_N))$.

## 5.4 Visit-based State Initialization

Where to begin each episode on the phase space has a big impact on the performance and convergence of learning. Peng et al. [2018] suggested choosing initial states randomly along the reference trajectory. Won et al. [2019] proposed an MCMC (Markov Chain Monte Carlo) method that samples initial states based on value functions. The curriculum learning method by Lee et al. [2021a] suggested alternating between exploration and refinement phases based on values and sample density. We present an alternative state initialization method that is particularly suitable for learning multi-dimensional phases. The key insight is that recently visited tuples would probably have better phase alignments because the control policy improves gradually throughout the learning process. So, we sample initial states unbiasedly from recently visited tuples.

To implement our visit-based state initialization, we use two priority queues $A$ and $B$. Queue $A$ stores phase tuples recently visited by RL. The density of samples in $A$ is continuously monitored. The density is estimated by using Equation (6). If the density around a tuple in $A$ is beyond a certain threshold $\alpha$, this tuple is considered a candidate for the initial states of subsequent episodes (see Figure 5). This tuple is moved to queue $B$ if the density around the tuple in $B$ is below a certain threshold $\beta$. Queue $B$ maintains the candidates and facilitates uniform sampling of initial states. The tuples in $B$ are uniformly populated within the threshold $\beta$. With a high threshold $\alpha$, RL tends to revisit phases near the best-so-far policies. With a lower threshold, RL can be adventurous to explore the phase space more aggressively. In our experiments, $\alpha$ is density above the top 30 percent of the tuples in the current queue $A$ and $\beta = 0.5$. Queues $A$ and $B$ accommodate up to $2.5 \times 10^5$ and $5 \times 10^2$ tuples, respectively.

## 6 EXPERIMENTS

In our system, the optimization and physics simulation parts are written in C++. The simulation system is based on Dart [Lee et al. 2018] and linear-time stable PD controllers [Yin and Yin 2020]. The simulation timestep is 120 Hz and the control timestep is 30 Hz. The reinforcement learning and the assembler network are written in Python with TensorFlow2 [Abadi et al. 2016]. The policy network consists of four fully-connected layers of 1024 nodes, and the value

Table 1. Learning parameters.

| | |
|---|---|
| Learning rate of policy network | [5e-5, 2e-4, 5e-5] |
| Learning rate of value/assembler network | 1e-3 |
| Discount factor ($\gamma$) | 0.95 |
| GAE and TD ($\lambda$) | 0.95 |
| Clip parameter ($\epsilon$) | 0.2 |
| # of tuples per policy update | 25000 |
| Batch size for policy/value update | 512 |
| Batch size for assmebler update | 128 |

Table 2. Properties of the source characters.

| Properties | Humanoid | Bear | Horse | T-rex |
|---|---|---|---|---|
| Height (m) | 1.75 | 1.59 | 1.73 | 2.48 |
| Weight (kg) | 60.8 | 235 | 210.8 | 386 |
| links | 22 | 22 | 30 | 25 |
| Degree of freedom | 69 | 69 | 93 | 78 |

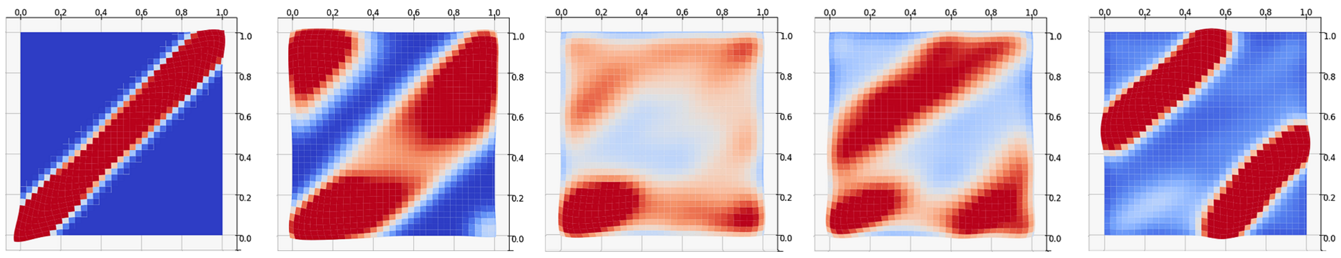
Fig. 5. The phase density in queue $A$ for the humanoid walking example in section 6.3. The X-coordinate represents the phase of the right leg, and the Y-coordinate represents the phase of the rest of the body. At the beginning of learning, the area around the initial phase alignments is densely populated. As the learning progresses, RL searches for better phase alignments and the high-density areas move accordingly.

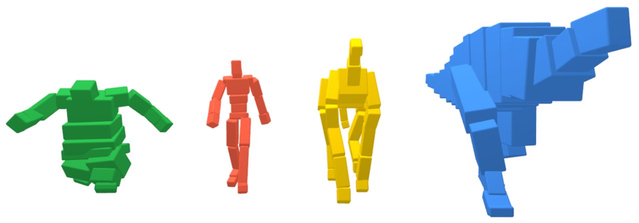
Fig. 6. Four source characters. Bear, humanoid, horse and T-rex (left to right).

network and the assembler network have two fully-connected layers of 512 nodes with ReLU activation. The learning parameters are summarized in Table 1. The learning rate of the policy network is $5 \times 10^{-5}$ at the beginning of learning. The learning rate increases gradually to its maximum $2 \times 10^{-4}$ when one million tuples are collected and then it decreases down to $5 \times 10^{-5}$ again.

We use four source characters for our demonstrations: Humanoid, Bear, Horse and T-rex (See Figure 6). The humanoid, the bear and the T-rex are bipedal, while the horse is quadrupedal. The humanoid and the bear have the same skeletal structure. The details of each character are summarized in Table 2. We designed chimeras from these source characters using our interactive skeleton editor, which allows copy-and-paste of body parts. The humanoid and the bear have four action skills { walk, run, jump, punch }. The horse has five action skills { walk, trot, run, jump, kick }, while the T-rex has two action skills { walk, tail-swing }. Each motion clip is short, less than 5 seconds. The characters and motion sets are available at the Unreal marketplace.

We used a Nvidia RTX 3070 to train our chimeras. It took 2 to 8 hours and 5 million to 30 million tuples to learn a single chimera motion. The training time mainly depends on mass distribution and the number of part animations. Policy learning is easier and more efficient with well-balanced body designs. A biased mass distribution makes it difficult to balance.

## 6.1 Chimeras and their Motor Skills

We generated a diverse set of chimera animations from the source characters (see Figure 7). The source body parts are resized to fit the design of chimera. The body mass is scaled in proportion to the volume change. Body parts in the same color share the same phase function. Body parts in different colors move in different phases although they come from the same source character. Chimeras can have highly variable structures, from well-known ones such as centipedes, four-legged dinosaurs and centaurs to exotic ones such as six-armed humanoids, two-headed bears and eight-legged horses (see Figure 7 and Figure 8). All chimeras are physically simulated and interactively controllable. The results are best viewed in our video. In the video, the chimeras in gray with colored borders represent the kinematic composition of part animations, while the fully colored chimeras show the final motion learned through our system.

The *Bear-Legged Racing* features four chimeras: The four-legged T-rex, the bear with extra spinal nodes, the short-legged quadruped, and the bear-legged humanoid with the T-rex tail. All source characters have running motions along a straight line. The learned motor skills are parameterized by the target direction such that chimeras can steer along the curved track.

The *Centaur* consists of the upper-body of the humanoid and the lower-body of the horse. Even though the torso and the two arms are taken from the same humanoid, we separated them into different body parts (the torso $\phi_1$, the right arm $\phi_2$, the left arm $\phi_3$, and the horse lower-body $\phi_4$) so that their phases can align differently according to the gaits of the horse. When the human upper-body is paired with the horse walking, the arm swing of human walking is reproduced. When the human upper-body is paired with horse galloping, the swing phases of the arms are synchronized to match the phases of the front legs. The states of the centaur include the heightmap around the body so that the centaur can walk and run on randomly-generated uneven terrain. The centaur also learned to jump over obstacles.

The *Bear-Armed Humanoid* has the bear's right arm transplanted on the humanoid. It is equipped with various motor skills including { walk, run, attack, attack while walking, attack while running, attack while jumping }. It can also steer while walking and running. This example shows how periodic locomotion and aperiodic action

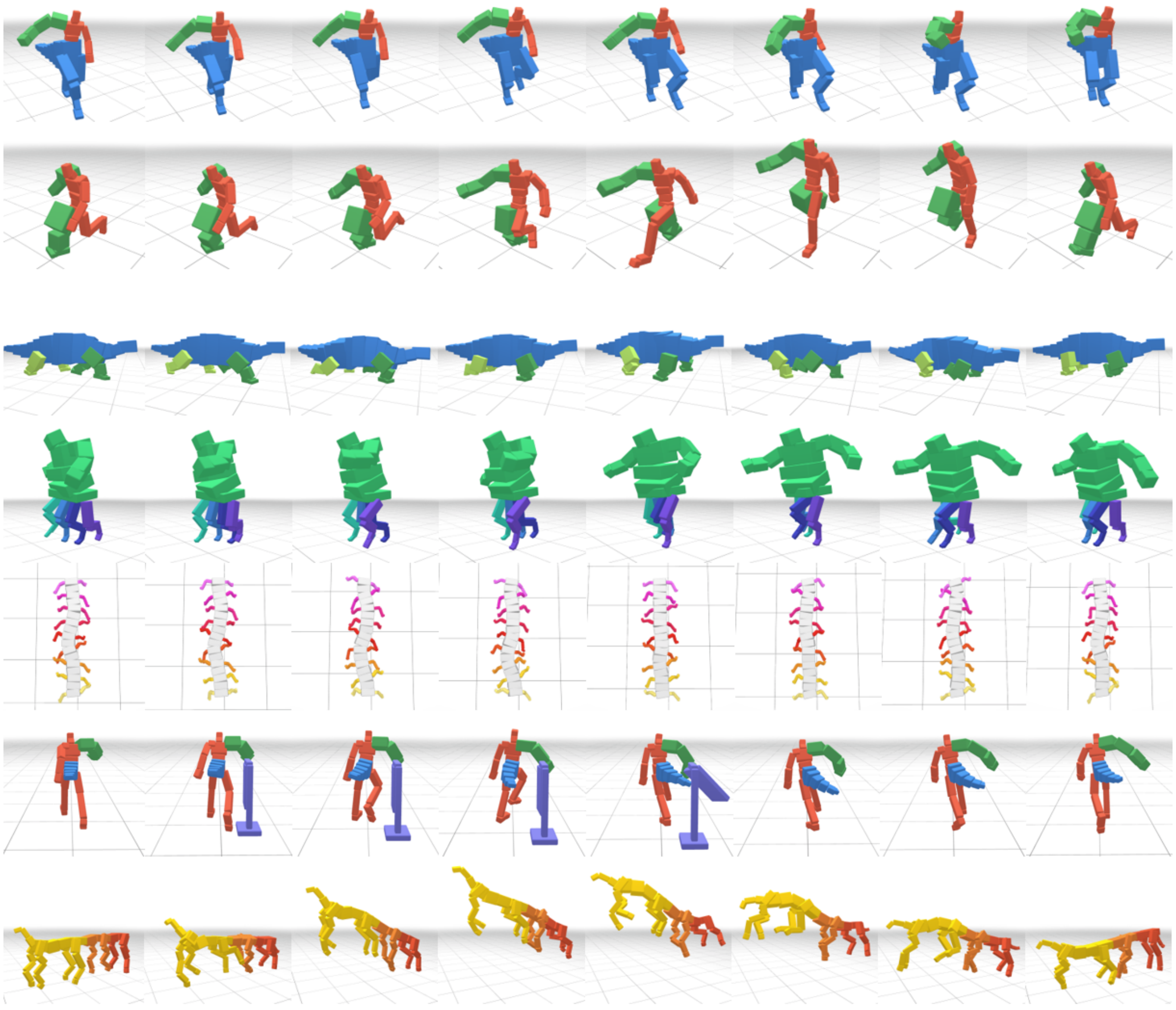
Fig. 7. Snapshots of chimeras.

can be combined and synchronized. To simulate the transitioning between actions, we trained additional policies that track interpolated motions.

The *Treant* consists of the right/left arms of the humanoid, two front legs and two hind legs of the horse, and the torso. The torso has no source motions and the compliant trunk joints respond freely to limb movements. The treant is equipped with four motor skills including { walk, punch, kick, and jump }. The walk motion is generated by combining horse galloping and humanoid walking. The punch motion is taken from the humanoid punching, while the kick motion is taken from the horse kicking the hind legs.

## 6.2 Ablation on Part Assembler

We conducted ablation studies on objective terms we designed for the part assembler. Three examples were used for the studies: The bear-armed humanoid attacking while running, The half-bear walking, and the eight-legged horse jumping. We compared the results optimized with a subset of the objectives and the results optimized

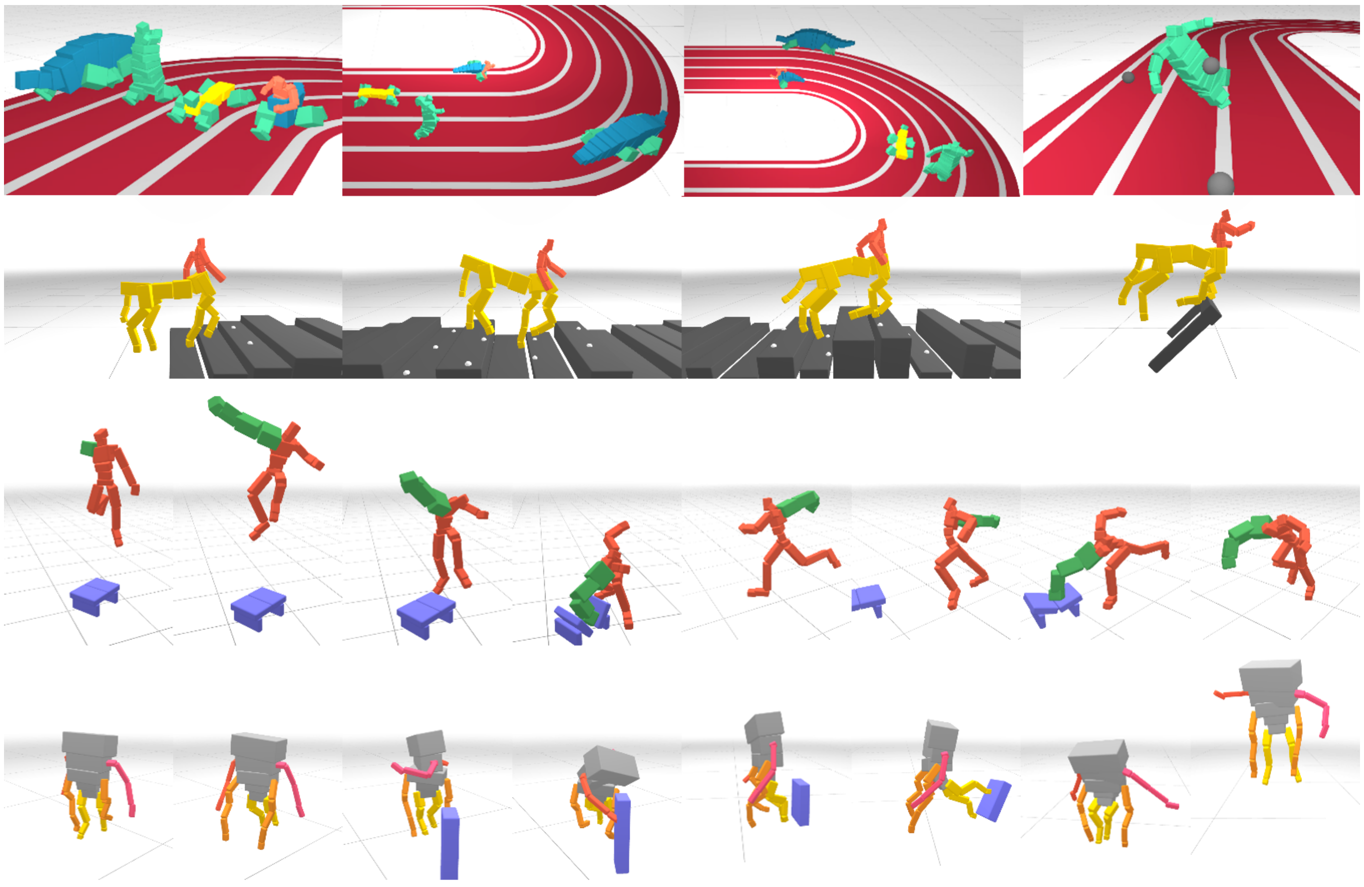
Fig. 8. Snapshots of the bear-legged racing, the centaur running on uneven terrain, the bear-armed humanoid and the treant (top to bottom).

with all objectives. Figure 9 shows the effectiveness of each objective term.

Without the global term, the context of source part motions in the global coordinate system is not preserved. The bear-armed humanoid punches in the wrong direction, the half-bear swings only halfway, and the spine of the horse curves towards the ground (see the second row of Figure 9). Without the contact term, the contact information of source animations is not preserved. The attack motion hits in the air, the short leg of the half-bear does not touch the ground, and the horse legs penetrate through the ground (see the third row of Figure 9). Without the regularization term, the resulting motion deviates from the source animations and often looks noisy due to the lack of temporal coherence. Our algorithm including all objective terms generated the best results preserving the context of source animations. The results can be best viewed in the supplementary video.

## 6.3 Natural Gait Discovery

In this section, we validate the effectiveness of our algorithm by comparing it with baseline algorithms. The baseline B1 algorithm imitates the initial composition by DRL, but does not allow time warping [Peng et al. 2018]. The baseline B2 algorithm uses a single timewarp function for the whole body [Lee et al. 2021a]. Our algorithm uses multi-dimensional timewarp functions to search for part-wise temporal alignments. We compared the three algorithms in terms of CoT and average torque over a period over all joints (see table 3).

Table 3. Energy efficiency.

| Task | Property | B1 | B2 | Ours |
|---|---|---|---|---|
| Humanoid - walk | CoT | 18.14 | 20.17 | 9.62 |
| | Torque $(N \cdot m)$ | 155.59 | 171.99 | 144.00 |
| Humanoid - run | CoT | - | - | 20.68 |
| | Torque $(N \cdot m)$ | - | - | 273.63 |
| Horse - velocity | CoT | - | 16.13 | 11.38 |
| | Torque $(N \cdot m)$ | - | 947.61 | 794.16 |
| Six-legged - walk | CoT | 11.63 | 10.43 | 8.15 |
| | Torque $(N \cdot m)$ | 351.76 | 363.80 | 318.60 |

The test sets were designed to verify if our algorithm can discover natural gait patterns. To do so, we separated the humanoid character into two body groups, the right leg and the others, such that the

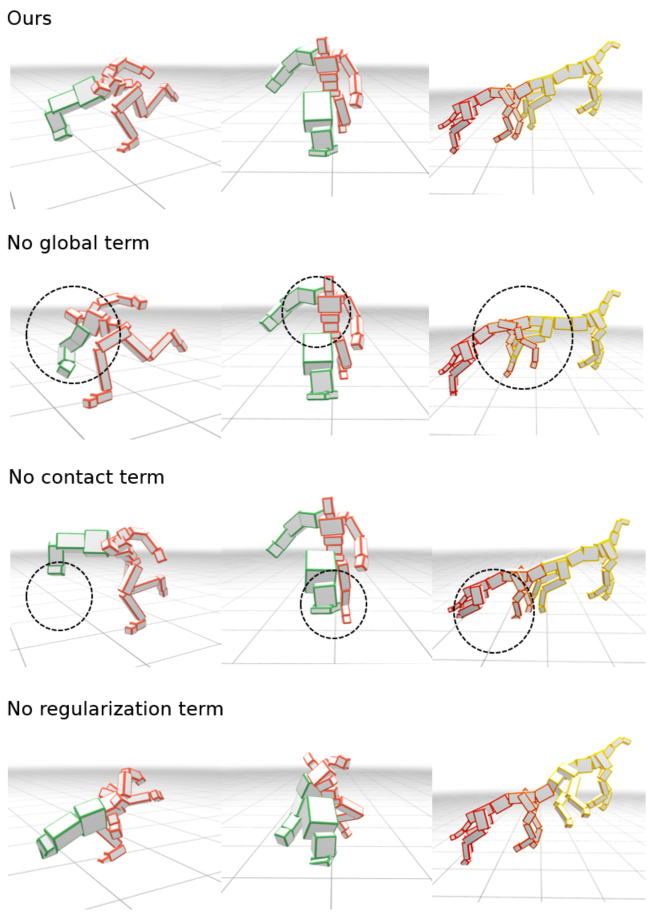

Fig. 9. Snapshots of ablations on the part assembler

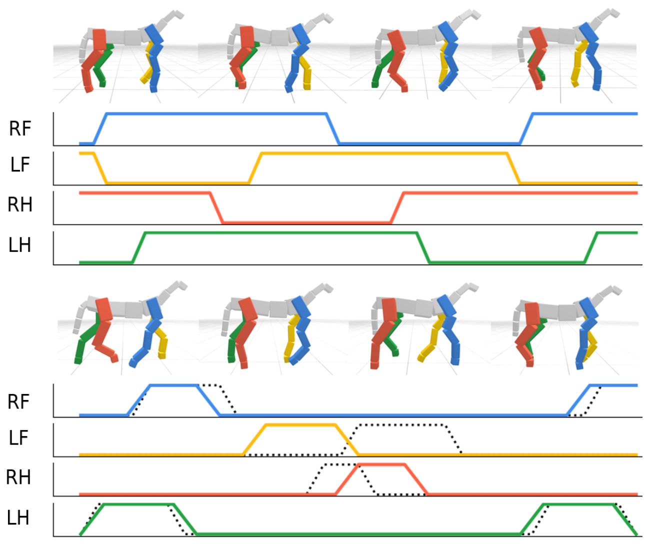

Fig. 10. The change of quadrupedal gait according to the change of speed. The target velocity is 1.0 m/s in the upper graph and 2.4 m/s in the lower graph. Each line represents a sequence of contacts with the ground. The solid line is taken from the simulation, while the dotted line is taken from the actual trot data.

right leg and the left leg can have different phases. The source motion of the right leg is the mirror reflection of the humanoid walking motion. Therefore, in the initial animation, both legs move back and forth synchronously (see Figure 5(left)). The baseline B1 and B2 algorithms result in hopping motions because part-wise phase aligning is unavailable. Our algorithm successfully recovered a natural biped gait with its two legs swinging alternatingly (see Figure 5(right)). As expected, our algorithm generated more energy-efficient motions than the baseline algorithms. Multi-dimensional timewarping also achieves better flexibility, better stability, and better computationally efficiency even with more parameters to optimize. A similar result was obtained with the running motion.

Horses perform different gaits depending on their moving speed. For example, horses walk at low speed and trot at higher speed. We tested if our algorithm could discover the trot gait starting from the walking gait. The horse character is set up to have five body groups: the torso and four limbs. Each body group is assigned an individual phase. The initial animation is a cycle of a horse walking. Our algorithm closely reproduced the initial animation at low speed (the target velocity $1m/s$). As the target speed increases, stance phases become shorter. The trotting gait is discovered when the target speed is $2.4m/s$. Table 3 shows that the trot is more energy-efficient than the walk at high speed. Increasing the target speed further did not result in a transition to galloping. The trot-to-gallop transition is detrimental in terms of energy efficiency but beneficial in terms of musculoskeletal forces [Farley and Taylor 1991]. Realistic musculoskeletal modeling would be needed to reproduce this transition.

It is well-known that insects perform a tripod gait when running. Insects move three legs simultaneously while having the other three legs in contact with the ground. Our six-legged T-rex learned a similar tripod gait. The chimera has the body of the T-rex and six humanoid legs. In the initial animation, all legs on one side swing forwards simultaneously while the legs on the other side move in the opposite direction. Multi-dimensional timewarp functions discovered the natural insect gait without any prior knowledge of how insects move because the tripod gait is energy-efficient and advantageous for balancing.

## 6.4 Comparison of state initialization methods

We compared our visit-based state initialization method with three baseline methods. The first baseline algorithm always begins at a fixed point, $\Phi = 0$. The second algorithm randomly chooses initial states in the fixed range set up by initial conditions [Peng et al. 2018]. The third algorithm chooses initial states adaptively based on marginal values and MCMC sampling [Won and Lee 2019]. The states with higher marginal values are sampled more frequently.

We conducted experiments with three examples in section 6.3: Humanoid-walk, Humanoid-run and Six-legged-walk. The initial phase alignments of these examples were far from the optimal found in section 6.3. Figure 11 shows the learned phases. In all three examples, fixed-point or fixed-range algorithms fell into local minima close to the initial alignments. The value-based algorithm successfully finds the optimal phase alignments for Humanoid-walk, but it fell into local optima for the other two examples. The CoT of Humanoid-run learned by the value-based algorithm is 26.0 and the average torque is $283.0Nm$. For Six-legged-walk, The CoT is 14.80 and the average torque is $370.0Nm$, which is even higher than those learned with the fixed initial time alignment (see Table 3). Compared to the baseline algorithms, our algorithm is better in terms of finding the globally optimal solution.

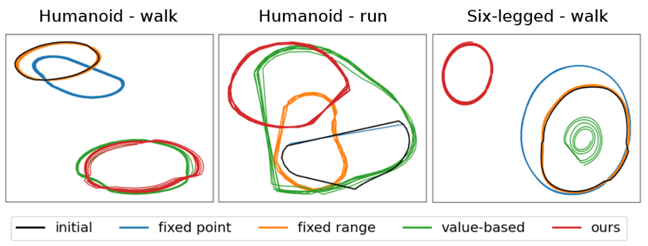

Fig. 11. Visualization of phase alignments. Periodic trajectories are visualized as closed curves. The fixed-point algorithm failed to learn Humanoid-run.

## 6.5 Exploration vs Computation Cost

In this section, we evaluate the effect of threshold $\alpha$ in DRL. Low threshold values are supposed to encourage more aggressive exploration in phase space. To do so, we generated three centaurs (denoted by M1, M5, and M10) with different mass distributions. M1, M5, and M10 have upper-to-lower body weight ratios of 1/19, 5/19, and 10/19, respectively. Since the lower body of a horse is heavier than the upper body of a man, arm swing phases tend to synchronize with front leg swing phases. This trend is more pronounced in M10 because the upper-body dynamics have a greater effect on whole-body coordination. The DRL algorithms fail to synchronize arm swing and leg swing when it converges to a local minimum. Figure 12 shows how threshold $\alpha$ affects the arm swing phase of centaurs. All three models fell into local minima near the initial phase alignment when $\alpha = 0.95$. Smaller threshold values allow the algorithm to explore the phase space more aggressively and thus provide better chances to escape from local minima. There exists a trade-off between better exploration and computational cost. We need to make a reasonable choice between two extremes.

## 7 DISCUSSION

We propose an algorithm that generates animations of various chimeras from source motions and characters. The kinematic optimization plays an important role in maintaining style/semantics of the motions, and the physics simulation adds dynamic effects such as weight shift and balancing that do not exist in the existing source motion. Conversely, given a task, it would be possible to find a suitable chimera skeleton. The work of Zhao et al. [2020] finds the structure of a robot that can best traverse a given terrain. In a similar sense, as our algorithm can synthesize various actions, it can be optimized for various tasks, such as finding the structure of a chimera that can fight best.

We confirmed through experiments that our algorithm can create highly variable combinations from very limited source motions and characters. If the increased number of source motions and characters is given, it will be possible to create infinitely many combinations of chimera motions. In order to generate a massive amount of chimera animations, the need for an automatic system increases. Currently, there are many parts that the user has to manually decide, such as chimera designs, the parameter for global constraints in part assembly, and PD gains for the physics simulation. If these parts are automated, the system can be conveniently applied to actual products such as games or animation. Additionally, learning a universal control policy for multiple body structure would increase the time and memory efficiency of the system.

Learning time largely depends on part assembly. This is especially costly if the character has high degrees of freedom or if the motion needs to deviate a lot from the initial position to satisfy the constraints. We dramatically reduce the total learning time by learning a network with the optimization results. Still, the increased number of part animations enlarges the number of optimizations exponentially and increases the training time of the regression network. We have learned up to 4 part animations (treant) using part assembly. In the case of the centipede with 9 part animations, we used the initial position as the reference motion without optimization. The types of chimera animation that can be created without the part assembly process are extremely limited. Additional methods to reduce the computation of part assembly process will be helpful for the scalability of the system.

Editing the timing of the motion by several phase variables greatly increases the range of resulting motions. The main mechanism controlling the phase in our algorithm is energy efficiency at the joint torque level, but it does not always work. We could not reproduce the trot-to-gallop transition, which requires task design at the musculoskeletal level. A variety of meaningful time alignments can be discovered through realistic modeling and reward designs such as metabolic energy, ground reaction force, or symmetry. Though our algorithm mainly deals with chimera animation, multiple phase variables can be applied to various motion synthesizing problems, such as style transfer or synchronizing with audio. We believe our work shows the potential of editing motion in a time domain, and we expect many interesting variations.

## ACKNOWLEDGMENTS

This work was supported by NCSOFT AI Center under Project Number NCSOFT2022-0982-K.

## REFERENCES

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 265–283.
Kfir Aberman, Rundi Wu, Dani Lischinski, Baoquan Chen, and Daniel Cohen-Or. 2019. Learning Character-Agnostic Motion for Motion Retargeting in 2D. *ACM Trans.*

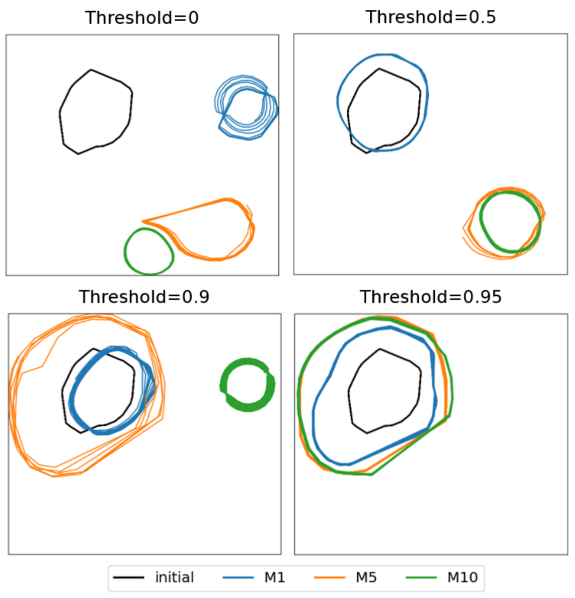

Fig. 12. Arm swing phase visualization for centaurs of various mass distributions. Threshold=0 means no threshold, and threshold=0.95 means accepting only the top 5 percent.

*Graph.* 38, 4, Article 75 (2019).
Kevin Bergamin, Simon Claver, Daniel Holden, and James Richard Forbes. 2019. DReCon: Data-Driven Responsive Control of Physics-Based Characters. *ACM Trans. Graph.* 38, 6, Article 1 (2019).
James M. Bern, Kai-Hung Chang, and Stelian Coros. 2017. Interactive Design of Animated Plushies. *ACM Trans. Graph.* 36, 4, Article 80 (2017).
Stelian Coros, Sebastian Martin, Bernhard Thomaszewski, Christian Schumacher, Robert Sumner, and Markus Gross. 2012. Deformable Objects Alive! *ACM Trans. Graph.* 31, 4, Article 69 (2012).
Claire T Farley and C Richard Taylor. 1991. A mechanical trigger for the trot-gallop transition in horses. *Science* 253, 5017 (1991).
Thomas Geijtenbeek, Michiel van de Panne, and A. Frank van der Stappen. 2013. Flexible Muscle-based Locomotion for Bipedal Creatures. *ACM Trans. Graph.* 32, 6, Article 206 (2013).
Michael Gleicher. 1998. Retargetting Motion to New Characters. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98).* 33–42.
N. Hansen and A. Ostermeier. 1996. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In *Proceedings of IEEE International Conference on Evolutionary Computation.* 312–317.
Rachel Heck, Lucas Kovar, and Michael Gleicher. 2006. Splicing upper-body actions with locomotion. 25, 3 (2006), 459–466.
Chris Hecker, Bernd Raabe, Ryan W. Enslow, John DeWeese, Jordan Maynard, and Kees van Prooijen. 2008. Real-time Motion Retargeting to Highly Varied User-Created Morphologies. In *Proceedings of ACM SIGGRAPH '08.*
Edmond S. L. Ho, Taku Komura, and Chiew-Lan Tai. 2010. Spatial Relationship Preserving Character Motion Adaptation. *ACM Trans. Graph.* 29, 4 (2010), 33:1–33:8.
Jessica K Hodgins and Nancy S Pollard. 1997. Adapting simulated behaviors for new characters. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques.* ACM, 153–162.
Daniel Holden, Taku Komura, and Jun Saito. 2017. Phase-Functioned Neural Networks for Character Control. *ACM Trans. Graph.* 36, 4, Article 42 (2017).
Wenlong Huang, Igor Mordatch, and Deepak Pathak. 2020. One policy to control them all: Shared modular policies for agent-agnostic control. In *International Conference on Machine Learning.* 4455–4464.
Leslie Ikemoto and David A Forsyth. 2004. Enriching a motion collection by transplanting limbs. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation.* 99–108.
Deok-Kyeong Jang, Soomin Park, and Sung-Hee Lee. 2022. Motion Puzzle: Arbitrary Motion Style Transfer by Body Part. *ACM Trans. Graph.* (2022).
Won-Seob Jang, Won-Kyu Lee, In-Kwon Lee, and Jehee Lee. 2008. Enriching a motion database by analogous combination of partial human motions. *The Visual Computer* 24, 4 (2008), 271–280.
Taeil Jin, Meekyoung Kim, and Sung-Hee Lee. 2018. Aura Mesh: Motion Retargeting to Preserve the Spatial Relationships between Skinned Characters. *Computer Graphics Forum* 37, 2 (2018), 311–320.
Manmyung Kim, Kyunglyul Hyun, Jongmin Kim, and Jehee Lee. 2009. Synchronized multi-character motion editing. *ACM Trans. Graph.* 28, 3, Article 79 (2009).
Paul G Kry, Lionel Revéret, François Faure, and M-P Cani. 2009. Modal locomotion: Animating virtual characters with natural vibrations. In *Computer Graphics Forum.* 289–298.
Jehee Lee. 2008. Representing Rotations and Orientations in Geometric Computing. *IEEE Computer Graphics and Applications* 28, 2 (2008), 75–83.
Jeongseok Lee, Michael X Grey, Sehoon Ha, Tobias Kunz, Sumit Jain, Yuting Ye, Siddhartha S Srinivasa, Mike Stilman, and C Karen Liu. 2018. DART: Dynamic animation and robotics toolkit. *The Journal of Open Source Software* 3, 22 (2018).
Jehee Lee and Sung Yong Shin. 1999. A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques.* 39–48.
Kyungho Lee, Sehee Min, Sunmin Lee, and Jehee Lee. 2021b. Learning Time-Critical Responses for Interactive Character Control. *ACM Trans. Graph.* 40, 4, Article 147 (2021).
Seyoung Lee, Sunmin Lee, Yongwoo Lee, and Jehee Lee. 2021a. Learning a Family of Motor Skills from a Single Motion Clip. *ACM Trans. Graph.* 40, 4, Article 93 (2021).
Seunghwan Lee, Moonseok Park, Kyoungmin Lee, and Jehee Lee. 2019. Scalable Muscle-actuated Human Simulation and Control. *ACM Trans. Graph.* 38, 4, Article 73 (2019).
Yoonsang Lee, Sungeun Kim, and Jehee Lee. 2010. Data-driven Biped Control. *ACM Trans. Graph.* 29, 4, Article 129 (2010).
Hung Yu Ling, Fabio Zinno, George Cheng, and Michiel Van De Panne. 2020. Character controllers using motion vaes. *ACM Trans. Graph.* 39, 4, Article 40 (2020).
Libin Liu, Michiel Van De Panne, and Kangkang Yin. 2016. Guided Learning of Control Graphs for Physics-Based Characters. *ACM Trans. Graph.* 35, 3, Article 29 (2016).
Libin Liu, KangKang Yin, Bin Wang, and Baining Guo. 2013. Simulation and Control of Skeleton-driven Soft Body Characters. *ACM Trans. Graph.* 32, 6, Article 215 (2013).
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 Trans. Graph.* 39, 4, Article 38 (2020), 10 pages.
Josh Merel, Saran Tunyasuvunakool, Arun Ahuja, Yuval Tassa, Leonard Hasenclever, Vu Pham, Tom Erez, Greg Wayne, and Nicolas Heess. 2020. Catch Carry: Reusable Neural Controllers for Vision-Guided Whole-Body Tasks. *ACM Trans. Graph.* 39, 4, Article 39 (2020).
Sehee Min, Jungdam Won, Seunghwan Lee, Jungnam Park, and Jehee Lee. 2019. SoftCon: Simulation and Control of Soft-Bodied Animals with Biomimetic Actuators. *ACM Trans. Graph.* 38, 6, Article 208 (2019).
Igor Mordatch, Kendall Lowrey, Galen Andrew, Zoran Popovic, and Emanuel V Todorov. 2015. Interactive control of diverse complex characters with neural networks. *Advances in neural information processing systems* 28 (2015).
Uldarico Muico, Jovan Popović, and Zoran Popović. 2011. Composite Control of Physically Simulated Characters. *ACM Trans. Graph.* 30, 3, Article 16 (2011).
Jungnam Park, Sehee Min, Phil Sik Chang, Jaedong Lee, Moon Seok Park, and Jehee Lee. 2022. Generative GaitNet. In *Proceedings of ACM SIGGRAPH '22.*
Soohwan Park, Hoseok Ryu, Seyoung Lee, Sunmin Lee, and J. Lee. 2019. Learning predict-and-simulate policies from unorganized human motion data. *ACM Trans. Graph.* 38, 6, Article 205 (2019).
Deepak Pathak, Christopher Lu, Trevor Darrell, Phillip Isola, and Alexei A Efros. 2019. Learning to control self-assembling morphologies: a study of generalization via modularity. *Advances in Neural Information Processing Systems* 32 (2019).
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 Trans. Graph.* 37, 4, Article 143 (2018).
Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. 2021. AMP: Adversarial Motion Priors for Stylized Physics-Based Character Control. *ACM Trans. Graph.* 40, 4, Article 144 (2021).
Daniele Reda, Hung Yu Ling, and Michiel Van De Panne. 2022. Learning to Brachiate via Simplified Model Imitation. In *Proceedings of ACM SIGGRAPH '22.*
Christian Schulz, Christoph von Tycowicz, Hans-Peter Seidel, and Klaus Hildebrandt. 2014. Animating Deformable Objects Using Sparse Spacetime Constraints. *ACM Trans. Graph.* 33, 4, Article 109 (2014).
Yeongho Seol, Carol O'Sullivan, and Jehee Lee. 2013. Creature features: online motion puppetry for non-human characters. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* 213–221.
Hyun Joon Shin, Jehee Lee, Sung Yong Shin, and Michael Gleicher. 2001. Computer puppetry: An importance-based approach. *ACM Trans. Graph.* 20, 2 (2001), 67–94.
Karl Sims. 1994. Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques.* 15–22.

Kwang Won Sok, Manmyung Kim, and Jehee Lee. 2007. Simulating biped behaviors from human motion data. *ACM Trans. Graph.* 26, 3, Article 107 (2007).

Sebastian Starke, Ian Mason, and Taku Komura. 2022. DeepPhase: Periodic Autoencoders for Learning Motion Phase Manifolds. *ACM Trans. Graph.* 41, 4 (2022).

Sebastian Starke, Yiwei Zhao, Fabio Zinno, and Taku Komura. 2021. Neural Animation Layering for Synthesizing Martial Arts Movements. *ACM Trans. Graph.* 40, 4, Article 92 (2021).

Jie Tan, Greg Turk, and C. Karen Liu. 2012. Soft Body Locomotion. *ACM Trans. Graph.* 28, 3, Article 26 (2012).

Tianxin Tao, Matthew Wilson, Ruiyu Gou, and Michiel Van De Panne. 2022. Learning to Get Up. In *Proceedings of ACM SIGGRAPH '22*.

C Richard Taylor, Knut Schmidt-Nielsen, and Jacob L Raab. 1970. Scaling of energetic cost of running to body size in mammals. *American Journal of Physiology-Legacy Content* 219, 4 (1970).

Vance A. Tucker. 1970. Energetic cost of locomotion in animals. *Comparative Biochemistry and Physiology* 34, 4 (1970).

Kevin Wampler and Zoran Popović. 2009. Optimal gait and form for animal locomotion. *ACM Trans. Graph.* 28, 3, Article 60 (2009).

Kevin Wampler, Zoran Popović, and Jovan Popović. 2014. Generalizing Locomotion Style to New Animals with Inverse Optimal Regression. *ACM Trans. Graph.* 33, 4, Article 49 (2014).

Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. 2018. Nervenet: Learning structured policy with graph neural networks. In *International conference on learning representations*.

Jungdam Won, Deepak Gopinath, and Jessica Hodgins. 2020. A Scalable Approach to Control Diverse Behaviors for Physically Simulated Characters. *ACM Trans. Graph.* 39, 4, Article 33 (2020).

Jungdam Won and Jehee Lee. 2019. Learning body shape variation in physics-based characters. *ACM Trans. Graph.* 38, 6, Article 207 (2019).

Jungdam Won, Jongho Park, Kwanyu Kim, and Jehee Lee. 2017. How to Train Your Dragon: Example-Guided Control of Flapping Flight. *ACM Trans. Graph.* 36, 6, Article 198 (2017).

KangKang Yin, Kevin Loken, and Michiel Van de Panne. 2007. Simbicon: Simple biped locomotion control. *ACM Trans. Graph.* 26, 3, Article 105 (2007).

Zhiqi Yin, Zeshi Yang, Michiel Van De Panne, and Kangkang Yin. 2021. Discovering Diverse Athletic Jumping Strategies. *ACM Trans. Graph.* 40, 4, Article 91 (2021).

Zhiqi Yin and KangKang Yin. 2020. Linear Time Stable PD Controllers for Physics-Based Character Animation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 191–200.

Wenhao Yu, Greg Turk, and C. Karen Liu. 2018. Learning Symmetric and Low-Energy Locomotion. *ACM Trans. Graph.* 37, 4, Article 144 (2018).

He Zhang, Sebastian Starke, Taku Komura, and Jun Saito. 2018. Mode-Adaptive Neural Networks for Quadruped Motion Control. *ACM Trans. Graph.* 37, 4, Article 145 (2018).

Allan Zhao, Jie Xu, Mina Konaković Luković, Josephine Hughes, Andrew Speilberg, Daniela Rus, and Wojciech Matusik. 2020. RoboGrammar: Graph Grammar for Terrain-Optimized Robot Design. *ACM Trans. Graph.* 39, 6 (2020).