DRL-Controlled Fluid-Rigid Simulation

Zhiwei Zhao*, Yuting Weng, Kaiwen Li

Abstract

In this project, we mainly target on creating a DRL (Deep Reinforcement Learning) model to simulate the fluid-rigid dynamic system, and achieve control to the fluid as well as solid object to fulfill the given task, giving a appealing vision without losing the visual realism or defying any physical law. In this work, we combine the fluid simulation method in computer graphics with deep reinforcement learning to derive a physics-based fluid-rigid controlling model to achieve certain goal, that is, to get the physics information from the simulated fluid-rigid field and realize the control to the fluid, which directs the rigid body, thus realize the control to the solid as well. For specific design, we firstly create a simulation solver based on MPM (Material Point Method) to integrate the physical information into the whole domain, and then get the observation state that will be input into the network architecture. The network includes the auto-encoder and the actor-critic framework。The former one plays the role of encoding observation state into small vector and the latter one uses Soft Actor Critic algorithm to train a policy network which can output the best policy based on the observation state. Besides, to reach the goal of generalizing models into new tasks and environment settings, we introduce a latent vector into the structure which can be trained to represent the environment information. This can help the model adapt to new environment setting never seen before. Sufficient experiments have shown that the structure is efficient in specific task and has a strong generalization ability to new tasks.

Keywords: Deep Reinforcement Learning, Material Point Method, Auto-Encoder, Soft Actor Critic Algorithm, Meta Learning

Contents

1 Control through Deep Reinforcement Learning

1.1 Introduction

Nowadays, fluid animation has become a hot topic in computer graphics and related field, this field involves simulating the behavior of fluids such as water, smoke, and fire to create realistic and visually appealing animations in computer-generated imagery (CGI), video games, and simulations. Fluid animation has been widely researched over these years, quantities of techniques have been proposed to achieve visual plausibility to people. Such as MAC (Marker-and-Cell) method to solve partial differential equations on a grid, capturing fluid flow and pressure, Particle-based methods SPH (Smoothed Particle Hydrodynamics) using particles to compute different physical properties and so on. Through continuous research, lots of progress has been made and fluid animation is being more and more near realistic(i.e. the achievement of add the research of vorticity and turbulence common in real fluid process) and visually appealing(i.e. the development of many surface tracking and rendering techniques leading to the visually convincing and appealing animation).

Recent years, with the rise of GPUs (graphics processing units) and wave of DL (Deep Learning), large number of researchers apply DL techniques to fluid animation, data-driven approaches aimed to capture the intricate details of fluid efficiently largely due to the power of GPUs. While deep learning is very promising in various areas, it does have set of challenges and limitations, especially when applied to fluid animation.

A specific application in physics-based fluid animation, controlling the fluid behaviors to achieve certain goals leads to additional challenges, due to its high-dimensionality and strong coupling between degrees-of-freedom [1]. One main obstacles for realistic fluid controllers is non-smoothness, which is caused by discontinuous fluid-solid boundary conditions, so it is no more applicable for differentiable systems and pure DL. As for a physical fluid-solid system, the physical properties in current state are highly relevant to themselves in previous states attributed to the hyperbolic nature of PDE (partial differential equation) in real physical system. Due to the superiority of DRL (deep reinforcement learning) in real-time decision-making applications, since the real process to achieve a certain goal can be seen as a Markov decision process, so if the DRL models are fully trained and fine-tuned within various situations or specific fluid simulation scenarios, it can result in a large leap in less

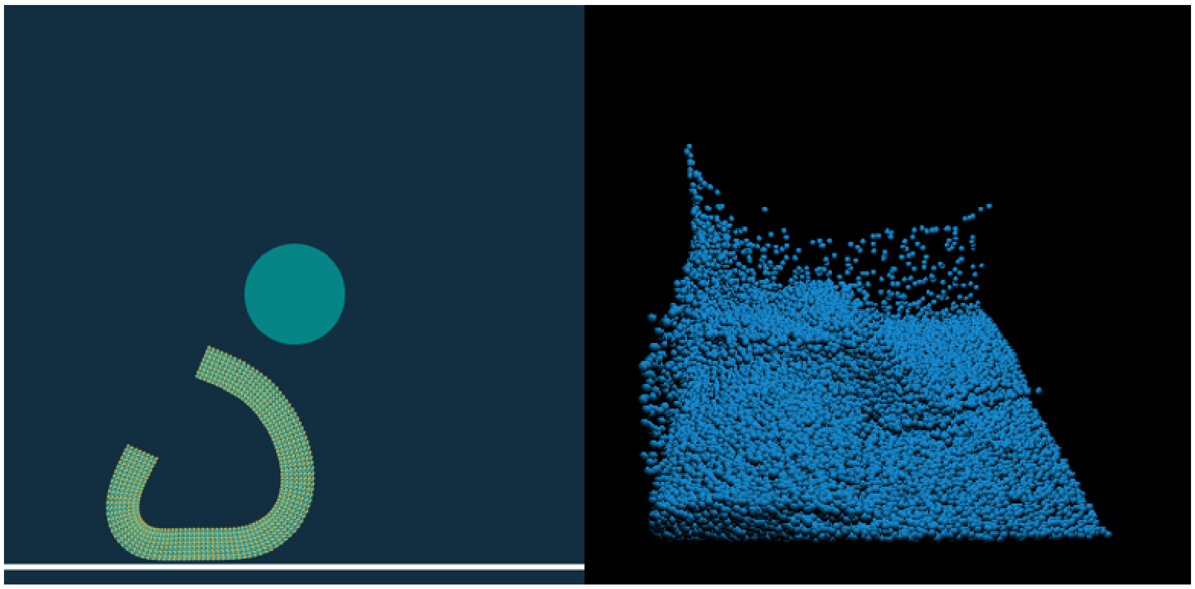
Figure 1: Non-smoothness and lack of differentiability in fluid rather than solid body.

need for data and larger flexibility and versatility for use.

In reality, fluid-directed solid body control often contains multiple tasks, so how to develop robust models for corresponding simulation scenarios in 2D and 3D spaces is a key point that needs attempt to accomplish for further advancement in fluid animations and even animation industry with related fields.

1.2 Related work

The following part consists of the development process about some techniques and ideas that have been used for the control of fluids as well as solid objects, and we will also explain how to incorporate DRL into the controllers to achieve certain tasks. To control fluid, so-called fully actuated dynamic systems had been used for many fluid control algorithms[2, 3, 4, 5] where ghost forces can be injected everywhere in the computed domain and further, the single-phase flows assumption, these actually aim to form a differentiable system and therefore model-based control can be performed, but this is still costly to compute the gradient and won't bring very high visual realism. Also ,a lot of non-physical fluid animation editing algorithms have been proposed, which can be seen very fast, but will lead to the loss of physical realism. So, changing boundary conditions without any modification to fluid dynamic model can be a super idea in this research which has been done by several researchers similarly[6, 7, 8].

Due to the low-dimension nature of rigid or near-rigid solid objects, there are many solid control algorithms using DL and DRL[9, 10, 11, 12, 13]. In these researches, unactuated near-rigid objects are controlled using high-dimensional fluid bodies. What needs to notice is that this work covers the implement of transfer learning RL, which enable the controller to adapt to various environments from small amounts of experiences and solve multiple tasks reasonably well.

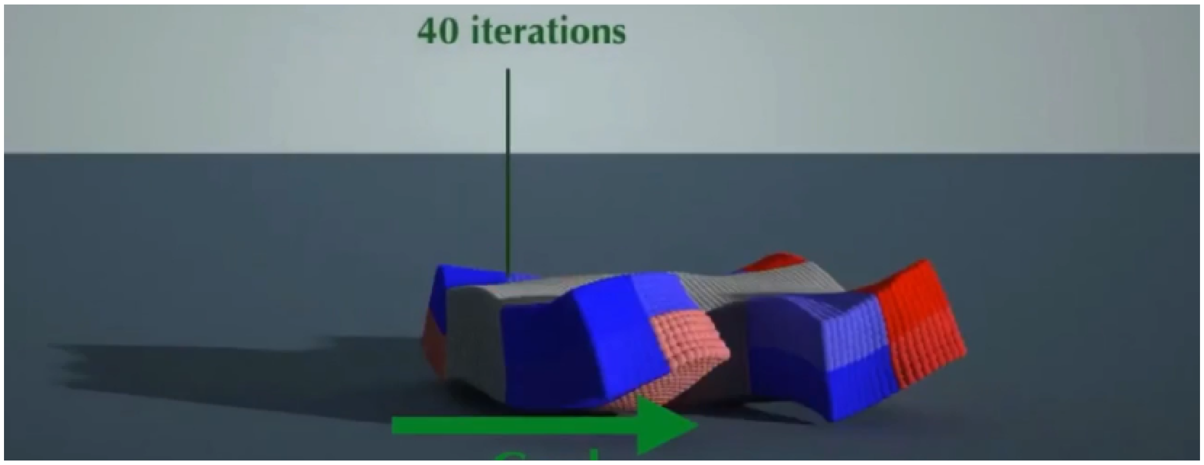

Figure 2: Solid elastic body control through deep learning [9].

2 Environment setup

2.1 MPM simulation algorithm

Basically, our tasks are accomplished by two parts, one is recognizer, while the other is controller. The recognizer can adapt to the change of the object in the simulated field as well as different simulation environments which actually can be done through users' modification, giving encoded information to the controller such as the ball indices if the solid object is ball.

Of course, to simulate a Fluid-Solid dynamics system, the fluid simulation algorithms are surely needed, here, what we focus on is MPM (material point method) which is proposed by Hu[6], which is commonly used for simulating the behavior of materials under various conditions, including deformations, fluid flows, and other physical processes, and is particularly effective in handling large deformation and material interfaces. The basic ideas of MPM are:

- Particles and Grids: MPM uses a combination of particles and a background grid. The material is represented by particles, each carrying material properties, and interactions with the surrounding grid are used to compute forces and update the simulation.
- Particle Movement: Particles move through the grid, and their properties are transferred to the grid to compute forces and obtain updated values. The method accounts for both Lagrangian and Eulerian descriptions of motion.
- Contact Handling: MPM is well-suited for handling contact and material interfaces, making it applicable to problems involving complex geometries and multiple materials.

In our research, we mainly use the extended algorithm of MPM, called **MLS-MPM** (Moving Least Square Material Point Method), and this incorporates Moving Least Squares (MLS) approximation for interpolating and transferring properties between particles and the grid. The advantages using this are: providing a more accurate representation of material behavior by interpolating values between particles and the grid, achieving higher order accuracy in terms of spatial interpolation, as well as offering improved stability in certain scenarios.

2.2 Modelling of fluid-rigid system

As stated before, our research aims to solve the two main obstacles, non-smoothness and underactuation, for the former, we choose not to use the model-based but **learning-based**, for the latter, as explained above, the fully actuated algorithms will lead to unphysical results. In the fluid-solid dynamic systems, the dimension to be controlled is much larger than available controlling parameters, basically latent space can help address this problem.

The physical models we are based on to achieve the control are essentially physical laws in the form of PDEs, in our target coupled fluid/rigid system, actually they are respectively Navier-Stokes equations and Newton-Euler equations which are shown below:

$$\begin{aligned}
&\forall \mathbf{x} \in \Omega_f : \begin{cases} \dot{\mathbf{u}} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \mathbf{f} \\ \dot{\rho} + (\mathbf{u} \cdot \nabla)\rho = 0 \end{cases} \\
&\forall \mathbf{x} \in \Omega_r : \begin{cases} \mathbf{u} = \mathbf{v} + \omega \times (\mathbf{x} - \mathbf{o}) \\ \mathbf{M} \begin{pmatrix} \dot{\mathbf{v}} \\ \dot{\omega} \end{pmatrix} + \begin{pmatrix} \int_{\partial\Omega_r} \mathbf{p} d\mathbf{s} \\ \omega \times \mathbf{I}\omega + \int_{\partial\Omega_r} \mathbf{p}(\mathbf{x} - \mathbf{c}) \times d\mathbf{s} \end{pmatrix} = 0 \\ \begin{pmatrix} \dot{\mathbf{c}} \\ \dot{\mathbf{R}} \end{pmatrix} = \begin{pmatrix} \mathbf{v} \\ [\omega]\mathbf{R} \end{pmatrix} \end{cases} \\
&\forall \mathbf{x} \in \Omega_f \cup \Omega_r : \mathbf{u} \in C^0.
\end{aligned} \tag{1}$$

where Ω_f denotes for the domain of fluid body, and Ω_r denotes for the domain of rigid body, what needs to notice is the last formula $\forall x \in \Omega_f \cup \Omega_r : u \in C^0$, this is how we enforce the boundary condition between fluid and rigid bodies, by making u C^0-continuous in the whole domain.

To implement our project ideas, the most important step is to train a robust and stable model that is truly able to achieve the goals of fluid-rigid control. Firstly, we create a solver to simulate the state of the fluid and rigid body based on the MLS-MPM, and also make it able to generate training data velocity **u** and **v** automatically, then, we encode this data to get the corresponding state of the whole simulation field. To accelerate the training process, we use GPU parallel optimization to optimize the index search and boundary problem which is originally sacrificed for efficiency in VRAM, furthermore, we use fewer particles to save simulation burden by optimization with sorting algorithm. The parallel optimization is implemented using Taichi with cuda.

3 Algorithm

Our transferable fluid-rigid body control can be treated as a MDP (Markov decision process) in the continuous state-action spaces, inspired by the definition in the work of Haarnoja et al.[14]. To our single-task setting, we can design our control task by the tuple

(S, A, p, r, γ), where S is the state-space, A is the action space, $p(s_t+1|s_t, a_t)$ is the transition probability. Our objective function to maximize is the cumulative reward as shown below:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{z}) \sim \rho_\pi} \left[r\left(\mathbf{s}_t, \mathbf{a}_t, \mathbf{z}\right) + \alpha \mathcal{H}\left(\pi\left(\cdot \mid \mathbf{s}_t, \mathbf{z}\right)\right) \right] \tag{2}$$

As the On-Policy algorithms require new trajectory samples even for a small update of policy parameters, so in our research, we choose the Off-Policy Reinforcement Learning method which could be more sample efficient since fluid simulation samples are highly costly to compute. Our network architecture can basically be separated into two parts, the auto-encoder and the actor-critic framework. The auto-encoder is used to encode the state of the environment including the velocity field of jelly and the state of both the ball and the rigid body to a vector to reduce the shape of input state of the actor-critic framework. The construction of the auto-encoder is shown in Fig. 3. In the actor-critic framework, the critic network is what we choose to approximate cumulative reward and the actor network is what we choose to output exploration policies in the training stage and deterministic policies in the eval stage. Experience replay buffer is what we also incorporate to deal with the sparse rewards in our project, owing to the characteristic of the intermittent impact on the rigid body by fluid and our target task maintaining the ball over a certain height with only one fluid jet.

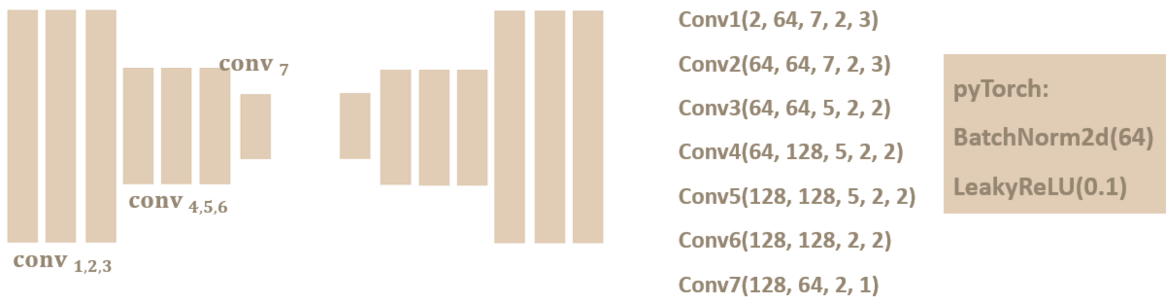

Figure 3: The construction of auto-encoder

3.1 Off-Policy Reinforcement Learning

We use soft actor critic algorithm to accomplish off-policy reinforcement learning. Similar to actor critic algorithm, soft actor critic algorithm construct an all-around agent that can directly output policies and evaluate the quality of current policies in real-time through value functions. So in this algorithm there are two networks, one Actor neural network responsible for generating policies and one Critic neural network responsible for evaluating policies. For evaluating policies. Besides, to calculate the policy gradient, the estimated value of Q value is used, so a separate neural network needs to be added to fit the Q value function. In our structure, we use double Q-learning to fit the Q-value function and its advantage is that we can reduce the over-estimating of Q values in the training stage by choosing the lower estimated one. Notably, we don't use actor critic algorithm but soft one[14]. Soft here means that some regularization terms are added to avoid overfitting. In the algorithm, the regularization term of entropy is added into the loss function of policy network. By maximize the entropy of actions output by the policy neural network, we let the strategy be randomized, that is, the probability of each action output is spread as much as possible instead of concentrating on one action. So the optimized policy based on the newly-constructed loss function is shown as

$$\pi^{*}=\arg\max_{\pi}\mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[\sum_{t}\underbrace{R\left(s_t,a_t\right)}_{\text{reward}}+\alpha\underbrace{H\left(\pi\left(\cdot\mid s_t\right)\right)}_{\text{entropy}}] \tag{3}$$

Then we can construct the loss function of critic network and actor network. The critic network can be updated by the following critic loss:

$$L_{\text{critic}}=\mathbb{E}_{\substack{(s_t,a_t,s_{t+1})\sim D\\ z\sim q_\phi(z\mid s_t,a_t,s_{t+1})}}\left[Q_\theta\left(s_t,a_t,z\right)-\left(r+\gamma V\left(s_{t+1},z\right)\right)\right]^2 \tag{4}$$

where V is the target netwrok in double Q-learning and D is the replay buffer. The actor loss is:

$$L_{\text{actor}}=\mathbb{E}_{\substack{(s_t,a_t,s_{t+1})\sim D\\ z\sim q_\phi(z\mid s_t,a_t,s_{t+1})}}\left[D_{KL}\left(\pi_\varphi\left(\cdot\mid s_t,z\right)\|\frac{\exp\left(Q_\theta\left(s_t,\cdot,z\right)\right)}{Z_\theta\left(s_t,z\right)}\right)\right] \tag{5}$$

where $Z_\theta(s_t,z)$ is used to normalize the distribution. The reward function that the algorithm uses in updating parameter is shown below:

$$r(S,a) = w_c \ln\left(e - n_c \left\|c - c^*\right\|^2\right) + w_v \ln\left(e - n_v \|v\|^2\right) \tag{6}$$

where c is the height of the ball and c^* is the expected height, v is the speed of ball including linear velocity and angular velocity。

3.2 Context Transfer

A latent vector z is used to represent the changes of simulation environment. Since we want the latent vector z be updated within the training stage, we create a context encoder to encode the observation space, action space, reward spae, environment changing in a specific transition of a trajectory into a latent vector z. This can be constructed since we assume that z can be inferred from transitions, which obeys a Gaussian distribution. The reason why we assume it obeys a Gaussian distribution is that through the variational method, the Gaussian distribution can be used to estimate the distribution of each layer to approximately calculate the mutual information. To avoid overfitting, information bottleneck is introduced in the context encoder that we are going to use the KL divergence of the output of the context encoder and the hidden layer coding, that is,

$$L_{\mathrm{KL}} = \beta \mathrm{D}_{\mathrm{KL}}(q_\phi(z|b))||p(z) \tag{7}$$

where b is the transition tuple and $p(z)$ is the unit normal distribution wich is used to estimation the distribution of hidden layer. So the loss functionm of the context encoder is

$$L_{\text{context encoder}} = L_{\mathrm{KL}} + L_{\text{critic}} \tag{8}$$

As a conclusion, the pseudo code of our algorithm is shown in Algorithm. 1.

3.3 Meta Reinforcement Learning

The goal of meta learning is to enable the model to acquire the ability to "learn to learn", allowing it to quickly learn new tasks on the basis of acquiring existing "knowledge". So

Algorithm 1: General algorithm

```
A set of random simulators $\{\tau_i\}$, learning rates $\alpha_{1,2,3}$
Initial parameters $\phi, \theta, \psi$ corresponding to context encoder, critic network and actor
  network, replay buffer $D \leftarrow$ initial randomized sampling trajectories
for each iteration do
    for each $\tau_i$ do
        Initialize transition $b_i \leftarrow \emptyset$
        Sample Trajectories
        for $t = 1, \dots, K$ do
            Sample $z \sim q_\phi(z|b)$
            Sample $a_t \sim \pi_\theta(s_t, z)$
            Advance simulator and add
            $D \leftarrow D \cup \{(s_t, a_t, s_{t+1}, z, r_t)\}$
            $b_i' \leftarrow b_i \cup \{(s_t, a_t, s_{t+1})\}$
        end
        for sampled batch from D do
            Calculate $L_{\text{critic}}, L_{\text{actor}}, L_{\text{KL}}$
            $\phi \leftarrow \phi - \alpha_1 \nabla_\phi (L_{\text{critic}} + L_{\text{KL}})$
            $\theta \leftarrow \theta - \alpha_2 \nabla_\theta L_{\text{critic}}$
            $\psi \leftarrow \psi - \alpha_3 \nabla_\psi L_{\text{actor}}$
        end
    end
end
```

under our settings, existing ”knowledge” means the models that have learned the controlling patterns based on the environment constructed by several simulators which will be fixed before training. ”Learn to learn” means that the trained model has the ability to transfer what it has learned in previous fixed simulators to new ones never seen before.

More specifically, in the training stage, the agent is exposed to several simulators constructed from a task distribution and it learns a set of parameters that allows it to quickly adapt to new tasks it has never seen before. So in the testing stage, given a new, the agent can use its learned initialization parameters to adapt quickly with a small amount of task-specific data.

3.4 Implementation

For implementation, the construction of auto-encoder has been shown already. In the actor-critic framework, first for the critic network, it is constructed by two q-function network and one v-function network. All are constructed from `rlkit.jittor.networks.FlattenMlp` which is realized by previous studies in `rlkit`. For the actor network, it is constructed from `rlkit.jittor.sac.policies.TanhGaussianPolicy`. This network output the mean and standard derivation of the action distribution and construct the action distribution as a Gaussian distribution. In the training stage, the network will sample actions from the Gaussian distribution constructed for exploration. However, in the testing stage, the network will directly output the mean of the distribution to make it deterministic. Both output actions will pass a Tanh transformation before the environment advances. The replay buffer is constructed from `rlkit.data_management.env_replay_buffer.MultiTaskReplayBuffer`. Such a buffer can store trajectories for different simulators.

4 Result

4.1 Targets of the control

The designing of our project is as following. Within a fixed box (whose right, left and bottom sides are free to pass, while the top side is a bouncing condition), the tube is originally placed at the bottom, with a ball initially appearing at the top. Then the tube continues to eject water from the outlet, and it can be adjusted by exerting horizontal and rotational acceleration. As time goes by, it is required to target the ball at certain height though the liquid-solid interactions. Note that by adapting the coefficients, the maximum reward can be obtained per transition is 11. For the discount factor, we have adopted a value of 0.99, the reason behind is that we evaluating every moment and want the ball to be continuously placed at that height. We also add turbulent fluctuation to the tube exit, so it is impossible to find a fixed position and orientation that can achieve the purpose. The control needs to be done consciously.

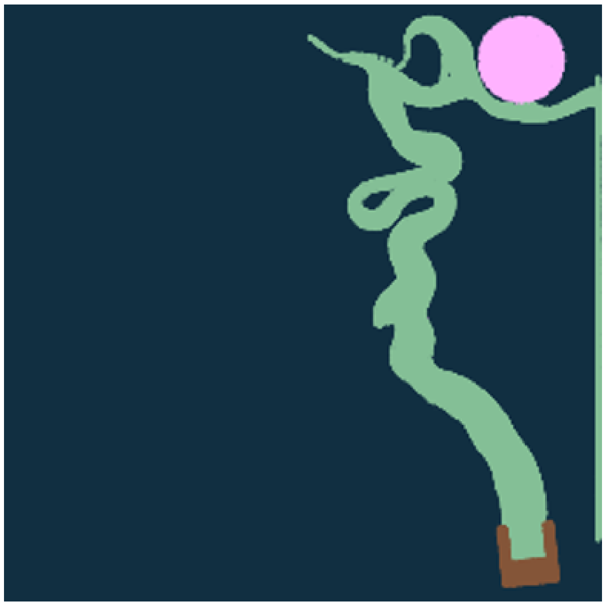

Figure 4: We want to control the tube to lift the ball at certain height.

4.2 Effects of auto-encoder

As the results can show, compared with the original results, the reconstructed results can capture the main outline of the field, only losing some small eddies and high-frequency points which leads to no much difference for the control to the ball.

4.3 Training process

To implement the aforementioned algorithm, we have sampled trajectories for 1000 times with newly updated value functions and learned policy. For every trajectory, the sampled tuples (s_t, a_t, s_{t+1}) cover 1000+ transitions and are put into the experience replay buffer, whereas the initial pool for the replay buffer is filled with 1500 transitions. After each sampling process, the value function network is updated, which is followed by 1000 epochs training for double Q-network and policy network. During these training processes, each epoch tends to minimize corresponding losses in Algorithm. 1 over a sampled batch with size of 100 transitions.

In total, the trajectory sampling and training time have added up to about 16 hours,

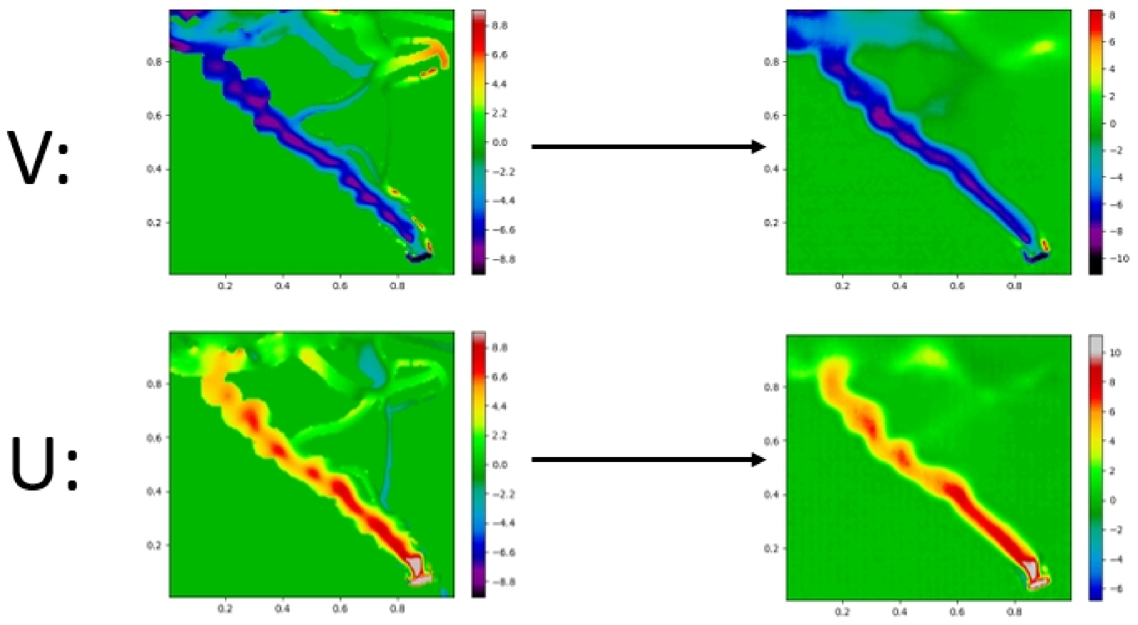

Figure 5: The results of auto-encoder

which is computed on a single Nvidia RTX 4090 GPU (24G). During these processes, CUDA is efficiently adopted for both simulations (which is most burdened and dependent on particle numbers) and training. With regard of the software environment, we mainly utilize pyTorch [15] for the Deep Learning, as well as jittor [16] and gym [17] packages for the Reinforcement Learning framework.

From the result, the accumulated award has on average risen from $6.5k$ with a normal Gaussian distribution policy to $10.5k$ with our trained DL-model-driven policy.

4.4 Demonstration

4.4.1 Human Control

Immediately after we finished the coding of MPM ejecting-tube solver, we have tested the feelings and difficulties of control if it is done by human beings. We have set 'A' and '$\leftarrow$' for accelerating the tube leftwards, 'D' and '$\rightarrow$' for accelerating the tube rightwards, left-click of the mouse for rotating the tube clockwise and right-click of the mouse for rotating the tube counterclockwise. Starting the game, due to imposed turbulence exit for the fluid, the ball can hardly be stabilized by finding a suitable position and orientation without any continuous adjustment. Besides, from human behaviors, one usually needs considerable time to respond and predicate possible falling-downs of the ball. In consequence, there are lots of downs and then ups of the ball behavior because human response lags the immediacy to prevent probable failures. This can be found in Fig. 6, the height of the ball oscillates up and down, where human can hardly control it well. Another noticeable feature is that human control tends to be less flexible and with fewer mobility, and it also matches the realistic facts. In real life, even acrobats can not achieve the same efficiency as an optimized machine. This again verifies why we do not use behavior-cloning strategy, for a desired manifestation even wins humans.

4.4.2 Random action

To compare the behavior and effect of our trained policy (Gaussian distribution with mean and variation given though trained MLP) with a random action (e.g. standard normal distribution), we have recorded the behavior of balls by the random wavering tube. The results are shown in Fig. 7, where it is clearly found that the tube would neglect ball's position and velocity, but just waves itself carelessly. In turn, the ball can fall on the ground much more frequently than a desired control (either bu human or by our trained model).

4.4.3 Trained result

For the trained policy, giving the deliberate reward optimization, it can be found that the adopted policy follows a Gaussian distribution with much smaller variation and meaningful mean. These both help the tube to achieve desired purposes, pushing the ball at certain height and continuously stay around. From Fig. 8, it clearly has achieved the target and perform quite well at pushing the ball at certain height. Especially, different with controlling by human, the trained policy normally can achieve the control in a smoother way. Namely, it takes less time and as if has expected or calculated the path the ball would go (e.g. is it going to fall down and how to prevent). It is due to the taring nature for the policy, and as a consequence it also behaves better than human achievements. In the trak of the ball, it is found for less time the ball drops from the targeted height.

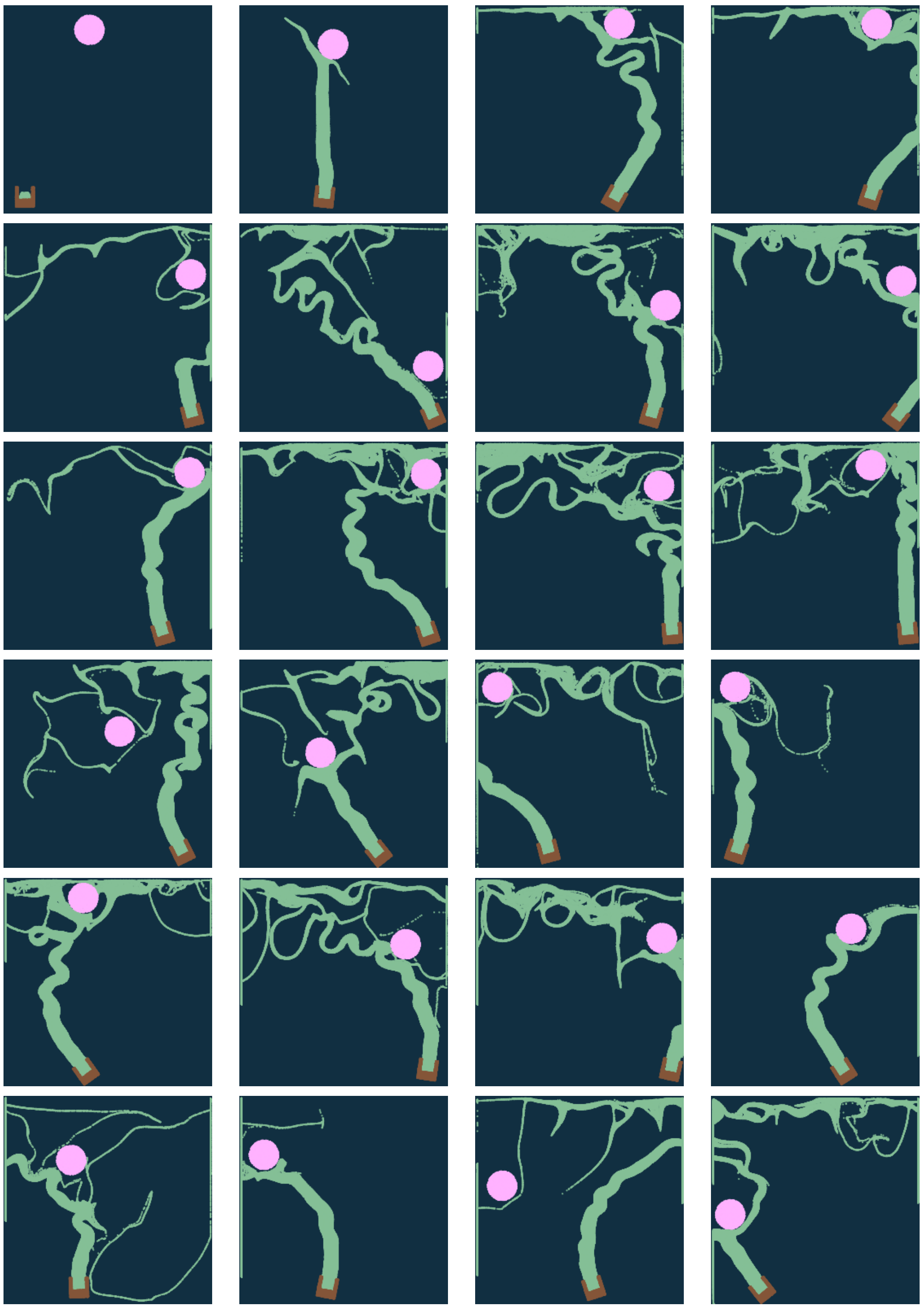

Figure 6: Snapshots of human control.

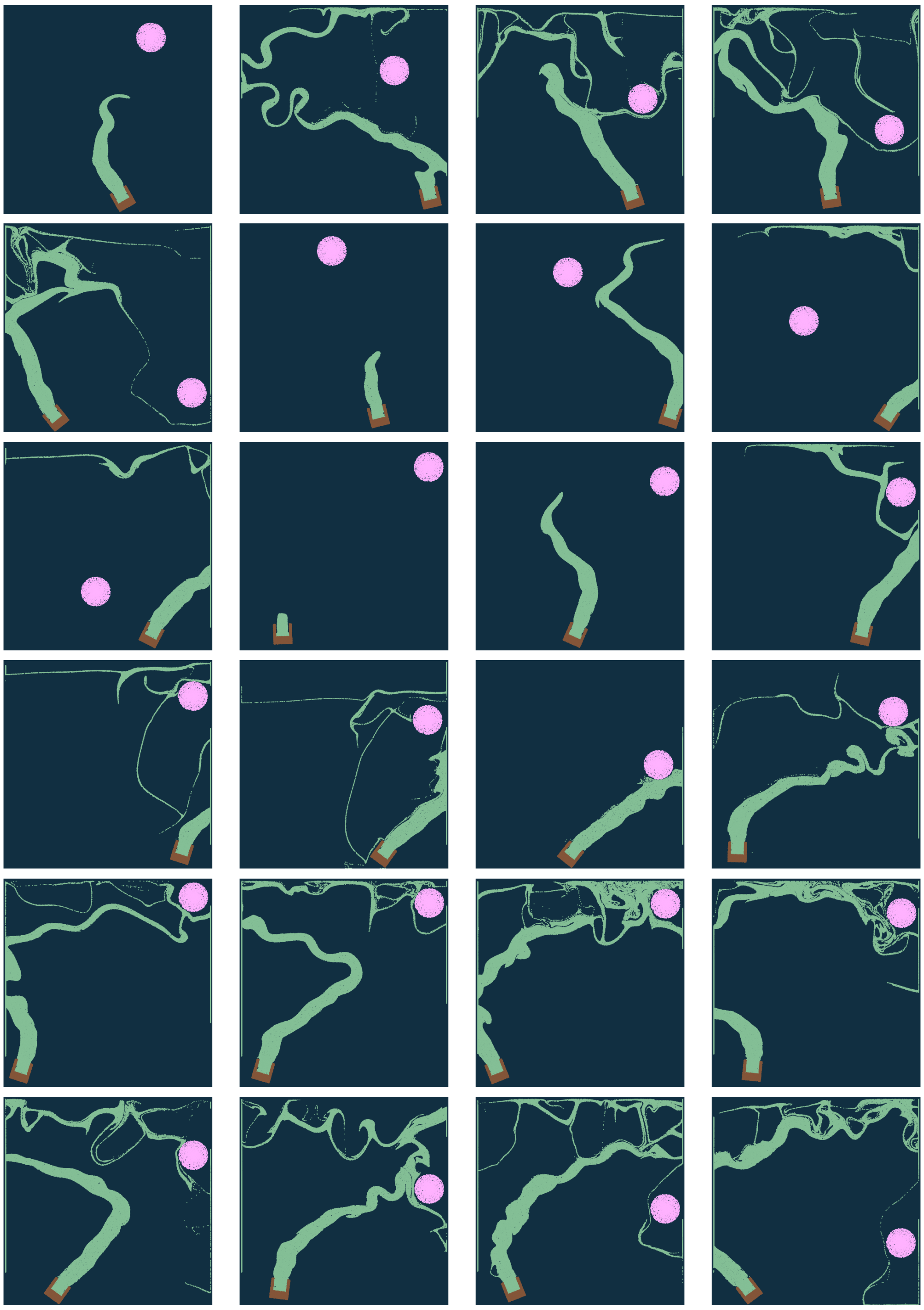

Figure 7: Snapshots of random tube action.

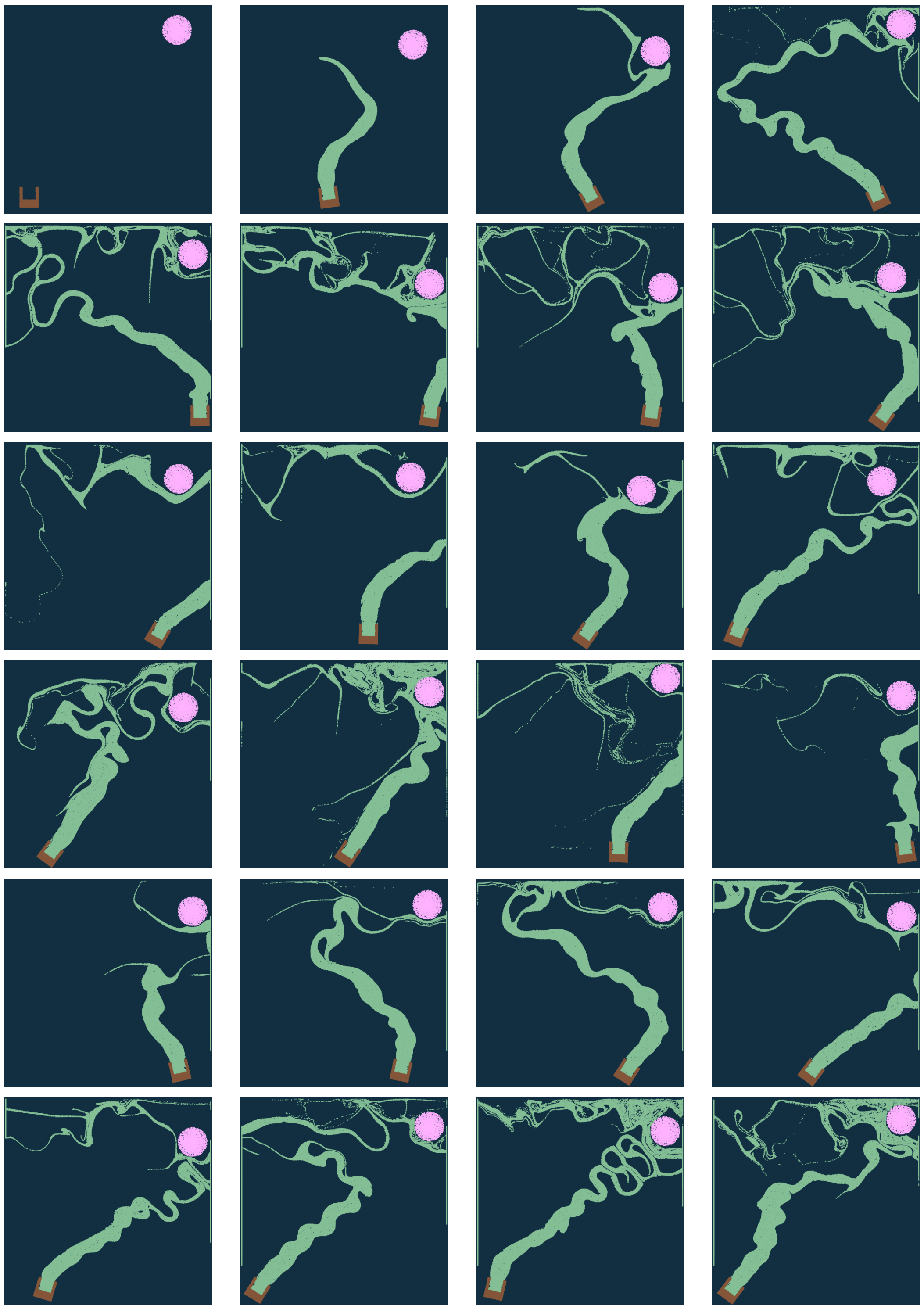

Figure 8: Snapshots of trained control.

5 Conclusion, Extension and Future Work

In current work, what we have achieved is referring from a SAC and Meta-RL learning practice on a *Versatile Control of Fluid-directed Solid Object*, and realize our design of a coupled rigid-solid system control optimization. Specifically, to target the ball by an continuously-injecting turbulent water jet through Reinforcement Learning. We have implemented the 2D-simulation and rendering through Taichi, treated as the environment for our control problem. During this project, we have also optimized potential problems in Taichi as out-of-index in Vram usage, and more efficiently used particles out of ball-interaction domain to save simulation burdens. For high under-actuation and highly correlated velocity field, we also employed auto-encoder to downsize the dimension of state variables so as to reduce MDP sizes and computational consumption. For reinforcement learning part, we have explored the idea of entropy bonus and test it on the soft AC algorithm, as well as learned the meta-RL concept by utilization of a latent context space to achieve transfer learning dealing with more generalized tasks. In deep learning framework, we have seen its capacity of describing a complex problem via two basic kinds of structure, CNN and MLP.

For a close extension, our codes can be easily extended to a 3D scenario. In the aspect of simulation, both MLS-MPM method and rendering have the general form of temporal and spatial movements. The current work only needs additional one dimension to achieve 3D visual effect. For the learning part, the job related with auto-encoder can also be easily generalized to 3D CNN, maybe with increased dimension of the encoded parameters. To this sense, the MDP only increase its state size while increasing dimension of the ball movement. These mainly cause the difference of training time, yielding a higher demand, which is hardly achievable at this momentum. Besides, it should also be interesting to see how result can be achieved (e.g. the change of shape of the learned policy, achieved rewards, visual effect) and upgraded by spending more sampling numbers and training states. And to demonstrate the 3D animation, it is also a good idea to adopt volume rendering for realistic texture. Above are all potential extensions can be safely done based on our current work, leaving an intriguing outcome to discover.

References

[1] Bo Ren et al. "Versatile Control of Fluid-directed Solid Objects Using Multi-task Reinforcement Learning". In: (Oct. 2022). DOI: 10.1145/3554731.

[2] Zherong Pan and Dinesh Manocha. "Efficient Solver for Spacetime Control of Smoke". In: 36.5 (July 2017). ISSN: 0730-0301. DOI: 10.1145/3016963. URL: https://doi.org/10.1145/3016963.

[3] Antoine McNamara et al. "Fluid Control Using the Adjoint Method". In: *ACM SIGGRAPH 2004 Papers*. SIGGRAPH '04. Los Angeles, California: Association for Computing Machinery, 2004, pp. 449–456. ISBN: 9781450378239. DOI: 10.1145/1186562.1015744. URL: https://doi.org/10.1145/1186562.1015744.

[4] Jingwei Tang et al. "Honey, I Shrunk the Domain: Frequency-aware Force Field Reduction for Efficient Fluids Optimization". In: *Computer Graphics Forum* 40.2 (2021), pp. 339–353. DOI: https://doi.org/10.1111/cgf.142637.

[5] Adrien Treuille et al. "Keyframe Control of Smoke Simulations". In: *ACM Trans. Graph.* 22.3 (July 2003), pp. 716–723. ISSN: 0730-0301. DOI: 10.1145/882262.882337. URL: https://doi.org/10.1145/882262.882337.

[6] Yuanming Hu et al. "ChainQueen: A Real-Time Differentiable Physical Simulator for Soft Robotics". In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 6265–6271. DOI: 10.1109/ICRA.2019.8794333.

[7] Pingchuan Ma et al. "Fluid Directed Rigid Body Control Using Deep Reinforcement Learning". In: *ACM Trans. Graph.* 37.4 (July 2018). ISSN: 0730-0301. DOI: 10.1145/3197517.3201334. URL: https://doi.org/10.1145/3197517.3201334.

[8] Connor Schenck and Dieter Fox. "SPNets: Differentiable Fluid Dynamics for Deep Neural Networks". In: *CoRR* abs/1806.06094 (2018). arXiv: 1806.06094. URL: http://arxiv.org/abs/1806.06094.

[9] Yuanming Hu et al. "Difftaichi: Differentiable programming for physical simulation". In: *arXiv preprint arXiv:1910.00935* (2019).

[10] Kevin Bergamin et al. "DReCon: Data-Driven Responsive Control of Physics-Based Characters". In: *ACM Trans. Graph.* 38.6 (Nov. 2019). ISSN: 0730-0301. DOI: `10.1145/3355089.3356536`. URL: `https://doi.org/10.1145/3355089.3356536`.

[11] Sehee Min et al. "SoftCon: Simulation and Control of Soft-Bodied Animals with Biomimetic Actuators". In: *ACM Trans. Graph.* 38.6 (Nov. 2019). ISSN: 0730-0301. DOI: `10.1145/3355089.3356497`. URL: `https://doi.org/10.1145/3355089.3356497`.

[12] Wenhao Yu, Greg Turk, and C. Karen Liu. "Learning Symmetric and Low-Energy Locomotion". In: *ACM Trans. Graph.* 37.4 (July 2018). ISSN: 0730-0301. DOI: `10.1145/3197517.3201397`. URL: `https://doi.org/10.1145/3197517.3201397`.

[13] Soohwan Park et al. "Learning Predict-and-Simulate Policies from Unorganized Human Motion Data". In: *ACM Trans. Graph.* 38.6 (Nov. 2019). ISSN: 0730-0301. DOI: `10.1145/3355089.3356501`. URL: `https://doi.org/10.1145/3355089.3356501`.

[14] Tuomas Haarnoja et al. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *CoRR* abs/1801.01290 (2018). arXiv: `1801.01290`. URL: `http://arxiv.org/abs/1801.01290`.

[15] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *arXiv preprint arXiv:1912.01703* (2019).

[16] Yunming Chen et al. "Jittor: a high-performance deep learning library based on JIT compiling and meta-operators". In: *arXiv preprint arXiv:2008.02981* (2020).

[17] Greg Brockman et al. *OpenAI Gym.* `https://gym.openai.com/`. 2016.