공학박사학위논문

Physics-based Motion Editing for Diverse Conditions and Tasks

다양한 조건에 대한 물리 기반 동작 편집

2023 년 2 월

서울대학교 대학원

컴퓨터 공학부

이 세 영

Physics-based Motion Editing for Diverse Conditions and Tasks

다양한 조건에 대한 물리 기반 동작 편집

지도교수 서진욱

이 논문을 공학박사 학위논문으로 제출함

2023 년 1 월

서울대학교 대학원

컴퓨터 공학부

이세영

이세영의 박사 학위논문을 인준함

2023 년 1 월

Abstract

A rich repertoire of motion data is required to make the virtual character move like a real human. To efficiently acquire motion data, there has been many researches on motion editing algorithm that modifies the source motion to satisfy the novel task/condition. The purpose of this study is to generate various motions through physics-based motion editing. Physics-based character animation is a method of generating motion by moving a character in a physical environment. Combining motion editing and physics-based animation has merit that the physically simulated character can explore novel task/condition deviating from the source motion. The key challenge is to design an algorithm that encourages deformation in desirable directions while inhibiting deformation in unnecessary directions. By combining motion optimization with physics simulation, we have significantly increased the quality and diversity of generated motion.

Our thesis propose three physics-based approach that edit motion for various tasks/conditions and motions. The first work propose an algorithm that creates a motion that satisfies a wide task space from a single motion clip. Simultaneous learning of motion parameterization and motor skills significantly improves the performance and visual quality of learned motor skills. The second work provides an algorithm that creates the animation for a new character by combining multiple characters and motion clips. Our algorithm finds proper spatial and temporal alignments of its composing parts considering physical properties of the character and the motion. The third work discusses motion synthesis for style which is an implicit task. Our algorithm searches the style feature directly from the source motions and correct the style feature in the physical environment to enable style transfer from a small number of motion data. We demonstrate the versatility of our physics-based motion editing algorithms with highly dynamic motor skills for novel conditions, body structures and styles.

Keywords: Computer Animation, Physics Simulation, Physics-based Control, Data-Driven Animation, Deep Reinforcement Learning, Motion Synthesis, Motion Editing

Student Number: 2017-25468

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Movements of a real human are very diverse, from everyday movements such as walking, running, and interacting with objects to dynamic movements such as dancing, gymnastics, and various exercises. Reproducing these rich movements of real human with a virtual character is a key challenge in the field of computer animation. The quality and amount of motion data greatly affects life-likeness of the character's movement. Most widely used method to obtain new motion data is by motion capture or manual key-framing. Motion capture requires a laborious process of recruiting actors, setting up the camera, capturing, and post-processing. Manual key-framing is also time consuming in that artists have to draw all the frames. It is practically impossible to acquire all necessary motion data by the above method.

Motion synthesis that generates new motions from existing motions has always been an important topic. The purpose of motion synthesis is diverse: interpolating motion to match user input, generating smooth transitions between motions, or discovering motions that satisfy new tasks. Among them, we are interested in editing the motion to satisfy the novel task and condition. To build good motion editing system, three things should be considered. The most fundamental element is diversity. The system must be able to edit the motion widely outside the given data range. To do so, the system should be able to handle broad range of motions and tasks/conditions. The conditions and tasks that are the objective of the motion editing can be very divergent. They can be the character configurations such as body length and mass distribution, space-time constraints such as velocity or position of end effectors, and environmental conditions such as gravity and the weight of interacting objects.

The second is the quality of the motion. The quality is determined comprehensively by how well the generated motion preserves various elements of original motion. Elements such as shape/trajectory of the joint rotation, and contact with the ground or object have a great influence on the semantic and human-likeness of the motion. The physical properties of the motion such as gravity, frictional force, the maximum force and the mass distribution of the character are another important elements. These elements are not explicitly shown in the motion, but they affect the physical plausibility of the movement and make the movement realistic. The third is speed. A system that can generate a new motion as quickly as possible is a better system if diversity and quality are guaranteed. The purpose of this thesis is to improve the diversity of motion editing systems without compromising the other two factors as much as possible.

In motion editing problems, the motion is optimized to satisfy the given task and preserve the quality. Most of the previous work on motion editing use a data-driven approach. These methods directly control the rotation of each joint through spacetime optimization or learning network. The quality of the motion can be evaluated from selected properties of a source motion such as shape and contact event, or from the overall similarity with the given motion dataset. Because physical plausibility is not specifically considered, the generated motion violates the laws of physics when it deviates from the motion dataset.

Physics-based character animation is a method of generating motion by moving a character in a physical environment. Instead of directly controlling the joints, these methods apply force to each joint, causing the joint to move according to the laws of physics. Combining motion editing and physics-based animation has merit that the generated motion always follows the laws of physics even if it deviates a lot from the source motion. This nature of physics-based animation allows for the discovery of new motions. However, due to the high complexity of the physics simulation, it was difficult to generate a rich repertoire of full body motion in the physical environment. Physics-based motion editing approaches used a simplified model or limited the range of motions that can be reproduced. These limited the diversity in a different way from the kinematic approach.

Recent deep reinforcement learning based policy learning approaches have achieved significant improvements in terms of stability, performance and scalability of control. Deep reinforcement learning made it possible to reproduce high-quality, highly dynamics motor skills by learning their control policies [1]. Recent studies demonstrated that DRL-based continuous control is not encumbered by the structure of the dynamics system and the type of motor skills.

In this thesis, we propose novel algorithms that generate various motions in the physical environment based on the high representation power of DRL. The key challenge is to the quality of the synthesized motion. Physics-based simulation itself does not guarantee that it will produce a human-like motion. As the motion deviates from the original motion, the movement of each joint becomes different from that of a real human. Therefore, the deformation of the motion cannot be entirely left to the physics simulation, and other methods are needed to encourage deformation in desirable directions while refraining deformation in unnecessary directions. To solve this problem, we used motion optimization along with physics simulation. We allowed some degree of freedom of motion to be freely deformed during the physics simulation process, and the rest to be constrained to motion optimization results. We designed proper optimization objectives for preserving the shape and the important events such as contact of the motion. We devised efficient methods to reduce the time for motion optimization. We propose three physics-based approach that edit motion for various tasks/conditions and motions. The first work propose an algorithm that creates a motion that satisfies a wide task space from a single motion clip, and the second work provides an algorithm that creates a new character and motion by combining multiple characters and motion clips. The third work introduces a method for style transfer from a small number of motion data.

Except for the issues mentioned above, there is an additional problem with how to formulate the task when editing for the implicit tasks. In the first two work, the task can be defined as a constant value so it is a simple process to evaluate how close the task of the generated motion and the task of the source motion are. On the other hand, a task such as style is implicitly inherent in motion so it is difficult to formulate it as a constant value. Style transfer problem has been mainly approached in a data-driven way, which infers a stylized motion from a model trained with a large amount of motion data. Adapting physics based simulation to motion style transfer can correct physical errors occurs in the process of applying the style, so more diverse stylized motions can be created. Finding an appropriate style representation suitable for a physics-based approach is another topic in this paper.

Learning a family of motor skills from a single motion clip. In

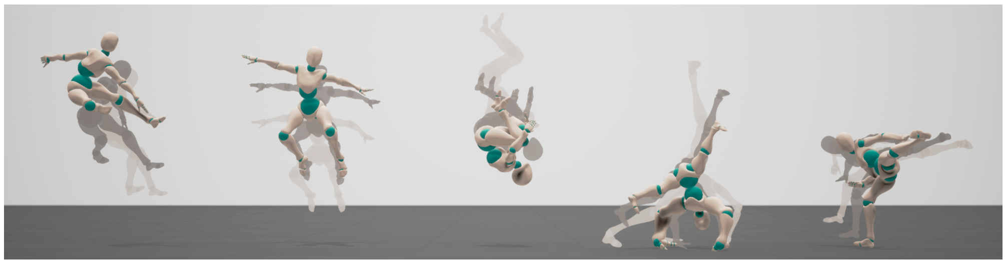

Figure 1.1: The parameterized motor skills of obstacle jump, jump, backflip, cartwheel and kick motions generated from single motion clip.

chapter 3, we present a new algorithm that learns a parameterized family of motor skills from a single motion clip. The motor skills are represented by a deep policy network, which produces a stream of motions in physics simulation in response to user input and environment interaction by navigating continuous action space. Three novel technical components play an important role in the success of our algorithm. First, it explicitly constructs motion parameterization that maps action parameters to their corresponding motions. Simultaneous learning of motion parameterization and motor skills significantly improves the performance and visual quality of learned motor skills. Second, continuous-time reinforcement learning is adopted to explore temporal variations as well as spatial variations in motion parameterization. Lastly, we present a new automatic curriculum generation method that explores continuous action space more efficiently. We demonstrate the flexibility and versatility of our algorithm with highly dynamic motor skills that can be parameterized by task goals, body proportions, physical measurements, and environmental conditions.

Learning Virtual Chimeras by Dynamic Motion Reassembly. In chapter 4, we present a novel algorithm that creates and animates chimeras by dynamically reassembling source characters and their movements. The *Chimera* is a mythological hybrid creature composed of different animal parts. Compos-

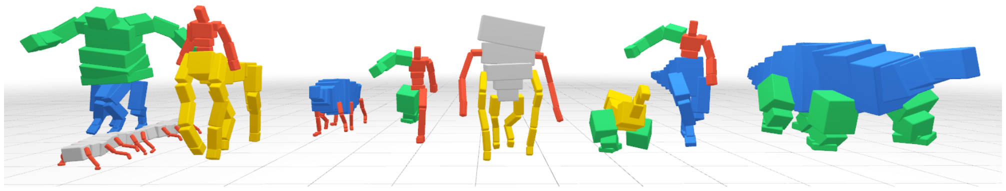
Figure 1.2: Physically synthesized locomotion of various chimeras generated from four source character and their walking motion.

ing different animal parts have a great impact on the dynamics of individual part motion because the body and motion of a creature are closely related to each other. We found that finding proper spatial and temporal alignments of its composing parts is key to the chimera motion. 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.

Learning Style Transfer from Minimal Motion Data While chapter 3 and 4 synthesize motion for a task/condition that can be expressed explicitly as a specific value, chapter 5 discusses motion synthesis for style which is an implicit task. Researchers have successfully conducted style transfer of diverse motion and styles by learning-based methods. Since this method depends on

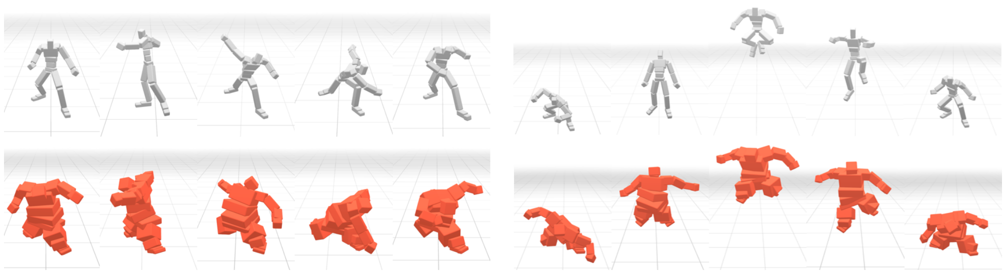

Figure 1.3: Input motion and stylized motion generated by our system.

the distribution of training data, it limits the skeleton structure and the range of styles that can be generated. We propose an algorithm for style transfer from a minimum amount of motion data. Our system consists of two sequential processes. Our algorithm first searches the style feature to be applied to the each frame of the input motion from the source motion pair. This search process is performed adaptively according to the similarity with the source motion, so that synthesizing can be performed efficiently from a small amount of data. Then synthesized motion is deformed in space and time to add proper physical effects and corrects errors related to interpenetration and timing. We adopt the work of Lee et al [2] to simulate a character in a physical environment and compensate for the lack of information about movement. We demonstrate the effectiveness of our system by generate diverse stylized motions from locomotion pair of 6 seconds.

Chapter 2

Background

High-quality motion capture data serve as building blocks of expressive character animation in movies, video games, and digital media. Each motion clip captures the motion of a specific subject in a specific environment and in a specific mood and style. In the field of computer graphics and computer vision, various studies have been conducted to increase the efficiency of motion data generation. Section 2.1 discusses researches on motion reconstruction from video or sensors, and section 2.2 introduces researches on synthesizing existing motions to generate new motion. Finally, section 2.3 provides background on motion generation with physics-based simulation.

2.1 Human Motion Reconstruction

The traditional motion capture process involves attaching dozens of markers to the body, calibrating the camera, capturing the motion, and manual post-processing. In order to improve the convenience of the process of motion capture, various markerless motion capture methods have been introduced to simplify the process of motion capture. Two main directions are to reconstruct motions with minimal sensors without attaching markers and to reconstruct motions from 2d video.

Diverse types of sensors have been used to replace markers. Representatively, many studies have been conducted on attaching IMU sensors to the body [3, 4, 5], and using head mount displays like AR/VR devices [6, 7, 5]. As the number of sensors decreases, the convenience increases but sensor-based motion capture alone cannot provide sufficient information about motion. A method of inferring motion from physical simulation or existing motion data is used to fill the missing information.

Video-based motion reconstruction has an advantage in that it can utilize many existing videos and does not require any equipment other than a camera. Various studies on motion reconstruction from video have been conducted in the computer vision community. Early approaches track the video motion from a statistical model learned with source motion data [8, 9]. Recently, deep learning models such as CNN, LSTM, and transformer are used to reconstruct the character's pose for various motions [10, 11, 12]. With the advent of SMPL [13], many researches that recover meshes of characters as well as joint position have been introduced [12, 14, 15]. On the other hand, physics-based simulation is adopted to generate motion for a moving camera or to reconstruct dynamic motion [16, 17].

2.2 Motion Synthesis

There is a large array of literature that explores the adaptation and reuse of motion capture data to deal with new conditions. We briefly introduce three motion synthesis techniques widely used in computer graphics: *blending*, *optimization* and *splicing*. Additionally, we provide the history of motion synthesis for three specific tasks and conditions which are related to our work.

Large motion datasets are a valuable source of constructing continuous, parameterized spaces of motion. *Motion blending* according to blending weights is a frequently used technique for character animation. RBF (Radial Basis Function) interpolation has been popular in early studies to deal with multivariate interpolation [18, 19, 20]. Learning-based techniques, such as GPLVM (Gaussian Process Latent Variable Model), allow more flexibility to learn a generative dynamic model from unorganized motion data [21, 22, 23]. Recently, deep learning approaches prevail in constructing scalable generative models from large, unorganized motion datasets [24, 25, 26].

Motion Optimization has been widely used in motion editing problems to transform a given motion task into a novel task. In this method, it is important to design appropriate objectives. The objectives in motion optimization problems commonly minimize the difference between the source motion and the generated motion, and the difference between the target task and the task of the generated motion [27, 28]. The variables for optimization are the degree of the freedoms of the movement, and they can be designed in various ways from the transformation of a single joint to the full body pose. The higher the dof, the greater diversity of the generated movement, but also the greater the possibility that the movement will be distorted unintentionally. In order to maintain the quality of motion, abstracted forms like control points of multilevel spline, or root or COM trajectory of the motion were used instead of fullbody pose [29, 30, 31].

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 [32, 33], analogy-based [34], and neural network-based [35] methods have been studied. The motion puzzle system recently developed by Jang et al. [36] locally transfers different styles for each body part through an attention-based neural network model.

Motion editing and motion reconstruction are both used for the motion generation, but motion reconstruction aims to reproduce the actual human's movement as much realistic as possible, and motion editing aims to discover motion different from existing motions. Motion reconstruction can create motion without a user-defined goal or task. Therefore, motion reconstruction has the advantage of conveniently reproducing various types of daily movements. On the other hand, motion editing can deal with wider range of motion because it can even generate the motion of imaginary character or in dangerous situation that is difficult to obtain from human movement.

2.2.1 Physics-Motivated Motion Editing

The data-driven motion editing methods mentioned above have difficulty dealing with dynamic motions or tasks. In order to compensate for these shortcomings, various studies have been conducted on reflecting the physical properties of motion in the motion editing process. Because of high complexity of physics simulation, previous methods used a simplified model or physics-motivated constraints rather than directly simulating character in the physics environment. McCann et al. [37] designed optimization terms for contact force to change the timing of the motion sequence. Pollard and Behmaram-Mosavat [38] calculated the simplified force model from kinematic motion data to control force of the motion. Modifying the COM trajectory rather than full body motion are methods usually chosen to edit the physical measures like momentum or the external force of dynamic motions [39, 30, 40]. Our DRL-based work performs full body motion control in physics simulation and deals with much diverse motions and tasks with a single framework.

2.2.2 Motion Retargeting

Motion retargeting methods transfer the motion of one character to another while maintaining the semantics of the original motion as much as possible [41, 31, 42, 43, 44, 45]. Contact and interaction events between characters are commonly selected as key aspects to be preserved for motion retargeting [41, 42, 43, 44, 29]. Most retargeting algorithms assumed that the structures of the source and target characters are similar. On the other hand, Hecker et al. [46] 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. [47] retargets the motion of human actors to imaginary creatures by learning the direct mapping between them. Wampler et al. [48] synthesized the gaits of a wide range of new creatures by learning coherent patterns from motion databases. Aberman et al. [49] generated character-agnostic motions by decomposing the motions into latent components. Our work reassembles the motions of multiple characters according to the user's intention so that the structure of the target character is not constrained by the those of the source characters.

2.2.3 Motion Style Transfer

Motion style transfer is a motion synthesis method that applies the style of the source motion to the target motion. The key to this technique is how to extract style features while preserving the contents of the motion. Previous work is divided into a method of directly extracting the difference from base motions and a stylized motions [50], and a method of inferring a feature from a large amount of motion data with diverse styles [51, 52, 36]. These data-driven methods does not guarantee physical correctness of the stylized motion. Adapting physics based simulation to motion style transfer can correct physical errors occurs in the process of applying the style. The work of Ma et al. [53] reproduced a stylized dynamic motion in a physics-based method through designing a heuristic reward. In Chapter 5, we designed the algorithm that learns the stylized motion in the physical environment from a small amount of source motion pairs.

2.3 Physics-based Control and Simulation

Physics-based character simulation and control have many advantages over kinematic/data-driven approaches. Physics-based method guarantees the generated motion is physically correct, generate interaction between the character and the environment such as terrain or obstacles, and make it possible to control elements that implicitly affects the motion, such as muscles. Designing high dimensional control system of an articulated body model has been a long-standing challenge in computer graphics. Early methods simplified complex problems by using hand crafted state transition rules and simplified models like lower-body-only models or inverted pendulum models [54, 55, 56]. These methods are combined with trajectory optimization to generate proper feedback for the simulation result. Trajectory optimization is the process of minimizing some performance measure, such as total joint torque and metabolic energy expenditure, while satisfying a set of constraints. Model predictive control (MPC), which solves trajectory optimization for a finite time-horizon repeatedly and execute only the first step of the trajectory to proceed, has also been studied in computer graphics literature [57, 58]. Since the dynamics of articulated bodies are not smooth at collision and contact, derivative-free, sampling based optimization methods, such as CMA-ES (Covariant Matrix Adaptation Evolutionary Steps), have often been exploited to address trajectory optimization [59, 60, 61, 62, 63].

Deep reinforcement learning based policy learning approaches have achieved significant improvements in terms of stability, performance and scalability of control. Although DRL is capable of learning complex motor skills from scratch without any reference motion to imitate, [64], the best human-like motor skills are often achieved when high-quality motion capture data are provided as a reference [1]. Recent studies demonstrated that DRL could learn an integrated control policy equipped with many heterogeneous motor skills from data-driven motion generators [65, 66]. Won *et al.* [67] demonstrated the scalability of DRL by learning a large-scale control policy from eight hours of motion data. The DRL-based methods have further been explored to diverse body shapes/proportions [68, 69], anatomical body modeling [70, 71], visiomotor control [72], improving controllability [73, 74] and animal locomotion [75, 76, 77, 78, 79]. To generate high-quality motion, deep reinforcement learning requires a delicate reward design. To reduce this burden, various GAIL-based method has been introduced recently [80, 81]. These methods evaluate the naturalness of simulated motion through a discriminator instead of a well-designed reward.

2.3.1 Discovering New Motor Skills

The physics-based approaches can reproduce novel motions without full reference motion unlike kinematic-based approaches. This is possible because movement is generated according to the physical properties of the character during the physics simulation. Generating motions from scratch by physics simulation has enabled various studies for cases where it is difficult to obtain reference motions. Mordatch et al. [82] and Al Borno et al. [83] generates full body movements with complex contact by sampling-based optimization. Yu et al. [84] and Kry et al. [85] simulated energy-efficient locomotion for various morphologies. The locomotion of soft bodies [76, 86, 87] and elastic models [88, 89, 90] have also been studied. Recently, DRL-based methods boost exploration and enable discovery of more complex motor skills, such as athletic movements, get-ups in various conditions and chopstick motions with diverse styles [91, 92, 93]. The method of discovering new motor skills without reference motion requires a delicate design of the mechanism of a movement. Use of partial reference motion can increase the diversity of generated motions and can make the character move according to the user's intentions [17, 75]. Discovering new motor skills by exploring physics environment is an attractive option given the nature of the motion editing problem, which requires generating novel motions from a limited amount of motion. Our work applies DRL-based physic simulation to the motion editing problem and increases the diversity of generated motion.

Chapter 3

Learning a Family of Motor Skills from a Single Motion Clip

3.1 Overview

Humans can learn new motor skills from a demonstration. A student observes an instructor performing a punch and learns to imitate the movement by providing adequate actuation torques at joints. Furthermore, the student may also learn to repurpose the learned motor skill for striking in different directions and forces. The key technical challenge of such generalization is repurposing motor skills based on physical and biological principles. For example, to punch harder, it is advisable to pull the arm back more and then swing the arm faster to generate bigger momentum. Making a change to the task requires harmonious coordination of the arm swing trajectory and its timing, linear/angular momentum, and force/torque at joints. Previous approaches often exploited a comprehensive set of motions to describe how motor skills generalize. Collecting such large datasets requires considerable time and effort.

In this paper, we present a new approach to construct a parameterized family of motor skills from a single motion clip. Our algorithm generalizes a base motor skill, that mimics the reference motion, to generate a wide variety of skills that meet novel conditions and goals. The parameterized motor skills are represented by a deep policy network, which produces a stream of motions in physics simulation in response to user input and environment interaction by navigating continuous task space.

Specifically, we need to address two sub-problems. One is to learn a motor skill that mimics an input motion, and the other is to adapt the input motion in a physically valid manner for a range of conditions and tasks, which form a continuous task space. Previously, the former has been addressed using Deep Reinforcement Learning (DRL) [1, 65, 66], while the latter has been formulated as nonlinear trajectory optimization [82, 94, 58]. The brute-force algorithm would sample a condition/task parameter from the task space, run a trajectory optimization algorithm to adapt the input motion clip to meet the condition and run a DRL algorithm to learn a motor skill for the adapted motion. This process should repeat for a dense set of condition/goal parameters so the brute-force algorithm could be prohibitively slow for large task spaces. Our new algorithm achieves considerable performance improvements over the brute-force algorithm by effectively reusing experience tuples and planning task space exploration.

Our learning algorithm includes novel technical components. First, it explicitly constructs motion parameterization as a deep network in the learning process to allow a family of motor skills to be learned concurrently while sharing simulation experiences. The parameterization network also allows trajectory optimization to be incorporated seamlessly into reinforcement learning. Secondly, we adopt continuous-time reinforcement learning to explore temporal variations of the reference motion as well as its spatial variations. Thirdly, we present a new automatic curriculum generation method to explore continuous task space more efficiently. It rapidly explores unvisited regions in the task space, while steadily maintaining already visited regions not to forget previously learned motor skills.

We will demonstrate the flexibility and versatility of our algorithm with highly dynamic motor skills. Our algorithm provides great flexibility in choosing condition/task parameters, which we simply call *task parameters* for brevity, that can be selected from task goals (e.g., target position), body proportions (e.g., height and limb lengths), physical measurements and quantities (e.g., linear/angular velocities, linear/angular momentum, time duration, impact, kinetic energy, mass, inertia, gravity), and environment conditions (e.g., obstacle heights). Our characters are simulated and controlled interactively to perform a range of motor skills in the real-time physics simulation.

3.2 Motion Parameterization

The reference motion $\mathbf{m}(t)$ is represented by a sequence of full-body poses at discrete time instances. Given reference time $\phi \in [0, T]$, the motion $\mathbf{m}(\phi)$ can be considered as a continuous, piecewise linear function that maps time to full-body pose. The poses at discrete time instances interpolate linearly to construct inbetween poses in the continuous time domain. The full-body pose of an articulated figure is represented by a heterogeneous array $(p_0, q_1, \cdots, q_L)$, where $p_0 \in R^3$ and $q_1 \in S^3$ are the position and orientation of the body root, respectively, and the other unit quaternions q_i's for $i > 1$ are joint angles.

The motion is supposed to take place under certain body conditions (e.g., height, weight, and maximum strength at joints) and physical conditions (e.g., gravity and friction coefficients), and may have task goals (e.g., jump height and punch targets). We can construct task space $\mathcal{A}$ by selecting several parameters of interest from body conditions, physical conditions, and task goals. The reference motion corresponds to the origin of the task space, and each parameter axis corresponds to the change in motion according to the parameter change. Each parameter vector $a \in \mathcal{A}$ corresponds to a parametrically-varied motion m_a, which looks similar to the reference motion but satisfies new conditions and tasks represented by a. This mapping constructs a parameterized family of motions $\mathcal{M} = \{\mathbf{m}_a | a \in \mathcal{A}\}$. Hereafter, we denote the reference motion by $\mathbf{m}_0$.

The spatiotemporal variant $\mathbf{m}_a$ of the reference motion is represented by motion displacement mapping and time warping.

$$\mathbf{m}_a(t) = (\mathbf{m_0} \oplus \mathbf{d})(\phi(t)), \tag{3.1}$$

where $\mathbf{d}(\phi) = (d_0(\phi), \cdots, d_L(\phi))$ adds motion displacements to the reference motion to modify its spatial trajectory and joint angles such that

$$\mathbf{m_0} \oplus \mathbf{d} = (p_0 + d_0, q_1 \exp(d_1), \cdots, q_L \exp(d_L)), \tag{3.2}$$

where $d_0 \in R^3$ is the linear displacement of the root position, d_1 is its angular displacement, and d_i for $i > 1$ is the angular displacement of a joint. Please refer to [31] for a detailed description of motion displacement mapping. The time warp function $\phi(t)$ reparameterizes the motion to accelerate, deaccelerate, and change the timing of the task (see Figure 3.1). The time warp should be strictly monotonic everywhere to avoid traveling backward and stalling in time. We construct a monotonically increasing function by accumulating positive time increments $\exp(\tau_k)$ such that

$$\phi_i = \sum_{0 \le k \le i-1} \exp(\tau_k), \tag{3.3}$$

where $\phi_0 = 0$. $\exp(\tau_k)$ is always positive regardless of the value of τ_k. From the discrete samples ϕ_i, a continuous, piecewise linear, monotonic function $\phi(t)$

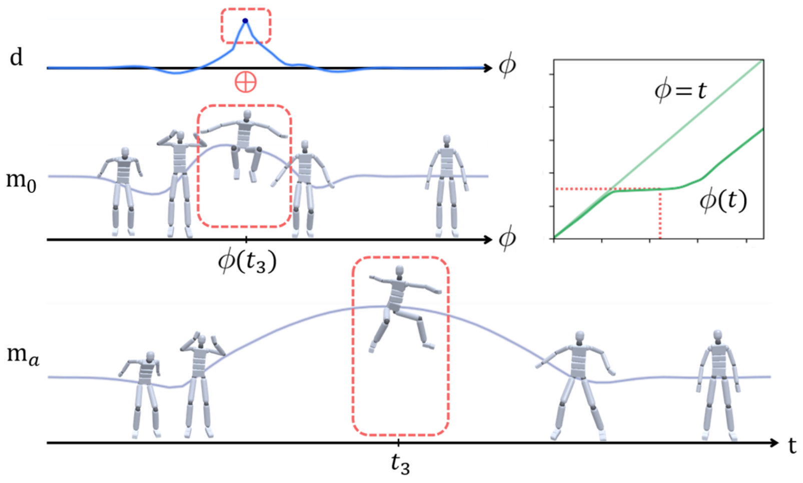

Figure 3.1: Visualization of motion displacement mapping and time warping. The gray curves represent the y position of the pelvis over time in the Jump motion, and the blue curve represents the motion displacement between the two gray curves. The graph in the upper right corner shows the mapping between ϕ and t.

can be constructed by linear interpolation. Each motion $\mathbf{m}_a$ is represented by the trace of motion displacements and time increments over the duration of the motion.

Motion parameterization is constructed explicitly by learning a multilayer perceptron that maps parameter a to motion $\mathbf{m}_a$. We will discuss the learning procedure in the following sections.

3.3 Learning Motor Skills

It has been demonstrated in previous studies that deep reinforcement learning can learn a motor skill that mimics input motion capture data in physics simulation [1]. The motor skill is represented by a deep policy network π that generates actions (e.g., joint torques, muscle activations, or PD targets) at any states (e.g., joint angles, end-effector positions, and/or proprioceptive sensing). Our learning algorithm is also based on imitation learning but fundamentally different from the previous formulation in three aspects. First, our algorithm learns a policy network π_a that represents (infinitely many) motor skills parameterized by $a \in \mathcal{A}$. Second, parameterized motions $\mathbf{m}_a$ to imitate are not known at the beginning of learning. Our algorithm learns both parameterized motions $\mathbf{m}_a$ and their motor skills π_a simultaneously. Lastly, we adopt continuous-time reinforcement learning to allow the timing of parameterized motions to deviate from the reference motion by learning their time warp functions explicitly. In this section, we will discuss how these issues are dealt with.

3.3.1 State and Action

The *state* of the agent includes the positions and velocities of joints/end-effectors /task goals with respect to a local, moving coordinate system, and the height and up-vector of the pelvis (the root of the articulated tree) with respect to a world, reference coordinate system. The state also includes task parameter a, the current reference time ϕ, and a full-body pose (e.i. joint positions and velocities) at the next frame $\phi + \Delta\phi$. As discussed by Park et al. [65], the rich state description including its future prediction supplements a scalar value ϕ to disambiguate phases in long motion data. In our formulation, the future frame from the reference motion serves as a future prediction.

The *action* of the agent is defined by spatial displacement $\mathbf{d}$ and time increment τ.

$$(\mathbf{d}_t, \tau_t) = \pi_a(s_t). \tag{3.4}$$

This action makes transition to its subsequent state by advancing reference and simulation time $\phi' = \phi + \exp(\tau_t)$ and $t' = t + \Delta t$, where Δt is constant. The state update by spatial displacement uses PD control in physics simulation, where the displaced pose $\mathbf{m}_0(\phi') \oplus \mathbf{d}_t$ serves as a PD target to generate joint torques. Advancing the dynamics simulation by time Δt updates the state at the next time instance.

3.3.2 Reward

The goal of reinforcement learning is to find the optimal policy that maximizes the expected cumulative reward. If the agent follows the optimal policy π_a^* in physics simulation, it will act out the reference motion approximately while satisfying the conditions and tasks denoted by a. The trace of spatial displacements and time increments forms the optimal motion $\mathbf{m}_a^*$ for parameter a.

The *reward* includes two terms for motion tracking and conditions/tasks enforcement.

$$r = w_{\text{tracking}} r_{\text{tracking}} + w_{\text{task}} r_{\text{task}}, \tag{3.5}$$

The agent receives tracking rewards r_{tracking} if it imitates the reference motion $\mathbf{m}_a$ accurately, while it receives task rewards r_{task} if it achieves the conditions/tasks represented by parameter a.

The tracking reward includes four sub-terms

$$r_{\text{tracking}} = w_q r_q + w_{\text{ee}} r_{\text{ee}} + w_{\text{com}} r_{\text{com}} + w_{\text{time}} r_{\text{time}}, \tag{3.6}$$

which respectively penalize the discrepancies in joint angles, end-effector positions, COM (center of mass), and time between the reference motion and simulation.

$$\begin{aligned}
r_q &= \exp\Big(-\frac{1}{\sigma_q^2} \sum_{i \in \text{joints}} \parallel \log(q_i^{-1}\hat{q}_i) \parallel^2 \Big) \\
r_{\text{ee}} &= \exp\Big(-\frac{1}{\sigma_{\text{ee}}^2} \sum_{j \in \text{ee}} \parallel \hat{x}_j - x_j \parallel^2 \Big) \\
r_{\text{com}} &= \exp\Big(-\frac{1}{\sigma_{\text{com}}^2} \parallel \hat{x}_{\text{com}} - x_{\text{com}} \parallel^2 \Big) \\
r_{\text{time}} &= \exp\Big(-\frac{1}{\sigma_{\text{time}}^2} \parallel (\exp(\hat{\tau}_t) - \exp(\tau_t) \parallel^2 \Big),
\end{aligned} \tag{3.7}$$

where $q_j \in S^3$ are joint angles, $x_j \in R^3$ are joint positions, $x_{\text{com}} \in R^3$ is the center of mass of the full-body. The hat symbol indicates measurements from $\mathbf{m}_a$. In our experiments, weights are common for all examples: $\sigma_q = 3.16, \sigma_{\text{ee}} = 0.8, \sigma_{\text{com}} = 0.35, \sigma_{\text{time}} = 0.1, w_q = 0.28, w_{\text{ee}} = 0.28, w_{\text{com}} = 0.28$, and $w_{\text{time}} = 0.16$.

The task reward is defined by a similar form.

$$r_{\text{task}} = \exp\Big(-\frac{1}{\sigma_{\text{task}}^2} \parallel \hat{x}_{\text{task}} - x_{\text{task}} \parallel^2 \Big), \tag{3.8}$$

where x_{task} can be any measurement taken from the simulation including position, orientation, linear/angular velocity, linear/angular momentum, time duration, impact, pressure, and kinetic energy for a certain duration. We can have great flexibility in choosing parameters and defining their corresponding rewards. Any parameter with a zero range serves as a constraint. We use constraints to specify invariant factors that a family of parameterized motions should commonly satisfy.

There are two types of rewards: *continuous* and *spike.* The *continuous* rewards are received uninterruptedly over a certain period of time, while the *spike* reward is received instantly when a certain condition is satisfied. All tracking rewards are continuous. The task reward can be either of the two types. A reward defined over a duration (e.g., total joint torque or total kinetic energy during a time interval) is *spike* because the reward is received instantly when the simulation reaches the end of the duration. In our experiments, $w_{\text{tracking}} = 1.0$ and $w_{\text{task}} = 10.0$ for spike rewards, and $w_{\text{tracking}} = 0.9$ and $w_{\text{task}} = 0.1$ for continuous rewards. $w_{\text{tracking}} = 1.0$, if the task is not defined.

Physical quantities (e.g., mass, inertia, gravity) and body proportions (e.g., limb length) can also be chosen to form motion parameterization. In this case, it is not necessary to define corresponding rewards, but the changes in quantities and proportions are reflected in physics simulation and state transitions.

3.3.3 Variable Time-Step Policy Update

The learning algorithm is episodic. It generates many episodes of physics simulation tracking parameterized motions and collects experience tuples to update the value and policy networks. The Proximal Policy Optimization (PPO) is a popular policy-gradient method that addresses discrete-time Markov decision problems with uniform time steps. We derive its variant to address variable time-step problems. The length of time steps critically affects how rewards are discounted over time and how generalized advantages are estimated. Given a time step $\Delta\phi = \exp(\tau_t)$, the discount factor during $[\phi, \phi + \Delta\phi]$ is $\gamma^{\Delta\phi}$. Similarly, generalized advantage estimation (GAE) parameter during the time interval is $\lambda^{\Delta\phi}$. Provided that both continuous rewards r_c and spike rewards r_s are given, GAE with variable time stepping is

$$\begin{aligned} \delta(\phi) &= \int_0^{\Delta\phi} \gamma^v R_c(\phi + v)dv + R_s(\phi) + \gamma^{\Delta\phi} V(\phi + \Delta\phi) - V(\phi) \\ A(\phi) &= \delta(\phi) + \gamma^{\Delta\phi}\lambda^{\Delta\phi} A(\phi + \Delta\phi). \end{aligned} \tag{3.9}$$

Here, the continuous rewards are integrated and discounted over the time interval, while the spike rewards are estimated instantly. This modified PPO algorithm makes policy learning invariant under time warping and therefore

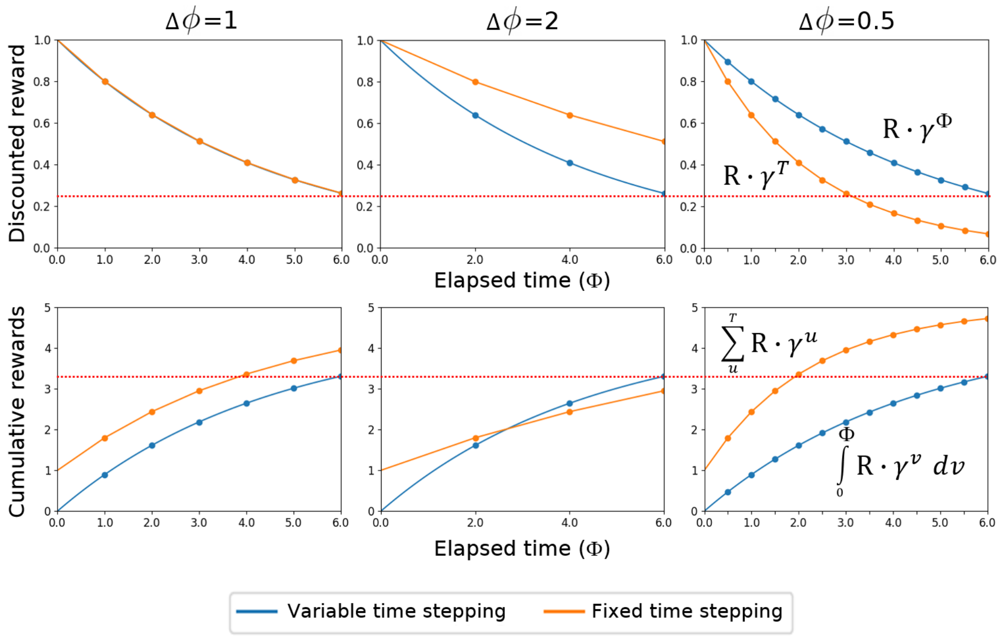

Figure 3.2: The elapsed time and the reward graphs. T is elapsed simulation time, and Φ is elapsed reference time. The reward is received once in the first row and the reward is added every time step in the second row. The red lines are the baseline discounted reward and cumulative rewards at $\Phi = 6$ when time step $\Delta\phi = 1$, respectively. With our variable time step policy update, the discounted reward and cumulative rewards are invariant under reference time step. In this figure, we set $R = 1$ and $\gamma = 0.8$.

prevents $\mathbf{m}_a$ from unintentionally speeding up or slowing down regardless of their conditions/tasks (see Figure 3.2).

3.4 Motion Optimization

Finding any motion $\mathbf{m}_a$ that satisfies the conditions/tasks given by a while minimizing the deviation from the reference motion $\mathbf{m}_0$ can be formulated as trajectory optimization. The brute-force construction of motion parameterization requires solving this trajectory optimization for a continuous domain $a \in \mathcal{A}$. In this work, we do not optimize individual motions one-by-one, but learn the parameter-to-motion mapping gradually in a supervised manner. Since we reuse simulation episodes generated during reinforcement learning, motion parameterization can be constructed at a fraction of the computation cost of reinforcement learning.

3.4.1 Fitness

Each simulation episode generated by policy π_a forms the trace of motion displacements and time increments $\{(\mathbf{d}_t, \tau_t) | t = 0, \cdots, T\}$, which can be converted into a motion using Equation (3.1). We align the root trajectory of the episode with that of the $\mathbf{m}_0$ before converting. Though the goal a is pursued in the simulation, many episodes do not achieve this intended goal if the policy is sub-optimal. Since we are dealing with a continuous target domain, we use the insight of hindsight experience replay to reuse simulation rollouts [95]. The sub-optimal episode is useful if we retrospectively imagine that the agent was trying to achieve the goal $a' \in \mathcal{A}$ they actually ended up with. This episode can be used to learn the correlation between a' and $\mathbf{m}_{a'}$ if the episode represents a

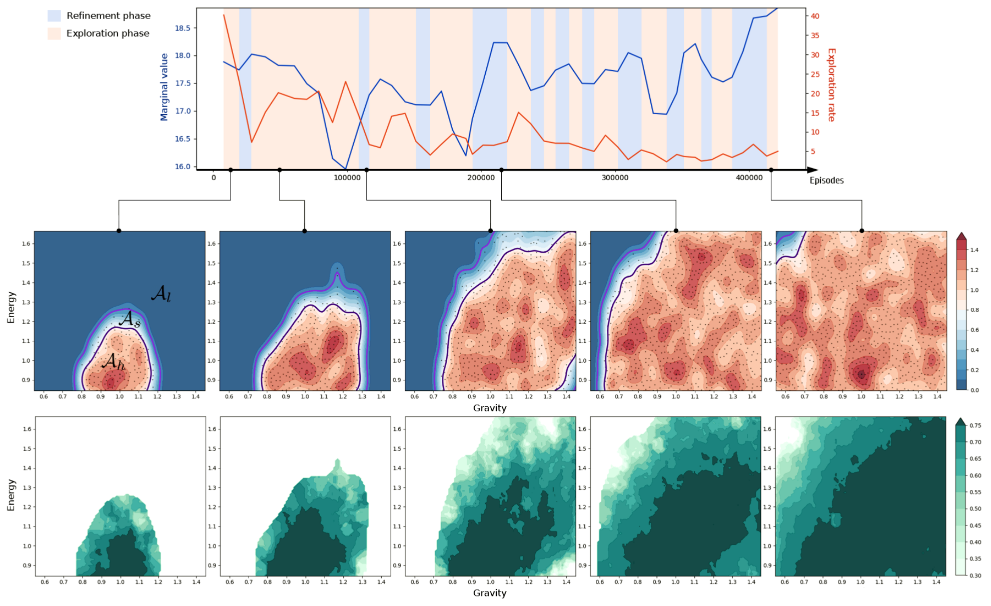

Figure 3.3: Curriculum of learning jumps parameterized by gravity and kinematic energy. (Top) The blue and red plots, respectively, represent the change in mean marginal value and exploration rate as learning progresses. The peach and blue shades, respectively, represent the exploration and refinement phases. (Middle) The elite episodes are shown as black dots and their density is shown as red-to-blue shades. (Bottom) The fitness values of the elite episodes.

quality motion. The motion quality is measured using a fitness function

$$F = f_{\text{pose}} \cdot f_{\text{velocity}} \cdot f_{\text{contact}} \cdot f_{\text{drag}} + w_{\text{constraint}} \cdot r_{\text{constraint}}, \tag{3.10}$$

which compares the poses, velocities, and contact states of the episode to the reference motion $\mathbf{m}_0$.

$$f_{\text{pose}} = \exp\Big(- \frac{1}{\sigma^2_{\text{pose}}} \frac{1}{T} \sum_{\phi=0}^{T} \sum_{i \in \text{joints}} \parallel \log(q_i(\phi)^{-1} \bar{q}_i(\phi)) \parallel^2 \Big)$$

$$f_{\text{velocity}} = \exp\Big(- \frac{1}{\sigma^2_{\text{velocity}}} \frac{1}{T} \sum_{\phi=0}^{T} \sum_{i \in \text{joint}} \frac{\parallel \bar{v}_i(\phi) - v_i(\phi) \parallel^2}{\max(V, \parallel \bar{v}_i(\phi) \parallel^2)} \Big)$$

$$f_{\text{contact}} = \exp\Big(- \frac{1}{\sigma^2_{\text{contact}}} \frac{1}{T} \sum_{\phi=0}^{T} \sum_{j \in ee} \bar{C}_j(\phi) \parallel \bar{y}_j(\phi) - y_j(\phi) \parallel^2 \Big)$$

$$f_{\text{drag}} = \exp\Big(- \frac{1}{\sigma^2_{\text{drag}}} \frac{1}{T} \sum_{\phi=0}^{T} \sum_{j \in \text{ee}} \bar{C}_j(\phi) \bar{C}_j(\phi - 1) \parallel T_j(\phi) \parallel^2 \Big),$$

where q and v are joint positions and velocities with respect to the body local coordinate system. y is the height of the end-effector from the ground surface. $C_j(\phi) \in \{0, 1\}$ is a Boolean flag that indicates the contact state of the j-th joint at time ϕ. $T_j(\phi)$ is the translation of the end-effector in the xz plane. The bar over a symbol indicates that the value is measured from the reference motion $\mathbf{m}_0$. V is a constant that prevents vanishing denominators. The fitness function takes into account constraints (zero-range parameters) to evaluate if the episode satisfies the desired qualities specified by the user. In our experiments, the weights are $\sigma_{\text{pose}} = 2.67, \sigma_{\text{velocity}} = 75.5, \sigma_{\text{contact}} = 1.3, \sigma_{\text{drag}} = 0.7, w_{\text{constraint}} = 0.1$ and $V = 0.5$.

3.4.2 Parameterization Network

The policy learning algorithm in the previous section generates a large number of good episodes with its fitness value above a user-specified threshold. These episodes serve as training data for learning a parameterization MLP P_θ that takes the parameter a as input and outputs $\mathbf{m}_a$ such that

$$P_\theta(a) = \{(\mathbf{d}_t, \tau_t) | t = 0, \cdots, T\}. \tag{3.11}$$

The MLP is a simple regression network that learns the nonlinear mapping between parameters and motions minimizing the loss

$$\theta = \underset{\theta}{\operatorname{argmin}} \sum_t \|\mathbf{d}_t - \mathbf{d}_t^*\|^2 + (\tau_t - \tau_t^*)^2, \tag{3.12}$$

which measures the discrepancy between training data and network outputs. The MLP is learned in parallel with policy learning.

Since the performance of the MLP largely depends on the quality of its training data, feeding the MLP learning with consistently high fitness episodes is necessary. To do so, we maintain a list of elite episodes E. Since policy learning generates simulation episodes in an arbitrary order, it is important to ensure that new episodes do not override previous ones with higher fitness values. Whenever a new, above-threshold episode with parameter a and fitness F is generated, we compare it with its neighbors within radius r in the elite set. Let a' and F' be the parameter and fitness of a neighbor, respectively. The weighted distance between a and a' is within r

$$(a - a')^\top A(a - a') < r^2 \tag{3.13}$$

in the neighborhood, where A is a diagonal weight matrix. The radius r controls the granularity of samples in the elite set. In our experiments, we set $A = I$ and $r = 0.1$. If the new episode is better than any of its neighbors ($F > F'$), the new episode will be used to update the parameterization network and be included in the elite set to replace the inferior neighbors. Otherwise, the new episode is dismissed. In this way, every episode in the elite set is the best in its neighborhood, and the parameterization network is progressively updated with a series of monotonically-improving training data. We experimentally found that the quality of the converged motion is not sensitive to the threshold of fitness. In our experiments, we set the value 0.1 to exclude completely failed episodes. The elite set is stored in a k-d tree so that neighborhood searches can be performed efficiently.

3.4.3 Bootstrapping

In the early stages of learning, the premature parameterization network may generate low-quality motions, which negatively affect subsequent motor skill learning. To bootstrap network learning, two strategies are adopted. First, we start by training a tracking controller (a.k.a. policy π_0) that mimics the original reference m_0 until the learning curve plateaus. This base policy provides a good starting point to explore similar motor skills in motion parameterization. Second, we use a more reliable proxy to replace the network outputs $P_\theta(a)$ until the regression loss decreases below the threshold. The proxy is computed by locally linear regression of elite episodes. To do so, we take k-nearest neighbors of a in the elite set E and linearly interpolate them with weights

$$w_k = F_k \cdot \exp\big(-\frac{1}{\sigma_k^2}(a_k - a)^T A (a_k - a)\big). \tag{3.14}$$

The weights take into account both distance in the task space and fitness values. The k-nearest neighbor interpolation always generates decent quality motions because the neighbors to be interpolated are chosen from the elite set. We set $k = 5$, $\sigma_k = 0.28$ in our experiments.

Once the regression loss goes below the threshold, the system chooses outputs randomly between direct network outputs and locally-linear estimates. In this way, motor skill learning is consistently provided with high-quality motions to imitate while progressively learning motion parameterization.

3.5 Curriculum Generation

The order in which the learning algorithm explores the task space is important because motor skills are learned over a continuous domain. Uniform random sampling often fails when the RL algorithm tries to learn motor skills far away from the reference motion. A popular approach to overcome this problem is to use curriculum learning [96]. The policy for harder (far from the reference motion) motor skills is discoverable after mastering easier (near the reference motion) motor skills. Curriculum learning suggests that the learning algorithm needs to explore the task space in a near-to-far order, mastering easy ones and proceeding to harder ones. This simple curriculum needs elaboration to address two issues. First, mastering a (even easy) motor skill is computationally demanding. Mastering one by one to cover the entire task space is impractical, if not impossible. Second, even if motor skills are learned up to a certain threshold in the early stages of learning, policies may forget the skills while learning the others. Perhaps, mastering individual motor skills early on is not necessary. More importantly, easier ones should be mixed in while learning harder ones to avoid forgetting.

In this section, we present a new automatic curriculum generation method that allows the learning algorithm to rapidly explore unvisited regions in the task space while continuously refining already visited regions not to forget (see Figure 3.3). The method alternates between exploration phases and refinement phases. A batch of action parameters are picked from less visited regions in exploration phases, whereas they are picked from heavily visited regions in refinement phases. In our experiments, each batch includes 20 task parameters.

The motor skill learner produces 30 simulation episodes for each parameter and updates the value/policy networks based on the episodes. We estimate the density at parameter $a \in \mathcal{A}$ by kernel density estimation.

$$\text{density}(a) \propto \sum_{a_k \in \text{neighbor}(a)} \exp\big(-(a_k - a)^T A (a_k - a)\big), \tag{3.15}$$

If the density is above a user-specified threshold, parameter a is in the heavily visited region $\mathcal{A}_h \subset \mathcal{A}$. Otherwise, it is in the less visited region $\mathcal{A}_l = \mathcal{A} \setminus \mathcal{A}_h$.

3.5.1 Marginal Value and Exploration Rate

Catching up with how well each task parameter is achieved and deciding when to make transitions between the phases are the keys to robust and efficient learning. Ideally, the method should be independent of the choice of reference motions and task parameters and should not require careful parameter tuning. We use two measures, *marginal value* and *exploration rate.* The state of the agent is high-dimensional, and only a few of the dimensions are selected to form task space. Let state $s = (s_a, s_b)$ be a composition of task parameters s_a and the rest of the parameters s_b. Marginalizing s_b, the value function is reduced to

$$\bar{V}(s_a) = \int_{s_b} V(s_a, s_b)\, ds_b. \tag{3.16}$$

Here, we assume uniform density of s_b. The marginal value function $\bar{V}(s_a)$ estimates how much rewards will be received if the agent follows the current policy pursuing task a. The *mean marginal value* over $\mathcal{A}$ is $\bar{V}(\mathcal{A}) = E_{a \in \mathcal{A}}(\bar{V}(s_a))$.

The *exploration rate* estimates how successful the rollout of simulation episodes was in discovering new motions and improving motor skills. We define the exploration rate by a number of new episodes added to the elite set for a single batch. To prevent the exploration rate from increasing meaninglessly, the exploration rate increases when the fitness of new episode is clearly higher than those of dropped ones. We set the threshold 0.02.

3.5.2 Exploration and Refinement

The goal of exploration phases is the expansion of $\mathcal{A}_h$ by visiting its surrounding areas. To explore the task space gradually in a near-to-far order, a batch of task parameters B are sampled in the surrounding area $\mathcal{A}_s$ where the density is between D' and D for $D' < D$. $\mathcal{A}_s$ is part of $\mathcal{A}_l$ and adjacent to $\mathcal{A}_h$. In exploration phases, the exploration rate is proportional to the mean marginal value $\bar{V}(B)$, because the control policy is likely to improve if many good episodes are discovered. Similarly, the exploration rate and the average fitness value tend to be proportional to each other. D and D' decide how aggressively or conservatively we want the explore. In our experiments, $D = 0.8$ and $D' = 0.2$.

The goal of refinement phases is to improve the policy gradually and evenly across $\mathcal{A}_h$ while avoiding forgetting. We use an MCMC (Markov Chain Monte Carlo) method with multiple starting points [68] to sample a batch of task parameters in $\mathcal{A}_h$. The MCMC algorithm with a probability distribution

$$p(s_a) = \exp(-k\frac{\bar{V}_(s_a) - \mu}{\mu}) \tag{3.17}$$

samples more where the marginal value is low and therefore continuously pursues a state where marginal values are evenly distributed across $\mathcal{A}_h$. In our experiments, the constant k is 10.

3.5.3 Transition Criteria

Our learning algorithm monitors the change of exploration rates in $\mathcal{A}_h$ and its surrounding area $\mathcal{A}_s$ by occasionally taking samples. The exploration rate tends to decrease in exploration phases. The transitioning to a refinement phase occurs when the exploration rate in $\mathcal{A}_s$ becomes smaller than the exploration rate in $\mathcal{A}_h$. In refinement phases, the samples drawn from $\mathcal{A}_h$ influence not only $\mathcal{A}_h$ but also its surrounding areas $\mathcal{A}_s$. Therefore, the mean marginal value and the exploration rate in $\mathcal{A}_s$ improves gradually during refinement. The transitioning to an exploration phase occurs when the exploration rate in $\mathcal{A}_s$ becomes larger than the exploration rate in $\mathcal{A}_h$.

3.6 Experiments

Our simulation system is written in C++ and based on DART dynamics toolkit [97]. Reinforcement learning and regression networks are written in Python and based on Tensorflow library [98]. The base model of our animated characters is 1.65 m tall and weighs 61.6 kg consisting of rigid bones connected by 21 ball-and-socket joints. The articulated skeleton is actuated by joint torques, which are computed by linear-time stable PD controllers [99]. The PD gains are $k_p = 600$ and $k_d = 49$. The simulation time step is 150 Hz, and the control time step is 30 Hz.

The policy network has four fully-connected layers of 1024 ReLU nodes. The value network and the parameterization network have two fully-connected layers of 512 ReLU nodes. The clipping range of PPO is 0.2, the learning rate of the policy function is 0.0002, the learning rate of value/parameterization networks is 0.001, the discount factor is $\gamma = 0.95$, and the GAE parameter is $\lambda = 0.95$. The minibatch size of the policy and value networks is 1024. The minibatch size of the marginal value and parameterization network is 128. The parameterization network is updated 10 times every ten policy updates. The MCMC algorithm draws 1000 samples for every five policy updates.

We used motion data available on public motion databases including Mix-

Table 3.1: Hyperparameters and performance statistics.

skills	task parameters	range	constraints	N	ER
Backflip	angular velocity	[0.5, 2.0]	angular momentum rotation	2.1M	0.9
	height	[0.8, 1.3]			
BoxJump	distance	[1.0, 2.15]	-	2.2M	0.91
	height	[-0.9, 0.9]			
Cartwheel	length of arms	[0.65, 1.65]	rotation	3.5M	1
	length of legs	[0.65, 1.65]			
	width of arms	[1.0, 2.0]			
	width of legs	[1.0, 1.6]			
Dodge	angle	[-1.5, 1.5]	-	0.2M	1
Jump	energy	[0.8, 1.8]	foot contact	1.65M	0.92
	gravity	[0.2, 1.65]			
ObstacleJump	height of obstacle	[0.9, 3.0]	-	0.5M	1
Pivot	rotation	[0.8, 2.5]	-	0.12M	1
Punch	angle	[0.0, 1.0]	-	4.2M	0.72
	distance	[0.8, 1.4]			
	force	[1.0, 5.5]			
	height	[0.9, 1.1]			
Push	mass of object	[0.5, 10]	distance	0.7M	1
SideKick	angle	[0.0, 0.8]	-	1.52M	0.81
	force	[1.0, 6.0]			
	kick height	[0.9, 1.1]			
SpinKick	kick height	[0.5, 1.5]	-	0.93M	1
	rotation	[0.5, 2.0]			
Swing	height of obstacle	[1.0, 1.55]	-	1.5M	1
WallJump	height of obstacle	[0.9, 2.0]	-	1M	1

amo [100] and CMU motion databases [101]. The computation time for learning varies depending on the dimension and range of task spaces. It takes 12 hours to 72 hours on a single PC with AMD Ryzen 9 3950x (3.5 GHz, 16 cores). Since sequential computation for physic simulation is hard to benefit from the parallelization power of GPU, most of the computation is done in the CPU.

3.6.1 Motor Skills and Their Parameterization

We conducted experiments with various motor skills. The task parameters, hyperparameters, and performance statistics are summarized in Table 3.1. The total number N of episodes the algorithm generated measures computational efficiency. ER (exploration ratio), which is the percentage of the heavily visited regions in the task space, measures how successful the algorithms are in exploring the task space. The unit of each task parameter is a multiple of that in the reference data. Exceptionally, the unit of angle parameter is radian, and the unit of height parameter in BoxJump is meter. Some reward terms and fitness terms look similar and are sometimes interchangeable. The rule of thumb is that the reward function is designed to learn motor skills. The reward function includes essential terms that are necessary for all examples. In contrast, the fitness function is designed to improve the visual quality of simulation results and can be customized for individual examples.

The Jump example takes two task parameters: gravity and kinetic energy (see Figure 3.3). The X-axis of the task space represents the gravity ranging from $0.5G$ to $1.65G$. The Y-axis represents the kinetic energy ranging from $0.9E$ to $1.7E$. Here, G is the gravity on Earth, and E is the full-body kinetic energy at the onset of jump in the reference motion. Higher kinetic energy and lower gravity result in higher jumps.

The SpinKick example has two task parameters: kick height and angular momentum about the vertical axis. The kick height is 1.5 meters, and the spinning angle is 6.0 radians in the reference motion. The task space spans the range of $[0.75, 2.25]$ meters and $[3.0, 12.0]$ radians. The character spins in the air in the reference motion.

The Punch skills parameterized by four parameters (distance, height, angle, impact) allows the character to throw a punch in the desired position with the desired power. We measure the impact between the fist and the target object by actually placing the object at the strike point predicted from the current trajectory in the physics simulation. The after-impact momentum of the object estimates the punch impact.

The Backflip skills are parameterized by jump height and angular velocity. The angular velocity parameter is calculated as the average angular velocity of the torso during the body is in the air. Since the whole-body angular momentum is preserved while the body is in the air, the angular velocity and the moment of inertia of the whole-body are inversely proportional to each other. Using our learning algorithm, the simulated character learned to squeeze its body to reduce its moment of inertia and consequently increase the angular velocity. Motion parameterization maps angular velocity parameters to the degree of whole-body extension.

The Cartwheel example demonstrates task parameterization by body conditions such as limb lengths and weights. Specifically, we selected four parameters, the length and radius of the arms and the legs. The weight is proportional to the length and the square of the radius. The total rotation angle for the duration of Cartwheel is constrained to prevent over-rotation and under-rotation. The angular velocity of the learned cartwheel varies nonlinearly depending on the mass distribution on limbs.

In the Push motion data, the simulated character pushes forward a box

Table 3.2: Comparison of ablated algorithms.

skills	parameters	A1	A2	A3
spinkick	kick height	[0.5, 1.5]	[0.5, 1.5]	[0.5, 1.5]
	rotation	[0.725, 1.3]	[0.5, 2.0]	[0.5, 2.0]
jump	gravity	[0.4, 1.65]	[0.3, 1.65]	[0.2, 1.65]

weighing 18kg. The Push skill is parameterized by the mass of the box ranging from 9kg to 180kg. We force the hands to stick to the box when they are close enough to the box within a certain phase. The total moving distance is fixed as a constraint regardless of the box weight. As the mass increases, the simulated character moves slowly and leans more on the box.

3.6.2 Ablations

In this section, we conduct ablation studies to demonstrate the effectiveness of our approach. Our algorithm is reduced to the DeepMimic algorithm [1] if motion parameterization is removed. So, the DeepMimic algorithm is considered as the baseline of comparison (A1). The second algorithm allows spatial variations in motion parameterization while the task timing is fixed (A2). Our algorithm allows both temporal and spatial variations in motion parameterization (A3).

We compared the three algorithms for Jump and SpinKick examples. To simplify the experiments, the kinetic energy parameter is fixed in the comparison. All parameters and reward weights were tuned for the best performance of the A1 algorithm. $\Delta\phi$ was set to a constant for the A2 algorithm to disable time warping. We adopted a simple curriculum generation method that explores the task space in a near-to-far order while alternating exploration and refinement phases of fixed duration. We measured the ratio of successfully explored regions

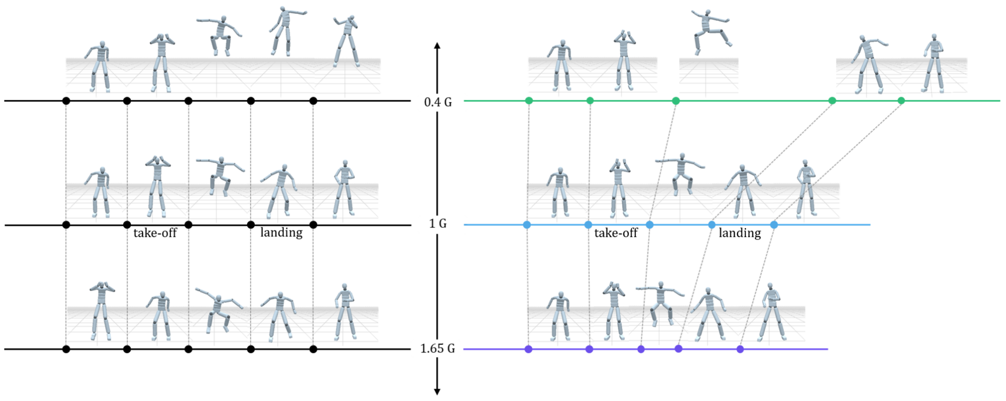

Figure 3.4: The snapshots on timeline of jumping in different gravity settings with (left) A2 and (right) A3 algorithms. The dotted lines align the same phases in the jump motion. The time warping graph for each timeline is represented by a line of the same color in Figure 3.5.

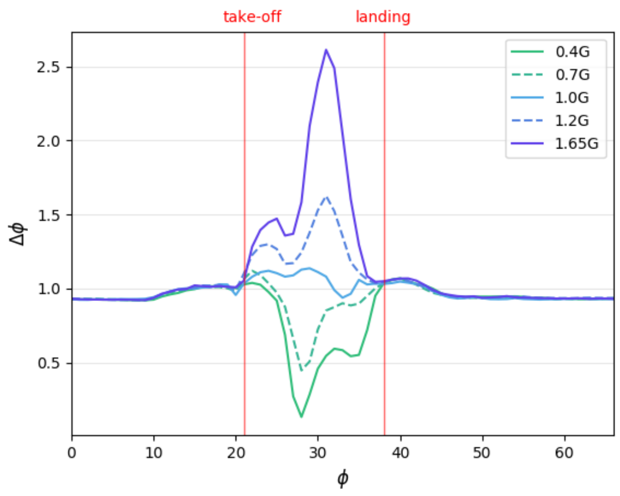

Figure 3.5: The reference time and delta graphs for jumps in different gravity settings.

in the task space. The learning algorithm terminates when the exploration rate goes down to zero and the mean marginal value plateaus.

As expected, our algorithm (A3) successfully explored wider ranges in task spaces than the other algorithms (see Table 3.2). For the SpinKick example, it is relatively easy to change kick height, thus even the baseline algorithm explored the full range of kick heights. However, The baseline algorithm is not suitable to deal with large changes in full-body dynamics such as adjusting spinning angle and jump height. The comparison shows that our algorithm better deals with large changes in full-body dynamics. Our algorithm manifests exaggeration of anticipation and follow-through effects before and after the jump. The simulation results are best viewed in the accompanied video.

Figure 3.4 demonstrates the effectiveness of timewarping. The character is supposed to jump higher in the low gravity setting (0.4G). Although both A2 (no timewarp) and A3 (ours) algorithms generate higher jumps as expected, the result of A3 looks much better than the result of A2 because the A2 simulation does not align with the reference motion. In the figure (top, left), the pose in the middle of jumping matches the after-landing pose in the reference motion. This misalignment is noticeable in the simulation. Figure 3.5 shows the timelines for the range of gravity settings. Our algorithm learned to speed-up or slow-down appropriately to deal with gravity changes.

3.6.3 Comparison of Curriculum Methods

In this section, we compare our curriculum generation method to other methods to demonstrate its effectiveness. The baseline algorithm draws task parameters randomly from the task space without exploration and refinement phases. The second algorithm explores the entire task space first (ER=1) and moves on to the refinement phase. The third algorithm masters motor skills at heavily visited regions and proceeds to the next exploration phase. The mastery of motor skills is determined based on the threshold of mean marginal values. We also set a time limit at each phase so that the algorithm can move on. The mastering algorithm could be extremely slow without timeout. The fourth algorithm exploits the alternation between exploration and refinement phases and draws a constant amount of samples (20 value/policy updates) at each phase. Our algorithm uses the transition rules explained in Section 3.5.

We conducted experiments with four examples: SpinKick, Dodge, Jump and Push (see Figure 3.6). The SpinKick and Dodge are easier than Jump and Push examples. The ranges of task space are the same as Table 3.1 except for Jump. We fixed the kinetic energy parameter to simplify the experiment. The baseline algorithm expands quickly in the beginning for the easy examples but slows down as the probability of exploring heavily visited regions increases. The exploration-only algorithm performs well for the easy examples because intermittent refinement is not essential for robust exploration. Even though the exploration-only algorithm reaches ER=1 first in Spinkick example, it is not the fastest in the group because it has to spend some time in the refinement phase. The aggressive algorithms (baseline and exploration-only) often fail to explore the task space for the harder examples. For example, the baseline algorithm explored only 43% of the task space for Jump on average, and the exploration-only algorithm explored 87% for Jump and 80% for Push. On the other hand, the conservative algorithms (alternating and mastering) perform well for the challenging examples without being stuck in premature convergence. The refinement phases play an important role in incrementally refining motor skills and making the exploration phases gradually expand from the well-refined heavily visited regions. Our algorithm performs well regardless of the difficulty level and the type of examples because it seeks a good balance between performance

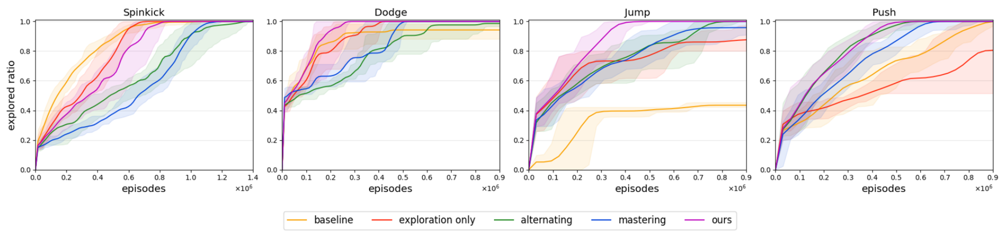

Figure 3.6: The explored ratio in the task space for four different motor skills. The colored line is the average value of multiple attempts. The lower border remaining the same over time means one or more attempts got stuck in premature convergence.

and quality while being consistently successful in exploration.

3.6.4 Joint Torque Limits

In our experiments, setting reasonable joint torque limits is essential to construct believable motion parameterization (see Table 3.3). The character simulated without torque limits use superhuman forces to achieve goals or tasks in unrealistic ways. The superhuman character does not need to make bigger moves to generate more power. Motion parameterization can learn interesting, dynamically-realistic variants of the reference motion when the character has human physical abilities.

We conducted experiments to demonstrate the effects of joint limits on two motor skills: Punch and Push. We set joint torque limits heuristically. When the torque computed by a PD servo is beyond its limit, the computed torque is clipped by the limit. Figure 3.7 shows the simulation of the reference motion at the top row. The superhuman character in the second row can punch harder and push the heavier box without changing its moves. The motion of the superhuman character is almost identical to the reference motion. Our character

Table 3.3: The range of joint torque.

Joint	Torque range $(N \cdot m)$
Hips	[0, 0]
Spine	[-300, 300]
Spine1, Spine2	[-150, 150]
Neck, Head	[-75, 75]
Shoulders, Arms	[-150, 150]
ForeArms, Hands	[-90, 90]
UpLegs	[-300, 300]
Legs	[-225, 225]
Feet	[-145, 145]
Toes	[-90, 90]

with limited power learned that it could make a stronger punch impact when it takes a wider stance and a longer swing path. It also learned to lean more to push heavier boxes.

3.7 Discussion

We presented a new learning-based framework to construct a rich variety of parameterized motor skills learned from motion capture data. The key to the success of our approach is learning motor policies and motion parameterization concurrently. The concurrent learning approach achieves both computational efficiency and coherent visual quality over a wide range of the parameter domain. The databases of parameterized human motions are readily available to the public [100]. We envision that the databases of parameterized motor skills for physics simulation will be available soon to the public as well.

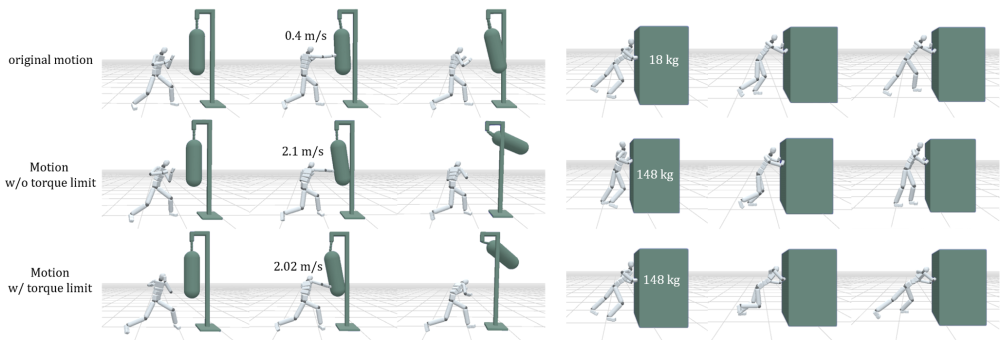

Figure 3.7: The snaps of motor skills with/without torque limits. The first row is the simulation results tracking original reference, the second row is the simulation results without joint limit in harder/heavier settings, and the third row is the simulation results with joint limit in harder/heavier settings. The maximum velocity of the object after a stroke and the mass of the object are written on the figure.

Our algorithm is particularly effective when the reference motion is highly dynamic and energetic because dynamics effects are easily visualized, manipulated, and amplified in the motion parameterization. An interesting observation is that the effectiveness of exploration along the energy/dynamics axis is not symmetric. Exploration in one direction may be rapid, but not in the opposite direction. The algorithm easily deals with motor skills in heavier, faster, and stronger settings than the reference motion, while it sometimes struggles in lighter, slower, and weaker settings. The rationale is not clear yet. Perhaps, infusing more energy happens naturally as required by parameter settings, but reducing the level of energy requires more than parameter settings such as energy minimization/regularization.

There are many exciting directions to explore in future work. Currently, parameterized motor skills are learned for individual motion clips. It requires extra efforts to allow transitioning between motor skills that are learned independently from each other. Ideally, we wish to learn multiple, integrated motor skills simultaneously from unorganized motion datasets [65].

Although our system runs successfully with various task types (ranging from static to highly energetic) and various parameter types (including body/physics /environment conditions), the current system implementation does not scale well with the dimension of the task space. We have tested up to four dimensions so far. Dealing with massively-parameterized motor skills will be an interesting challenge. The complexity of the real world problem requires many parameters for accurate modeling. Musculoskeletal gait is a good example [70]. There are many anatomical elements in the human musculoskeletal system (such as bones and muscles), and the condition of each element (such as bone length and muscle strength) affects gait. In this case, the dimension of the task space is proportional to the number of bones and muscles, which is close to 1000. Scalable algorithms would open up new possibilities in many practical applications.

Chapter 4

Learning Virtual Chimeras by Dynamic Motion Reassembly

4.1 Overview

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.

4.2 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

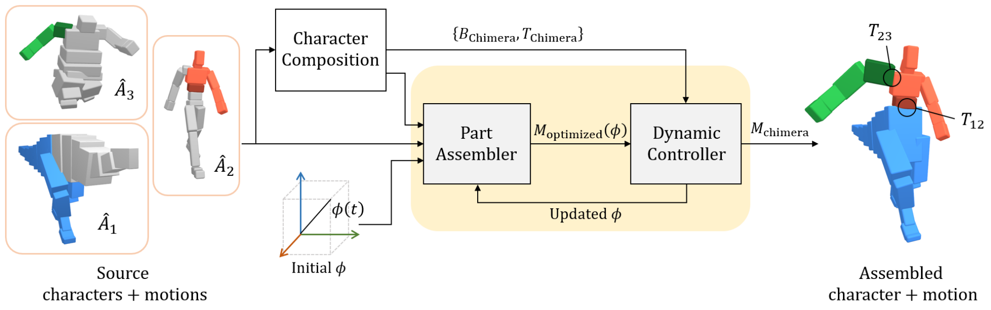

Figure 4.1: System overview

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 part animations. In Figure 4.1, 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 4.2). Our algorithm learns the policy conditioned by multi-dimensional phase alignments $\Phi = (\phi_i) \in R^N$, 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.

4.3 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.

$$\mathbf{M}_{\text{optimized}}(\Phi) = \operatorname{argmin} E_{\text{contact}} + E_{\text{global}} + E_{\text{reg}}. \tag{4.1}$$

We use a derivative-free optimization method, CMA-ES, to solve this optimization problem [102].

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

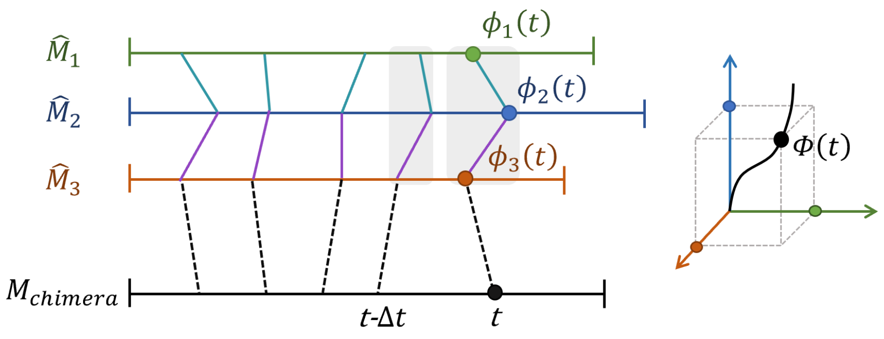

Figure 4.2: Temporal alignments of the motions. In the timeline on the left, the sky blue and purple lines temporally map the frames in each timeline of the source motion. The connected colored lines are mapped to the time of M_{chimera} by the black dashed lines. When the interval between $t - \Delta t$ 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.

should be reproduced in the chimera animation as well. The contact objective includes two terms.

$$E_{\text{contact}} = \sum_{p \in \text{points}} \min(h(p), 0)^2 + \sum_{c \in \text{contacts}} (\text{dist}(p_c) - k_c)^2. \tag{4.2}$$

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 [42]. Here, p_c is the point on the body in contact with the ground surface or any object when the contact occurs at time ϕ_c. The target distance k_c is modulated based on the importance w_c at the current time ϕ such that

$$\begin{aligned} k_c &= (1 - w_c)\text{dist}(p_c) + w_c \text{dist}(\hat{p}_c), \\ w_c &= \max(1 - \frac{|\phi - \phi_c|}{\sigma}, 0), \end{aligned} \tag{4.3}$$

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. σ 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 4.3). E_{global}

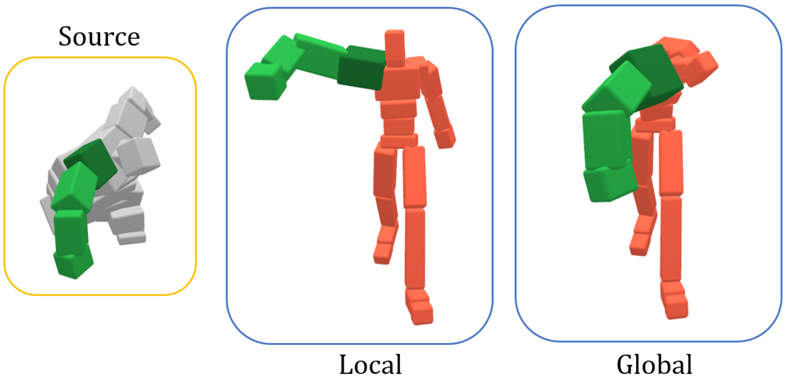

Figure 4.3: Arm transplanting with respect to the local coordinate system vs the global coordinate system. The dark green colored body is the root of the transplanted arm.

serves this purpose.

$$E_{\text{global}} = \sum_{i \in N} \| \log(\bar{q}_i^{-1} q_i) \|^2, \tag{4.4}$$

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 [103]. 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.

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

$$E_{\text{reg}} = \| \mathbf{p} - \hat{\mathbf{p}} \|^2 + w_{\text{reg}} \sum_{\Phi_k \in \text{neighbor}(\Phi)} \| \mathbf{p} - \mathbf{p}_{\Phi_k} \|^2, \tag{4.5}$$

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 Φ within a user-specified threshold $\Delta\Phi$.

4.3.1 Training

Policy learning we will discuss in the next section generates numerous phase Φ 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 Φ, the optimization is performed and added to the training dataset only when the density around Φ is lower than a threshold. The density is estimated by

$$\text{density}(\Phi) \propto \sum_{\Phi_k \in \text{neighbor}(\Phi)} \exp\big(-\frac{1}{N}(\Phi_k - \Phi)^T(\Phi_k - \Phi)\big). \tag{4.6}$$

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

The regression network P_θ is a feedforward network with fully-connected layers.

$$P_\theta(\Phi) = \mathbf{d}_\Phi, \tag{4.7}$$

The regression network learns a mapping between phase alignment Φ 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

$$\theta = \underset{\theta}{\operatorname{argmin}} \sum_{\Phi} \|\mathbf{d}_\Phi - \mathbf{d}^*_\Phi\|^2, \tag{4.8}$$

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

4.4 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

$$\hat{M}_i(\phi + L_i) = \hat{M}_i(\phi), \tag{4.9}$$

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 ϕ_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 π 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.

4.4.1 State and Action

As reported by Park et al. [65], 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 $\Psi = (\psi_1, \cdots, \psi_N)$.

$$(D_t, \psi_t) = \pi(s_t). \tag{4.10}$$

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.

4.4.2 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 [2]. Our reward function consists of four reward terms.

$$r = w_{\text{tracking}} r_{\text{tracking}} + w_{\text{energy}} r_{\text{energy}} + w_{\text{align}} r_{\text{align}} + w_{\text{task}} r_{\text{task}}. \tag{4.11}$$

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

$$\begin{aligned} r_{\text{tracking}} &= r_{\text{q}} \cdot r_{\text{ee}}, \quad \text{where} \\ r_{\text{q}} &= \exp\Big(- \frac{1}{\sigma_{\text{q}}^2} \sum_{j \in \text{joints}} \| \log(q_j^{-1} \hat{q}_j) \|^2 \Big), \\ r_{\text{ee}} &= \exp\Big(- \frac{1}{\sigma_{\text{ee}}^2} \sum_{j \in \text{ee}} \| \hat{p}_j - p_j \|^2 \Big). \end{aligned} \tag{4.12}$$

r_{q} and r_{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.

$$\begin{aligned} r_{\text{energy}} &= -r_{\text{effort}} - r_{\text{CoT}}, \quad \text{where} \\ r_{\text{effort}} &= \sum_{j \in \text{dof}} |\ddot{q}_j|^2, \\ r_{\text{CoT}} &= \frac{\sum_{j \in \text{dof}} |\tau_j \cdot \dot{q}_j|}{m \|v\|}. \end{aligned} \tag{4.13}$$

$\dot{q}_j$ and $\ddot{q}_j$, respectively, are the angular velocity and acceleration of each joint. $\|v\|$ is the distance the chimera travels in Δt. m is the body mass and τ_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 [104, 105]. Both r_{tracking} and r_{energy} are continuous rewards.

Let ω be the estimated period of the episode.

$$\omega = \frac{1}{N}\sum_i \frac{\phi_i}{L_i}. \tag{4.14}$$

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

$$r_{\text{align}} = \exp\Big(-\frac{1}{\sigma^2_{\text{align}}}\sum_i (\cos(\phi_i^*) - \cos(\phi_i))^2 + (\sin(\phi_i^*) - \sin(\phi_i))^2\Big), \tag{4.15}$$

where ϕ_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.

$$r_{\text{task}} = \exp\Big(-\frac{\|v - (v\cdot\hat{d})\hat{d}\|^2}{\sigma^2_{\text{dir}}}\Big) + \exp\Big(-\frac{\|max(\hat{f} - f, 0)\|^2}{\sigma^2_{\text{force}}}\Big) \tag{4.16}$$

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.

4.4.3 Policy Update

We use the variable time-step PPO (Proximal Policy Optimization) algorithm to update the value and policy networks [2]. This algorithm is stable and invariant under the choice of time steps. Let R_c and R_s, respectively, be the sum of

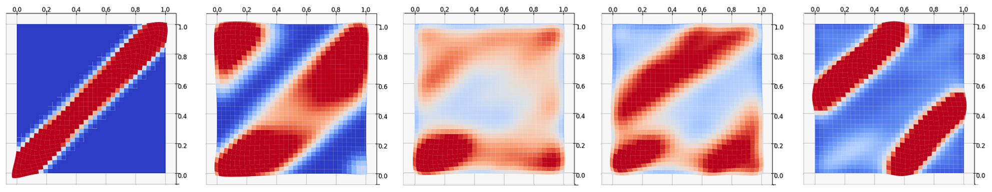

Figure 4.4: The phase density in queue A for the humanoid walking example in section 4.5.3. The X-coordinate represents the phase of the right leg, and the Y-coordinate represents the phase of the rest of the body. At the beginning of learning, the area around the initial phase alignments is densely populated. As the learning progresses, RL searches for better phase alignments and the high-density areas move accordingly.

continuous and spike rewards. Generalized Advantage Estimation (GAE) with variable time step $\Delta\omega$ is

$$\delta(\Phi) = \int_0^{\Delta\omega(\Phi)} \gamma^{\upsilon} R_{\mathrm{c}}(\Phi) d\upsilon + R_{\mathrm{s}}(\Phi) + \gamma^{\Delta\omega(\Phi)} V(\Phi + \Delta\Phi) - V(\Phi),$$

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

4.4.4 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. [1] suggested choosing initial states randomly along the reference trajectory. Won et al. [68] proposed an MCMC (Markov Chain Monte Carlo) method that samples initial states based on value functions. The curriculum learning method by Lee et al. [2] 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 (4.6). If the density around a tuple in A is beyond a certain threshold α, this tuple is considered a candidate for the initial states of subsequent episodes (see Figure 4.4). This tuple is moved to queue B if the density around the tuple in B is below a certain threshold β. Queue B maintains the candidates and facilitates uniform sampling of initial states. The tuples in B are uniformly populated within the threshold β. With a high threshold α, 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, α 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 2.5×10^5 and 5×10^2 tuples, respectively.

4.5 Experiments

In our system, the optimization and physics simulation parts are written in C++. The simulation system is based on Dart [97] and linear-time stable PD controllers [99]. 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 [98]. The policy network consists of four fully-connected layers of 1024 nodes, and the value network and the assembler network have two fully-connected layers of 512 nodes with ReLU activation. The

Table 4.1: Learning parameters.

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

Table 4.2: 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

learning parameters are summarized in Table 4.1. The learning rate of the policy network is 5×10^{-5} at the beginning of learning. The learning rate increases gradually to its maximum 2×10^{-4} when one million tuples are collected and then it decreases down to 5×10^{-5} again.

We use four source characters for our demonstrations: Humanoid, Bear, Horse and T-rex (See Figure 4.5). 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 4.2. 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.

4.5.1 Chimeras and their Motor Skills

We generated a diverse set of chimera animations from the source characters (see Figure 4.6). 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,

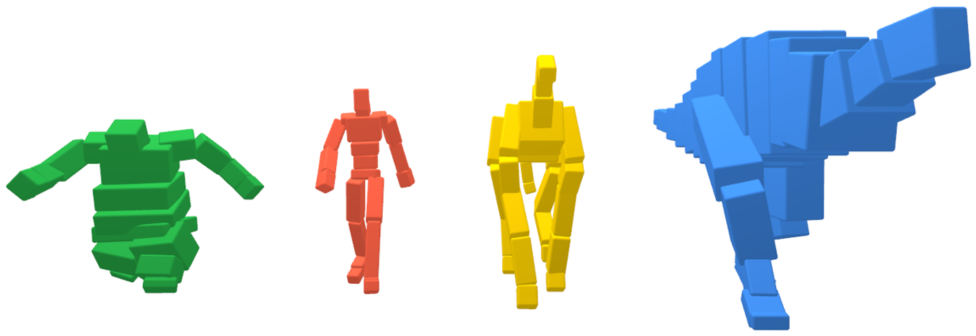

Figure 4.5: Four source characters. Bear, humanoid, horse and T-rex (left to right).

four-legged dinosaurs and centaurs to exotic ones such as six-armed humanoids, two-headed bears and eight-legged horses (see Figure 4.6 and Figure 4.7). 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 ϕ_1, the right arm ϕ_2, the left arm ϕ_3, and the horse lower-body ϕ_4) so that their

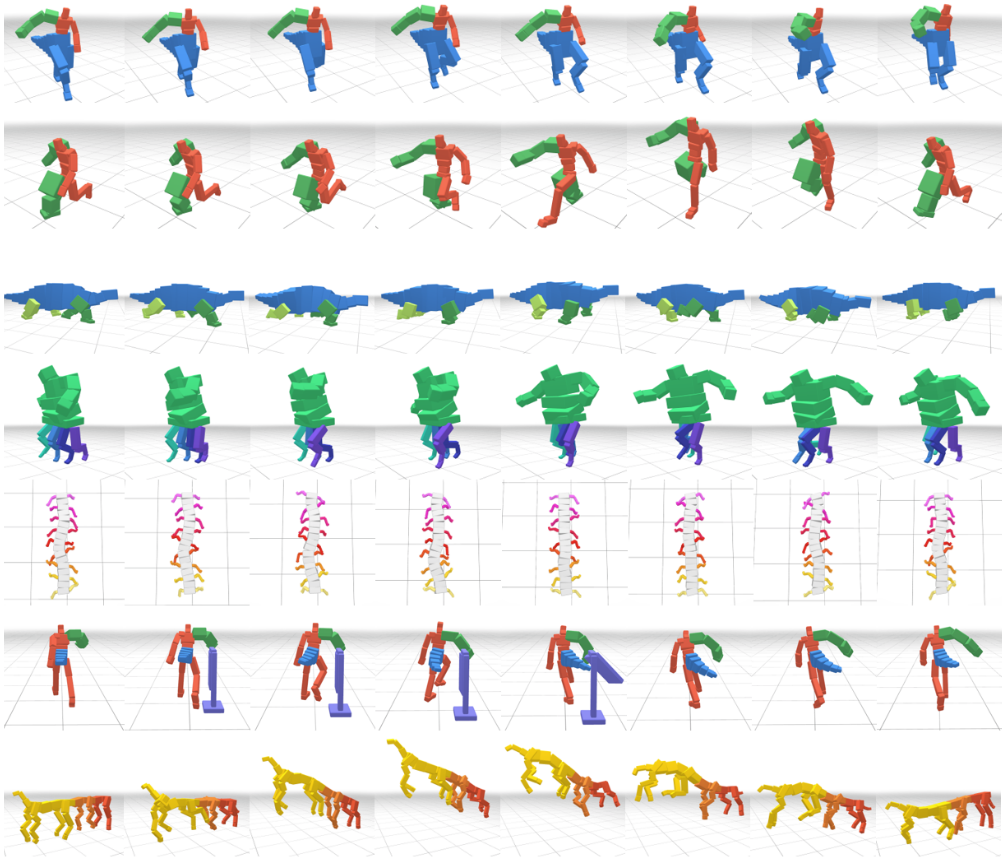

Figure 4.6: Snapshots of chimeras.

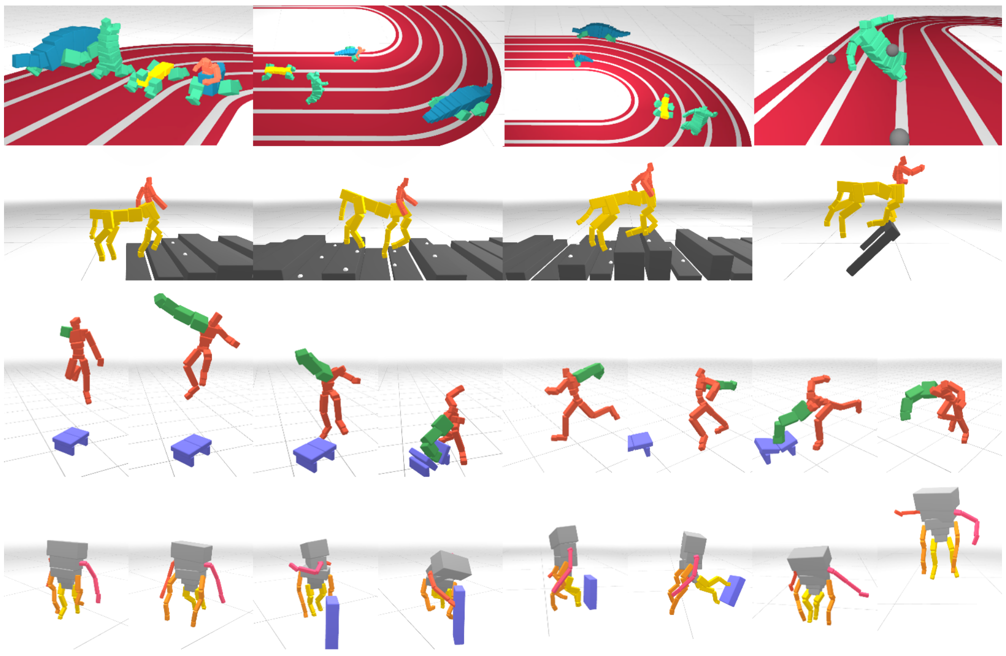

Figure 4.7: Snapshots of the bear-legged racing, the centaur running on uneven terrain, the bear-armed humanoid and the treant (top to bottom).

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 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.

4.5.2 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

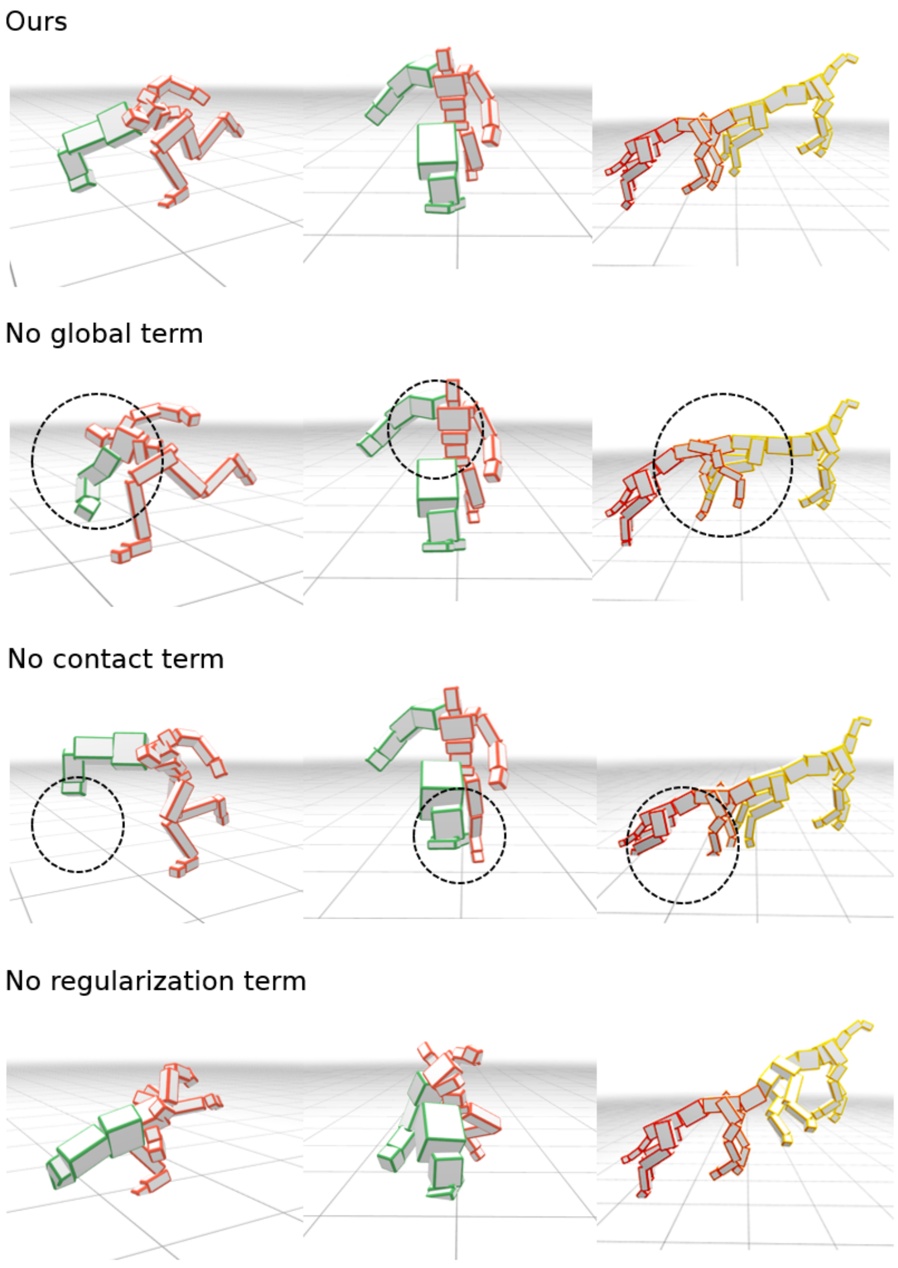

Figure 4.8: Snapshots of ablations on the part assembler

and the results optimized with all objectives. Figure 4.8 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 4.8). 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 4.8). 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.

4.5.3 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 [1]. The baseline B2 algorithm uses a single timewarp function for the whole body [2]. 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 4.3).

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 right leg and the left leg can have different phases. The source motion of the right leg is the mirror reflection

Table 4.3: 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

of the humanoid walking motion. Therefore, in the initial animation, both legs move back and forth synchronously (see Figure 4.4(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 4.4(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.

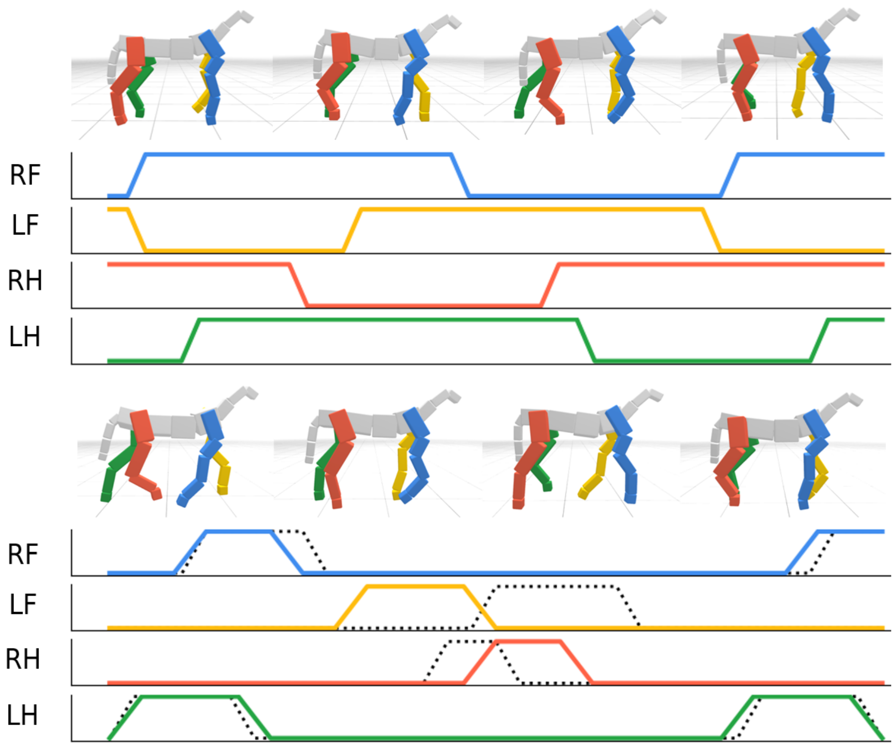

Figure 4.9: The change of quadrupedal gait according to the change of speed. The target velocity is 1.0 m/s in the upper graph and 2.4 m/s in the lower graph. Each line represents a sequence of contacts with the ground. The solid line is taken from the simulation, while the dotted line is taken from the actual trot data.

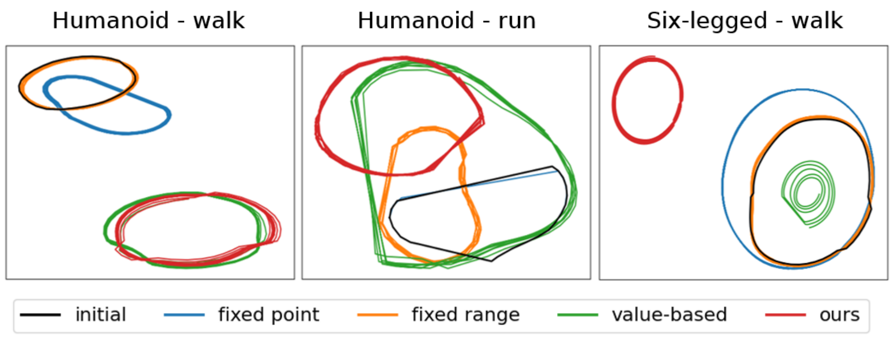

Figure 4.10: Visualization of phase alignments. Periodic trajectories are visualized as closed curves. The fixed-point algorithm failed to learn Humanoid-run.

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 4.3 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 [106]. 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.

4.5.4 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 [1]. The third algorithm chooses initial states adaptively based on marginal values and MCMC sampling [68]. The states with higher marginal values are sampled more frequently.

We conducted experiments with three examples in section 4.5.3: Humanoid-walk, Humanoid-run and Six-legged-walk. The initial phase alignments of these examples were far from the optimal found in section 4.5.3. Figure 4.10 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 4.3). Compared to the baseline algorithms, our algorithm is better in terms of finding the globally optimal solution.

4.5.5 Exploration vs Computation Cost

In this section, we evaluate the effect of threshold α 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

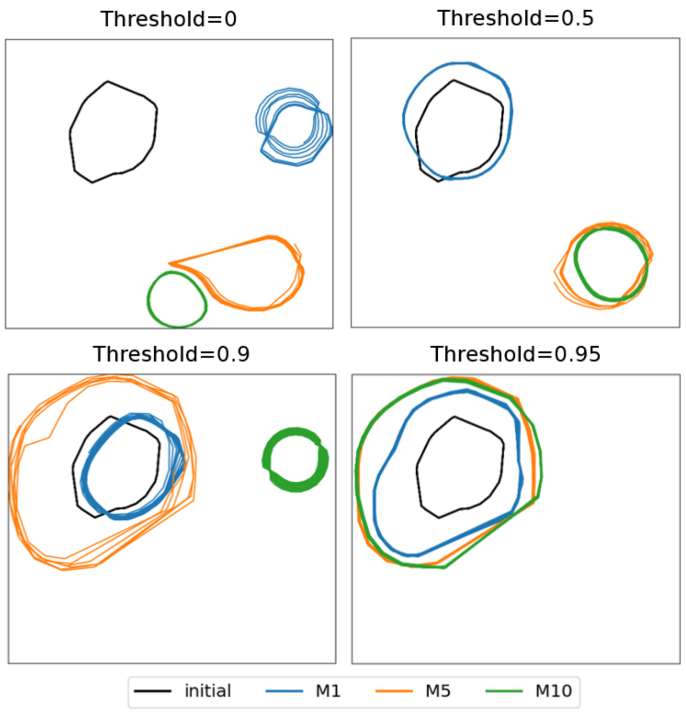

Figure 4.11: Arm swing phase visualization for centaurs of various mass distributions. Threshold=0 means no threshold, and threshold=0.95 means accepting only the top 5 percent.

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 4.11 shows how threshold α 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.

4.6 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. [107] 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.

Chapter 5

Learning Style Transfer from Minimal Motion Data

5.1 Overview

The meaning of human motion is basically determined by what kind of actions and what tasks are being performed. However, different people produce different movements even if they do the same actions with the same task. This difference is called style of the motion. There are a variety of factors that affect style. For example, it could be a personality/mood, such as proud, sad and happy, or it could be physical characteristics/conditions, such as heavy, old and fatigued. Style is an important factor that generates reality and add meaning to the motion. In order to effectively express the personality and state of the virtual characters, there have been many efforts to build a database for stylized motion in industries such as games and movies.

In the field of character animation, various studies have been conducted to generate stylized motion. The most well-known technique is motion style transfer, which extracts the style of an existing motion and applies it to other motions. The way how style affects motion is different for each motion and each style, so it is difficult to formulate an appropriate expression that extracts only the style from the motion while preserving its content. Researchers have successfully conducted style transfer of diverse motion and styles by learning networks or LTI models from large amounts of data [52, 51, 108]. In this method, the learned model immediately outputs the stylized result for the input instead of explicitly extracting and applying the style to the input motion. Since this method depends on the distribution of training data, it limits the skeleton structure and the range of styles that can be generated. It cannot generate stylized motion of a character different from the human form, and it is difficult to create a natural motion when the style is deeply related to the physical characteristics of the character.

We propose an algorithm for style transfer from a small number of motion data. Motion data may be a single motion pair consisting of one base motion and a stylized motion, or it may be multiple motion pairs. Our system consists of two sequential processes (See Figure 5.1). Our algorithm first searches the style feature to be applied to the each frame of the input motion from the source motion pair. Then it corrects the stylized motion to obey the laws of physics through physics simulation. As our algorithm does not depend heavily on the amount of source data, it can be applied to various characters and styles with minimum amount of motion data.

The contribution of our work is as follows. First, we design a style representation suitable for small motion data. In learning-based style transfer, the result is extrapolated and generate excessively distorted motion when training data is insufficient. Instead, we directly extract style feature through interpolated displacement mapping of nearest source motion data. Second, we propose

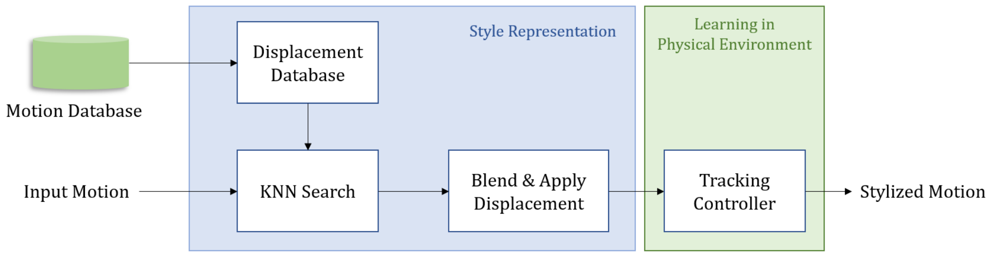

Figure 5.1: System overview.

adaptive methods to utilize source data more efficiently. According to the similarity of the source motion and the input motion, our system determines how many poses are required to calculate a displacement and which body parts will apply the displacement. Finally, the synthesized motion can be deformed in space and time to add proper physical effects and corrects errors like timing and penetration. We adopt the tracking controller of Section 3 to simulate a character in a physical environment and compensate for the lack of information about movement. We demonstrate the effectiveness of our system by generate diverse stylized motions like locomotion for various directions, jump, and attack motion from walk and run motion pair of 3 seconds each.

5.2 Style Representation

The purpose of our system is apply style to the input motion $\mathbf{M}^I$ given a base motion $\mathbf{M}^B$ and a stylized motion $\mathbf{M}^S$. Here, the base motion and stylized motion have the same content, and the input motion have the different content. $\mathbf{M}$ can be expressed as a sequential set of information about root transformation and joint rotations. Instead of inferring the style from a large amount of unlabeled data, we propose a way to extract styles from the differences between the pair of motions with the same content.

To compare the difference between the stylized motion and base motion, we temporally align the two motions through dynamic time warping. The objective of dynamic time warping is to find the best alignment between different curves by minimizing the hand-crafted loss function. We designed the difference in joint position as a loss function. As a result of time warping, each frame of $\mathbf{M}^B$ and $\mathbf{M}^S$ can be expressed as follows.

$$(\{\mathbf{M}^B(\phi_0), \mathbf{M}^S(\phi_0)\}, \{\mathbf{M}^B(\phi_1), \mathbf{M}^S(\phi_1)\}, \cdots, \{\mathbf{M}^B(\phi_T), \mathbf{M}^S(\phi_T)\}) \tag{5.1}$$

where ϕ is the phase variable on the normalized timeline.

In section 3.2, we define the difference between two motion as the spatio-temporal displacement formulation. Inspired by that, we define style as the displacement in three elements between two temporally paired frames: joint position, root movement, and speed. The displacement $\mathbf{d}_{\text{joint}}$ of the joint position is the difference between the joint angles of the two frames. $\mathbf{d}_{\text{joint}}$ of frame $\mathbf{p}^B$ and frame $\mathbf{p}^S$ can be defined as follows.

$$\mathbf{d}_{\text{joint}} = ((q_0^B)^{-1} q_0^S, \cdots, (q_L^B)^{-1} q_L^S) \tag{5.2}$$

where d_0 is the angular displacement of root joint, d_i for $i > 1$ is the angular displacement of a joint, and L is the number of joints. Note that root translation is not included in this displacement because it is processed separately.

Unlike the displacement formulation of section 3.2, we handle the translation of the root relative to previous frame. This formulation allows the style to be applied to input motions where the distance and direction of the route are different from the source motion. The displacement in root movement $(\mathbf{d}_{\text{length}}, \mathbf{d}_{\text{angle}})$ is defined as the difference in the moved distance and the moved direction of the root movement.

$$\begin{aligned} \mathbf{d}_{\text{length}} &= \frac{\| \Delta p_0^S \|}{\| \Delta p_0^B \|}, \\ \mathbf{d}_{\text{angle}} &= (\cos\theta, \sin\theta) \\ \theta &= \text{sign} \cdot \cos^{-1}(\frac{\Delta p_0^S \cdot \Delta p_0^B}{\| \Delta p_0^S \| \| \Delta p_0^B \|}) \end{aligned} \tag{5.3}$$

where Δp_0 is the distance of the root from the previous frame to the root in the current frame, and θ is the signed angle between the two vectors. The angle between the two vectors is expressed as a continuous value through the sinusoidal function for later calculation.

The displacement in speed $\mathbf{d}_{\text{speed}}$ is defined as the difference in the time interval between the current phase ϕ_t and the previous phase $\phi_t - 1$ in the normalized timeline.

$$\mathbf{d}_{\text{speed}} = \frac{(\phi^S)^{-1}(t) - (\phi^S)^{-1}(t-1)}{(\phi^B)^{-1}(t) - (\phi^B)^{-1}(t-1)} \tag{5.4}$$

where $(\phi^S)^{-1}, (\phi^B)^{-1}$ are functions that return the original frame index in stylized motion and base motion given a phase in the normalized timeline.

Given the frame in an input motion $\mathbf{p}^I$ and displacement ($\mathbf{d}_{\text{joint}}, \mathbf{d}_{\text{length}}, \mathbf{d}_{\text{angle}}, \mathbf{d}_{\text{speed}}$), a new output frame $\mathbf{p}^O$ can be obtained by inversely calculating the process above. The following section introduces the process of searching for appropriate displacement for each frame of the input motion.

5.2.1 Adaptive Nearest Search

Through the process above, the stylized displacement corresponding to each frame of the base motion can be calculated. Our system finds the frames in the base motion database most similar to each frame of the input motion and applies the corresponding displacement to the input frame. The searching algorithm to find a similar frame in the base motion is based on the nearest neighbor search algorithm of of Xia et al. [108].

We find the K closest frames in the base motion. Given a frame in the input motion $\mathbf{p}^I_\psi$ and a frame in the base motion $\mathbf{p}^B_\phi$, distance is defined as follows.

$$\text{distance}(\mathbf{p}^I_\psi, \mathbf{p}^B_\phi) = \frac{1}{2s+1} \sum_{-s \leq o \leq s} \parallel (\mathbf{p}^I_{o+\psi})^{-1} \mathbf{p}^B_{o+\phi} \parallel \tag{5.5}$$

where s is the size of the window. We use s=1.5 for all examples. The displacement for each frame of the input motion is defined as the average value of displacement for the K poses with the smallest distance. Interpolation of K displacements can produce smooth motions compared to the method of applying displacement from motion with the highest similarity. In the previous work, K was set to a fixed value. In our case, K is adjusted adaptively by the term uncertainty.

The uncertainty $\mathbf{u}$ was defined as the distance between the frame $\mathbf{p}^I$ of the input motion and the most similar frame $\mathbf{p}^B$ in the base motion. As the uncertainty increases, the style in the closest frame cannot be reproduced by applying the displacement of that frame. In this case, reflecting the average displacement of the a large number of postures rather than reflecting the style from a small number of postures is the better choice to reduce distortion and increase the quality of the stylization. We set K as follows.

$$K = \text{ceil}(\mathbf{u} * 10 + 3) \tag{5.6}$$

where ceil is a function that rounds down a value. Uncertainty is a variable that determines the number of K of the nearest search and also determines how accurately the reference postures should be tracked during the physics simulation process.

5.2.2 Part-wise Style Transplantation

The movement of human body parts is closely related each other. The displacement between the two motions includes information on correlation, and this correlation is preserved in the output motion by applying the style displacement of the base frame similar to the current input frame. However, when uncertainty is high, it is highly likely that displacement may not provide adequate information about spatial relationship. In addition, previous works demonstrated that various styles can be successfully transferred to new motions by processing each joint separately [108, 51]. Motivated by these ideas, we propose adaptive part-wise style transplantation.

In part-wise style transplantation, the body parts are divided into 4 body parts {LeftArm, RightArm, LeftLeg, RightLeg}. We search for similar frames from the input motion for each body parts and calculate the displacement $\mathbf{d}_{\text{part}}$. The process of finding the new displacement for the body part is the same as finding the full body displacement in section 5.2.1. We modify the displacement calculated for full body $\mathbf{d}_{\text{full}}$ according to the following equation.

$$\mathbf{d}^j_{\text{full}} = \begin{cases} w_{\text{part}}\mathbf{d}^j_{\text{part}} + (1 - w_{\text{part}})\mathbf{d}^j_{\text{full}} & \text{if } j \text{ in body part} \\ \mathbf{d}^j_{\text{full}} & \text{otherwise} \end{cases} \tag{5.7}$$

$$w_{\text{part}} = \text{clip}(5 * (\bar{\mathbf{u}}_{\text{full}} - (\bar{\mathbf{u}}_{\text{part}} + 0.2)), 0.0, 1.0)$$

where $\bar{\mathbf{u}}_{\text{full}}$ and $\bar{\mathbf{u}}_{\text{part}}$ are uncertainties calculated for the body part only. The reason for adding 0.2 to $\bar{\mathbf{u}}_{\text{part}}$ is to preserve the spatial relationship as much as possible without transplanting when the two values are not significantly different. In this equation, only the displacement of the motion is updated and the uncertainty is not updated. This is because the displacement with respect to the root or time may no longer be correct because the motion of the body part has changed.

5.3 Learning in Physical Environment

In Section 5.2, we synthesized a stylized motion by searching the best displacement based on the similarity with the base motion data. Since physics properties are not taken into account directly in the synthesis process, the generated motion often do not obey the laws of physics. This problem arises more frequently, especially when the amount of source motion is smaller. To correct such errors and boost the quality of the final motion, we simulate the synthesized motion in the physical environment.

5.3.1 Reinforcement learning formulation

The movement of the character is controlled by the tracking controller. Tracking controller is a control policy learned by deep reinforcement learning that makes a character track the synthesized motion in a physical environment. Our controller design is based on the tracking controller in Section 3. The state includes the rotation and velocity of the joints, the position of the end effectors, the next reference pose and current phase. The action is defined by the spatio-temporal displacement $(\mathbf{d}_t, \tau_t)$. The temporal displacement τ_t advances the phase $\phi' = \phi + \exp(\tau_t)$. The spatial displacement $\mathbf{d}_t$ is added to current reference pose and serves as a PD target to generate joint torques. Advancing the dynamics simulation by constant time updates the state at the next time step.

The reward design modified for our system is as follows.

$$r = w_{\text{tracking}} r_{\text{tracking}} + w_{\text{root}} r_{\text{root}} + w_{\text{regul}} r_{\text{regul}} \tag{5.8}$$

The tracking reward is for imitating a sequence of optimized poses and speed of the synthesized motion. The tracking reward includes three sub-terms,

$$r_{\text{tracking}} = w_q r_q * w_{\text{ee}} r_{\text{ee}} + w_{\text{time}} r_{\text{time}}, \tag{5.9}$$

which respectively penalize the discrepancies in joint angles, end-effector positions, and time between the reference motion and simulation.

$$\begin{aligned} r_q &= \exp\Big(- \frac{1}{\sigma_q^2} \sum_{i \in \text{joints}} \parallel \log(q_i^{-1} \hat{q}_i) \parallel^2 \Big) \\ r_{\text{ee}} &= \exp\Big(- \frac{1}{\sigma_{\text{ee}}^2} \sum_{j \in \text{ee}} \parallel \hat{x}_j - x_j \parallel^2 \Big) \\ r_{\text{time}} &= \exp\Big(- \frac{1}{\sigma_{\text{time}}^2} \parallel \exp(\hat{\tau}_t) - \exp(\tau_t) \parallel^2 \Big), \end{aligned} \tag{5.10}$$

where q_i is the joint rotation, x_j is the end effector 3d position and τ_t is the phase increment of the motion in current time t.

The root reward is for controlling the movement of character's root. The root reward includes two sub-terms,

$$r_{\text{root}} = w_{\text{length}} r_{\text{length}} + w_{\text{angle}} r_{\text{angle}}, \tag{5.11}$$

which respectively penalize the discrepancies in scale and direction the character moved between the reference motion and simulation.

$$\begin{aligned} r_{\text{length}} &= \exp\Big(- \frac{\|\hat{v}\| - \|v\|^2}{\sigma_{\text{length}}^2} \Big) \\ r_{\text{angle}} &= \exp\Big(- \frac{\theta^2}{\sigma_{\text{angle}}^2} \Big), \end{aligned} \tag{5.12}$$

where v is the root translation difference between the reference pose at current timestep and the reference pose at previous timestep. θ is the angle between the two vectors $\hat{v}$, v, which can be calculated according to Equation 5.3.

The regularization reward is for regularizing excessive torque.

$$r_{\text{regul}} = - \sum_{j \in \text{dof}} |\ddot{q}_j|^2, \tag{5.13}$$

where $\ddot{q}_j$ is the acceleration of each joint.

5.3.2 Uncertainty-based reward adjustment

How tightly the character should track the reference motion is determined by the σ value of each term of the reward. The larger the σ, the higher the reward even if you do not exactly follow the reference motion. We previously defined the term uncertainty as to how similar each frame of the input motion is to the base motion data. It is highly likely that the style is not applied correctly to the synthesized motion which has high uncertainty. If the tracking controller tries to follow this motion as reference motion, the controller has possibility of generating motion with low quality.

We propose an uncertainty-based reward adjustment that differentiates the degree of tracking the reference motion according to uncertainty. We adjusted σ of each term of reward to be proportional to the uncertainty $\mathbf{u}$ value of the current reference frame.

$$\begin{aligned} \sigma &= w_u \sigma_{\min} + (1 - w_u)\sigma_{\max} \\ w_u &= \min(0.5\mathbf{u}, 1), \end{aligned} \tag{5.14}$$

where $\sigma_{\min}$ and $\sigma_{\max}$ are boundary values of σ experimentally set for each term. When $\mathbf{u}$ is high, the agent can receive high rewards even if the character do not follow the reference well.

5.4 Experiments

Our simulation system and network setting is the same as the system in Section 3.6. We got bear character motion from NCsoft and other stylized locomotion data from Lafan dataset [109]. The computation time for learning varies depending on the length and type of the input motion. It takes 30 minutes to 4 hours on a single PC with AMD Ryzen 9 5950x. Most of the computation is done in the CPU.

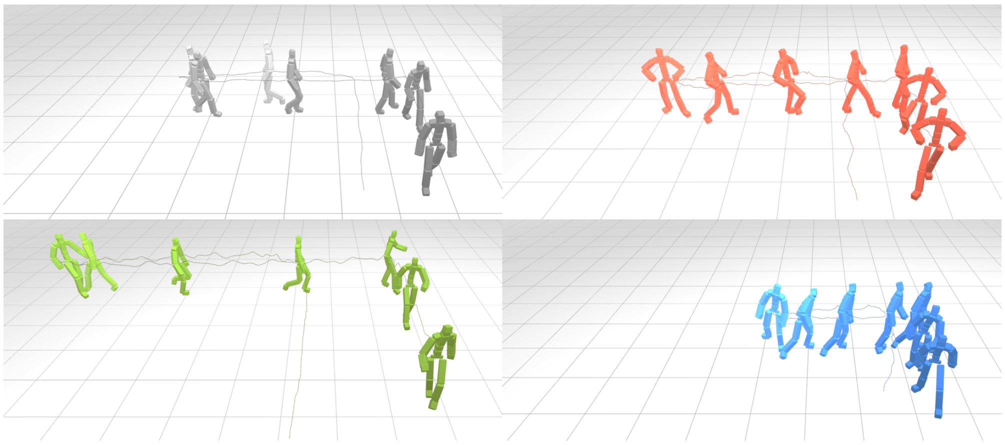

Figure 5.2: Long locomotion sequence with diverse styles. From top left: base, proud, brisk, and limping. Depending on the style, not only the movement of the joint but also the movement of the root and timing are different.

5.4.1 Stylized Motor Skills

Locomotion with diverse styles We transferred three different styles to a long locomotion sequence. The source motion pairs is walk-forward motion, each of which is 3 seconds long. Three styles are proud, brisk, and limping. The input motion is a long locomotion sequence that contains actions not exist in the source motion, such as 90 degree rotation, 180 degree rotation, and pause. Figure 5.2 shows the stylized result. The generated motion preserve the content of the long motion sequence, and the characteristics of each style such as the timing and movement of the root and joint are well transferred.

Dynamic bear motion We transferred the bear style to dynamic motion data. The source motion pairs are run-forward and walk-forward motions of humanoid and bear, each of which is 3 seconds long. The stylized motions are a motion that turns 90 degrees while walking, a motion that turns 180 degrees

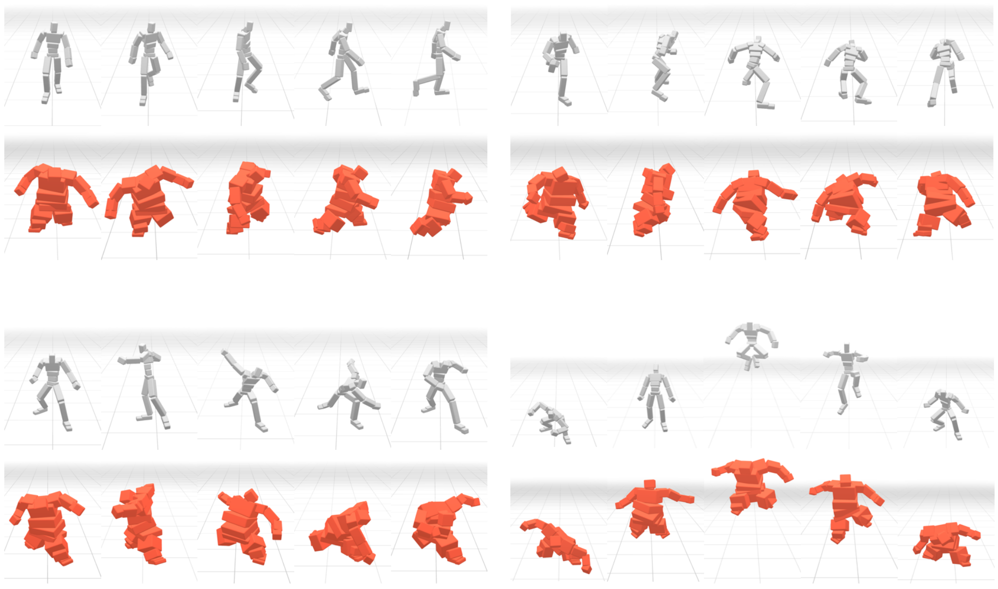

Figure 5.3: Input motions and bear-stylized motions. From top left: turn 90 degrees, turn 180 degrees, jump and attack.

while walking, a motion that turns 180 degrees while running, a jump motion, and an attack motion (See Figure 5.3). Compared to the input motion, the stylized motion is more logged and staggered, which makes the heavy bodies of the bear stand out. Our system shows that styles can be well applied to the motions even with contents not included in source motion database.

5.4.2 Comparison of Style Representation Methods

In this section, we compared our style representation method with two baseline methods. The first baseline algorithm is local LTI model used in the work of Xia et al. [108]. This method trains the parameters of the model which infers the displacement of the posture. The second algorithm is using displacement formulation of Section 3.2. Unlike our algorithm, this formulation handles root displacement in the same way as joint displacement.

We conducted experiments with three walking motions: a motion that turns 45 degrees, a motion that turns 90 degrees, and a motion that turns 180 degrees. As in the previous section, only walk straight motion was used as the source motion. Adaptive knn and part-wise transplantation were applied identically for all three algorithms. All algorithms generate natural stylized motion for the motion that turns 45 degrees. For the motion that turns 90 degrees, the first algorithm shows the movement of the legs trembling at the moment of rotation. The second algorithm results in the root motion vibrating in the wrong direction after rotating. For the motion that turns 180 degrees, the trembling of the leg in the first algorithm is even worse, and make it difficult to recognize the movement of the leg. In the second algorithm, the root, which should be stationary, shakes back and forth at the moment of rotation. On the other hand, our style representation method shows little deterioration in quality as the similarity of motion is lowered, and aligns the style of the root motion properly according to the input motion.

5.4.3 Effectiveness of Learning in Physical Environment

In this section, we validate the effectiveness of physics simulation. We used three examples: a motion that turns 180 degrees while running, a jump motion, and an attack motion. We compared the motion generated by section 5.2 with the motion learned in the physics environment based on it. Without learning in physical environment, various physical violations makes the motions unnatural (See Figure 5.4. In a motion that turns 180 degrees while running, the foot penetrates the ground when making a turn. In the jump motion, penetration occurs too, and the bear jumps with a light follow-through despite its heavy

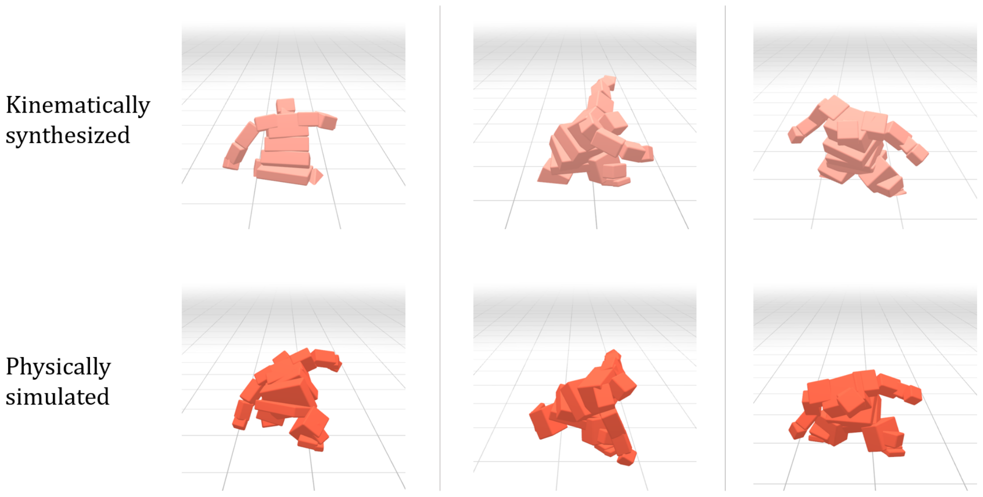

Figure 5.4: The first row is the motion generated by section 5.2, and the second row is the motion learned in the physics environment based on the first row. Each action is turning 180 degrees, attacking and landing after jump from the left.

body. In the attack motion, it seems unnatural that the bear maintains balance easily while swinging its heavy arm quickly. Our algorithm including physical simulation adds a variety of physical effects and thus makes the motion look more realistic.

5.5 Discussion

We propose an algorithm that extracts and applies styles directly to new motion from source motions. Because our algorithm formulates style as displacement of temporally aligned motion, it can extract style with minimal amount of source motion data compared to the existing learning-based model. The use of an adaptive number of neighbors for style interpolation and part-wise translation compensates for the lack of source motion volume. Additionally, as it is physically simulated, it can reflect the physical properties of characters and motions that are difficult to obtain from source motion data.

The quality of the motion is degraded when a motion similar to the input motion cannot be found in the source database. To prevent this, our tracking controller loosely tracks the motion when the uncertainty of the reference motion is high. It generates better results than tight tracking of an incorrect reference motion, but at the same time the generated motion may look unnatural as it entirely relies on the result of physics simulation. Adopting the results of the pretrained network for the open-source motion database when similar motion does not exist in the source database can be a good alternative.

Our displacement design and metric for similarity are mainly focused on joint rotations. It is true that joint rotation has a big influence on style, but there are many other factors that can be seen from motion, such as global orientation of joints, end effector positions and relationship with other joints. If these elements are also considered in style formulation, the types of styles that can be reproduced will be more diverse.

Another limitation of our work is that the source motion must exist in pairs. Since it is difficult to collect paired data, many researches on style transfer from unpaired data have been introduced recently. In the case of our work, it is not realistically impossible to prepare paired data because it does not require a large amount of data unlike other works. Nevertheless, it would be more convenient if the system can do style transfer from a small amount of unpaired data.

Despite these limitations, our work shows the potential for style transfer from small amounts of motion. It goes without saying that the quality of style transfer is proportional to the amount of motion data in our work, too. However, some actions and styles can be created with high quality with a small amount of data, while others will require a larger amount of data. Flexibly use the amount of source data greatly increases the practicality of the system. For example, designers can estimate the amount of data needed to generate stylize motion from our work and draw only as much source data as they need.

Chapter 6

Conclusion

6.1 Contribution

Physics-based motion editing has merit in that it can edit motions for tasks that are deeply affected by dynamic factors. Because of the high dimensional control space of physics simulation, physics-based motion editing has been less studied than data-driven motion editing. Recent advances in deep reinforcement learning have made it possible to reproduce dynamic motions in a physical environment. In this thesis, we explored various methods that discover novel motions by simulating motions in the physical environment.

The key to motion editing is to increase diversity while maintaining the quality of the motion. We solved the problem by combining motion optimization with the physics simulation. First, we introduce a method of editing a single motion clip for parameterized task spaces. Our algorithm learns to modify the joint positions of the character and the spacing between each frame, so it can edit the motion not only spatially but also temporally. The system boosts both the learning efficiency and the quality of edited motions by reusing high-quality motions generated during learning process. Second, we propose a method to create a new combination of motion and character by reassembling multiple motions and characters. In the first work, the time interval between frames is expressed as a one-dimensional function, but in this work, the temporal alignments between multiple motions are designed as a multi-dimensional function. This design allows exploring infinite combinations of motions to be generated by varying temporal alignments. At the same time, the motion is optimized so that the semantics of motions such as contact and shape are always preserved. Finally, we efficiently extracted and applied style to diverse motions with small motion dataset by adaptive nearest search and physics based simulation.

The purpose of using motion optimization and physics simulation in our systems is to find a space of human-like movement in the entire area that can be created by combinations of movements of all joints. This can also be inferred from large amounts of motion data. This method has the advantage of reducing the effort of designing objective terms to maintain the quality of synthesized motion. Evaluating human-likeness from data is greatly influenced by the distribution of data. Although there exist a large amount of data online, the distribution of data is uneven, and the proportion of daily movements is overwhelmingly large. In order to create various motions deviating from existing motion data, it is necessary to consider the nature of human-like movement like our methods do.

Our motion editing algorithms can contribute to creating a massive amount of physically simulated motion database. This motion database includes a rich repertoire of motions that require a lot of effort to capture or key-frame. For example, highly dynamic motion, motion that interacts with complex environments, motion in imaginary environment or motion of imaginary body structure can be included. The controller equipped with this database enables the character to reproduce much more diverse and lifelike movements. In a digital twin or mixed reality environment that has recently received a lot of attention, this diversity of motion will make the virtual characters more realistic.

6.2 Future Work

We showed novel motion editing algorithms for various conditions such as body configuration, environmental condition, spacetime constraints, and style. We are also interested in editing motions for complex conditions not covered in our work. Motion editing for multi-person scene where interaction of multiple characters occurs is one of them. How interactions with other characters affects the motion of each characters should be considered in the editing process. Moving environment, abstract conditions that are difficult to formulate as mathematical expressions and conditions with complex rules like sports are other examples of conditions that can be explored as a future work.

We optimized the motion for several terms designed to keep the semantics of the original motion. Nonetheless, they often result in unnatural motion. Naturalness is determined by complex elements of motion, and sometimes different criteria are required for each type of motion. Therefore, the terms we defined cannot exactly match the area that looks natural for human perception, and in some cases, it will be narrower or wider than the area. Evaluating naturalness from real human motion data would work similarly to human perception. However, as mentioned above, the range of motion that can be evaluated by this way is limited to the range of existing data. If the above two methods can be combined properly, it will be possible to further boost the quality of the generated motion. Similarly, our system is evaluated qualitatively rather than quantitatively in our experiments because it is difficult to define a measure of naturalness for unseen motion. Designing an appropriate metric for naturalness will be also be meaningful for quantitative evaluation of the algorithm.

Our system must learn anew whenever a new task or motion is given. If editing different types of motions in a single network or editing for unseen motions or conditions is possible, a large amount of motion data can be generated in a minimum amount of time. Recently, in the field of character control, methods that quickly adapts to a new task by finetuning a pretrained network equipped with a large number of motor skills have been proposed [110, 111]. An reusable network that can be fine-tuned at a low cost even with many variations would be helpful.

To use this system in industry, it should be possible to generate motions automatically in large quantities or be easy for artists to use. For a massive production, it is necessary to automate processes that are currently performed manually. Designing task space and setting appropriate parameters for the physics simulation process every time a new character and motion is laborious even for experts who are familiar with physics simulation and the mechanism of each motion. It takes a lot of efforts as the amount of source motion increases. If there is an algorithm that automates the above factors, the practicality of our physics-based motion editing system will be increased. Developing an algorithm that serves as a medium to convert editing conditions written in numbers into a user-friendly interface will also be helpful.

Bibliography

[1] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills," *ACM Transactions on Graphics*, vol. 37, no. 4, 2018.

[2] S. Lee, S. Lee, Y. Lee, and J. Lee, "Learning a family of motor skills from a single motion clip," *ACM Transactions on Graphics*, vol. 40, no. 4, 2021.

[3] Y. Huang, M. Kaufmann, E. Aksan, M. J. Black, O. Hilliges, and G. Pons-Moll, "Deep inertial poser: Learning to reconstruct human pose from sparse inertial measurements in real time," *ACM Transactions on Graphics*, vol. 37, no. 6, 2018.

[4] X. Yi, Y. Zhou, and F. Xu, "TransPose: real-time 3d human translation and pose estimation with six inertial sensors," *ACM Transactions on Graphics*, vol. 40, no. 4, 2021.

[5] A. Winkler, J. Won, and Y. Ye, "Questsim: Human motion tracking from sparse sensors with simulated avatars," in *Proceedings of ACM SIGGRAPH Asia '22*, 2022.

[6] A. Dittadi, S. Dziadzio, D. Cosker, B. Lundell, T. J. Cashman, and J. Shotton, "Full-body motion from a single head-mounted device: Gen-

erating smpl poses from partial observations," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 11687–11697, 2021.

[7] S. Aliakbarian, P. Cameron, F. Bogo, A. Fitzgibbon, and T. J. Cashman, "FLAG: Flow-based 3d avatar generation from sparse observations," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13253–13262, 2022.

[8] R. Urtasun, D. J. Fleet, and P. Fua, "3d people tracking with gaussian process dynamical models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 238–245, 2006.

[9] D. Ormoneit, H. Sidenbladh, M. Black, and T. Hastie, "Learning and tracking cyclic human motion," *Advances in Neural Information Processing Systems*, 2000.

[10] M. R. I. Hossain and J. J. Little, "Exploiting temporal information for 3d human pose estimation," in *Proceedings of the European Conference on Computer Vision*, pp. 68–84, 2018.

[11] D. Mehta, S. Sridhar, O. Sotnychenko, H. Rhodin, M. Shafiei, H.-P. Seidel, W. Xu, D. Casas, and C. Theobalt, "VNect: Real-time 3d human pose estimation with a single rgb camera," *ACM Transactions On Graphics*, vol. 36, no. 4, 2017.

[12] K. Lin, L. Wang, and Z. Liu, "End-to-end human pose and mesh reconstruction with transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1954–1963, 2021.

[13] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black, "SMPL: A skinned multi-person linear model," *ACM Transactions On Graphics*, vol. 34, no. 6, 2015.

[14] M. Kocabas, N. Athanasiou, and M. J. Black, "Vibe: Video inference for human body pose and shape estimation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 5253–5263, 2020.

[15] A. Kanazawa, J. Y. Zhang, P. Felsen, and J. Malik, "Learning 3d human dynamics from video," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 5614–5623, 2019.

[16] X. B. Peng, A. Kanazawa, J. Malik, P. Abbeel, and S. Levine, "Sfv: Reinforcement learning of physical skills from videos," *ACM Transactions On Graphics*, vol. 37, no. 6, 2018.

[17] R. Yu, H. Park, and J. Lee, "Human dynamics from monocular video with dynamic camera movements," *ACM Transactions on Graphics*, vol. 40, no. 6, 2021.

[18] S. I. Park, H. J. Shin, and S. Y. Shin, "On-line locomotion generation based on motion blending," in *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, p. 105–111, 2002.

[19] C. Rose, M. F. Cohen, and B. Bodenheimer, "Verbs and adverbs: Multidimensional motion interpolation," *IEEE Computer Graphics and Applications*, vol. 18, no. 5, pp. 32–40, 1998.

[20] T. Mukai and S. Kuriyama, "Geostatistical motion interpolation," *ACM Transactions on Graphics*, vol. 24, no. 3, 2005.

[21] J. M. Wang, D. J. Fleet, and A. Hertzmann, "Gaussian process dynamical models for human motion," *IEEE transactions on pattern analysis and machine intelligence*, vol. 30, no. 2, pp. 283–298, 2007.

[22] K. Grochow, S. L. Martin, A. Hertzmann, and Z. Popović, "Style-based inverse kinematics," *ACM Transactions on Graphics*, vol. 23, no. 3, p. 522–531, 2004.

[23] S. Levine, J. M. Wang, A. Haraux, Z. Popović, and V. Koltun, "Continuous character control with low-dimensional embeddings," *ACM Transactions on Graphics*, vol. 31, no. 4, 2012.

[24] D. Holden, J. Saito, and T. Komura, "A deep learning framework for character motion synthesis and editing," *ACM Transactions on Graphics*, vol. 35, no. 4, 2016.

[25] K. Lee, S. Lee, and J. Lee, "Interactive character animation by learning multi-objective control," *ACM Transactions on Graphics*, vol. 37, no. 6, 2018.

[26] S. Starke, Y. Zhao, T. Komura, and K. Zaman, "Local motion phases for learning multi-contact character movements," *ACM Transactions on Graphics*, vol. 39, no. 4, 2020.

[27] A. Witkin and M. Kass, "Spacetime constraints," *ACM Transactions on Graphics*, vol. 22, no. 4, 1988.

[28] M. Gleicher, "Motion editing with spacetime constraints," in *Proceedings of the 1997 symposium on Interactive 3D graphics*, pp. 139–ff, 1997.

[29] M. Kim, K. Hyun, J. Kim, and J. Lee, "Synchronized multi-character motion editing," *ACM Transactions on Graphics*, vol. 28, no. 3, 2009.

[30] K. W. Sok, K. Yamane, J. Lee, and J. Hodgins, "Editing dynamic human motions via momentum and force," in *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer animation*, pp. 11–20, 2010.

[31] J. Lee and S. Y. Shin, "A hierarchical approach to interactive motion editing for human-like figures," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pp. 39–48, 1999.

[32] L. Ikemoto and D. A. Forsyth, "Enriching a motion collection by transplanting limbs," in *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 99–108, 2004.

[33] R. Heck, L. Kovar, and M. Gleicher, "Splicing upper-body actions with locomotion," vol. 25, no. 3, pp. 459–466, 2006.

[34] W.-S. Jang, W.-K. Lee, I.-K. Lee, and J. Lee, "Enriching a motion database by analogous combination of partial human motions," *The Visual Computer*, vol. 24, no. 4, pp. 271–280, 2008.

[35] S. Starke, Y. Zhao, F. Zinno, and T. Komura, "Neural animation layering for synthesizing martial arts movements," *ACM Transactions on Graphics*, vol. 40, no. 4, 2021.

[36] D.-K. Jang, S. Park, and S.-H. Lee, "Motion puzzle: Arbitrary motion style transfer by body part," *ACM Transactions on Graphics*, 2022.

[37] J. McCann, N. S. Pollard, and S. S. Srinavisa, "Physics-based motion retiming," in *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, p. 205–214, 2006.

[38] N. S. Pollard and F. Behmaram-Mosavat, "Force-based motion editing for locomotion tasks," in *2000 IEEE International Conference on Robotics and Automation*, pp. 663–669, 2000.

[39] Y. Abe, C. K. Liu, and Z. Popović, "Momentum-based parameterization of dynamic character motion," in *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 173–182, 2004.

[40] A. Majkowska and P. Faloutsos, "Flipping with physics: motion editing for acrobatics," in *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 35–44, 2007.

[41] M. Gleicher, "Retargetting motion to new characters," in *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pp. 33–42, 1998.

[42] H. J. Shin, J. Lee, S. Y. Shin, and M. Gleicher, "Computer puppetry: An importance-based approach," *ACM Transactions on Graphics*, vol. 20, no. 2, p. 67–94, 2001.

[43] T. Jin, M. Kim, and S.-H. Lee, "Aura mesh: Motion retargeting to preserve the spatial relationships between skinned characters," *Computer Graphics Forum*, vol. 37, no. 2, pp. 311–320, 2018.

[44] E. S. L. Ho, T. Komura, and C.-L. Tai, "Spatial relationship preserving character motion adaptation," *ACM Transactions on Graphics*, vol. 29, no. 4, pp. 33:1–33:8, 2010.

[45] J. K. Hodgins and N. S. Pollard, "Adapting simulated behaviors for new characters," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 153–162, ACM, 1997.

[46] C. Hecker, B. Raabe, R. W. Enslow, J. DeWeese, J. Maynard, and K. van Prooijen, "Real-time motion retargeting to highly varied user-created morphologies," in *Proceedings of ACM SIGGRAPH '08*, 2008.

[47] Y. Seol, C. O'Sullivan, and J. Lee, "Creature features: online motion puppetry for non-human characters," in *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 213–221, 2013.

[48] K. Wampler, Z. Popović, and J. Popović, "Generalizing locomotion style to new animals with inverse optimal regression," *ACM Transactions on Graphics*, vol. 33, no. 4, 2014.

[49] K. Aberman, R. Wu, D. Lischinski, B. Chen, and D. Cohen-Or, "Learning character-agnostic motion for motion retargeting in 2d," *ACM Transactions on Graphics*, vol. 38, no. 4, 2019.

[50] M. E. Yumer and N. J. Mitra, "Spectral style transfer for human motion between independent actions," *ACM Transactions on Graphics*, vol. 35, no. 4, 2016.

[51] E. Hsu, K. Pulli, and J. Popović, "Style translation for human motion," *ACM Transactions on Graphics*, vol. 24, no. 3, p. 1082–1089, 2005.

[52] K. Aberman, Y. Weng, D. Lischinski, D. Cohen-Or, and B. Chen, "Unpaired motion style transfer from video to animation," *ACM Transactions on Graphics*, vol. 39, no. 4, 2020.

[53] L.-K. Ma, Z. Yang, X. Tong, B. Guo, and K. Yin, "Learning and exploring motor skills with spacetime bounds," vol. 40, no. 2, pp. 251–263, 2021.

[54] K. Yin, K. Loken, and M. Van de Panne, "Simbicon: Simple biped locomotion control," *ACM Transactions on Graphics*, vol. 26, no. 3, 2007.

[55] T. Kwon and J. Hodgins, "Control systems for human running using an inverted pendulum model and a reference motion capture sequence," in *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, p. 129–138, 2010.

[56] Y.-Y. Tsai, W.-C. Lin, K. B. Cheng, J. Lee, and T.-Y. Lee, "Real-time physics-based 3d biped character animation using an inverted pendulum model," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 2, p. 325–337, 2010.

[57] P. Hämäläinen, Perttu, J. Rajamäki, and C. K. Liu, "Online control of simulated humanoids using particle belief propagation," *ACM Transactions on Graphics*, vol. 34, no. 4, 2015.

[58] S. Hong, D. Han, K. Cho, J. S. Shin, and J. Noh, "Physics-based full-body soccer motion control for dribbling and shooting," *ACM Transactions on Graphics*, vol. 38, no. 4, 2019.

[59] L. Liu, K. Yin, M. van de Panne, T. Shao, and W. Xu, "Sampling-based contact-rich motion control," *ACM Transactions on Graphics*, vol. 29, no. 4, 2010.

[60] Y. Ye and C. K. Liu, "Synthesis of detailed hand manipulations using contact sampling," *ACM Transactions on Graphics*, vol. 31, no. 4, 2012.

[61] G. Bharaj, S. Coros, B. Thomaszewski, J. Tompkin, B. Bickel, and H. Pfister, "Computational design of walking automata," in *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, p. 93–100, 2015.

[62] L. Liu, K. Yin, and B. Guo, "Improving sampling-based motion control," *Computer Graphics Forum*, vol. 34, no. 2, p. 415–423, 2015.

[63] L. Liu, K. Yin, M. van de Panne, and B. Guo, "Terrain runner: Control, parameterization, composition, and planning for highly dynamic motions," *ACM Transactions on Graphics*, vol. 31, no. 6, 2012.

[64] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. Eslami, M. Riedmiller, *et al.*, "Emergence of locomotion behaviours in rich environments," *arXiv preprint arXiv:1707.02286*, 2017.

[65] S. Park, H. Ryu, S. Lee, S. Lee, and J. Lee, "Learning predict-and-simulate policies from unorganized human motion data," *ACM Transactions on Graphics*, vol. 38, no. 6, 2019.

[66] K. Bergamin, S. Claver, D. Holden, and J. R. Forbes, "Drecon: Data-driven responsive control of physics-based characters," *ACM Transactions on Graphics*, vol. 38, no. 6, 2019.

[67] J. Won, D. Gopinath, and J. Hodgins, "A scalable approach to control diverse behaviors for physically simulated characters," *ACM Transactions on Graphics*, vol. 39, no. 4, 2020.

[68] J. Won and J. Lee, "Learning body shape variation in physics-based characters," *ACM Transactions on Graphics*, vol. 38, no. 6, 2019.

[69] I. Mordatch, K. Lowrey, G. Andrew, Z. Popovic, and E. V. Todorov, "Interactive control of diverse complex characters with neural networks," *Advances in neural information processing systems*, vol. 28, p. 3132–3140, 2015.

[70] S. Lee, M. Park, K. Lee, and J. Lee, "Scalable muscle-actuated human simulation and control," *ACM Transactions on Graphics*, vol. 38, no. 4, 2019.

[71] J. Park, S. Min, P. S. Chang, J. Lee, M. S. Park, and J. Lee, "Generative gaitnet," in *Proceedings of ACM SIGGRAPH '22*, 2022.

[72] J. Merel, S. Tunyasuvunakool, A. Ahuja, Y. Tassa, L. Hasenclever, V. Pham, T. Erez, G. Wayne, and N. Heess, "Catch carry: Reusable neural controllers for vision-guided whole-body tasks," *ACM Transactions on Graphics*, vol. 39, no. 4, 2020.

[73] K. Lee, S. Min, S. Lee, and J. Lee, "Learning time-critical responses for interactive character control," *ACM Transactions on Graphics*, vol. 40, no. 4, 2021.

[74] H. Y. Ling, F. Zinno, G. Cheng, and M. Van De Panne, "Character controllers using motion vaes," *ACM Transactions on Graphics*, vol. 39, no. 4, 2020.

[75] J. Won, J. Park, K. Kim, and J. Lee, "How to train your dragon: Example-guided control of flapping flight," *ACM Transactions on Graphics*, vol. 36, no. 6, 2017.

[76] S. Min, J. Won, S. Lee, J. Park, and J. Lee, "Softcon: Simulation and control of soft-bodied animals with biomimetic actuators," *ACM Transactions on Graphics*, vol. 38, no. 6, 2019.

[77] D. Reda, H. Y. Ling, and M. Van De Panne, "Learning to brachiate via simplified model imitation," in *Proceedings of ACM SIGGRAPH '22*, 2022.

[78] H. Zhang, S. Starke, T. Komura, and J. Saito, "Mode-adaptive neural networks for quadruped motion control," *ACM Transactions on Graphics*, vol. 37, no. 4, 2018.

[79] Y.-S. Luo, J. H. Soeseno, T. P.-C. Chen, and W.-C. Chen, "Carl: Controllable agent with reinforcement learning for quadruped locomotion," *ACM Transactions on Graphics*, vol. 39, no. 4, 2020.

[80] X. B. Peng, Z. Ma, P. Abbeel, S. Levine, and A. Kanazawa, "Amp: Adversarial motion priors for stylized physics-based character control," *ACM Transactions on Graphics*, vol. 40, no. 4, 2021.

[81] J. Merel, Y. Tassa, D. TB, S. Srinivasan, J. Lemmon, Z. Wang, G. Wayne, and N. Heess, "Learning human behaviors from motion capture by adversarial imitation," *arXiv preprint arXiv:1707.02201*, 2017.

[82] I. Mordatch, E. Todorov, and Z. Popović, "Discovery of complex behaviors through contact-invariant optimization," *ACM Transactions on Graphics*, vol. 31, no. 4, 2012.

[83] M. Al Borno, M. De Lasa, and A. Hertzmann, "Trajectory optimization for full-body movements with complex contacts," *IEEE transactions on visualization and computer graphics*, vol. 19, pp. 1405–1414, 2012.

[84] W. Yu, G. Turk, and C. K. Liu, “Learning symmetric and low-energy locomotion,” *ACM Transactions on Graphics*, vol. 37, no. 4, 2018.

[85] P. G. Kry, L. Revéret, F. Faure, and M.-P. Cani, “Modal locomotion: Animating virtual characters with natural vibrations,” in *Computer Graphics Forum*, pp. 289–298, 2009.

[86] J. Tan, G. Turk, and C. K. Liu, “Soft body locomotion,” *ACM Transactions on Graphics*, vol. 28, no. 3, 2012.

[87] J. M. Bern, K.-H. Chang, and S. Coros, “Interactive design of animated plushies,” *ACM Transactions on Graphics*, vol. 36, no. 4, 2017.

[88] C. Schulz, C. von Tycowicz, H.-P. Seidel, and K. Hildebrandt, “Animating deformable objects using sparse spacetime constraints,” *ACM Transactions on Graphics*, vol. 33, no. 4, 2014.

[89] S. Coros, S. Martin, B. Thomaszewski, C. Schumacher, R. Sumner, and M. Gross, “Deformable objects alive!,” *ACM Transactions on Graphics*, vol. 31, no. 4, 2012.

[90] L. Liu, K. Yin, B. Wang, and B. Guo, “Simulation and control of skeleton-driven soft body characters,” *ACM Transactions on Graphics*, vol. 32, no. 6, 2013.

[91] Z. Yin, Z. Yang, M. Van De Panne, and K. Yin, “Discovering diverse athletic jumping strategies,” *ACM Transactions on Graphics*, vol. 40, no. 4, 2021.

[92] T. Tao, M. Wilson, R. Gou, and M. Van De Panne, “Learning to get up,” in *Proceedings of ACM SIGGRAPH ’22*, 2022.

[93] Z. Yang, K. Yin, and L. Liu, “Learning to use chopsticks in diverse gripping styles,” *ACM Transactions on Graphics*, vol. 41, no. 4, 2022.

[94] S. Agrawal, S. Shen, and M. van de Panne, “Diverse motions and character shapes for simulated skills,” *IEEE transactions on visualization and computer graphics*, vol. 20, no. 10, 2014.

[95] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, “Hindsight experience replay,” in *Advances in Neural Information Processing Systems*, pp. 5048–5058, 2017.

[96] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, “Curriculum learning,” in *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, 2009.

[97] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. K. Liu, “Dart: Dynamic animation and robotics toolkit,” *The Journal of Open Source Software*, vol. 3, no. 22, 2018.

[98] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, “Tensorflow: A system for large-scale machine learning,” in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283, 2016.

[99] Z. Yin and K. Yin, “Linear time stable pd controllers for physics-based character animation,” in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 191–200, 2020.

[100] Adobe Systems Incs., “Mixamo,” 2018.

[101] CMU, "Cmu graphics lab motion capture database," 2002.

[102] N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation," in *Proceedings of IEEE International Conference on Evolutionary Computation*, pp. 312–317, 1996.

[103] J. Lee, "Representing rotations and orientations in geometric computing," *IEEE Computer Graphics and Applications*, vol. 28, no. 2, pp. 75–83, 2008.

[104] C. R. Taylor, K. Schmidt-Nielsen, and J. L. Raab, "Scaling of energetic cost of running to body size in mammals," *American Journal of Physiology-Legacy Content*, vol. 219, no. 4, 1970.

[105] V. A. Tucker, "Energetic cost of locomotion in animals," *Comparative Biochemistry and Physiology*, vol. 34, no. 4, 1970.

[106] C. T. Farley and C. R. Taylor, "A mechanical trigger for the trot-gallop transition in horses," *Science*, vol. 253, no. 5017, 1991.

[107] A. Zhao, J. Xu, M. Konaković Luković, J. Hughes, A. Speilberg, D. Rus, and W. Matusik, "Robogrammar: Graph grammar for terrain-optimized robot design," *ACM Transactions on Graphics*, vol. 39, no. 6, 2020.

[108] S. Xia, C. Wang, J. Chai, and J. Hodgins, "Realtime style transfer for unlabeled heterogeneous human motion," *ACM Transactions on Graphics*, vol. 34, no. 4, 2015.

[109] F. G. Harvey, M. Yurick, D. Nowrouzezahrai, and C. Pal, "Robust motion in-betweening," vol. 39, no. 4, 2020.

[110] X. B. Peng, Y. Guo, L. Halper, S. Levine, and S. Fidler, "ASE: Large-scale reusable adversarial skill embeddings for physically simulated characters," *ACM Transactions on Graphics*, vol. 41, no. 4, 2022.

[111] J. Won, D. Gopinath, and J. Hodgins, "Physics-based character controllers using conditional vaes," *ACM Transactions on Graphics*, vol. 41, no. 4, 2022.

요약

가상의 캐릭터가 실제 사람처럼 움직이게 하려면 풍부한 모션 데이터 레퍼토리가 필요하다. 모션 데이터를 효율적으로 획득하기 위해 새로운 조건을 만족시키도록 소스 모션을 수정하는 모션 편집 알고리즘에 대한 연구가 많이 진행되어 왔다. 본 연구의 목적은 물리 기반의 모션 편집을 통해 다양한 모션을 생성하는 것이다. 물리 기반의 캐릭터 애니메이션은 물리 환경에서 캐릭터를 움직여 동작을 생성하는 방식이다. 모션 편집과 물리 기반 애니메이션을 결합하면 물리적으로 시뮬레이션된 캐릭터가 원본 모션에서 벗어나는 새로운 조건의 움직임을 만들어낼 수 있다는 장점이 있다. 물리 기반 모션 편집의 핵심 과제는 불필요한 방향으로의 변형을 억제하면서 바람직한 방향으로의 변형을 장려하는 알고리즘을 설계하는 것이다. 우리는 모션 최적화와 물리 시뮬레이션을 결합하여 생성된 모션의 품질과 다양성을 크게 향상시켰다.

이 논문은 다양한 조건에 대한 동작을 편집하는 세 가지 물리 기반 접근 방식을 제안한다. 첫 번째로 하나의 모션 클립에서 넓은 조건 공간을 만족하는 모션 집합을 생성하는 알고리즘을 소개한다. 동작 매개변수화와 동작 추적을 동시에 학습해서 생성되는 동작의 성능과 시각적 품질을 크게 향상했다. 두 번째로 여러 캐릭터와 모션 클립을 결합하여 새로운 캐릭터의 애니메이션을 만드는 알고리즘을 제안한다. 우리의 알고리즘은 캐릭터와 동작의 물리적 특성을 고려하여 각 부분 동작들의 적절한 공간적 및 시간적 정렬을 찾는다. 마지막으로, 암시적 조건인 스타일을 위한 모션 합성에 대해 논의한다. 우리의 알고리즘은 소스 동작으로부터 스타일 특징을 직접적으로 추출하고 물리 환경에서 스타일 특징을 교정하여 적은 수의 동작 데이터에서의 스타일 전송을 가능하게 한다. 우리의 연구는 새로운 조건, 신체 구조 및 스타일에 대한 매우 역동적인 동작의 생성을 통해 물리 기반 모션 편집 알고리즘의 다양성을 보여주었다.

주요어: 컴퓨터 애니메이션, 물리 시뮬레이션, 데이터 기반 애니메이션, 심층 강화 학습, 동작 합성, 동작 편집
학번: 2017-25468