Physics-based Augmentation with Motion Generation and Motion Imitation for Humanoid Terrain Traversal

by

Michael Xu

B.A.Sc., University of Toronto, 2020

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Masters of Science

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Fall 2025

Declaration of Committee

Name:	**Michael Xu**
Degree:	**Masters of Science**
Thesis title:	**Physics-based Augmentation with Motion Generation and Motion Imitation for Humanoid Terrain Traversal**
Committee:	**Dr. Xue Bin Peng** Supervisor Assistant Professor, Computing Science
	Dr. Mo Chen Committee Member Associate Professor, Computing Science
	Dr. Angel Xuan Chang Examiner Associate Professor, Computing Science
	Dr. Hang Ma Chair Associate Professor, Computing Science

Abstract

Humans excel in navigating diverse, complex environments with agile motor skills, exemplified by parkour practitioners performing dynamic maneuvers, such as climbing up walls and jumping across gaps. Reproducing these agile movements with simulated characters remains challenging, in part due to the scarcity of motion capture data for agile terrain traversal behaviors and the high cost of acquiring such data.

This thesis introduces PARC (**P**hysics-based **A**ugmentation with **R**einforcement Learning for Character **C**ontrollers), a framework that combines generative modeling and physics-based simulation to iteratively augment motion datasets and expand the capabilities of terrain-traversal controllers. Our framework, begins by training a motion generator on a small dataset consisting of core terrain traversal skills. The motion generator is then used to produce synthetic data for traversing new terrains. However, these generated motions often exhibit artifacts, such as incorrect contacts or discontinuities. To correct these artifacts, we train a physics-based tracking controller to imitate the motions in simulation. The corrected motions are then added to the dataset, which is used to continue training the motion generator in the next iteration. This iterative process jointly expands the capabilities of the motion generator and tracker, creating agile and versatile models for interacting with complex environments. PARC provides an effective approach to develop controllers for agile terrain traversal, which bridges the gap between the scarcity of motion data and the need for versatile character controllers.

Keywords: Physics-based Character Control, Generative Modeling

Dedication

To my family, for supporting me on this journey.

Acknowledgements

First and foremost, I wish to thank Professor Xue Bin Peng for taking me on as his student and guiding me throughout this journey. I am deeply grateful for the opportunity to enter this exciting field under one of its most influential researchers. Professor Peng was patient with my progress and generously supported my pursuit of a challenging project I was passionate about. He devoted countless hours to teaching me reinforcement learning and character animation, brainstorming ideas, debugging code, and refining our papers and presentations. I would not be where I am today without his guidance and belief in my potential.

I am also deeply grateful to Professor KangKang Yin for her generous support, for her insightful expertise on parkour motions, and for motivating and organizing our visits to the motion capture studio, which greatly enriched this work. My sincere thanks go as well to Yi Shi for his valuable collaboration on PARC, for his meticulous efforts in identifying and resolving numerous issues in the codebase, and for producing the high-quality animations that elevated the presentation of this project.

Finally, I would like to thank all my fellow lab members and friends at Simon Fraser University, the friends I made in Switzerland, and the friends I remained close to from Toronto. The time we spent together, including our hiking trips and our many discussions about research, personal goals, and long-term aspirations, made my graduate school experience significantly more meaningful and memorable.

Above all, I want to thank my family for their unwavering support. I took a risk in pursuing graduate school, but my parents believed in me while providing a safety net to fall back on. My sister has been a constant source of comfort and strength, always there to listen, to ground me, and to remind me of what truly matters in life. And of course, I can't forget Rocky, who is unequivocally the best boy in the world.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Humans possess the remarkable ability to navigate through diverse and complex environments by employing a broad range of agile motor skills. A prime example of human agility can be seen in parkour practitioners, who regularly demonstrate extraordinary athleticism by stylishly traversing obstacles using dynamic combinations of maneuvers, such as vaulting, climbing, and jumping. Developing simulated characters that can achieve comparable versatility remains a significant challenge. Current state-of-the-art methods for training physics-based controllers predominantly rely on motion capture data, using imitation objectives to guide the learning process towards natural and human-like behaviors. Due to the challenges of capturing highly athletic interactions with complex environments, there is little data available of human athletes traversing diverse terrains with agile motor skills.

While capturing large volumes of motion data from human athletes is prohibitively expensive, it is often feasible to record a small number of high-quality demonstrations that showcase core terrain-traversal skills. If this limited dataset could be effectively augmented, it would dramatically expand the diversity of motions available for training. Generative models offer a natural mechanism for such augmentation: by learning the underlying distribution of the seed motions, they can synthesize new trajectories that enrich the dataset's spatiotemporal and combinatorial variety. A straightforward strategy is to first train a motion generator on the small initial dataset, then use it to automatically produce additional synthetic motions. These synthesized trajectories can be incorporated back into training, enabling the generator to progressively improve its ability to handle increasingly challenging and diverse environments. Repeating this *self-consuming* loop iteratively can, in principle, produce a dataset large enough to train a general and robust terrain-traversal controller.

However, this approach faces a key challenge: motion generation models are susceptible to producing low-quality or physically implausible motions. These include issues such as incorrect contacts, floating, sliding, or temporal discontinuities. These issues are exacerbated when the dataset is small and lacks combinatorial or spatiotemporal variations. If left unaddressed, these artifacts accumulate over iterations and degrade the generator's performance rather than improving it. To prevent this compounding failure mode, we intro-

duce a physics-based correction step within each iteration. Specifically, we train a motion imitation controller to track and physically correct the generator's synthetic motions. Instead of adding the raw generated trajectories to the dataset, we add only the physically corrected versions, thereby improving realism, preventing drift, and stabilizing the iterative data-expansion process.

Building on this idea, we introduce PARC (Physics-based Augmentation with Reinforcement Learning for Character Controllers), a framework that takes a small initial motion dataset as input and outputs a motion tracking controller for traversing complex terrains. PARC iteratively trains a motion generator and motion tracker to augment a motion dataset, progressively expanding the capabilities of both models. The motion generator synthesizes new terrain traversal motions, while the motion tracker refines them to ensure physical plausibility before adding the motion clips to the dataset. This expanded dataset is used to continuously train both the generator and tracker, enhancing their versatility. Through multiple iterations, PARC develops an expressive motion generator and an agile motion-tracking controller. Together, these components enable precise control of a simulated character, allowing it to navigate complex obstacle-filled environments with agility.

1.1 Thesis Overview

This thesis presents a framework for training terrain traversal controllers via a self-consuming self-correcting training loop consisting of generative modeling and reinforcement learning components. Although much of this work was presented at SIGGRAPH '25 [79], this thesis offers expanded explanations of the technical and implementation details underlying PARC. Chapter 2 and Chapter 3 introduce the necessary background and prior work. Chapter 4 describes PARC in more detail with its data generation pipeline. Chapter 5 describes kinematic motion generator training and inference. Chapter 6 describes motion tracking controller training and recording simulated motions. Chapter 7 presents the experiments and results produced by PARC. Finally, Chapter 8 concludes this thesis and discusses the broader research direction of PARC.

1.2 PARC Method Overview

In this work, we present PARC (**P**hysics-based **A**ugmentation with **R**einforcement Learning for Character **C**ontrollers), an iterative data augmentation framework for training agile terrain traversal controllers for physically simulated characters. An overview of PARC is available in Figure 1.1. Our framework consists of two main components: a motion generator that generates kinematic motions given a target terrain, and a physics-based motion tracker that corrects artifacts in the generated motions by leveraging physical simulation. These two components are applied iteratively to augment the motion dataset, progressively expanding the versatility of the generator and capabilities of the tracker.

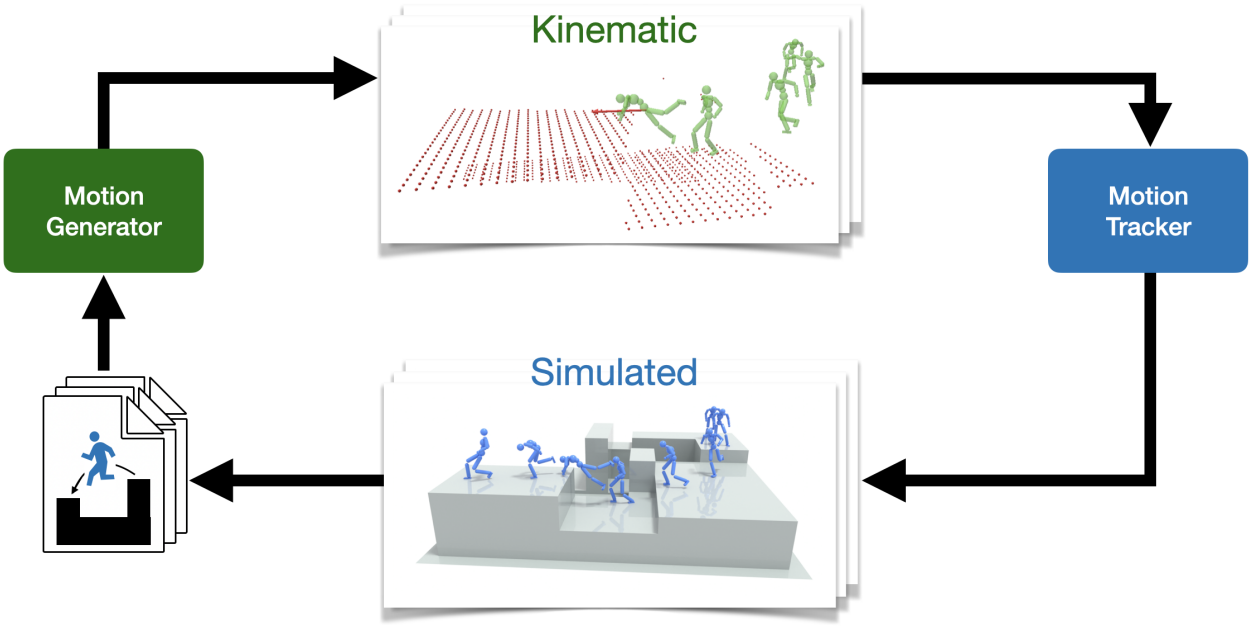

Figure 1.1: Overview of the PARC framework. PARC iteratively trains a motion generator and motion tracker with self-generated motion data. The motion generator produces kinematic motion sequences to train the motion tracker, while the motion tracker corrects physics-related artifacts in a simulator, enabling the motion generator to continue training on new physics-based motions.

PARC starts with a small initial motion-terrain dataset $\mathcal{D}^0$. At each iteration i, PARC trains a motion generation model G^i on the dataset $\mathcal{D}^{i-1}$ from the previous iteration. Next, G^i is used to synthesize new motions $\widetilde{\mathcal{M}}^i$ for traversing new terrains. After synthesizing new motions, a motion tracking policy π^i is trained to enable a simulated character to imitate the motions from the combined dataset $\mathcal{D}^{i-1} \cup \widetilde{\mathcal{M}}^i$. Once trained, π^i is used to record physically corrected motions $\mathcal{M}^i$ by tracking $\widetilde{\mathcal{M}}^i$ in simulation. Finally, the physically corrected motions are then added to the dataset $\mathcal{D}^i \leftarrow \mathcal{D}^{i-1} \cup \mathcal{M}^i$.

An important characteristic of PARC is that the models are trained in a continual manner. The generator G^i and policy π^i of each stage are initialized with the trained models from the previous stage, utilizing past experience to accelerate the learning of new motions. The final motion generation model can be used to synthesize target motions on new terrains, and the physics-based motion tracker can then follow the target motion to control a physically simulated character to traverse new environments. An implementation of PARC as well as the initial and generated datasets can be found at this link: https://github.com/mshoe/PARC.

1.3 Contributions

Michael Xu implemented the motion diffusion model training and inference framework, algorithms for retargeting motion capture data under the guidance of Prof. Peng, the motion tracking framework based on a DeepMimic implementation in Isaac Gym provided by Prof. Peng and with Prof. Peng's guidance, the rest of the data generation pipeline, and the codebase used for editing motions and terrains. Michael Xu also proposed the idea for training motion generators on their physically corrected outputs given user generated motions, and Prof. Peng advocated for automating this idea without needing a human in the loop.

Chapter 2

Background

In this section, we review core machine learning concepts for humanoid motion generation. First, we discuss diffusion models, the primary architecture used for the motion generator of the PARC system. Next, we cover reinforcement learning, the paradigm used to train the motion tracking controller for simulated characters in the PARC system.

2.1 Diffusion Models

Given a limited dataset $\mathcal{D}$ of ground truth data, can we generate samples from the underlying true data distribution? This is the primary goal of generative modeling, which has a rich history from variational autoencoders (VAE) [25], generative adversarial networks (GAN) [11], and recently, diffusion models [61, 14, 63]. In practice, generative models take samples from known probability distributions like the standard Gaussian distribution, and transform them into a samples from an unknown target data distribution. When going from the samples in the target distribution to the standard Gaussian distribution, a known diffusion process can be used such as a Markov noising process:

$$q(\mathbf{x}_k \mid \mathbf{x}_{k-1}) := \mathcal{N}(\mathbf{x}_k \; ; \; \sqrt{1-\beta_k}\mathbf{x}_{k-1}, \beta_k \mathbf{I}) \tag{2.1}$$

$$q(\mathbf{x}_{1:K} \mid \mathbf{x}_0) = \prod_{k=1}^{K} q(\mathbf{x}_k \mid \mathbf{x}_{k-1}) \tag{2.2}$$

This *forward* diffusion process q slowly corrupts a sample $\mathbf{x}_0 \sim \mathcal{D}$ into pure Gaussian noise by iteratively applying noise to the sample. The noise is applied by resampling from the Gaussian distribution with mean $\mu = \sqrt{1-\beta_k}\mathbf{x}_{k-1}$ and variance $\sigma = \beta_k \mathbf{I}$, following a noise schedule β_k. The noise schedule is designed such that at the final diffusion timestep K, $\beta_K = 1$ and the distribution $\mathbf{x}_K$ converges to a standard Gaussian distribution $\mathbf{x}_K \sim \mathcal{N}(0, \mathbf{I})$. Examples of such noise schedules are linear [14] and cosine [43].

An important property of the forward diffusion process, commonly known as the *nice property* [75], is the ability to sample $\mathbf{x}_k$ in a closed form without needing to sequentially sample via equation 2.1 for arbitrary timestep k. Let $\alpha_k = 1 - \beta_k$ and $\bar{\alpha}_k = \prod_{i=1}^{k} \alpha_i$. Then,

$$q(\mathbf{x}_k \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_k \; ; \; \sqrt{\bar{\alpha}_k}\mathbf{x}_0, \; (1 - \bar{\alpha}_k)\mathbf{I}) \tag{2.3}$$

Since we know a computationally cheap equation that is equivalent to the forward diffusion process, we can train a generator model G_θ to reconstruct the clean sample $\mathbf{x}_0$ from noisy samples $\mathbf{x}_k$ at various different noise levels k via supervised learning:

$$\mathcal{L}_{\text{rec}}(G_\theta) := \mathbb{E}_{\mathbf{x}_0, \, \mathcal{C} \sim D} \mathbb{E}_{k \sim p(k)} \mathbb{E}_{\mathbf{x}_k \sim q(\mathbf{x}_k | \mathbf{x}_0)} \left[||\mathbf{x}_0 - G_\theta(\mathbf{x}_k, \; k, \; \mathcal{C})||^2 \right], \tag{2.4}$$

where $p(k)$ represents the diffusion timestep distribution (e.g. uniform distribution between $[1, K]$), and $\mathcal{C}$ is a context associated with the sample $\mathbf{x}_0$ (e.g. text, control signal, etc.).

We note that the original DDPM formulation trains a denoising model to predict the noise ϵ_k applied to the noised data sample $\mathbf{x}_k$ [14], and derives a reverse diffusion process that iteratively predicts the noise to remove in order to reach the original sample $\mathbf{x}_0$. Many motion diffusion models choose to train a denoising model G_θ that directly predicts the denoised motion sample instead [68, 7], due to qualitatively better predicted motions. Please refer to [22] for a more detailed discussion between the two options.

Technically, G_θ could be used as a one-step denoising model, but this gives poor results in practice. Instead, what is typically done is using $G_\theta(\mathbf{x}_k, k, \mathcal{C})$ to obtain an approximate clean sample $\hat{\mathbf{x}}_0$, which is then noised again via the forward diffusion process to $\mathbf{x}_{k-1}$. This is the reverse diffusion step, which is repeated until timestep $k = 0$ is reached.

The reverse conditional probability is tractable when conditioned on $\mathbf{x}_0$, and has been previously derived [14, 75]:

$$q(\mathbf{x}_{k-1} | \mathbf{x}_k, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{k-1} \; ; \; \tilde{\mu}(\mathbf{x}_k, \mathbf{x}_0), \; \tilde{\beta}_k \mathbf{I}) \tag{2.5}$$

$$\text{where} \quad \tilde{\mu}_k(\mathbf{x}_k, \mathbf{x}_0) = \frac{\sqrt{\alpha_k}(1 - \bar{\alpha}_{k-1})}{1 - \bar{\alpha}_k}\mathbf{x}_k + \frac{\sqrt{\bar{\alpha}_{k-1}}\beta_k}{1 - \bar{\alpha}_k}\mathbf{x}_0 \quad \text{and} \quad \tilde{\beta}_k = \frac{1 - \bar{\alpha}_{k-1}}{1 - \bar{\alpha}_k}\beta_k \tag{2.6}$$

This is enough to derive the stochastic reverse diffusion process of DDPM [14]. Instead, we will derive a deterministic reverse diffusion process introduced in DDIM [62], which can be further extended to take longer strides in the reverse diffusion process which significantly speed up computation. We can rewrite $q(\mathbf{x}_{k-1} | \mathbf{x}_k, \mathbf{x}_0)$ such that it is parameterized by a desired standard deviation σ_k [62], and replacing $\mathbf{x}_0$ with our denoised sample $\hat{\mathbf{x}}_0 := G_\theta(\mathbf{x}_k, k, \mathcal{C})$:

$$q_\sigma(\mathbf{x}_{k-1} \mid \mathbf{x}_k, \hat{\mathbf{x}}_0) = \mathcal{N}(\mathbf{x}_{k-1} \; ; \; \sqrt{\bar{\alpha}_{k-1}}\hat{\mathbf{x}}_0 + \sqrt{1 - \bar{\alpha}_{k-1} - \sigma_k^2} \; \cdot \; \frac{\mathbf{x}_k - \sqrt{\bar{\alpha}_k}\hat{\mathbf{x}}_0}{\sqrt{1 - \bar{\alpha}_k}} \; , \; \sigma_k^2 \mathbf{I}) \tag{2.7}$$

This relates to Eq. 2.5 through $\sigma_k^2 := \tilde{\beta}_k$. The special case where $\sigma_k = 0$ results in the deterministic reverse diffusion process in DDIM [62], where $\mathbf{x}_{k-1}$ is a deterministic function of $\mathbf{x}_k$ and $\hat{\mathbf{x}}_0$.

$$\mathbf{x}_{k-1} = \sqrt{\bar{\alpha}_{k-1}}\hat{\mathbf{x}}_0 + \sqrt{1-\bar{\alpha}_{k-1}} \cdot \frac{\mathbf{x}_k - \sqrt{\bar{\alpha}_k}\hat{\mathbf{x}}_0}{\sqrt{1-\bar{\alpha}_k}} \tag{2.8}$$

$$= \left(\sqrt{\bar{\alpha}_{k-1}} - \frac{\sqrt{1-\bar{\alpha}_{k-1}}\sqrt{\bar{\alpha}_k}}{\sqrt{1-\bar{\alpha}_k}}\right)\hat{\mathbf{x}}_0 + \frac{\sqrt{1-\bar{\alpha}_{k-1}}}{\sqrt{1-\bar{\alpha}_k}}\mathbf{x}_k \tag{2.9}$$

Strided sampling can be used to speed up the reverse diffusion process further, and has been shown to give much better results when using DDIM instead of DDPM [62]. Given a stride d and an initial noise $x_K \sim \mathcal{N}(0, \mathbf{I})$, we can iteratively apply the following equation until we achieve the final predicted clean sample $\mathbf{x}_0$.

$$\mathbf{x}_{k-d} = \left(\sqrt{\bar{\alpha}_{k-d}} - \frac{\sqrt{\bar{\alpha}_k}\sqrt{1-\bar{\alpha}_{k-d}}}{\sqrt{1-\bar{\alpha}_k}}\right)\hat{\mathbf{x}}_0 + \frac{\sqrt{1-\bar{\alpha}_{k-d}}}{\sqrt{1-\bar{\alpha}_k}}\mathbf{x}_k \tag{2.10}$$

We've now described a supervised learning method for training diffusion models G_θ (Eq. 2.4), and a deterministic reverse diffusion process for generating samples from the target data distribution (Eq. 2.10).

2.2 Reinforcement Learning

Reinforcement learning [66] has been an effective paradigm for training controllers for a wide range of tasks ranging from games to locomotion [39, 60, 47, 44, 72]. In reinforcement learning, an agent interacts with its environment according to a policy π to maximize an objective. At each time step t, the agent observes the state of the environment s_t. The agent then samples and executes an action from a policy $\mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t)$. The next state $\mathbf{s}_{t+1}$ is then determined by the environment dynamics $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$. After every state transition, the agent receives a scalar reward determined by a reward function $r_t = r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$. The return G_t at time t with discount factor $\gamma \in [0, 1]$ is defined as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.11}$$

An important term in reinforcement learning is the value function $V_\pi(\mathbf{s})$, which intuitively measures how valuable a given state $\mathbf{s}$ is under the policy π. This value is quantified by the expected discounted return $V(\mathbf{s}) = \mathbb{E}_\pi\left[G_t|\mathbf{s}_t = \mathbf{s}\right]$. Likewise, the action-value can be defined as $Q_\pi(\mathbf{s}, \mathbf{a}) = \mathbb{E}_\pi\left[G_t|\mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}\right]$. Finally, the advantage function $A_\pi(\mathbf{s}, \mathbf{a})$ determines how much more valuable an action $\mathbf{a}$ is compared to other actions at a given state $\mathbf{s}$: $A_\pi(\mathbf{s}, \mathbf{a}) = Q_\pi(\mathbf{s}, \mathbf{a}) - V_\pi(\mathbf{s})$.

The goal of reinforcement learning is to find a policy which maximizes the expected discounted return $J(\pi)$ under the initial state distribution $s_0 \sim p(\mathbf{s}_0)$, defined as:

$$J(\pi) = \mathbb{E}_{\mathbf{s}_0 \sim p(\mathbf{s}_0)} V_\pi(\mathbf{s}_0), \tag{2.12}$$

Let π_θ be the policy defined by parameters θ. If the policy $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, environment dynamics $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, and reward function $r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$ then Eq. 2.12 could be optimized directly with respect to the policy parameters θ. However, in most cases the environment dynamics are not differentiable. Policy gradient algorithms for reinforcement learning circumvent this issue by defining methods to compute approximate policy gradients without requiring differentiable environment dynamics. Proximal Policy Optimization (PPO) [57] is a widely used policy gradient method which has become a standard baseline for many continuous control tasks [49, 8]. PPO optimizes a clipped objective function,

$$J^{\text{CLIP}}(\theta) = \mathbb{E}_\pi \Big[\min\big(r(\theta)\hat{A},\ \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}\big) \Big], \tag{2.13}$$

where $r(\theta) = \frac{\pi_\theta(\mathbf{a}|\mathbf{s})}{\pi_{\theta_{\text{old}}}(\mathbf{a}|\mathbf{s})}$ is the probability ratio between the new and old policies, and $\hat{A}$ is an estimator of the advantage function on the old policy $A_{\pi_{\theta_{\text{old}}}}(s, a)$. The intuition for the term $r(\theta)\hat{A}$ is to increase the probability of taking the action $\mathbf{a}$ at state $\mathbf{s}$ when that action has positive advantage. When the advantage is negative, the action is not good and descending by the policy gradient should decrease the probability of taking action $\mathbf{a}$ and state $\mathbf{s}$. The clipping term constrains the new policy to stay within an approximates trust region, which is the region in policy space that is safe to stably update the policy. The clipped objective is differentiable with respect to θ when the advantage estimator is differentiable. The standard method for estimating advantage is to use Generalized Advantage Estimate (GAE) [56] with an actor-critic framework, where the critic is a learned model which estimates the state value $V_\phi(\mathbf{s}_t)$.

Chapter 3

Related Work

Recent advances in machine learning have led to a surge of techniques that are capable of automatically producing high-fidelity human motions, which span both kinematics-based methods [18, 65, 54] and physics-based methods [49, 52, 51, 9, 81]. While differing in methodology, most of these methods leverage a corpus of motion data that captures natural human behaviors. These data-driven methods have been successful in developing models for a large variety of tasks, including text-to-motion generation [68, 91, 21, 82], music-to-dance [3, 69], character control [33, 64], and character-scene interactions [13, 67]. However, generating highly dynamic character-scene interaction behaviors remains a persistent challenge due to limited data availability. The constraints imposed by complex scenes also place more stringent demands on motion quality, as they are susceptible to more pronounced artifacts, such as floating and terrain collisions. In this section, we review the most relevant prior work on generating humanoid motion with scene constraints.

3.1 Kinematic Motion Generation Models

Given abundant, high-quality motion data, kinematic motion generation models can effectively synthesize complex human behaviors. Early data-driven approaches modeled motion–scene interactions by constructing graph structures that stitched motion clips together into coherent sequences [26, 28, 30]. More recently, deep learning–based methods have become increasingly popular. For example, [18, 17] introduced a phase-conditioned autoregressive model capable of generating realistic locomotion behaviors over irregular terrain. [83, 32] leveraged the expressiveness of diffusion models to synthesize scene-aware motions. While these works have shown promising results, they often struggle to generalize to new scenarios not captured in the original training dataset. Furthermore, their lack of physics-based simulation often leads to artifacts such as floating, ground penetration, and self-collision, compromising the realism of the synthesized motions.

3.2 Physics-based Humanoid Control

Physics-based humanoid character simulation has been explored as a means to procedurally generate novel behaviors that are compliant to physically accurate environments. For example, to model human athletic skills, existing works have developed physics-based controllers capable of replicating a wide range of dynamic sports, including parkour [35], tennis [89], table tennis [73], soccer [78], boxing [76], basketball [74, 34], climbing [41], and various Olympic sports [37]. The majority of these methods build off of seminal motion imitation works such as DeepMimic [49] and AMP [52]. Motion imitation requires high-quality motion data, .

Methods that combine physics-based and kinematic methods can leverage the advantages of these two paradigms to further enhance motion quality, diversity, and generalization, while also mitigating artifacts that violate physical principles. [5, 46] employed motion generators as a kinematic planners and trained a physics-based controllers to track the planner's motions. [20] utilized generative motion priors and projective dynamics for natural character behaviors. [85] applied physics-based tracking to refine motion diffusion outputs. These approaches rely on high-quality motion datasets for training their kinematic and tracking models. In contrast, our method iteratively trains a motion generator and tracker using an initially small dataset, expanded with physics-corrected motions.

3.3 Motion Control for Terrain Traversal

Developing motor controllers capable of agile traversal across complex terrains has been an active area of research spanning multiple fields, from robotics to computer graphics. [35] trained controllers specialized for traversing different types of obstacles and then employed manually designed planners to sequence these skills, enabling the traversal of sequences of different obstacles. [50] proposed a hierarchical reinforcement learning framework for training controllers capable of complex locomotion tasks such as ball dribbling across terrains, trail following, and obstacle avoidance. [90] presented a framework for evolving both the morphology and control for terrain traversal robots using evolutionary algorithms. [84] presented an algorithm that produces a control policy and scene arrangement to imitate dynamic terrain-traversal motions from video.

In the field of robotics, there is a large body of work that applied reinforcement learning methods to train controllers that enable legged robots to traverse through environments with obstacles using agile locomotion skills [16, 87, 88, 92]. While some techniques can generate highly dynamic locomotion skills without relying on demonstrations or reference motion data, the resulting controllers are prone to producing unnatural behaviors and require significant reward engineering that isn't currently scalable to a large and diverse set of behaviors.

3.4 Data Augmentation

Data augmentation has been an essential technique to prevent overfitting and improve generalization since the advent of the deep learning [27, 58]. Data augmentation has also been a vital tool in many character animation frameworks [46, 18]. To generate motions on irregular terrain, [18] applied a terrain fitting method to automatically fit mocap clips to appropriate terrain meshes. [19] use IK and smoothing on parametrized objects to create spatial variations of human-scene interactions. [80] use an interaction mesh [42] to create spatial variations of robot-objection interactions. Our method may for data augmentation is related to [46], where motion data was enriched using random walks on a motion graph [29], enriching spatial and temporal variations using Laplacian editing [23]. However, our method uses motion diffusion models instead of motion graphs, and instead of performing random walks, we generate motions conditioned on randomly generated terrain and paths. Finally, our method applies data augmentation in a self-consuming generative model training loop, where the outputs of our motion generator are used to continue training it.

More recently, researchers have explored self-consuming generative models as a more powerful methodology to automate data augmentation. [10] showed that self-consuming motion diffusion models experience mode collapse, unless a self-correcting function is used. To mitigate model collapse when training on self generated data, [10] incorporated a pre-trained physically simulated motion tracking controller [36] as a correction function for physically implausible motion artifacts. While the previous work uses physics-based motion trackers as correction functions, our work integrates the motion tracker as a component of the self-consuming loop by continually training the tracker to correct the synthetic motions produced by the motion generator. Our work also applies the self-consuming, self-correcting loop to the challenging task of terrain traversal.

Chapter 4

Data Generation Pipeline

PARC employs a motion-terrain data generation pipeline (Figure 4.1) to synthesize new, physically plausible motions that continuously enrich and diversify the motion dataset. As this dataset expands, both the motion generator and the motion tracker become increasingly expressive and capable. The pipeline begins with a terrain generation module (Section 4.5), which proposes new procedurally generated terrains as the foundation for motion synthesis. A path planning module (Section 4.6) then constructs feasible traversal paths on these terrains. Next, the motion generator G^i (Chapter 5) which was trained on $\mathcal{D}^{i-1}$ is used to autoregressively generate candidate motion sequences along each planned path. These motions are subsequently refined and curated using a kinematic heuristic model (Section 5.3) to filter out low-quality or physically implausible samples, leaving us with $\widetilde{\mathcal{M}}^i$. Finally, the motion tracking module (Chapter 6) trains a physics-based controller to reproduce the generated motions $\widetilde{M}^i$ in simulation while preserving the ability to track motions from the existing dataset $\mathcal{D}^{i-1}$. In PARC, this pipeline is used each iteration i to add approximately 1000 physics-based motions $\mathcal{M}^i$ to the dataset: $\mathcal{D}^i \leftarrow \mathcal{D}^{i-1} \cup \mathcal{M}^i$. Pseudocode for the PARC loop with the data generation pipeline is provided in Algorithm 1.

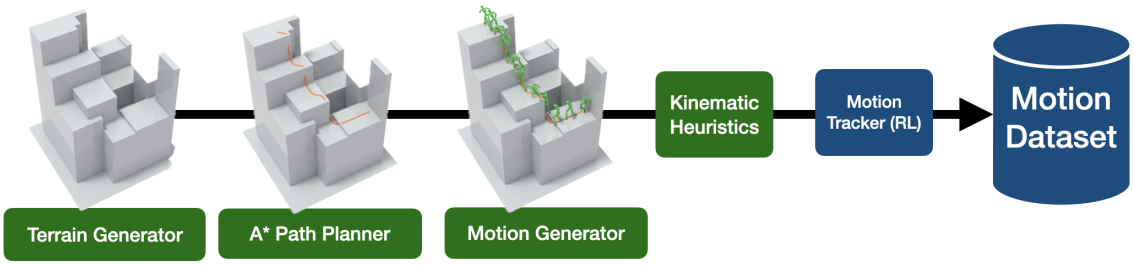

Figure 4.1: An overview of our motion-terrain data generation pipeline. On the kinematic side, this pipeline uses a terrain generation module, a path planning module, a trained motion generation model, and a kinematic heuristics module. Finally, to ensure physical realism, a motion tracker module physically corrects motions before they get added to the dataset.

Algorithm 1: PARC with Data-Generation Pipeline

Input: Initial dataset $\mathcal{D}^0$, initial motion generator G^0, initial motion tracker π^0, simulator $\mathcal{S}$

Output: Final dataset $\mathcal{D}^N$ after N iterations

for $i = 1$ **to** N **do**

- `// Train motion generator on current dataset`
- $G^i \leftarrow \text{TrainMotionGenerator}(G^{i-1}, \mathcal{D}^{i-1})$;
- $N_{\text{new motions}} \leftarrow \text{DecideNumberOfNewMotions}()$;
- $\widetilde{\mathcal{M}}^i \leftarrow \emptyset$;
- **foreach** $j = 1$ **to** $N_{\text{new motions}}$ **do**
  - $\mathbf{T} \leftarrow \text{GenerateTerrain}()$; `// Sec. 4.5`
  - $p \leftarrow \text{PlanPath}(\mathbf{T})$; `// Sec. 4.6`
  - `// Batched autoregressive motion generation (Chap. 5)`
  - $\mathbf{X} \leftarrow G^i(p, \mathcal{T})$;
  - `// Kinematic heuristics module (Sec. 5.3)`
  - $\mathbf{x}, \mathbf{T} \leftarrow \text{KinematicSelection}(\mathbf{X}, \mathcal{T})$;
  - $\mathbf{x} \leftarrow \text{KinematicOptimization}(\mathbf{x}, \mathbf{T})$;
  - $\widetilde{\mathcal{M}}^i \leftarrow \widetilde{\mathcal{M}}^i \cup \{\mathbf{x}, \mathbf{T}\}$;
- `// Physics-based motion tracking (Chap. 6)`
- $\pi^i \leftarrow \text{MotionTrackerRL}(\pi^{i-1}, \widetilde{\mathcal{M}}^i \cup \mathcal{D}^{i-1}, \mathcal{S})$;
- $\mathcal{M}^i \leftarrow \text{RecordMotions}(\pi^i, \mathcal{S}, \widetilde{\mathcal{M}}^i)$;
- `// 6. Update dataset`
- $\mathcal{D}^i \leftarrow \mathcal{D}^{i-1} \cup \mathcal{M}^i$;

return $\mathcal{D}^N$;

In the rest of this section, we will describe how motions and terrains are represented, motion-consistent terrain augmentation methods useful for training motion generators, then simple augmentation methods for producing spatial variations of motion-terrain pairs, and finally the algorithms used in the terrain generation and path planning modules.

4.1 Motion Representation

We use the character model from MimicKit [48], which is 1.62m tall humanoid composed of simple sphere, box, and capsule primitives. The humanoid has both spherical and hinge joints, and some bodies such as the pelvis and torso consist of multiple shapes. A visualization of the humanoid in its rest-pose can be seen in Figure 4.2.

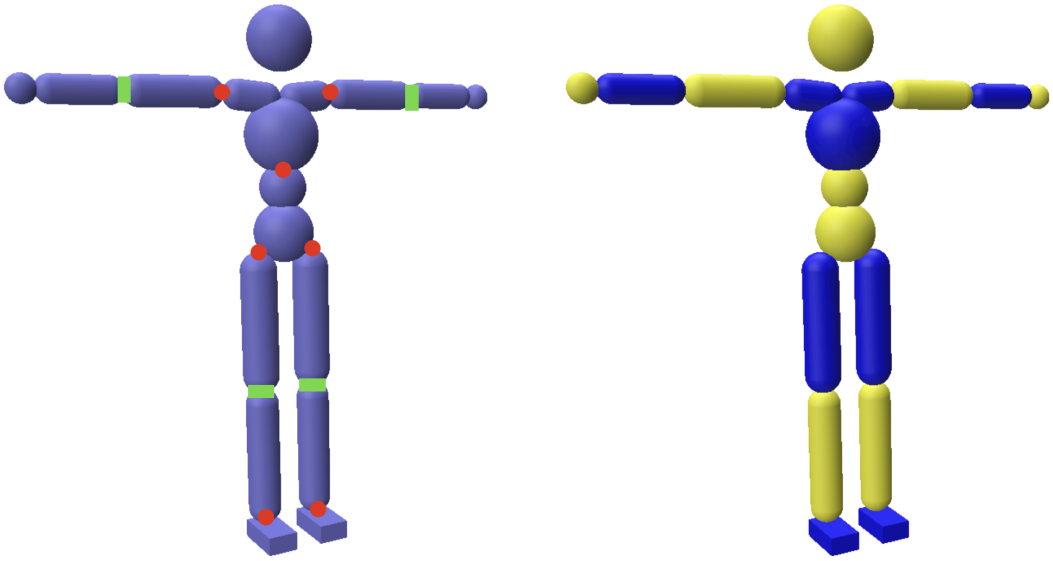

Figure 4.2: (Left) The humanoid model with spherical joints marked with red spheres, and hinge joints marked with green rectangles. (Right) The humanoid model with contact bodies colored blue and yellow, visualized with no adjacently linked body parts having the same color for clarity. The pelvis and torso are composed of multiple shapes.

A motion frame $\mathbf{x}^i$ is represented as a set of features consisting of the root position $\mathbf{p}^0 \in \mathbb{R}^3$, the root rotation $\mathbf{q}^0 \in \mathbb{R}^3$, the joint rotations $\mathbf{q} \in \mathbb{R}^{J\times 3}$, and the contact labels $\mathbf{c} \in [0,1]^B$, where J and B denotes the number of joints and bodies in the character model, respectively. When a body's contact label is 1, it means the body is either in contact with the terrain or another body part. A motion sequence $\mathbf{x} = \{\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^N\}$ consists of sequential motion frames. An example of a motion sequence can be seen in Figure 4.4.

4.2 Terrain Representation

Our goal is to design a physics-based character controller capable of traversing diverse and complex terrain. This is however a very challenging task, and providing sufficient observations of the terrain to our character controller can be very memory intensive. Therefore, we apply some constraints to the terrain in PARC to simplify the task while still being able to achieve impressive results. These result in a block style 2.5D terrain reminiscent of terrains in the popular computer game Minecraft.

Our terrains are represented as 2.5D grids. Each grid index i, j corresponds to a box centered at a terrain point $(x_0 + i\Delta x, y_0 + j\Delta y)$, where (x_0, y_0) represent the minimum box center coordinates, and $\Delta x, \Delta y$ denote the box dimensions. In particular, we use $\Delta x = \Delta y = 0.4$m. The top surface of the box is located at $\mathbf{h}(i, j)$ meters, while the bottom surface effectively extends to $-\infty$ since our terrains are 2.5D. A terrain $\mathbf{T}$ can therefore be

fully represented with (x_0, y_0), Δx, Δy, and $\mathbf{h}(i, j)$. This representation is converted to a triangle mesh for physical simulation using a naive meshing algorithm that converts each face of the grid cell boxes into two triangles, while making sure to merge redundant vertices. An example of a terrain is shown in Figure 4.3.

4.3 Motion-Consistent Terrain Augmentation

Given a motion sequence and its associated terrain, we can alter the terrain without interfering with the motion, augmenting our dataset with more possible terrain observations. We use a simple a method by placing random boxes on the terrain. In order for these boxes to make physical sense with the motion, we use a non-interfering augmented heightmap condition. Given the original motion sequence, it is possible to calculate upper and lower height bounds for each grid cell of the terrain heightmap such that the terrain does not intersect with any frame of the motion sequence. Then, after augmenting the terrain with randomly placed boxes, we clamp the terrain heights to be within the precomputed bounds. When augmenting local terrain heightmaps for training the motion generator, we apply an approximate non-interfering augmented heightmap condition by assuming the local heightmap corresponds to a terrain geometry. An example of what these non-interfering terrain augmentations look like on a motion clip can be seen in Figure 4.4.

4.4 Motion-Terrain Spatial Augmentation

A dataset containing only a handful of demonstrations per skill often lacks the spatial diversity needed to train a capable motion generator. Although the PARC loop does not require a perfectly comprehensive dataset at initialization, it does benefit from one with sufficient spatial variation to enable the generator to produce motions that generalize to slight terrain variations. To address this issue, we designed a simple algorithm for augmenting motion-terrain pairs.

Given an input motion-terrain pair, we either randomly vary the height of the main obstacle in the terrain for parkour skills, or randomly add small boxes along the motion's trajectory in the terrain. To further enrich spatial diversity, we also apply small random rotations to the root trajectory around the origin. We also randomly stretch or squish motions along a random axis on the horizontal plane. Then, we apply kinematic optimization (described in section 5.3.3) to fix contact and terrain penetration issues that arise modifying the terrain. To improve the physical plausibility of the adjusted motions, we also train a physics-based motion tracking controller (described in chapter 6) on augmented motions, and then record those motions to get physically corrected motions. The simulation environment is able to detect and label accurate contact labels automatically. Instead of training a tracking controller for each motion individually, our motion tracker can be trained to track them all together, saving compute time since behaviors it learns for tracking one

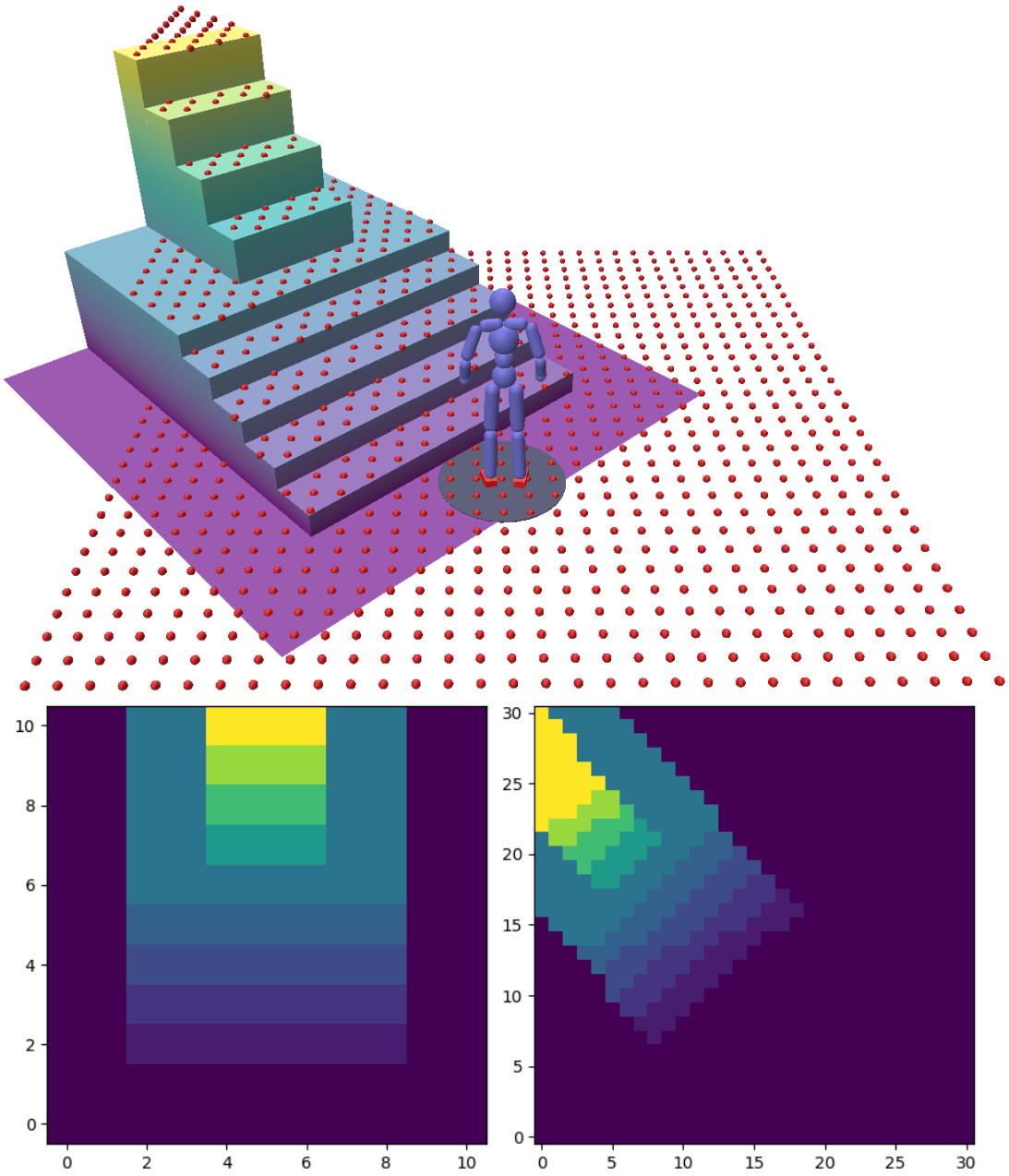

Figure 4.3: (Top) An example of a terrain used in our experiments, as well as a character standing on the terrain. The character's local heightmap is visualized as red points. (Bottom left) The global heightmap of the terrain. (Bottom right) The local heightmap **h** that is input to the motion generator.

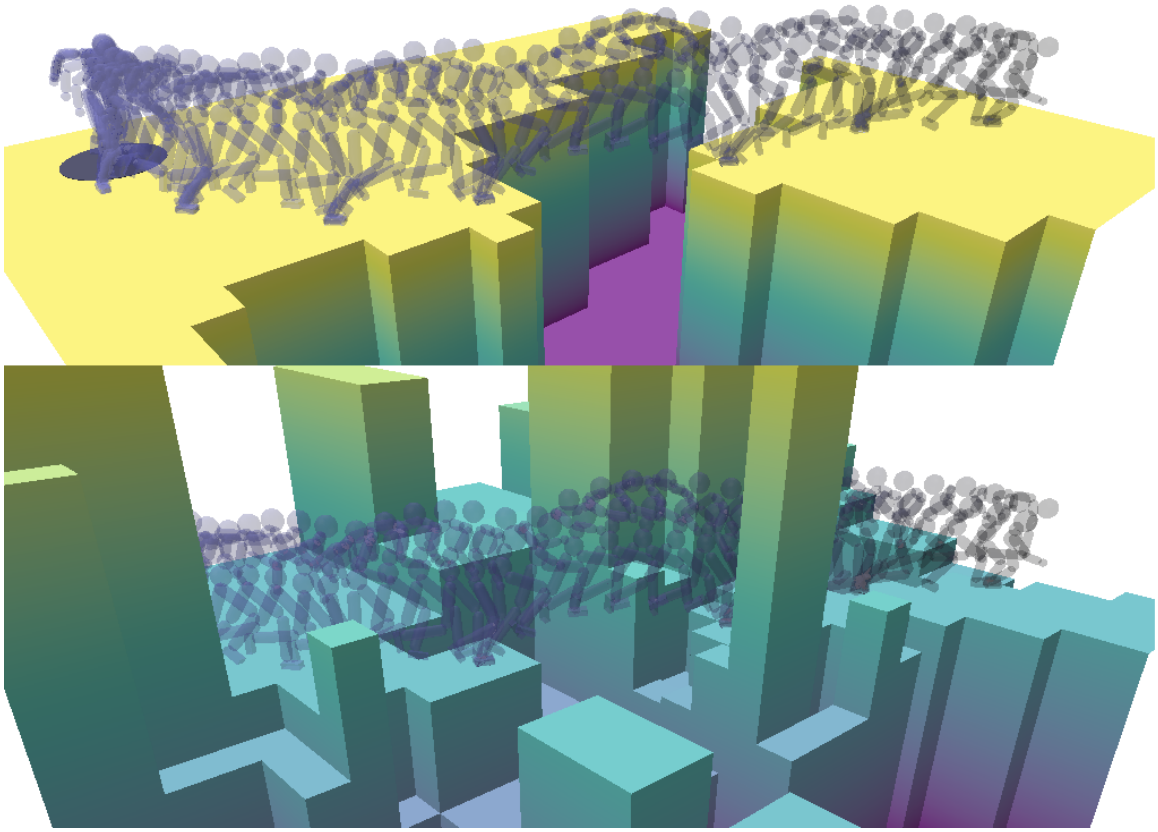

Figure 4.4: (Top) A running and jumping motion sequence as well as its associated terrain. (Bottom) The terrain augmented motion, where the terrain does not interfere with the original motion trajectory.

Algorithm 2: Motion–Terrain Spatial Augmentation

Input: Dataset $\mathcal{D}$ of motion-terrain pairs and labels describing motion type, simulator $\mathcal{S}$
Output: Augmented dataset $\widetilde{\mathcal{D}}$
$\widehat{\mathcal{D}} \leftarrow \emptyset$;
foreach *(*motion $\mathbf{x}$*,* terrain $\mathbf{T}$*,* motion type*)* $\in \mathcal{D}$ **do**
  $\widetilde{\mathbf{T}} \leftarrow$ TerrainModification($\mathbf{T}$, motion type);
  $\mathbf{x} \leftarrow$ RandomRotation($\mathbf{x}$);
  $\mathbf{x} \leftarrow$ RandomStretchOrSquish($\mathbf{x}$);
  $\widehat{\mathbf{x}} \leftarrow$ MotionOptimization($\mathbf{x}, \widetilde{\mathbf{T}}$); `// Sec. 5.3.3`
  $\widehat{\mathcal{D}} \leftarrow \widehat{\mathcal{D}} \cup \{\widehat{\mathbf{x}}, \widetilde{\mathbf{T}}\}$;
$\pi \leftarrow$ MotionTrackerRL($\widehat{\mathcal{D}}, \mathcal{S}$); `// Chapter 6`
$\widetilde{D} \leftarrow$ RecordMotions($\pi, \mathcal{S}, \widehat{\mathcal{D}}$);
return $\widetilde{\mathcal{D}}$

motion-terrain pair can be transferred to track other motion-terrain pairs. Pseudocode for this augmentation method is described in Algorithm 2.

4.5 Terrain Generation

We employ a set of simple but effective terrain generation algorithms to propose challenging terrains for our data-generation pipeline.

4.5.1 Random Boxes

Given a terrain grid of size NxM, the Random Boxes method sequentially generates B boxes with random centers, widths, lengths, and heights. For generating new terrains in iteration 1 and 2 of PARC, we apply the Random Boxes method to a flat 16x16 grid, with box widths and lengths within 5-10 grid cells, and box heights within -2m to 2m. To simplify the terrain and make it easier to generate motions on, we ensure there are no thin 1 block gaps or walls. We do this by using a sliding 2x2 window with a stride of 2 to flatten grid cells within the window. The flattening is done by setting grid cells within a window to the maximum of height within the window. An example of the Random Boxes method for generating a new terrain can be seen in Figure 4.5.

4.5.2 Random Walk Terrain Generation Algorithm

The random walk terrain generation algorithm is used to generate our test terrains. We do this to better measure the performance of our motion generator and motion tracker on terrains generated by algorithms different from the generation algorithms used for training. This algorithm generates paths with random walks on a flat input terrain. The height of a

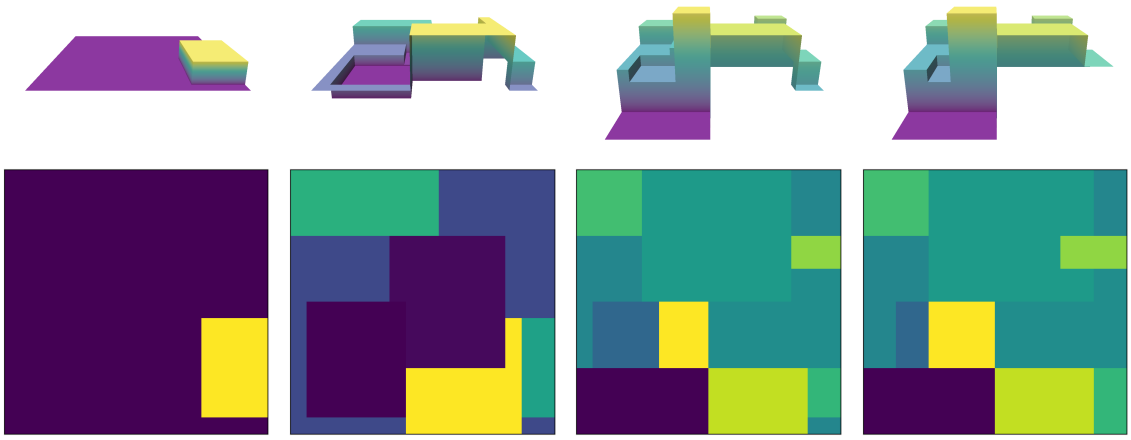

Figure 4.5: An illustration of our Random Boxes terrain generation method. From left to right: we begin with a single randomly generated box on a flat terrain. Next, we show terrains with five and ten randomly generated boxes, respectively, where each new box is added cumulatively. In the final step, we apply a sliding 2×2 max-pooling operation to the terrain heightmap, which removes narrow corridors and small platforms—producing smoother, more navigable terrain for the motion generator.

path is randomly determined. For our 100 randomly generated test terrains, we initialized 32x32 flat terrains, then generated 10 random walk paths with random heights.

4.5.3 Random Terrain Slices

Given a large terrain, we can randomly select small slices of the terrain to run our path planner and motion generator on. We use a 100x100 manually designed terrain, shown in Figure 4.6, and select random 16x16 slices to generate new motions on.

4.6 Path Planning

We use a custom A* [12] algorithm to search for suitable paths between start and end points on terrains. These paths are later used to provide target directions to our motion generator. In order to apply A* to our task, we need to define the weighted graph using the terrain grid. We also need to define a cost function for traversing the edges of the graph.

4.6.1 Navigation Graph

The graph for A* search is constructed as follows:

- Each cell has a directed edge connecting it to it's adjacent cells that are within an allowable height difference. We set the maximum difference to be 2.1 meters, which is slightly higher than the maximum height differences in the original dataset.

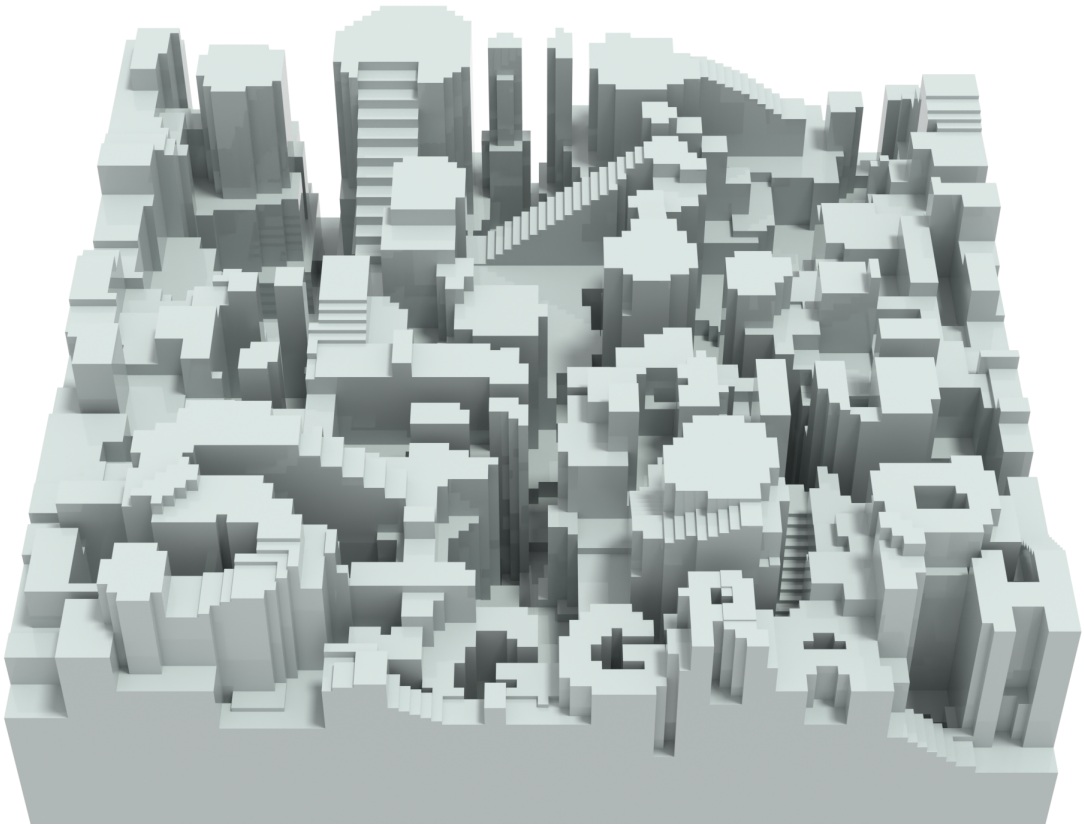

Figure 4.6: A 100x100 terrain first generated with a variant of the random boxes algorithm, then manually sculpted to add finer details. This terrain was used to help generate motions and terrains for the third and fourth iteration of our experiment using the random terrain slices method.

- To enable jumping over gaps, for each node we search for nodes within a certain jump radius and within a min and max jump height for each cell. If a node that satisfies the jump conditions exist, we add a directed jump edge.
- We also make sure not to add jump edges that intersect with walls by computing line box intersections for each potential jump edge.

A visualization of a terrain and it's navigation graph can be seen in Figure 4.7. A clear example of how jump edges interact with walls can be seen in Figure 4.8.

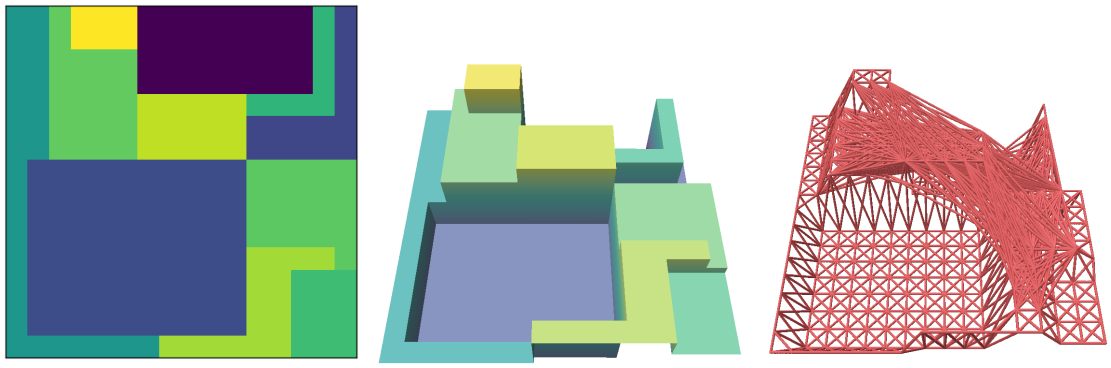

Figure 4.7: (Left) An example terrain heightmap grid. (Middle) The terrain's 3D visualization. (Right) The terrain's navigation graph, which is used for A* path planning.

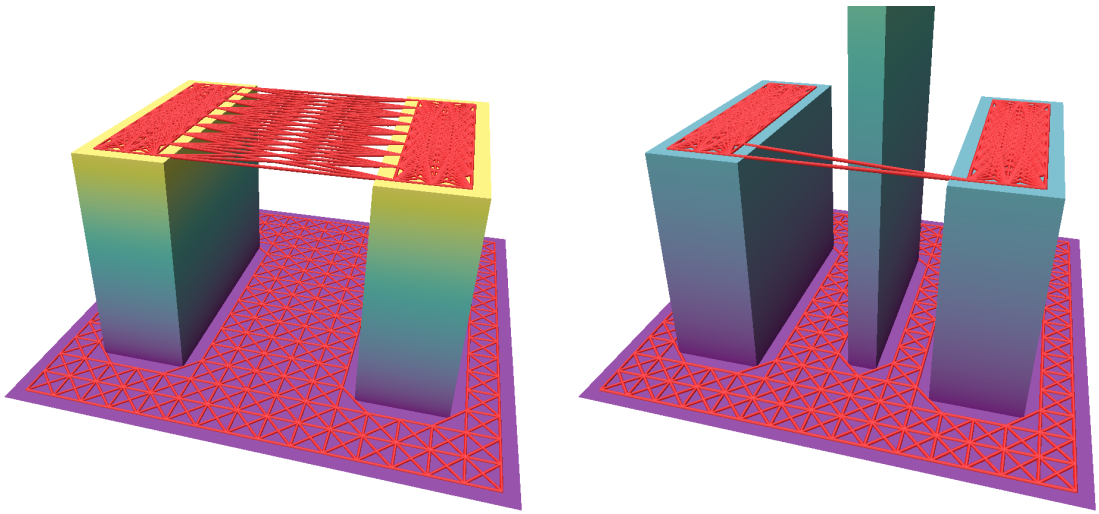

Figure 4.8: A visualization of the navigation graph with jump edges. (LEFT) We connect edges between cliff nodes within a jump radius, which are determined by having adjacent nodes that are at a much lower height. This creates connections in the graph that allows the path planner to jump across platforms. (RIGHT) Our navigation graph does not allow jump edges when a wall is interfering with the jump trajectory.

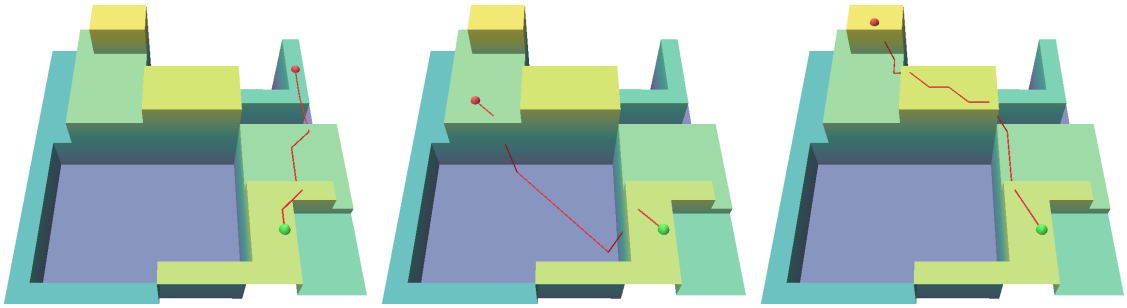

Figure 4.9: Examples of paths generated by our custom A* path planner.

4.6.2 Cost

The cost to move along an edge is a function of the horizontal and vertical distance between the source node and potential target node. The horizontal and vertical distances can be weighted differently. In order to allow for more diversity in the generated paths, we also add a stochastic value to the cost function. In total, the cost function g to move between the current node $\mathbf{x}_1 = (x_1, y_1, z_1)$ and a potential next node $\mathbf{x}_2 = (x_2, y_2, z_2)$ is:

$$g(\mathbf{x_1}, \mathbf{x_2}) = w_{xy}((x_1 - x_2)^2 + (y_1 - y_2)^2) + w_z(z_1 - z_2)^2 + X, \tag{4.1}$$

where $X \sim U(c_{\min}, c_{\max})$ is a random variable to add stochasticity to the cost function. In our implementation, $w_{xy} = 1$, $w_z = 0.15$, $c_{\min} = 0$, $c_{\max} = 0.5$. The reason we use a higher horizontal distance weight is to prioritize short horizontal paths, therefore encouraging our path planner to make use of vertical terrain traversal skills such as climbing.

4.6.3 Path Generation on New Terrains

To generate a candidate path for a new motion, we begin by randomly sampling start and end nodes near the terrain boundaries and running our custom A* planner to connect them. If A* fails to find a feasible path, we resample the start and end nodes and retry. In the rare case where no valid path is discovered after multiple attempts, we simply discard the terrain and procedurally generate a new one. An example of a generated path on a terrain can be shown in Figure 4.9.

Chapter 5

Kinematic Motion Models

The motion generator is one of the main components of PARC's self-augmentation loop, and acts as a planner that generates kinematic motions for traversing a given terrain. The motion generator is represented as a diffusion model [14] trained to generate motion sequences while conditioned on a local terrain heightmap and target direction. We believe other generative models may have their tradeoffs compared to diffusion models, so the techniques in this section can be applied to other models too. An overview of our network architecture is shown in Figure 5.1.

Given an input context $\mathcal{C}$, the motion generator predicts a motion sequence $\mathbf{x}$ consisting of motion frames $\{\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^N\}$ for traversing the terrain. These motion frames consist of the same features as described in section 4.1, as well as body positions. The input context $\mathcal{C}$ to the diffusion model consists of a heightmap $\mathbf{h}$, recorded in the character's local coordinate frame, the horizontal target direction $\mathbf{d} \in \mathbb{R}^2$, and the first two frames of the motion sequence. These first two frames are an optional condition, allowing our model to both generate an initial motion sequence given no previous frames and also generate long-horizon motion sequences autoregressively. Conditioning on two input frames, instead of one, provides the model with velocity information.

The diffusion model is implemented with a transformer encoder architecture, similar to MDM [68]. An illustration of the model architecture is available in Figure 5.1. The generator G receives as input the context, $\mathcal{C} = \{\mathbf{h}, \mathbf{d}, \mathbf{x}^1, \mathbf{x}^2\}$, the noisy motion frames $\mathbf{x}_k$, and diffusion timestep k. The generator then predicts the clean motion sequence $\hat{\mathbf{x}}_0$,

$$G(k, \mathbf{x}_k, \mathcal{C}) = \hat{\mathbf{x}}_0 = \{\hat{\mathbf{x}}_0^1, \hat{\mathbf{x}}_0^2, \cdots, \hat{\mathbf{x}}_0^N\} \tag{5.1}$$

The input diffusion timestep k, heightmap $\mathbf{h}$, target direction $\mathbf{d}$, and noisy frames $\mathbf{x}_k$ are first encoded using different embedding networks to map each into tokens for the transformer. Before encoding, spherical joint and root rotations are represented as exponential maps, while hinge joint rotations are represented as angles. The heightmap observations are processed with a convolutional neural network, and the image patches are extracted and

processed into tokens by an MLP. The target direction is encoded with an MLP to produce one token, and each frame of the input motion sequence $\mathbf{x}_k$ is encoded into a token with an MLP. When the first two frames $(\mathbf{x}_0^1, \mathbf{x}_0^2)$ are given, they replace the frames $(\mathbf{x}_k^1, \mathbf{x}_k^2)$ in the output sequence, and are encoded with the same encoding network. Positional encoding is applied to all tokens. The output token sequence is passed through a final MLP that maps it to the denoised motion $\hat{\mathbf{x}}_0$.

5.1 Motion Data Sampling

The motion generator is trained to generate motions for traversing new terrains using a dataset of motion clips paired with their respective terrains. When a motion clip is sampled, a half-second motion sequence is selected uniformly from the frames of the motion clip. Figure 5.1 illustrates the features that are extracted from each sequence. The motion sequence is split into 13 future frames and 2 past frames. Each frame within a sequence is canonicalized relative to the second frame, which is treated as the most recent frame that the generator is conditioning on. A random future frame is used to determine the target direction.

The sampling of the local heightmap from the global terrain geometry associated with the motion clip is done using a 31×31 uniform grid of points, canonicalized to the second frame of the motion. The sampled heightmaps are augmented with randomly oriented boxes with varying heights to improve generalization when given out of distribution heightmap observations. A collision-avoidance technique is employed to ensure these boxes do not add terrain penetration to the motion, as detailed in Section 4.2.

5.2 Training

The motion generator is trained following the standard DDPM process illustrated in Eq. 2.4, but with additional geometric loss terms, which are introduced to better capture the spatial coherence and physical plausibility of generated motions. Training is done by iteratively sampling $\mathbf{x}_k$ for $k \sim [1, K]$, predicting $\hat{\mathbf{x}}_0$, and computing a diffusion model loss. The total diffusion model loss consists of a reconstruction loss $\mathcal{L}_{\text{rec}}$, a velocity loss $\mathcal{L}_{\text{velocity}}$, and a joint consistency loss $\mathcal{L}_{\text{joint}}$.

5.2.1 Training Loss

The simple reconstruction loss $\mathcal{L}_{\text{rec}}$ would compute arithmetic distances between the rotations, which is not a valid metric for rotation differences. Instead, we split the reconstruction loss into positional $\mathbf{p} = \{\mathbf{p}^{\text{root}}, \mathbf{p}^{1:J}\}^{1:N}$ and rotational $\mathbf{q} = \{\mathbf{q}^{\text{root}}, \mathbf{q}^{1:J}\}^{1:N}$ components to compute the reconstruction loss appropriately, where N is the number of motion frames and J is the number of joints. We also extract the contact label component $\mathbf{c} = \{\mathbf{c}^{1:J}\}^{1:N}$.

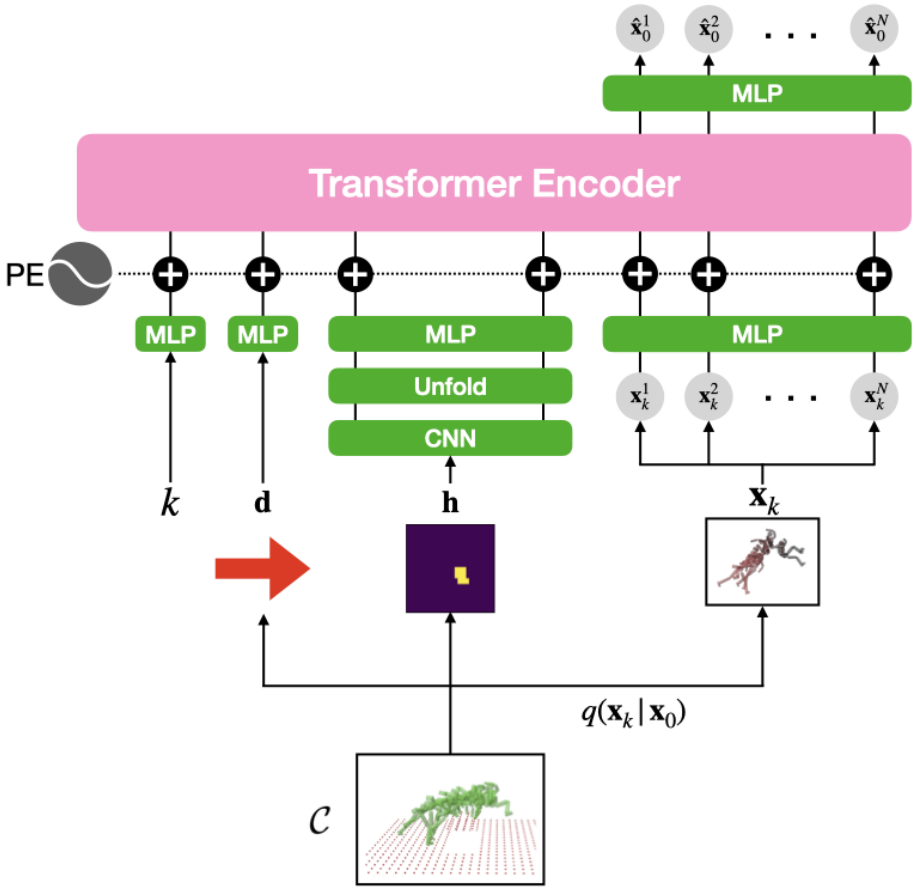

Figure 5.1: The transformer encoder based architecture of the terrain-conditioned motion generator. $\mathbf{h}$ is first processed by a CNN into an image of shape 64x16x16, then unfolded into 64 non-overlapping image patches of shape 64x2x2. The image patches are then embedded into tokens with an MLP. The target direction $\mathbf{d}$ is embedded into a single token with an MLP. Each frame of the noisy motion sequence $\mathbf{x}_k$ is embedded into a token using an MLP.

We denote $\hat{\mathbf{p}}_0$, $\hat{\mathbf{q}}_0$, and $\hat{\mathbf{c}}_0$ as the predicted positional component, rotational component, and contact labels extracted from $G(k, \mathbf{x}_k, \mathcal{C}) = \hat{\mathbf{x}}_0$, respectively. Thus, the new reconstruction loss is:

$$\mathcal{L}_{\text{rec}}(G) = \mathbb{E}_{\mathbf{x}_0,\, \mathcal{C}\sim D}\mathbb{E}_{k\sim p(k)}\mathbb{E}_{\mathbf{x}_k\sim q(\mathbf{x}_k|\mathbf{x}_0)}\Big[||\mathbf{p}_0 - \hat{\mathbf{p}}_0||^2 + ||\mathbf{q}_0 \ominus \hat{\mathbf{q}}_0||^2 + ||\mathbf{c}_0 - \hat{\mathbf{c}}_0||^2\Big] \tag{5.2}$$

The velocity loss helps with ensuring generating motions have physically realistic smoothness. The positional velocities $\dot{\mathbf{p}}$ are calculated using finite differences on the positional components of the motion frames. The angular velocities $\dot{\mathbf{q}}$ are calculated by first calculating quaternion finite differences, then converting them to exponential map form.

$$\mathcal{L}_{\text{velocity}}(G) = \mathbb{E}_{\mathbf{x}_0,\, \mathcal{C}\sim D}\mathbb{E}_{k\sim p(k)}\mathbb{E}_{\mathbf{x}_k\sim q(\mathbf{x}_k|\mathbf{x}_0)}\Big[||\dot{\mathbf{p}}_0 - \hat{\dot{\mathbf{p}}}_0||^2 + ||\dot{\mathbf{q}}_0 - \hat{\dot{\mathbf{q}}}_0||^2\Big] \tag{5.3}$$

The joint position consistency loss intends to connect the predicted joint positions and the joint positions computed using a forward kinematics function denoted as $FK(\cdot)$ on root position, root rotation, and joint rotations extracted from $\hat{\mathbf{x}}_0$.

$$\mathcal{L}_{\text{joint}}(G) = \mathbb{E}_{\mathbf{x}_0,\, \mathcal{C}\sim D}\mathbb{E}_{k\sim p(k)}\mathbb{E}_{\mathbf{x}_k\sim q(\mathbf{x}_k|\mathbf{x}_0)}\left[||\hat{\mathbf{p}}_0 - \text{FK}(\hat{\mathbf{x}}_0)||^2\right] \tag{5.4}$$

The total training loss is given by:

$$\mathcal{L}(G) = \mathcal{L}_{\text{rec}}(G) + \mathcal{L}_{\text{velocity}}(G) + \mathcal{L}_{\text{joint}}(G) \tag{5.5}$$

5.2.2 Terrain-Aware Motion Generation

One of the core challenges for training a terrain-conditioned motion generator is ensuring the generated motions respect the surrounding terrain, either by not penetrating the terrain, or by employing physically plausible motor skills to interact with the terrain. When trained only on the small initial dataset $\mathcal{D}^0$, G tends to produce motions that ignore the physical constraints imposed by the terrain such as running into a wall instead of climbing over it. We hypothesize that when generating a motion autoregressively, G is overfitting to the previous frames as a consequence of using a small dataset. This focus on the previous frame results in the motion generator ignoring the terrain condition, leading to motions that fail the comply with the surrounding terrain.

Incorporating terrain-penetration loss during training encourages the motion generator to better adhere to the surrounding terrain. However, we found that results could still be improved by using an additional technique to further enhance terrain compliance. From our hypothesis that motion-terrain penetration is a result of overfitting to previous frames, we blend the output of our model when conditioned with and without the previous frames. This approach is similar to classifier-free guidance [15], and provides a tradeoff between temporally smooth motions with respect to the previous frames, and terrain-compliant motions.

Motions with smoothness artifacts can be corrected by the physics-based motion tracking controller, while motions that severely violate terrain constraints, such as running through a wall, cannot be reproduced in simulation. The blended denoising update is determined by:

$$\begin{aligned} &G_{\text{blend}}\left(k, \mathbf{x}_k, \mathcal{C} = \left(\mathbf{h}, \mathbf{d}, \mathbf{x}_0^1, \mathbf{x}_0^2\right)\right) \\ &= sG\left(k, \mathbf{x}_k, \mathcal{C} = (\mathbf{h}, \mathbf{d})\right) + (1-s)G\left(k, \mathbf{x}_k, \mathcal{C} = \left(\mathbf{h}, \mathbf{d}, \mathbf{x}_0^1, \mathbf{x}_0^2\right)\right), \end{aligned} \tag{5.6}$$

where s is the blending coefficient. We found that $s = 0.65$ works well. To facilitate this blending during inference, $G(k, \mathbf{x}_k, \mathcal{C} = \{\mathbf{h}, \mathbf{d}\})$ is trained simultaneously with $G(k, \mathbf{x}_k, \mathcal{C} = \{\mathbf{h}, \mathbf{d}, \mathbf{x}_0^1, \mathbf{x}_0^2\})$ by randomly masking the attention to the two previous frame tokens $\mathbf{x}_0^1$ and $\mathbf{x}_0^2$ with a 15% chance. At test time, evaluating the unconditional and conditional models is done by supplying the corresponding attention mask to the inputs.

5.3 Kinematic Heuristics

Given a terrain and a planned path, our trained model autoregressively generates long motion sequences by conditioning on its previously generated outputs. Although motion tracking can correct many physics-based inconsistencies in a reference motion, severely degraded motions remain difficult or impossible to track. To mitigate such failure cases, we incorporate two techniques aimed at reducing artifacts in the generated sequences.

First, motions are generated in batches of 64 candidate sequences, and a heuristic scoring function is used to select the most promising sample within each batch. This heuristic combines a contact loss, a terrain penetration loss, and a path incompletion penalty to favor motions that are both feasible and physically plausible. Second, the selected motion undergoes a kinematic refinement stage, where optimization techniques are applied to reduce jitter, correct body–terrain penetrations, and improve contact consistency.

5.3.1 Heuristic Losses

Our kinematic selection and optimization techniques require heuristic losses to be defined first. We approximate geometric losses by computing signed distance functions between points sampled on the surface of the character and the terrain. Due to the way our terrain is constructed, it can be represented as a signed distance field. Each cell (i, j) can be represented with a box centered at $(x, y) = (x_0 + i\Delta x, y_0 + j\Delta y)$ and with a top surface at $\mathbf{h}(i, j)$. We call the signed distance function to the terrain sdTerrain, which consists of the union of sdBox [53] for each box in the terrain.

We approximate distance computations between the character body and terrain by using points sampled on the surface of the character, and then use sdTerrain to get the signed distances between the surface sampled points and the terrain.

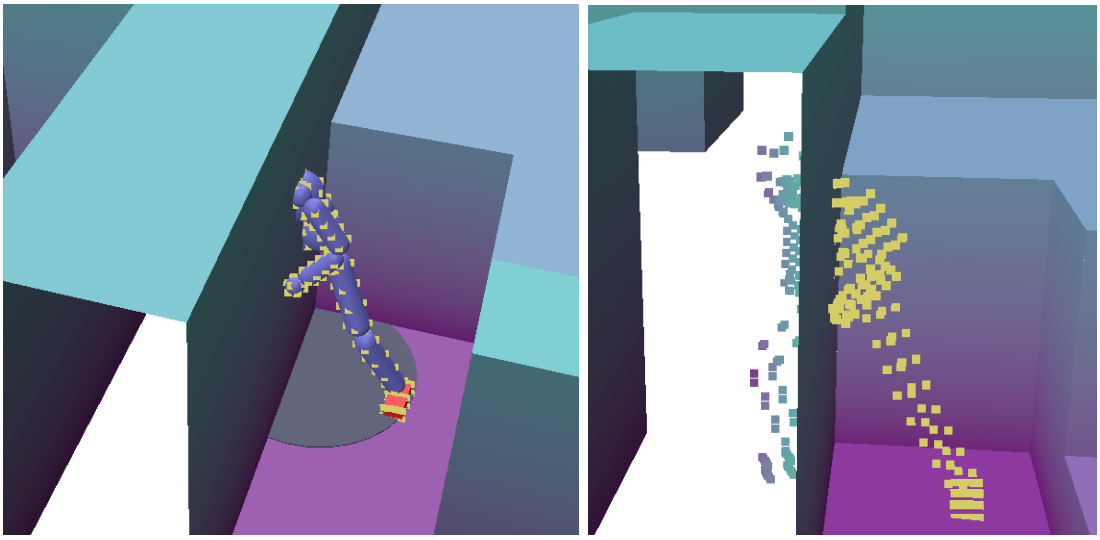

Figure 5.2: Visualizations of the approximate terrain penetration distances using points sampled on the surface of the character. The points are colored with a viridis color map, where darker represents more penetration.

Terrain Penetration Loss

Let $\mathbf{p}_i$ denote the i^{th} point sampled on the body of the character. Then the terrain penetration loss is formally:

$$\mathcal{L}_{\text{pen}} = \sum_{i=1}^{N_{\text{points}}} -\min(\text{sdTerrain}(\mathbf{p}_i), 0) \tag{5.7}$$

A visualization of the signed distances $\text{sdTerrain}(\mathbf{p}_i)$ on the points sampled on the character's body surface can be seen in Figure 5.2.

Terrain Contact Loss

We use the contact labels to determine when a body part is supposed to be in contact with the terrain. If a body is supposed to be in contact, then the distance between at least one of the sampled points on the body part and the terrain should be 0. Let $P(b)$ denote the set of points sampled on the surface of the b^{th} body part. Let $\mathbf{c}_b$ denote the binary contact label for the b^{th} body part. Then the contact loss for one motion frame is:

$$\mathcal{L}_{\text{contact}} = \sum_{b=1}^{N_{\text{bodies}}} \mathbf{c}_b \min_{\mathbf{p} \in P(b)} |\text{sdTerrain}(\mathbf{p})| \tag{5.8}$$

Jerk Loss

We compute the third derivative of joint position, which is joint jerk, using finite differences. We then add a penalty when a frame has a jerk magnitude greater than a specified maximum

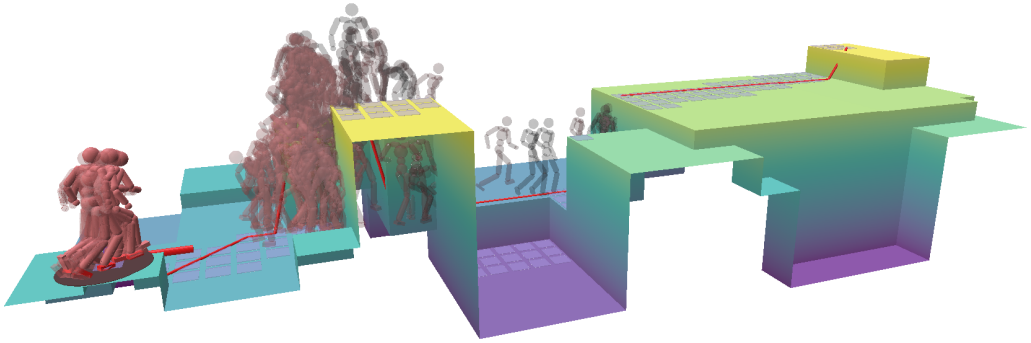

Figure 5.3: Example of 32 motions generated under the same terrain and path conditions, visualized at one frame per second. The sequences exhibit substantial variability in quality, with the most noticeable issues being terrain penetration and hesitation when attempting to climb over the first block.

value. Let $\mathbf{p}_b$ denote the position of the b^{th} joint, and $\text{jerk}_{\max}$ denote the max jerk we want to allow, then the jerk loss is:

$$\mathcal{L}_{\text{jerk}} = \sum_{b=1}^{N_{\text{joints}}} \max(|\dddot{\mathbf{p}}_b| - \text{jerk}_{\max}, 0) \tag{5.9}$$

5.3.2 Selection Heuristic

Motion generators can exhibit substantial variability in output quality, especially when conditioned on unfamiliar or challenging terrains. These variations affect not only the visual plausibility of the motions but also their potential to be tracked in simulation via DeepMimic motion tracking. Figure 5.3 illustrates this phenomenon: 32 motion sequences generated under the same terrain and path conditions using a motion diffusion model trained on a small initial dataset display a wide range of motion quality.

Since attempting to track physically infeasible motions wastes significant compute, we generate a batch of candidate motions in parallel and score them using a simple selection heuristic to identify those most likely to be physically feasible under DeepMimic-style tracking. Beyond filtering out low-quality outputs, batching offers an additional advantage: it increases the chance of discovering rare, high-quality motions on terrains that might otherwise appear too challenging. Even if only 1 out of 100 generated sequences for a diffi-

cult terrain is physically reproducible, that single successful motion becomes an extremely valuable datapoint that can be fed back into the dataset through the PARC loop.

Our selection heuristic combines the terrain penetration loss in Equation 5.7 with the terrain contact loss in Equation 5.8. We also impose a path incompletion constraint to discard motions that fail to reach the end of the path within a user-specified time horizon. This constraint is implemented softly in the loss with a very large weight (e.g., 1000 when other loss terms are typically < 1). The full selection heuristic we use is:

$$\mathcal{L}_{\text{select}} = \mathcal{L}_{\text{pen}} + \mathcal{L}_{\text{contact}} + 1000 \cdot \mathbf{1}_{\text{reached end of path}} \tag{5.10}$$

In our framework, we use a batch size of 128 for generating motions on new terrains and paths. The two generated motions with the lowest $\mathcal{L}_{\text{select}}$ are added to the reference motion dataset to train our motion tracker. In some cases, a terrain and path may fail to produce any reasonably good motion. We use a maximum threshold for $\mathcal{L}_{\text{select}}$ to decide when a motion is reasonably good. If no generated motions are underneath this threshold, we regenerate the terrain and path, then try generating motions in batches again.

5.3.3 Kinematic Motion Optimization

Generated motions may exhibit artifacts such as inconsistent contact timing, jittering limb trajectories, root drift, or body penetrations with terrain. These artifacts make motion tracking more challenging and sometimes physically impossible. These artifacts are more apparent when generating motions on challenging out of distribution terrain. To address these artifacts, we optimize the generated motion sequences using a heuristic loss function designed to remove unnatural behaviours. This loss function consists of:

- A regularization loss $\mathcal{L}_{\text{reg}}$ to prevent the optimized motion sequence from changing its root position, root rotation, and joint rotations too much from the original motion sequence. We also regularize joint velocity, which is computed via finite differences of joint positions. This regularization loss simply computes the difference between the current optimized variables and the original source variables. We use this loss to prevent the optimized motion from being wildly different from the input motion.
- A terrain penetration loss that penalizes body-terrain penetration, as described in Equation 5.7. This helps ensure physical consistency.
- A terrain contact loss that penalizes missed body-terrain contact based on each motion frame's contact labels, as described in Equation 5.8. This helps encourage desired contact events.
- A jerk loss that penalizes body joint jerk greater than $1000 m/s^3$, as described in Equation 5.9. This qualitatively makes motions more visually appealing, while also improving simulator stability.

We optimize the full motion trajectory, including root position, root orientation, and joint rotations, while keeping the original contact labels fixed. The full loss that we optimize for generated motions before the motion tracking stage is:

$$\mathcal{L} = w_{\text{reg}}\mathcal{L}_{\text{reg}} + w_{\text{pen}}\mathcal{L}_{\text{pen}} + w_{\text{contact}}\mathcal{L}_{\text{contact}} + w_{\text{jerk}}\mathcal{L}_{\text{jerk}}. \tag{5.11}$$

The weights we use are $w_{\text{reg}} = 1$, $w_{\text{pen}} = 1000$, $w_{\text{contact}} = 1000$, and $w_{\text{jerk}} = 1000$. We optimize using Adam [24] with a step size of 0.001 and 3000 iterations. We implement this optimization with PyTorch, relying on autograd to produce the gradients. A typical 10 second clip takes about 1 minute to optimize on an A6000 GPU.

We note that this optimizations objective is not intended to produce animation-quality motions on its own. Instead, it serves to remove the most destabilizing artifacts so that the downstream physics-based motion tracking can converge reliably. Without this optimization, the motion tracker frequently fails due to early penetration or inconsistent contacts. After optimization, failure cases are significantly reduced. We notice qualitatively that the optimized generated motions can sometimes still look unnatural, but after being tracked in simulation they look more physically realistic.

Chapter 6

Physics-based Tracking Models

In this section, we detail the process for training a controller capable of executing the generated kinematic motion sequences in simulation to enable a physically simulated character to traverse complex environments. The character is controlled using a motion tracking controller, trained with reinforcement learning on the kinematically generated motions as well as the initial dataset. Our motion tracking controller follows the DeepMimic [49], based on the implementation in Mimickit [48] and adapted for motions with associated terrains. All physics simulations are performed using Isaac Gym [38].

6.1 Observation and Action Representation

The actions from the policy specifies the target orientations for PD controllers positioned at each joint. The policy is queried at $30Hz$, while the physics simulation is performed at $120Hz$. The observations of the agent consist of its proprioceptive state, its local terrain observations, and future target frames from the reference motion. The proprioceptive state consists of the agent's root position $\mathbf{p}^{\text{root}}$, root rotation $\mathbf{q}^{\text{root}}$, joint rotations $\mathbf{q}^{1:N_{\text{joints}}}$, joint positions $\mathbf{p}^{1:N_{\text{joints}}}$ and contact labels $\mathbf{c}^{1:N_{\text{joints}}}$. The local terrain observations are represented using a heightmap of points $\mathbf{h}$ sampled around the character. The target states $\mathbf{s}^{\text{ref}}$ are specified by the reference motion clip, and all features are canonicalized with respect to the character's local frame. The future reference frames visible to our policy are 1, 2, 3, 10, 20, and 30 frames into the future. The local frame of the character is defined with the origin at the root, the x-axis facing the root link's facing direction, and the z-axis aligned with the global up vector.

6.2 Training

The physics-based controller is trained with a DeepMimic-inspired motion-tracking objective, enabling it to navigate diverse terrains by replicating the kinematic motions generated by the motion generator. The tracking reward is designed to encourage the controller to

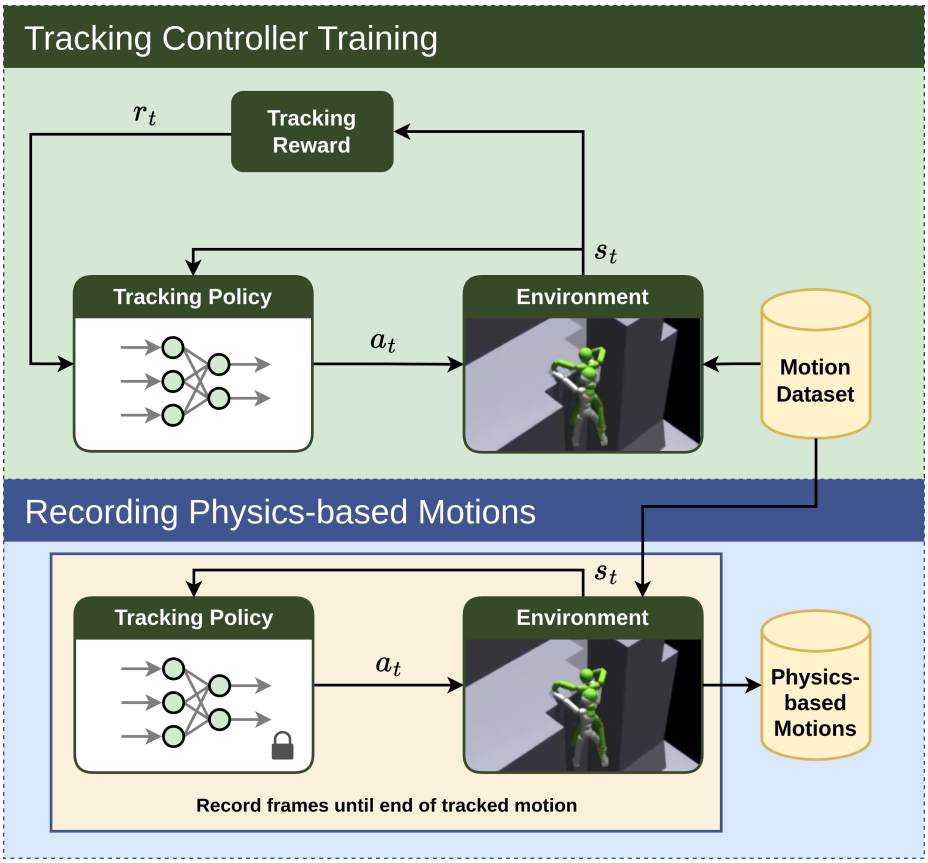

Figure 6.1: An overview of the reinforcement learning training method for the physics-based motion tracking controller, as well as the method for recording physics-based motions using the trained motion tracker.

minimize the differences between the agent and reference motion's root position, root velocity, joint rotations, joint velocities, key body positions, and contact labels. The contact label is a binary signal that specifies whether a body is in contact with the environment. We found that matching the reference motion's contacts is vital for ensuring the simulated character interacts with an environment using naturalistic contact configurations. The policy network is represented with three fully connected layers with 2048, 1024, and 512 units, and is trained using Proximal Policy Optimization [57], with advantages computed using GAE(λ) [55]. The value function is trained using target values computed with TD(λ) [66]. An overview of training the motion tracker and recording physics-based motions can be seen in Figure 6.1.

6.2.1 Rewards

We use a distance of 0.7m for the pose termination criteria of each joint, except for the foot joints which we do not use a pose termination criteria on. This is to give the motion tracking

Table 6.1: Individual weights w_j for each joint used in the joint rotation loss in Eq 6.1 and Eq 6.2.

Joint	**Weight**
abdomen	1.0
neck	0.6
right shoulder	0.6
right elbow	0.4
left shoulder	0.6
left elbow	0.4
right hip	1.0
right knee	0.6
right ankle	0.4
left hip	1.0
left knee	0.6
left ankle	0.4

agent more freedom in finding physically plausible solutions for tracking the kinematically generated motions.

The root position, root velocity, joint rotation, joint velocity, and key body position rewards are similar to [49]. Given the character and the reference motion's joint rotations at time t, $\mathbf{q}_t^j$ and $\hat{\mathbf{q}}_t^j$, the joint rotation or "pose" reward is:

$$r_t^{\text{pose}} = \exp\left[-0.25 \sum_j w_j ||\hat{\mathbf{q}}_t^j \ominus \mathbf{q}_t^j||^2\right] \tag{6.1}$$

where w_j is a tunable weight for the jth joint. The weights used can be found in table 6.1.

Given the character and reference motion's local joint velocities, $\dot{\mathbf{q}}_t^j$ and $\hat{\dot{\mathbf{q}}}_t^j$, the joint velocity reward is:

$$r_t^{\text{pose velocity}} = \exp\left[-0.01 \sum_j w_j ||\hat{\dot{\mathbf{q}}}_t^j - \dot{\mathbf{q}}_t^j||^2\right] \tag{6.2}$$

Given the character and reference motion's root position and root rotations, $\mathbf{p}_t^{\text{root}}$, $\mathbf{q}_t^{\text{root}}$ and $\hat{\mathbf{p}}_t^{\text{root}}$, $\hat{\mathbf{q}}_t^{\text{root}}$, the root position reward is:

$$r_t^{\text{root}} = \exp\left[-5\left(||\hat{\mathbf{p}}_t^{\text{root}} - \mathbf{p}_t^{\text{root}}||^2 + 0.1||\hat{\mathbf{q}}_t^{\text{root}} \ominus \mathbf{q}_t^{\text{root}}||^2\right)\right] \tag{6.3}$$

Given the character and reference motion's root velocity and root angular velocity, $\dot{\mathbf{p}}_t^{\text{root}}$, $\dot{\mathbf{q}}_t^{\text{root}}$ and $\hat{\dot{\mathbf{p}}}_t^{\text{root}}$, $\hat{\dot{\mathbf{q}}}_t^{\text{root}}$, the root velocity reward is:

$$r_t^{\text{root velocity}} = \exp\left[-\left(||\hat{\dot{\mathbf{p}}}_t^{\text{root}} - \dot{\mathbf{p}}_t^{\text{root}}||^2 + 0.1||\hat{\dot{\mathbf{q}}}_t^{\text{root}} - \dot{\mathbf{q}}_t^{\text{root}}||^2\right)\right] \tag{6.4}$$

Given the character and reference motion's key body positions, $\mathbf{p}_t^i$ and $\hat{\mathbf{p}}_t^i$, where i denotes the key body index, the key body reward is:

$$r_t^{\text{key}} = \exp\left[-10 \sum_i ||\hat{\mathbf{p}}_t^i - \mathbf{p}_t^i||^2\right] \tag{6.5}$$

The key bodies in our experiments are the hands and feet of the humanoid character.

The contact label reward is both a reward and a penalty. It penalizes the agent when it's contact labels do not match the reference motion, but also rewards the agent when it does. This is to prevent the agent from learning motions that use unnatural contacts. Given reference contact labels $\hat{\mathbf{c}}_t$ and simulator computed contact labels for the character $\mathbf{c}_t$, the contact reward is:

$$r_t^{\text{contact}} = \frac{1}{N_{\text{joints}}} \sum_j \left[\hat{\mathbf{c}}_t^j \mathbf{c}_t^j - (1 - \hat{\mathbf{c}}_t^j)\mathbf{c}_t^j\right] \tag{6.6}$$

The full tracking reward which is a weighted sum of the previously described rewards is:

$$r_t = 0.5r_t^{\text{pose}} + 0.1r_t^{\text{pose velocity}} + 0.15r_t^{\text{root}} + 0.1r_t^{\text{root velocity}} + 0.15r_t^{\text{key}} + r_t^{\text{contact}} \tag{6.7}$$

6.2.2 Prioritized State Initialization

Training a tracking controller on a large motion dataset, particularly one comprising numerous contact-rich parkour motions, presents a significant multi-task learning challenge for reinforcement learning (RL) agents. Sampling motion clips uniformly across the dataset often results in an imbalance, where challenging motion clips receive disproportionately fewer samples. This issue is exacerbated by the use of early termination, as difficult motions that consistently fail are further deprived of sampling opportunities, while easier motions with lower failure rates consume a disproportionate share of resources. In our framework, we track the failure rates of individual motion clips and incorporate these rates as sampling weights within a multinomial distribution. The sampling weights are used to sample a reference motion whenever an environment is reset. A motion clip is considered a failure if the agent fails to meet the pose termination criterion before completing the clip. By leveraging failure rates as sampling weights, PARC ensures that both challenging and unlearned motions receive proportionally more samples compared to easier or already mastered motions. To mitigate the risk of catastrophic forgetting, a minimum sampling weight of 0.01 is enforced for all motion clips, ensuring every motion continues to be sampled throughout training. Prior work uses variations of this technique known as prioritized state initialization [77, 46, 78, 67].

6.3 Physics-Based Motion Correction

Once the tracking controller is trained in each iteration, it is utilized to generate physics-corrected versions of the kinematically produced motions. These corrected motions, recorded within the simulation, effectively reduce physics-related artifacts present in the original kinematic motions. The successfully recorded motions are subsequently added to the motion dataset, which is then used to continue training the motion generator in the next iteration of PARC. A recorded motion is considered unsuccessful if the character fails to reach the end of the motion. Some reference motions may have large artifacts or challenging frames in their early frames, making it difficult to track. To help increase the output of successful recorded motions from our tracking controller, we initialize the character at different times along the motion, and then the earliest initialization that successfully reaches the final frame will be recorded as additional motion data.

Chapter 7

Experiments and Results

Our initial parkour dataset consists of new motion capture data for various terrain traversal skills such as vaulting, climbing, jumping, and running. 5.5 seconds of motion clips are also recorded from the Unreal Engine Game Animation Sample Project [70]. The corresponding terrains for each motion clip are manually reconstructed to fit each motion. The contact labels in the original dataset are manually labeled. The original dataset contains a total of 14 minutes and 7 seconds of motion data, depicting parkour skills such as climbing, vaulting, running on flat and bumpy ground, moving on and off platforms, and going up and down stairs. Examples of motions in the original dataset can be seen in Figure 7.1.

Since the original motion dataset is relatively small, there is a severe lack of spatial diversity, with most terrain-traversal skills being performed on terrains with fixed heights. Therefore, to improve spatial diversity, we use the augmentation method described in section 4.4 to generate 50 spatial variations of each motion clip in the original dataset, increasing the coverage of terrain variations for our input dataset to the PARC framework. The initial synthetic data expansion does not take advantage of the generative capabilities of the motion generator to discover new skills, and is only used to improve the motion generator in the initial iteration of PARC.

Given the initial dataset, the PARC framework is applied for three iterations using a single A6000 GPU, requiring approximately one month to complete. In the first two iterations, the motion generator is used to generate approximately 1000 new motions on randomly generated terrains. In both the third and fourth iterations, the motion generator is used to generate 2000 motions on a large manually designed terrain, more details of which are available in Section 4.2. All generated motions have a maximum length of 10 seconds. A visualization of the dataset expansion can be seen in Figure 7.2, where the net root horizontal displacement of every motion clip is plotted as a distribution. The initial dataset contained many forward-moving motions, but as training progressed, trajectories became more diverse, covering a wider range of directions.

7.1 Novel Behaviors

Our PARC framework can generate novel behaviors that go beyond those in the original dataset, enabling traversal of significantly more complex and diverse terrains. Examples of new behaviors generated by our framework are shown in Figure 7.3. The physically simulated character is able to sequence together different skills such as jumping across a gap and catching onto a ledge. Furthermore, the models can be used to generate significantly longer motions for traversing complex terrains. First, given a target path and a large terrain, the motion generator is used to autoregressively generate a kinematic target motion. Then, the motion tracker follows the target motion to traverse across the terrain. Examples of behaviors produced by the simulated character are shown in Figures 7.4, 7.5, 7.6, 7.7, and 7.8. The time required to generate each 0.5 seconds of motion with a batch size of 32 is approximately 12 seconds on an A6000 GPU. For the long-horizon examples, a batch size of 32 is used and heuristic criteria are applied to automatically select the best motion (see Section 5.3.2).

Table 7.1: Quantitative results of our motion generators across PARC iterations with the best values bolded. These metrics measure various aspects of motion quality, and include FWD (final waypoint distance), TPL (terrain penetration loss), TCL (terrain contact loss), and %HJF (percentage of high jerk frames). The motion generators from each iteration are used to generate 32 motions for each of the 100 test terrains. The average value across all 3200 generated test motions is reported for each metric.

Iteration	**FWD ↓**	**TPL ↓**	**TCL ↓**	**%HJF ↓**
1	1.908	2093	114.1	10.70
2	1.586	705.5	9.761	4.387
3	0.747	448.2	**8.070**	3.238
4	**0.596**	**179.6**	9.763	**2.730**
no physics correction	1.572	547.3	17.44	18.68

7.2 Motion Generator Performance

To evaluate the improvements to the motion generator from each PARC iteration, we conducted a quantitative experiment evaluating a large number of generated motions. We created 100 new test terrains and target paths using a procedural terrain generation algorithm different from the one used for the augmented dataset. Next, we applied various iterations of our PARC motion generator to produce 32 motions for each terrain, resulting in a total of 3200 motions for each PARC iteration. Blended denoising is applied with a coefficient of $s = 0.65$, and DDIM with a stride of 5 is used to generate motions along the target paths. No kinematic or physics-based motion corrections are applied to the generated kinematic motions. Four heuristic metrics are used to evaluate the quality of the generated motions:

final waypoint distance (FWD), terrain penetration loss (TPL), terrain contact loss (TCL), and percentage of high jerk frames (%HJF). The final waypoint distance measures the distance between the root position of the last generated frame and the target position at the end of a path. High jerk frames are determined as any frame where the jerk of any joint is greater than the maximum joint jerk observed in the original mocap dataset, which is approximately 11666 m/s^3. Table 7.1 and Figure 7.10 summarize the performance of the different models. The metrics exhibit significant improvements as the number of PARC iterations progresses. To demonstrate the importance of physics-based correction, we include an experiment where we trained a motion generator using uncorrected motions from the first PARC iteration (labeled "no correction" or NC). We use the selection heuristic (described in Section 5.3.2) to filter motions for terrain penetration and contact losses, with no other correction technique to highlight the importance of the physics-based motion tracking stage. We use as many training steps as PARC iteration 2 in the NC iteration.

Table 7.2: Quantitative results of our motion tracker for different PARC iterations. The success rate is the tracker's average rate of motion completion over 100 generated test motions. The joint tracking error is computed using an average of 2048 episodes for each of the 100 generated test motions at random initial timesteps.

Iteration	**Success Rate (%)** ↑	**Joint Tracking Error (m)** ↓
1	27	0.08294
2	44	0.05851
3	60	0.05321
4	**68**	**0.05167**

The motion generator trained with uncorrected motions produces significantly more jerk related artifacts, which can be seen in the %HJF metric. Qualitatively, these motions tended to include physically impossible feats, such as changing the trajectory of the character in the middle of a jump, which is likely due to the motion generator being trained on kinematically generated motions with no physics-based correction. Figure 7.9 compares the behavior of the motion generator from different iterations on a challenging example. The motion generator from the final iteration (Iteration 3) is able to generate higher quality and more physically realistic motions that successfully follow the target path across the environment.

7.3 Motion Tracker Performance

To evaluate the performance of the tracking controllers from different PARC iterations, we assess their success rates and average joint errors when tracking the test motions generated by their corresponding motion generator. The success rate is the percentage of episodes where the motion tracking controller is able to reach the final frame of motion when starting from the first frame. The average joint error is the average distance between the joints of the

simulated character and the corresponding joints in the reference motion, and is computed using the average of 2048 episodes with random initial frames for each motion. We report these metrics in Table 7.2. The generated motions come from the previous experiment on motion generator performance, except we use the selection heuristic (Section 5.3.2) to first select the best motion generated for each test terrain. Therefore, the motion tracking controllers are tested with 100 generated target motions.

7.4 Blended Denoising

To demonstrate the importance of the blended denoising as described in Section 5.2.2, we experimented with different values for s in the blended denoising update in Equation 5.6. To do this, we used the iteration 4 motion generator to generate 3200 test motions on our test terrain set. Lower values of the blending coefficient tend to produce smoother motions as indicated by the percentage of high jerk frames metric, but perform much worse in terms of terrain penetration and terrain contact losses, as seen in Table 7.3.

Table 7.3: Quantitative results of our 4th iteration motion generator using different blending coefficients s. These metrics measure various aspects of motion quality, and include FWD (final waypoint distance), TPL (terrain penetration loss), TCL (terrain contact loss), and %HJF (percentage of high jerk frames). The motions with the best quality have a balance between terrain compliance (low FWD, TPL, TCL) and temporal continuity (low %HJF). The coloring theme is best = green, worst = red, in between = yellow/orange. We used $s = 0.65$ for automatically augmenting the dataset through PARC, and $s = 0.5$ for generating long horizon motions on complex terrain using the final motion generator that later get tracked by the motion tracker.

Blending Coefficient	**FWD ↓**	**TPL ↓**	**TCL ↓**	**%HJF ↓**
0	0.908	40796	185.3	1.479
0.25	0.776	7411	44.93	1.113
0.5	0.571	4872	32.58	1.017
0.65	0.596	179.6	9.763	2.730
0.75	0.574	132.2	7.718	8.434
1	0.537	129.8	6.751	54.82

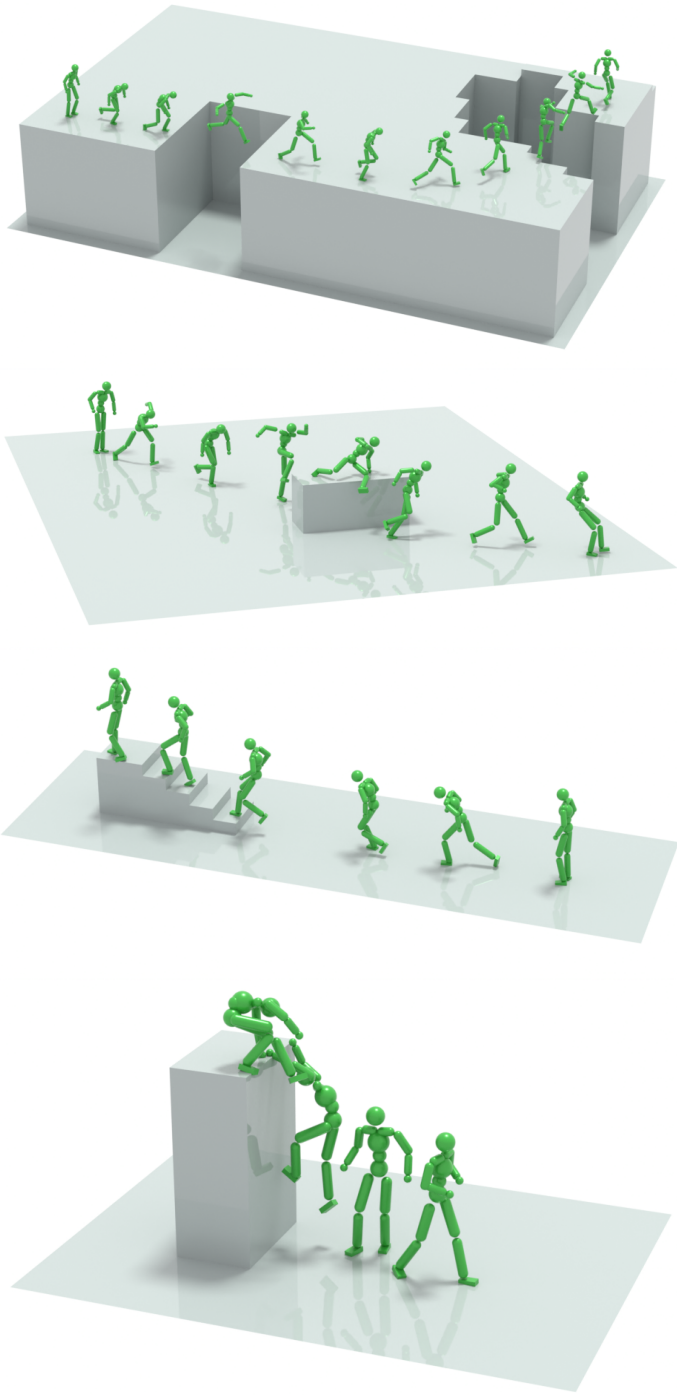

Figure 7.1: Examples of terrain-traversal motions found in our original dataset. The terrain is typically very simple, and the vast majority of clips focus on showcasing one particular parkour skill. The example clips we show, from top to bottom, are jumping, vaulting, running up stairs, and climbing walls.

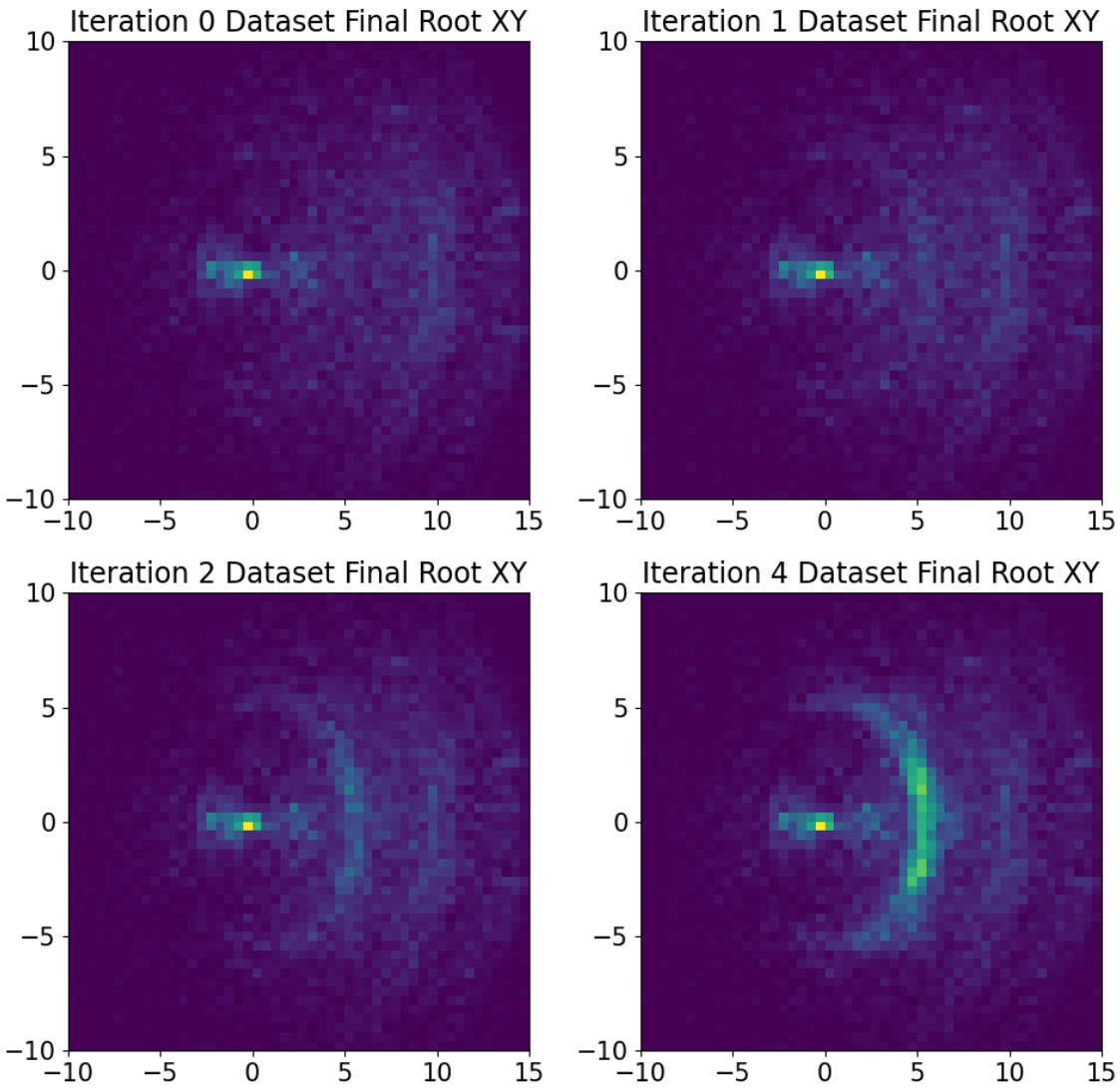

Figure 7.2: A visualization of the distribution of the final relative horizontal (XY) root positions from motion clips in the dataset at different PARC iterations. As the PARC iterations increase (left to right, top to bottom), the dataset expands and increases the diversity of trajectories.

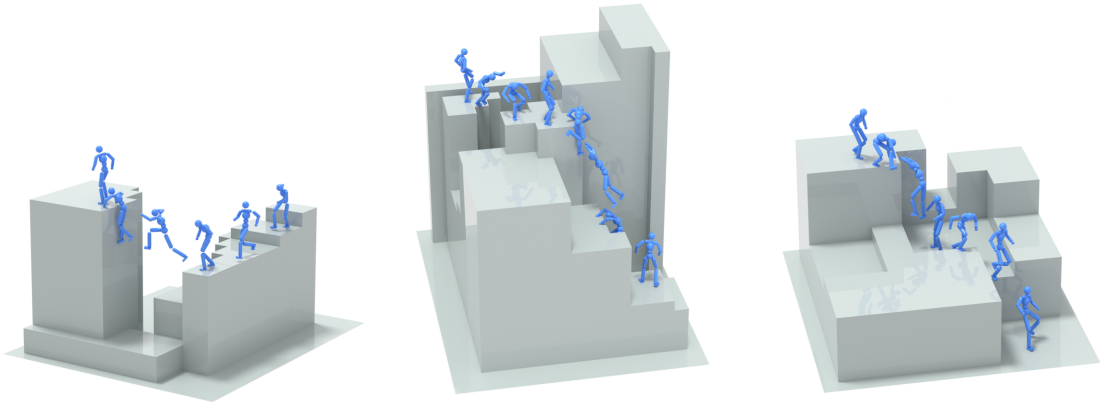

Figure 7.3: Examples of novel physics-based motions generated by PARC. (Left) A character combines a jumping motion with a climbing motion to catch onto a higher ledge. (Middle) A character first jumps a gap, then holds onto a ledge and drops. While falling, the character uses their hands to catch onto another ledge before landing. (Right) A character climbs down and then runs on and off a platform, landing on a lower ground level.

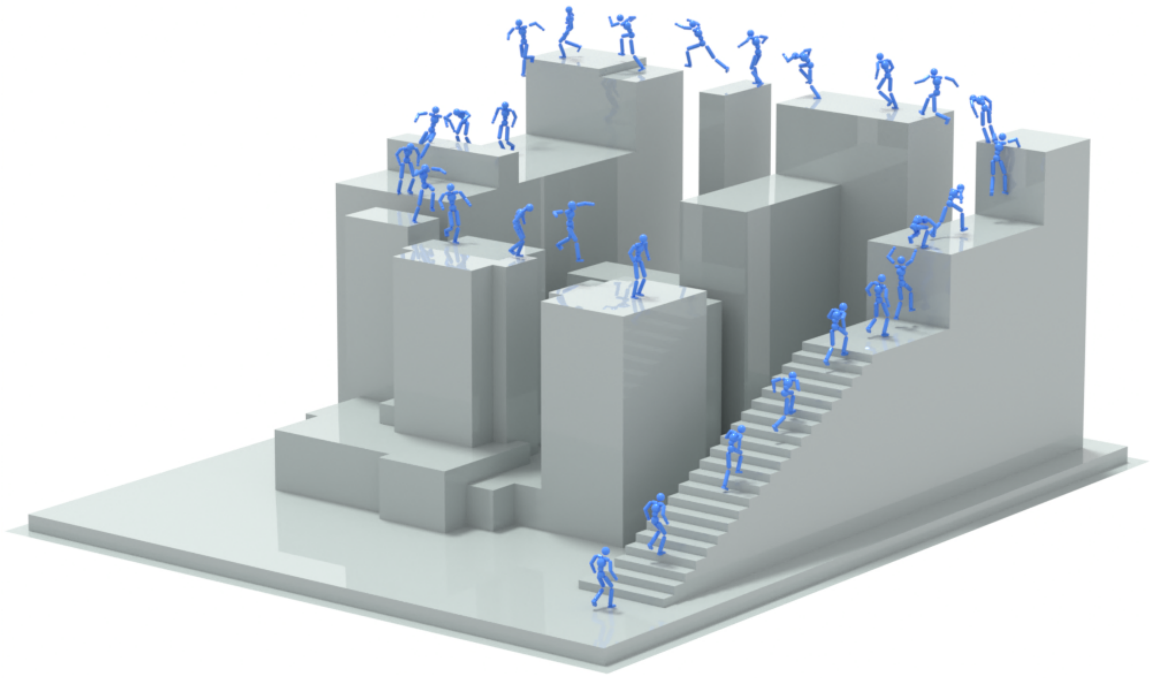

Figure 7.4: A long-horizon physics based motion generated using the final motion generator and motion tracker of PARC on a manually designed monument style terrain, showcasing a diverse mixture of stairs, climbing, and jumping skills.

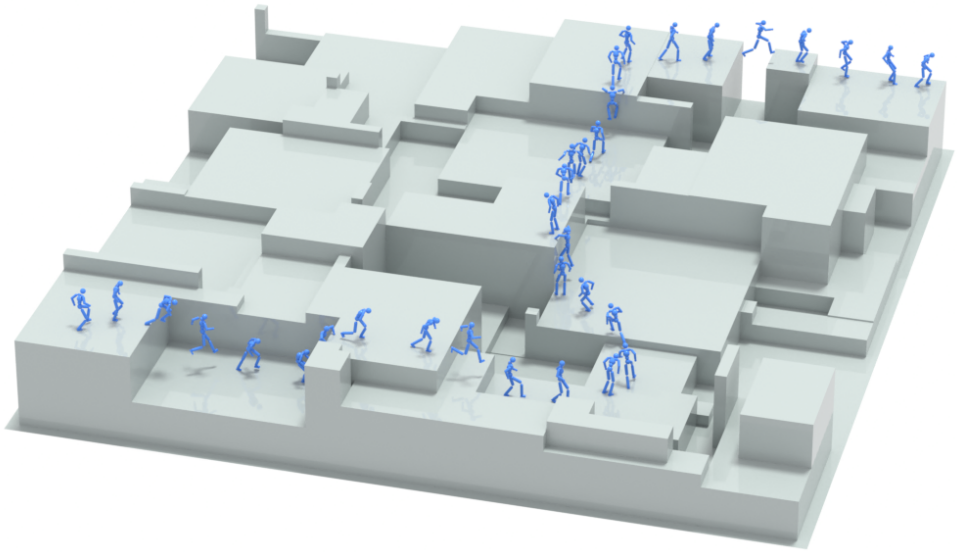

Figure 7.5: A long-horizon physics based motion generated using the final motion generator and motion tracker of PARC on a manually designed random boxes style terrain, generated using the random boxes terrain generation algorithm.

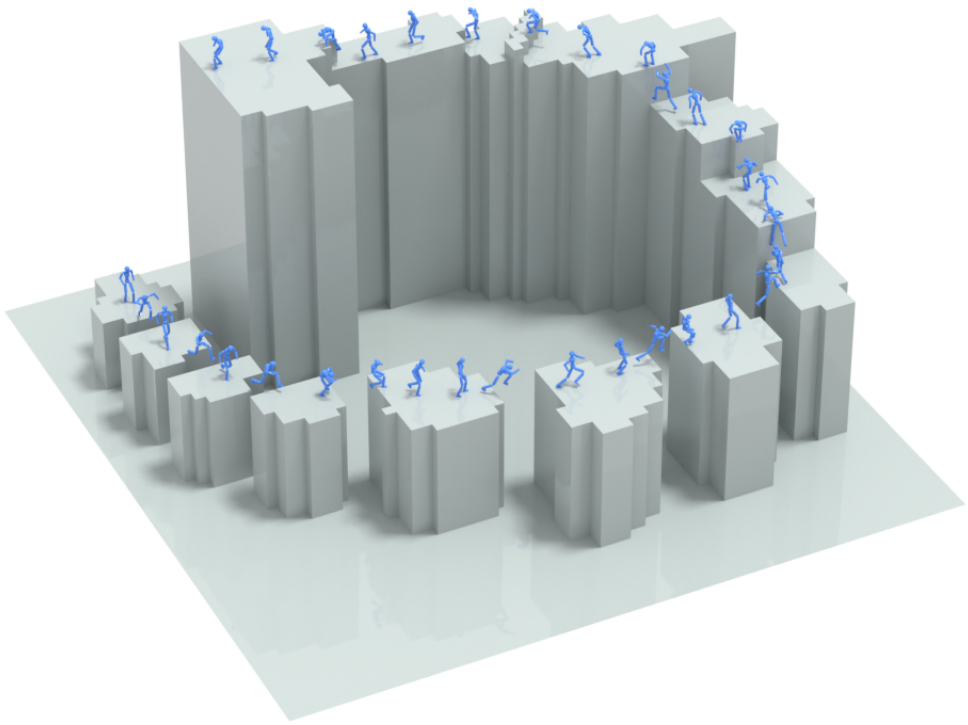

Figure 7.6: A long-horizon physics based motion generated using the final motion generator and motion tracker of PARC on a manually designed spiral terrain, showcasing an impressive chaining of jumping and climbing skills.

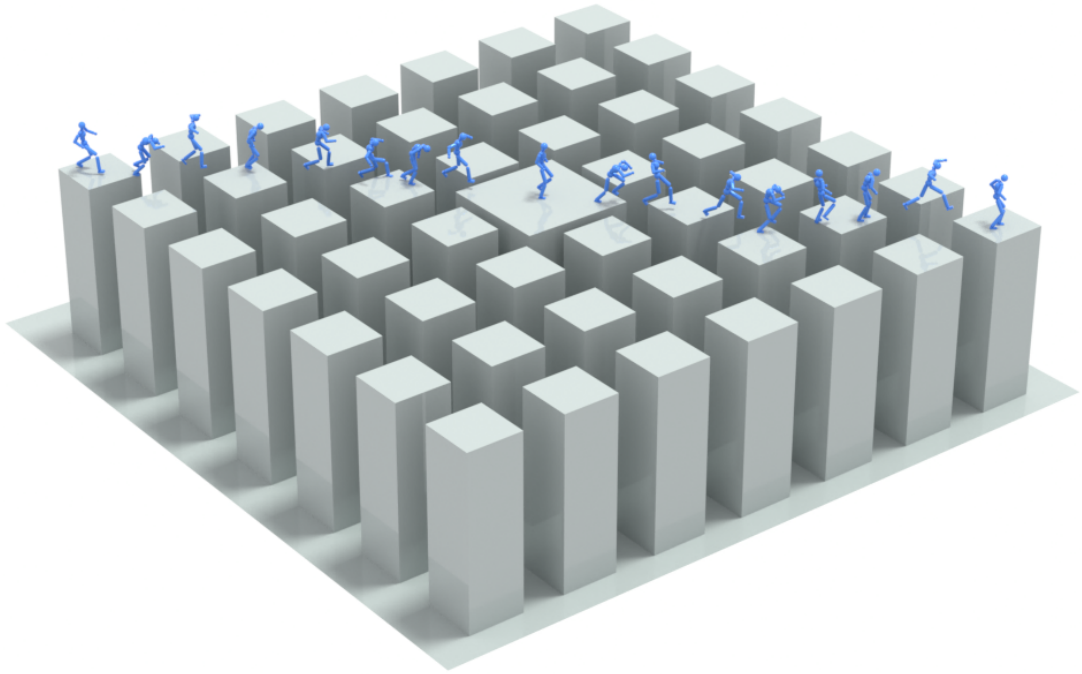

Figure 7.7: A long-horizon physics based motion generated using the final motion generator and motion tracker of PARC on a spaced boxes-style terrain, showcasing lots of jumping skills across unnatural gaps.

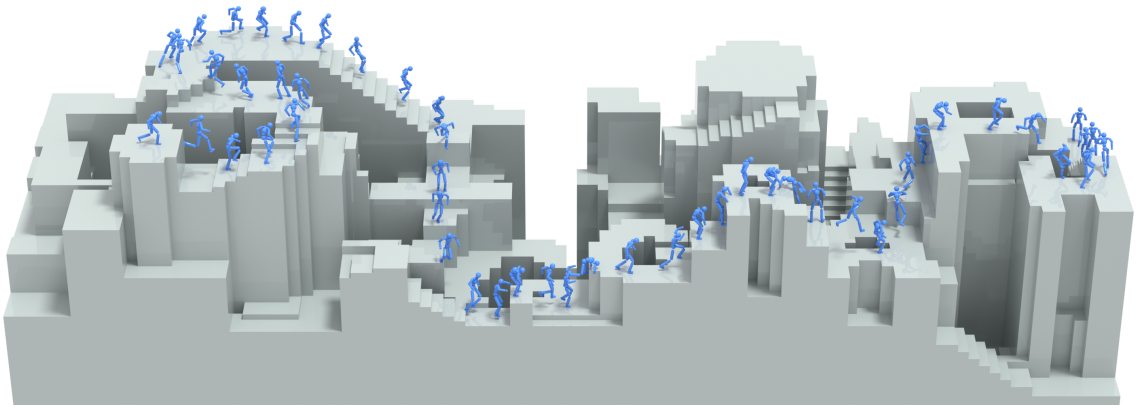

Figure 7.8: A long-horizon physics-based motion generated using the final motion generator and motion tracker of PARC on a manually designed terrain for the SIGGRAPH paper teaser figure.

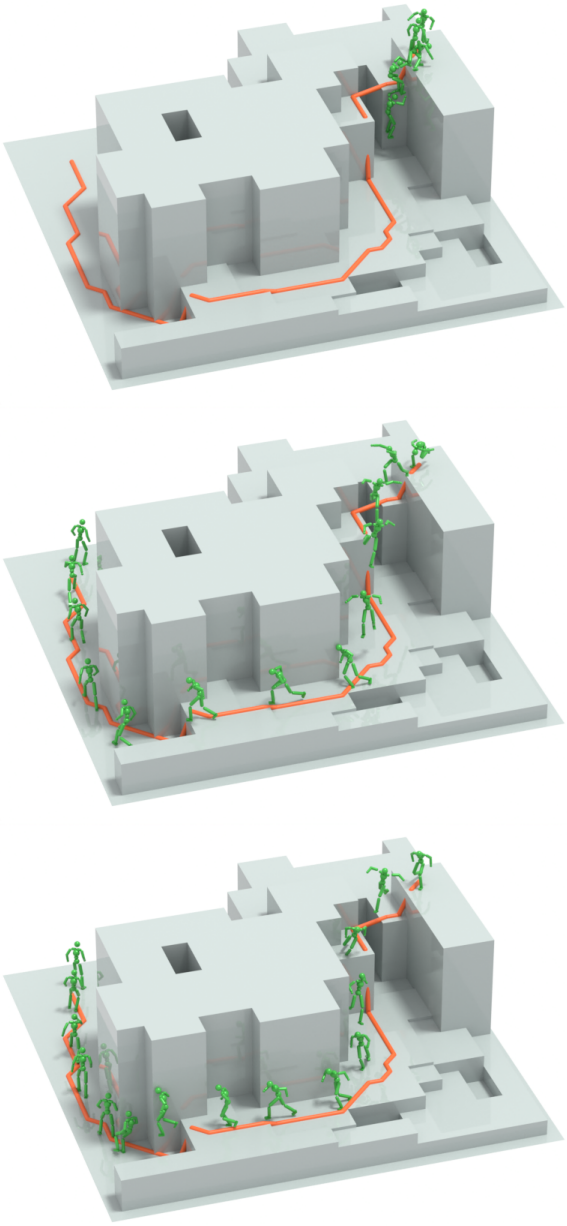

Figure 7.9: Motions generated on a test terrain for different iterations of PARC. Each motion was generated using a batch of 32 for up to 15 seconds of motion time and then automatically selected based on a heuristic incorporating terrain penetration, contact loss, and incompletion penalty. (Top) The iteration 1 motion generator is only trained on the initial dataset, and struggles to navigate across complex terrain. The character was only able to get off the cliff within 15 seconds. (Middle) The motion produced by a motion generator trained on uncorrected generated data from the iteration 1 motion generator. It exhibits physically implausible artifacts such as changing directions while flying through the air. (Bottom) The motion generated by the iteration 3 generator shows the character utilizing contacts with the terrain to navigate to the end of the path.

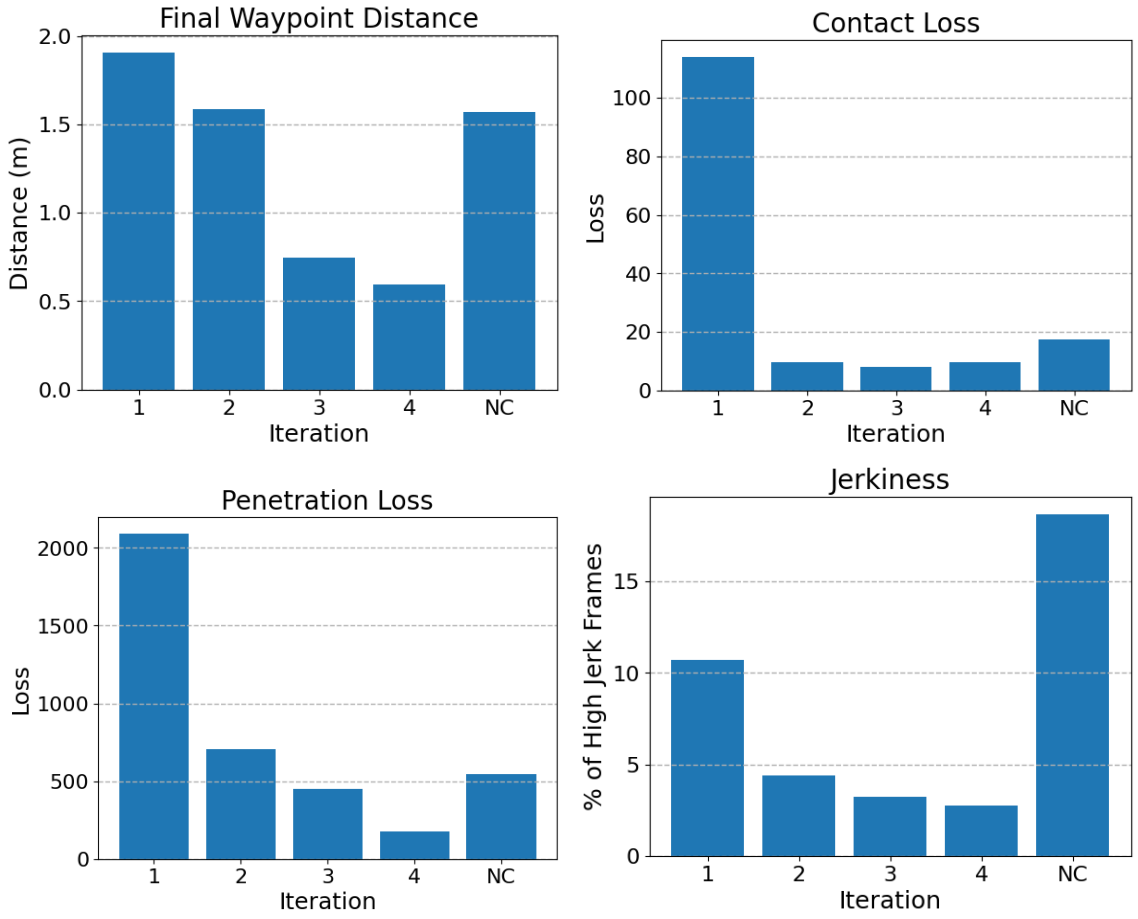

Figure 7.10: Plots showing the measured quantitative results of generated motions from the kinematic motion generator across different PARC iterations, including an iteration with no physics-based motion correction (labeled "NC"). Each metric reports the mean calculated over 3200 motions that were generated across 100 different test terrains for each motion generator. Without physics-based correction, the models generate motions that are much less physically realistic.

Chapter 8

Conclusion

In this work, we introduced PARC, a data-augmentation framework for training versatile physics-based terrain traversal controllers starting with only a small motion dataset. PARC enhances training by jointly optimizing a motion generation model and a physics-based motion tracking controller, with the two models generating data for each other in a symbiotic process. The motion dataset noticeably expands across iterations to include more diverse behaviors, including the discovery of very novel combinations of skills such as jumping into climbing. Once trained, our system enables simulated characters to agilely traverse complex and diverse environments, leveraging the improved motion generation and control capabilities developed through this iterative co-training approach.

8.1 Future Work

PARC leverages the motion generation model and motion tracking controller to progressively expand the capabilities in the dataset, while mitigating physical artifacts by leveraging a physics simulation. However, mitigating physical artifacts does not fully eliminate the risk of the model developing unnatural behaviors. Examples of motions that don't exhibit physical artifacts but are still very unnatural include prolonged sequences of hesitation or dangerous stunts. Exploring more sophisticated techniques to identify and filter out unnatural movements can potentially improve the realism of the synthesized motions.

In our experiments, our models are trained using procedurally generated block-style terrain that lack the diversity and complexity of real world environments. Enhancing our framework to accommodate more complex, realistic, and dynamic scenes will enable the models to synthesize behaviors better suited for interactions with more life-like environments.

Finally, our motion generator is currently designed for offline open-loop motion planning. Incorporating techniques to accelerate inference may enable its use for online closed-loop planning, which can lead to more responsive and robust behaviors. Closed-loop planning will be necessary to bring our framework into real-time video games and robotics.

8.2 Concluding Remarks

There remains a significant generalization gap between character motion generation models and modern image/video/language generation systems, and many attribute this disparity primarily to differences in data scale. In the robotics domain, a growing research trend seeks to narrow this gap by ushering in an era of large-scale data collection via teleoperation [86], manipulation interfaces [6], and translating videos to motion [4, 31]. Complementary efforts explore ways to amplify or enhance data collection through automated cleanup [40], curated data selection [1], and high-quality augmentation for interaction motions [80, 45]. The work presented in this thesis contributes to this direction as well. Although this work focuses on terrain-traversal motions for virtual characters, we believe that PARC offers a compelling pathway toward scaling data-driven learning in robotics more broadly.

There are valid concerns regarding the quality and long-term stability of training on synthetic data. Recent studies indicate that synthetic samples are already being incorporated, sometimes unintentionally, into the training pipelines of new models [71]. Moreover, recursively training on model-generated data in a self-consuming loop can lead to significant degradation in model performance [2, 59]. PARC is also a self-consuming framework, but unlike purely recursive approaches, it incorporates a self-correction stage [10] designed to counteract the pitfalls of repeated self-consumption while retaining key benefits such as dramatic expansion of data diversity and coverage. This highlights a broader research direction: generative models can act as powerful data amplifiers that enable the creation of high-quality motion data for behaviors that would be prohibitively difficult or expensive to capture in controlled environments. By augmenting small datasets of core skills, this approach opens the door to training characters and robots to operate across a wide spectrum of challenging scenarios.

Bibliography

[1] Christopher Agia, Rohan Sinha, Jingyun Yang, Rika Antonova, Marco Pavone, Haruki Nishimura, Masha Itkina, and Jeannette Bohg. Cupid: Curating data your robot loves with influence functions. *arXiv preprint arXiv:2506.19121*, 2025.

[2] Sina Alemohammad, Josue Casco-Rodriguez, Lorenzo Luzi, Ahmed Imtiaz Humayun, Hossein Babaei, Daniel LeJeune, Ali Siahkoohi, and Richard G. Baraniuk. Self-consuming generative models go mad, 2023.

[3] Simon Alexanderson, Rajmund Nagy, Jonas Beskow, and Gustav Eje Henter. Listen, denoise, action! audio-driven motion synthesis with diffusion models. *ACM Trans. Graph.*, 42(4):44:1–44:20, 2023.

[4] Arthur Allshire, Hongsuk Choi, Junyi Zhang, David McAllister, Anthony Zhang, Chung Min Kim, Trevor Darrell, Pieter Abbeel, Jitendra Malik, and Angjoo Kanazawa. Visual imitation enables contextual humanoid control. In *Proceedings of the Conference on Robot Learning (CoRL)*, 2025.

[5] Kevin Bergamin, Simon Clavet, Daniel Holden, and James Richard Forbes. Drecon: data-driven responsive control of physics-based characters. *ACM Trans. Graph.*, 38(6), November 2019.

[6] Cheng Chi, Zhenjia Xu, Chuer Pan, Eric Cousineau, Benjamin Burchfiel, Siyuan Feng, Russ Tedrake, and Shuran Song. Universal manipulation interface: In-the-wild robot teaching without in-the-wild robots. In *Proceedings of Robotics: Science and Systems (RSS)*, 2024.

[7] Setareh Cohan, Guy Tevet, Daniele Reda, Xue Bin Peng, and Michiel van de Panne. Flexible motion in-betweening with diffusion models. *arXiv preprint arXiv:2405.11126*, 2024.

[8] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep rl: A case study on ppo and trpo. In *International Conference on Learning Representations*, 2020.

[9] Levi Fussell, Kevin Bergamin, and Daniel Holden. Supertrack: motion tracking for physically simulated characters using supervised learning. *ACM Trans. Graph.*, 40(6), dec 2021.

[10] Nate Gillman, Michael Freeman, Daksh Aggarwal, Chia-Hong Hsu, Calvin Luo, Yonglong Tian, and Chen Sun. Self-correcting self-consuming loops for generative model training, 2024.

[11] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[12] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[13] Mohamed Hassan, Yunrong Guo, Tingwu Wang, Michael Black, Sanja Fidler, and Xue Bin Peng. Synthesizing physical character-scene interactions. 2023.

[14] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *NIPS*, 2020.

[15] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance, 2022.

[16] David Hoeller, Nikita Rudin, Dhionis Sako, and Marco Hutter. Anymal parkour: Learning agile navigation for quadrupedal robots. *Science Robotics*, 9(88):eadi7566, 2024.

[17] Daniel Holden, Oussama Kanoun, Maksym Perepichka, and Tiberiu Popa. Learned motion matching. *ACM Trans. Graph.*, 39(4), August 2020.

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

[19] Nan Jiang, Zhiyuan Zhang, Hongjie Li, Xiaoxuan Ma, Zan Wang, Yixin Chen, Tengyu Liu, Yixin Zhu, and Siyuan Huang. Scaling up dynamic human-scene interaction modeling, 2024.

[20] Yifeng Jiang, Jungdam Won, Yuting Ye, and C Karen Liu. Drop: Dynamics responses from human motion prior and projective dynamics. *SIGGRAPH Asia*, 2023.

[21] Jordan Juravsky, Yunrong Guo, Sanja Fidler, and Xue Bin Peng. Padl: Language-directed physics-based character control. In *SIGGRAPH Asia 2022 Conference Papers*, pages 1–9, 2022.

[22] Korrawe Karunratanakul, Konpat Preechakul, Supasorn Suwajanakorn, and Siyu Tang. Guided motion diffusion for controllable human motion synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2151–2162, 2023.

[23] Manmyung Kim, Kyunglyul Hyun, Jongmin Kim, and Jehee Lee. Synchronized multi-character motion editing. *ACM Trans. Graph.*, 28(3), July 2009.

[24] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[25] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[26] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. *ACM Trans. Graph.*, 21(3):473–482, July 2002.

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. 25, 2012.

[28] Jehee Lee, Jinxiang Chai, Paul S. A. Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. Interactive control of avatars animated with human motion data. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, page 491–500, New York, NY, USA, 2002. Association for Computing Machinery.

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

[30] Kang Hoon Lee, Myung Geol Choi, and Jehee Lee. Motion patches: building blocks for virtual environments annotated with motion data. In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, page 898–906, New York, NY, USA, 2006. Association for Computing Machinery.

[31] Marion Lepert, Jiaying Fang, and Jeannette Bohg. Masquerade: Learning from in-the-wild human videos using data-editing. *arXiv preprint arXiv:2508.09976*, 2025.

[32] Jiaman Li, Alexander Clegg, Roozbeh Mottaghi, Jiajun Wu, Xavier Puig, and C. Karen Liu. Controllable human-object interaction synthesis. In *ECCV*, 2024.

[33] Hung Yu Ling, Fabio Zinno, George Cheng, and Michiel van de Panne. Character controllers using motion vaes. *ACM Trans. Graph.*, 39(4), 2020.

[34] Libin Liu and Jessica Hodgins. Learning basketball dribbling skills using trajectory optimization and deep reinforcement learning. *ACM Trans. Graph.*, 37(4), July 2018.

[35] Libin Liu, KangKang Yin, Michiel van de Panne, and Baining Guo. Terrain runner: control, parameterization, composition, and planning for highly dynamic motions. *ACM Trans. Graph.*, 31(6), nov 2012.

[36] Zhengyi Luo, Ryo Hachiuma, Ye Yuan, and Kris Kitani. Dynamics-regulated kinematic policy for egocentric pose estimation. *NIPS*, 2021.

[37] Zhengyi Luo, Jiashun Wang, Kangni Liu, Haotian Zhang, Chen Tessler, Jingbo Wang, Ye Yuan, Jinkun Cao, Zihui Lin, Fengyi Wang, Jessica Hodgins, and Kris Kitani. Smplolympics: Sports environments for physically simulated humanoids, 2024.

[38] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu-based physics simulation for robot learning, 2021.

[39] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[40] Yuxuan Mu, Hung Yu Ling, Yi Shi, Ismael Baira Ojeda, Pengcheng Xi, Chang Shu, Fabio Zinno, and Xue Bin Peng. Stablemotion: Training motion cleanup models with unpaired corrupted data. In *SIGGRAPH Asia 2025 Conference Papers (SIGGRAPH Asia '25 Conference Papers)*, 2025.

[41] Kourosh Naderi, Joose Rajamäki, and Perttu Hämäläinen. Discovering and synthesizing humanoid climbing movements. *ACM Trans. Graph.*, 36(4), July 2017.

[42] Shin'ichiro Nakaoka and Taku Komura. Interaction mesh based motion adaptation for biped humanoid robots. In *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, pages 625–631, 2012.

[43] Alex Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models, 2021.

[44] OpenAI, :, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique P. d. O. Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning, 2019.

[45] Chuer Pan, Litian Liang, Dominik Bauer, Eric Cousineau, Benjamin Burchfiel, Siyuan Feng, and Shuran Song. One demo is worth a thousand trajectories: Action-view augmentation for visuomotor policies. In *9th Annual Conference on Robot Learning*, 2025.

[46] Soohwan Park, Hoseok Ryu, Seyoung Lee, Sunmin Lee, and Jehee Lee. Learning predict-and-simulate policies from unorganized human motion data. *ACM Trans. Graph.*, 38(6), nov 2019.

[47] Xue Bin Peng. Developing locomotion skills with deep reinforcement learning. Master's thesis, University of British Columbia, 2017.

[48] Xue Bin Peng. Mimickit: A reinforcement learning framework for motion imitation and control, 2025.

[49] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018.

[50] Xue Bin Peng, Glen Berseth, Kangkang Yin, and Michiel Van De Panne. Deeploco: dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Trans. Graph.*, 36(4), jul 2017.

[51] Xue Bin Peng, Yunrong Guo, Lina Halper, Sergey Levine, and Sanja Fidler. Ase: Large-scale reusable adversarial skill embeddings for physically simulated characters. *ACM Trans. Graph.*, 41(4), July 2022.

[52] Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. Amp: Adversarial motion priors for stylized physics-based character control. *ACM Trans. Graph.*, 40(4), July 2021.

[53] Inigo Quilez.

[54] Davis Rempe, Tolga Birdal, Aaron Hertzmann, Jimei Yang, Srinath Sridhar, and Leonidas J. Guibas. Humor: 3d human motion model for robust pose estimation. In *International Conference on Computer Vision (ICCV)*, 2021.

[55] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[56] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018.

[57] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[58] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6:1–48, 2019.

[59] Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Yarin Gal, Nicolas Papernot, and Ross Anderson. The curse of recursion: Training on generated data makes models forget, 2024.

[60] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[61] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics, 2015.

[62] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *ICLR*, 2021.

[63] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations, 2021.

[64] Sebastian Starke, Ian Mason, and Taku Komura. Deepphase: Periodic autoencoders for learning motion phase manifolds. *ACM Transactions on Graphics (TOG)*, 41(4):1–13, 2022.

[65] Sebastian Starke, He Zhang, Taku Komura, and Jun Saito. Neural state machine for character-scene interactions. *ACM Trans. Graph.*, 38(6):209–1, 2019.

[66] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.

[67] Chen Tessler, Yunrong Guo, Ofir Nabati, Gal Chechik, and Xue Bin Peng. Maskedmimic: Unified physics-based character control through masked motion inpainting. *ACM Transactions on Graphics (TOG)*, 2024.

[68] Guy Tevet, Sigal Raab, Brian Gordon, Yonatan Shafir, Daniel Cohen-Or, and Amit H Bermano. Human motion diffusion model. *ICLR*, 2023.

[69] Jonathan Tseng, Rodrigo Castellon, and C Karen Liu. Edge: Editable dance generation from music. *arXiv preprint arXiv:2211.10658*, 2022.

[70] Unreal Engine. Game animation sample project, 2024.

[71] Veniamin Veselovsky, Manoel Horta Ribeiro, and Robert West. Artificial artificial artificial intelligence: Crowd workers widely use large language models for text production tasks, 2023.

[72] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[73] Jiashun Wang, Jessica Hodgins, and Jungdam Won. Strategy and skill learning for physics-based table tennis animation. In *ACM SIGGRAPH 2024 Conference Papers*, pages 1–11, 2024.

[74] Yinhuai Wang, Qihan Zhao, Runyi Yu, Ailing Zeng, Jing Lin, Zhengyi Luo, Hok Wai Tsui, Jiwen Yu, Xiu Li, Qifeng Chen, Jian Zhang, Lei Zhang, and Tan Ping. Skillmimic: Learning reusable basketball skills from demonstrations. *arXiv preprint arXiv:2408.15270*, 2024.

[75] Lilian Weng. What are diffusion models? *lilianweng.github.io*, Jul 2021.

[76] Jungdam Won, Deepak Gopinath, and Jessica Hodgins. Control strategies for physically simulated characters performing two-player competitive sports. *ACM Trans. Graph.*, 40(4), July 2021.

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

[78] Zhaoming Xie, Sebastian Starke, Hung Yu Ling, and Michiel van de Panne. Learning soccer juggling skills with layer-wise mixture-of-experts. In *ACM SIGGRAPH 2022 Conference Proceedings*, SIGGRAPH '22, New York, NY, USA, 2022. Association for Computing Machinery.

[79] Michael Xu, Yi Shi, KangKang Yin, and Xue Bin Peng. Parc: Physics-based augmentation with reinforcement learning for character controllers. In *SIGGRAPH 2025 Conference Papers (SIGGRAPH '25 Conference Papers)*, 2025.

[80] Lujie Yang, Xiaoyu Huang, Zhen Wu, Angjoo Kanazawa, Pieter Abbeel, Carmelo Sferrazza, C. Karen Liu, Rocky Duan, and Guanya Shi. Omniretarget: Interaction-preserving data generation for humanoid whole-body loco-manipulation and scene interaction, 2025.

[81] Heyuan Yao, Zhenhua Song, Baoquan Chen, and Libin Liu. Controlvae: Model-based learning of generative controllers for physics-based characters. *ACM Trans. Graph.*, 41(6), nov 2022.

[82] Heyuan Yao, Zhenhua Song, Yuyang Zhou, Tenglong Ao, Baoquan Chen, and Libin Liu. Moconvq: Unified physics-based motion control via scalable discrete representations. *ACM Trans. Graph.*, 43(4), July 2024.

[83] Hongwei Yi, Justus Thies, Michael J. Black, Xue Bin Peng, and Davis Rempe. Generating human interaction motions in scenes with text control. In *Computer Vision – ECCV 2024: 18th European Conference, Milan, Italy, September 29–October 4, 2024, Proceedings, Part IV*, page 246–263, Berlin, Heidelberg, 2024. Springer-Verlag.

[84] Ri Yu, Hwangpil Park, and Jehee Lee. Human dynamics from monocular video with dynamic camera movements. *ACM Transactions on Graphics (TOG)*, 40(6):1–14, 2021.

[85] Ye Yuan, Jiaming Song, Umar Iqbal, Arash Vahdat, and Jan Kautz. Physdiff: Physics-guided human motion diffusion model. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

[86] Yanjie Ze, Zixuan Chen, João Pedro Araújo, Zi ang Cao, Xue Bin Peng, Jiajun Wu, and C. Karen Liu. Twist: Teleoperated whole-body imitation system. *arXiv preprint arXiv:2505.02833*, 2025.

[87] Chong Zhang, Nikita Rudin, David Hoeller, and Marco Hutter. Learning agile locomotion on risky terrains, 2024.

[88] Chong Zhang, Wenli Xiao, Tairan He, and Guanya Shi. Wococo: Learning whole-body humanoid control with sequential contacts, 2024.

[89] Haotian Zhang, Ye Yuan, Viktor Makoviychuk, Yunrong Guo, Sanja Fidler, Xue Bin Peng, and Kayvon Fatahalian. Learning physically simulated tennis skills from broadcast videos. *ACM Trans. Graph.*, 42(4), jul 2023.

[90] Allan Zhao, Jie Xu, Mina Konaković-Luković, Josephine Hughes, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. Robogrammar: graph grammar for terrain-optimized robot design. *ACM Trans. Graph.*, 39(6), November 2020.

[91] Kaifeng Zhao, Gen Li, and Siyu Tang. A diffusion-based autoregressive motion model for real-time text-driven motion control. *https://arxiv.org/abs/2410.05260*, 2024.

[92] Ziwen Zhuang, Shenzhe Yao, and Hang Zhao. Humanoid parkour learning, 2024.